Universidade Federal do Ceará
Departamento de Computação
Mestrado e Doutorado em Ciência da Computação

Avaliação de Desempenho de uma Plataforma de Componentes Paralelos

Cenez Araújo de Rezende

Fortaleza – Ceará
Setembro 2011

Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação
Mestrado e Doutorado em Ciência da Computação

Avaliação de Desempenho de uma Plataforma de Componentes Paralelos

Autor
Cenez Araújo de Rezende

Orientador
Prof. Dr. Francisco Heron de Carvalho Junior

Dissertação de mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal do Ceará como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Fortaleza – Ceará
Setembro 2011

Resumo

Reduzir a complexidade do software e permitir o desenvolvimento em larga escala de aplicações voltadas à Computação de Alto Desempenho (CAD) tem exigido o desenvolvimento de ferramentas com potencial capacidade de abstração na construção de sistemas. As tecnologias que envolvem o desenvolvimento de componentes procuram alcançar esses requisitos, buscando oferecer suporte a reuso, interoperabilidade, produtividade e maior flexibilidade de manutenção e desenvolvimento de aplicações de alto desempenho. No entanto, conciliar alto poder de abstração com alto poder de expressividade na construção de componentes de aplicações não é algo trivial, o que as atuais tecnologias não têm conseguido solucionar, uma vez que adotam as tradicionais formas de paralelismo por processos. Diante disso, a plataforma HPE (*Hash Programming Environment*), baseada no modelo de componentes Hash, tem buscado suportar formas mais gerais de paralelismo, conciliando expressividade com alto poder de abstração, uma vez que o modelo Hash é baseado em interesses de software e não em processo, como é feito tradicionalmente. Nesse contexto, esta dissertação busca explorar os recursos do HPE, certificando-se de sua viabilidade no contexto de aplicações de alto desempenho e validando suas técnicas de programação paralela baseadas em componentes. Isso tem resultado em um processo de construção de aplicações científicas sob a abordagem de componentes, tendo como base o conjunto de aplicativos NPB (*Nas Parallel Benchmarks*), o qual passa por um processo rigoroso de conversão para ser suportado pelo HPE. No processo de conversão e refatoração em componentes, busca-se conservar as estruturas originais do NPB, sem alterações significativas nos códigos que declaram e inicializam as estruturas de dados, bem como os que descrevem computações, topologia de processos e comunicação entre os processos. Para validação da plataforma, uma avaliação sistemática de desempenho é feita, tendo como princípio isolar e mensurar o peso ou o efeito da refatoração do NPB em componentes Hash.

Abstract

In order to deal with programming-in-the-large requirements in emerging applications of High Performance Computing (HPC), it is still necessary the development of new software development tools for reconciling high level of abstraction, expressiveness and high performance. The technologies behind CBHPC (Component-Based High Performance Computing) target these requirements, looking for reuse of software parts, interoperability across execution platforms, high development productivity and easy maintenance. However, to reconcile high level of abstraction, high performance and high expressiveness for parallel programming models and patterns when building HPC applications is not trivial. For this reason, most of the current technologies fail in this context, since they adopt the traditional process-oriented perspective in the architecture of parallel programs. The HPE platform (Hash Programming Environment) sits on top of the Hash component model to support general forms of parallelism, by combining high expressiveness with high level of abstraction. The Hash component model proposes a concern-oriented perspective to parallel programming, in alternative to the traditional process-oriented approach. In this context, this dissertation is about the efficacy and efficiency of HPE for HPC applications, also validating some of its parallel programming techniques based on components. For that, a set of programs from NPB (NAS Parallel Benchmarks), a widely disseminated collection of benchmarks for evaluating the performance of parallel computing platforms, written in Fortran, C and Java, have been refactored into components aimed at the HPE platform. In such refactoring, the original structure of the benchmarks has been preserved, with minimal changes in the code that declare and initialize data structures, as well as those that describe computations and communication patterns. Using the component-based versions of the benchmarks, a systematic performance evaluation has been performed for quantifying the overheads caused strictly by the component-based structure.

Agradecimentos

Nesta oportunidade de expressar os agradecimentos, reflito sobre as dificuldades no decorrer da jornada de estudo. Devo avaliar que esta não foi uma caminhada fácil. Entretanto, foi gratificante. Diante de grandes desafios e dificuldades, as motivações se sobressaíram, pois foram maiores, convertendo-se em força. O conhecimento é resultado desses pormenores, implicitamente contidos no currículo de quem almeja e realiza um objetivo nas proporções de um mestrado. Neste texto, corro o risco de não conseguir atingir um "muito obrigado *ideal*", pois faltam palavras para expressar a devida gratidão. Agradeço pelas inúmeras vezes que contei com a solidariedade de um amigo, velho ou novo. Agradeço especialmente ao meu orientador Heron, pela orientação competente no desenvolvimento deste trabalho. Quero expressar minha admiração e agradecimento a todos os funcionários e professores do Departamento de Computação, com os quais pude contar irrestritamente. Agradeço aos meus pais, Antonio P. Rezende e Teresa M. Araújo Rezende, pelo incentivo e apoio. Ao meu irmão, Marcos Araújo de Rezende, por igual motivo. À UFC e ao Departamento de Computação pela oportunidade. À Funcap pelo apoio financeiro através de bolsa de estudo. Finalmente, agradeço a Deus pela oportunidade e momento especial de vida.

Sumário

Capítulo 1

Introdução

O assunto principal abordado neste trabalho é a avaliação de desempenho de uma plataforma de componentes paralelos voltada às necessidade de aplicações de Computação de Alto Desempenho (CAD), em especial aquelas de interesse das ciências e das engenharias. A plataforma de componentes paralelos tem sido desenvolvida pelo grupo de pesquisa coordenado pelo Professor orientador deste trabalho nos últimos anos. Chama-se HPE (*Hash Programming Environment*) e serve-se ao gerenciamento do ciclo de vida de componentes que seguem o modelo de componentes HASH, bem como de aplicações constituídas por meio desses componentes, sobre plataformas de *cluster computing*. Este trabalho tem especial interesse em suprir a necessidade por uma avaliação de desempenho com metodologia e fundamentação rigorosa para a plataforma HPE.

Nas seções subsequentes, são apresentadas as motivações principais, os objetivos (geral e específicos) e a estrutura da Dissertação.

1.1 Motivações

A partir do final da década de 80, a evolução da capacidade do *hardware* foi marcante, alavancando mudanças proporcionalmente significativas com relação a escala e a complexidade do *software* [63]. Isso foi possível com o aumento da capacidade de lidar com essas aplicações, especialmente com o apoio de soluções emergentes em banco de dados, linguagens de programação e metodologias de desenvolvimento, como a orientação a objetos, componentes e padrões de projeto. Mais recentemente, as aplicações de computação de alto desempenho com origem nos domínios das ciências computacionais e engenharias tem despertado o interesse da indústria de software, devido ao reconhecimento de seu impacto no desenvolvimento

científico e tecnológico [27]. Geralmente, são os centros de pesquisa de instituições acadêmicas e as indústrias de base tecnológica e fortemente dependentes de inovação os mais expressivos usuários dessas aplicações, as quais seguem áreas diversas como previsão climática, dinâmica dos fluidos, problemas de otimização combinatória, bioinformática, dentre outras [27, 61]. Entretanto, o desenvolvimento dessas aplicações é uma tarefa árdua com as ferramentas tradicionais de que se dispõe para o desenvolvimento de *software*, por exigir o controle sobre detalhes mais específicos da plataforma de execução a fim de explorar o seu potencial de desempenho.

Para atender à demanda de aplicações CAD, uma grande contribuição tem sido dada pela consolidação e proliferação de plataformas de computação paralela de ótima relação custo/benefício, como os *clusters* e as grades computacionais, as quais já há quase duas décadas constituem alternativa real frente às plataformas específicas de CAD convencionalmente denominadas de *supercomputadores*. Essas plataformas têm possibilitado e incentivado o acesso de novos usuários às tecnologias de CAD, os quais motivam o surgimento de novas aplicações para essas arquiteturas.

Outra contribuição importante tem origem no estudo de algoritmos capazes de explorar ao máximo o desempenho das arquiteturas de computação paralela. O projeto de um algoritmo paralelo envolve vários aspectos peculiares, não presentes no projeto de algoritmos sequenciais, como técnicas de decomposição em tarefas, exploração das características de interações, técnicas de mapeamento para balanceamento de carga e métodos que reduzem sobrecarga de interações entre processos [45].

Embora existam muitas inovações relativas às plataformas de computação paralelas e algoritmos paralelos capazes de explorar o seu desempenho, os recursos de programação não acompanharam a mesma evolução. As interfaces de programação que tem conseguido explorar o potencial de desempenho das plataformas de computação paralela oferecem baixo nível de abstração e modularidade, a despeito da alta portabilidade, como é o caso das bibliotecas que implementam a interface de passagem de mensagens MPI (*Message Passing Interface*). Algumas soluções de alto nível são adotadas de forma disseminada, como as bibliotecas científicas de propósito especial. Entretanto, como o próprio termo sugere, não são capazes de conciliar expressividade, abstração, alto desempenho e portabilidade. Uma abordagem que tem sido amplamente discutida para esse problema são os chamados esqueletos de algoritmos. Segundo Murray Cole [36], "trata-se de uma percepção de modelos genéricos em rotinas paralelas. Formam padrões que podem ser usados como um

"*kit de ferramentas*" para o desenvolvedor, possibilitando a resolução de problemas tradicionais de engenharia de software paralelo como portabilidade, simplicidade e desempenho". Entretanto, esqueletos de algoritmos não tem obtido a esperada disseminação em aplicações industriais, a despeito do forte interesse acadêmico desde a década de 1990 que até hoje perdura.

Uma segunda abordagem, a qual é de interesse dessa Dissertação, refere-se aos componentes de software. Componentes podem ser vistos como unidades de programa que podem ser implantadas independentemente em uma plataforma de computação, seguindo um conjunto de regras de comportamento e implementando padrões de interfaces com a finalidade de permitir conexão com outros componentes [66]. Analogamente a esse conceito, os dispositivos de *hardware* permitem a conexão de peças de diversos fornecedores. Em *software*, vantagens podem ser vistas por três fatores. O primeiro é a padronização através de uma modelagem melhor elaborada (modelo do componente, modelo de conexão e modelo de implantação). O segundo diz respeito ao reuso, que tem consequência direta na produtividade no processo de desenvolvimento de softwares. O terceiro é estabelecer um grau maior de confiabilidade nas aplicações, pois suas partes são bem definidas, através dos componentes, validadas e testadas.

Para atender as necessidades de CAD, e tendo em vista a precariedade dos modelos de componentes existentes para atender aos requisitos de aplicações de CAD, surgiram novos modelos e suas plataformas de componentes, tais como CCA [12], Fractal [23] e GCM [16]. Esses modelos apresentam várias extensões ortogonais que visam o suporte ao paralelismo, porém não tão expressivos para expressar padrões de computação paralela quanto bibliotecas de passagem de mensagens puras, tais como o MPI.

O suporte ao paralelismo nessas plataformas de componentes é ainda baseado na utilização de processos como unidades básicas de decomposição do *software*, herdado da forma tradicional de programação paralela em pequena escala usando bibliotecas de passagem de mensagens. Diferente dessa visão, e tendo origem na linguagem *Haskell#* [29], os componentes do modelo HASH seguem o princípio da divisão do processo em fatias de mesmo interesse, permitindo que fatias de diferentes processos possam se agrupar em interesses comuns. Dessa forma, o modelo propõe uma visão sob a perspectiva de orientação a interesses de software (*concerns*), e não a processos como é feito tradicionalmente. Forma-se o componente através do envolvimento de unidades de um conjunto de processos, onde a unidade representa a contribuição do

processo naquele interesse. Como consequência, tem-se uma definição mais geral de componente paralelo, onde o paralelismo encontra-se intrínseco às definições dos componentes e não através de extensões ortogonais aos modelos, adotados tradicionalmente, os quais não têm sido capazes de expressar os diversos padrões de interação entre processos em programas paralelos.

1.1.1 HPE

O HPE foi desenvolvido com o objetivo de validar o uso de componentes paralelos do modelo HASH, chamados de componentes #, em plataformas de *cluster computing*. Sua arquitetura é composta por três módulos distintos, denominados *Front-End*, *Back-End* e *Core*. Através do *Front-End*, os usuários podem configurar e controlar o ciclo de vida de componentes #. O *Core* oferece serviços de descoberta e catalogação de componentes #, reusáveis através de configurações construídas por meio do *Front-End*. O *Back-End* é responsável por gerenciar e monitorar a execução de componentes em uma plataforma de execução paralela. Os serviços do *Core* e do *Back-End* são implementados por meio de *Web Services*, tendo em vista a sua independência em relação ao *Front-End*.

No desenvolvimento do *Front-End* é usado o *plug-in Graphical Editing Framework* (GEF) [1] da plataforma *Eclipse*. No desenvolvimento do *Core*, é usada a linguagem de programação *Java*, com o propósito de coordenar as *locations* [2]. No desenvolvimento do *Back-End* é usado o Mono, uma plataforma de execução virtual baseada em componentes e de código aberto compatível com o padrão CLI[3] (*Common Language Infrastructure*) [6].

Portanto, a construção do HPE envolve o uso de tecnologias cujo desempenho e eficácia tem sido constantemente avaliadas por aplicações reais de escala industrial, como é o caso do Mono, Eclipse e Java. Apesar disso, ainda são precisos estudos que avaliem o desempenho e a eficácia do uso combinado dessas tecnologias na construção do HPE, mostrando a sua viabilidade no desenvolvimento de aplicações em escala industrial. Especialmente, é necessário avaliar aspectos de desempenho do Mono. Este trabalho fecha portanto uma lacuna, apresentando a primeira avaliação de desempenho sistemática da plataforma de componentes paralelos HPE.

[1] *Graphical Editing Framework*, oferece recursos visuais para configuração de componentes.
[2] Locais onde os componentes são fisicamente arquivados.
[3] Padrão para infra-estrutura de componentes.
[4] *Global Address Cache*.
[5] *Distributed Global Address Cache*.

1.1.2 Importância da avaliação na área de CAD

Com as inovações constantes na área de computação, várias instituições firmaram o interesse em melhorar a qualidade de sistemas através da avaliação de seu desempenho. Dentre essas instituições, podemos citar as que promovem anualmente conferências sobre análise de desempenho: *ACM Sigmetrics* (*Association for Computing Machinery*), *IEEE* (*Institute of Electrical and Electronic Engineers*), *CMG* (*Computer Measurement Group*), *IFIP* (*International Federation for Information Processing*), *SIAM* (*Society for Industrial and Applied Mathematics*), *ORSA* (*Operations Research Society of America*).

O interesse pela avaliação de desempenho é algo que possibilita o aumento de qualidade em arquiteturas, algoritmos e artefatos de programação. Em cada análise, é preciso um íntimo conhecimento do sistema e uma cuidadosa escolha de metodologias, cargas de trabalho e ferramentas de auxílio à análise [47]. A aplicação de técnicas de avaliação pode nos dar respostas a questionamentos como requisito de desempenho, diferentes decisões de projetos de *software*, comparação de dois ou mais sistemas, determinar valor ótimo de parâmetros, encontrar gargalos, caracterizar carga de sistemas e prever desempenho em futuras sobrecargas. As técnicas principais para avaliação são três: modelagem analítica, simulação e medição direta. A modelagem analítica, como o próprio nome sugere, envolve modelos matemáticos, os quais agregam abstrações essenciais do comportamento de um sistema. No caso de simulação, geralmente faz-se o uso de alguma linguagem de programação de alto nível para simular o comportamento do sistema. Em medição, tem-se a instrumentação e monitoramento de um sistema real.

O HPE representa uma proposta viável para aplicações de CAD de larga escala. Entretanto, isso também faz com que ele precise de constantes pesquisas com o objetivo de aumentar e validar a qualidade pertinente a grandes projetos. Em particular, a sua avaliação deve basear-se em critérios sistemáticos que obedeçam aos princípios e aos rigores necessários na área de CAD. Fazendo isso, avalia-se paralelamente a plataforma Mono, buscando gargalos que podem eventualmente ser tratados em uma pesquisa para implementação de um compilador específico para o HPE. Neste, podem ser incorporadas suposições gerais às aplicações com carga intensiva, como o uso constante de vetores multidimensionais, que podem representar um impacto significante no desempenho, devido à intensa verificação de limites de vetores. Tal preocupação é citada por diversos pesquisadores [19, 56, 70]. Por isso, é discutida nesta dissertação.

A importância da avaliação de desempenho em plataformas tem contribuído para estudos e desenvolvimentos de *benchmarks*, como o *NAS Parallel Benchmarks (NPB)*, desenvolvido pela divisão NAS (*NASA Advanced Supercomputing*) da NASA, agência espacial dos Estados Unidos da América. Ele é projetado para efetuar carga intensiva de computação numérica e movimentação de dados em plataformas de computação paralela, típicas de aplicações de Dinâmica dos Fluidos Computacional (CFD). Sua estrutura está composta por cinco núcleos (EP, MG, CG, FT e IS), destinados a explorar características gerais de máquinas paralelas, e três módulos de simulação (LU, SP e BT) [15]. Os códigos do NPB são *abertos*, estando disponíveis em versões *Fortran*, *C* e *Java*, sob licença especial[1] do governo dos Estados Unidos. O NPB é usado neste trabalho como carga de trabalho, sendo detalhado na Seção 3.3.5.

1.1.3 Contribuições

Através do uso de *benchmarks* e técnicas para avaliação de desempenho, alguns pesquisadores têm buscado otimizar sistemas paralelos ou desenvolver artefatos para CAD, como pode ser visto em alguns trabalhos que citamos na Seção 6.2. Nesse sentido, esta dissertação busca contribuir com a avaliação sistemática da plataforma de componentes paralelos HPE, certificando-se de sua viabilidade no contexto de aplicações de Alto Desempenho e validando suas técnicas de programação paralela baseadas em componentes #. Para isso, usa-se a técnica de avaliação denominada "medição", explorando o conjunto de aplicativos NPB. Esses aplicativos necessitam passar por um processo rigoroso de conversão, uma vez que se encontram disponíveis nativamente na linguagem de programação Fortran e o HPE atualmente tem suportado o desenvolvimento de componentes sob a linguagem C#. Assim, são construídas versões paralelas do NPB em C# e posteriormente são desenvolvidos diversos componentes #, refatorados a partir das técnicas de paralelismo inerente ao NPB, resultando em um repositório de componentes reusáveis para explorar o paralelismo de aplicações científicas.

1.2 Objetivos

Nesta seção, apresentamos de forma explícita os objetivos geral e específicos da dissertação.

[1]Licença em: http://www.opensource.org/licenses/nasa1.3.php

1.2.1 Objetivo Geral

Avaliar o desempenho da plataforma de componentes paralelos HPE, com o propósito de certificar a viabilidade de sua utilização no contexto de computação de alto desempenho e obter subsídios para otimização do seu desempenho.

1.2.2 Objetivos Específicos

i. Apresentar versões do NPB na linguagem C#, para execução sobre plataformas CLI, como Mono e .NET , permitindo a avaliação de desempenho dessa máquina virtual para aplicações de CAD;

ii. Exercitar e demonstrar técnicas de refatoração em componentes de programas científicos para a plataforma HPE;

iii. Discutir a necessidade de rigor metodológico na avaliação de desempenho na área de CAD.

1.3 Estrutura desta Dissertação

A dissertação encontra-se dividida em seis capítulos. O Capítulo 1 introduz as motivações, os trabalhos relacionados e as contribuições deste trabalho. O Capítulo 2 apresenta detalhes sobre as plataformas de componentes paralelos voltados a aplicações de CAD, dando ênfase ao modelo HASH e a plataforma HPE. O Capítulo 3 apresenta conceitos sobre avaliação de desempenho em sistemas computacionais, com ênfase no ferramental que será usado para avaliação de desempenho do HPE. O Capítulo 4 aborda a metodologia de conversão do NPB e o esboço de seu processo de fatoração. O Capítulo 5 apresenta a avaliação de desempenho HPE. O Capítulo 6 apresenta as conclusões deste trabalho, indicando as dificuldades encontradas na sua realização e possíveis trabalhos futuros motivados pelos seus resultados.

Capítulo 2

Plataformas de Componentes Paralelos

A exploração do potencial de desempenho do hardware oferecido pela indústria de computadores, especialmente no que se refere às arquiteturas paralelas, é a motivação das pesquisas na área de Computação de Alto Desempenho (CAD), notadamente tendo em vista aplicações de computação intensiva nos domínios das ciências computacionais. As soluções recentes propostas pela área de CAD tem permitido uma maior disseminação de recursos de processamento paralelo para esses domínios de aplicação, como os clusters e grades computacionais, melhorando a relação custo/benefício no uso dessas tecnologias e despertando o interesse de aplicações corporativas, áreas de finanças e de negócios em geral.

Simultaneamente ao aparecimento de soluções que disseminaram as tecnologias de CAD para uma grande escala potencial de usuários, e fortemente influenciada por tais, a complexidade do software nos domínios das ciências e das engenharias tem aumentado, assim como aplicações corporativas com requisitos de CAD têm surgido, exigindo a aplicação de melhores práticas de desenvolvimento a fim de obter uma maior produtividade no desenvolvimento e na manutenção desses softwares. Entretanto, esse é um contexto relativamente novo. A programação em pequena escala (*programming-in-the-small*) ainda é predominante em aplicações científicas e de engenharias, o que tem mudado desde o final dos anos 1990. Note que esse contexto é o oposto ao ocorrido com aplicações nos domínios corporativos e de negócios, onde as técnicas de programação de larga escala (*programming-in-the-large*) proliferaram rapidamente desde os anos de 1970 para fazer frente à crescente complexidade do software nos domínios dos negócios, situação

que levou ao contexto que ficou conhecido como "Crise do Software" no final da década de 1960 [39]. Esse contexto motivou um grande avanço em relação aos artefatos de desenvolvimento, tais como linguagens, modelos e técnicas para programação de larga escala.

Por várias décadas, o software de domínio das ciências e das engenharias esteve satisfeito com as técnicas de programação em pequena escala, o que justifica a grande popularidade das linguagens Fortran e C no desenvolvimento dessas aplicações. Porém, com o advento de tecnologias de grades computacionais e o crescimento da própria internet, a qual abriu caminho sem precedentes à integração e interação multidisciplinar das ciências modernas, essas aplicações começaram a exigir novos mecanismos que permitissem uma escala maior na programação. Contudo, o legado de tecnologias para desenvolvimento em larga escala propostas e validadas pela prática para aplicações corporativas e de negócios, com início de desenvolvimento a partir da Crise do Software, não teve como foco principal o desempenho, considerado requisito pouco relevante, salvo raras exceções. Assumia-se a existência de uma uniformidade nas plataformas de execução, abstraindo-se dos detalhes intrínsecos das arquiteturas. Portanto, surge a necessidade de novas técnicas de desenvolvimento para satisfazer os requisitos de programação em larga escala para aplicações nos domínios de interesse de CAD. Essa foi a conclusão que alguns grupos de pesquisa de laboratórios nacionais dos EUA chegaram. Motivados pelo sucesso dos componentes em aplicações corporativas, alguns pesquisadores viram no princípio de fatoração do software em componentes paralelos uma alternativa viável, o que levou ao surgimento do padrão CCA (*Common Component Architecture*) [12]. Com motivação semelhante, porém em contextos distintos, outras alternativas foram surgindo como Fractal/ProActive [24] e GCM (*Grid Component Model*) [16].

No entanto, há ainda muitas dificuldades a superar com esses atuais modelos, especialmente a necessidade por noções gerais de componentes paralelos e melhores conectores entre esses para promover sincronização paralela eficiente, o que tem motivado novas pesquisas para estendê-los [34]. Uma das justificativas para essas dificuldades é a tendência herdada dos paradigmas tradicionais de programação paralela de tratar processos como unidades de decomposição primárias do software, ao invés de concentrar-se nos seus intereses como convém às técnicas modernas de engenharia de software. Isso leva a problemas de modularidade quando usamos técnicas modernas de desenvolvimento de larga escala em programação paralela. Uma solução que tem surgido nesse contexto é o modelo de componentes Hash,

objeto de estudo desta dissertação.

Componentes Hash têm como base os interesses de software, ao invés de processos. Processos são tratados como unidades de decomposição transversais aos interesses [31] e podem ser vistos de forma ortogonal. Isso proporciona maior abrangência e suporte ao paralelismo em modelos de componentes paralelos. Nas seções a seguir, discutiremos sobre as atuais plataformas de componentes paralelos, e a abordagem dada por elas ao paralelismo. Em seguida, é explicado o modelo de componentes Hash e a plataforma HPE.

2.1 CCA - Common Component Architecture

O padrão CCA foi desenvolvido por pesquisadores de laboratórios nacionais e universidades nos EUA, organizados no Fórum CCA[1], tendo como inspiração o padrão *CORBA* (*Common Object Request Broker Architecture*) [3]. Semelhante ao *CORBA*, o CCA utiliza portas *provides/uses* para composição de componentes em aplicações. A especificação diz que uma porta é um recurso que pode ser exportado ou importado por outros componentes, o que normalmente representa rotinas definidas por alguma linguagem através de uma interface. A implementação das rotinas definidas pela interface permite a conexão entre componentes através de serviços CCA, como é visto na Figura 2.1.

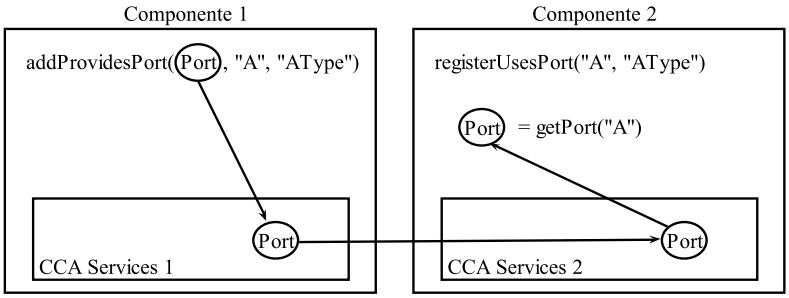


Figura 2.1: Diagrama provides/uses. Fonte [12]

O padrão CCA não especifica como o paralelismo deve ser abordado pelos *frameworks* computacionais que implementam o CCA. Isso se deve ao fato de que na época de sua proposta, os pesquisadores não sabiam exatamente como isso poderia ser feito de maneira a garantir alta eficiência e generalidade no suporte a padrões de paralelismo. Assim, os projetistas e desenvolvedores de *frameworks* estariam encarregados de avaliar diversas alternativas. A partir dessa experiência, esperavam

[1] http://www.cca-forum.org

especificar o suporte ao paralelismo em futuras versões do CCA, o que ainda não foi realizado após uma década.

A forma de maior sucesso de suporte ao paralelismo em *frameworks* CCA é o SCMD (*Single Component Multiple Data*), implementado pelo *framework* CCAffeine [11], análogo ao paradigma SPMD (*Single Program Multiple Data*) da biblioteca de passagem de mensagens MPI (*Message Passing Interface*). O SPMD é a forma comum de suportar computação paralela, onde um programa idêntico é executado nos nós de processamento da plataforma de execução paralela e os dados são decompostos e distribuídos entre os processos resultantes [12]. A mesma lógica é utilizada no paradigma SCMD, porém sob a perspectiva de componentes, onde cada componente é configurado e instanciado de forma idêntica em cada nó de processamento de um cluster gerenciado pelo *framework* CCAffeine. O conjunto de instâncias idênticas de componentes em processos distintos é conhecido por regimento[2]. Cada componente integrante de um regimento é ligado pelas portas *provides/uses* a um integrante de um outro regimento através da memória compartilhada, como pode ser visto na Figura 2.2. Os componentes de um regimento podem interagir através de troca de mensagens utilizando alguma biblioteca de troca de mensagens tradicional, além da comunicação local através de suas portas. No caso do CCAffeine, MPI é empregado para troca de mensagens. Entretanto, o paradigma SCMD não contempla noções gerais ou complexas de componentes paralelos, tal como em uma situação que não se enquadre na idéia de usarem-se "componentes idênticos" para representar os processos. Não há essa flexibilidade prevista com a biblioteca MPI, a qual não dificulta a construção de programas MPMD (*Multiple Program Multiple Data*), onde existe paralelismo sob diferentes programas que interagem com dados possivelmente diferentes. A alternativa adotada pelo CCA para permitir situações mais complexas está na simulação de MCMD (*Multiple Component Multiple Data*) através de múltiplos SCMD. O MCMD é análogo ao MPMD. Entretanto, a interação de regimentos com números diferentes de membros leva ao problema do acoplamento $M \times N$, que surge quando M componentes de um regimento precisam conectar suas portas a N componentes de um outro regimento que estão em um conjunto diferente de nós de processamento [7, 20].

Os *frameworks* DCA [21] e SciRun2 [50] tem tentado lidar com o problema $M \times N$ em seus projetos. Além disso, bibliotecas PRMI (*Parallel Remote Method Invocation*) tem sido propostas, especialmente pelos pesquisadores que desenvolvem

[2] do inglês *cohort*.

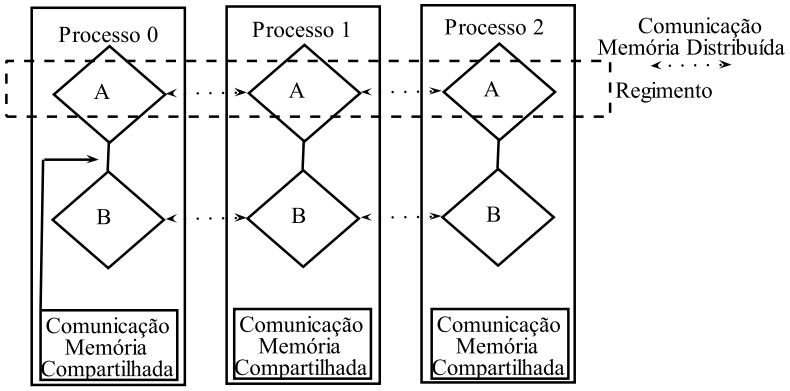


Figura 2.2: Diagrama SCMD

e mantém a ferramenta Babel [38]. O DCA é definido como um *framework* paralelo distribuído, diferenciando-se do CCAffeine (somente paralelo) e do XCAT3 [52] (somente distribuído), permitindo que vários regimentos de componentes coexistam em nós distintos de uma plataforma de computação paralela e troquem mensagens através de comunicadores MPI. Entretanto, ainda apresenta algumas restrições.

2.2 Fractal

Fractal é um modelo de componentes com composição hierárquica que pode ser aplicado em ambientes paralelos e/ou distribuídos [18]. Um componente Fractal é composto por duas partes essenciais: a *membrana* e o *conteúdo*. Seu caráter hierárquico permite a existência de componentes primitivos e compostos por outros componentes, ditos aninhados, que por sua vez podem ser primitivos ou compostos.

A *membrana* divide o componente em parte interna e externa, controladas por interfaces internas e externas. As interfaces no Fractal são semelhantes às portas de comunicação do CCA. Elas são do tipo *servidoras*, para receber chamadas, ou do tipo *cliente*, para solicitar chamadas. Uma conexão entre uma porta *cliente* e uma porta *servidora* de mesmo tipo efetiva a *ligação*. As interfaces internas são usadas para introspecção e reconfiguração do componente, através de *reflexão computacional*.

O *conteúdo* é a parte encapsulada do componente, a qual define a sua funcionalidade. No caso de um componente composto, pode ser constituído por um número finito de componentes envoltos pela *membrana*, ditos componentes aninhados. A *membrana* contém o *controlador* (*controller*) do comportamento do componente, no sentido de ser capaz de efetuar operações de parada, início ou *ligação* (*binding*) de interfaces pertinentes ao seu meio interno ou externo.

Dessa forma, o componente pode parar a comunicação externa enquanto termina atividades internas. Todas essas operações podem ser propagadas recursivamente para cada componente interno. O *controlador* da *membrana* também tem a função de interceptador, capaz de exportar a interface de um componente aninhado e expondo-a como uma interface externa do componente que o contém. A Figura 2.3 exemplifica as características principais do Fractal.

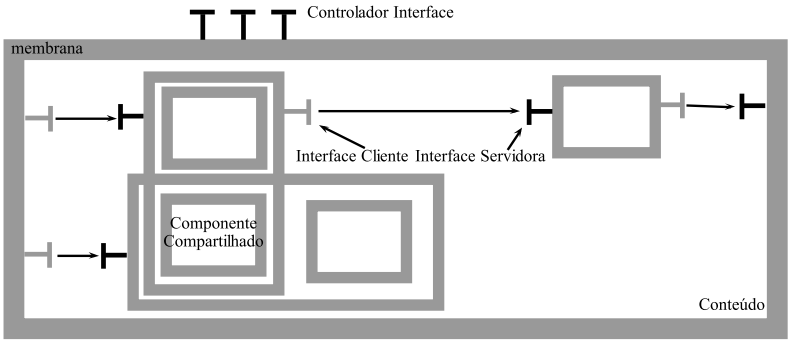


Figura 2.3: Modelo Fractal

O modelo Fractal distingue dois tipos de *ligações* entre componentes: primitivas e compostas. As primeiras são a forma mais simples de *ligação*, e representam a *ligação* direta entre portas *cliente* e *servidora* em memória compartilhada. Normalmente são efetivadas a partir de ponteiros ou referências a objetos. As últimas constituem caminhos de comunicação entre um número arbitrário de interfaces de componentes [23], contendo várias *ligações* primitivas e podendo incorporar comunicação remota entre interfaces, em plataforma de memória distribuída.

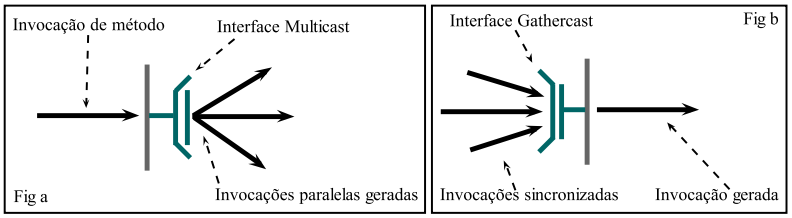


Figura 2.4: Transformação em interfaces. Fonte [16]

O suporte ao paralelismo no Fractal é originalmente definido através de conjuntos de componentes aninhados no *conteúdo* cujas interfaces são agrupadas e expostas na interface do componente dito paralelo. Tais conjuntos de portas são chamadas de *group proxies*, as quais permitem a orquestração de uma chamada coletiva externa ao componente para seus componentes aninhados. Trabalhos subsequentes propuseram

então a noção de *interfaces coletivas*, as quais podem ser do tipo *multicast*, através da qual uma porta *cliente* pode conectar-se a N portas *servidoras* ($1-N$), ou do tipo *gathercast*, por meio da qual M portas *clientes* podem conectar-se a uma porta *servidora* ($M-1$) [17]. O mapeamento de invocações de métodos em portas gathercast e multicast é ilustrado na Figura 2.4. Além dessa proposta, vale ressaltar a abordagem de comunicação através de *grupos de objetos*, voltada para plataformas orientadas a objetos paralelas e distribuídas baseadas no Fractal [14].

Entre as plataformas de componentes que oferecem suporte ao modelo Fractal, estão o ProActive [18] e Julia [2], descritos a seguir.

2.2.1 Julia

Julia é uma plataforma de componentes baseada no modelo Fractal escrita em Java, desenvolvida para plataformas leves. Utiliza a biblioteca ASM, a qual foi desenvolvida principalmente para manipulação de *bytecode* Java. O uso do ASM permite ao Julia construir em tempo de execução instâncias de componentes Fractal. Na última versão, foi incorporado o módulo *Koch*, para novas noções de controle com base na *membrana* Fractal. Julia atualmente está sendo mantido pelo consórcio OW2 [4].

2.2.2 ProActive

ProActive é uma plataforma *middleware* desenvolvida em Java para suporte a computação paralela, distribuída e móvel. Uma aplicação ProActive é desenvolvida através de entidades conhecidas como objetos ativos (*active objects*), que podem migrar entre diferentes implementações do padrão JVM. Os objetos são acessados remotamente por invocação de métodos [18]. O ProActive oferece também suporte a comunicação coletiva, através da definição de grupos de comunicação, semelhantes aos comunicadores MPI, porém em alto nível. Para desempenhar o paralelismo e distribuição de dados, adotou as *interfaces coletivas* mencionadas anteriormente. O modelo de *grupos de objetos* também foi implementado sobre a plataforma ProActive.

2.3 GCM

O GCM [16] (*Grid Component Model*) é proposto por grupos de pesquisa ligados ao consórcio europeu CoreGrid, como uma extensão do Fractal para ser aplicada em plataformas de computação em grade. O GCM especifica o mapeamento dos componentes aos recursos da grade de duas formas. A primeira é através de nós

virtuais, os quais representam abstrações capazes de separar uma estrutura virtual da estrutura física de nós, oferecendo a possibilidade de controle pelo usuário. A segunda propõe mapeamento automática, útil quando o usuário não tem informações sobre a arquitetura física. A granularidade é proposta para ser mediana, entre fina e grossa, embora flexível para qualquer granularidade.

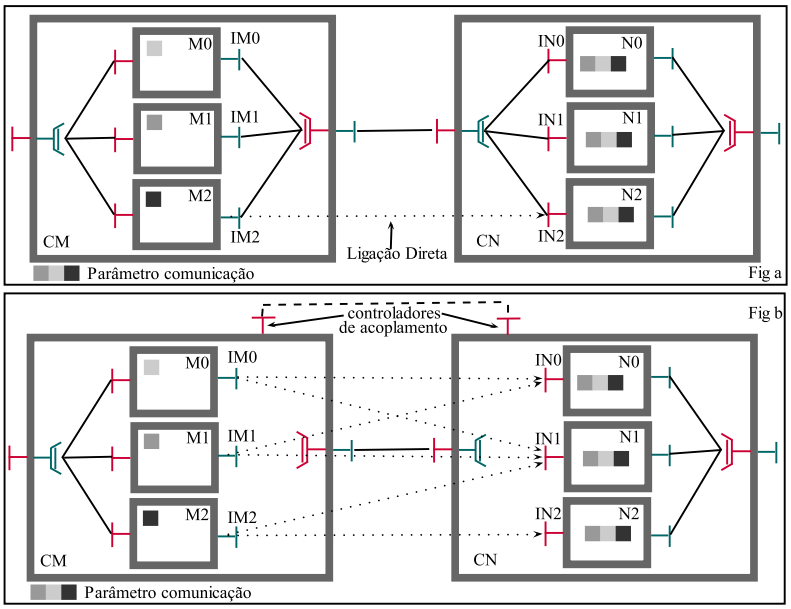


Figura 2.5: Modelo de comunicação $N \times M$. Fonte [16]

GCM suporta operações sobre interfaces coletivas, herdadas do Fractal. Introduz porém mecanismos concretos para implementação de comunicação $M-N$, através da qual um grupo de M portas clientes podem ser ligadas a um grupo de N portas servidoras. Para isso, propõe duas alternativas, ilustradas na Figura 2.5. A primeira é a ligação de portas *gathercast* com portas *multicast* para formar uma ligação $M-N$, com o potencial de ocasionar congestionamento de dados. A segunda emprega um controlador de acoplamento para intermediar as trocas de dados diretas entre as portas envolvidas.

Como discutido no caso dos *frameworks* CCA, conexões $M-N$ são muito importantes no contexto de programas paralelos em CAD. Em qualquer das duas abordagens propostas pelo GCM, o(s) cliente(s) e/ou o(s) servidor(es), precisam estar cientes da natureza paralela das chamadas sobre as interfaces das suas respectivas portas envolvidas na comunicação. Por exemplo, se o cliente e o servidor são programas paralelos que manipulam alguma estrutura de dados distribuída, o que é um caso comum, ambos devem conhecer a distribuição dos dados do lado

oposto a fim de realizarem corretamente as trocas de dados necessárias. Isso tem o potencial indesejável de forçar o acoplamento entre os componentes, e diminuindo sua coesão.

2.4 O Modelo de Componentes Hash

Com origens no Haskell$_{\#}$ [30], extensão paralela da linguagem funcional Haskell, o modelo Hash usa uma abordagem diferente dos demais modelos de componentes usados em CAD, com a finalidade de permitir maior expressividade para empregar qualquer padrão de paralelismo que poderia ser programado com bibliotecas de passagem de mensagens. Em oposição ao paradigma tradicional de programação paralela, onde os processos são tratados como unidades de decomposição do software, a qual é viável sob uma perspectiva de programação em pequena escala, o modelo Hash leva em conta os interesses de software, os quais encontram-se espalhados entre os processos que formam o programa paralelo, como unidades de decomposição. Interesses são as unidades naturais de decomposição de software nos artefatos modernos de engenharia de software de larga escala. Portanto, tal abordagem tem por objetivo aproximar a programação paralela dos artefatos de engenharia de software.

2.4.1 Interesses e processos

O modelo de componentes Hash foi proposto para suportar as técnicas tradicionais empregadas na programação por passagem de mensagem, usando bibliotecas como MPI e PVM, ferramentas que têm conseguido explorar o desempenho de arquiteturas paralelas de memória distribuída. As estratégias de comunicação e de particionamento dos processos ocorrem sob o ponto de vista dos interesses comuns entre processos. Por definição, o princípio fundamental do componente Hash pode ser entendido como: "a separação de interesses através de agrupamento de fatias (*slices*) de processos em componentes de acordo com seus interesses; e a ortogonalidade entre processos e interesses, representando unidades de decomposição do software" [31].

Os interesses representam uma abstração que possibilita a definição de unidades básicas de decomposição do software. Eles podem ser funcionais ou não funcionais. Em programação paralela, interesses funcionais podem descrever computações, como multiplicação de matrizes, solução de sistemas lineares, renderização de imagens; ou operações de comunicação, como o envio (*send*) e recebimento (*receive*) de dados, ou mesmo primitivas de comunicação coletiva. Como exemplo de interesse

não-funcional, temos o mapeamento dos processos às unidades de processamento, a escolha de um certo algoritmo para realizar um cálculo, a determinação de propriedades de uma instância de problema, etc. Um interesse está propício a conter dependência hierárquica entre fatias de vários processos, como em casos de operações de envio (*send*) e recebimento (*receive*) de dados.

2.4.2 Identificação e decomposição em interesses

A divisão dos processos em fatias representa a identificação de afinidades entre as operações realizadas pelos processos com a finalidade de identificar interesses mútuos. Para exemplificar esse processo, tomemos como modelo a seguinte operação simples de multiplicação entre matrizes e vetores: sejam A *e* B duas matrizes quadradas de dimensão n, X *e* Y dois vetores de dimensão n. Queremos decompor paralelamente a computação $r = (A \times X) \times (B \times Y)$ em interesses comuns (algoritmo - Apêndice A.3). Iremos considerar uma execução paralela em quatro processadores, onde são executados os processos p_0, p_1, p_2 e p_3. O primeiro processo (p_0) é responsável por coordenar a distribuição inicial das matrizes entre processos e coletar/somar o resultado r calculado por cada processo.

O processo p_0 possui os dados das matrizes A e B, e dos vetores X e Y. Ele então realiza as seguintes operações: divide cada matriz A e B em duas partes, superior (A_s e B_s) e inferior (A_i e B_i); deixa consigo a parte superior A_s e o vetor X; envia a parte inferior A_i e o vetor X para p_1; envia a parte superior B_s e o vetor Y para p_2; e finalmente envia a parte inferior B_i e o vetor Y para p_3.

Com a distribuição feita pelo processo p_0, é possível realizar a multiplicação das matrizes e vetores de forma paralela. Para isso, p_0 deve calcular $V_s = A_s \times X$, p_1 deve calcular $V_i = A_i \times X$, p_2 deve calcular $U_s = B_s \times Y$, e p_3 deve calcular $U_i = B_i \times Y$. A ilustração do cálculo de cada processo pode ser visto na Figura 2.6. Realizados esses cálculos, os resultados são os vetores V *e* U, distribuídos nos conjuntos de processos $\{p_0, p_1\}$ e $\{p_2, p_3\}$, respectivamente. Esses vetores devem agora ser redistribuídos entre todos os processos participantes para realização do produto interno entre eles em paralelo.

Antes de redistribuir os vetores U e V, o processo p_0 contém a partição V_s que deve ser dividido em duas partes, V_{ss} e V_{si}; o processo p_1 contém a partição V_i que deve ser dividido em duas partes, V_{is} e V_{ii}; p_2 contém a partição U_s que deve ser dividido em U_{ss} e U_{si}; e p_3 contém a partição U_i que deve ser dividido em U_{is} *e* U_{ii}.

A redistribuição é efetuada da seguinte forma: p_0 envia V_{si} para p_1; p_1

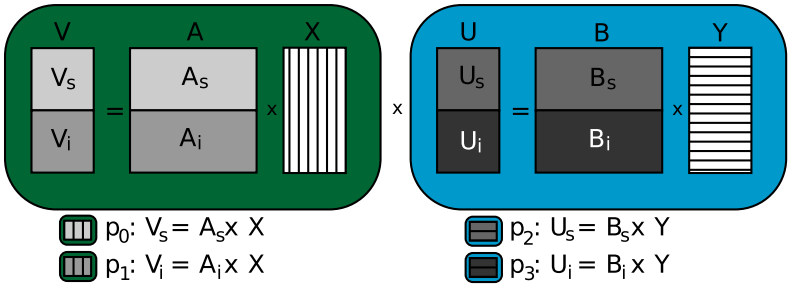


Figura 2.6: Multiplicação paralela $A \times X$ *e* $B \times Y$.

envia V_{is} *e* V_{ii} para p_2 e p_3, respectivamente; p_2 envia U_{ss} *e* U_{si} para p_0 e p_1, respectivamente; p_3 envia U_{is} para p_2. Após essa redistribuição, cada processo deverá fazer localmente o produto interno das partições dos vetores V e U, de tal forma que os processos p_0, p_1, p_2 e p_3 tenham seus resultados nas suas respectivas variáveis locais r_0, r_1, r_2 e r_3, respectivamente. Ao final do processo, os processos realizam uma soma coletiva $r0 + r1 + r2 + r3$, com o resultado sendo armazenado na variável local r em p_0, como ilustrado na Figura 2.7.

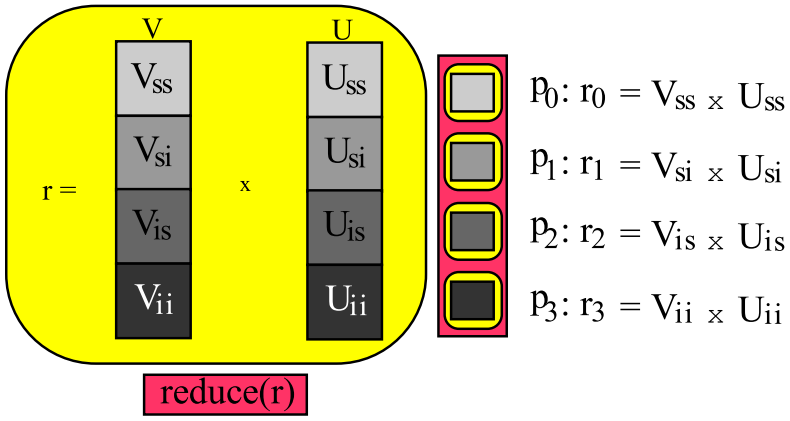


Figura 2.7: Multiplicação paralela $U \times V$.

Como observado nas Figuras 2.6 e 2.7, a decomposição ocorre tomando como critério o *fatiamento* dos processos para agrupar interesses comuns, onde a cor representa o interesse no processo. Tal abstração nos permite destacar e organizar o componente paralelo em uma abordagem mais próxima da engenharia de software. A Figura 2.8 ilustra a multiplicação paralela sob a perspectiva de interesses, com a formação de diversos componentes-#, como são chamados os componentes do modelo Hash, onde elipses representam componentes e retângulos representam processos e suas fatias. São denominadas de *unidades* as fatias de processos que

formam um componente-#, relegando o termo *fatia* às unidades de componentes aninhados que formam uma unidade do componente de mais alto nível. Na figura, você deve observar a composição hierárquica, onde o componente-# que representa a operação como um todo é formada pela combinação de outros componentes-#, ditos componentes aninhados. A essa forma de composição de componentes paralelos, atribui-se o nome de *sobreposição de componentes* [32].

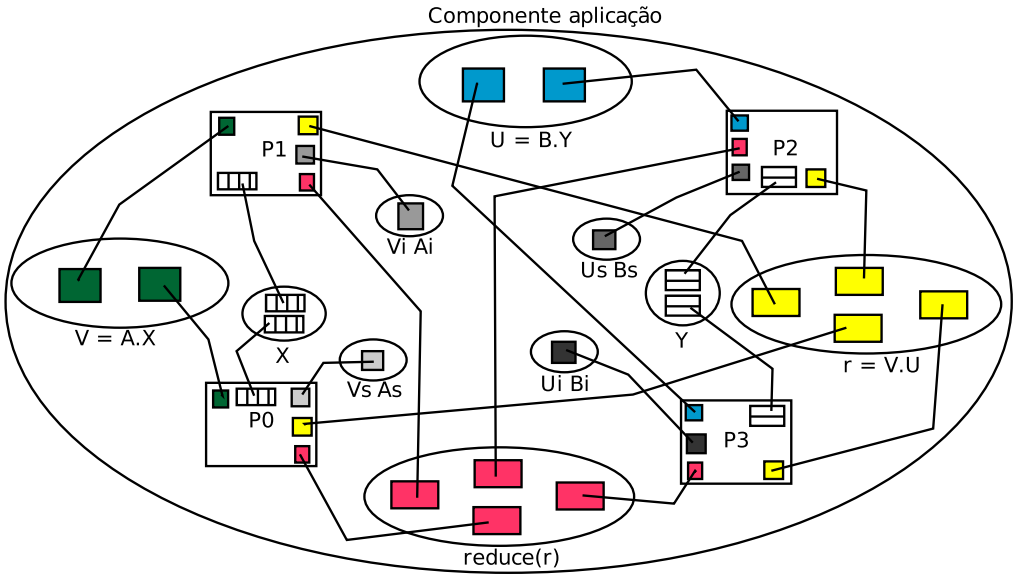


Figura 2.8: Componentes dos exemplos ilustrados nas Figuras 2.6 e 2.7

Assim, sob a perspectiva do modelo Hash, a elipse maior representa o componente responsável por dar início à computação paralela, enquanto as unidades p_0, p_1, p_2 e p_3, representadas pelos retângulos, definem os processos paralelos resultantes da composição dos componentes aninhados. Dessa forma, a partir de uma composição de componentes, sob uma perspectiva de composição de interesses de softwares, e não de processos, consegue-se obter a mesma composição de processos originalmente obtida. Note que os componentes aninhados $r = V \cdot U$ e Σr possuem fatias em todos os processos, enquanto os componentes $V = A \times X$ e $U = B \times Y$ possuem fatias apenas nos processos p_0, p_1 e p_2, p_3, respectivamente. Os componentes X e Y representam os vetores de dados para início da computação. X é de interesse de p_0 e p_1, enquanto Y é de interesse de p_2 e p_3.

2.4.3 Sistema de Programação Hash

Um sistema de programação baseado em componentes segue o modelo Hash se satisfaz três características:

- os componentes são formados por partes chamadas *unidades*, as quais podem individualmente executar em um nó de processamento de uma plataforma de execução paralela de memória distribuída;
- componentes podem ser compostos por *sobreposição*, recursivamente, formando estruturas hierárquicas;
- componentes pertencem a uma ou mais *espécies*, as quais definem conjuntos de componentes com modelos comuns de implantação, conexão e ciclo de vida.

O modelo Hash não especifica a natureza concreta de um componente #, deixando isso a cargo dos seus sistemas de programação. Isso está embutido na definição das espécies de componentes, as quais definem como os componentes da espécie são materializados em termos de unidades de software habituais e como pode ser combinados com componentes da mesma espécie ou de espécies distintas. São possíveis espécies de componentes que endereçam aspectos tipicamente não-funcionais, bem como espécies nas quais os componentes são entidades abstratas, não existindo concretamente em termos de código. Espécies de componentes podem ser usadas ainda para promover a interoperabilidade entre componentes de plataformas distintas através de um sistema de programação # projetado para intermediar a comunicação entre essas plataformas, onde os componentes de cada plataforma podem ser agrupados em espécies.

2.5 HPE (*Hash Programming Environment*)

Para suporte a componentes paralelos do modelo Hash, foi proposto o HPE (*Hash Programming Environment*). O HPE é uma plataforma de desenvolvimento e execução de componentes paralelos voltados a arquiteturas paralelas de *cluster computing*. Trata-se de uma plataforma experimental, ainda não proposta para uso em produção, voltada a experimentação e avaliação da viabilidade do uso de componentes paralelos do modelo Hash em aplicações de CAD. As seções seguintes discutem tópicos relevantes sobre o projeto e implementação atuais do HPE.

2.5.1 A Arquitetura do HPE

A arquitetura do HPE é dividida em três serviços principais: o *Front-End*, o *Back-End* e o *Core* [33]. A estrutura da arquitetura pode ser vista na Figura 2.9. Suas interfaces estão implementadas por meio de Web Services, sendo

portanto agnósticos com relação a plataforma de programação sobre a qual foram desenvolvidas e quanto a localização em um ambiente geograficamente distribuído.

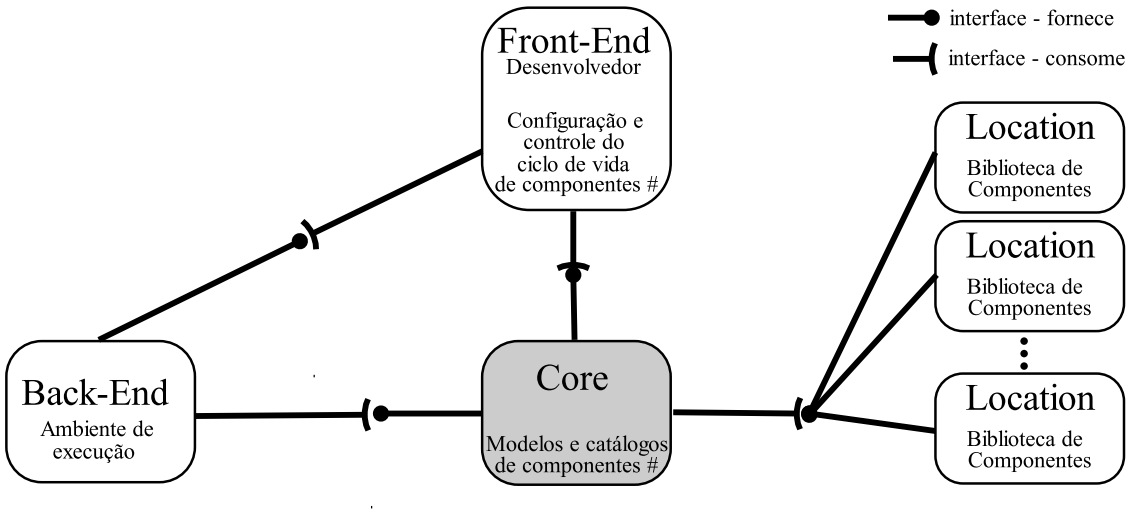


Figura 2.9: Arquitetura HPE.

Front-End

O *Front-End* representa a interface do usuário e do desenvolvedor de componentes com o HPE, através do qual é possível controlar todo o ciclo de vida de um componente #, desde sua configuração, através de uma ADL visual, passando pelo seu registro em uma biblioteca, através do serviço *Core*, até a sua implantação e utilização em uma plataforma de execução paralela, através do serviço *Back-End*. Portanto, o *Front-End* é cliente tanto do *Core* quanto do *Back-End*.

A versão atual do *Front-End* está implementada em Java, como um *plugin* para a plataforma de programação Eclipse. A interface visual de configuração de componentes #, seu elemento principal, está implementada através de outro *plugin* do Eclipse, chamado GEF (*Graphical Editing Framework*) [1].

A Figura 2.10 ilustra uma imagem da configuração visual de um componente #, chamado Farm, onde elipses denotam seus componentes aninhados e retângulos denotam suas unidades. Os quadrados internamente posicionados às unidades são fatias. As fatias de uma unidade estão ligadas por uma linha a uma unidade de um componente aninhado. Isso define a composição por sobreposição usando a interface visual do Front-End. Cada unidade está associada a um arquivo de código fonte que descreve sua implementação (na Figura 2.10, com extensão ".cs" por serem pacotes C#). Uma seta de um componente aninhado A para um componente aninhado B

indica que B é um componente aninhado público de A. No componente representado pela configuração, seus componentes públicos são aqueles com linha cheia, enquanto os demais (linha pontilhada) são ditos componentes privados. Portanto, somente **input** e **output** são visíveis por componentes que usam FARM. Note que dois componentes aninhados podem compartilhar um componente aninhado. No caso de espécies cujos componentes possuem estado, como estruturas de dados, isso significa que compartilharão as mesmas instâncias na execução. A figura indica ainda *parâmetros de contexto* da configuração, cujo significado será explicado na Seção 2.5.3.

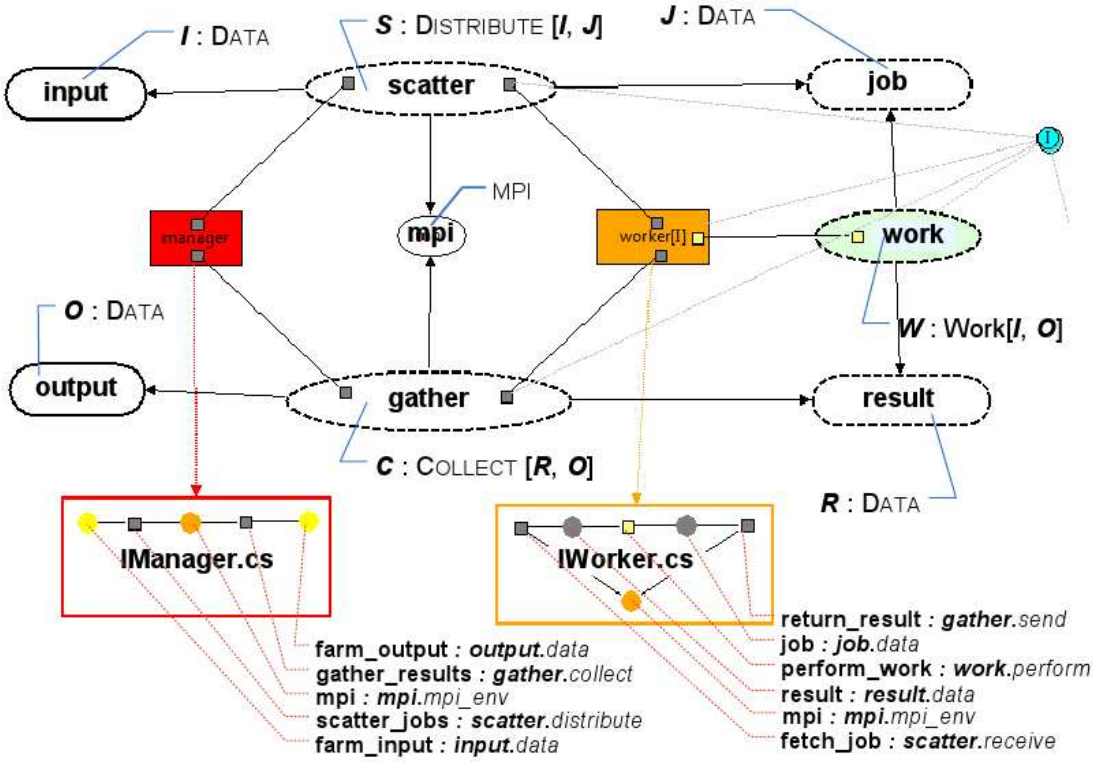


Figura 2.10: Configuração Visual de um Componente Abstrato (Front-End do HPE)

Back-End

Representa o componente HPE responsável por gerenciar e monitorar a execução de componentes Hash a partir da sua implantação. O *Back-End* está implementado sob a plataforma de execução virtual Mono, a qual segue o padrão CLI (*Common Language Infrastructure*) [6]. O padrão CLI é o resultado de esforços na indústria para dar suporte a uma nova geração de linguagens intermediárias executáveis sob máquinas virtuais, as quais trabalham com compilação *Just-In-Time* na tentativa

de obter melhores desempenhos para a computação, sendo capazes de executar código nativo de forma segura. A linguagem intermediária do padrão CLI (chamada *IL-Intermediate Language*) possui tipos polimórficos que possibilitam o suporte a várias linguagens de programação.

Uma implementação comercial do padrão CLI com grande popularidade é o .NET, oferecida pela *Microsoft Corporation*. Já o Mono [6] foi desenvolvido sob licença GPL (*GNU General Public License*) pela *Novell Corporation*, sendo de código aberto. Essa última tem sido utilizada pelo *Back-End*, embora este tenha também sido testado com sucesso sobre o .NET. Plataformas CLI possuem um módulo especial que tem a responsabilidade de gerenciar componentes implantados na máquina virtual, chamado GAC (*Global Assembly Cache*), o qual passou por uma generalização para atender as necessidades distribuídas do HPE. Essa generalização foi denominada DGAC (*Distributed Global Address Cache*) [61].

O *Back-End* do HPE segue o padrão CCA, sendo portanto considerado um *framework* CCA paralelo e distribuído. No contexto de *frameworks* CCA, o *Back-End* do HPE introduz um conjunto importante de inovações em relação às alternativas existentes que suportam paralelismo e distribuição, como o *DCA* (paralelo e distribuído), *SciRun2* (paralelo e distribuído), *CCAffeine* (paralelo), *XCAT* (distribuído) [28].

O *Back-End* do HPE é acessível ao *Front-End* por meio dos seguintes Web Services:

- *Deploy*, oferecendo serviços para implantação de componentes no *cluster*;
- *ComponentRepository*, do padrão CCA, oferecendo serviços para consultar os componentes implantados no *cluster*;
- *BuilderServices*, também requerido pelo padrão CCA, oferecendo serviços para instanciação de componentes e ligação de suas portas (*uses* e *provides*);
- *Run*, oferecendo serviços para execução e monitoração de componentes da espécie aplicação, instanciando os seus componentes transitivamente aninhados, ligando suas portas automaticamente por meio da ativação de suas portas *AutomaticSlicesPort*, as quais implementam o algoritmo de resolução dinâmica de componentes, e ativando suas portas *Go* para iniciar a sua execução.

Core

O *Core* está representado por um Web Service, atualmente desenvolvido em Java, e que tem a função de repositório de configurações de componentes. É interligado com uma ou mais *Locations*, as quais são Web Services gestores de bibliotecas de componentes em um repositório local. O *container* que tem sido usado no *Core* é o *Apache Tomcat*.

2.5.2 Espécies de Componentes do HPE

As espécies de componentes do HPE dão suporte aos componentes #, tratando de preocupações tanto funcionais como não-funcionais. As principais espécies do HPE podem ser vistas em seguida:

- **Qualificadores**: representam características não-funcionais relevantes à eficácia e à eficiência de um componente, tais como: estratégia de particionamento de uma estrutura de dados às unidades, algoritmo de solução de um certo problema, tipo de escalonamento entre *threads* no espaço de endereçamento de uma unidade, etc;
- **Plataformas**: representam as arquiteturas lógicas e físicas que apoiam a implantação de componentes #, onde unidades representam nós computacionais;
- **Ambientes**: representam tecnologias de software que dão suporte ao paralelismo nas plataformas, geralmente na forma de *middlewares* ou bibliotecas de subrotinas;
- **Computações**: faz parte de um interesse funcional da aplicação visto de forma paralela;
- **Estrutura de Dados**: representa estruturas de dados processadas por computações paralelas e trocadas entre as unidades através de mensagens;
- **Sincronizadores**: representam os recursos que coordenam a sincronização das comunicações e estado em um ciclo de execução do componente ou aplicação, como operações de comunicação ponto-a-ponto e coletivas, barreiras, etc;
- **Aplicações**: espécie que representa uma aplicação final, a qual contém o método de execução principal e que dá início às computações. A espécie

aplicação também contém computação, sendo vista portanto como pertencente à essa espécie; Dessa forma, componentes da espécie aplicação também pertencem a espécie computação.

- **Enumeradores**: Tornam possível a especificação estática de componentes com quantidades arbitrárias de unidades ou componentes aninhados. Tais unidades ou componentes aninhados são enumerações. Para uma unidade ser enumerada, deve ter a unidade de um componente enumerador como fatia. Na execução, de acordo com o mapeamento das unidades do componente aos nós de processamento da plataforma paralela, são criadas N cópias da unidade. Através da fatia enumeradora, uma unidade pode inspecionar o seu *rank* na enumeração, usado para diferenciar as unidades. Os componentes aninhados são enumerados implicitamente, segundo a enumeração das unidades que carregam suas unidades como fatias.

2.5.3 Componentes Abstratos e Contextos de Implementação

Uma característica fundamental em plataformas de componentes, também presente em linguagens orientadas a objetos, é a separação entre a especificação e o que se deve implementar. Mais especificamente em relação a uma plataforma de componentes voltados a CAD, a capacidade de especializar a implementação de um componente de acordo com suposições do ambiente no qual ele vai executar, o que envolve tanto o hardware quanto o software, permitindo ainda que essas especializações coexistam e estejam acessíveis às aplicações, é importante para explorar o desempenho potencial da plataforma de execução. O sistema de tipos Hash (HTS) aborda esses requisitos ao estabelecer dois tipos de componentes: abstratos e concretos.

Um *componente abstrato* define um contrato a ser cumprido na implementação de um conjunto de componentes concretos que implementam o mesmo interesse de software, definido pelo contrato, para diferentes suposições em relação ao ambiente de execução do componente. Ao conjunto de suposições possíveis que podem ser feitas para variar a implementação dos componentes concretos de um componente abstrato dá-se o nome de *contexto*. No HTS, o *contexto* de um componente abstrato é definido através de um conjunto de *parâmetros de contexto*, os quais assemelham-se a parâmetros de tipo de linguagens de programação usuais. Cada parâmetro de contexto formal possui um nome e um limite superior definido por um componente abstrato.

Por exemplo, considere o componente abstrato FARM, cuja configuração foi apresentada na Figura 2.10. Ele implementa um esqueleto de programação *farm*, no qual um processo gerente (*manager*) distribui trabalho para um conjunto de processos trabalhadores (*workers*) e recolhe os resultados de cada trabalho. Sua assinatura de parâmetros de contexto formais é definida a seguir, usando a notação empregada pelo HPE:

FARM[$input\_type = I$: DATA,
$output\_type = O$: DATA,
$job\_type = J$: DATA,
$result\_type = R$: DATA,
$scatter\_strategy = S$: SCATTER[$data\_source = I, data\_target = J$],
$gather\_strategy = C$: GATHER[$data\_source = R, data\_target = O$],
$work = W$: WORK[$data\_input = J, data\_output = R$]]

Nessa notação, utilizam-se variáveis associadas aos identificadores dos parâmetros de contexto a fim de evitar conflitos de nomes. Ao todo, FARM define 7 parâmetros de contexto. Esses parâmetros permitem que um componente concreto de FARM seja especializado conforme o contexto definido pelas seguinte suposições representadas pelos parâmetros:

- *input_type*: tipo da estrutura de dados da *entrada* que será particionada em tarefas (*jobs*), as quais serão distribuídas entre os processos trabalhadores;
- *job_type*: tipo da estrutura de dados que representa uma *tarefa*;
- *scatter_strategy*: tipo do componente abstrato que mapeará a *entrada* em *tarefas* que serão distribuídas entre os processos trabalhadores;
- *work*: tipo do componente abstrato que define a computação realizada sobre uma *tarefa* pelos processos trabalhadores;
- *result_type*: tipo da estrutura de dados que representa o *resultado* de uma *tarefa*;
- *gather_strategy*: tipo do componente abstrato que mapeará os *resultados* calculados por cada processo trabalhador na *saída*;
- *output_type*: tipo da estrutura de dados que representa a *saída* da computação realizada pelo FARM.

Por exemplo, é possível definir a implementação de um componente concreto genérico para FARM, o qual denominar-se-á FarmImpl, suprindo os seus parâmetros de contexto com os seus próprios limites superiores:

$$\begin{aligned}\mathsf{FarmImpl} := \textbf{FARM}[&\mathit{input\_type} = \text{DATA},\\ &\mathit{output\_type} = \text{DATA},\\ &\mathit{job\_type} = \text{DATA},\\ &\mathit{result\_type} = \text{DATA},\\ &\mathit{scatter\_strategy} = \text{SCATTER}[\mathit{data\_source} = \text{DATA},\\ &\qquad\qquad \mathit{data\_target} = \text{DATA}],\\ &\mathit{gather\_strategy} = \text{GATHER}[\mathit{data\_source} = \text{DATA},\\ &\qquad\qquad \mathit{data\_target} = \text{DATA}],\\ &\mathit{work} = \text{WORK}[\mathit{data\_input} = \text{DATA},\\ &\qquad\qquad \mathit{data\_output} = \text{DATA}]]\end{aligned}$$

Por ser genérico, esse componente concreto poderá ser ligado a qualquer componente aninhado de algum componente em execução cujo tipo seja FARM aplicado a algum contexto bem formado, ou seja, um contexto no qual cada parâmetro de contexto formal é suprido por um componente abstrato cujo tipo é um subtipo do limite superior do parâmetro. Por exemplo, seja o componente abstrato NUMERICALINTEGRATOR, cujos componentes concretos implementam uma integração numérica de forma paralela utilizando um *farm*. Para isso, possui um componente aninhado chamado ***farm*** do seguinte tipo:

$$\begin{aligned}\textbf{FARM}[&\mathit{input\_type} = \text{INTEGRALCASE}[F],\\ &\mathit{output\_type} = \text{DOUBLE},\\ &\mathit{result\_type} = \text{LIST}[\text{DOUBLE}],\\ &\mathit{job\_type} = \text{LIST}[\text{INTEGRALCASE}[F]],\\ &\mathit{scatter\_strategy} = \text{DISTRIBUTEINTERVAL}[\mathit{data\_source} = \text{INTEGRALCASE}[F],\\ &\qquad\qquad \mathit{data\_target} = \text{INTEGRALCASE}[F]],\\ &\mathit{gather\_strategy} = \text{SUMAREAS}[\mathit{data\_source} = \text{DOUBLE},\\ &\qquad\qquad \mathit{data\_target} = \text{DOUBLE}],\\ &\mathit{work} = \text{APPROXIMATEINTEGRAL}[\mathit{data\_input} = \text{LIST}[\text{INTEGRALCASE}[F]],\\ &\qquad\qquad \mathit{data\_output} = \text{DOUBLE}]]\end{aligned}$$

, onde F é a variável associada ao parâmetro de contexto *function* de NUMERICALINTEGRATOR, o qual define o componente abstrato que especifica a função que será integrada no intervalo $[0, 1]$.

Portanto, em uma integração numérica conforme NUMERICALINTEGRATOR, a *entrada* é definida por um componente do tipo INTEGRALCASE[F], o qual define um caso de integração como uma função representada por um componente do tipo F associado a um intervalo de integração pré-definido em $[0, 1]$. Um caso de integração é quebrado por um componente do tipo DISTRIBUTEINTERVAL em várias listas de casos de integração que definem as *tarefas*, particionando-se o intervalo inicial em vários sub-intervalos. Assim, cada processo trabalhador deverá receber uma lista de casos de integração e calcular, através de um componente do tipo APPROXIMATEINTEGRAL, um valor de ponto-flutuante de precisão dupla que será encapsulado em um componente do tipo DOUBLE, representando o *resultado* de uma *tarefa*. Os resultados calculados por cada processo trabalhador serão coletados por um componente do tipo SUMAREAS, o qual calcula a soma dos *resultados* e o encapsula em um componente do tipo DOUBLE que define a *saída*.

Os componentes abstratos associados aos parâmetros de contexto do tipo de um componente aninhado, como ***farm***, especificam tanto o contexto que será usado para o *sistema de resolução automática* de componentes implementado pelo *Back-End* escolher uma implementação apropriada de FARM quanto o contexto atual que será aplicado ao componente concreto escolhido. Sob a primeira perspectiva, o componente abstrato aplicado a um contexto define um polimorfismo existencial, permitindo especializar a implementação do componente aninhado de acordo com o contexto, enquanto sob a segunda perspectiva, define um polimorfismo universal, uma vez que o componente concreto pode ser mais genérico do que o contexto requisitado, possuindo parâmetros de tipo cujos limites superiores são definidos pelos parâmetros de contexto atuais que definiram o contexto de implementação do componente concreto. Por exemplo, o componente aninhado ***farm*** de NUMERICALINTEGRATOR pode ser ligado em tempo de execução a uma instância do componente concreto FarmImpl, pois seu tipo é um subtipo do tipo associado a ***farm***. Caso houvesse alguma implementação de FARM mais específica, seria escolhida ao invés de FarmImpl. Por exemplo, alguém poderia desenvolver um componente concreto para FARM especializado para lidar com o caso onde tarefas são listas de itens a serem submetidos aos processos trabalhadores, como é o caso do tipo de ***farm***. De fato, o HTS tem sido formalizado pelo mapeamento a tipos existenciais e universais.

2.5.4 A Implementação de Componentes do HPE

A implementação de componentes do HPE é atualmente realizada por meio da linguagem C#, uma vez que o *Back-End* está atualmente implementado sobre a plataforma de execução virtual CLI/Mono. Entretanto, o suporte a outras linguagens é possível, embora não implementado ainda, tendo em vista que essa plataforma suporta múltiplas linguagens.

Para configuração de componentes abstratos e concretos, o desenvolvedor tem o auxílio do ambiente gráfico da plataforma Eclipse oferecido pelo *framework* GEF, sobre o qual está implementado o *Front-End*. Através dessa interface visual de configuração, é possível também modelar e gerar a estrutura fundamental do código do componente, através da geração de *classes* e *interfaces*, respectivamente associadas às unidades de componentes concretos e componentes abstratos. Embora uma estrutura de código padronizada seja oferecida, com o suporte a geração de código que abstrai os detalhes da programação segundo o modelo do CCA, bem como a definição de parâmetros de contexto através de tipos genéricos, o *Front-End* deixa livre a manipulação de códigos pelo desenvolvedor, responsável de fato pela programação dos componentes. De fato, como componentes do HPE seguem o padrão CCA, o modelo de programação do CCA é plenamente suportado e o programador poderia abdicar do recurso de geração de código do *Front-End*.

arquivo base:

```
namespace pkg \{
   public interface BaseIVerify<C>:IComputationKind where C:IClass \{
          IMultiPartitionCells<C> Cells_info {get;}
          IProblem<C> Problem {get;}
   \}
\}
```

arquivo de usuário:

```
namespace pkg \{
     public interface IVerify<C>:BaseIVerify<C> where C:IClass \{
           void validateSum(double[] sum);
     \}
\}
```

Figura 2.11: Codificação exemplo de um componente abstrato

Os códigos gerados são organizados para cada componente, abstrato ou concreto, em dois arquivos. Um arquivo *base* que contém definições de *propriedades* associadas às unidades de componentes aninhados (fatias) que constituem a unidade em questão, as quais definem portas *uses*, e um arquivo de *usuário*, que herda o arquivo base, aonde o programador deve escrever o código referente ao interesse tratado pelo componente (porta *provides*).

Na classe *base* associada a uma unidade de um componente concreto, é gerado o código para instanciar as portas *uses* através da chamada ao método *getPort* da interface *Services* do CCA. Essas chamadas só são de fato realizadas quando a propriedade associada a porta é acessada.

Tanto a classe/interface base quanto a classe/interface de usuário são genéricas a fim de implementar os parâmetros de contexto. De fato, os parâmetros de tipos correspondem a projeção dos parâmetros de contexto do componente sobre a unidade. No caso das classes, o limite superior do parâmetro está associado ao valor atual do parâmetro de contexto associado ao componente concreto em questão, de forma que o componente concreto possa ser aplicado em contextos mais específicos em relação aquele para o qual foi desenvolvido, mas nunca em um contexto mais geral.

Em componentes abstratos, a interface *base* estende uma interface pré-definida para a espécie do componente (Seção 2.5.2), como *IComputationKind*, *IApplicationKind*, *IDataStructureKind*, dentre outras que atualmente estão disponíveis. A interface de usuário contém métodos e propriedades que componentes concretos podem implementar.

A Figura 2.11 mostra um exemplo de arquivos gerados pelo HPE, onde temos a interface *IVerify*, associada a única unidade de um componente abstrato chamado VERIFY, o qual é genérico no parâmetro C, cujo limite superior é *IClass*, uma vez que o componente VERIFY possui um parâmetro de contexto *class* correspondente, cujo limite superior é definido pelo componente abstrato CLASS. De fato, *IClass* é a interface associada à unidade de CLASS. Nesse exemplo, a interface *base* contém duas propriedades, *CellsInfo* e *Problem*, responsáveis por informar a estrutura de bloco e característica do problema, para que o vetor *sum* seja validado pelo método *validadeSum*. Essas prorpriedades são provenientes dos componentes aninhados públicos ***cells_ info*** e ***problem*** de VERIFY.

Em componentes concretos, a classe *base* implementa a interface *base* do componente abstrato. Adicionalmente, a classe de usuário do componente concreto implementa a *interface* de usuário do componente abstrato, e estende a classe base pertencente ao próprio componente concreto. A Figura 2.12 expressa o relacionamento de extensão e implementação entre componentes abstratos e concretos. Toda classe associada a unidade de componente concreto herda a classe *UnitImpl*, cujas propriedades e métodos definem várias informações importantes, como o objeto *services* necessário pelo padrão CCA, o comunicador MPI associado

ao componente, o *rank* da unidade que permite diferenciá-la das demais unidades do componente, etc. A classe *UnitImpl* implementa a interface *IUnit*, a qual por sua vez pertence a hierarquia de qualquer interface de unidade de componente abstrato.

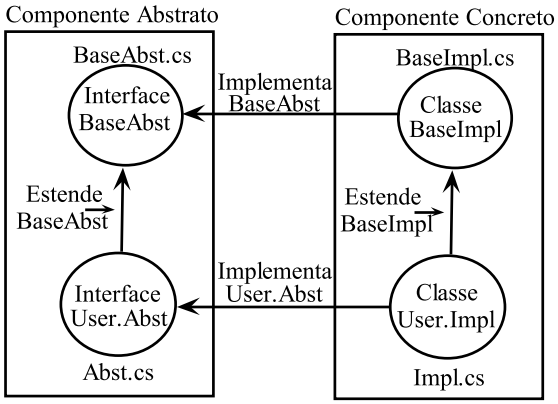


Figura 2.12: Relacionamento entre componentes abstratos e concretos

2.5.5 Considerações sobre Avaliação de Desempenho do HPE

Neste capítulo, podemos conhecer algumas das atuais plataformas e modelos de componentes paralelos. O ênfase maior é dado ao HPE, objeto de estudo desta dissertação. É notório que o HPE faz uso de várias tecnologias, como Java, Mono, MPI.NET e Web Services. Tal agrupamento tecnológico é um dos motivos a ser considerado para uma avaliação e validação integrada dessas tecnologias sob o modelo de componentes Hash. Outro motivo é a necessidade de se ter um estudo de desempenho com componentes de domínio científico operando sobre plataformas de componentes paralelos, especificamente o HPE, visando assim subsídios para que em futuras pesquisas se possa otimizar ou desenvolver recursos para serem aplicados em CAD. Sendo assim, o processo de estudo de desempenho é feito no Capítulo 5 desta dissertação, tendo como característica a abordagem definida pela fundamentação teórica do Capítulo 3, e a utilização de programas do NPB (NAS Parallel Benchmark), os quais foram refatorados em componentes, como será apresentado no Capítulo 4.

Capítulo 3

Avaliação de Desempenho de Sistemas Computacionais

Os aspectos de desempenho em dispositivos e sistemas computacionais são aqueles que na maioria das vezes regem a preferência de pessoas, empresas ou instituições quando necessitam de soluções para a computação de problemas nos seus diversos domínios de aplicação. Devido a isso, o objetivo de obter o maior desempenho com menor custo está relacionado tanto na concepção de produtos quanto na sua aquisição. A ciência da computação, especialmente a área de computação de alto desempenho, possui um papel fundamental nesse contexto. Para isso, os profissionais podem fazer o uso de fundamentos, conceitos e técnicas de avaliação de desempenho que são aplicadas em diversas situações, com o objetivo de adquirir informações relevantes sobre respostas de desempenho de um determinado sistema com o maior grau de confiança possível, possibilitando otimizações. A avaliação é feita principalmente quando o projetista necessita efetuar escolhas entre duas ou mais alternativas para a construção ou análise de um projeto. Como exemplo, poderíamos enumerar situações de escolhas entre dispositivos de hardware, redes, topologia de redes, algoritmos paralelos, algoritmos de roteamento de pacotes, escalonadores de processos, sistemas de arquivos, compiladores, máquinas virtuais, modelos de componentes, dentre outros. Caso não se queira escolher entre alternativas, ainda assim a análise de desempenho é fundamental, pois nos permite verificar o quanto um sistema ou solução está desempenhando seu papel de forma eficiente.

A grande dificuldade em avaliar o desempenho de um sistema computacional está na amplitude que a computação envolve, com inúmeras soluções e situações.

Não há medida, ambiente ou técnica padrão para essa finalidade [47]. Dessa forma, inicialmente é preciso tomar conhecimento de medidas de desempenho, bem como de ambientes e técnicas existentes de avaliação de desempenho para serem aplicadas de forma pertinente ao propósito de cada avaliação. De posse desses conhecimentos, o analista de desempenho tem os subsídios necessários para realizar avaliações direcionadas para uma determinada necessidade.

3.1 Rigor metodológico

O primeiro passo em um estudo de desempenho é estabelecer os procedimentos e técnicas a serem seguidas. Como alternativa, existe a sistemática proposta por Jain [47] em 1991, a qual pode ser usada para gerir estudos e avaliação de desempenho, adicionando um maior rigor metodológico nas pesquisas. Essa sistemática apresenta uma lista de questionamentos fundamentais que propõem-se a evitar erros comuns em muitas situações (Tabela 3.1). A partir dessas perguntas, estabelecem-se atividades básicas que definem uma sequencia de etapas a serem seguidas. Os parágrafos subsequentes propõem-se a detalhar essas atividades, as quais constituem uma forma de reduzir riscos de erros nos resultados finais da análise.

- **Definição de objetivos, bem como o planejamento da análise**: nessa atividade, é possível fazer algumas reuniões de planejamento com pessoas interessadas nos resultados da análise, o que ajudaria a definir objetivos gerais, específicos, e particularidades da avaliação. Pode-se definir também o cronograma e os requisitos da análise.

- **Evitar tendências e conceitos predefinidos**: a abordagem dos objetivos de forma imparcial na análise de desempenho é de fundamental importância para garantir resultados confiáveis. Por exemplo, ao comparar um ou mais sistemas, deve-se abolir conceitos como "o nosso sistema é melhor do que o outro". É importante que os testes, medidas ou técnicas não favoreçam resultados.

- **Seleção de métricas de desempenho**: as métricas determinam a forma como se aferir o desempenho de um sistema. Nos objetivos, pode-se especificar uma ou mais métricas a serem utilizadas, de acordo com os propósitos da avaliação. Exemplos de métricas: tempo de resposta, volume de trabalho realizado, precisão de cálculo, taxa de disponibilidade, taxa de erros, taxa de quebra, dentre outras.

Tabela 3.1: Lista de questionamentos [47].

1. Objetivos claramente definidos?
2. Objetivos imparciais?
3. Os passos da análise seguem uma sistemática?
4. Sistema bem entendido para analisá-lo?
5. Métricas relevantes para o sistema?
6. Carga de trabalho correta para o sistema?
7. Técnica apropriada?
8. Listagem de parâmetros do sistema?
9. Descrição dos fatores e níveis?
10. O projeto experimental está correto?
11. Foi explorado o nível apropriado de detalhes?
12. Os dados medidos foram interpretados e analisados?
13. A estatística está correta?
14. Variância de entrada observada?
15. Variância de resultados analisada?
16. Análise fácil de explicar?
17. Estilo adequado de apresentação de resultados?
18. Houve maximização de apresentação de resultados na forma gráfica?
19. Limitações e premissas da análise estabelecidas?

- **Identificação e descrição dos parâmetros do sistema**: o comportamento de um sistema é determinado pelos seus parâmetros. Tais parâmetros representam as características agregadas ao sistema, como também à sua carga de trabalho. Por exemplo, ao analisar o desempenho de uma rede de computadores, os parâmetros são características como o tipo de cabeamento, a topologia, a categoria de cabo, a largura de banda, etc.

- **Decisão dos fatores a serem estudados**: em um processo de análise, pode-se descrever diversos parâmetros do sistema. Entretanto, analisar todos esses parâmetros pode representar uma tarefa árdua frente aos recursos e tempo disponível para realizar a avaliação. Muitas vezes, alguns parâmetros não possuem relevância para contribuir com os resultados finais da análise. Por esse motivo, escolhem-se parâmetros de maior importância, os quais são chamados de fatores. A característica essencial de um fator é que ele é

constituído de um ou mais níveis. Por exemplo, suponha a análise de um servidor de arquivos. Esse servidor provavelmente contém dispositivos de armazenagem de dados, capacidade de memória, freqüência do processador, dentre outros atributos. Destacando o fator memória, poderia-se verificar níveis que estão associados à quantidade de memória disponível, de forma discreta: dois, quatro ou oito *gigabytes*. A identificação correta dos níveis terá impacto fundamental no processo de avaliação desse servidor.

- **Seleção de cargas de trabalho**: a carga de trabalho é um fator que está relacionado com o tipo e o volume de serviço que é atribuído ao sistema. No critério de escolha da carga de trabalho para a qual o sistema será avaliado, deve-se levar em consideração níveis que caracterizam atividades padrões e de pico do sistema durante a execução.

- **Escolha de técnicas**: a escolha das técnicas representa uma etapa importante no processo de avaliação de desempenho. Atualmente, estão classificadas como modelagem analítica, simulação e medição. Cada uma possui vantagens e desvantagens, o que sugere, em algumas situações, o uso conjunto a fim melhorar os resultados da avaliação, especialmente com problemas de grande complexidade [9,22]. Na Seção 3.2, são abordados detalhes sobre essas técnicas.

- **Interpretação e análise de dados**: a atividade de interpretar e analisar os dados de experimentos que envolvem um ou mais sistemas pode ser realizada com o apoio de ferramentas estatísticas. Os resultados estatísticos permitem estabelecer indicadores de desempenho para auxílio nas tomadas de decisões, uma vez que cada experimento está condicionado a valores aleatórios. Será visto com maiores detalhes no Capítulo 5 algumas possibilidades para comparar sistemas através de fundamentos estatísticos.

- **Apresentação de resultados**: a apresentação dos resultados é a etapa responsável pela exposição de um grande número de informações para tomadores de decisões. É importante que tenha-se alguns cuidados com exposições de textos, gráficos e tabelas. Deve-se buscar clareza e simplicidade, e dar preferência para gráficos que ofereçam ao leitor facilidade para encontrar informações, para que ele recorra aos textos apenas para maiores detalhes.

3.2 Técnicas

A atividade de avaliar o desempenho de um sistema exige o conhecimento do maior número possível de suas características. Ainda assim, não é uma tarefa fácil. O simples fato de executar e observar o tempo de resposta não proporciona informações abrangentes sobre o estado de desempenho de um software ou dispositivo. O software possui muitas instruções para serem executadas por microprocessadores, os quais são constituídos por inúmeros transistores que operam em altas freqüências. Cada segundo de processamento normalmente representa bilhões de instruções dependendo do microprocessador. Devido a isso, o grau de precisão desejado em uma análise é fortemente relacionado com as dificuldades que poderão ser encontradas e com técnicas que deverão ser aplicadas. A complexidade na análise pode ainda aumentar quando está envolvido o processamento paralelo. Quando bem aplicado, representa ganhos consideráveis em desempenho. Quando mal aplicado, representa perdas no desempenho ou ganhos pouco significativos perante as expectativas. Atualmente, existem três técnicas para análise de desempenho: modelagem analítica, simulação e medição.

A modelagem analítica é voltada para modelos matemáticos que descrevem as variáveis do sistema que se deseja avaliar, como cadeias de Markov [68] ou Redes de Petri, capazes de esboçar detalhes sobre diversos aspectos de um sistema. A modelagem analítica pode ser usada para fazer avaliações em cada estágio do desenvolvimento de um sistema, permitindo que durante todas as etapas do projeto sejam tomadas decisões entre alternativas. Um modelo analítico não é simples de ser especificado, especialmente quando a modelagem envolve sistemas paralelos ou concorrentes. A técnica de simulação consiste em usar recursos de linguagens de programação para simular atividades específicas, necessárias para otimização ou desenvolvimento de sistemas. A técnica de medição consiste na instrumentação de um sistema desenvolvido, do qual desejamos obter resultados de experimentos com base em suas métricas. As principais características para a escolha de uma ou mais técnicas são apresentadas na Seção 3.2.1.

3.2.1 Características

O primeiro indicador para decisão sobre o uso de uma ou mais técnicas está relacionado com a fase do ciclo de vida do sistema. A técnica de medição possui a limitação de que apenas sistemas prontos podem ser medidos. Geralmente, usamos medições para melhorar sistemas existentes e propor novas versões. Diferentemente,

simulação e modelagem analítica podem ser usadas em todos os estágios do ciclo de vida do sistema.

Um outro indicador que tem influência sobre a escolha entre as técnicas é o tempo que requerem, parâmetro decisivo para situações em que o analista deve dar respostas rápidas. Para cronogramas curtos, sugere-se modelagem analítica. A técnica de medição possui tempo variável pois envolve características particulares de cada sistema a ser medido. É também considerada a mais instável em termos de tempo, podendo ser rápida quando tudo ocorre bem ou extremamente demorada quando ocorrem problemas diversos. A simulação é a técnica de maior previsibilidade no consumo de tempo, uma vez que provavelmente recorrere-se às linguagens de programação com a finalidade de caracterizar detalhes simulados de um sistema para a análise.

Um outro indicador é a precisão. Como primeira opção, a simulação se comporta como uma ótima alternativa, pois permite explorar detalhes na busca da precisão ideal, através de linguagens de programação ou ferramentas que simulem o comportamento do sistema. Como segunda opção, a medição é a técnica que apresenta aspectos variantes. Embora a expectativa seja de precisão real, os indicadores podem variar de maneira não esperada. Isso é justificável porque o estado ideal das interações entre os parâmetros pode não ser atingido. De fato, explorar ou simplesmente adquirir informações sobre interações unicamente com medição talvez não seja a melhor opção. Instabilidades incentivadas por características aleatórias do ambiente são problemas a tratar para garantir o alcance de resultados precisos [47]. Como terceira opção, temos a modelagem analítica, para a qual devemos avaliar que seus princípios passam por suposições e modelos que se abstraem do comportamento de um sistema. Mas não é o sistema, fato que torna incoerente a esperança de grande acurácia.

Um outro aspecto a ser levado em conta é o custo. Para o fator custo, a modelagem analítica possui destaque. O requisito financeiro fica apenas por conta do tempo e conhecimento intelectual do analista. Oposto a isso, a medição exige o equipamento real e a preparação do ambiente para o equipamento, além do tempo do analista. A simulação pode ser usada para diminuição de custos, especialmente para testar e verificar a viabilidade do comportamento de sistemas caros, com base em modelos que buscam ser fiéis ao sistema real. Tais modelos podem ser desenvolvidos com o auxílio de ferramentas ou linguagens de alto nível.

Apesar das vantagens atribuídas à modelagem analítica, essa é a técnica de

Tabela 3.2: Características principais. Fonte [47].

Critério	**Modelagem Analítica**	**Simulação**	**Medição**
Ciclo de vida	Qualquer	Qualquer	Pós-Protótipo
Requisito de tempo	Pouco	Médio	Variante
Ferramentas	Analistas	Linguagens	Instrumentação
Precisão	Baixa	Moderada	Variante
Custo	Baixo	Médio	Alto
Aceitável	Baixo	Médio	Alto

menor aceitação quando se pretende convencer pessoas sobre resultados. O índice de confiança fica maior quando são usadas medições, e mediano quando são usadas simulações. É considerado mais fácil convencer pessoas quando dispõe-se do amparo de uma medição real. A Tabela 3.2 faz um resumo das principais características das técnicas para análise de desempenho.

3.2.2 Métricas

Quando analisa-se o desempenho de um sistema computacional, deve-se estabelecer e organizar métricas para quantificar o comportamento esperado de determinados serviços, como roteamento de pacotes, processamento de imagens, processamento de equações diferenciais, dentre outros. Quanto maior o número e a complexidade dos serviços, maior o número de métricas que poderemos utilizar. Para definir essas métricas, primeiramente é necessário definir o que é esperado de um serviço. De modo geral, espera-se a execução correta, a execução incorreta ou a não execução, expectativas descritas a seguir:

- **execução correta**: mensurado por **métricas de velocidade**, permitindo dizer o quão rápido, lento ou estável é o serviço.
- **execução incorreta**: mensurado por **métricas de confiabilidade**, permitindo dizer o quanto de credibilidade e precisão tem o serviço.
- **não execução**: mensurado por **métricas de disponibilidade**, permitindo dizer o quão é possível dispor do serviço.

Classificação XM

Outro aspecto que devemos estabelecer é a classificação quanto aos melhores valores das métricas. Também classificadas em [47, 60][1], as classes são:

- **AM (Alto Melhor)**: para algumas métricas, é desejado que os valores sejam os maiores possíveis. Por exemplo, a métrica **mops** tenta medir as milhões de operações por segundo que um microprocessador é capaz de processar.
- **BM (Baixo Melhor)**: para outras métricas, é desejado que os valores sejam os menores possíveis. Por exemplo, a métrica latência é o tempo gasto por um evento, como a leitura de dados da memória pelo microprocessador.
- **NM (Nominal Melhor)**: temos também as métricas das quais são desejáveis valores que não sejam baixos nem altos. São esperados valores próximos a um valor esperado pré-estabelecido. Um exemplo é a métrica *taxa de solicitação de serviço* de um sistema. Sugere-se que, em alguns casos, uma baixa taxa de solicitação represente serviço irrelevante, enquanto uma alta taxa represente congestionamento.

3.2.3 Cargas de trabalho

O processo de medir sistemas reais ou simulados tem como princípio o uso de cargas de trabalho, as quais são usadas para auxiliar no trabalho de coletar informações sobre o desempenho de um sistema, devendo ser escolhidas rigorosamente. Para o processo de escolha, é necessário entender alguns aspectos fundamentais sobre cargas que foram preparadas para determinadas atividades. Historicamente, quando os computadores executavam apenas instruções matemáticas fundamentais, tais como adição, multiplicação, divisão e subtração, analisava-se o desempenho através de cargas preparadas para explorar a capacidade de cálculo sob essas instruções, especialmente a adição. Então, foram propostas as cargas do tipo "Instruções Aditivas" [41,47]. Depois vieram algumas funcionalidades como indexação e tomadas de decisão (E, OU), e esses novos recursos tiveram que ser considerados. Então foram propostas as cargas do tipo "Instruções Mistas" [41]. Posteriormente, tivemos novos recursos, como instruções de *cache* e *pipelines*, os quais deveriam ser analisados de forma conjunta às demais instruções. Para isso, pesquisadores observaram serviços mais freqüentes relacionados com

[1] HB (*Higher is Better*), LB (*Lower is Better*), NB (*Nominal is Best*)

os processadores e definiram funções que são aplicadas como carga de trabalho. O conjunto dessas funções ficou conhecido como *kernels* [41, 49]. Esses *kernels* iniciais tinham a desvantagem de não serem projetados para explorar dispositivos de entrada e saída ou chamadas de serviços de sistemas operacionais. Preocupavam-se apenas com aspectos de desempenho relacionados aos processadores. Como muitas tecnologias não estão voltadas apenas para o processamento bruto, as operações de entrada e saída passaram a ser parte de uma carga de trabalho realística. Então, foi proposto o conceito de "programas sintéticos", também conhecidos como "aplicações simuladas", como carga de trabalho [41, 47, 49]. Esses programas possuem laços de instruções responsáveis por fazer finitas chamadas de serviços ou finitas solicitações de entrada e saída para explorar o desempenho de vários recursos em sistemas operacionais. Essa técnica é fundamentada pelo princípio do laço exercitador proposto por Buchholz em 1969 [25]. De fato, o conceito de carga de trabalho é muito útil para a proposta de ferramentas analíticas de recursos de sistemas operacionais e processadores, para que sejam viabilizados subsídios para otimizações. Dessa forma, foi possível o surgimento de um maior número de ferramentas direcionadas a explorar maiores peculiaridades em sistemas, de modo que foram propostos os *benchmarks* de aplicações.

De modo geral, os tipos de cargas de trabalho podem ser escolhidos conforme critérios e necessidades de cada analista de desempenho. Na Seção 3.3, discutimos mais detalhes sobre *benchmarks* de uso disseminado.

3.3 Benchmarks

Aplicações contemporâneas de computadores possuem muitos aspectos comportamentais que podem definir sua eficácia e eficiência. Cada comportamento pode estar associado a muitos fatores que são levados em conta em um processo de avaliação. A avaliação desses fatores necessita de artefatos capazes de explorar o potencial de recurso de cada alternativa a ser avaliada, permitindo medir seu potencial de desempenho. Para isso, temos como alternativa o uso de *benchmarks*. Comercialmente, o termo *benchmark* é referenciado como carga de trabalho, a qual é destinada a ser submetida a um sistema para que se efetue *benchmarking*, nomenclatura usada para o processo de medir e comparar desempenho entre dois ou mais sistemas. A Tabela 3.3 apresenta alguns *benchmarks* de uso disseminado que são descritos nos próximos parágrafos.

Tabela 3.3: Categorias e Exemplos de *Benchmarks*

Categoria para Avaliação	**Exemplo de *benchmarks***
Computadores uniprocessados	*SPEC CPU* [40]
	SciMark [59]
Computadores paralelos	SPLASH [69]
	NAS Parallel Benchmarks [15]
	DARPA HPCS Benchmarks

3.3.1 SPEC

O *Standard Performance Evaluation Corporation* (SPEC) consiste em *benchmarks* desenvolvidos cooperativamente por um pequeno grupo de fornecedores de estações de trabalho em 1988, que perceberam a necessidade de mercado para testes de desempenho padronizados. A metodologia foi baseada em fazer *benchmarking* através de códigos fonte que representavam aplicativos existentes para uma variedade de plataformas, desenvolvidas por membros do grupo proponente. A cooperação resultou em *benchmarks* desenvolvidos de acordo com aplicações reais dos usuários, que medem desempenho de processador, memória e compiladores [40]. O conjunto é composto por quatorze (14) programas de ponto flutuante escritos em C/Fortran e onze (11) programas para inteiros. A versão livre para *download* é desenvolvida em Java (*SPECjvm2008*).

3.3.2 SciMark

O SciMark inclui *benchmarks* escritos em Java, desenvolvidos por Roldan Pozo e Bruce Miller através do *National Institute of Standards and Technology* (NIST), nos Estados Unidos. Possui o propósito de medir o desempenho de algoritmos numéricos que são comuns em aplicações científicas e de engenharias. É composto de cinco núcleos: FFT, relaxamento *Gauss-Seidel*, multiplicação de matrizes esparsas, integração *Monte Carlo* e fatoração LU. O principal objetivo do SciMark é estabelecer um indicador de desempenho desses núcleos na máquina virtual Java. Para isso, são definidos problemas pequenos que isolam os efeitos da memória com a finalidade de concentrar a avaliação na JVM, JIT[2] e CPU [49].

[2] Compilador *Just In Time*.

3.3.3 SPLASH

O SPLASH representa *benchmarks* paralelos que são utilizados como carga de trabalho para explorar desempenho de sistemas de computação para aplicações em computação gráfica, ciências e engenharia. Foi proposto por um grupo de pesquisadores da Universidade de *Stanford* [69], nos Estados Unidos. Contém aplicações simuladas e kernels com finalidades diversas: ***Barnes*** (simula a interação de corpos em três dimensões, como galáxias ou partículas, através do método *N-Body*); ***Cholesky*** (similar à fatoração LU, porém com matrizes esparsas); **FFT** (implementa a transformada rápida de *Fourier*); **FMM** (similar ao *Barnes*, porém simula a interação de corpos em duas dimensões); ***Fatoração LU*** e ***Ocean*** (simulação de movimentação oceânica em larga escala).

3.3.4 DARPA HPCS Benchmarks

São *benchmarks* selecionados para os domínios da computação de alto desempenho (CAD), com o propósito de contribuir com pesquisas ligadas à DARPA (*Defense Advanced Research Projects Agency*) e outras instituições governamentais dos Estados Unidos. Tem como característica a busca por soluções economicamente viáveis para sistemas computacionais de alta produtividade, dentro do contexto do programa HPCS (*High Productivity Computing Systems*) da DARPA, guiando a construção, o desenvolvimento e a avaliação de projetos de software e hardware. Dentre os *benchmarks* ligados ao projeto DARPA HPCS, podem ser citados o HYCOM (*Hybrid Coordinate Ocean Model*), o qual é utilizado em estudos de movimentação oceânica, e o NPB, o qual é apresentado na Seção 3.3.5.

3.3.5 NAS Parallel Benchmarks (NPB)

O NPB constitui um conjunto de *benchmarks* que foi proposto pela divisão *NAS* (*Nasa Advanced Supercomputing*) da NASA (*National Aeronautics and Space Administration*), órgão governamental responsável pelo programa espacial dos Estados Unidos da América. O conjunto está dividido em cinco *kernels* e três aplicações simuladas [15], as quais realizam computação intensiva e movimentação de dados através de técnicas intrínsecas de aplicações de Dinâmica dos Fluidos Computacional (CFD) para simular aerodinâmica de veículos espaciais. Os *benchmarks* que representam as aplicações simuladas são o SP, o LU e o BT. Os *kernels* paralelos são o EP, o MG, o FT, o IS e o CG, descritos adiante. O uso de aplicações simuladas foi proposto porque os autores do projeto acreditavam que

Tabela 3.4: NAS Parallel Benchmarks

	Tamanho do problema					Tempo seg
Benchmark	Classe S	Classe W	Classe A	Classe B	Classe C	Classe A
EP	2^{24}	2^{25}	2^{28}	2^{30}	2^{32}	49,86
MG	32^3	128^3	256^3	256^3	512^3	9,61
CG	1400	7000	14000	75000	150000	6,47
FT	64^3	32×128^2	128×256^2	512×256^2	512^3	21,71
IS	2^{16}	2^{20}	2^{23}	2^{25}	2^{27}	1,71
LU	12^3	33^3	64^3	102^3	162^3	282,15
SP	12^3	36^3	64^3	102^3	162^3	272,03
BT	12^3	24^3	64^3	102^3	162^3	214,05

kernels sozinhos eram insuficientes para mensurar o desempenho das aplicações científicas reais em máquinas paralelas de arquitetura MPP (*Massive Parallel Processing*) [15]. Outra preocupação foi em projetar aplicações de modo a fazer uso genérico de recursos para não favorecer particularidades em arquiteturas paralelas [15].

Nos anos 90, quando teve início o projeto NPB, os códigos foram desenvolvidos em C e Fortran com comunicação através de biblioteca MPI. Mais tarde, foram também disponibilizadas versões seriais e multiprocessadas em Java [42]. Como uma das contribuições deste trabalho de dissertação, foram desenvolvidas versões seriais e paralelas para memória distribuída, usando C# com MPI.NET para plataformas CLI. As versões C# com MPI.NET pertencem ao projeto NPB-FOR-HPE, disponível em [5]. Cada versão do NPB possui testes de verificação de corretude e apresenta de forma simples os resultados medidos. Possui também um controle de tamanho de problema correlacionado com iterações. O tamanho do problema é determinado mediante a escolha de uma classe (S, W, A, B ou C) que é definida em tempo de compilação nas versões nativas e em tempo de execução nas versões Java e C#. Na Tabela 3.4, são apresentados os parâmetros que caracterizam o tamanho de problema com classes dos *benchmarks*, junto a tempos de execução da versão Fortran sob a classe A. O computador responsável pelos resultados tem a seguinte configuração: 1 processador *AMD Athlon 2000 MHz* com *cache* de 515 KB; *512 MB* de memória. A descrição de cada *benchmark* é apresentada em seguida.

- **EP (*Embarrassingly Parallel*):** O nome *EP* significa "embaraçosamente paralelo". Ele representa o *kernel* mais simples do NPB, não necessitando de sincronização entre processos para a computação do problema. Serve para

avaliar o desempenho de arquiteturas através de operações de ponto flutuante com pares de desvios *Gaussianos*;

- **MG (*MultiGrid*):** Trata-se de um *kernel* simplificado que explora a comunicação de longa e curta distância na solução do problema discreto de *Poisson*;
- **CG (*Conjugate Gradient*):** Um *kernel* com grade não estruturada que trabalha com matrizes esparsas e comunicação irregular;
- **FT (*Fourier Transform*):** *Kernel* que soluciona equação diferencial parcial em três dimensões utilizando Transformada Rápida de *Fourier*. Pode ser usado em teste de desempenho de comunicação por troca de mensagens, devido à sua característica de trabalhar com muitas operações coletivas [15];
- **IS (*Integer Sort*):** Trata-se de um *kernel* com ordenação *bucketsort*. Trabalha com computação de números inteiros e avaliação de comunicação;
- **LU (*Lower Upper Factorization*):** Aplicação simulada *CFD* (*Dinâmica dos fluidos computacional*). Emprega o método *SSOR* (*Symmetric Successive Over Relaxation*) para discretização baseada em diferenças finitas de equações *Navier-Stokes* [65] em duas dimensões, aplicando a técnica de divisão de matrizes em triângulos inferiores (*lower*) e superiores (*upper*);
- **BT (*Block Tridiagonal*):** Aplicação simulada *CFD* que resolve equações *Navier-Stokes* com base em diferenças finitas, através do método *ADI* (*Alternating Direction Implicit*). O problema é decomposto nas dimensões x, y e z. A matriz bloco-tridiagonal resultante é solucionada ao longo de cada dimensão;
- **SP (*Scalar Pentadiagonal*):** Aplicação simulada *CFD* similar ao BT. Resolve equações de *Navier-Stokes* através do método de fatoração *Beam-Warming*, o qual também decompõe o problema nas dimensões x, y e z. O sistema *pentadiagonal* escalar resultante é solucionado ao longo de cada dimensão [26].

Neste trabalho, o NPB é utilizado como carga de trabalho em experimentos, especificamente os *benchmarks BT, SP, LU e FT*. Justifica-se a escolha desses programas pela complexidade de seus códigos e a complexidade dos métodos

numéricos neles contidos, com mapeamento à classificação *Dwarfs*, descrita na Seção 3.4. No Capítulo 4, será explorado o processo de fatorar o NPB em componentes, tendo em vista a maximização do reuso de componentes.

3.4 Seven Dwarfs

Em meados de 2005, a indústria computacional conseguiu um marco histórico na construção de processadores, lançando no mercado os primeiros processadores de dois núcleos que asseguravam compatibilidade com os tradicionais paradigmas de desenvolvimento de software. O conceito de processadores de múltiplos núcleos, ou *multicore*, abriu mais uma perspectiva para exploração do paralelismo em aplicações, trazendo novos questionamentos e preocupações. Alguns pesquisadores têm chamado a atenção para as dificuldades com a aplicação das metodologias tradicionais de programação paralela [12]. Outros afirmam que processadores *multicore* devem estabelecer um marco para o hardware, e o grande passo para definir novos paradigmas de software paralelo [13]. Essas preocupações são fatores relevantes na geração de novas pesquisas sobre modelos de programação, de arquiteturas e de algoritmos.

Em 2004, Phil Colella descreveu modelos padrões de algoritmos usados por aplicações em vários domínios científicos e comumente paralelizados [37]. Colella identificou sete padrões numéricos que são usados com frequência por aplicações das áreas de ciência e engenharia, os quais chamou de "*Seven Dwarfs*" ou "sete anões". Um *Dwarf* pode ser visto como um método algorítmico que captura padrões de computação e comunicação de uso disseminado.

Inspirados no trabalho de Colella, um grupo multidisciplinar de pesquisadores da Universidade da Califórnia em Berkeley defendeu o uso dos *Dwarfs* como uma estratégia para projetar e avaliar modelos de programação paralela e arquiteturas [13]. Eles discutiram sobre o futuro da pesquisa na área da computação paralela, tendo em vista a necessidade de mudança do paradigma serial para o paralelo guiado pelas novas arquiteturas *multicore*. Desenvolveram um relatório técnico com sete questões críticas[3] sobre computação paralela que devem ser tratadas no século 21 (vinte e um) [13]. Entre as questões, precisavam responder quais aplicações devem ser paralelizadas e quais os *kernels* comuns entre tais aplicações. Consideraram que as aplicações deveriam ser primeiramente classificadas através dos *Dwarfs*, podendo

[3] 1-Quais são as aplicações? 2-Quais são os núcleos comuns das aplicações? 3-Quais os hardwares? 4-Como conectá-las? 5-Como descrever as aplicações e kernels? 6-Como programar o hardware? 7-Como medir com sucesso?

uma aplicação estar relacionada com um ou mais *Dwarfs*. Então, seria possível identificar e definir *kernels* comuns entre elas. Como contribuição, propuseram mais seis (6) *Dwarfs*, estendendo a classificação para treze (13). Os *Dwarfs* adicionais diferenciam-se dos métodos numéricos propostos por Colella, pois incluem padrões encontrados em *benchmarks* disseminados e em alguns métodos científicos para construção de algoritmos. A sequência é descrita a seguir:

- **01.** Álgebra linear Densa;
- **02.** Álgebra linear esparsa;
- **03.** Métodos espectrais;
- **04.** Métodos N-body;
- **05.** Grades estruturadas;
- **06.** Grades não estruturadas;
- **07.** Monte Carlo;
- **08.** Lógica combinatorial;
- **09.** Percurso em grafos;
- **10.** Programação dinâmica;
- **11.** Métodos de ramificação e poda;
- **12.** Modelos em grafo;
- **13.** Simulação de máquina de estado finito.

Neste trabalho de dissertação é feito o uso dessa classificação, especialmente porque contém um vínculo entre o *NAS Parallel Benchmarks* e os níveis de problemas definidos por *Dwarfs*. O relacionamento entre os *Dwarfs* e os *benchmarks* que serão aplicados neste trabalho pode ser visto na Tabela 3.5. O BT e LU encontram-se na classificação um (1). O FT encontra-se na classificação três (3). O SPna classificação cinco (5).

[4]Biblioteca de álgebra linear para computação paralela.

Tabela 3.5: Relacionamento entre *Dwarfs* e *benchmarks*. Fonte [13].

Dwarf	Descrição	Padrão de comunicação	*NAS P. Benchmarks*
01. Álgebra linear Densa	Os dados são matrizes densas ou vetores. Geralmente, essas aplicações fazem o acesso à memória de forma consecutiva com dados adjacentes para as linhas e colunas(unit-stride memory accesses)	Faz uso pesado de ScaLAPACK[4] para paralelismo em álgebra linear densa. É típico em uma vasta classe de algoritmos numéricos.	Block Tridiagonal Matrix, Lower Upper Symmetric
02. Álgebra linear esparsa	Os conjuntos de dados incluem muitos valores nulos. Os Dados são normalmente armazenados em matrizes compactadas para reduzir armazenamento e requisitos de largura de banda para acessar todos os valores diferentes de zero.	Usa o método "Linhas esparsas comprimidas"para implementar a fatoração esparsa LU.	Conjugate Gradient/ Vector computers with gather/scatter
03. Métodos espectrais	Normalmente métodos espectrais utilizam múltiplos estágios na forma borboleta com a combinação de operações de adição e multiplicação, e um padrão específico de troca de dados por comunicação all-to-all para alguns estágios e estritamente locais para outros.	Requer comunicação all-to-all para implementar uma transposição 3D, necessitando comunicação entre todos os links.	Fourier Transform
04. Métodos N-body	Depende de interações entre muitos pontos discretos. Cada ponto depende de todos os outros, levando a uma operação O(N2). Métodos de partículas combinam potenciais de pontos para reduzir a complexidade para O(nlogn) ou O(n).	Padrão de comunicação, malhas de partículas.	Sem benchmark
05. Grades estruturadas	Representado por uma grade estruturada, onde os pontos são conceitualmente atualizados juntos. Essas atualizações podem ser diretas ou entre duas versões da grade. A grade pode ser subdivida dinamicamente por áreas de interesses nas mais finas granularidades.	Comunicação padrão para soluções de equações diferenciais parciais (EDP) usando blocos 3D estruturados em grades.	Multi-Grid, Scalar Pentadiagonal
06. Grades não estruturadas	Uma grade não estruturada onde os dados locais são selecionados normalmente por características marcantes da aplicação. Dados locais e conectividade com dados de vizinhos devem ser explícitos. Os pontos são conceitualmente atualizados juntos. A atualização de um ponto requer primeiramente determinar uma lista de pontos vizinhos e então carregar valores desses pontos.		Unstructured Adaptive/ Vector computers with gather/scatter
07. Monte Carlo	Cálculos possuem dependências de resultados estatísticos de repetições aleatórias. Considerado embaraçosamente paralelo.	Comunicação normalmente não é dominante em métodos Monte Carlo.	Embarrassingly Parallel

3.5 Uma Abordagem Estatística para Avaliação de Desempenho

As tradicionais metodologias para avaliações de desempenho de sistemas computacionais muitas vezes consistem na comparação direta de resultados de medições, sem observações mais detalhadas acerca dos fatores que envolvem os sistemas. Nesta seção, a metodologia de avaliação de desempenho busca explorar maiores detalhes dos fatores envolvidos em uma medição, recorrendo aos princípios de *Experimentos Fatoriais* e comparações de alternativas através de *Intervalos de Confiança*. Os modelos estatísticos e métodos experimentais adotados são baseados na abordagem utilizada em [47], bastante disseminada para avaliação de desempenho de sistemas computacionais.

3.5.1 Terminologia Utilizada

O estudo de desempenho é composto por alguns termos essenciais que são referenciados ao longo das próximas seções. Esses termos são predominantemente estatísticos. Suas características principais são descritas em seguida.

- **Variável aleatória**: uma função mensurável de um espaço de probabilidade que pode gerar valores diferentes a cada medida [10];
- **Média ou Valor Esperado**: a soma de cada probabilidade $p(x)$ multiplicada pelo valor x. $\mu = E(x) = \sum_{i=1}^{n} p(x_i)x_i$ [47];
- **Variância**: sendo $E(x)$ o valor esperado da ocorrência da variável aleatória x, a variância é o valor esperado que envolve o quadrado do desvio de x da sua média: $Var(x) = E[(x-\mu)^2] = \sum_{i=1}^{n} p(x_i)(x_i-\mu)^2$. A variância é representada pelo símbolo σ^2;
- **Desvio padrão**: o **desvio padrão** é a raíz quadrada da variância. Por esse motivo, é conhecido pelo símbolo σ;
- **Distribuição Normal**: É conhecida como Distribuição *Gaussiana*, importante em distribuições contínuas da probabilidade. Estrutura-se por uma função densidade de probabilidade (*PDF-Probability density function*), dada por:

  Equação 3.1 $p(x) = \frac{1}{\sigma \times \sqrt{2\pi}} \times e^{-(x-\mu)^2/2\sigma^2}$

- **Variável de resposta**: são os resultados adquiridos nos experimentos;
- **Fatores**: para o estudo de desempenho de sistemas, são as características que influenciam a variável de resposta, sendo que um fator é composto por níveis. Especificamente para sistemas computacionais, pode-se mencionar como fatores: tamanho de memória, tipo de CPU, tipo da carga de trabalho, topologia de redes;
- **Níveis**: os níveis são os valores que o fator pode assumir. Por exemplo, o fator tamanho de memória pode ser expresso com os níveis 1024MB, 2048MB, 4096MB;
- **Replicação**: replicações são as repetições de um experimento, efetuadas com várias medições com uma mesma configuração de sistema, formando uma amostra de valores. Normalmente, a média, a moda ou a mediana da amostra representa o experimento, e a amostra representa as replicações;
- **Efeitos**: indica numericamente o comportamento do fator em um experimento fatorial. O efeito pode definir a característica de cada nível do fator, ou pode definir a intensidade de interações entre níveis de vários fatores;
- **Interação**: a interação ocorre quando os efeitos dos níveis de um fator sofrem variações em relação aos efeitos dos níveis de outros fatores na mudança de um nível para o outro. Por exemplo, seja $S[S_1, S_2]$ um fator com níveis S_1 e S_2, respectivamente representando os sistemas 1 e 2; $P[P_1, P_4]$ outro fator com níveis P_1 e P_4, respectivamente representando a quantidade de processadores (1 e 4) utilizados por S_1 e S_4 nos experimentos; I/O a quantidade de entrada e saída por segundo que os sistemas S_1 e S_2 utilizam com P_1 ou P_4. As interações são observadas nos gráficos da Figura 3.1. No primeiro gráfico, os valores I/O dos níveis S_1 e S_2 permanecem constantes quando os níveis de processadores mudam. Entretanto, no segundo gráfico, os níveis S_1 e S_2 sofrem interferência à medida em que os níveis de processadores são mudados, o que gera o parâmetro de variação, denominado interação.

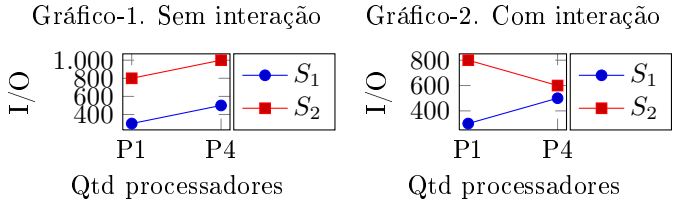


Figura 3.1: Exemplo de interação.

3.5.2 Comparação entre dois Sistemas

A comparação entre sistemas pode ser vista sob dois aspectos: o primeiro consiste em comparar especificamente duas alternativas, observando em experimentos a ocorrência de potencial diferença de desempenho; o segundo tem caráter mais complexo, caracterizado pela análise conjunta de vários fatores que envolvem os sistemas. Nesta seção, com o auxilio de fundamentos estatísticos, descrevemos o primeiro aspecto, comparando duas alternativas.

Comparações entre duas alternativas têm sido feitas utilizando intervalos de confiança (IC). Um IC é constituído pela limitação inferior e superior de um parâmetro estatístico. No caso de comparações entre sistemas, normalmente o parâmetro estatístico é uma média dos valores que representam o desempenho dos sistemas. Porém, em estatística, existe a média da população e a média da amostra. A média da amostra é obtida a partir de um subconjunto da população, enquanto a média da população é mais realista, envolvendo todos os valores que descrevem o fenômeno. Mas a média da população nem sempre é conhecida. Por isso, usa-se o intervalo de confiança para estimar a média da população. Nas medições de sistemas, não são conhecidas todas as possíveis variações de resposta que podem ser obtidas nos experimentos, o que justifica o uso do IC para estimar a média populacional das medições a partir de amostras.

Ao comparar alternativas, o IC auxilia na decisão sobre superioridade de um sistema em relação ao outro. Para isso, pode ser usado algum critério que defina essa superioridade. Por exemplo, um dos critérios consiste na medição e subtração de tempo de desempenho das alternativas, onde um valor zero nessa subtração significa que os desempenhos foram iguais. Ao ser feita a subtração (*TempoSistema1* - *TempoSistema2*) de várias medições, pareando-as uma a uma, uma amostra para estudo é gerada. O valor zero não pode fazer parte do IC calculado a partir dessa amostra. Caso isso ocorra, é irrelevante a diferença de desempenho entre os sistemas. Proporcionalmente, quanto mais próximo de zero estiver o IC, menor a superioridade

	A	B	A-B
	50	48,5	1,5
	48,7	49,3	-0,6
	48,2	49	-0,8
média			0,03
Desvio Padrão			1,27
IC da média			[-2,11 ... 2,18]

Tabela 3.6: Exemplo amostras em par

de um sistema em relação ao outro. Tomemos como exemplo o seguinte cenário: A e B são dois sistemas; cada um usa de um método diferenciado para realizar um determinado cálculo; três medições são feitas em cada sistema até terminar o cálculo, com valores em segundos; as métricas desse experimento são classificadas como BM (Baixo Melhor). Os dados desse exemplo são apresentados na Tabela 3.6.

A subtração nos dá a superioridade de desempenho em termos de tempo entre os sistemas A e B. Como a métrica é BM , o resultado positivo da subtração favorece B. Assim, pode-se analisar a Tabela 3.6 como uma tentativa de verificar a superioridade de desempenho de B em relação a A. Na primeira medição, o sistema B obteve $1,5$ segundos de vantagem em relação a A. Porém, nas demais medições, o desempenho foi pior, com desvantagem de -1,4 segundos para B. Dessa forma, não é possível dizer que o sistema B possui desempenho superior a A, pois o intervalo de confiança contém o valor zero $[-2,11, 2,18]$, fato que desqualifica a superioridade de B em relação a A. O IC da média ($[-2,11, 2,18]$) é obtido a partir do cálculo $(0,03 \pm 2,92 \times 1,27/\sqrt{3})$. O método para calcular o IC é descrito na Seção seguinte.

Intervalo de confiança da média

O intervalo de confiança é uma estimativa para um valor probabilístico. Para comparação entre sistemas, o valor probabilístico é a média de uma amostra de dados que representa o desempenho dos sistemas, onde é esperado que essa média esteja nos limites do IC, com algum nível de confiança. Definem-se os limites inferior e superior de um IC respectivamente como $[l_1, l_2]$, formalmente obedecendo a $[l_1 \leq \mu \leq l_2]$ com um determinado α, onde α é o nível de significância e μ a média. Pelo α, é estabelecido o coeficiente de confiança, dado por $(1 - \alpha)$. Pelo coeficiente de confiança, é estabelecido o nível de confiança, dado por $100(1 - \alpha)$. Níveis de confiança geralmente são representados por porcentagens como 90% e 95%, com respectivos níveis de significância de $0,1$ e $0,05$.

Para encontrar o intervalo com um determinado nível de confiança, recorre-se ao conceito de teorema do limite central [47, 51]. O teorema diz que "se existem várias amostras de tamanho n independentes, com mesma distribuição de probabilidade, pertencentes a uma mesma população com média μ e desvio padrão σ, então as médias amostrais convergem para a distribuição normal com média μ e desvio padrão $\sigma/\sqrt{n}$ à medida em que n aumenta", tal que $\bar{x} \sim N(\mu, \sigma/\sqrt{n})$, onde $\bar{x}$ representa as médias amostrais. Dessa forma, a média das médias amostrais tende à média da população com distribuição normal. A Figura 3.2 ilustra vários intervalos atingindo a média μ de uma população. Para um IC com 95% de confiança, a cada 100 intervalos na Figura 3.2, 5 não atingirão μ.

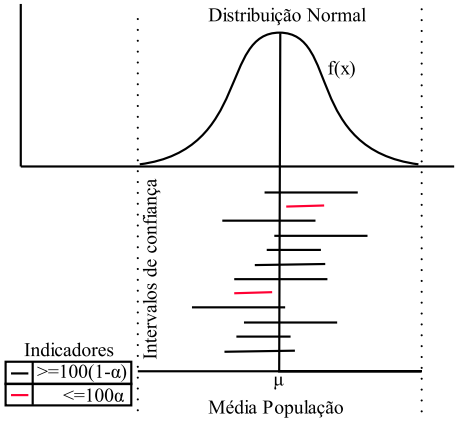


Figura 3.2: Característica de Intervalo de confiança [47].

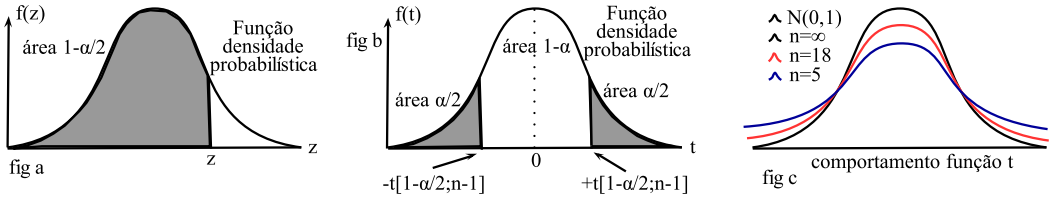


Figura 3.3: Localização z e t.

Os limites l_1 *e* l_2 do IC com $100(1-\alpha)\%$ de nível de confiança para uma média populacional são dados por $[\bar{x} \mp z_p \times s/\sqrt{n}]$, onde $\bar{x}$, s e n são respectivamente a média, o desvio padrão e o número de elementos de uma amostra da população. A variante z_p pode ser obtida pela identificação do limite superior da integral que contém a função densidade da distribuição normal, *Equação 3.1*, com média zero e desvio padrão 1 (*N(0,1)*). Por exemplo, colocando-se zero (0) e um (1) para média

e desvio padrão na Equação 3.1, temos: $f(x) = \frac{1}{\sqrt{2\pi}} \times e^{-(x)^2/2}$. O valor z_p é o limite superior z em: $\int_{-\infty}^{z} f(x)dx = 1 - \alpha/2$. Esta integral calcula a área limitada pelo eixo de domínio e sua função, representados na Figura 3.3a.

Exemplo 3.1 *Intervalo de confiança com distribuição z: para simplificar o conceito de IC, tomemos como exemplo os indicadores tamanho da amostra ($n = 38$), média ($\bar{x} = 4,95$), desvio padrão ($s = 0,91$). O estudo do intervalo de confiança da média da população para esse exemplo é definido em seguida.*

- ***Intervalo com 90% de confiança:***
  1: $1 - \alpha/2 = 1 - 0,1/2 = 0,95 = \int_{-\infty}^{z_p} \frac{1}{\sqrt{2\pi}} \times e^{-(x)^2/2}dx = \int_{-\infty}^{1,645} \frac{1}{\sqrt{2\pi}} \times e^{-(x)^2/2}dx$
  2: $z_p = z_{1-\alpha/2} = z_{1-0,1/2} = z_{0,95} = 1,645$
  3: $[\bar{x} - z_p \times s/\sqrt{n}, \quad \bar{x} + z_p \times s/\sqrt{n}]$
  4: $[4,95 \mp 1,645 \times 0,91/\sqrt{38}] = [4,707, 5,192]$

- ***Intervalo com 95% de confiança:***
  1: $1 - \alpha/2 = 1 - 0,05/2 = 0,975 = \int_{-\infty}^{z_p} \frac{1}{\sqrt{2\pi}} \times e^{-(x)^2/2}dx = \int_{-\infty}^{1,960} \frac{1}{\sqrt{2\pi}} \times e^{-(x)^2/2}dx$
  2: $z_p = z_{1-\alpha/2} = z_{1-0,05/2} = z_{0,975} = 1,960$
  3: $[\bar{x} - z_p \times s/\sqrt{n}, \quad \bar{x} + z_p \times s/\sqrt{n}]$
  4: $[4,95 \mp 1,960 \times 0,91/\sqrt{38}] = [4,660, 5,239]$

No exemplo anterior, para o intervalo de confiança a 90%, note que a integral definida tem como limite superior o valor $z_p = z_{0,95} = 1,645$. Analogamente, para o intervalo de confiança a 95%, a integral definida tem como limite superior o valor $z_p = z_{0,975} = 1,960$. Esse valores são obtidos a partir da Tabela de probabilidades B.1. Essa integral calcula a área entre $-\infty$ e z_p, limitada pelo eixo z e a função densidade, ilustrada na Figura 3.3a. As áreas para 90% e 95% correspondem respectivamente a $0,95$ e $0,975$ para a distribuição z.

Os cálculos do Exemplo 3.1 são indicados para amostras com $n > 30$, segundo define a literatura [54, 58]. Em pequenas amostras com distribuição normal, o analista de desempenho deve utilizar a distribuição-t [8,64]. A distribuição-t permite definir um IC com os limites $[\bar{x} \mp t[1-\frac{\alpha}{2}; n-1] \times \frac{s}{\sqrt{n}}]$, onde $\bar{x}$, s e n são respectivamente média, desvio padrão e tamanha da amostra. Em $t[1-\frac{\alpha}{2}; n-1]$, $(1-\frac{\alpha}{2})$ é o coeficiente de confiança, e $n-1$ representa os graus de liberdade, que na prática normalmente é utilizado com valor inferior ou igual a 30. O valor de $t[1-\frac{\alpha}{2}; n-1]$ é tal que a média da

população tem probabilidade $\frac{\alpha}{2}$ de ser maior que $+t[1-\frac{\alpha}{2}; n-1]$, probabilidade $\frac{\alpha}{2}$ de ser menor que $-t[1-\frac{\alpha}{2}; n-1]$ e probabilidade $1-\alpha$ de estar entre $\pm t[1-\frac{\alpha}{2}; n-1]$ [47]. O gráfico com a função de densidade da distribuição-t é ilustrado na Figura 3.3b. Um aspecto importante em relação a esse gráfico é que a curva de sua função tem tendência à curva da distribuição normal padrão (*N(0,1)*). Esse fenômeno ocorre à medida em que n aumenta, equiparando-se quando $n = \infty$ [8], como é ilustrado na Figura 3.3c.

Valores z e t são constantemente usados em estatística. Dessa forma, uma tabela com os principais valores z e t é apresentada no Anexo (B.1 e B.2) respectivamente. Observando a Tabela B.2 para encontrar t[5], desenvolveremos o exemplo seguinte para intervalo de confiança.

Exemplo 3.2 *Intervalo de confiança com distribuição t: definindo A como conjunto de valores da amostra, $\bar{x}$ como média da amostra, s como desvio padrão da amostra, temos:* $A = \{270,137;\ 258,648;\ 258,823;\ 269,921;\ 269,748;\ 259,107;\ 259,574;\ 258,499;\ 258,962;\ 259,864;\ 259,038\}$*;* $n = 11$*;* $\bar{x} = 262,029$*;* $s = 5,092$.

- ***Intervalo com 90% de confiança:***
  1: $t[1-\alpha/2; n-1] = t[1-0,1/2; 11-1] = t[0,95; 10] = 1,812$
  2: $[(\bar{x} - t[1-\alpha/2; n-1] \times s/\sqrt{n},\quad \bar{x} + t[1-\alpha/2; n-1] \times s/\sqrt{n})]$
  3: $[262,029 \pm 1,812 \times 5,092/\sqrt{11}] = [259,24, 264,81]$

- ***Intervalo com 95% de confiança:***
  1: $t[1-\alpha/2; n-1] = t[1-0,05/2; 11-1] = t[0,975; 10] = 2,228$
  2: $[(\bar{x} - t[1-\alpha/2; n-1] \times s/\sqrt{n},\quad \bar{x} + t[1-\alpha/2; n-1] \times s/\sqrt{n})]$
  3: $[262,029 \pm 2,228 \times 5,092/\sqrt{11}] = [258,60, 265,45]$

No exemplo anterior, para os intervalos de confiança a 90% e 95%, os valores obtidos a partir da Tabela de probabilidade B.2 são $t[0,95; 10] = 1,812$ e $t[0,975; 10] = 2,228$, respectivamente.

Seja $f(t)$ a função densidade da distribuição-t, $\int_{-\infty}^{-1,812} f(t)dt$ e $\int_{-\infty}^{-2,228} f(t)dt$ calculam a área extrema limitada por $f(t)$ e o eixo t, conforme ilustrado na Figura 3.3b.

[5] Coeficientes de confiança 0,95 para 90% e 0,975 para 95%, com graus de liberdade 10 (t[0,95;10], t[0,975;10])

Amostras pareadas e não pareadas

As amostras de sistemas a serem comparados são passivas de estarem em par ou não em par. Em cada situação deve-se abordar uma forma diferente para gerar o desvio padrão e desenvolver o intervalo de confiança. Na primeira situação, retratando amostras em par, temos uma quantidade idêntica de medições entre os dois sistemas. Para comparação de sistemas, gera-se uma amostra de diferenças e calcula-se o seu desvio padrão. Por exemplo, sejam $X[x_1, x_2, x_3]$ e $Y[y_1, y_2, y_3]$ respectivamente os sistemas X e Y com medições (x_1,x_2,x_3) e (y_1,y_2,y_3). A amostra resultante para calcular o desvio padrão é: $x_1 - y_1, x_2 - y_2, x_3 - y_3$. No caso de amostras não pareadas, como por exemplo $X[x_1, x_2, x_3, x_4]$ e $Y[y_1, y_2, y_3]$, onde há quantidades de medições diferentes e/ou não relacionadas uma-a-uma, deve-se calcular o desvio padrão e o intervalo de confiança segundo a seguinte abordagem: sejam X e Y conjuntos representantes de medições de dois sistemas, respectivamente com tamanhos de amostras n_x e n_y com variâncias diferentes, o intervalo de confiança é calculado pelos procedimentos seguintes.

- Calcula-se a média de cada amostra: μ_x e μ_y;
- Calcula-se o desvio padrão de X: S_x;
- Calcula-se o desvio padrão de Y: S_y;
- Calcula-se o desvio padrão das diferenças: $S_{xy} = \sqrt{\frac{S_x^2}{n_x} + \frac{S_y^2}{n_y}}$;
- Calcula-se o intervalo de confiança dado por: $(\mu_x - \mu_y) \mp t[1 - \alpha/2; gl] \times S_{xy}$, onde gl é o grau de liberdade calculado no item seguinte;
- Grau de liberdade gl: $gl = \frac{(S_x^2/n_x + S_y^2/n_y)^2}{\frac{(\frac{S_x^2}{n_x})^2}{n_x - 1} + \frac{(\frac{S_y^2}{n_y})^2}{n_y - 1}} - 2$ (Eq. de *Welch Satterthwaite* [55]);
- Valor t : para amostras não pareadas e variâncias diferentes o valor t é dado por $t = \frac{\mu_x - \mu_y}{\sqrt{(S_x^2/n_x) + (S_y^2/n_y)}}$. *Valor-t* é usado na fundamentação do *Teste-t*, discutido em seguida.

Teste-t: o *Teste-t* é um teste de hipótese que pode ser usado para checar se uma amostra é significativamente diferente de outra, calculando a probabilidade das médias serem significativamente diferentes, sugerindo que ambas pertençam a populações distintas. *Teste-t* pode ser aplicado em amostras não pareadas. Para isso, é calculado o *valor-t*, pelas fórmulas anteriormente apresentadas, supondo

que t satisfaça à distribuição-t. Pelo *valor-t* é possível que a função densidade da distribuição-t estabelecer a probabilidade dos sistemas serem diferentes, calculando-se as áreas extremas da distribuição e comparando-as com o nível de significância. Caso essas áreas superem o nível de significância, a hipótese de sistemas diferentes é rejeitada. Isso é ilustrado na Figura 3.4, com as seguintes hipóteses: $H_0 : \mu_x = \mu_y$, para hipótese nula, ou sistemas iguais; e $H_1 : \mu_x \neq \mu_y$, para hipótese alternativa, ou sistemas diferentes. Essas hipóteses definem uma estrutura bicaudal, uma vez que *Teste-t* pode definir um ou outro sistema como melhor, dependendo do t negativo ou positivo correspondente na distribuição t.

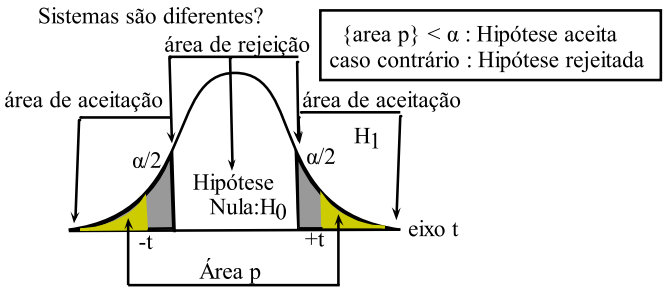


Figura 3.4: Exemplo Teste-t.

3.5.3 Projeto de Experimentos Fatoriais

Complementar à comparação entre sistemas pelo *IC*, os projetos de experimentos fatoriais viabilizam a análise de variância de dados experimentais. O projeto consiste em definir os fatores utilizados, níveis de fatores, número de experimentos e quantidade de replicações. Projetos de experimentos fatoriais podem envolver diversos fatores com ou sem replicações. Quanto maior o número de fatores, maior a quantidade de efeitos, potencialmente aumentando a complexidade do modelo estatístico. Por exemplo, caso tenhamos dois fatores A e B, teremos: um efeito para A, outro para B e outro para interação AB. A quantidade de efeitos de fatores representa todas as possibilidade de combinações de fatores, sendo calculada por: $2^f - 1$, para f sendo a quantidade de fatores. Dessa forma, um projeto que tenha cinco fatores teria trinta e uma tabelas de efeitos $(2^5 - 1)$, o que pode dificultar a análise. Nesta seção, vamos nos atentar ao modelo de experimento fatorial para dois fatores com replicações, facilitando a compreensão do modelo. Posteriormente, o modelo dois fatores é expandido para três fatores.

Modelo de Experimento Fatorial com dois fatores e r replicações

O modelo tem como característica a utilização dos fatores representados pelos símbolos A e B. O fator A tem a quantidade de a níveis, e o fator B tem b níveis, resultando em $a \times b$ experimentos. Cada experimento é representado pela média aritmética de um total de r medições, denominadas replicações. No modelo adotado nesta seção, a média do experimento é representada por $\bar{y}_{ij.}$, onde os indexadores i e j fazem referência aos níveis dos fatores B e A, respectivamente. O ponto (.) refere-se às replicações que são indexadas por k replicações, para $0 < k \leq r$, tal que $\bar{y}_{ij.} = \sum_{k=1}^{r} y_{ijk}/r$, onde y_{ijk} é cada valor medido em sequências de r medições. A diferença entre o valor esperado das replicações e o valor medido é conhecida como erro $e_{ijk} = y_{ijk} - \bar{y}_{ij.}$. Os efeitos de cada nível do modelo é representado por A_j e B_i, respectivamente para os fatores A e B. Formalmente, o Modelo 3.1 especifica dois fatores e r replicações, sendo exemplificado na Tabela 3.7.

Modelo 3.1 $y_{ijk} = \mu + A_j + B_i + AB_{ij} + e_{ijk}$

$\bar{y}_{ij.} = \frac{y_{ij1}+y_{ij2}+y_{ij2}+y_{ij4}}{r}$

		A			
	níveis	j=1	j=2	j=3	efeito
B	i=1	$\bar{y}_{ij.}$	$\bar{y}_{ij.}$	$\bar{y}_{ij.}$	B_1
	i=2	$\bar{y}_{ij.}$	$\bar{y}_{ij.}$	$\bar{y}_{ij.}$	B_2
	efeito	A_1	A_2	A_3	

		replicações r=4			
		k-ésima medição y_{ijk}			
1	$\bar{y}_{11.}$	y_{111}	y_{112}	y_{113}	y_{114}
. .	. .	. .	. .	. .	. .
6	$\bar{y}_{23.}$	y_{231}	y_{232}	y_{233}	y_{234}

	k-ésimo erro (e)			
1	$y_{111} - \bar{y}_{11.}$	$y_{112} - \bar{y}_{11.}$	$y_{113} - \bar{y}_{11.}$	$y_{114} - \bar{y}_{11.}$
. .	. .	. .	. .	. .
6	$y_{231} - \bar{y}_{23.}$	$y_{232} - \bar{y}_{23.}$	$y_{233} - \bar{y}_{23.}$	$y_{234} - \bar{y}_{23.}$

Tabela 3.7: Exemplificação do modelo.

► **Definições**

- A, B: símbolos representativos de fatores;
- a, b, r: a e b são as quantidades de níveis dos fatores A e B respectivamente, e r, a quantidade de replicações para formar um experimento;
- i, j, k: indexação que representa ab níveis e r replicações;
- y_{ijk}: resposta observada na k-*ésima* medição, para formar o experimento

Níveis	$LU_1(S)$	$LU_2(W)$	$LU_3(A)$
M_1	2,714	472,212	3144,134
	2,672	473,408	3124,952
	2,645	471,004	3121,226
	2,663	473,727	3128,214
M_2	1,855	286,025	1961,698
	1,812	273,498	1785,908
	1,789	310,914	1959,884
	1,769	383,781	1883,31

Tabela 3.8: Dados medidos com 4 replicações

envolvendo o fator A no nível j e o fator B no nível i;

• μ: a média de todas as medições;

• A_j: indicador de efeito provocado pelo fator A no nível j: $A_j = \bar{y}_{.j.} - \mu$, tal que $\bar{y}_{.j.}$ é a média da coluna j;

• B_i: indicador de efeito provocado pelo fator B no nível i: $B_i = \bar{y}_{i..} - \mu$, tal que $\bar{y}_{i..}$ é a média da linha i;

• $\bar{y}_{ij.}$: média de r replicações;

• AB_{ij}: indicador de efeito provocado pela interação do fator A no nível j com o fator B no nível i: $AB_{ij} = \bar{y}_{ij.} - (\mu + A_j + B_i)$. Ou seja, é o Modelo 3.1 sem o erro;

• e_{ijk}: erro experimental para a k-*ésima* medição: $e_{ijk} = y_{ijk} - \bar{y}_{ij.}$. Erro é a diferença entre o valor medido e o valor esperado.

Estudo de Caso 3.1 *Aplicamos o modelo dois fatores com r replicações em dois computadores operando com três tamanhos de problema. Resume-se os itens de estudo em: dois computadores, simbolizados M_1 e M_2; um jogo de teste codificado em C# (benchmark LU); o jogo de teste é executado com três tamanhos de problema, simbolizados S, W, A, representando matrizes de tamanho (10.140, 179.685, 1.352.000) respectivamente; a quantidade de replicações é quatro (4). O cenário de teste possui o fator B, representado o computador, com níveis M_1 e M_2, e o fator A, representando o tamanho da carga de trabalho, com níveis $LU_1(S)$, $LU_2(W)$, $LU_3(A)$. Os fatores A e B têm como base o Modelo 3.1.*

A Tabela 3.8 possui os dados medidos para o Estudo de Caso 3.1. A partir dessas medições, faz-se necessária a aplicação da média $\bar{y}_{ij.}$ para representar os experimentos relacionados com os níveis (M_1, M_2), e os níveis ($LU_1(S)$, $LU_2(W)$

Níveis	$LU_1(S)$	$LU_2(W)$	$LU_3(A)$
M_1	2,6735	472,58775	3129,6315
M_2	1,80625	313,5545	1897,7

Tabela 3.9: Média das replicações

e $LU_2(A)$). A média está disponível na Tabela 3.9. Porém, quando o tempo de processamento é mensurado, tem-se a medida dada de forma multiplicativa, diferente do Modelo 3.1, que é uma equação aditiva ($y_{ijk} = \mu + A_j + B_i + AB_{ij} + e_{ijk}$). Por exemplo, se existem I instruções para serem executadas em um algoritmo e o computador consegue realizar P instruções por segundos, temos o tempo dado de forma multiplicativa: $y_{ij} = \frac{1}{P_i} \times I_j$. Para melhorar a precisão dos resultados vindos de uma forma multiplicativa, ao invés de usar o modelo aditivo diretamente, definem-se conversões de multiplicativo para aditivo nas médias dos experimentos, o que pode ser feito por logaritmos: $y_{ij} = \frac{1}{P_i} \times I_j$, convertido em $log(y_{ij}) = log(\frac{1}{P_i}) + log(I_j)$, onde $log(y_{ij}) = y'_{ij}$. Utilizando-se do Modelo 3.1, teríamos um novo y' representando a forma multiplicativa pelo modelo aditivo: $y'_{ijk} = \mu + A_j + B_i + AB_{ij} + e_{ijk}$. Embora os dados possam ser utilizados sem a aplicação de *logaritmo*, tem-se obtido melhores resultados na análise de desempenho de sistemas (medindo tempo de processamento) quando é aplicado o logaritmo nos dados medidos [47]. Dessa forma, nos experimentos deste trabalho são aplicados *logaritmos* nas médias obtidas a partir das medições. A transformação *logarítmica* é feita na Tabela 3.10 para o Estudo de Caso 3.1, permitindo maior precisão para calcular o efeito de cada nível i e j.

Os cálculos dos efeitos são obtidos através da subtração de médias. Por exemplo, para calcular o efeito B_i, subtrai-se da média de cada linha i a média de todas as medições, representada por ($\mu = 2,104$), como pode ser visto na Tabela (3.11). De forma semelhante, para calcular o efeito A_j, subtrai-se da média de cada coluna j a média μ. Tal procedimento resulta em: $B_1 = 2,199 - 2,104 = 0,0942$; $B_2 = 2,010 - 2,104 = -0,0942$; $A_1 = 0,683 - 2,104 = -1,762$; $A_2 = 2,585 - 2,104 = 0,480$; $A_3 = 3,386 - 2,104 = 1,282$.

Pode-se perceber que dos níveis j apenas o LU_1 está abaixo da média μ (2,104), com diferença de (-1,762), justificável pela utilização de um tamanho de problema menor para o *benchmark*. Dos níveis i, o M_2 ficou abaixo da média em (-0,0942), caracterizando uma máquina com maior capacidade de processamento. Uma particularidade em experimentos fatoriais é que a soma dos efeitos de B_i resulta

Níveis	$LU_1(S)$	$LU_2(W)$	$LU_3(A)$
M_1	0,427	2,674	3,495
M_2	0,256	2,496	3,278

Tabela 3.10: Logaritmo das médias

Níveis	$LU_1(S)$	$LU_2(W)$	$LU_3(A)$	Soma	Média	B_i
M_1	0,427	2,674	3,495	6,597	2,199	0,0942
M_2	0,256	2,496	3,278	6,031	2,010	-0,0942
Soma	0,683	5,170	6,773			
Média	0,341	2,585	3,386		2,104	
A_j	-1,762	0,480	1,282			

Tabela 3.11: Identificação de efeitos

em zero, bem como a soma dos efeitos A_j também resulta em zero. Após efeitos principais A_j e B_i definidos, pode-se calcular os efeitos de interação entre os níveis i e j dos fatores. Para isso, observe o Modelo 3.1, estabelecido anteriormente. Nas definições desse modelo, a interação entre fatores e níveis pode ser calculada pela seguinte expressão: $AB_{ij} = \bar{y}_{ij.} - (\mu + A_j + B_i)$. Cada efeito de interação pode ser visto na Tabela 3.12.

Para M_1, a interação que obteve maior desempenho foi $M_1 \times LU_1$, -0,009 abaixo da média de interações, e a pior $M_1 \times LU_3$, 0,014 acima da média. Para M_2, a interação que obteve maior desempenho foi $M_2 \times LU_3$, -0,014 abaixo da média de interações, e a pior $M_2 \times LU_1$, 0,009 acima dessa média.

Verificação de dispersão e normalidade dos dados: verificar o comportamento e distribuição dos dados experimentais tem um papel importante no estudos de desempenho, especialmente na observação de medições que não seguem um padrão normal de resposta. Para isso, é comum a utilização de gráficos de probabilidade, como *QQ (Quantil Quantil)*, feito a partir dos erros experimentais ($e_ijk = y_{ijk} - \bar{y}_{ij.}$) e a função inversa de distribuição acumulada. Por exemplo, seja

Níveis	$LU_1(S)$	$LU_2(W)$	$LU_3(A)$
M_1	-0,009	-0,005	0,014
M_2	0,009	0,005	-0,014

Tabela 3.12: Interação entre níveis de fatores

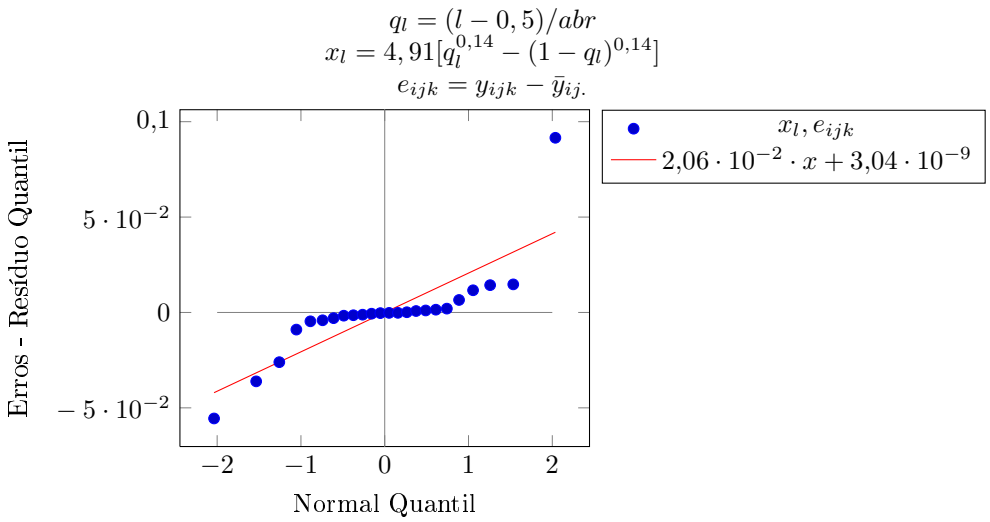


Figura 3.5: Gráfico de probabilidade Normal Quantil Quantil.

$q_l = F(x_l)$ a função de distribuição acumulada, para $q_l = \frac{l-0,5}{n}$ e $1 \leq l \leq n$, onde $n = abr$ para o modelo com dois fatores, então a inversa é dada por $x_l = F^{-1}(q_l)$. Uma aproximação utilizada para F^{-1} é dada na literatura por $x_l = 4,91[q_l^{0,14} - (1-q_l)^{0,14}]$. Dessa forma, pode-se desenvolver o gráfico e verificar se os dados estão distribuídos corretamente em torno de zero, com uma aproximação à distribuição normal padrão N(0,1), o que é uma característica importante para as considerações feitas a partir dos resultados do experimento fatorial. Isso porque os efeitos estão contidos no intervalo $[-\mu, +\mu]$, com zero sendo um efeito nulo. O gráfico QQ para o Estudo de Caso 3.1 é apresentado na Figura 3.5. Cada ponto representa uma medição. Note que existe um padrão de distribuição, onde os dados seguem a reta, com ocorrência de dados mais dispersos na calda, o que é considerado comum para esse tipo de gráfico. Ainda, podemos perceber a simetria dos dados em torno de zero. Não é a simetria ideal, especialmente porque a amostra do estudo de caso é razoavelmente pequena, mas aproxima-se de uma amostra satisfatória para o modelo, uma vez que não apresenta dispersão.

Expansão do Modelo para vários fatores

A expansão do Modelo 3.1 é uma combinação de fatores, semelhante ao relacionamento de dados em matrizes multidimensionais, com cada fator sendo uma dimensão, e os níveis sendo os índices de cada dimensão. Em um modelo que tenha os fatores $A[a_1, a_2, a_3]$, $B[b_1, b_2]$ e $C[c_1, c_2, c_3, c_4]$, serão $3 \times 2 \times 4$ experimentos, o mesmo número de elementos de uma matriz multidimensional $3 \times 2 \times 4$.

Para definir um modelo com três fatores, considere $A[1..a]$, $B[1..b]$, $C[1..c]$ como

três fatores, com a quantidade de níveis *a, b, c* respectivamente. Com estes fatores, o modelo é dado por:

Modelo 3.2 $y_{ijmk} = \mu + A_i + B_j + C_m + AB_{ij} + AC_{im} + BC_{jm} + ABC_{ijm} + e_{ijmk}$

► **Definições:**

- y_{ijmk}: é o valor medido na k-ésima replicação;
- A_i: efeito principal de cada nível $(1..a)$ do fator A;
- B_j: efeito principal de cada nível $(1..b)$ do fator B;
- C_m: efeito principal de cada nível $(1..c)$ do fator C;
- AB_{ij}: a primeira ordem de interação de cada nível $([1..a] \times [1..b])$ dos fatores AB;
- AC_{im}: a primeira ordem de interação de cada nível $([1..a] \times [1..c])$ dos fatores AC;
- BC_{jm}: a primeira ordem de interação de cada nível $([1..b] \times [1..c])$ dos fatores BC;
- ABC_{ijm}: a segunda ordem de interação de cada nível $([1..a] \times [1..b] \times [1..c])$ dos fatores ABC;
- e_{ijmk}: são erros experimentais. Representa a diferença entre o valor medido e o esperado: $e_{ijmk} = y_{ijmk} - \sum_{k=1}^{r} y_{ijmk}/r$, para r o número de replicações.

Para calcular o efeito principal, basta subtrair da média do nível a média geral μ. Por exemplo, considere que $A[a_1, a_2]$, $B[b_1, b_2]$ e $C[c_1, c_2]$ sejam três fatores, com seus valores dados na Tabela 3.13.1, efeitos principais na Tabela 3.13.2, primeira ordem de interação na Tabela 3.13.3 e segunda ordem na Tabela 3.13.4, com métrica BM (Baixo Melhor). O efeito principal B_1 é a média das linhas b_1 subtraída pela média geral μ, resultando em $B_1 = [0,72 + 0,34 + 0,74 + 0,3]/4 - 1,576 = -1,051$. O efeito da primeira ordem de interação envolve mais números, porém segue o mesmo princípio. Por exemplo, para calcular o efeito de interação em primeira ordem AB_{11}, os procedimentos são: calcula-se a média dos valores comuns entre os níveis $a_1 \times b_1$ ([0,72+ 0,34]/2); subtrai-se dessa média a soma da média geral (μ) com os efeitos principais A_1 e B_1; o resultado é $AB_{11} = [0,72+0,34]/2-[1,576-0,001-1,051] = 0,006$. O efeito de interação em segunda ordem é mais complexo, sendo dependente

dos efeitos principais e primeira ordem de interação, bem como da média geral, resultando em $ABC_{ijm} = y_{ijm} - [\mu + A_i + B_j + C_m + AB_{ij} + AC_{im} + BC_{jm}]$.

Tabela 1			
A	B	C	
		c_1	c_2
a_1	b_1	0,720	0,340
	b_2	2,900	2,340
a_2	b_1	0,740	0,300
	b_2	2,920	2,350
Media geral (μ)		1,576	

Tabela 4			
Segunda ordem de interação			
Interação ABC_{ijm}			
		C_1	C_2
A_1	B_1	-0,006	0,006
	B_2	0,006	-0,006
A_2	B_1	0,006	-0,006
	B_2	-0,006	0,006

Tabela 3								
Primeira ordem de interação								
Interação AB_{ij}			Interação AC_{im}			Interação BC_{jm}		
	B_1	B_2		C_1	C_2		C_1	C_2
A_1	0,006	-0,006	A_1	-0,009	0,009	B_1	-0,039	0,039
A_2	-0,006	0,006	A_2	0,009	-0,009	B_2	0,039	-0,039

Tabela 2					
Efeitos principais					
Efeito A_i		Efeito B_j		Efeito C_m	
A_1	A_2	B_1	B_2	C_1	C_2
-0,001	0,001	-1,051	1,051	0,244	-0,244

Tabela 3.13: Exemplo de tabelas de efeitos para os fatores A[a_1,a_2], B[b_1,b_2], C[c_1,c_2].

Nas Tabelas 3.13, os resultados para os efeitos principais podem ser entendidos como: os níveis do fator A refletem as menores variações, sendo potencialmente semelhantes, com -0,001 favorecendo A_1; porém é necessário que se calcule o intervalo de confiança para os efeitos A_i, com a inclusão de zero nesse intervalo caracterizando insignificância estatística para os níveis do fator A; os níveis de B têm variações consideráveis, favorecendo B_1 em -1,051 abaixo da média; portanto, B contém níveis potencialmente diferentes; os níveis de C contém razoáveis diferenças, com -0,244 favorecendo C_2. É importante perceber que há um intervalo que parametriza a intensidade que um nível é melhor que outro. Esse intervalo é a média geral negativa e positiva: [-1,576, +1,576]. Dessa forma, o valor mínimo do efeito é a média negativa, o que poderia acontecer no exemplo B_1 se todo valor b_1 fosse zero. Nessas condições, sendo B um fator sistema, com níveis sistema b_1 e sistema b_2, seria razoável dizer que o tempo de b_1 seria nulo diante de b_2, e B_2 teria efeito igual

a $+\mu$. É importante perceber também que a insignificância estatística é definida pelo intervalo de confiança, possível de ser calculado para cada efeito principal do modelo fatorial. Portanto, intervalos de confiança que incluem zero definem sistemas insignificantes estatisticamente.

Análise de Variância - ANOVA

Tabela 1: ANOVA-Modelo sem replicação		
Componentes SQ	Porcentagem de variação	Grau de liberdade
$SQY = \sum y_{ijm}^2$		abc
$SQ0 = abc\mu^2$		1
$SQT = SQY - SQ0$	100	$abc - 1$
$SQA = bc\sum A_i^2$	$100\frac{SQA}{SQT}$	$a - 1$
$SQB = ac\sum B_j^2$	$100\frac{SQB}{SQT}$	$b - 1$
$SQC = ab\sum C_m^2$	$100\frac{SQC}{SQT}$	$c - 1$
$SQAB = c\sum AB_{ij}^2$	$100\frac{SQAB}{SQT}$	$(a-1)(b-1)$
$SQAC = b\sum AC_{im}^2$	$100\frac{SQAC}{SQT}$	$(a-1)(c-1)$
$SQBC = a\sum BC_{jm}^2$	$100\frac{SQBC}{SQT}$	$(b-1)(c-1)$
$SQABC = \sum ABC_{ijm}^2$	$100\frac{SQABC}{SQT}$	$(a-1)(b-1)(c-1)$

Tabela 2: ANOVA-Exemplo			
Componente	Soma Quad.	Porcentagem	G.L
SQY	29,20610		8
SQ0	19,87651		1
SQT	9,32959	100,00000	7
A	0,00001	0,00013	1
B	8,84101	94,76317	1
C	0,47531	5,09468	1
AB	0,00031	0,00335	1
AC	0,00061	0,00657	1
BC	0,01201	0,12876	1
ABC	0,00031	0,00335	1

Tabela 3.14: ANOVA

O comportamento dos fatores e suas interações pode ser caracterizado através de uma tabela *ANOVA* (*Analysis of variance*), complementando o estudo sobre experimento fatorial. A *ANOVA* consiste na soma de quadrados dos dados fatoriais, estabelecendo a porcentagem de variação que cada componente fatorial representa. Para o Modelo 3.2, a tabela *ANOVA* pode ser representada pelos modelos de Tabelas

3.14.1 e 3.15, onde os símbolos SQ são uma abreviação para soma dos quadrados. Para o exemplo das Tabelas 3.13, os componentes SQ podem ser calculados a partir das definições seguintes, com resultados resumidos na Tabela 3.14.2.

- $SQY = 0,72^2 + ... + 2,35^2 = 29,20610$;
- $SQ0 = abc\mu^2 = 2 \times 2 \times 2 \times 1,576^2 = 19,87651$;
- $SQT = 29,20610 - 19,87651 = 9,32959$;
- $SQA = (-0,001^2 + 0,001^2) \times 2 \times 2 = 0,00001$;
- $SQB = (-1,051^2 + 1,051^2) \times 2 \times 2 = 8,84101$;
- $SQC = (0,244^2 - 0,244^2) \times 2 \times 2 = 0,47531$;
- $SQAB = (0,006^2 - 0,006^2 - 0,006^2 + 0,006^2) \times 2 = 0,00031$;
- $SQAC = (-0,009^2 + 0,009^2 + 0,009^2 - 0,009^2) \times 2 = 0,00061$;
- $SQBC = (-0,039^2 + 0,039^2 + 0,039^2 - 0,039^2) \times 2 = 0,01201$;
- $SQABC = (-0,00625^2 + ... + 0,00625^2) = 0,00031$;

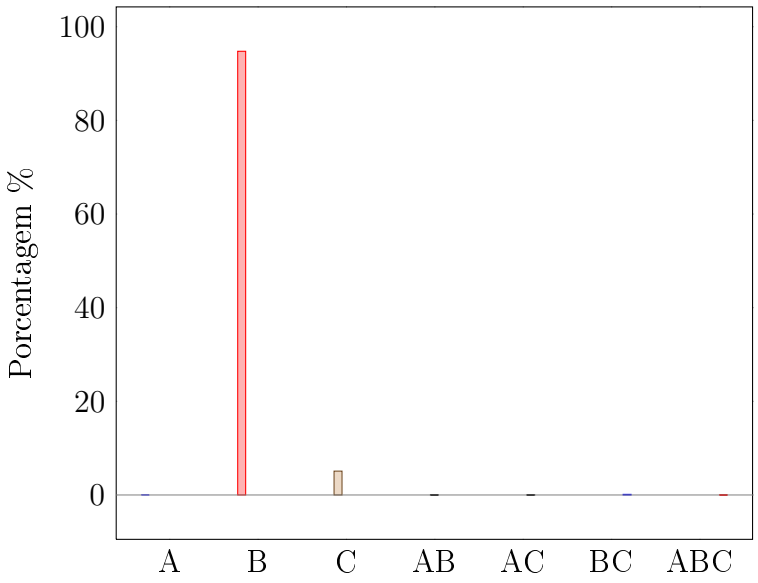


Figura 3.6: Gráfico ANOVA-Exemplo

Os componentes SQ calculados não possuem potencial em variações, com exceção de SQB que representa o fator B. O princípio de comparação para variação de

fatores pela ANOVA é semelhante às comparações dos efeitos principais A_i, B_j e C_m, onde temos o maior efeito com B, o mediano com C e o mínimo com A. Pela comparação desses efeitos principais, a variação ocorre no intervalo negativo e positivo da média geral $[-\mu, +\mu]$, onde a tendência de igualdade segue a aproximação de zero. Pela tabela *ANOVA*, a diferença dos fatores pode ser definida pela porcentagem que representam na ANOVA. Quando a porcentagem do fator ou da interação entre fatores é mínima, fica a critério do analista de desempenho selecionar as ocorrências que realmente considera importante para o estudo em questão.

ANOVA-Modelo com replicações		
Componentes SQ	Porcentagem de variação	Grau de liberdade
$SQY = \sum y_{ijmk}^2$		$abcr$
$SQ0 = abcr\mu^2$		1
$SQT = SQY - SQ0$	100	$abcr - 1$
$SQA = bcr \sum A_i^2$	$100\frac{SQA}{SQT}$	$a - 1$
$SQB = acr \sum B_j^2$	$100\frac{SQB}{SQT}$	$b - 1$
$SQC = abr \sum C_m^2$	$100\frac{SQC}{SQT}$	$c - 1$
$SQAB = cr \sum AB_{ij}^2$	$100\frac{SQAB}{SQT}$	$(a-1)(b-1)$
$SQAC = br \sum AC_{im}^2$	$100\frac{SQAC}{SQT}$	$(a-1)(c-1)$
$SQBC = ar \sum BC_{jm}^2$	$100\frac{SQBC}{SQT}$	$(b-1)(c-1)$
$SQABC = r \sum ABC_{ijm}^2$	$100\frac{SQABC}{SQT}$	$(a-1)(b-1)(c-1)$

Tabela 3.15: Modelo ANOVA com replicação.

Capítulo 4

Refatoração do NPB em Componentes

A refatoração do *NAS Parallel Benchmarks* (NPB) em componentes a partir do código monolítico original escrito em Fortran foi realizada segundo as convenções do modelo de componentes # sobre a plataforma de componentes HPE, por se tratar atualmente de seu sistema de programação de referência. Uma vez que o HPE encontra-se implementado sobre a plataforma CLI, empregando a linguagem C# para escrita do código dos componentes, recursos disponíveis em linguagens de alto nível são empregados ao longo da fatoração, porém buscando conservar a estrutura original de paralelismo dos códigos nativos. Esse é um requisito importante da refatoração em componentes discutida neste capítulo, tendo em vista os objetivos da avaliação de desempenho que serão apresentados no Capítulo 5, o qual tentará isolar a sobrecarga intrínseca do uso de componentes no desempenho de um subconjunto dos programas do NPB.

Antes da fatoração em componentes, faz-se necessário um processo bem elaborado de conversão do código paralelo escrito Fortran/MPI para C#/MPI.NET. Essa conversão não tem o propósito de competir com o desempenho obtido no código nativo. Uma comparação entre as duas versões serviria apenas para mensurar a distância entre o desempenho dos códigos executados na máquina virtual da plataforma Mono e o desempenho do código nativo, semelhante ao que foi realizado pela equipe do NAS ao desenvolver versões do NPB para a linguagem Java [42].

Com o processo de refatoração em componentes, deseja-se explorar os sofisticados recursos da engenharia de software utilizando paralelismo de memória distribuída, medir desempenho da máquina virtual, e avaliar plataforma de componentes

paralelos executando aplicações científicas. Para isso, a refatoração é realizada em quatro principais etapas: definição do subconjunto do NPB a ser refatorado; análise dos interesses dos programas, identificando especialmente aqueles relacionados a configuração paralela; definição do método de conversão; e reorganização do código segundo o método definido. Seguem-se então os testes, com verificação dos resultados numéricos calculados. Como premissa, assume-se que o código refatorado em componentes deve preservar os códigos computacionais, padrão de comunicação e implementações das estruturas de dados usadas nos programas originais, demonstrando o potencial de reaproveitamento do código legado e permitindo um melhor isolamento da sobrecarga de desempenho inerente ao uso de componentes na avaliação de desempenho.

4.1 Escolha dos Benchmarks

Em um processo de avaliação de desempenho, é de fundamental importância a escolha correta das cargas de trabalho a serem submetidas ao dispositivo computacional cujo desempenho será estudado. Nesse sentido, sendo o objeto de estudo deste trabalho uma plataforma de componentes, necessitamos de *benchmarks* confiáveis e, quando possível, já difundidos em vários trabalhos científicos. O conjunto de aplicativos do NPB satisfaz essa condição, oferecendo boa documentação e credibilidade na comunidade científica [13, 26, 48, 62]. Além disso, o mapeamento aos *Dwarfs* descrito no Capítulo 3 torna o NPB uma ferramenta que contempla vários problemas matemáticos de interesse das ciências, especialmente as equações diferenciais contidas nos *benchmarks* de interesse do FT, BT, LU e SP, os quais cobrem três dos sete *Dwarfs* propostos por Colella [37].

BT e LU contemplam principalmente a classificação *Dwarf* número um (1). Foram escolhidos pelo potencial que oferecem para refatoração em componentes. Possuem como característica predominante o acesso à memória de forma consecutiva, com dados adjacentes para as linhas e colunas (*unit-stride memory accesses*), importante para avaliar o peso da verificação de limites de vetores em máquinas virtuais [70].

O FT está na classificação 3. Foi escolhido por representar o maior *kernel* do NPB em codificação, contendo potencial para refatoração. Como características principais, executa cálculos intensos de multiplicações e adições com números complexos, organizando-se em vários estágios (*Butterfly network* [45]) e explorando recursos de comunicação coletiva.

O SP está na classificação 5. Foi escolhido também pelo potencial que oferece para refatoração, tendo em vista a complexidade do seu código. Executa resolvedores em três dimensões sobre uma grade estruturada e utiliza comunicação assíncrona para envio e recebimento de mensagens entre processos. A comunicação entre os processos organiza-se segundo padrões de comunicação em anel, com cada processo comunicando-se com seus vizinhos anterior e posterior.

A classificação 2 contém ainda o CG (*Conjugate Gradient*), o qual não será usado por possuir um código de arquitetura simplificado cuja escala não convém ao processo de refatoração. As classificações 4 e 6 não possuem relação com o NPB. A classificação 7 contém o EP, o qual também não será usado pelo mesmo motivo do CG. Além disso, não apresenta comunicação entre os processos [13], característica importante para o contexto deste trabalho. Assim, foram selecionados 3 *aplicações simuladas* (SP, BT e LU) e 1 *kernel* (FT), os quais serão refatorados em componentes e utilizados no processo de avaliação de desempenho do HPE.

4.2 Construção da versão C#/MPI.NET

A construção da versão C#/MPI.NET consiste na extração da estrutura lógica de códigos nativos (Fortran) com o objetivo de recodificá-los sob a sintaxe C# com MPI.NET [46]. Para isso, dividimos o trabalho de construção da nova versão em três etapas.

A primeira etapa visa reaproveitar a versão serial do NPB em Java, construída pela equipe do NAS. Essa é uma versão oficial, disponível na versão 3.0 do NPB, a qual simula matrizes multidimensionais em vetores, pois Java não suporta vetores multidimensionais reais. Somente vetores multidimensionais aninhados (vetores de vetores [1]) são suportados, os quais não apresentam teoricamente bom potencial de desempenho para códigos numéricos onde a localidade dos dados ao percorrer os vetores em suas várias dimensões é relevante. Por esse motivo, foi necessário fazer a conversão direta de Java para C# e posteriormente retirar a simulação de índices, aproveitando o fato de que C# possui suporte a vetores multidimensionais reais. A simulação de índices da versão Java é explicada no Exemplo 4.1.

Exemplo 4.1 *A simulação de índices pode ser explicada tomando como exemplo as referências* u e m *para vetor unidimensional e multidimensional respectivamente. Seja* u *um vetor de tamanho* 8, m *um vetor multidimensional de tamanho* 2×4, *e*

[1] *Jagged arrays.*

$n = 8$. *A relação de indexação entre ambos vetores é dada por:* $u(i \times n/2 + j) = m(i, j)$.

A eliminação da simulação de índices na versão C# devolve a originalidade dos vetores multidimensionais que estão presentes nos códigos Fortran, e serve para avaliar custos da verificação de limites de vetores [70], característica de linguagens de programação seguras, dentre as quais estão Java e C#, que tende a afetar o desempenho de códigos científicos. Com o procedimento de conversão, foram geradas 2 versões de cada programa escolhido do NPB, totalizando 4 versões, uma com simulação de índices e outra usando vetores multidimensionais. Um trecho da conversão do SP de Java para C# multidimensional pode ser visto no Apêndice A.1.

A segunda etapa consiste em fazer o levantamento da estratégia do código paralelo dos programas em Fortran/MPI para desenvolver as versões paralelas em C#/MPI.NET. Uma vez que o paralelismo inerente ao código nativo está implementado sob bibliotecas de passagem de mensagem MPI, pode ser facilmente convertido para MPI.NET. Algumas codificações C# serial da primeira etapa puderam ser reaproveitadas na codificação das versões paralelas C#/MPI.NET, observando e mantendo a estratégia dos códigos originais Fortran/MPI. Porém, a conversão direta das versões Fortran/MPI foi predominante. Essa foi uma etapa bastante rigorosa deste trabalho e constitui contribuição tornada pública[2] para aqueles que desejam ter acesso a código C#/MPI.NET dos programas do NPB. Em particular, as versões sequenciais tem sido empregadas para comparar o desempenho das máquinas virtuais Mono, .NET, JVM e OpenJVM.

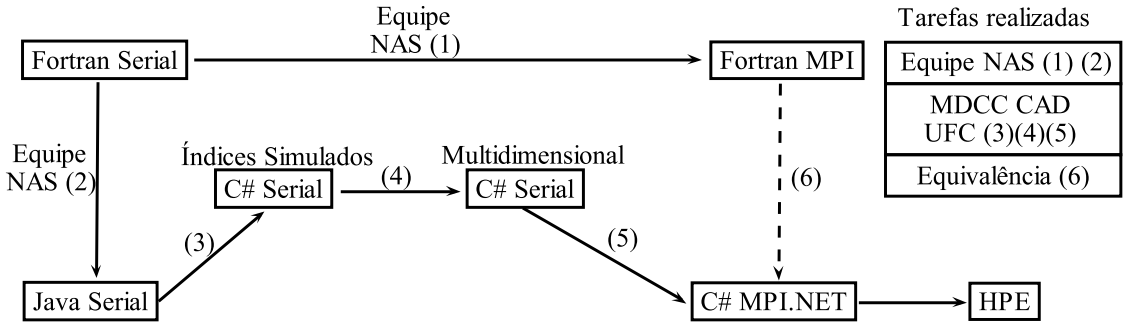


Figura 4.1: Ciclo de codificação do NPB.

A terceira e última etapa recorre às otimizações, organização e reaproveitamento de códigos comuns entre os *benchmarks*. O processo de codificação do NPB é

[2] http://npb-for-hpe.googlecode.com

ilustrado na Figura 4.1. A figura expressa que a primeira versão do NPB foi em Fortran Serial, o que possibilitou a codificação da versão Fortran/MPI, e posteriormente a versão Java. As demais versões, C# Serial com índices simulados e multidimensional, são desenvolvidas a partir da versão Java. Finalmente, C# MPI.NET é desenvolvida observando o princípio de equivalência, o que significa dizer que a estratégia do código nativo foi mantida. Nessa última versão, não há predominância de orientação a objetos, de modo que apenas uma classe é desenvolvida, contendo as características fundamentais do código nativo. Portanto, a refatoração em componentes para o HPE tem origem a partir de uma codificação C# primitiva, com uma única classe, contendo a estratégia dos procedimentos e estruturas de dados das versões Fortran/MPI.

4.3 Considerações Gerais sobre o Processo de Refatoração

Ao serem construídas versões monolíticas do NPB em C#/MPI.NET, os programas resultantes mantiveram a arquitetura herdada de suas respectivas versões originais em Fortran, especialmente no que diz respeito a organização procedural, definição das estruturas de dados e uso das subrotinas de MPI. Houve também sucesso ao aplicar o mesmo critério na fatoração do código C#/MPI.NET em componentes da plataforma HPE, sem alterações significativas nos códigos que declaram e inicializam as estruturas de dados e que descrevem as computações. Manter a originalidade do código é um fator importante para a avaliação de desempenho efetuada no Capitulo 5, pois deseja-se calcular o peso ou efeito da fatoração dos códigos em componentes, onde mudanças nas estruturas de dados e algoritmos do programa não seriam fáceis de terem seus efeitos isolados.

É importante ressaltar que a capacidade de refatoração em componentes sem grandes alterações no código é bastante útil em um contexto de aplicações onde a existência de códigos legados é preponderante, como acontece no domínio das ciências e engenharias, uma vez que torna menos árdua a tarefa de garantir a corretude dos programas resultantes.

O grau de granularidade da refatoração é semelhante ao grau de decomposição em procedimentos dos aplicativos originais, porém com componentes adicionais, que têm a função de controlar interesses diversos como topologia, padrão de interação entre processos, identificação da dimensão, definição de classes de problema, entre outros. Ao todo, a refatoração do SP, BT, LU e FT resultou na identificação de 263 componentes, sendo 116 componentes abstratos e 147

componentes concretos, dispostos em 5 espécies suportadas pelo HPE, sendo elas: computação, estrutura de dados, ambiente, qualificador, sincronizador e aplicação. A quantidade de componentes tem excedido os 185 procedimentos nos códigos monolíticos. Justifica-se o excesso devido aos componentes abstratos, aspecto não usual na versão monolítica, e também aos componentes que expressam configuração e controle do paralelismo, recursos espalhados ao longo da codificação original. Porém, comparando-se apenas componentes concretos temos uma redução significativa de 38[3] unidades.

Considerando-se os benefícios, a refatoração evita códigos replicados ou redundantes como ocorre na versão monolítica, e ainda permite o reuso de componentes entre *benchmarks*, especialmente entre SP e BT, pois possuem estrutura semelhante. O compartilhamento de componentes entre *benchmarks* é demonstrado na Tabela 4.1, onde claramente SP e BT demonstram significativo uso comum de recursos, coincidentemente compartilhando 38 componentes. O reaproveitamento de componentes entre as *aplicações* do NPB permite sugerir que também exista um potencial de reaproveitamento em aplicações que não são do NPB, um benefício usual ao se desenvolver componentes. O potencial está principalmente nos componentes que foram isolados através de interesses comuns em configurações e controle do paralelismo, o que não é simples de separar com ferramentas tradicionais, apesar do significativo potencial de expressividade dessas ferramentas. Os principais interesses em configurações e controle do paralelismo no NPB são definidos em seguida.

- **topologias de processos**: define a forma como os processos são organizados para a troca de informações. Por exemplo, malhas tridimensionais, hipercubos etc;
- **particionamentos das estruturas de dados**: interesse que representa a estratégia de encapsulamento da partição e distribuição dos dados às aplicações;
- **padrões de comunicação recorrentes**: representa a forma padrão de comunicação entre processos vizinhos, definindo o fluxo de movimentação dos dados na topologia de processos;
- **classe de problema**: parametriza o tamanho de problema que o *benchmark*

[3] 38=185-147

resolve. A classe de problema define todas as informações essenciais para execução do *benchmark*, como número de iterações, constantes numéricas para validação de resultados, etc. No NPB, as classes de problema padrões são representadas por letras do alfabeto, tais como S (teste), W, A, B, C, D etc, da menor para a maior carga de trabalho representada. Na versão fatorada em componentes, as informações de classes são encapsuladas em componentes do tipo INSTANCE e CLASS (Seção 4.4.6). Todo componente do NPB que utiliza informação sobre classe de problema obrigatoriamente tem parâmetros de contexto genéricos do tipo INSTANCE e CLASS;

- **estruturas de dados paralelas**: encapsula a estrutura de dados em matrizes e vetores, armazenando informações definidas por processos de forma paralela;
- **verificação de erros**: efetua a checagem final da execução, indicando erros e validando cálculos feitos ao longo da execução do *benchmark*;
- **cronômetro**: responsável por calcular o tempo gasto para solucionar o problema numérico.

Nas seções seguintes, são apresentados os principais componentes do NPB. Utilizam-se duas fontes de texto para distinguir componentes abstratos de componentes concretos, como a seguir: COMPONENTEABSTRATO, ComponenteConcreto.

4.4 SP e BT

As aplicações simuladas SP e BT possuem estrutura semelhante, pois resolvem três sistemas de equações não-acoplados, referentes aos eixos x, y e z no espaço tridimensional, através do método ADI (*Alternating Direct Implicit*). O sistema é pentadiagonal no SP, enquanto que é bloco-tridiagonal, com blocos 5×5, no BT.

A arquitetura comum de SP e BT encontra-se ilustrada na Figura 4.2. Ambos são definidos pelo componente abstrato ADISOLVER3D. Como é possível observar, a diferença fundamental entre as arquiteturas de SP e BT está na implementação dos resolvedores aplicados a cada uma das três dimensões, representados pelos componentes aninhados *x_ solve*, *y_ solve* e *z_ solve*. De fato, através dos parâmetros de contexto que guiam a especificação de seus componentes concretos, o componente ADISOLVER3D possibilita utilizar resolvedores distintos em componentes concretos que o implementam. Os parâmetros de contexto de ADISOLVER3D são discutidos de forma mais aprofundada na Seção 4.4.1.

Espécies	**SP**	**BT**	**LU**	**FT**	(1)	(2)	(3)
Componentes Abstratos							
Computação	16	24	13	14	11	-	-
Sincronização	3	3	2	1	3	1	1
Qualificador	16	15	21	17	14	7	7
Estrutura de Dados	4	4	5	2	4	-	-
Ambiente	7	7	8	9	5	3	3
Total	**46**	**53**	**49**	**43**	**38**	**11**	**11**
Componentes Concretos							
Computação	27	37	15	28	10	-	-
Sincronização	4	4	11	2	4	2	2
Estrutura de Dados	4	4	5	8	4	2	2
Ambiente	10	10	9	5	5	3	3
Total	**45**	**55**	**40**	**43**	**23**	**7**	**7**

(1) Compartilhado entre **SP** e **BT**;

(2) Compartilhado entre **SP**, **BT** e **LU**;

(3) Compartilhado entre **SP**, **BT**, **LU** e **FT**.

Tabela 4.1: Quantidade de Componentes Fatorados e Compartilhados

Componentes concretos de ADISolver3D possuem o componente aninhado *adi* como o de maior importância em sua configuração, cujo tipo é determinado pelo componente abstrato ADI, o qual especifica os passos da implementação do método ADI. Componentes concretos de ADI possuem os componentes aninhados *copy_faces*, *compute_rhs*, *x_solve*, *y_solve*, *z_solve* e *add*, cujos tipos são determinados pelos componentes abstratos CopyFaces, ComputeRHS, Solver e Add. Os componentes aninhados *x_solve*, *y_solve*, *z_solve* representam os resolvedores aplicados a cada um dos três eixos do espaço tridimensional, os quais, como mencionado anteriormente, diferenciam as aplicações SP e BT. Portanto, SP e BT são essencialmente diferenciados por conjuntos distintos de implementações do componente abstrato Solver para cada uma das três dimensões. O componente Solver é explicado com mais detalhes na Seção 4.4.2. Os demais componentes de ADISolver3D são compartilhados integralmente entre SP e BT.

O componente CopyFaces, cujo componente concreto é denominado CopyFacesImpl, tem a função de copiar valores nas *faces* de um conjunto de *células* de um processo, onde essas células representam partições do domínio segundo a estratégia de multi-partição (Seção 4.4.4). As faces de uma célula representam as bordas de sua partição do domínio compartilhadas com processos vizinhos que possuem as suas células adjacentes. Por sua vez, ComputeRHS tem a função de calcular os valores da matriz *rhs* do sistema, utilizada também por componentes Solver. O componente Add é responsável por somar os valores contidos na matriz *rhs* com valores da matriz principal *u*.

Na estrutura de ADISolver3D, há três componentes para inicialização das estruturas de dados processadas pelas computações, as quais estão encapsuladas no componente aninhado *problem_data*, do tipo ProblemData (Seção 4.4.7). O componente aninhado *initialize*, do tipo Initialize, é o responsável por inicializar a matriz *u*, definindo valores iniciais para as faces nos sentidos norte, sul, leste, oeste, acima e abaixo. O componente aninhado *lhs_init*, do tipo LHSInit, é responsável por inicializar os valores das diagonais da matriz *lhs* para 1. Por fim, o componente aninhado *exact_rhs*, do tipo ExactRHS, percorre todas as células do domínio mantidas pelo processo, definindo os dados de vetores usados pelo componente *compute_rhs*.

Os componentes ADISolver3D e Solver são apresentados detalhadamente nas seções seguintes, com ênfase no significado dos seus parâmetros de contexto. Outros componentes que foram isolados, representando interesses de configuração

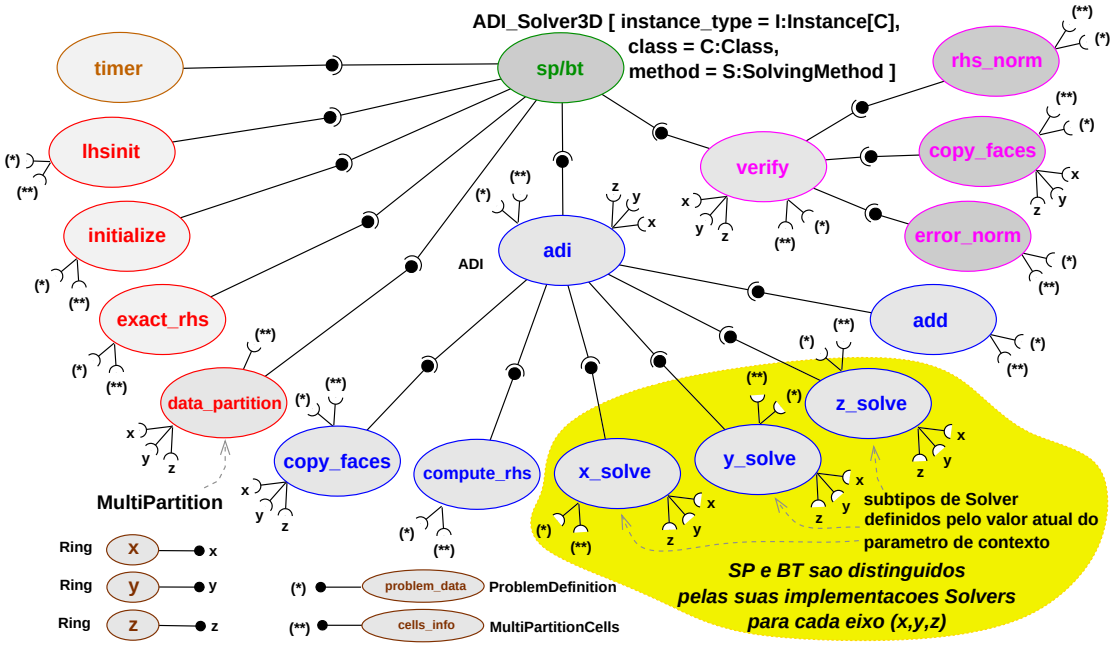


Figura 4.2: Arquitetura SP/BT.

BT - Classe W	
ADISolver3D [	method=BTSolvingMethod, class=Class_W, instance=Instance_BT[Class_W]]
SP - Classe B	
ADISolver3D [	method=SPSolvingMethod, class=Class_B, instance=Instance_SP[Class_B]]

Tabela 4.2: Instanciações possíveis para ADISolver3D

paralela, como topologias entre processos, padrões de comunicação entre processos, particionamento e distribuição de dados, etc, também são detalhados nas próximas seções.

4.4.1 O Componente Abstrato ADISolver3D: Aplicação

No componente abstrato ADISolver3D, a diferença entre SP e BT é determinada pelos componentes aninhados resolvedores do tipo Solver, chamados *x_solve*, *y_solve* e *z_solve*, respectivamente nos eixos x, y e z. Como a implementação de tais resolvedores é diferente para cada eixo do espaço tridimensional, Solver possui o parâmetro de contexto *axis*.

Para estabelecer as diferenças entre SP e BT, ADISolver3D exige três

parâmetros de contexto, denominados *method*, *class* e *instance_type*.

O parâmetro de contexto *method* determina o método de resolução aplicado. Por isso, seu tipo é limitado pelo componente abstrato SolvingMethod, da espécie qualificador. Os métodos utilizados por SP e BT são determinados pelos componentes abstratos SPSolvingMethod e BTSolvingMethod, respectivamente, subtipos de SolvingMethod. O parâmetro *method* determina o valor do parâmetro de contexto de mesmo nome dos componentes *x_solve*, *y_solve* e *z_solve*. Logo, além do parâmetro de contexto *axis*, o componente Solver também possui um parâmetro de contexto *method*, de forma a diferenciar suas implementações específicas do método ADI de solução.

O parâmetro de contexto *class* determina a classe de problema que será aplicada durante a execução, sendo limitada pelo componente abstrato Class da espécie qualificador. Para isso, há subtipos de Class para cada classe de problema padrão do NPB (Class_S, Class_W, Class_A, Class_B e Class_C), as quais serão aplicadas na instanciação de ADISolver3D para uma determinada classe de problema.

O terceiro parâmetro, denominado *instance_type*, é limitado pelo componente abstrato Instance, da espécie ambiente. Determina os valores atuais dos parâmetros da aplicação para uma determinada classe de problema. Para isso, Instance também possui o parâmetro de contexto *class*, embora os componentes concretos de ADISolver3D sejam *genéricos* nesse parâmetro, quando o valor atual é o próprio limite do parâmetro de contexto. Os subtipos específicos de Instance para SP e BT são Instance_SP e Instance_BT, aplicados aos parâmetros de contexto atuais *instance_type* em componentes concretos que necessitam de valores de parâmetros específicos da aplicação. Por exemplo, esse é o caso de componentes concretos de Solver, o qual possui os parâmetros de contexto *class* e *instance_type*. Nos componentes concretos de Solver para SP e BT, *instance_type* é suprido por Instance_SP e Instance_BT, respectivamente.

Em ADISolver3DImpl, o componente concreto de ADISolver3D desenvolvido para SP e BT, os parâmetros de contexto atuais são definidos pelos limites dos parâmetros de contexto formais. Portanto, trata-se de um componente concreto genérico, ou universalmente polimórfico, atendendo a necessidade de SP ou BT por uma única implementação de ADISolver3D. Tal generalização permite ao ADISolver3DImpl ser aplicado a implementações alternativas do método ADI, sendo suficiente para isso definir subtipos apropriados de SolvingMethod e Instance,

bem como implementações específicas de INSTANCE e SOLVER, para cada eixo e para cada subtipo de SOLVINGMETHOD e INSTANCE definidos.

A principal dificuldade em compartilhar componentes entre SP e BT está justamente na implementação de componentes que aceitam parâmetros de forma genérica. O sistema de tipos do HPE, chamado HTS (*Hash Type System*), é bastante útil para esse propósito, pois torna possível o desenvolvimento de uma única versão concreta de ADISOLVER3D, a qual permite que parâmetros de contexto sejam supridos por qualquer método, classe ou instância. Tal dinâmica possibilita a instanciação de qualquer componente concreto de SOLVER (Figura 4.3) dentro do ADISolver3DImpl, unicamente guiada pelos parâmetros *method* e *axis*. Um esboço do suprimento de parâmetros de contexto de ADISOLVER3D, visando instanciações apropriadas de ADISolver3DImpl, é apresentado na Tabela 4.2, onde são instanciadas as aplicações BT e SP, respectivamente para as classes de problema W e B.

4.4.2 O Componente Abstrato SOLVER: Resolvedores ADI

Ao longo da execução do SP ou do BT, diversos componentes armazenam e recuperam configurações e dados dispostos em matrizes que são compartilhadas na configuração do componente ADISOLVER3D através do componente aninhado *problema_data*. Sobre esses dados, que definem o estado de execução da aplicação, são aplicados os componentes resolvedores. Como discutido anteriormente, o tipo desses componentes resolvedores, denominados *x_solve*, *y_solve* e *z_solve*, é determinado pelo componente abstrato SOLVER, com implementações diferentes para BT e SP, bem como para cada eixo (x, y e z).

Em SOLVER, o método de solução é determinado pelo parâmetro de contexto *method*, limitado pelo componente abstrato SOLVINGMETHOD da espécie qualificador. Os componentes resolvedores recebem o valor atual de *method* a partir do parâmetro de contexto de mesmo nome da configuração de ADISOLVER3D. Porém, diferente dos parâmetros genéricos aplicados a ADISolver3DImpl, o valor atual dos parâmetros *method* aplicados aos componentes concretos de SOLVER são específicos, sendo definidos por BTSOLVINGMETHOD ou SPSOLVINGMETHOD, subtipos de SOLVINGMETHOD, para representar resolvedores de BT ou SP, respectivamente. Sendo assim, a escolha do resolvedor adequado a SP ou BT é diretamente ligada ao parâmetro de contexto dado a ADISolver3DImpl em sua instanciação, o qual aceita que seu parâmetro *method* seja suprido por qualquer subtipo de SOLVINGMETHOD.

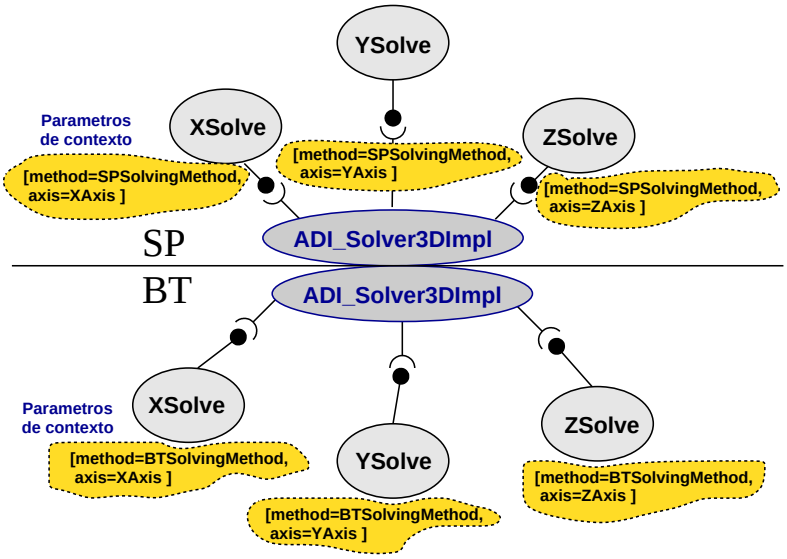


Figura 4.3: Componente SOLVER utilizado por ADISolver3DImpl em cada eixo, para SP e BT.

Por sua vez, a implementação específica de SOLVER para cada uma dos eixos x, y e z do espaço tridimensional é controlada pelo parâmetro de contexto *axis*, limitado pelo componente abstrato AXIS, da espécie qualificador. Assim, dentro do ADISOLVER3D, *x_solve*, *y_solve* e *z_solve* recebem, em seus parâmetros de contexto *axis*, os valores atuais XAXIS, YAXIS e ZAXIS, respectivamente, os quais constituem subtipos de AXIS. Dessa forma, existe uma implementação específica (componente concreto) de SOLVER para cada eixo, bem como para cada método.

O componente abstrato SOLVER utiliza também a classe de problema, definida pelos parâmetros de contexto *class* e *instance_type*, subtipos de CLASS e INSTANCE respectivamente, uma vez que os resolvedores precisam ter acesso aos parâmetros da aplicação cujos valores determinam a classe de problema.

O componente SOLVER pode ser melhor entendido quando o relacionamos com ADISOLVER3D, especialmente quando ADISOLVER3D supre seus parâmetros de contextos. A Figura 4.3 foi desenvolvida enfatizando esse relacionamento, onde ocorre a possibilidade de ADISolver3DImpl ser instanciado com os métodos SPSOLVINGMETHOD ou BTSOLVINGMETHOD.

4.4.3 O Componente Abstrato MESH3D: Topologia de Processos

Os processos paralelos no SP e BT são organizados obedecendo o mapeamento dos processos às células que representam partições do domínio tridimensional, formando um toro estruturado como uma composição de anéis que se sobrepõem

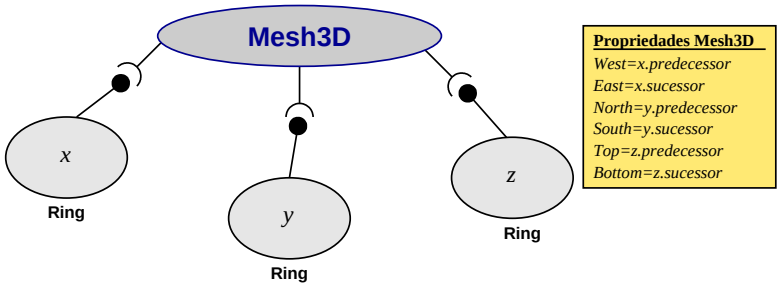


Figura 4.4: Componente Mesh3D.

nos três eixos (Figura 4.5). Detalhes sobre a estratégia de partição das células e como são distribuídas entre os processos são discutidos na Seção 4.4.4. A topologia dos processos determina os caminhos de comunicação entre eles.

O componente abstrato responsável pela topologia dos processos é denominado MESH3D, da espécie ambiente, o qual objetivamente permite que os processos identifiquem seus processos vizinhos em cada eixo: x (oeste e leste), y (norte e sul) e z (acima e abaixo).

Em sua configuração, MESH3D possui três componentes aninhados do tipo RING, denominados x, y e z, como ilustrado na Figura 4.4. RING é um componente abstrato da espécie ambiente. Sobrepostos de forma apropriada, como ilustrado na Figura 4.5, os anéis determinados por x, y e z formam o toro. Para isso, unidades de componentes RING possuem duas propriedades: *sucessor* e *predecessor*[4]. Os valores dessas propriedades são valores inteiros que permitem a um processo identificar os seus processos vizinhos posterior e anterior. De fato, constituem *ranks* de processos no padrão MPI, para comunicação usando MPI.NET no código dos componentes do tipo SHIFT da espécie sincronizador (Seção 4.4.5). A vizinhança entre os processos é calculada pelo componente aninhado *data_partition* na configuração de ADISOLVER3D, cujo tipo é determinado pelo componente abstrato MULTIPARTITION que será explicado na Seção 4.4.4.

A interface C# associada às unidades de MESH3D possui seis propriedades, indicando os identificadores dos processos nos sentidos do toro: *West*, *East*, *North*, *South*, *Top* e *Bottom*. *West* e *East* definem os vizinhos do processo no eixo X, retornando os valores de $x.predecessor$ e $x.successor$, onde x é uma propriedade que permite acessar a unidade importada do componente aninhado x. *North* e *South* definem os vizinhos no eixo Y, retornando os valores de $y.predecessor$ e $y.successor$

[4]No caso do HPE, tratam-se de propriedades das classes C# que definem as unidades do componente.

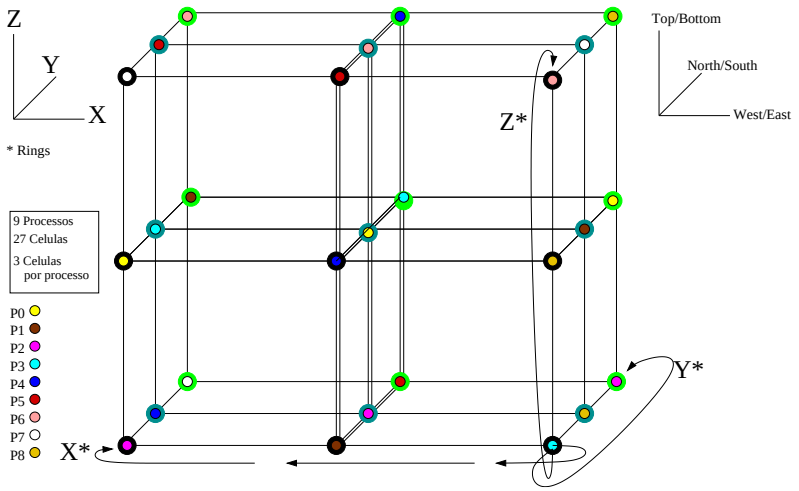


Figura 4.5: Topologia toro 3D 3x3x3.

do componente aninhado *y*. *Top* e *Bottom* definem os vizinhos no eixo *Z*, retornando os valores das propriedades *z.predecessor* e *z.successor* do componente aninhado *z*.

4.4.4 O Componente MultiPartition: Particionamento e Distribuição de Dados

O componente MultiPartition é responsável pelo encapsulamento do interesse de particionamento e distribuição de dados entre processos, usando a estratégia de multi-partição[5].

Seja n^2 o número de processos. Conforme a estratégia de multi-partição, o domínio espacial tridimensional é particionado em n^3 células, formando um toro quadrado tridimensional com dimensões $n \times n \times n$. Os processos são então mapeados às células do toro de tal forma que cada processo é associado a um conjunto de n células, conforme ilustrado na Figura 4.5 para o caso onde $n = 3$. Na ilustração, observe que o mapeamento tem a propriedade de preservar a vizinhança das células na topologia de processos resultante.

A fim de atender tais requisitos, MultiPartition possui três componentes aninhados: *cells_info*, de tipo MultiPartitionCells, da espécie ambiente; *instance*, do tipo Instance; e *topology*, do tipo Mesh3D. Os dois últimos já foram discutidos. O primeiro, MultiPartitionCells, é um componente de estado, responsável por guardar dados que definem a topologia das células e dos processos, calculados pelo procedimento de inicialização de MultiPartition. Todos os componentes de SP e BT que precisam dessas informações possuem um

[5]do inglês *multipartition*.

componente aninhado público denominado *cells_info*, os quais são compartilhados nas configurações onde são utilizados conforme ilustrado na Figura 4.2. Algumas das propriedades mais importantes expostas pelas unidades de MultiPartitionCells para os processos são o número de células associadas e suas coordenadas e o intervalo de início e fim para acesso a partição do domínio definido pela célula. Uma visão geral da configuração do componente abstrato MultiPartition pode ser vista na Figura 4.7, onde as elipses pontilhadas representam seus componentes aninhados privados, não visíveis na configuração de componentes que possuem componentes aninhados tipados pelo componente abstrato em questão. Portanto, são visíveis (públicos) somente os componentes aninhados *cells_info*, *x*, *y* e *z*.

Quatro vetores de dimensão *ncells* (número de células associadas a cada processo) definem o estado encapsulado por *cells_info* que representa informações sobre as células: *cell_coord*, que define as coordenadas da célula no toro; *cell_low*, que define os índices mais baixos em cada dimensão da partição do domínio referente a cada célula; *cell_high*, que define os índices mais altos em cada dimensão da partição do domínio referente a cada célula; *cell_size*, que define o tamanho da partição do domínio de cada célula. Essas matrizes são essenciais para orientar as travessias paralelas sobre o domínio tridimensional nos procedimentos computacionalmente intensivos do SP e do BT, a fim de realizar cálculos. No trecho de código da Figura 4.6, referente às unidades de MultiPartitionImpl, o componente concreto de MultiPartition, é apresentado o cálculo das propriedades *ncells* e *cells_coord*, onde pode-se observar que as coordenadas das células são definidas em função do número de identificação do processo (*node*) e o número de células de cada processo definido por *ncells*.

Para calcular o estado encapsulado em *cells_info*, o componente MultiPartition necessita de valores de parâmetros que definem o tamanho do problema, encontrados no seu componente aninhado *instance*, o que justifica sua existência na configuração. O componente MultiPartition ainda atualiza a vizinhança entre os processos acessando as propriedades do componente *topology*, que é do tipo Mesh3D, o que encontra-se também demonstrado no código da Figura 4.6, onde as propriedades *Successor* e *Predecessor* de cada anel *x*, *y* e *z*, tem seus valores atribuídos.

```
public class IMultiPartitionImpl<I,C>:BaseIMultiPartitionImpl<I,C>, IMultiPartition<I,C>
        where I:IInstance<C> where C:IClass {
            ..........
61    ncells=sqrt(total_nodes); /*Função sqrt() retorna a raíz quadrada*/
            ..........
73    cell_coord[0, 0]=mod(node, ncells); /*Função mod() retorna o resto da divisão*/
74    cell_coord[0, 1]=node/ncells;
75    cell_coord[0, 2]=0;
            ..........
112   X.predecessor=mod(i-1+ncells, ncells)+ncells*j;
113   Y.predecessor=i+ncells*mod(j-1+ncells, ncells);
114   Z.predecessor=mod(i+1, ncells)+ncells*mod(j-1+ncells, ncells);
115   X.successor=mod(i+1, ncells)+ncells*j;
116   Y.successor=i+ncells*mod(j+1, ncells);
117   Z.successor=mod(i-1+ncells, ncells)+ncells*mod(j+1, ncells);
            ..........
} /* X, Y, Z são RINGS */
```

Figura 4.6: Trecho de código do componente concreto MULTIPARTITIONIMPL.

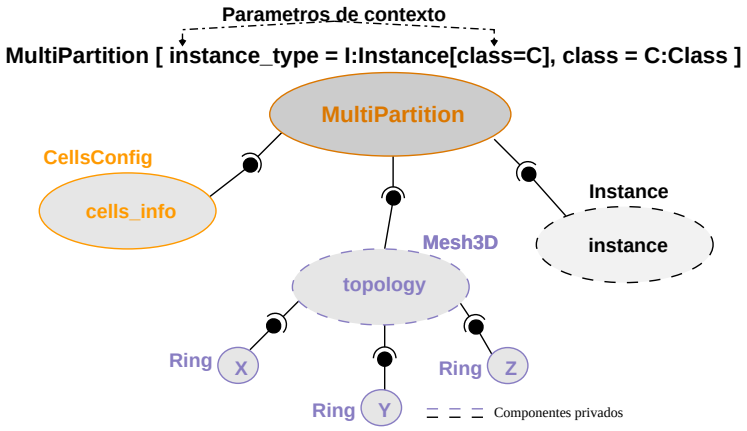


Figura 4.7: Componente Abstrato MULTIPARTITION.

4.4.5 Os componentes Shift e AllShift: Padrões de Comunicação entre Processos

Nesta seção, são apresentados os dois principais componentes de comunicação do SP e BT, denominados Shift e AllShift, da espécie sincronizador.

Shift

O componente Shift é utilizado para encapsular o padrão de comunicação primitivo entre os processos do SP e do BT, o qual trata-se de um deslocamento em sentido único de dados através do anel de processos em uma dos eixos possíveis (x, y ou z). Esse componente encapsula portanto sequências de chamadas a subrotinas do MPI.NET. O sentido do deslocamento de dados é controlada pelo parâmetro de contexto *direction*, o qual é limitado pelo componente abstrato Direction da espécie qualificador. Os subtipos de Direction, denominados Backward e Forward, indicam o sentido do deslocamento de dados. Em sua configuração, Shift contém um componente aninhado *topology* do tipo Ring, o qual define os vizinhos anterior e posterior do processo.

Foram desenvolvidos dois componentes concretos para Shift: ShiftBackwardImpl, para o qual *direction*=Backward, e ShiftForwardImpl, para o qual *direction*=Forward. Na codificação desses componentes, os identificadores dos processos anterior e posterior tem papéis diferentes. Por exemplo, se, no primeiro, o vizinho posterior é sempre o destino de mensagens de envio e o vizinho anterior é sempre a fonte de mensagens de recebimento, os papéis são invertidos no último.

A configuração de Shift especifica ainda os componentes aninhados *input_buffer* e *output_buffer*, do tipo Buffer da espécie estrutura de dados, os quais representam os *buffers* onde os dados recebidos e que serão enviados são respectivamente armazenados.

Os métodos do componente Shift podem ser vistos na Tabela 4.3, e a arquitetura do componente abstrato Shift pode ser vista na Figura 4.8, onde nota-se que somente os componentes *topology*, *input_buffer* e *output_buffer* são públicos.

AllShift

O componente AllShift determina uma estrutura complexa de comunicação, por tratar-se de uma composição de seis componentes aninhados do tipo Shift, denominados *shift_x_west*, *shift_x_east*, *shift_y_north*, *shift_y_south*,

Métodos Shift	SHIFT[*direction*=FORWARD]	SHIFT[*direction*=BACKWARD]
	Funções MPI.NET implementadas dentro do método	
initialize_send()	send(output_Buffer.Array, successor)	send(output_Buffer.Array, predecessor)
initialize_recv()	receive(input_Buffer.Array, predecessor)	receive(input_Buffer.Array, sucessor)

Tabela 4.3: Métodos do componente SHIFT.

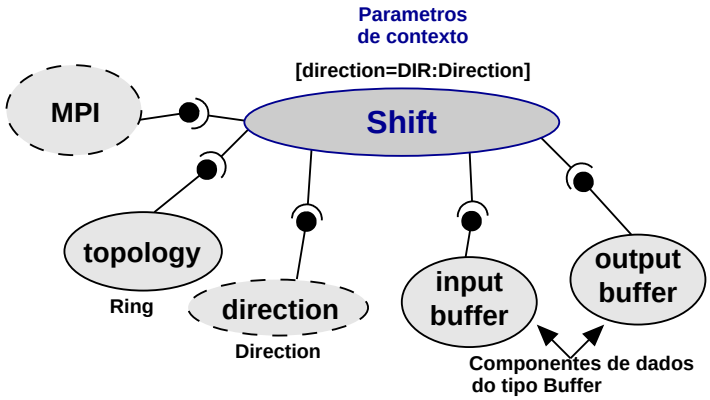


Figura 4.8: Arquitetura do componente abstrato SHIFT.

shift_z_top, *shift_z_bottom*. Como sugerido pelos seus nomes, são usados para definir um deslocamento paralelo de dados independentes para os dois sentidos possíveis ("para frente" e "para trás") nos eixos x, y e z. Para isso, os componentes aninhados SHIFT referentes ao mesmo eixo, para os dois sentidos possíveis, tem seus componentes aninhados *topology* compartilhados e renomeados conforme o nome do eixo (x, y ou z). Além disso, os valores reais associados aos parâmetros de contexto *direction* dos componentes aninhados do tipo SHIFT devem determinar o sentido da sua comunicação. Por exemplo, para os deslocamentos no eixo x, de responsabilidade dos componentes *shift_x_west* e *shift_x_east*, os parâmetros de contexto *direction* devem ser respectivamente associados aos componentes abstratos BACKWARD e FORWARD.

Os métodos definidos pela interface das unidades de SHIFTALL são os seguintes:

- **initiate_send_top()** e **initiate_recv_top()**: iniciando operações de comunicação assíncrona no sentido "para trás" do eixo z;
- **initiate_send_bottom()** e **initiate_recv_bottom()**: iniciando operações de comunicação assíncrona no sentido "para frente" do eixo z;
- **initiate_send_north()** e **initiate_recv_north()**: iniciando operações de

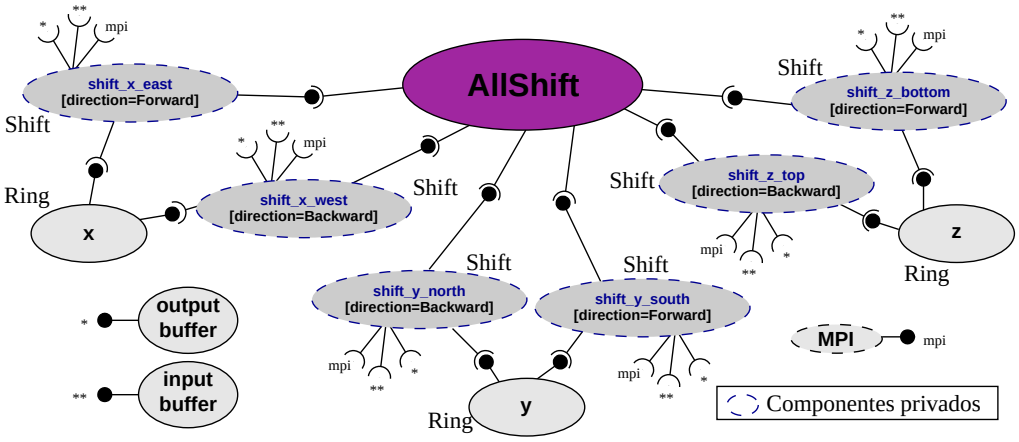


Figura 4.9: Arquitetura do componente abstrato AllShift.

comunicação assíncrona no sentido "para trás" do eixo y;

- **initiate_send_south()** e **initiate_recv_south()**: iniciando operações de comunicação assíncrona no sentido "para frente" do eixo y;
- **initiate_send_west()** e **initiate_recv_west()**: iniciando operações de comunicação assíncrona no sentido "para trás" do eixo x;
- **initiate_send_east()** e **initiate_recv_east()**: iniciando operações de comunicação assíncrona no sentido "para frente" do eixo x.

Uma chamada ao método *go* do componente AllShift causa o aguardo de que todas as operações iniciadas se completem. Porém, conforme o protocolo de chamadas estabelecido para a interface desse componente, é necessário que todas as operações tenham sido iniciadas uma única vez.

O componente de SP e BT que utiliza o AllShift é o CopyFaces, o qual realiza a cópia dos dados do domínio contidos nas faces compartilhadas entre as células adjacentes. A arquitetura do AllShift é apresentada na Figura 4.9.

4.4.6 Os componentes Class e Instance: Classes de Problema

Class é um componente abstrato da espécie qualificador, relacionado às classes de problema, um conceito herdado da versão Fortran, onde classes são informadas em tempo de compilação. A classe de problema está relacionada ao tamanho de problema aplicado ao *benchmark*, o qual determinará a quantidade do esforço computacional executado.

Instance_SP_S_Impl [class=Class_S]	Instance_SP_W_Impl [class=Class_W]
Instance_SP_A_Impl [class=Class_A]	Instance_SP_B_Impl [class=Class_B]
Instance_SP_C_Impl [class=Class_C]	Instance_BT_S_Impl [class=Class_S]
Instance_BT_W_Impl [class=Class_W]	Instance_BT_A_Impl [class=Class_A]
Instance_BT_B_Impl [class=Class_B]	Instance_BT_C_Impl [class=Class_C]

Figura 4.10: Componentes concretos que implementam subtipos de Instance

Há cinco subtipos de Class, cada qual associado a uma das classes de problema determinados pelo NPB: Class_S (S), Class_W (W), Class_A (A), Class_B (B) e Class_C (C). Uma vez que os componentes de SP e BT são genéricos com relação a classe de problema, ou seja, podem executar qualquer classe de problema, esses componentes são utilizados para suprir os parâmetros de contexto de ADISolver3D na instanciação de uma aplicação, determinando a classe de problema a ser aplicada.

Para cada programa do NPB, há um conjunto de parâmetros que determinam o tamanho do problema a ser executado, os quais são encapsulados por subtipos do componente abstrato Instance para cada programa, tais como Instance_SP e Instance_BT no caso de SP e BT, respectivamente. Os componentes concretos desses componentes abstratos são responsáveis por configurar os parâmetros de problema utilizados ao longo da execução do aplicativo como, por exemplo, tamanho das matrizes que representam o domínio, número de iterações, valores de referência para verificação de resultados (Subseção 4.4.8), constantes utilizadas para controlar iterações nos eixos x, y e z, etc. Uma vez que os valores desses parâmetros são constantes que dependem da classe de problema, Instance possui o parâmetro de contexto *class*, limitado por Class. Assim, há versões de componentes concretos de Instance_SP e Instance_BT para cada classe de problema, associando-se o parâmetro de contexto *class* com um dos subtipos possíveis de Class. No caso de SP e BT, as implementações resultam em dez componentes concretos atendendo às cinco classes de problema, como ilustrado na Figura 4.10, onde destacamos a nomenclatura utilizada em cada versão concreta de Instance_SP e Instance_BT, bem como seus parâmetros suprimidos.

LU e FT utilizam dos mesmos critérios com componentes de classes e instâncias, sendo as diferenças definidas pelos valores constantes da implementação.

4.4.7 O componente ProblemData: Estruturas de Dados

Em sua versão monolítica, escrita em Fortran, as estruturas de dados usadas nos programas do NPB são definidas por várias matrizes multidimensionais no escopo global. Essas matrizes definem o estado inicial do problema e armazenam cálculos efetuados pelos processos ao longo da execução. Na refatoração em componentes, as estruturas de dados foram encapsuladas em um único componente aninhado de ADISolver3D, denominado *problem_ data*, de tipo determinado pelo componente abstrato ProblemData da espécie estrutura de dados. Isso está de acordo com a premissa de manter a estruturação original da versão monolítica, tendo em vista o objetivo da avaliação de desempenho.

ProblemData é composto por componentes aninhados do tipo Field, cujos nomes, herdados da versão Fortran, são: *u*, *forcing*, *rhs*, *lhs*, *us*, *vs*, *ws*, *qs*, *rho*, *ainv*, *speed*, *square*, *lhsc*, *lhsb* e *lhsa*. O componente concreto de Field, denominado FieldImpl, constrói matrizes multidimensionais necessárias ao ProblemDataImpl, componente concreto de ProblemData. A matriz retornada é inicializada por FieldImpl a partir dos limites fornecidos por ProblemDataImpl, que utiliza o tamanho de problema obtido do componente *instance* para o seu cálculo, cujo tipo está vinculado ao parâmetro de contexto *instance_ type*. O encapsulamento da matriz multidimensional por componentes Field permite concentrar a modelagem da estrutura de dados em ProblemData, obedecendo à abordagem de componentes, mais próxima às práticas da engenharia de software moderna.

Todo componente que necessite de dados em matrizes multidimensionais têm internamente um componente aninhado *problem_ data*, os quais são compartilhados em configurações usuárias, até ADISolver3D.

A configuração de ProblemData é ilustrada na Figura 4.11. Observa-se que não há componentes públicos. Tendo em vista que a maioria das computações utiliza somente um pequeno subconjunto das matrizes, optou-se por oferecer o acesso a essas matrizes através de métodos da interface das unidades dos componentes *problem_ data*, evitando "poluir" as configurações de componentes computacionais com um número excessivo de componentes aninhados sem utilidade no seu contexto.

4.4.8 O componente Verify: Verificação de Resultados

Os programas do NPB possuem um procedimento para verificação dos resultados calculados, baseado na comparação com valores de referência definidos para cada classe de problema. Seu nome é *verify* e é executado logo após o cálculo da

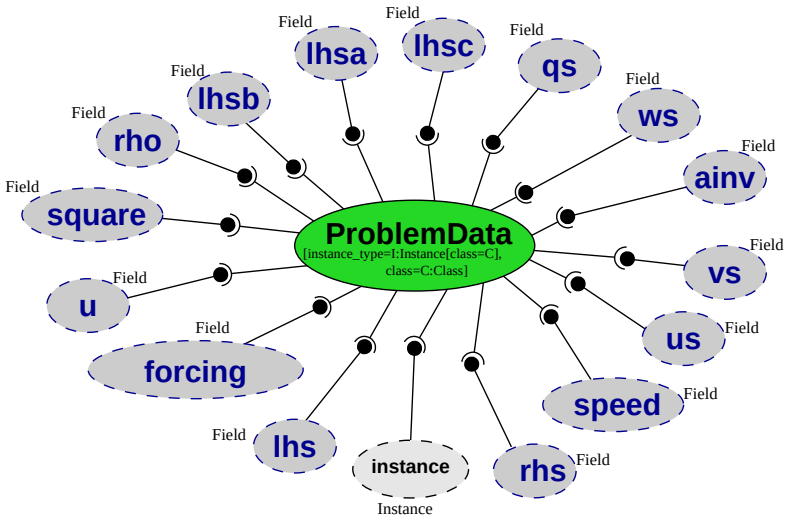


Figura 4.11: Arquitetura do componente abstrato ProblemData.

solução. Na refatoração em componentes, encapsulamos os testes de verificação no componente aninhado *verify* de ADISolver3D, cujo tipo é definido pelo componente abstrato Verify pertencente à espécie computação.

Verify possui os parâmetros de contextos *instance* e *class*, cujo significado foi anteriormente explicado, uma vez que precisa conhecer os valores reais dos parâmetros especificados pela classe de problema em execução. Os valores de referência para cada classe de problema estão definidos como constantes pelos componente concretos dos subtipos de Instance. No caso de SP e BT, recorde-se que são aqueles apresentados na Figura 4.10.

A configuração do componente Verify encontra-se ilustrada na Figura 4.12, onde pode-se observar que possui os componentes aninhados *error_norm*, do tipo ErrorNorm, e *rhs_norm*, do tipo RHSNorm, instanciados na execução pelos componentes concretos ErrorNormImpl e RHSNormImpl, respectivamente, responsáveis por calcular a diferença entre a solução calculada e a solução de referência através do cálculo de normas vetoriais. Outro componente aninhado também presente em Verify é o *copy_faces*, do tipo CopyFaces já mencionado por ocasião da descrição do componente ADI. Vale a pena observar que a instância de CopyFaces (*copy_faces*) de Verify é privada. Portanto, não é compartilhada com a instância privada de CopyFaces presente no componente *adi* de ADISolver3D. Isso não é essencial, tendo em vista que CopyFaces não é um componente com estado, o qual deveria ser mantido por uma única instância, como são os casos dos componentes *problem_data* e *cells_info*.

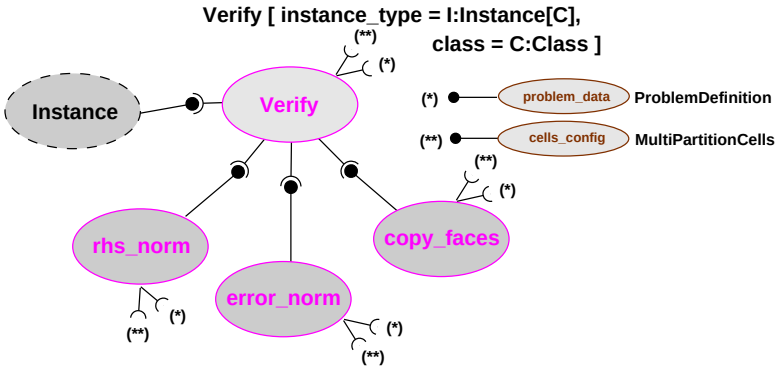


Figura 4.12: Arquitetura do componente abstrato VERIFY.

Para o LU e o FT, a configuração do componente VERIFY difere pela ausência dos componentes *error_ norm*, *rhs_ norm* e *copy_ faces*.

4.4.9 O componente TIMER: Cronometragem de Execução

O componente abstrato TIMER tem o propósito de medir o tempo de cálculo da solução pelos programas do NPB. Em TimerImpl, componente concreto de TIMER, o tempo é medido separadamente por cada processo. Posteriormente, as medições são comparadas para identificar o tempo máximo gasto entre os processos, o qual determina o tempo definitivo da execução. Para isso, é utilizada a subrotina *MPI_Reduce* do MPI ao final da execução.

TIMER é um componente comum entre os programas do NPB. Além disso, pode ser facilmente reusado por programas fora do contexto do NPB, oferecendo o suporte a cronometragem do tempo gasto na execução de subrotinas específicas de um programa devidamente instrumentado com uso do componente TIMER.

TIMER é um componente primitivo, não possuindo componentes aninhados, o que torna dispensável a apresentação de sua configuração.

Na configuração de SP e de BT, a instância de TIMER é definida pelo componente aninhado *timer* de ADISOLVER3D.

4.5 LU

O LU é uma aplicação simulada do NPB que implementa uma solução simplificada para equações diferenciais *Navier-Stokes*, destinadas a descrever escoamento de fluidos [48, 65]. O método utilizado é o SSOR[6], onde é empregada a abordagem de triangulação superior e inferior sobre matrizes multidimensionais.

[6] *Symmetric Successive Over Relaxation.*

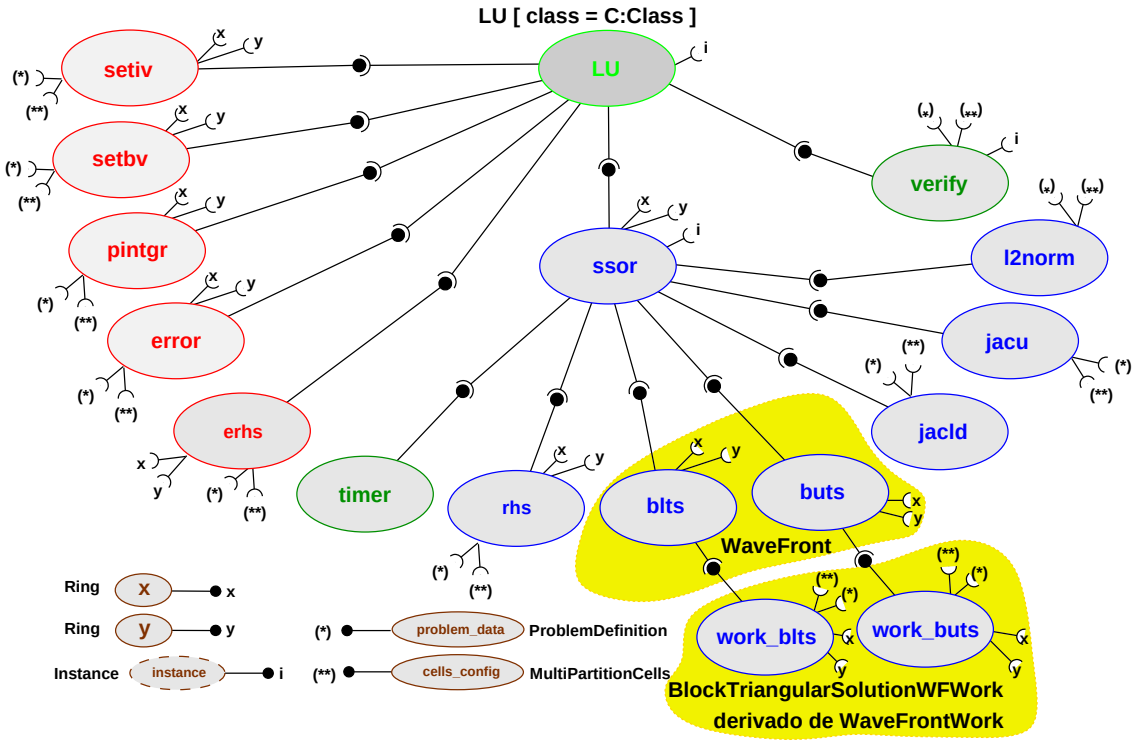


Figura 4.13: Arquitetura LU.

Assim como no SP e no BT, as matrizes (estruturas de dados) na versão refatorada em componentes do LU também são encapsuladas usando componentes FIELD, sendo as principais denominadas *a*, *b*, *c*, *d*, *u*, *rsd*, *frct*, *flux*. De fato, LU utiliza as mesmas estratégias de refatoração de interesses de configuração do paralelismo descritas para o SP e BT, sendo desnecessário repeti-las no contexto específico do LU. Alguns componentes são até compartilhados com SP e BT, como é possível observar na Tabela 4.1, tais como: CLASS, CLASS_A, CLASS_B, CLASS_C, CLASS_S, CLASS_W, INSTANCE, BUFFER, TIMER, DIRECTION, RING e MPIDIRECT.

De um modo geral, o LU é composto pelos componentes apresentados na Figura 4.13. Nessa figura, os componentes *setiv*, *setbv* e *erhs* definem valores iniciais para as matrizes de dados utilizadas pelo componente *ssor*, responsável pela computação propriamente dita da solução. Esses componentes têm tipos definidos pelos componentes abstratos SETIV, SETBV, ERHS e SSOR da espécie computação, respectivamente. Por sua vez, os componentes aninhados *pintgr* e *error*, repectivamente dos tipos PINTGR e ERROR, são componentes de apoio ao componente VERIFY. O componente do tipo PINTGR calcula o valor de uma integral

de superfície, enquanto o componente do tipo ERROR calcula os valores do vetor *errnm*, com 5 elementos representando produtos escalares. O principal componente computacional do LU é o SSOR. Por esse motivo, é destacado na próxima seção, especialmente no que diz respeito a sua estratégia de comunicação peculiar.

4.5.1 O componente SSOR

O componente SSOR é o componente computacionalmente intensivo do LU. Por isso, o componente aninhado *timer*, do tipo TIMER, está incluído em sua configuração. Além de *timer*, SSOR é composto por seis componentes aninhados de cada um dos componentes abstratos da espécie computação enumerados a seguir: RHS, BLTS, BUTS, JACLD, JACU e L2NORM.

JACLD calcula a parte triangular inferior de uma Matriz *Jacobiana*, enquanto JACU calcula a parte superior. RHS e L2NORM efetuam computações com o vetor denominado *rsd*, sendo definidos valores de normas vetoriais. Entretanto, a principal estratégia de comunicação é encapsulada pelos componentes dos tipos BLTS (*Block Lower Triangular Solver*) e BUTS (*Block Upper Triangular Solver*).

BLTS e BUTS efetuam computações com blocos de matrizes, onde as matrizes são divididas entre processos que se organizam pelo padrão de comunicação conhecido como *wavefront* [57].

O padrão de comunicação *wavefront* é definido em uma topologia de processos em duas dimensões, na qual cada processo possui vizinhos nos eixos x, com sentidos oeste (anterior) e leste (posterior), e y, com sentidos norte (anterior) e sul (posterior), formando uma malha. Cada processo executa os seguintes passos: receber dados dos processos nos sentidos anteriores de cada eixo; executar uma computação que depende dos dados recebidos; e enviar dados para os processos nos sentidos posteriores de cada eixo, calculados como efeito direto ou colateral da computação realizada no passo anterior. Processos que estão localizados nas extremidades da malha, os quais não possuem vizinhos em alguns dos sentidos, ignoram as operações de comunicação referentes a esses vizinhos ausentes. A Figura 4.14 descreve o padrão de comunicação resultante dessa abordagem. Observe que os dados trafegam através da malha formando uma "onda", o que justifica o nome atribuído ao padrão. Na direção da diagonal principal da malha, a onda pode seguir o sentido do processo superior para o processo inferior (progressiva), como é o caso da descrição dos passos apresentada, ou vice-versa (regressiva), bastando inverter a definição dos processos posteriores e anteriores. O padrão de comunicação *wavefront* é bastante difundido

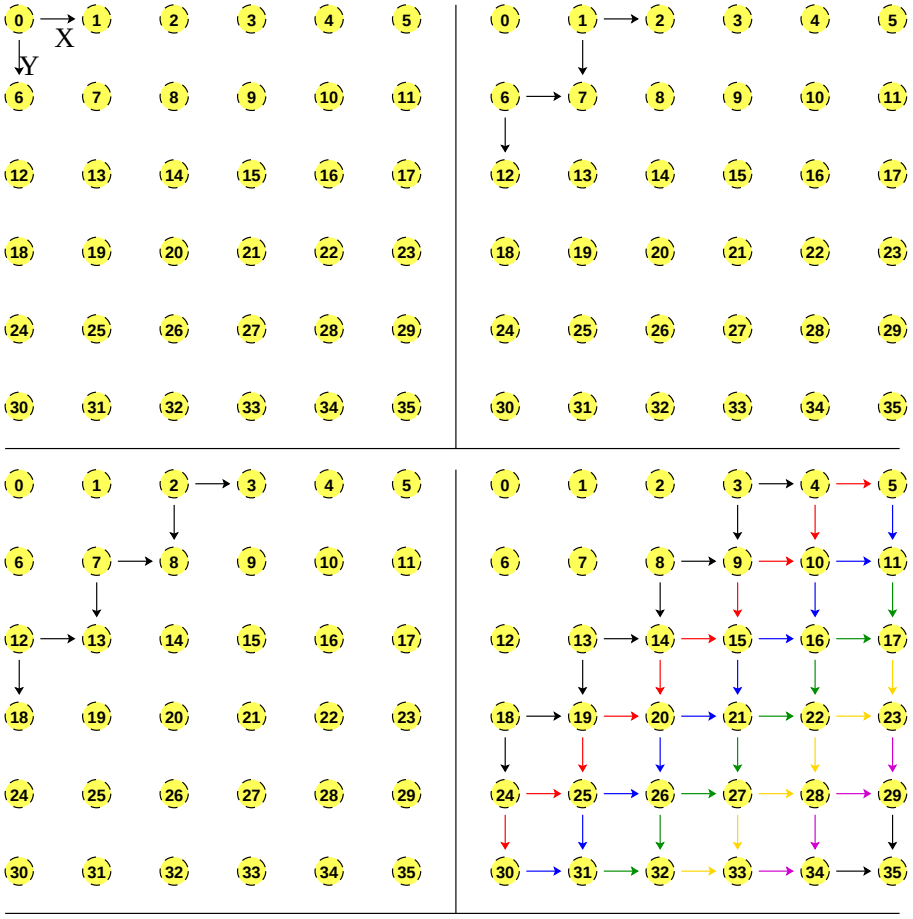


Figura 4.14: Comunicação WaveFront.

em computação paralela, especialmente no projeto de algoritmos em alto nível, oferecendo uma oportunidade interessante para criação de um componente reusável na forma de um *esqueleto* de computação paralela.

Esqueletos de algoritmos constituem uma técnica de programação paralela proposta por Murray Cole no final da década de 1980 [35]. A pesquisa com padrões de computação paralela encapsulada em esqueletos foi bastante difundida nas últimas duas décadas, embora a técnica em si não tenha atingido a disseminação esperada nas aplicações reais [36, 53]. No entanto, é vista como uma técnica promissora para programação paralela estruturada e de alto nível. Nesse trabalho, demonstra-se como esqueletos podem ser utilizados no contexto de componentes paralelos, já que descrevem padrões de computação paralela reusáveis, constituindo uma importante técnica de programação do HPE.

No LU, o padrão *wavefront* é representado pelo componente abstrato WaveFront, cuja configuração está ilustrada na Figura 4.15, o qual define os tipos

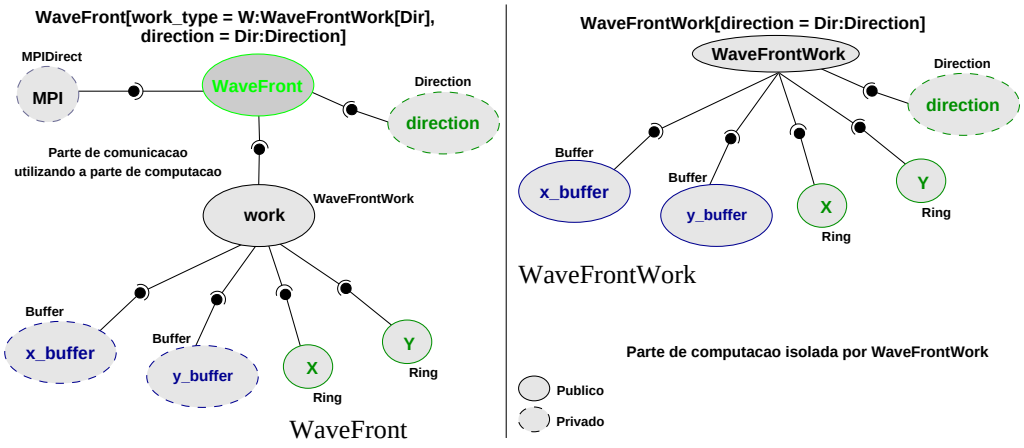


Figura 4.15: Componentes WAVEFRONT e WAVEFRONTWORK.

dos componentes aninhados *blts* e *buts* de SSOR. WAVEFRONT tem os parâmetros de contexto *direction* e *work_type*. O primeiro é limitado pelo componente abstrato DIRECTION, representando o sentido da "onda", a qual pode ser progressiva (*forward*) ou regressiva (*backward*). Para essa finalidade, aplicam-se os componentes abstratos FORWARD e BACKWARD, subtipos de DIRECTION, os quais também foram usados no SP e no BT. O último é limitado pelo componente abstrato WAVEFRONTWORK, também ilustrado na Figura 4.15, e representa a computação realizada por cada processo na malha. Para isso, há um componente aninhado chamado *work* em WAVEFRONT, cujo tipo é definido pelo valor de *work_type*. Dessa forma, podemos instanciar o componente WAVEFRONT para diferentes computações paralelas possíveis baseadas nesse padrão de comunicação, sendo suficiente fazer a escolha apropriada de um subtipo de WAVEFRONTWORK e atribuí-la a *work_type*. Para as necessidades particulares de *blts* e *buts*, foi desenvolvido o componente abstrato BLOCKTRIANGULARSOLUTIONWFWORK, para o qual há dois componentes concretos: BlockLowerTriangularSolutionWFWorkImpl e BlockUpperTriangularSolutionWFWorkImpl. Eles diferem pelo valor aplicado ao parâmetro de contexto *direction*, respectivamente FORWARD e BACKWARD. De fato, nos tipos de *blts* e *buts*, são respectivamente esses os tipos aplicados ao parâmetro de contexto *direction* de WAVEFRONT.

A utilização do componente BLOCKTRIANGULARSOLUTIONWFWORK, derivado de WAVEFRONTWORK, pode ser vista na Figura 4.13, representando a arquitetura LU.

Nas configurações dos componentes WAVEFRONT e WAVEFRONTWORK

apresentadas na Figura 4.15, observa-se que WAVEFRONTWORK possui os componentes aninhados *x_ buffer*, *y_ buffer*, *x* e *y*. Os dois primeiros são do tipo BUFFER e prestam-se a armazenar os dados recebidos e a serem enviados pelos processos aos seus vizinhos nos eixos x e y, respectivamente. Os dois últimos, do tipo RING, servem para que *work* conheça a posição relativa do processo na malha, especialmente no que diz respeito a sua localização em área interna ou em uma das extremidades da malha de processos.

Os componentes concretos de WAVEFRONT são WavefrontForwardImpl e WavefrontBackwardImpl, cujos parâmetros de contexto *direction* são respectivamente supridos por FORWARD e BACKWARD. Portanto, referem-se aos sentidos possíveis da comunicação no padrão *wavefront*. Porém, são genéricos com respeito ao parâmetro *work_ type*, sendo supridos com o limite WAVEFRONTWORK, de forma a serem aplicáveis a qualquer computação definida por herança a partir do componente WAVEFRONTWORK.

4.6 FT

O FT implementa o algoritmo FFT[7] sob números complexos. O cálculo é realizado no espaço tridimensional, nos eixos x, y e z. As estruturas de dados são matrizes multidimensionais de números complexos. As principais matrizes são denominadas *u*, *u0*, *u1* e *u2*. Assim como SP, BT e LU, também são encapsuladas pelo componente FIELD. Como já discutido com relação ao LU, as estratégias de refatoração em componentes paralelos usadas no FT são as mesmas aplicadas a os outros programas já descritos, sendo redundante apresentar detalhes. Portanto, esta seção concentra-se em aspectos peculiares da configuração do FT.

Ainda com relação às estruturas de dados, as matrizes são declaradas como do tipo `COMPLEX` na versão Fortran, suportado por essa linguagem. Porém, nas versões C#, incluindo a versão refatorada em componentes, os números complexos são simulados separando-se as partes real e imaginária em dimensões diferentes das matrizes, pois C# não suporta números complexos. Essa estratégia de implementação de números complexos também é adotada na versão Java do NPB, construída pela equipe do NAS. Assim, uma matriz de n valores do tipo `COMPLEX` no Fortran é representada por $2 \times n$ valores do tipo ponto flutuante de precisão dupla (`double`) em C# e Java, potencialmente gerando maior sobrecarga na verificação de limites das matrizes.

[7]Sigla em inglês de Transformada Rápida de Fourier

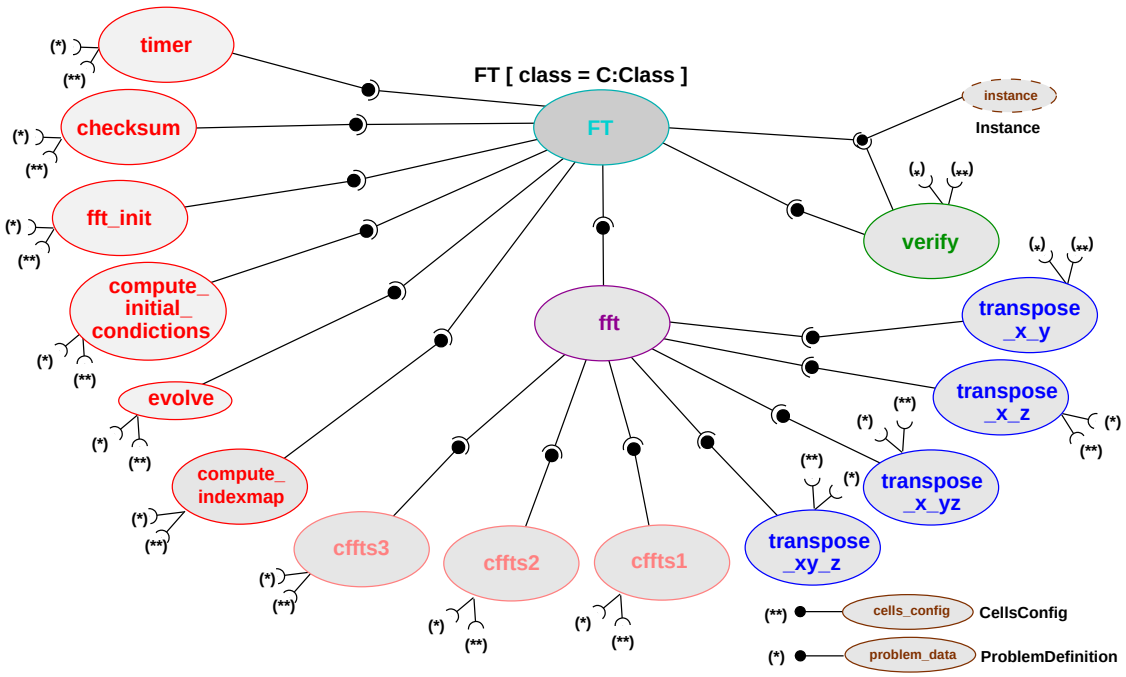


Figura 4.16: Arquitetura FT.

Outra importante peculiaridade do FT é a dinâmica de como são acessados os dados das matrizes multidimensionais, pois nem sempre as dimensões de uma matriz global são as mesmas dentro de diferentes procedimentos, com alteração da forma de acesso à área de memória ocupada pela matriz. Isso é uma característica suportada pela linguagem Fortran. Tal característica dificulta a implementação do FT em linguagens seguras, como Java e C#, em especial tendo em vista a premissa dessa refatoração de preservar as computações e estruturas de dados da versão original em Fortran. Na implementação C#, foram necessários ajustes de indexação, evitando o uso de variáveis do tipo ponteiro para manipular a memória.

É conveniente ressaltar que as sobrecargas resultantes dos ajustes de indexação necessários não se trata de uma sobrecarga intrínseca ao uso de componentes, mas relacionada ao projeto de linguagens de programação de execução segura, como Java e C#, as quais não foram construídas tendo em vista muitos requisitos importantes para programas científicos que são tratados pela linguagem Fortran. De fato, o processo de implementação e avaliação de desempenho de versões C# e Java dos programas NPB poderá oferecer subsídios para propostas de extensões sobre essas linguagens para melhor suporte a programas dessa natureza.

A comunicação entre os processos no FT é baseada em operações de comunicação coletiva do MPI. Na versão baseada em componentes, foram utilizados

os comunicadores MPI dos próprios componentes, bem como o componente MPIDIRECT, responsável por encapsular recursos fundamentais do MPI.NET. O principal componente do FT que utiliza comunicação coletiva é o TRANSPOSE, apresentado na Subseção 4.6.1.

Assim como o LU, FT também compartilha componentes com o SP e com o BT. Na Figura 4.16, são ilustrados os componentes que formam a sua configuração. Dentre eles, *fft_init*, *compute_initial_condictions*, *compute_index_map* e *evolve*, da espécie computação, definem configurações iniciais das estruturas de dados. Por sua vez, *checksum* faz uma soma global de valores calculados a partir do componente *fft* de cada processo. O resultado da soma é armazenado em um vetor denominado *sum*, passado por parâmetro ao componente *verify* para verificação dos resultados calculados. Dentre esses, o componente mais significativo é o *fft*, cujo tipo é definido pelo componente abstrato FFT.

4.6.1 O componente abstrato FFT

O FFT é um componente abstrato, da espécie computação, que encapsula o algoritmo *fft* para calcular a transformada rápida de Fourier. Tem como parâmetros de contexto *instance_type* e *class*, limitados por INSTANCE e CLASS respectivamente. Seus componentes internos são basicamente de dois tipos: CFFTS (*cffts1*, *cffts2* e *cffts3*) e TRANSPOSE (*transpose_x_yz*, *transpose_x_z*, *transpose_xy_z* e *transpose_x_z*).

CFFTS é responsável por efetuar computações que são armazenadas nas matrizes *u0*, *u1* e *u2*. Possui o parâmetro de contexto *axis*, limitado pelo componente AXIS da espécie qualificador. Os componentes concretos de CFFTS, para cada dimensão do domínio tridimensional, são três: Cffts1Impl, com o parâmetro de contexto *axis* suprido por XAXIS; Cffts2Impl, com *axis* suprido por YAXIS; e Cffts3Impl, com *axis* suprido por ZAXIS. Respectivamente são instanciados para os componentes aninhados *cffts1*, *cffts2* e *cffts3* de FFT.

4.6.2 O componente abstrato TRANSPOSE

TRANSPOSE é o principal componente de comunicação do FT. Assim como o CFFTS, possui o parâmetro de contexto *direction*, para especificar em que eixo (x, y ou z) a operação de comunicação se aplica.

O componente TRANSPOSE possui componentes aninhados dos tipos TRANSPOSELOCAL, TRANSPOSEGLOBAL, TRANSPOSEFINISH, respectivamente denominados *transpose_local*, *transpose_global* e *transpose_finish*. O primeiro

calcula valores locais dos processos. O segundo efetua uma operação coletiva de transposição de dados, implementada no MPI pela subrotina `MPI_Alltoall`. Por fim, o terceiro recalcula e armazena dados em matrizes, para permitir subsequentes operações de transposição em outras dimensões.

Vale ressaltar que o componente TRANSPOSE seria outro candidato natural para implementação como um esqueleto de computação paralela, abstraindo-se de alguns aspectos específicos de sua implementação para o FT, assim como foi realizado no caso do padrão WAVEFRONT para o LU. Entretanto, isso não foi implementado nesse trabalho, devendo ser implementado em futuras versões.

4.7 Discussão

Para o processo de refatoração em componentes, foi necessário um estudo aprofundado sobre os interesses de software que constituem os programas escolhidos do NPB, os quais foram discutidos neste capítulo. O estudo permitiu explorar o potencial de modularização e reuso de partes do NPB, de forma a demonstrar também o potencial de desenvolvimento de componentes da plataforma HPE a partir de códigos nativos pré-existentes.

Vários componentes identificados são potencialmente reusáveis, muitos dos quais são efetivamente compartilhados entre SP, BT, LU e FT. A existência desses componentes em bibliotecas oferece ao desenvolvedor a possibilidade de estruturar aplicações científicas de forma mais rápida e eficaz, permitindo o desenvolvimento em larga escala, o que não é oferecido pelas práticas convencionais da programação paralela.

O isolamento de interesses não-funcionais em componentes distinguíveis por suas espécies, especialmente *ambientes* e *qualificadores*, é uma importante capacidade de sistemas de programação #, como o HPE, o qual é extensivamente explorado na decomposição em componentes apresentada neste capítulo.

No desenvolvimento de programas paralelos, os interesses de configuração do paralelismo, como o particionamento e distribuição de dados, a topologia de comunicação entre processos e os padrões de interação entre esses estão explicitamente presentes no código de cada processo, exigindo um tratamento baseado em componentes nem sempre suportado pela maioria das ferramentas voltadas à programação paralela, as quais geralmente enxergam processos como elementos isolados de software.

Uma abordagem bastante aplicada no desenvolvimento de ferramentas de

programação paralela baseia-se na idéia de esconder os interesses de configuração do paralelismo entre processos através de mecanismos de abstração, oferecendo mecanismos implícitos ou semi-explícitos que possuem geralmente efeito negativo sobre a expressividade e desempenho, tendo em vista o menor controle dado ao programador sobre os recursos de processamento e memória e para definição de padrões de interação entre processos, especialmente os não regulares. Ao propôr o foco em mecanismos de modularização, ao invés de abstração, busca-se tornar o problema de configuração do paralelismo mais facilmente tratável, tendo em vista ser o desenvolvedor o sujeito capaz de explorar de forma mais eficiente as características da plataforma de execução paralela para implementar os requisitos da aplicação da forma mais eficaz e eficiente. Nesse contexto, vale ressaltar o maior grau de complexidade de plataformas de computação aplicadas a computação de alto desempenho e a crescente complexidade do software nos seus domínios de interesse.

Capítulo 5

Avaliação de Desempenho do HPE

Este capítulo tem o propósito de aplicar conceitos sobre avaliação de desempenho que foram abordados no Capítulo 3 a fim de realizar a avaliação de desempenho da plataforma de componentes HPE. A metodologia de avaliação aplicada é baseada na técnica de medição, através de *benchmarking*.

O capítulo está dividido conforme três assuntos considerados relevantes para a avaliação de desempenho de sistemas. Na Seção 5.1, é definido como serão aplicados os recursos matemáticos e estatísticos disponíveis para avaliação de desempenho do HPE. Na Seção 5.2, são detalhados os recursos utilizados para realizar os experimentos com o NPB a fim de avaliar o desempenho do HPE, assim definindo as características e propriedades da plataforma experimental. Na Seção 5.3, são apresentados os dados obtidos a partir dos experimentos e as conclusões tomadas a partir de sua análise. Para isso, essa seção é dividida em análise dos intervalos de confiança para cada experimento e análise de experimentos fatoriais, conforme a metodologia adotada.

5.1 Metodologia

A metodologia adotada consiste na comparação direta de medições de desempenho (tempo de execução) entre a versão monolítica (C#/MPI.NET) do NPB e sua versão refatorada em componentes para o HPE. A versão refatorada conserva a originalidade da monolítica, pois foi desenvolvida sem alterações significativas nos códigos que declaram e inicializam as estruturas de dados, bem como os que descrevem computações, topologia de processos e comunicação entre os processos. Assim, pretende-se isolar e mensurar o peso ou o efeito da refatoração sobre o desempenho de programas realísticos, já que o NPB é um código profissionalmente

desenvolvido por programadores especialistas nas aplicações e experientes em programação paralela e simulação numérica com uso de computadores.

São utilizadas as técnicas de avaliação de desempenho abordadas no Capítulo 3. O processo de avaliação é feito pela análise de um conjunto de quatro fatores, identificados por P, C, I e B, doravante denominado PCIB.

O fator P refere-se ao número de processadores utilizados nos experimentos, e seus níveis são p_1, p_2, p_4, p_8, p_9, e p_{16}, respectivamente associados a 1, 2, 4, 8, 9 e 16 processadores. O fator C refere-se a classe de problema, a qual define a carga de trabalho a ser aplicada sobre cada experimento, englobando os níveis S, W e A. O fator I diz respeito ao tipo de implementação, com níveis definidos por M (versão monolítica) e R (versão refatorada em componentes). O fator B refere-se à aplicação em si, sendo seus níveis definidos por SP, BT, LU e FT.

Os fatores PCIB são utilizados para a avaliação de desempenho, o qual é baseado em comparações entre intervalos de confiança para a média das medições e no estudo dos efeitos dos seus níveis através de experimentos fatoriais.

Para definir os fatores de um experimento e seus conjuntos de níveis aplicados, utiliza-se expressões do tipo $X[x_1, ..., x_n]$, onde X é um dos fatores PCIB, e $x_1, ..., x_n$ o conjunto de níveis utilizado por X. Fatores e níveis podem então ser combinados por meio de expressões tais como $B[SP, BT] \times C[S, W]$, designando os experimentos com as aplicações SP e BT sobre as classes de problema S e W.

5.1.1 Definições da Avaliação de Desempenho

O estudo de desempenho é feito a partir da técnica de medição, combinando-se níveis entre fatores a fim de definir vários experimentos. Ao todo são 88 experimentos, dispostos por duas combinações de níveis de fatores. A primeira combinação refere-se às aplicações SP e BT, definida por $I[M, R] \times B[SP, BT] \times C[S, W, A] \times P[p_1, p_4, p_9, p_{16}]$, resultando em 48 experimentos. A segunda combinação refere-se às aplicações LU e FT, definida por $I[M, R] \times B[LU, FT] \times C[W, A] \times P[p_1, p_2, p_4, p_8, p_{16}]$, resultando em quarenta experimentos. A separação SP/BT e LU/FT deve-se às restrições das aplicações em relação ao número de processadores utilizados e classes de problemas suportadas. SP e BT exigem um número quadrado de processadores (n^2), enquanto LU e FT requerem um número potência de dois (2^n). Além disso, LU e FT não suportam a carga S já a partir de 4 processadores, devido a impossibilidade de particionar a estrutura de dados entre vários processadores.

Cada experimento é realizado pela repetição de 40 medições de tempo

de computação, caracterizando uma variável aleatória cuja média representa a tendência central para o tempo de execução de um experimento. Assim, cada medição é também denominada replicação. A literatura estabelece que ao menos trinta replicações são necessárias para as técnicas descritas/utilizadas nas Seções subsequentes 5.1.1 e 5.1.1. O tempo de execução nos experimentos é medido em segundos e caracteriza uma métrica *BM* (Baixo Melhor). A medição é dada por y_{ijkmn}, onde i, j, k e m representam respectivamente níveis dos fatores PCIB que caracterizam o experimento, e n representa a repetição, tal que $1 \leq n \leq r$, com $r = 40 - o$, onde o representa o número de *outliers* do experimento.

Um *outlier* caracteriza um valor discrepante, definido por uma medição com tempo de resposta baixo ou elevado demais em comparação às respostas de outras medições, provavelmente devido a variações inesperadas em um ou mais nós do *cluster* durante uma execução particular. Neste experimento, são eliminados os *outliers* relativos a valores elevados. A identificação e eliminação de *outliers* é feita após o término das quarenta replicações, considerando todas as medições obtidas, com a principal finalidade de aproximar os dados amostrais a uma distribuição com tendências normais. Com as medições realizadas, são empregadas duas técnicas estatísticas que identificam diferenças de desempenho entre as versões R e M, definidas em seguida.

Doravante, as versões refatoradas em componentes para o HPE serão chamadas de versões R, enquanto as versões monolíticas serão chamadas de versões M, utilizando portanto a notação aplicada aos níveis do fator I.

Comparação entre Versões R e M

A comparação entre duas versões é realizada por meio de intervalos de confiança (IC), cuja definição encontra-se explicada na Seção 3.5.2. Na comparação de versões R e M por IC de amostras, são aproveitadas as definições dos fatores e níveis que serão empregados nos experimentos fatoriais, o que nos permite discutir seus resultados de forma conjunta. São adotadas duas formas para comparar versões:

- *análise com Teste-t*, definindo a rejeição ou aceitação da hipótese nula de diferenças entre as versões R e M; e
- *análise de sobreposição dos intervalos*, respectivamente associados às versões R e M, verificando se os intervalos se sobrepõem.

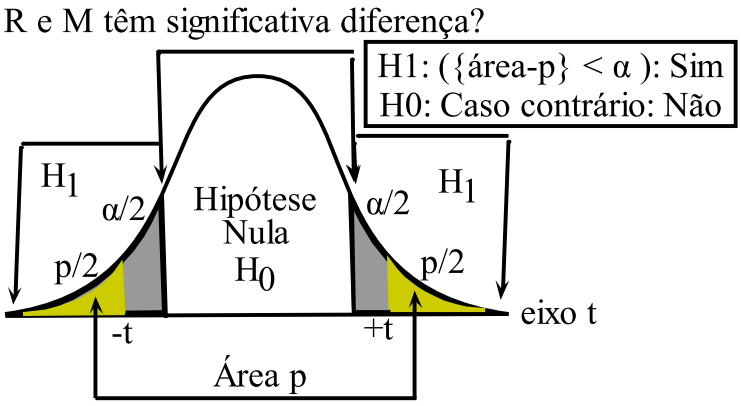


Figura 5.1: *Teste-t.*

Para a análise por meio do *Teste-t*, são levantadas as seguintes hipóteses: H_0 : $R = M$ para hipótese nula; e H_1 : $R \neq M$ para hipótese alternativa. Para as hipóteses, são adotadas as definições apresentadas na Seção 3.5.2 para *Teste-t*. Nesse contexto, calcula-se o parâmetro *valor-t*, possibilitando que a função densidade da distribuição-t encontre a probabilidade que satisfaça H_1, representada pela área p na Figura 5.1 em duas extremidades (bicaudal). Caso área p seja menor que o nível de significância α, os sistemas são significativamente diferentes (H_1). Caso contrário, são iguais (H_0).

O *Teste-t* encontra-se incluído nas tabelas de resultados. É representado pelo valor da área p, calculada a partir de amostras não pareadas com variâncias distintas. Como são 88 experimentos, distribuídos entre M e R, há 44 hipóteses testadas através do *Teste-t*. O nível de significância usado é $\alpha = 0,05$, representando 95% de confiança. Quando *Teste-t* $< \alpha$, a chamada *hipótese alternativa* é aceita, o que significa que os sistemas apresentam desempenhos diferentes segundo o *Teste-t*. Caso contrário, os sistemas podem ser considerados iguais.

Para *análise de sobreposição dos intervalos*, calcula-se o intervalo de confiança (IC) de cada versão em uma combinação de níveis, definido por $\overrightarrow{IC[R]}\{niveis\}[r_0, r_1]$ e $\overrightarrow{IC[M]}\{niveis\}[m_0, m_1]$, para *niveis* sendo os níveis envolvidos igualmente para ambas as versões R e M. Cada intervalo tem nível de confiança de 95%. Após definidos os *ICs*, desenvolve-se um gráfico de *ICs*, analisando se os intervalos se sobrepõem e incluem a média de um intervalo no outro. Os sistemas não serão significativamente diferentes se existe sobreposição. Por exemplo, na Figura 5.2 há as seguintes possibilidades: em A os sistemas são diferentes, com maior desempenho em $SIS1$, pois os intervalos não se sobrepõem e o intervalo de $SIS1$ é menor que o

intervalo de $SIS2$; em B acontece o fenômeno de sobreposição, mas não ao ponto de incluir a média de um intervalo no outro, o que sugere uma verificação também pelo *Teste-t*; em C, apesar de $SIS1$ obter um intervalo de tempo inferior a $SIS2$, considera-se não haver diferença significativa entre os sistemas, porque os intervalos estão sobrepostos e incluem a média de um IC no outro.

As sobrecargas são calculadas com base nos intervalos de confiança. Para isso, é usado o seguinte cálculo: sejam $[r_0, r_1]$ e $[m_0, m_1]$ dois ICs para R e M respectivamente. A sobrecarga mínima/máxima é dada por: $(SobrecargaMin = r_0 - m_1)$; e $(SobrecargaMax = r_1 - m_0)$. Nas tabelas de resultados, essas sobrecargas são apresentadas em porcentagens. Assim, são divididas pela média $\bar{m}$ da versão M, para $\bar{m} = (m_0 + m_1)/2$. Como resultado, tem-se $SobrecargaMin\% = SobrecargaMin/\bar{m}$ e $SobrecargaMax\% = SobrecargaMax/\bar{m}$.

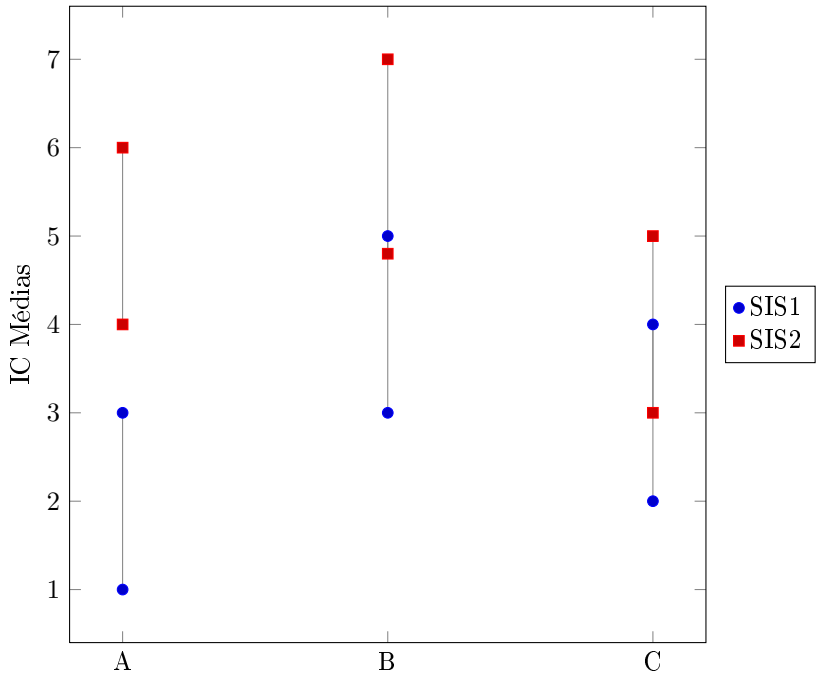


Figura 5.2: Gráfico de comparação entre sistemas pela sobreposição de intervalos.

Experimentos fatoriais

Nesta modalidade de avaliação, um conjunto de experimentos envolvendo fatores e seus níveis é realizado, permitindo verificar o efeito de cada nível do fator, como explicado na Seção 3.5.3. O nível do fator em estudo tem desempenho superior quando seu efeito é mínimo em relação ao efeito de outros níveis. Considerando os fatores PCIB, define-se o projeto de experimentos fatoriais na Tabela 5.2, onde é

apresentado o relacionamento dos níveis distintos para os pares de aplicações SP/BT e LU/FT.

Modelo 5.1 $y_{ijkmn} = \mu + P_i + C_j + I_k + B_m + PC_{ij} + PI_{ik} + PB_{im} + CI_{jk} + CB_{jm} + IB_{km} + PCI_{ijk} + PCB_{ijm} + PIB_{ikm} + CIB_{jkm} + PCIB_{ijkm} + e_{ijkmn}$

ANOVA-Modelo com Replicações			
Componentes SQ	**Soma Quad.**	**%**	**G.L**
SQY	$SQY = \sum y^2_{ijkmn}$		$pcıbr$
SQ0	$SQ0 = pcıbr\mu^2$		
SQT	SQT=SQY-SQ0	100	$pcıbr - 1$
Efeitos Principais	$Total\ Ef.\ Princ.$	$100\frac{Total\ E.P.}{SQT}$	
P	$SQP = cıbr \sum P_i^2$	$100\frac{SQP}{SQT}$	$p-1$
C	$SQC = pıbr \sum C_j^2$	$100\frac{SQC}{SQT}$	$c-1$
I	$SQI = pcbr \sum I_k^2$	$100\frac{SQI}{SQT}$	$ı-1$
B	$SQB = pcır \sum B_m^2$	$100\frac{SQB}{SQT}$	$b-1$
Interações 1	$Total\ Int.\ 1$	$100\frac{Total\ Int.\ 1}{SQT}$	
PC	$SQPC = ıbr \sum PC_{ij}^2$	$100\frac{SQPC}{SQT}$	$(p-1)(c-1)$
PI	$SQPI = cbr \sum PI_{ik}^2$	$100\frac{SQPI}{SQT}$	$(p-1)(ı-1)$
CI	$SQCI = pbr \sum CI_{jk}^2$	$100\frac{SQCI}{SQT}$	$(c-1)(ı-1)$
PB	$SQPB = cır \sum PB_{im}^2$	$100\frac{SQPB}{SQT}$	$(p-1)(b-1)$
CB	$SQCB = pır \sum CB_{jm}^2$	$100\frac{SQCB}{SQT}$	$(c-1)(b-1)$
IB	$SQIB = pcr \sum IB_{km}^2$	$100\frac{SQIB}{SQT}$	$(ı-1)(b-1)$
Interações 2	$Total\ Int.\ 2$	$100\frac{Total\ Int.\ 2}{SQT}$	
PCI	$SQPCI = br \sum PCI_{ijk}^2$	$100\frac{SQPCI}{SQT}$	$(p-1)(c-1)(ı-1)$
PCB	$SQPCB = ır \sum PCB_{ijm}^2$	$100\frac{SQPCB}{SQT}$	$(p-1)(c-1)(b-1)$
PIB	$SQPIB = cr \sum PIB_{ikm}^2$	$100\frac{SQPIB}{SQT}$	$(p-1)(ı-1)(b-1)$
CIB	$SQCIB = pr \sum CIB_{jkm}^2$	$100\frac{SQCIB}{SQT}$	$(c-1)(ı-1)(b-1)$
Interação 3 + SQE	$SQPCIB + SQE$	$100\frac{SQPCIB+SQE}{SQT}$	
PCIB	$SQPCIB = r \sum PCIB_{ijkm}^2$	$100\frac{SQPCIB}{SQT}$	$(p-1)(c-1)(ı-1)(b-1)$
SQE	$SQE = \sum e_{ijkmn}^2$	$100\frac{SQE}{SQT}$	$pcıb(r-1)$

Tabela 5.1: Modelo da ANOVA para os fatores PCIB.

Utiliza-se o Modelo 5.1 com quatro fatores para o estudo fatorial, bem como a *ANOVA* desse modelo para destacar variações dos dados, definida na Tabela 5.1. Embora haja muitos efeitos principais no modelo, representados por P_i, C_j, I_k e B_m, bem como interações, representadas por PC_{ij}, PI_{ik}, PB_{im}, CI_{jk}, CB_{jm}, IB_{km}, PCI_{ijk}, PCB_{ijm}, PIB_{ikm}, CIB_{jkm} e $PCIB_{ijkm}$, o Modelo 5.1 permite destacar as

aplicações (B) mais intensivas, o que não seria possível se fosse utilizado o modelo baseado em três fatores. Com o fator P, pode-se verificar a tendência de desempenho à medida em que se aumenta o número de processadores, com os efeitos principais mínimos representando maior desempenho. Com o fator C, é possível explorar as limitações do hardware devido ao aumento do tamanho de problema (carga de trabalho). Com o fator I, comparam-se as versões monolíticas e refatoradas, que é o objetivo principal desta avaliação de desempenho.

Cada célula $\bar{y}_{ijkm}$ da Tabela 5.2 é resultante da média de r replicações, livres de *outliers*. Antes de aplicar as médias experimentais, a resposta de cada medição y_{ijkmn} é convertida em logaritmo na base dez. Conforme a literatura, resultados mais confiáveis são obtidos com a aplicação de logaritmos, especialmente para análise de tempo de processamento [47]. A média geral é dada por $\mu = \sum_{i=1}^{p}\sum_{j=1}^{c}\sum_{k=1}^{\imath}\sum_{m=1}^{b}\sum_{n=1}^{r}\frac{y_{ijkmn}}{pc\imath br}$, com p, c, $\imath$ e b sendo a quantidade de níveis dos fatores PCIB respectivamente. O efeito e interação de cada nível de fator está compreendido no intervalo $[-\mu, +\mu]$, com a maior aproximação entre R e M dada pela menor distância de seus efeitos principais I_k ao valor zero.

As aplicações SP e BT usam algoritmos que restringem a quantidade de processadores para n^2, para $1 \leq n \leq \infty$. Dessa forma, tem-se uma incompatibilidade com LU e FT quanto ao número de processadores utilizados, pois LU e FT requerem 2^n processadores. Esse é o motivo pelo qual são realizados experimentos fatoriais separados, porém com quatro fatores, dados por $B[SP, BT]$ ou $B[LU, FT]$.

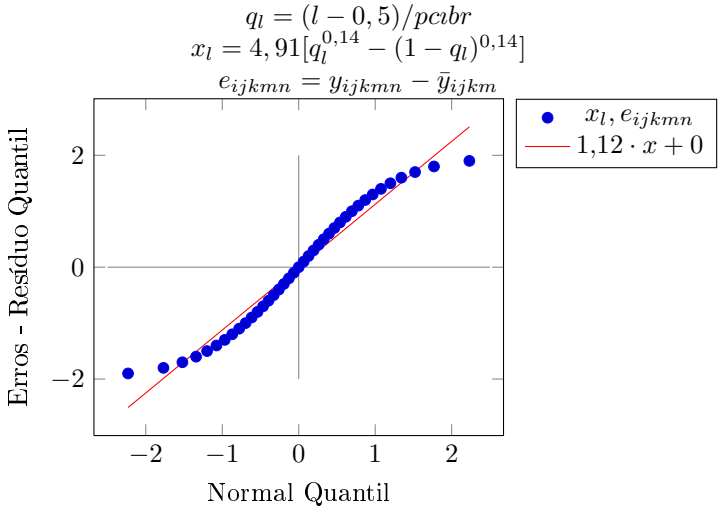


Figura 5.3: Gráfico com distribuição Normal Quantil Quantil .

Fatores	Níveis - distintos com SP/BT ou LU/FT		Definições de efeitos principais
P	$[p_1, p_4, p_9, p_{16}]$	$[p_1, p_2, p_4, p_8, p_{16}]$	$P_i \models 1 \leq i \leq p$, para [p=4] | [p=5]
C	[S, W, A]	[W, A]	$C_j \models 1 \leq j \leq c$, para [c=3] | [c=2]
I	M, R		$I_k \models 1 \leq k \leq \imath$, para $\imath$=2
B	SP, BT	LU, FT	$B_m \models 1 \leq m \leq b$, para b=2

Relacionamento de experimentos com SP/BT						
			p_1	p_4	p_9	p_{16}
SP	M	S	$\bar{y}_{1111}$	$\bar{y}_{2111}$	$\bar{y}_{3111}$	$\bar{y}_{4111}$
		W	$\bar{y}_{1211}$	$\bar{y}_{2211}$	$\bar{y}_{3211}$	$\bar{y}_{4211}$
		A	$\bar{y}_{1311}$	$\bar{y}_{2311}$	$\bar{y}_{3311}$	$\bar{y}_{4311}$
	R	S	$\bar{y}_{1121}$	$\bar{y}_{2121}$	$\bar{y}_{3121}$	$\bar{y}_{4121}$
		W	$\bar{y}_{1221}$	$\bar{y}_{2221}$	$\bar{y}_{3221}$	$\bar{y}_{4221}$
		A	$\bar{y}_{1321}$	$\bar{y}_{2321}$	$\bar{y}_{3321}$	$\bar{y}_{4321}$
BT	M	S	$\bar{y}_{1112}$	$\bar{y}_{2112}$	$\bar{y}_{3112}$	$\bar{y}_{4112}$
		W	$\bar{y}_{1212}$	$\bar{y}_{2212}$	$\bar{y}_{3212}$	$\bar{y}_{4212}$
		A	$\bar{y}_{1312}$	$\bar{y}_{2312}$	$\bar{y}_{3312}$	$\bar{y}_{4312}$
	R	S	$\bar{y}_{1122}$	$\bar{y}_{2122}$	$\bar{y}_{3122}$	$\bar{y}_{4122}$
		W	$\bar{y}_{1222}$	$\bar{y}_{2222}$	$\bar{y}_{3222}$	$\bar{y}_{4222}$
		A	$\bar{y}_{1322}$	$\bar{y}_{2322}$	$\bar{y}_{3322}$	$\bar{y}_{4322}$

Relacionamento de experimentos com LU/FT							
			p_1	p_2	p_4	p_8	p_{16}
LU	M	W	$\bar{y}_{1111}$	$\bar{y}_{2111}$	$\bar{y}_{3111}$	$\bar{y}_{4111}$	$\bar{y}_{5111}$
		A	$\bar{y}_{1211}$	$\bar{y}_{2211}$	$\bar{y}_{3211}$	$\bar{y}_{4211}$	$\bar{y}_{5211}$
	R	W	$\bar{y}_{1121}$	$\bar{y}_{2121}$	$\bar{y}_{3121}$	$\bar{y}_{4121}$	$\bar{y}_{5121}$
		A	$\bar{y}_{1221}$	$\bar{y}_{2221}$	$\bar{y}_{3221}$	$\bar{y}_{4221}$	$\bar{y}_{5221}$
FT	M	W	$\bar{y}_{1112}$	$\bar{y}_{2112}$	$\bar{y}_{3112}$	$\bar{y}_{4112}$	$\bar{y}_{5112}$
		A	$\bar{y}_{1212}$	$\bar{y}_{2212}$	$\bar{y}_{3212}$	$\bar{y}_{4212}$	$\bar{y}_{5212}$
	R	W	$\bar{y}_{1122}$	$\bar{y}_{2122}$	$\bar{y}_{3122}$	$\bar{y}_{4122}$	$\bar{y}_{5122}$
		A	$\bar{y}_{1222}$	$\bar{y}_{2222}$	$\bar{y}_{3222}$	$\bar{y}_{4222}$	$\bar{y}_{5222}$
$\bar{y}_{ijkm}$ é a média de 40 replicações sem outliers							

Tabela 5.2: Estrutura dos experimentos fatoriais

Verificação de dispersão de dados Para verificar o comportamento dos dados, é utilizado o gráfico de probabilidade *QQ (Quantil Quantil)*, construído a partir dos erros experimentais (e_{ijkmn}), definidos pela diferença entre o valor medido e o valor esperado, e a função inversa de distribuição acumulada x_l. A construção de gráficos QQ foi discutida na Seção 3.5.3, onde as medições são representadas graficamente por (x_l, e_{ijk}), para um modelo dois fatores com replicações. Para avaliação por experimentos fatoriais neste capítulo, o comportamento das medições é definido graficamente por $e_{ijkmn} = y_{ijkmn} - \bar{y}_{ijkm}$ e $x_l \approx 4.91[q_l^{0.14} - (1-q_l)^{0.14}]$, com $q_l = \frac{l-0.5}{pcıbr}$, tal que $1 \leq l \leq pcıbr$. Uma distribuição considerada normal é dada pelo seguimento dos erros em torno da reta do gráfico, com simetria em torno de zero, como na Figura 5.3.

Ferramentas Utilizadas para Cálculos e Gráficos do Estudo

As medições dos experimentos são inseridas em planilhas eletrônicas, especialmente para armazenagem dos dados e pela facilidade na geração de gráficos e cálculos estatísticos. Utiliza-se planilhas compatíveis com os aplicativos *Excel* da *Microsoft* e *OpenOffice* de código livre. Para este trabalho de dissertação, as planilhas são publicadas no repositório do Projeto *NPB for HPE*, com endereço eletrônico disponível em "http://code.google.com/p/npb-for-hpe/downloads/list". Cada planilha contém as tabelas de resultados apresentadas na Seção 5.3, onde são separadas em comparação por IC e experimentos fatoriais. Os nomes dos arquivos são relacionados com a técnica e aplicações envolvidas. Nos resultados com experimentos fatoriais apresentados nas planilhas, o leitor pode encontrar todas as subtabelas de interações, seguindo o Modelo 5.1.

5.2 Considerações Gerais sobre a Avaliação de Desempenho

Na avaliação de desempenho do HPE, está definida a seção *Experimentos com o NPB* para apresentar e discutir os resultados da avaliação. A seção está dividida em duas técnicas, sendo elas relativas a comparação por intervalos de confiança e aos experimentos fatoriais.

Com a comparação por intervalos de confiança (*Teste-t* e sobreposição de IC), busca-se resultados mais específicos e isolados. Necessita-se saber qual versão, definida pelo fator I (níveis R ou M), tem comportamento melhor para cada combinação de níveis dos demais fatores: número de processadores (P), carga de trabalho (C) e aplicação (B). Através desses resultados, mensura-se e analisa-se como se comportam as *sobrecargas* de R em relação a M. Os resultados são

apresentados em várias tabelas e gráficos. Esses gráficos permitem visualizar o comportamento dos intervalos de confiança de cada experimento, e estão divididos em dois tipos: gráficos de sobreposição de intervalos, como apresentado na Figura 5.4, e gráficos que permitem visualizar a curva de desempenho das aplicações, como apresentado na Figura 5.12. Para a apresentação de resultados em tabelas, não são utilizados os valores das medições diretamente, principalmente por ultrapassarem mil unidades. São apresentados sumários estatísticos dessas medições, incluindo média, desvio padrão, intervalo de confiança a 95%, porcentagem de sobrecarga e outros indicadores de comparação entre R e M.

Com a comparação através de experimentos fatoriais, busca-se obter resultados que não podem ser encontrados por intervalos de confiança de amostras isoladas. Envolvem-se todos os experimentos para tirar conclusões particulares sobre as interações entre os fatores e seus níveis. Experimentos são apresentados em tabelas que cruzam dados dos níveis de fatores, como na Tabela 5.2. Adicionalmente, são geradas tabelas aninhadas que especificam os efeitos que cada nível apresenta, tendo como base o Modelo 5.1. Finaliza-se o estudo pela discussão detalhada sobre resultados fatoriais, onde os efeitos mais e menos significativos são destacados.

5.2.1 Plataforma Experimental

Como plataforma de execução dos experimentos para esta avaliação, foi empregado o *Castanhão*, *cluster* implantado pelo Departamento de Computação da Universidade Federal de Ceará. O *Castanhão* conta com 16 nós dedicados aos experimentos. Cada nó é composto por dois processadores *Intel Xeon* e *1GB* de memória principal. Os processadores executam sob *clock* de *1,8 GHz* e possuem memória *cache* de *512 KB*, suportando a tecnologia *Intel Hyper-Threading*, o que permite quatro *threads* paralelas em um único nó. Nos experimentos, somente uma *thread* em cada processo foi utilizada para realizar o processamento, a fim de evitar instabilidades nas medições causadas pela execução *multi-threading* dentro dos nós.

Os nós do *cluster* estão interconectados através de duas redes distintas, de tecnologias *Fast Ethernet* (*100Mbps*) e *Gigabit Ethernet* (*1Gbps*). Foi utilizada a rede *Fast Ethernet*, a fim de exacerbar-se a sobrecarga de comunicação, permitindo uma análise mais apurada do efeito da refatoração em componentes sobre a comunicação entre processos.

Com relação ao ambiente de software, o *Castanhão* é um cluster *Rocks 5.0* com *Linux 2.6.18*. A versão do Mono utilizada nos experimentos é a *2.2*, enquanto a

biblioteca MPI utilizada é a *MPICH2 1.2.1*.

5.3 Experimentos com o NPB

Os dados desta avaliação de desempenho estão agrupados ao longo desta seção em várias tabelas, por meio das quais são discutidos os resultados. Para melhor organizar a avaliação, estudo foi dividido nas Seções 5.3.1 e 5.3.3, primeiramente efetuando a análise por intervalos de confiança de amostras e posteriormente por experimentos fatoriais.

5.3.1 Comparação entre M e R por Intervalos de Confiança

Os experimentos realizados geraram uma quantidade significativa de medições. A disponibilização integral desses dados é feita através do *Projeto NPB for* HPE, encontrado no endereço eletrônico "*http://npb-for-hpe.googlecode.com/svn/trunk/medicoes*". Dessa forma, esta seção concentra-se nas principais informações experimentais, como médias, desvios padrões, sobrecargas e intervalos de confiança. Os dados obtidos a partir dos experimentos para comparação por *ICs* estão agrupados em seis tabelas principais, contendo informações experimentais da combinação dos seguintes fatores/níveis:

- $C[S] \times I[R, M] \times B[SP, BT] \times P[p_1, p_4, p_9, p_{16}]$, disponível na Tabela 5.3;
- $C[W] \times I[R, M] \times B[SP, BT] \times P[p_1, p_4, p_9, p_{16}]$, disponível na Tabela 5.4;
- $C[A] \times I[R, M] \times B[SP, BT] \times P[p_1, p_4, p_9, p_{16}]$, disponível na Tabela 5.5;
- $C[W] \times I[R, M] \times B[LU, FT] \times P[p_1, p_2, p_4, p_8, p_{16}]$, disponível na Tabela 5.6;
- $C[A] \times I[R, M] \times B[LU, FT] \times P[p_1, p_2, p_4, p_8, p_{16}]$, disponível na Tabela 5.7.

As tabelas estão divididas em resultados de intervalos de confiança da diferença $R - M$, e resultados de intervalos R e M para análise por sobreposição. Descreve-se em seguida os significados das células principais dessas tabelas.

- ***Teste-t***: parâmetro probabilístico envolvendo as amostras de R e M, considerando variâncias diferentes e dados não pareados. *Teste-t* é representado pelo parâmetro probabilístico *área-p*, como discutido na metodologia;

- ***Teste-t*** $< 0,05$ **?**: compara *Teste-t* com o nível de significância $\alpha = 0,05$. Os possíveis valores são: *sim*, se *Teste-t* é menor que α (hipótese alternativa); ou *não*, caso contrário (hipótese nula);

- **Min sobrecarga**: é o parâmetro de referência de computação excessiva da versão R em relação a M. Tem como base a média das medições da versão M em um experimento, comparando-a com os limites mais otimistas do intervalo de confiança do tempo de execução da versão R em relação a versão M, ou seja considerando-se a diferença entre a melhor estimativa para o tempo de execução da versão R pela pior estimativa da versão M. Sobrecargas negativas significam que a versão refatorada obteve desempenho superior à monolítica;

- **Max sobrecarga**: semelhante à *Min sobrecarga*, porém considerando-se a diferença entre a pior estimativa para o tempo de execução da versão R e a melhor estimativa da versão M;

- μ **sobrecarga**: representa a média das sobrecargas mínima é máxima;

- **Resultado[*t*]**: a comparação é feita tendo como referência o *Teste-t*. Os valores possíveis são: R, quando a hipótese alternativa é aceita e a sobrecarga é negativa; M, quando a hipótese alternativa é aceita e a sobrecarga é positiva; e *Iguais*, para a hipótese nula.

- **Tamanho** R/M: refere-se ao tamanho da amostra R ou M;

- **Desvio Padrão** R/M: é o desvio padrão da amostra R ou M;

- **IC|R|, IC|M|**: o intervalo de confiança para as amostras de R ou M em um experimento;

- **Resultado**: refere-se à comparação pela sobreposição de intervalos de confiança. As hipóteses possíveis são: R, para experimentos onde $IC[R]$ e $IC[M]$ não se sobrepõem, apresentando intervalos $IC[R]$ com valores menores que intervalos $IC[M]$; M, para experimentos onde $IC[R]$ e $IC[M]$ não se sobrepõem, apresentando intervalos $IC[M]$ com valores menores que intervalos $IC[R]$; *Iguais*, para experimentos onde $IC[R]$ e $IC[M]$ se sobrepõem, incluindo ainda a média de um IC dentro do outro. Resultados classificados como *Iguais* devem ser também confirmados pelos resultados do atributo "Resultado[t]".

B/C	SP/S				BT/S			
P	**1**	**4**	**9**	**16**	1	4	9	16
Teste-t								
Teste-t	**0,000**	**0,000**	**0,000**	**0,000**	**0,001**	**0,002**	**0,000**	**0,000**
(Teste-t<α)?	**sim**	**sim**	**sim**	**sim**	**sim**	**sim**	**sim**	**sim**
Min sobrecarga	4,4%	-10,3%	-18,0%	-23,2%	1,2%	1,0%	5,7%	5,5%
Max sobrecarga	5,2%	-9,5%	-16,9%	-22,4%	5,6%	4,3%	7,1%	6,9%
μ sobrecarga	4,8%	-9,9%	-17,5%	-22,8%	3,4%	2,7%	6,4%	6,2%
Resultado[t]	**M**	**R**	**R**	**R**	M	M	M	M
Avaliação pela Sobreposição de Intervalos								
Tamanho R	**40**	**40**	**36**	**36**	40	39	40	39
Média R	**5,593**	**2,001**	**1,507**	**1,404**	22,479	6,459	4,068	2,990
Desvio Padrão R	**0,012**	**0,003**	**0,006**	**0,010**	0,189	0,014	0,012	0,009
IC[R]	**5,589**	**2,000**	**1,505**	**1,400**	22,420	6,454	4,065	2,987
	5,597	**2,002**	**1,509**	**1,407**	22,538	6,463	4,072	2,993
Tamanho M	**39**	**40**	**39**	**38**	39	40	40	39
Média M	**5,33**	**2,22**	**1,83**	**1,82**	21,748	6,291	3,824	2,815
Desvio Padrão M	**0,054**	**0,025**	**0,023**	**0,013**	1,290	0,312	0,075	0,050
IC[M]	**5,32**	**2,21**	**1,82**	**1,81**	21,343	6,194	3,801	2,799
	5,35	**2,23**	**1,83**	**1,82**	22,153	6,387	3,847	2,830
Resultado	**M**	**R**	**R**	**R**	M	M	M	M

Tabela 5.3: Resultados do SP/BT em carga S.

B/C	**SP/W**				BT/W			
P	**1**	**4**	**9**	**16**	1	4	9	16
Teste-t								
Teste-t	**0,000**	**0,000**	**0,778**	**0,003**	**0,001**	**0,103**	**0,031**	**0,012**
(Teste-t$<\alpha$)?	**sim**	**sim**	**não**	**sim**	**sim**	**não**	**sim**	**sim**
Min sobrecarga	4,1%	2,2%	-1,6%	-5,0%	1,1%	-2,5%	0,0%	0,2%
Max sobrecarga	5,3%	3,0%	2,1%	-0,9%	5,3%	0,4%	2,6%	2,3%
μ sobrecarga	4,7%	2,6%	0,2%	-3,0%	3,2%	-1,1%	1,3%	1,3%
Resultado[t]	**M**	**M**	**Iguais**	**R**	M	Iguais	M	M
Avaliação pela Sobreposição de Intervalos								
Tamanho R	**40**	**39**	**38**	**36**	40	40	40	39
Média R	**844,243**	**225,534**	**107,019**	**65,725**	723,062	193,474	99,022	60,866
Desvio Padrão R	**6,203**	**0,467**	**0,471**	**0,272**	9,351	1,289	0,526	0,168
IC[R]	**842,320**	**225,388**	**106,869**	**65,636**	720,164	193,075	98,859	60,813
	846,165	**225,681**	**107,169**	**65,814**	725,960	193,874	99,185	60,918
Tamanho M	**37**	**37**	**40**	**40**	40	40	40	40
Média M	**806,53**	**219,79**	**106,76**	**67,74**	700,760	195,554	97,795	60,098
Desvio Padrão M	**9,214**	**2,338**	**5,755**	**4,044**	36,862	7,771	3,438	1,824
IC[M]	**803,56**	**219,03**	**104,98**	**66,48**	689,336	193,145	96,730	59,533
	809,49	**220,54**	**108,54**	**68,99**	712,183	197,962	98,861	60,664
Resultado	**M**	**M**	**Iguais**	**R**	M	Iguais	M	M

Tabela 5.4: Resultados do SP/BT em carga W.

B/C	**SP/A**				BT/A			
P	**1**	**4**	**9**	**16**	1	4	9	16
Teste-t								
Teste-t	**0,000**	**0,011**	**0,000**	**0,017**	**0,000**	**0,000**	**0,000**	**0,051**
(Teste-t$<\alpha$)?	**sim**	**sim**	**sim**	**sim**	**sim**	**sim**	**sim**	**não**
Min sobrecarga	4,0%	0,2%	2,8%	-4,2%	5,6%	6,0%	-3,9%	-5,1%
Max sobrecarga	5,1%	2,3%	5,0%	-0,3%	6,9%	7,5%	-1,4%	0,2%
μ sobrecarga	4,5%	1,3%	3,9%	-2,2%	6,2%	6,7%	-2,7%	-2,4%
Resultado[t]	**M**	**M**	**M**	**R**	M	M	R	Iguais
Avaliação pela Sobreposição de Intervalos								
Tamanho R	**39**	**40**	**36**	**40**	40	40	39	40
Média R	**4923,843**	**1306,150**	**633,632**	**373,511**	15090,689	3917,879	1862,093	1059,992
Desvio Padrão R	**17,589**	**4,634**	**2,116**	**2,082**	113,210	21,293	8,056	7,320
IC[R]	**4918,323**	**1304,714**	**632,941**	**372,866**	15055,605	3911,280	1859,565	1057,724
	4929,363	**1307,586**	**634,323**	**374,156**	15125,773	3924,478	1864,621	1062,261
Tamanho M	**39**	**40**	**40**	**38**	39	35	40	40
Média M	**4.711,44**	**1.289,77**	**610,00**	**382,02**	14209,026	3670,361	1912,791	1086,518
Desvio Padrão M	**61,532**	**38,764**	**18,899**	**20,876**	172,184	74,798	74,964	83,167
IC[M]	**4692,13**	**1277,76**	**604,14**	**375,39**	14154,986	3645,580	1889,560	1060,744
	4730,75	**1301,79**	**615,86**	**388,66**	14263,066	3695,141	1936,023	1112,291
Resultado	**M**	**M**	**M**	**R**	M	M	R	R

Tabela 5.5: Resultados do SP/BT em carga A.

B/C	LU/W					FT/W				
P	**1**	**2**	**4**	**8**	**16**	**1**	**2**	**4**	**8**	**16**
Teste-t										
Teste-t	**0,000**	**0,000**	**0,000**	**0,000**	**0,004**	**0,014**	**0,000**	**0,067**	**0,004**	**0,000**
(Teste-t<α)?	**sim**	**sim**	**sim**	**sim**	**sim**	**sim**	**sim**	**não**	**sim**	**sim**
Min sobrecarga	2,3%	2,0%	3,8%	4,2%	-1,5%	-1,1%	-0,6%	-1,0%	-1,7%	-25,3%
Max sobrecarga	3,7%	3,0%	5,6%	6,7%	-0,2%	0,0%	-0,1%	0,1%	-0,3%	-16,9%
μ sobrecarga	3,0%	2,5%	4,7%	5,5%	-0,9%	-0,5%	-0,4%	-0,4%	-1,0%	-21,1%
Resultado[t]	M	M	M	M	R	R	R	Iguais	R	R
Avaliação pela Sobreposição de Intervalos										
Tamanho R	40	40	40	40	38	40	40	40	40	40
Média R	773,955	417,599	228,642	127,312	71,739	27,828	18,236	9,783	5,055	2,580
Desvio Padrão R	5,097	1,569	1,501	0,526	0,194	0,232	0,071	0,052	0,016	0,175
IC[R]	772,376	417,113	228,177	127,149	71,677	27,756	18,214	9,767	5,050	2,526
	775,535	418,086	229,107	127,475	71,800	27,900	18,258	9,799	5,060	2,634
Tamanho M	28	36	40	40	38	38	35	37	40	40
Média M	751,219	407,530	218,414	120,715	72,359	27,981	18,304	9,823	5,106	3,270
Desvio Padrão M	9,918	4,990	4,838	4,309	1,239	0,293	0,076	0,119	0,104	0,266
IC[M]	747,545	405,900	216,914	119,379	71,965	27,887	18,279	9,784	5,073	3,187
	754,893	409,160	219,913	122,050	72,752	28,074	18,329	9,861	5,138	3,352
Resultado	M	M	M	M	R	R	R	R	R	R

Tabela 5.6: Resultados do LU/FT em carga W.

B/C	LU/A					FT/A				
P	**1**	**2**	**4**	**8**	**16**	**1**	**2**	**4**	**8**	**16**
Teste-t										
Teste-t	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,184**	**0,001**	**0,486**	**0,000**	**0,000**
(Teste-t<α)?	**sim**	**sim**	**sim**	**sim**	**sim**	**não**	**sim**	**não**	**sim**	**sim**
Min sobrecarga	3,4%	1,8%	3,6%	1,4%	5,1%	-0,6%	-1,8%	-0,7%	-3,3%	-3,6%
Max sobrecarga	4,8%	3,6%	5,8%	3,9%	7,3%	0,2%	0,4%	0,4%	-1,1%	-1,4%
μ sobrecarga	4,1%	2,7%	4,7%	2,7%	6,2%	-0,2%	-0,7%	-0,1%	-2,2%	-2,5%
Resultado[t]	M	M	M	M	M	Iguais	R	Iguais	R	R
Avaliação pela Sobreposição de Intervalos										
Tamanho R	38	40	40	40	40	37	40	40	40	39
Média R	5315,306	2742,742	1446,729	760,410	418,997	465,042	302,454	164,658	86,278	46,743
Desvio Padrão R	27,728	12,337	4,951	2,551	2,027	1,364	0,723	0,812	0,385	0,293
IC[R]	5306,490	2738,919	1445,195	759,620	418,369	464,602	302,230	164,407	86,159	46,651
	5324,122	2746,565	1448,264	761,201	419,626	465,481	302,678	164,910	86,398	46,835
Tamanho M	34	40	40	40	39	38	37	36	39	39
Média M	5106,826	2670,576	1382,250	740,692	394,548	466,099	304,509	164,890	88,206	47,936
Desvio Padrão M	84,531	68,870	44,267	27,929	11,295	4,624	9,547	1,826	2,653	1,437
IC[M]	5078,412	2649,233	1368,532	732,037	391,004	464,629	301,433	164,294	87,374	47,485
	5135,241	2691,919	1395,968	749,347	398,093	467,570	307,585	165,487	89,039	48,387
Resultado	M	M	M	M	M	Iguais	Iguais	Iguais	R	R

Tabela 5.7: Resultados do LU/FT em carga A.

Discussão de Resultados da Comparação entre Intervalos

Quando é realizada uma refatoração em componentes a partir de uma versão monolítica sem que sejam alteradas as estruturas de dados, computações e operações de comunicação, deixando as versões distintas apenas pelas configurações que caracterizam o componente, espera-se pelo senso comum que exista alguma sobrecarga positiva, mesmo que mínima, relativa a versão refatorada. Espera-se então definir uma expressão $Tempo(M) = Tempo(R) - \varphi$, onde φ caracteriza a sobrecarga resultante da refatoração em componentes.

Contrariando o senso comum, sobrecargas negativas, caracterizando melhor desempenho para a versão R, foram obtidas em várias comparações de SP e BT, como pode ser confirmado nas tabelas 5.3, 5.4, 5.5, 5.6, e 5.7. Tal efeito negativo, retrata uma situação possível de ocorrer, principalmente quando os sistemas apresentam desempenhos muito próximos. Nos próximos parágrafos, são analisados com mais detalhes os resultados para cada programa.

De 12 experimentos relativos ao programa SP, 6 foram favoráveis à versão refatorada, apresentando sobrecargas negativas. Entre esses, apenas a combinação $C[W] \times P[p_9] \times I[M,R] \times B[SP]$ resultou em igualdade entre versões, pois tem resposta *não* no *Teste-t*, bem como apresentou sobrecarga mínima como negativa, embora a média das sobrecargas seja positiva.

Com o auxílio do gráfico da Figura 5.12 do SP, é possível verificar as características de desempenho à medida em que se aumenta o número de processadores. Por exemplo, observa-se uma curva decrescente com a cargas de trabalho W/A, começando com R em menor desempenho quando utiliza-se 1 processador, e aproximando-se de M a partir de 4 processadores, favorecendo R principalmente com 9 e 16 processadores. Os experimentos com carga de trabalho S possuem tempo de execução muito curtos, tendo sido definidos pelo NPB apenas para testar os programas. Por esse motivo, são mais sensíveis a instabilidades momentâneas do sistema durante a execução, o que pode causar instabilidades nas respostas de tempo.

Essa pode estar entre as causas das maiores sobrecargas negativas observadas nos experimentos com carga de trabalho S que favorecem as versões R nos experimentos com o programa SP, bem como as maiores sobrecargas positivas, favorecendo as versões M, para a carga de trabalho S no programa BT. Para as demais cargas de trabalho (W e A), há uma situação mais realística, pois tratam-se de problemas maiores que S, utilizados de fato para os propósitos do NPB, gerando sobrecargas

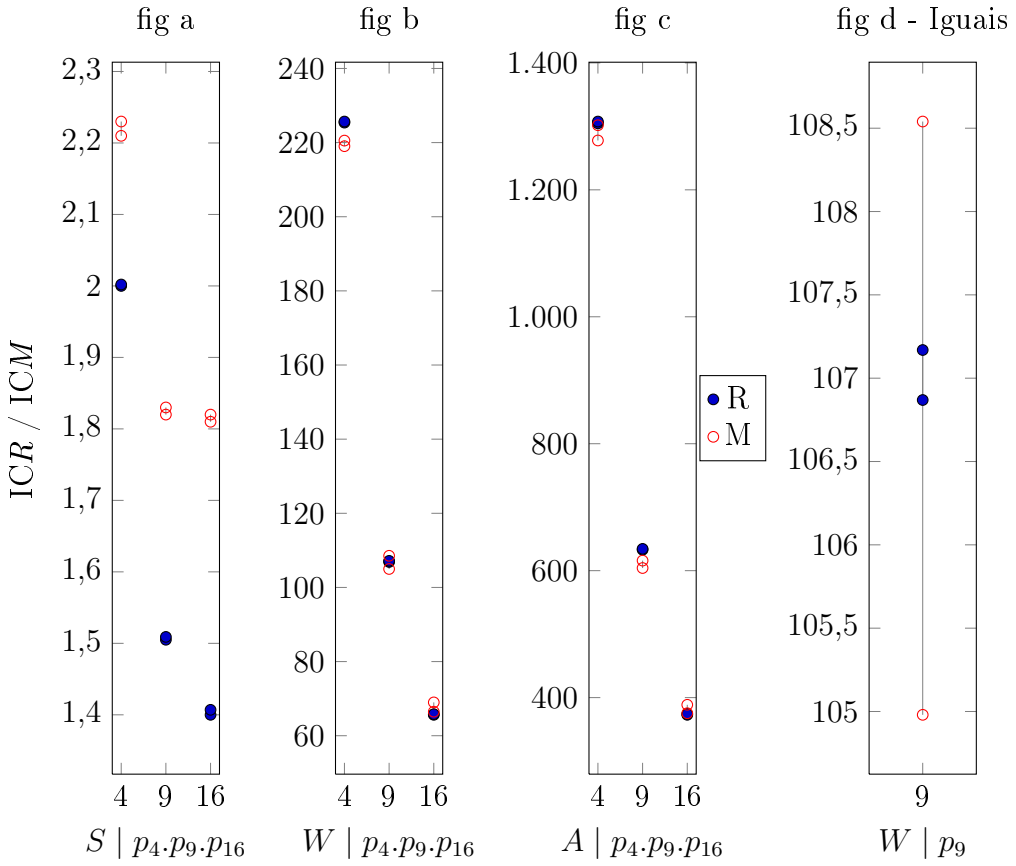


Figura 5.4: SP-Gráfico de sobreposição de intervalos. Processadores × IC.

menores, como pode ser visto em todas as tabelas de resultados do SP e BT.

Um fato perceptível é que o desvio padrão nos resultados da versão monolítica é maior que o desvio padrão da versão refatorada, o que pode ser visualmente notado pelos tamanhos dos intervalos expressos nos gráficos das Figuras 5.4 e 5.5. Isso é causado pela forma como são disparados os programas das versões M e R. O programa na versão R é instanciado uma única vez, tendo suas estruturas de dados inicializadas nesse momento e reutilizadas a cada repetição do experimento. Dessa forma, o alinhamento dos dados nas hierarquias da memória é mantido a cada repetição. Por sua vez, os programas na versão monolítica são instanciados a cada experimento, tendo suas estruturas de dados alocadas na memória a cada repetição. Essa é uma propriedade dos sistemas baseados em componentes. Portanto, deve ter seu efeito considerado nos experimentos. Apesar dos desvios padrões maiores em M, gerando intervalos maiores, existe apenas uma única sobreposição de intervalo, dada nos experimentos $\overrightarrow{IC[R]}\{SP, W, p_9\} \times \overrightarrow{IC[M]}\{SP, W, p_9\}$. Na sobreposição, o intervalo de R tem sua média inclusa no intervalo de M, o que resulta em igualdade

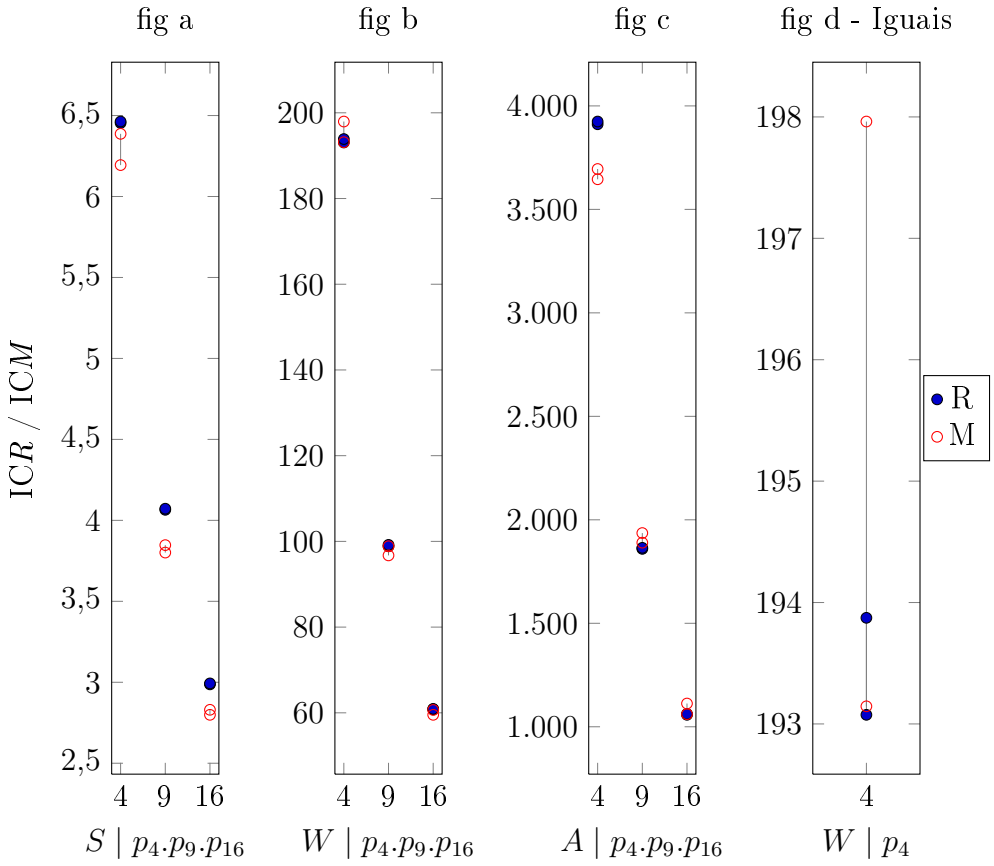


Figura 5.5: BT-Gráfico de sobreposição de intervalos. Processadores × IC.

das versões para esses intervalos.

No cômputo geral dos experimentos com o BT, a versão monolítica teve um comportamento melhor comparada à refatorada. De um total 12 experimentos, foram obtidos 9 com melhor desempenho em M, 2 com desempenhos semelhantes e apenas 1 com melhor desempenho em R, considerando os resultados do *Teste-t*. As igualdades foram em carga W com 4 processadores e em carga A com 16 processadores. Porém, essa última não é confirmada por sobreposição de intervalos, considerando R como melhor, especialmente porque não houve sobreposição de média, e existe uma sobrecarga negativa em favor da versão refatorada. Os demais resultados por sobreposição foram idênticos aos do *Teste-t*.

De um modo geral, os resultados que classificam as versões R como melhores que as versões M estão associados principalmente com os experimentos utilizando cargas de trabalho maiores, tanto para BT quanto para SP.

Em caso de analisar-se o número de processadores que melhor diminui o tempo de processamento dos experimentos com cargas maiores (W e A), observa-se que

o pico de desempenho da plataforma está em 16 processadores, como é esperado. Considerando apenas esse caso, os experimentos $C[W, A] \times P[p_{16}] \times B[SP]$ e $C[A] \times P[p_{16}] \times B[BT]$ apresentaram resultados favoráveis a R, enquanto o $C[W] \times P[p_{16}] \times B[BT]$ apresenta resultado favorável a M. Novamente, é importante salientar que a versão refatorada não pode ser classificada como melhor apenas por apresentar bons resultados em alguns dos principais experimentos. Porém, há resultados de desempenhos semelhantes entre as versões, existindo ainda sobrecargas negativas em diversas medições.

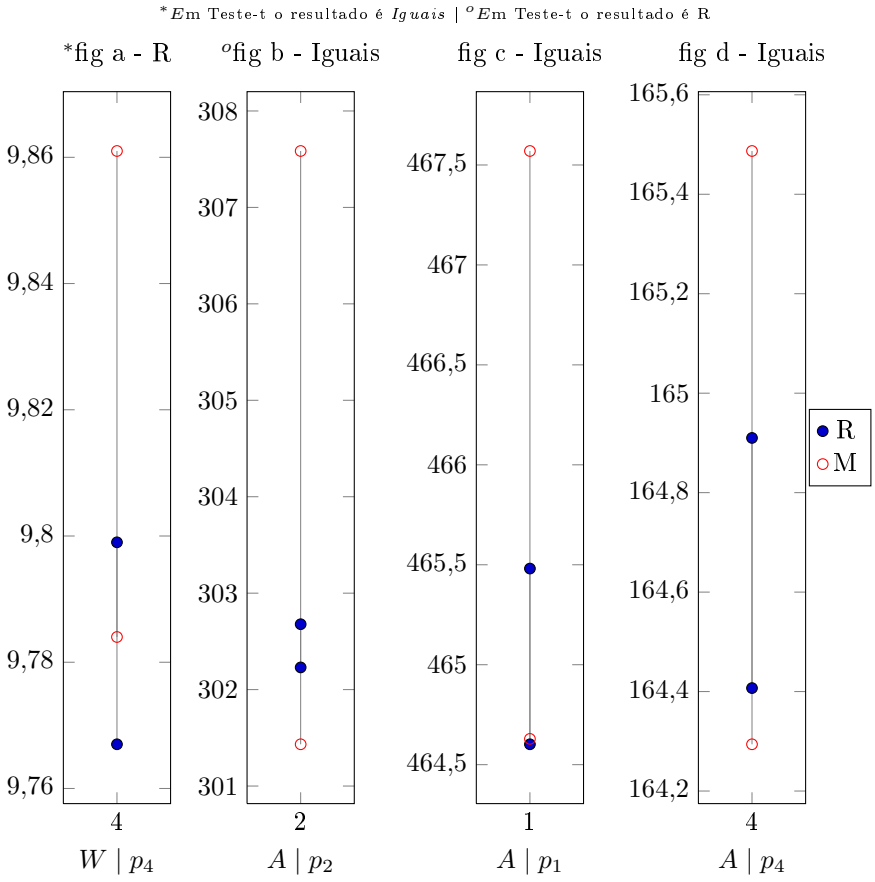


Figura 5.6: FT-Gráfico de sobreposição de intervalos. Processadores × IC.

Com o LU, a versão monolítica obteve ótimos resultados, superando a versão refatorada em nove (9) dos dez (10) experimentos realizados. Somente com a carga de trabalho W em 16 processadores, a versão refatorada supera a monolítica. Em uma análise rápida, os ganhos em favor de M com LU classificariam R como inferior em desempenho, especialmente por não existirem intervalos com sobreposição. Porém, com a inclusão do FT nos experimentos, a versão refatorada ganha maior equilíbrio nos experimentos. Com o FT, todos os testes experimentais realizados

foram em favor de R ou resultaram em igualdade. Sete (7) de dez (10) experimentos tiveram sobrecargas (mínima e máxima) negativas, classificando o desempenho de R como melhor e sendo aceita a hipótese *H1* do *Teste-t*, o que sugere novamente que as diferenças entre M e R são mínimas, considerando ganhos com FT e perdas com LU. Três (3) de dez (10) experimentos com FT geraram sobrecargas (apenas mínima) negativas e foram rejeitados pelo *Teste-t*, classificando as versões como iguais.

Nos resultados por sobreposição classificados como *Iguais*, apenas o experimentos FT em carga A com 2 processadores não confirma o resultado do *Teste-t*, como pode ser visto na Figura 5.6. Isso é possível pelo grande desvio padrão da versão monolítica em comparação ao desvio padrão da versão refatorada, o que gerou um intervalo de confiança extenso para M e mínimo para R. Com isso, o $\overrightarrow{IC[R]}\{FT, A, 2\}$ esteve todo sobreposto ao $\overrightarrow{IC[M]}\{FT, A, 2\}$, gerando igualdade. Entretanto, a média do $\overrightarrow{IC[R]}\{FT, A, 2\}$ está a mais de duas unidades de tempo afastada da média do $\overrightarrow{IC[M]}\{FT, A, 2\}$. Caso semelhante ocorreu com $\overrightarrow{IC[R]}\{BT, W, 4\} \times \overrightarrow{IC[M]}\{BT, W, 4\}$, como pode ser visto na Figura 5.5. Porém, o *Teste-t* confirmou a igualdade. De fato, sobreposição oferece um parâmetro analítico para *ICs*, mas seus resultados devem ser confirmados pelo *Teste-t*. Um segundo experimento por sobreposição que não esteve compatível com o *Teste-t* foi o FT executando em carga W com 4 processadores, como pode ser visto também na Figura 5.6. Dessa vez, o teste de sobreposição favoreceu à versão R enquanto *Teste-t* confirmou igualdade. Isso é possível porque não houve sobreposição de média.

De um modo geral, os indicadores de sobrecargas nas tabelas de resultados do LU e do FT estiveram com valores instáveis, apresentando sobrecargas negativas e positivas de maneira não uniforme em relação às aplicações. Nesse sentido, não há uma tendência padrão para as sobrecargas. Novamente, consideram-se semelhantes os sistemas, devido aos resultados favorecem ambas as versões, dependendo do programa considerado.

Considerando todas as aplicações (SP, BT, LU e FT), os experimentos apresentam resultados onde não é possível afirmar que uma das versões, R ou M, possui desempenho melhor do que a outra. Em quase todos os experimentos, os resultados são favoráveis a uma das versões, com uma leve vantagem em favor da versão monolítica e uma tendência a diminuição da sobrecarga a medida que o número de processadores aumenta. A igualdade entre as versões ocorre em poucos casos, principalmente pelo rigor estatístico empregado, onde valores mínimos fazem grande diferença para definição de um intervalo de confiança ou aplicação do *Teste-t*.

Entretanto, a realização de um estudo desses resultados de forma conjunta através de experimentos fatoriais oferece uma melhor idéia sobre o quanto iguais ou distantes estão as versões R e M. Portanto, o objetivo da avaliação nesta seção foi destacar as particularidades de cada experimento. Na próxima seção, destaca-se o conjunto experimental, verificando especialmente os efeitos principais de desempenho para R e M, bem como o intervalo de confiança desses efeitos.

5.3.2 Análise das Sobrecargas

Analisando-se os intervalos representados pelas sobrecarga mínima e máxima nas tabelas 5.3, 5.4, 5.5, 5.6 e 5.7, observa-se que a maior sobrecarga é 7,5%, observada para o BT na classe A em quatro processadores. Ou seja, a diferença de tempo de execução em favor da versão M de um dos programas SP, BT, LU ou FT não foi maior que 7,5% em relação a versão R.

A sobrecarga média observada foi de 0,98%, com desvio padrão de 4,13%. O intervalo de confiança a 95% é portanto [-0,10%, +1,23%]. Como inclui o zero, não é possível afirmar que a versão M apresenta maior desempenho que a versão R pela análise do intervalo de confiança das sobrecargas.

5.3.3 Experimentos Fatoriais PCIB

Em experimentos envolvendo os fatores PCIB, é importante verificar o comportamento dos dados adquiridos em medições. Como mencionado na metodologia, foram identificados e eliminados *outliers* (medições discrepantes), os quais são esperados pois normalmente ocorrem interferências na plataforma de execução, como instabilidades de acesso à memória ou dispositivos de rede congestionados. Seria possível incluir os *outliers* no estudo e deixar representado todo tipo de comportamento ocorrido na plataforma. Entretanto, algumas medições poderiam fugir significativamente do contexto real. Com a eliminação, pretende-se preservar os resultados que maximizam a capacidade de computação intensiva do problema a ser resolvido. Porém, os dados resultantes também incluem características de um ambiente realístico, passivo de sobrecarga de comunicação, o que pode ser observado pela existência de variações pequenas, porém significativas, nas medições, como concluído a partir dos experimentos com intervalos de confiança. A verificação do comportamento dos dados é feita através do gráfico de probabilidade normal, composto pelos pontos inseridos na Figura 5.7, representando as medições dos experimentos com os pares aplicações de SP/BT e LU/FT. É possível notar que as medições seguem a tendência da reta e estão distribuídas com razoável simetria

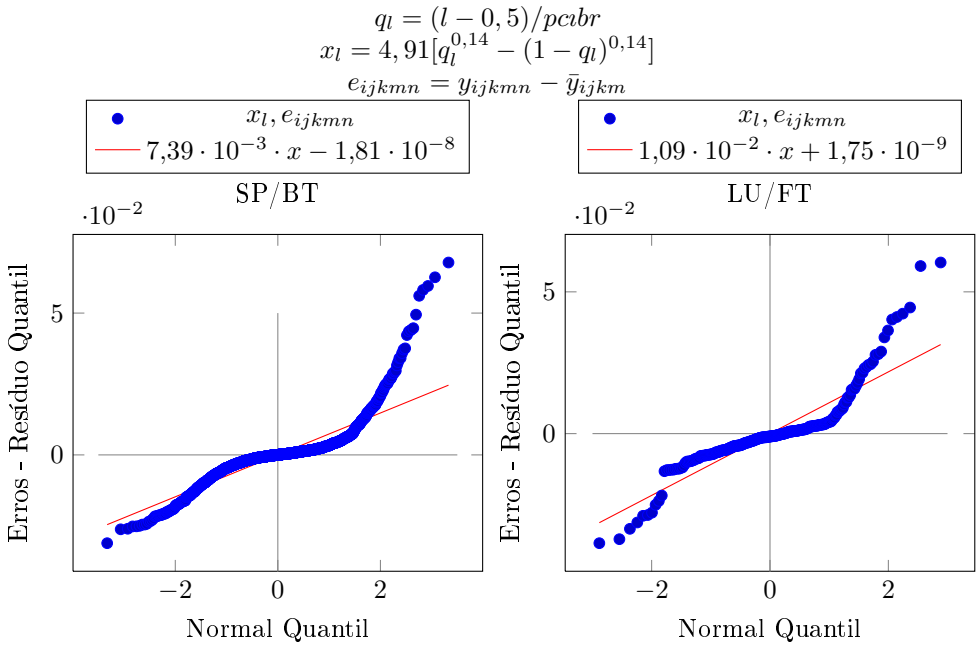


Figura 5.7: Gráfico de probabilidade Normal Quantil Quantil.

em torno de zero.

Com os dados normalmente distribuídos (sem dispersões significativas) no gráfico de probabilidade, a avaliação dos resultados dos experimentos com os fatores PCIB é realizado a partir dos dados da Tabela 5.8 e da Tabela 5.9, onde encontra-se apresentado o relacionamento entre níveis de cada experimento, os efeitos principais com desvio padrão, médias de colunas e linhas, média geral μ de todos os experimentos e informações sobre erros experimentais. Os efeitos são calculados com base no Modelo 5.1. Adicionalmente, cada efeito contém um intervalo de confiança, no qual a inclusão de zero representa efeitos insignificantes. Alguns dos principais dados das tabelas de resultados são:

- SQE: é a soma dos quadrados de todos os erros experimentais. Um erro experimental é o valor medido subtraído pelo valor esperado do experimento ($e_{ijkmn} = y_{ijkmn} - \bar{y}_{ijkm}$), para cada experimento sendo composto por r medições;
- EQM: é o erro quadrático médio experimental, dado por $EQM = \frac{SQE}{pcıb(r-1)}$, para $pcıb$ sendo a quantidade de níveis dos fatores PCIB;
- s_e: é o desvio padrão dos erros experimentais, dado por $s_e = \sqrt{EQM}$;

- Desvio padrão s_{I_k}: é o desvio padrão que representa os efeitos do fator I, dado por $s_{I_k} = s_e\sqrt{\frac{i-1}{pcibr}}$;
- Desvio padrão s_{P_i}: é o desvio padrão que representa os efeitos do fator P, dado por $s_{P_i} = s_e\sqrt{\frac{p-1}{pcibr}}$;
- Desvio padrão s_{B_m}: é o desvio padrão que representa os efeitos do fator B, dado por $s_{B_m} = s_e\sqrt{\frac{b-1}{pcibr}}$;
- Desvio padrão s_{C_j}: é o desvio padrão que representa os efeitos do fator C, dado por $s_{C_j} = s_e\sqrt{\frac{c-1}{pcibr}}$;
- IC Efeito I_k: é o intervalo de confiança para cada efeito principal I_k, dado por $(I_k \mp z \times s_{I_k})$, onde z representa o nível de confiança de 95%[1];
- IC Efeito C_j: é o intervalo de confiança para cada efeito principal C_j, dado por $(C_j \mp z \times s_{C_j})$;
- IC Efeito B_m: é o intervalo de confiança para cada efeito principal B_m, dado por $(B_m \mp z \times s_{B_m})$;
- IC Efeito P_i: é o intervalo de confiança para cada efeito principal P_i, dado por $(P_i \mp z \times s_{P_i})$.

5.3.4 Discussão de resultados dos Experimentos Fatoriais

A avaliação de desempenho feita anteriormente através de vários intervalos de confiança foi capaz de demonstrar as diferenças entre as versões monolítica (M) e refatorada (R) em cada experimento realizado. O estudo realizado aponta vários experimentos onde a versão M se comporta melhor, como também vários experimentos onde R se comporta melhor ou igual a M, em número ligeiramente inferior mas que não caracteriza uma diferença que permita concluir que a versão M tem melhor desempenho que a versão R.

Com os experimentos fatoriais, ao invés de observarem-se casos isolados, é possível avaliar os experimentos de forma conjunta, envolvendo todos os experimentos, e consequentemente possibilitando uma avaliação mais abrangente.

Para a avaliação de desempenho, há nas tabelas 5.8 e 5.9 valores que representam todas as medições. O parâmetro principal é a média geral μ dos experimentos. A

[1]Valores z são encontrados no apêndice B desta dissertação

replicações $r = 28$			p_1	p_2	p_4	p_8	p_{16}	*Média*
LU	M	W	2,8757	2,6080	2,3342	2,0739	1,8564	2,3496
		A	3,7054	3,4269	3,1325	2,8603	2,5898	3,1430
	R	W	2,8874	2,6200	2,3579	2,1040	1,8553	2,3649
		A	3,7244	3,4373	3,1598	2,8803	2,6212	3,1646
FT	M	W	1,4450	1,2638	0,9994	0,7111	0,5255	0,9890
		A	2,6667	2,4897	2,2151	1,9508	1,6741	2,1993
	R	W	1,4431	1,2608	0,9893	0,7031	0,3968	0,9586
		A	2,6671	2,4801	2,2155	1,9349	1,6683	2,1932
	Média		2,6769	2,4483	2,1755	1,9023	1,6484	$2,1702^o$

Informações sobre o fator I			Informações sobre o fator B		
Efeito I_k		IC Efeito I_k	Efeito B_m		IC Efeito B_m
M	-0,00006	(-0,00052, 0,000399)*	LU	0,5853	(0,584805, 0,585725)
R	0,00006	(-0,000399, 0,00052)*	FT	-0,5853	(-0,585725, -0,584805)
Desvio padrão s_{I_k}		0,0002	Desvio padrão s_{B_m}		0,0002

Informações sobre o fator C			Informações sobre o fator P		
Efeito C_j		IC Efeito C_j	Efeito P_i		IC Efeito P_i
W	-0,5047	(-0,505199, -0,504279)	p_1	0,5066	(0,5056, 0,5075)
A	0,5047	(0,504279, 0,505199)	p_2	0,2780	(0,2771, 0,2789)
Desvio padrão s_{C_j}		0,0002	p_4	0,0052	(0,0042, 0,0061)
Informações erros experimentais			p_8	-0,2680	(-0,2688, -0,2670)
Soma quadrado erros (SQE)		0,0608	p_{16}	-0,5219	(-0,5227, -0,5209)
Erro quadrático médio (EQM)		0,0001	Desvio padrão s_{P_i}		0,0004
Desvio padrão erros (s_e)		0,0075	*Insignificante, inclui zero. ^{o}Média μ.		

Tabela 5.8: Dados de experimentos fatoriais para LU e FT.

média geral μ é o parâmetro que estabelece o valor máximo que um efeito pode atingir, o que resulta em limites negativo e positivo para os efeitos, os quais estarão compreendidos dentro do intervalo $[-\mu, +\mu]$. Isso permite considerar que caso exista um fator com dois níveis, e um dos níveis estiver com valores nulos, o nível nulo terá efeito igual a $-\mu$ e o outro nível terá efeito igual a $+\mu$. Em uma outra situação, tendo valores semelhantes para os dois níveis, seus efeitos estarão próximos a zero. Essas hipóteses são mencionadas pelo fato de que este estudo está interessado em verificar a diferença entre o desempenho dos níveis M e R. Assim, quanto mais próximos de zero os efeitos M e R estiverem, mais semelhantes serão.

Para as aplicações SP e BT, foi obtida uma média μ de $2,0377$. Para os fatores I_k, versões M e R, os valores dos efeitos são $-0,00032$ e $+0,00032$, respectivamente. De fato, os efeitos de I_k aproximam-se de zero. Porém, apesar de mínimo, o

replicações $r = 34$			p_1	p_4	p_9	p_{16}	Média
SP	M	S	0,7263	0,3465	0,2626	0,2602	0,3989
		W	2,9056	2,3410	2,0315	1,8343	2,2781
		A	3,6716	3,1060	2,7806	2,5844	3,0357
	R	S	0,7474	0,3012	0,1779	0,1469	0,3434
		W	2,9259	2,3530	2,0290	1,8176	2,2814
		A	3,6921	3,1156	2,8017	2,5715	3,0452
BT	M	S	1,3291	0,7915	0,5798	0,4473	0,7869
		W	2,8376	2,2861	1,9855	1,7745	2,2209
		A	4,1512	3,5638	3,2763	3,0242	3,5039
	R	S	1,3513	0,8100	0,6091	0,4754	0,8114
		W	2,8580	2,2860	1,9953	1,7841	2,2308
		A	4,1779	3,5925	3,2695	3,0242	3,5160
Média			2,6145	2,0744	1,8166	1,6454	$2,0377^o$

Informações sobre o fator C		
Efeito C_j		IC Efeito C_j
S	-1,4526	(-1,453153, -1,451991)
W	0,2151	(0,214514, 0,215676)
A	1,2375	(1,236896, 1,238058)
Desvio padrão s_{C_j}		0,000296

Informações sobre o fator P		
Efeito P_i		IC Efeito P_i
p_1	0,5768	(0,576072, 0,577495)
p_4	0,0367	(0,035985, 0,037408)
p_9	-0,2211	(-0,221856, -0,220433)
p_{16}	-0,3923	(-0,393047, -0,391624)
Desvio padrão s_{P_i}		0,000363

Informações sobre o fator I			Informações sobre o fator B			Informações erros experimentais	
Efeito I_k		IC Efeito l_k	Efeito B_m		IC Efeito B_m	Soma quadrado erros (SQE)	0,113557
M	-0,00032	(-0,000736, 0,000086)*	SP	-0,14062	(-0,141027, -0,140205)	Erro quadrático médio (EQM)	0,000072
R	0,00032	(-0,000086, 0,000736)*	BT	0,14062	(0,140205, 0,141027)	Desvio padrão erros (s_e)	0,008467
Desvio padrão s_{l_k}		0,0002	Desvio padrão s_{B_m}		0,000209	* Insignificante, inclui zero.	
						^{o} Média μ.	

Tabela 5.9: Dados de experimentos fatoriais para SP e BT.

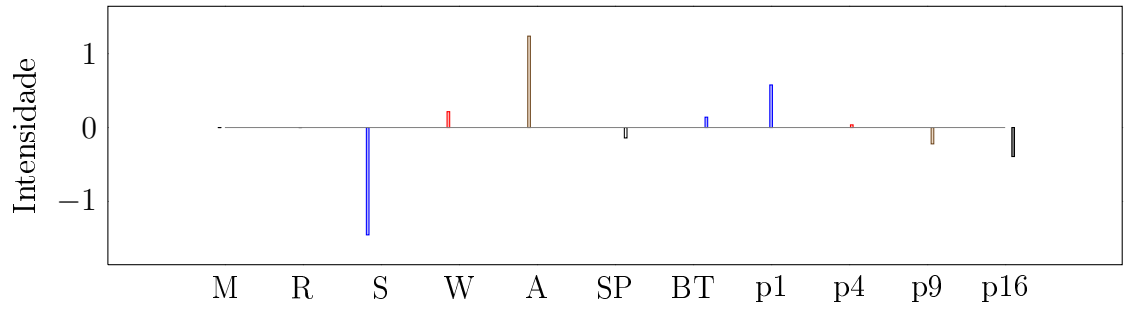


Figura 5.8: Comparação dos efeitos principais de experimentos com SP e BT.

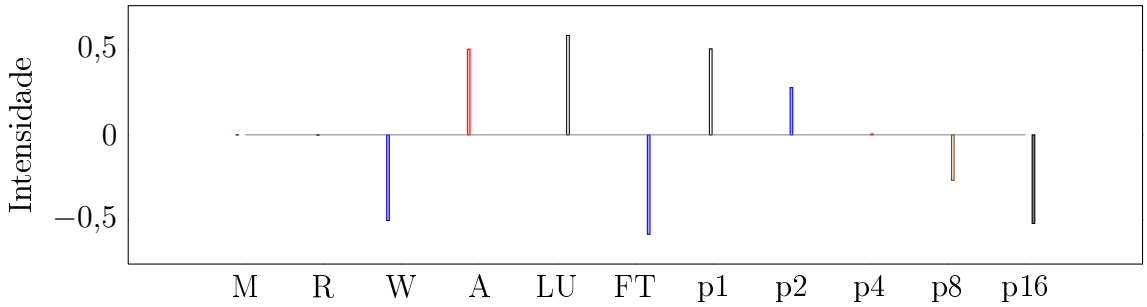


Figura 5.9: Comparação dos efeitos principais de experimentos com LU e FT.

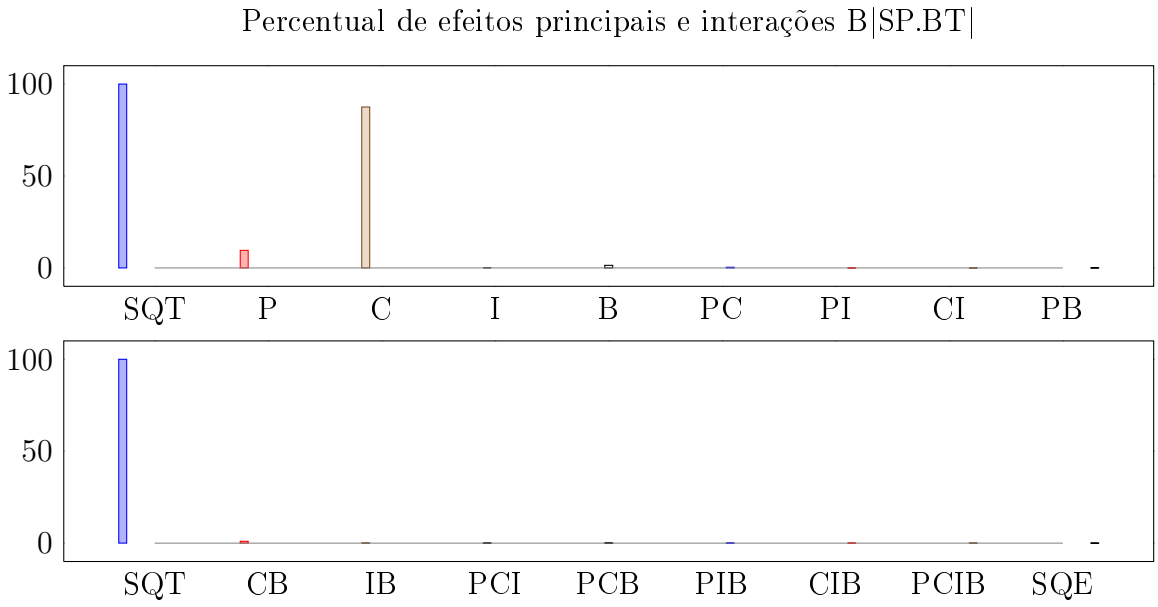
efeito negativo indica que houve um ligeiro melhor desempenho em favor da versão monolítica. O objetivo do estudo é verificar se o efeito que indica M como melhor em desempenho apresenta relevância estatística. Para isso, são utilizados intervalos de confiança para os efeitos. De acordo com os IC's dos efeitos de I_k, o valor zero está incluído nos intervalos, caracterizando como insignificante a diferença entre M e R, não permitindo afirmar se uma versão é melhor que a outra. Analisando os demais efeitos (P_i, C_j, B_m), pode-se perceber que apresentam valores significativos. O efeito C_j em S demonstra ser o menor tamanho de problema, sendo entre todos os efeitos de estudo o mais próximo de $-\mu$. Em sequencia, os mais significativos positivamente são C_j em A e P_i em p_1, o que é coerente pois representam o maior tamanho de problema e a execução com menor número de processadores. A intensidade de cada efeito pode ser observada pelo gráfico de resultados na Figura 5.8. Pelos valores dos efeitos, é possível destacar que o ápice do desempenho foi alcançado na versão monolítica (M) do programa SP sobre 16 processadores com carga de trabalho S.

Para as aplicações LU e FT, apesar de executarem em um tempo inferior a SP e BT, foi obtida uma média μ maior, pois foram utilizadas cargas de trabalho maiores (W,A). Contudo, os efeitos principais I_k diminuíram para $\mp 0,00006$, representando aproximadamente $0,002\%$ da média μ e próximo de zero, o que demonstra grande

semelhança entre as versões M e R. A versão M obteve um efeito principal I_k negativo, indicando maior desempenho dessa versão em comparação com R, embora insignificante. A insignificância é confirmada pela inclusão do valor zero no intervalo de confiança dos efeitos I_k, o que não permite apontar se uma versão é melhor do que a outra. Assim como para SP e BT, os demais efeitos P_i, C_j, B_m de LU e FT tem efeitos significativos, pois não incluem zero em seus intervalos. O efeito mínimo ficou com o B_m em FT, e máximo com LU, pois estão com efeito $\mp 0,584805$ da média $\mp\mu$. O pico de desempenho da plataforma ocorreu com a utilização da versão monolítica do programa FT sobre 16 processadores e com carga de trabalho W. A intensidade de cada efeito pode ser observada pelo gráfico de resultados na Figura 5.9.

	ANOVA: SP/BT				ANOVA: LU/FT		
Componente	Soma Quad.	%	G.L		Soma Quad.	%	G.L
SQY	9068,064		1632		6109,736		1120
SQ0	6776,570		1		5275,300		1
SQT	2291,494	100	1631		834,436	100	1119
Efeitos Princ.	2257,35	98,51	7		820,87	98,37	7
P	219,04	9,56	3		151,89	18,20	4
C	2006,04	87,54	2		285,33	34,19	1
I	0,00017	0,00001	1		0,00000	0,00000	1
B	32,269	1,40823	1		383,640	45,97589	1
Interações 1	32,272	1,40835	17		13,316	1,59576	15
PC	8,256	0,36029	6		0,075	0,00901	4
PI	0,085	0,00369	3		0,050	0,00596	4
CI	0,054	0,00238	2		0,016	0,00196	1
PB	0,764	0,03336	3		0,381	0,04569	4
CB	23,022	1,00469	2		12,699	1,52186	1
IB	0,090	0,00394	1		0,094	0,01128	1
Interações 2	1,686	0,07358	17		0,170	0,02043	13
PCI	0,021	0,00092	6		0,071	0,00846	4
PCB	1,492	0,06511	6		0,053	0,00639	4
PIB	0,044	0,00191	3		0,041	0,00490	4
CIB	0,129	0,00564	2		0,006	0,00068	1
Interações 3	0,072	0,00312	6		0,023	0,00280	4
PCIB	0,072	0,00312	6		0,023	0,00280	4
SQE	0,114	0,00496	1584		0,061	0,00728	1080

Tabela 5.10: Tabelas de análise de variância para SP/BT e LU/FT.



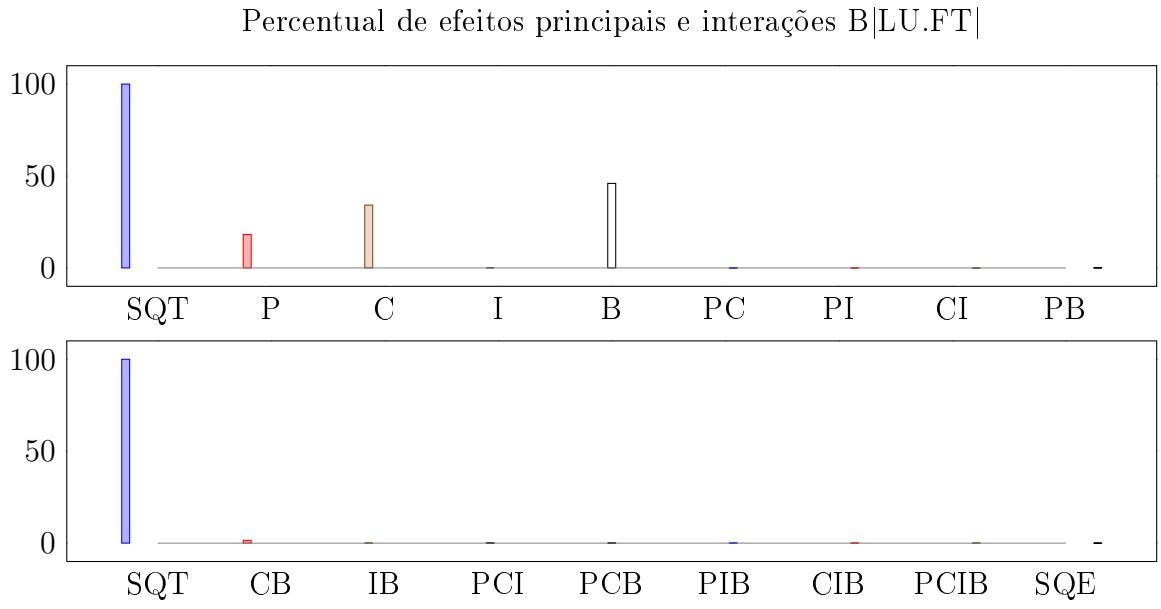
Interação PC_ij				
	p1	p4	p9	p16
S	-0,12341	-0,05956	0,04335	0,13962
W	0,05219	0,02700	-0,02135	-0,05785
A	0,07122	0,03256	-0,02200	-0,08177

Interação CB_jm		
	SP	BT
S	-0,07340	0,07340
W	0,16754	-0,16754
A	-0,09414	0,09414

Figura 5.10: Gráfico ANOVA e tabelas de interações de maior relevância para SP/BT.

Variância e Interações

De uma forma geral, para os programas SP, BT, LU e FT, as interações do Modelo 5.1 não resultaram em valores expressivos. Foi desenvolvida a Tabela 5.10 (ANOVA) para os fatores, graficamente apresentada nas Figuras 5.10 e 5.11, onde podem ser vistas as tabelas de maior relevância para as interações. As ilustrações mostram que os dados mais expressivos são definidos por CB para SP, BT, LU e FT, e PC apenas em SP e BT. CB reflete as mudanças da carga de trabalho perante cada aplicação, o que consiste em grandes variações ocorridas nas classes S, W e A. As interações CB significam que as classes e aplicações influenciaram significativamente nos resultados no sentido de que não permitiram um padrão linear à medida em que se alteram os níveis. A maior interferência no padrão linear ocorreu em W para os programas SP e BT. PC reflete as grandes variações que são geradas pelo impacto da troca de números de processadores e classes. Em PC, os maiores responsáveis pela influência dos resultados não lineares à medida em que se alteram os níveis são 1 e 16 processadores executando em carga S.



Interação CB_jm		
	LU	FT
W	0,10648	-0,10648
A	-0,10648	0,10648

Figura 5.11: Gráfico ANOVA e tabela de interação de maior relevância para LU/FT.

5.4 Conclusão do Experimento

Pelos resultados obtidos com os efeitos nos experimentos fatoriais e intervalos de confiança, não é possível afirmar se há diferença significativa entre as versões monolíticas e refatorada em componentes. As sobrecargas apresentadas nas tabelas 5.3, 5.4, 5.5, 5.6 e 5.7, de intervalos de confiança, possuem resultados significativos, porém não são homogêneos. Assim, não é possível definir um critério que estabeleça qual versão possui resultados melhores em um certo cenário (número de processadores, carga de trabalho, classe de programa), pois não se tem uma tendência padrão, alternando bons resultados entre as duas versões em diferentes experimentos. Através do estudo fatorial, houve a possibilidade de relacionar todos os experimentos, estabelecendo efeitos de cada nível de fator e verificando se o intervalo de confiança do efeito inclui o zero. Com a inclusão de zero, os efeitos são considerados insignificantes. Consequentemente, não se pode concluir se há diferença significativa entre as versões R e M.

Enfim, através da análise do percentual da sobrecarga das versões R e pela inspeção visual das curvas de desempenho de cada versão, apresentadas na Figura

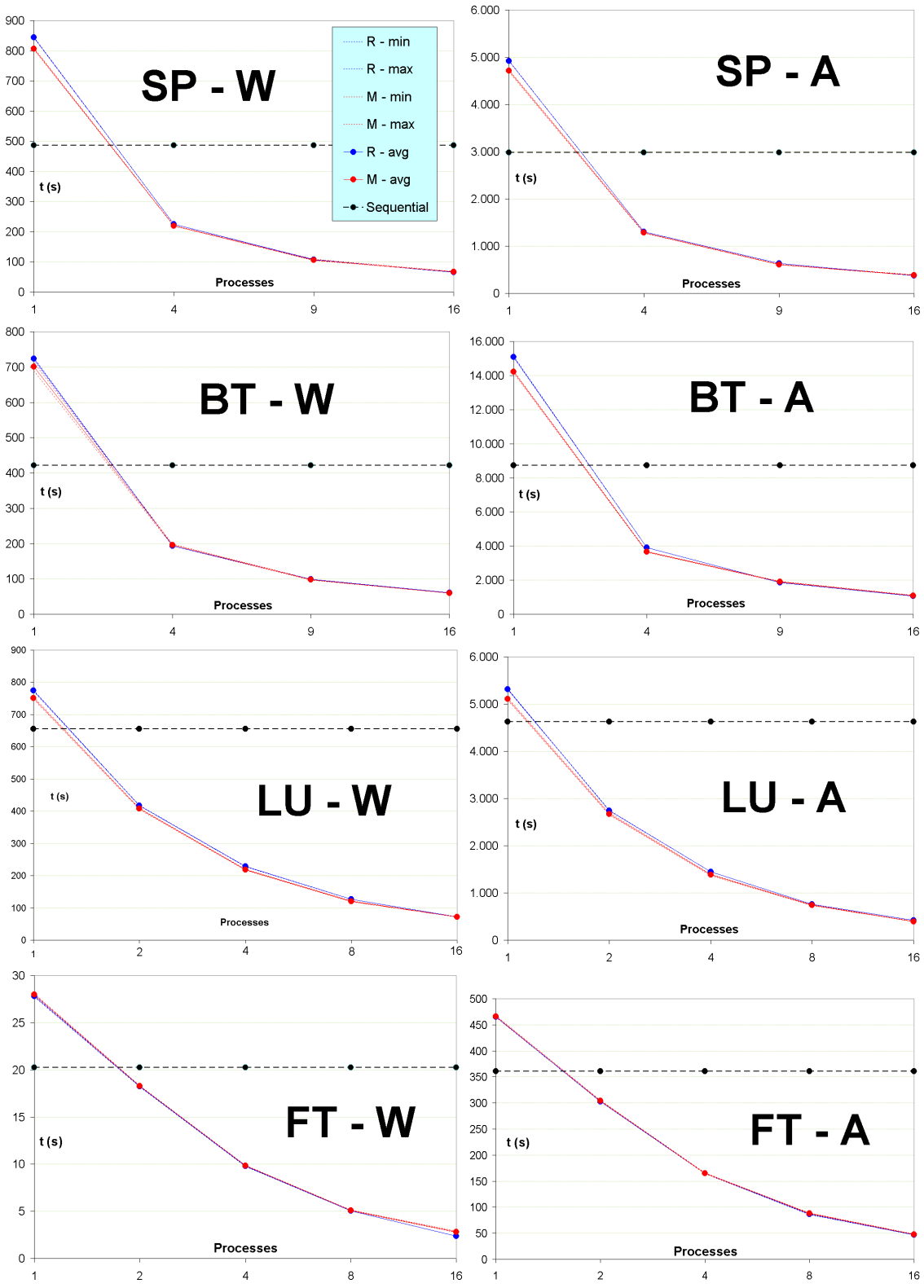


Figura 5.12: Perfis de Desempenho

5.12, observa-se que há uma proximidade muito grande dos resultados experimentais entre as versões M e R. Isso permite concluir que os desempenhos das versões monolítica e refatorada em componentes podem ser consideradas equivalentes para os cenários abordados nesse experimento, cuja generalidade permite alto potencial para extrapolação dos resultados para outros cenários de programas e plataformas de computação paralela.

Capítulo 6

Considerações Finais

Buscando viabilizar a produção de softwares em larga escala para aplicações de computação de alto desempenho (CAD), modelos e plataformas de componentes paralelos têm surgido para lidar com a complexidade de aplicações emergentes nos domínios das ciências e das engenharias, nas quais os requisitos de CAD são marcantes. Algumas soluções têm adotado os tradicionais paradigmas de orientação a processos, o que dificulta conciliar expressividade com alto nível de abstração, importante para aproximar o desenvolvimento de softwares paralelos com os artefatos modernos de engenharia de software. Alternativamente, uma nova perspectiva para a programação paralela tem surgido com base em interesses de software, ao invés de processos, onde o interesse representa a dimensão principal de composição do software. Os componentes paralelos do modelo Hash estão associados a essa perspectiva, onde processos são unidades transversais aos interesses.

O HPE (*Hash Programming Environment*) surge como o ambiente de programação Hash, fundamentado pelo modelo Hash, e que tem como característica oferecer recursos gerais de programação, com flexibilidade entre a abstração de software e expressividade do paralelismo. Entretanto, o HPE ainda não havia passado por um processo rigoroso de avaliação de desempenho, tanto no seu desempenho de processamento quanto na exploração da capacidade de desenvolvimento de aplicações científicas complexas. Esta dissertação teve como finalidade satisfazer essas necessidades e discutir temas importantes para CAD, como rigor na avaliação de desempenho e desenvolvimento de componentes reusáveis para aplicação paralela. O HPE atualmente requer que sejam implementados componentes sob a sintaxe C#, o que exigiu que os aplicativos do NPB fossem convertidos para essa linguagem. A conversão resultou em uma versão monolítica

C# do NPB, implementada segundo a estratégia original da versão nativa, utilizando apenas uma classe C# para cada aplicação. A versão monolítica possibilitou sua refatoração em componentes segundo o modelo de componentes Hash, onde foi possível conservar estruturas de dados, computações e estratégia de comunicação. Dessa forma, houve a possibilidade de medir o peso/efeito da refatoração, utilizando a técnica de medição.

Os resultados da avaliação entre a versão monolítica e refatorada permitem considerar que o custo adicional de processamento com o desenvolvimento de componentes para o HPE é estatisticamente insignificante, principalmente porque os intervalos de confiança das sobrecargas incluem zero. Os benefícios que se têm com reuso, organização de código e potencial produtividade em larga escala cobrem as insignificantes sobrecargas da refatoração. Diante disso, podemos considerar trabalhos futuros de otimizações da plataforma, especialmente pelas peculiaridades da linguagem de programação suportada, no caso C#. Como o peso da refatoração é estatisticamente insignificante, há a possibilidade de que o desempenho dos componentes seja aproximado ao desempenho dos códigos nativos, o que poderia ser realizado com a inclusão na plataforma do suporte à linguagem do código nativo, uma vez que o HPE tem previsto o suporte a múltiplas linguagens. De fato, as limitações por maior desempenho são consequência das limitações das linguagens de programação e não da estrutura do HPE, cuja sobrecarga intrínseca foi mostrada ser mínima nesta dissertação.

6.1 Trabalhos Futuros

Os principais trabalhos futuros que podem ser pesquisados a partir dos resultados desta dissertação são:

- Estudo comparativo do desempenho de linguagens de programação baseadas em máquinas virtuais dos padrões JVM e CLI, com vistas à identificação de gargalos e propostas para otimização de seu desempenho;
- Desenvolvimento de *frameworks* que encapsulam técnicas para avaliação de desempenho para plataformas de componentes;
- Implementação do suporte a novas linguagens de programação para o HPE, incluindo o suporte ao código nativo;
- Novos trabalhos de avaliação do HPE, tal como explorando detalhes acerca de sobrecargas negativas.

6.2 Trabalhos Relacionados

Os trabalhos relacionados ao tema desta dissertação têm sido desenvolvidos principalmente a partir de avaliações feitas em compiladores e arquiteturas. Existem ainda pesquisas sobre o desenvolvimento de artefatos de avaliação para análise de desempenho em dispositivos computacionais, como *frameworks* de avaliação. Os contextos encontrados nesses trabalhos discutem a utilização eficiente de técnicas de medição ou *benchmarking*, com a finalidade de qualificar, medir ou otimizar o desempenho de compiladores e plataformas. Entre os trabalhos observados, verificou-se soluções a partir dos quatro trabalhos seguintes:

i. ***Implementation of the NAS Parallel Benchmarks in Java*** [42]: o trabalho estende o desenvolvimento do ***NAS Parallel Benchmarks*** (Subseção 3.3.5), apresentando uma metodologia de conversão para *Java* dos *benchmarks* escritos em *Fortran*, com apoio de técnicas de simulação de índices de vetores. A conversão busca apresentar recursos para explorar a *JVM*[1] através de uma codificação *serializada* e outra *Multi-Threading*. Assim, o trabalho compara o desempenho da aplicação com ou sem *Threading* sob simulação de vetor multidimensional[2] através de vetor unidimensional indexado.

ii. ***Statistically Rigorous Java Performance Evaluation*** [43]: destaca que existe grande variedade de metodologias para avaliação de desempenho que são aplicadas em *Java*. Essencialmente essas metodologias fazem uso de técnicas de *benchmarking*. Discute que tal técnica necessita de atenção na medição e verificação de desempenho, sob risco de se chegar a conclusões enganosas pela não abordagem de rigor estatístico. O trabalho dá ênfase analítica ao *benchmark* (*SPECjvm98*, Seção 3.3.1).

iii. ***Performance Evaluation of the Concurrent Execution of NAS Parallel Benchmarks with BYTE Sequential Benchmarks on a Cluster*** [44]: efetua a análise de desempenho de um *Cluster* com o auxílio do NPB, executando simultaneamente aplicações paralelas e sequenciais com balanceamento de carga entre os nós do *cluster*. O trabalho busca destacar que existe uma carência de estudos experimentais para mostrar o desempenho e comportamento de aplicações paralelas e seriais operando simultaneamente.

[1] Java Virtual Machine
[2] Utilizado na versão *Fortran*

iv. ***The Performance Cockpit Approach: A Framework for Systematic Performance Evaluations*** [67]: consiste em um *framework* de avaliação de desempenho que trabalha com um conjunto de *benchmarks*, técnicas de avaliação e conceitos estatísticos. O trabalho apresenta os resultados do *framework* através de estudo de caso, discutindo sua viabilidade de aplicação. Sua abordagem é direcionada a aplicações seriais e paralelas de memória compartilhada.

Os trabalhos citados apresentam várias características que foram adotadas na pesquisa que levou a esta dissertação de Mestrado, como a conversão de *benchmarks*, a medição, o tratamento estatístico de dados e a apresentação gráfica de resultados. Entretanto, estudos com avaliação sistemática de desempenho no contexto de aplicações baseadas em componentes paralelos não têm sido frequentemente abordados, deixando uma determinada carência nessa área de pesquisa.

Referências Bibliográficas

[1] GEF-Graphical Editing Framework. http://www.eclipse.org/gef/ - Acessado em Abril/2011.

[2] Julia. http://fractal.ow2.org/julia/index.html - Acessado em Maio/2011.

[3] Object Management Group. CORBA Component. http://www.omg.org/spec/CORBA/3.1/ - Acessado em Maio/2011.

[4] OW2 Consortium. http://www.ow2.org - Acessado em Maio/2011.

[5] Project npb for hpe: Implementation of the nas parallel benchmarks for evaluating performance of hpe. http://code.google.com/p/npb-for-hpe/ - acessado em junho/2011.

[6] Projeto Mono: What is Mono. http://mono-project.com/What_is_Mono - Acessado em Março/2010.

[7] The MxN Problem in Distributed Scientific Computing. http://www.cs.indiana.edu/~febertra/mxn/.

[8] The T-Distribution and Its Use in Hypothesis Testing. http://simon.cs.vt.edu/SoSci/converted/T-Dist/activity.html - Acessado em Novembro/2010.

[9] On the roles of simulation, analytical modeling, and measurement in solving complex problems (panel). In *Proceedings of the 17th Conference on Winter Simulation* (New York, NY, USA, 1985), WSC '85, ACM, pp. 136–. Chairman-Shub, Charles M.

[10] Albuquerque, J. P., Fortes, J. M., and Finamore, W. A. *Probabilidade, Variáveis Aleatórias e Processos Estocásticos*. PUC-RIO, Rio de Janeiro, RJ, Brasil, 2008.

[11] Allan, B. A., Armstrong, R. C., Wolfe, A. P., Ray, J., Bernholdt, D. E., and Kohl, J. A. The cca core specification in a distributed memory spmd framework. *Concurrency and Computation: Practice and Experience*.

[12] Armstrong, R., Kumfert, G., McInnes, L. C., Parker, S., Allan, B., Sottile, M., Epperly, T., and Dahlgren, T. The cca component model for high-performance scientific computing. *Concurr. Comput. : Pract. Exper. 18*, 2 (February 2006), 215–229.

[13] Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., Patterson, D. A., Plishker, W. L., Shalf, J., Williams, S. W., and Yelick, K. A. The landscape of parallel computing research: A view from berkeley. Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[14] Baduel, L., Baude, F., and Caromel, D. Asynchronous Typed Object Groups for Grid Programming. *Journal of Parallel Programming 35*, 6 (dec 2007), 573–613.

[15] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V., and Weeratunga, S. K. The nas parallel benchmarkssummary and preliminary results. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 1991), Supercomputing '91, ACM, pp. 158–165.

[16] Baude, F., Caromel, D., Dalmasso, C., Danelutto, M., Getov, W., Henrio, L., and Prez, C. GCM: A Grid Extension to Fractal for Autonomous Distributed Components. *Annals of Telecommunications 64*, 1 (2009), 5–24.

[17] Baude, F., Caromel, D., Henrio, L., and Morel, M. Collective Interfacess for Distributed Components. In *7th International Symposium on Cluster Computing and the Grid (CCGrid 07)* (2007), IEEE Computer Society.

[18] Baude, F., Caromel, D., and Morel, M. From distributed objects to hierarchical grid components. In *In International Symposium on Distributed Objects and Applications (DOA* (2003), Springer-Verlag, pp. 1226–1242.

[19] Bentley, C., Watterson, S. A., Lowenthal, D. K., and Rountree, B. Implicit Array Bounds Checking on 64-bit Architectures. *ACM Transactions on Architecture and Code Optimization (TACO) 3*, 4 (2006), 502–527.

[20] Bertran, F., Bramley, R., Sussman, A., Bernholdt, D. E., Kohl, J. A., Larson, J. W., and Damevski, K. B. Data Redistribution and Remote Method Invocation in Parallel Component Architectures. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (Apr. 2005), IEEE.

[21] Bertrand, F., and Bramley, R. DCA: a distributed CCA framework based on MPI. In *Proceedings of the HIPS2004 - 9th International Workshop on Highlevel Parallel Programming Models and Supportive Environments* (2004).

[22] Bolch, G., and et al. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, Second Edition.* LNCS. Wiley Inter-Science, 2006.

[23] Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., and Stefani, J.-B. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper. 36*, 11-12 (September 2006), 1257–1284.

[24] Bruneton, E., Coupaye, T., and Stefani, J. B. Recursive and dynamic software composition with sharing. In *European Conference on Object Oriented Programming (ECOOP'2002)* (2002).

[25] Buchholz, W. A synthetic job for measuring system performance. *IBM Syst. J. 8*, 4 (December 1969), 309–318.

[26] Cantonnet, F., Yao, Y., Annareddy, S., Mohamed, A. S., and El-Ghazawi, T. A. Performance monitoring and evaluation of a upc implementation on a numa architecture. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing* (Washington, DC, USA, 2003), IPDPS '03, IEEE Computer Society, pp. 274.2–.

[27] Carvalho Junior, F. H. Computação de Alto Desempenho em Plataforma Windows, 2007.

[28] Carvalho Junior, F. H., and Corrêa, R. C. The Design of a CCA Framework with Distribution, Parallelism, and Recursive Composition. In *CBHPC* (2010).

[29] Carvalho Junior, F. H., and Lins, R. D. Haskell$_{\#}$: Parallel Programming Made Simple and Efficient. 776–794.

[30] Carvalho Junior, F. H., and Lins, R. D. Haskell$_{\#}$: Parallel Programming Made Simple and Efficient. *Journal of Universal Computer Science 9*, 8 (aug 2003), 776–794.

[31] Carvalho Junior, F. H., and Lins, R. D. Separation of Concerns for Improving Practice of Parallel Programming. *INFORMATION, An International Journal 8*, 5 (Sept. 2005).

[32] Carvalho Junior, F. H., and Lins, R. D. An Institutional Theory for #-Components. *Electronic Notes in Theoretical Computer Science 195* (Jan. 2008), 113–132.

[33] Carvalho Junior, F. H., Lins, R. D., Corrêa, R. C., and Araújo, G. A. Towards an Architecture for Component-Oriented Parallel Programming: Research Articles. *Concurr. Comput. : Pract. Exper. 19*, 5 (April 2007), 697–719.

[34] Carvalho Junior, F. H., Lins, R. D., Corrêa, R. C., Araújo, G. A., and Silva, J. C. On The Design of Abstract Binding Connectors for High Performance Computing Component Models. In *Proceedings of The 2007 Symposium on Component and Framework Technology in High-Performance and Scientific Computing* (New York, NY, USA, 2007), CompFrame '07, ACM, pp. 67–76.

[35] Cole, M. *Algorithm Skeletons: Structured Management of Paralell Computation*. Pitman, 1989.

[36] Cole, M. Bringing Skeletons Out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming. *Parallel Comput. 30*, 3 (2004), 389–406.

[37] Colella, P. Defining Software Requirements for Scientific Computing, Presentation, 2004.

[38] Damevski, K., Zhang, K., and Parker, S. Practical parallel remote method invocation for the babel compiler. In *Proceedings of the 2007 Symposium on Component and Framework Technology in High-Performance and Scientific Computing* (New York, NY, USA, 2007), ACM, pp. 131–140.

[39] Dijkstra, E. W. The humble programmer. *Commun. ACM 15* (October 1972), 859–866.

[40] Dixit, K., and et al. Standard Performance Evaluation Corporation (SPEC). http://www.spec.org/ - Acessado em Outubro/2010, 2010.

[41] Frisina, J. N. Metrics for microprogrammed instruction sets. *SIGMICRO Newsl. 8*, 4 (December 1977), 21–23.

[42] Frumkin, M. A., and et al. Implementation of The NAS Parallel Benchmarks in Java. http://www.nas.nasa.gov/News/Techreports/2002/PDF/nas-02-009.pdf - Acessado em Outubro/2010, 2009.

[43] Georges, A., Buytaert, D., and Eeckhout, L. Statistically rigorous java performance evaluation. *SIGPLAN Not. 42*, 10 (October 2007), 57–76.

[44] Goscinski, A. M., and Wong, A. K. L. Performance evaluation of the concurrent execution of nas parallel benchmarks with byte sequential benchmarks on a cluster. *Parallel and Distributed Systems, International Conference on 1* (2005), 313–319.

[45] Grama, A., Gupta, A., Karypis, G., and Kumar, V. *Introduction to Parallel Computing*. Addison Wesley, US, 2003.

[46] Gregor, D., and Lumsdaine, A. Design and implementation of a high-performance mpi for c# and the common language infrastructure. In *Proceedings of The 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2008), PPoPP '08, ACM, pp. 133–142.

[47] Jain, R. *The Art Of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, INC, 1991.

[48] Jin, H., and Van der Wijngaart, R. F. Performance characteristics of the multi-zone nas parallel benchmarks. *J. Parallel Distrib. Comput. 66*, 5 (May 2006), 674–685.

[49] John, L. K. Performance Evaluation: Techniques, Tools and Benchmarks. http://projects.ece.utexas.edu/ece/lca/ps/john_perfeval.pdf - Acessado em Setembro/2010, 2001.

[50] Johnson, C., Parker, S., and Weinstein, D. Component-based problem solving environments for large-scale scientific computing, 2002.

[51] Kerley, L. M. Comprehending the central limit theorem. *SIGCSE Bull. 20*, 2 (June 1988), 20–25.

[52] Krishnan, S., and Gannon, D. XCAT3: a framework for CCA components as OGSA services. In *Proceedings of the HIPS2004 - 9th International Workshop on Highlevel Parallel Programming Models and Supportive Environments* (2004).

[53] Kuchen, H., and Cole, M. e. Algorithm Skeletons. *Parallel Computing 32* (2006), 447–626.

[54] León, R. V. The Concept of Confidence Interval for the Mean. http://web.utk.edu/~ leon/stat201/Confidence Interval Concept.html - Acessado em Abril/2011.

[55] Miao, W., and Chiou, P. Confidence intervals for the difference between two means. *Comput. Stat. Data Anal. 52* (January 2008), 2238–2248.

[56] Nguyen, T. V. N., and Irigoin, F. Efficient and effective array bound checking. *ACM Trans. Program. Lang. Syst. 27*, 3 (May 2005), 527–570.

[57] Onaga, K., and Takechi, T. On design of rotary array communication and wavefront-driven algorithms for solving large-scale band-limited matrix equations. In *Proceedings of The 13th Annual International Symposium on Computer Architecture* (Los Alamitos, CA, USA, 1986), ISCA '86, IEEE Computer Society Press, pp. 308–315.

[58] Patil, S., and Lilja, D. J. Using resampling techniques to compute confidence intervals for the harmonic mean of rate-based performance metrics. *IEEE Computer Architecture Letters 9* (2010), 1–4.

[59] Pozo, R., and Miller, B. SciMark. http://math.nist.gov/scimark2 - Acessado em Outubro/2010, 2010.

[60] Ross, P. J. *Taguchi Techniques for Quality Engineering.* McGraw-Hill, New York, NY, USA, 1996.

[61] Silva, J. C. Infra-estrutura de Componentes Paralelos para Aplicações de Computação de Alto Desempenho. Master's thesis, Universidade Federal do Ceará, Fortaleza-CE, Junho 2008.

[62] Sokolowski, P. J., and Grosu, D. Performance of the nas parallel benchmarks on grid enabled clusters. In *Proceedings of the Network Computing and Applications, Third IEEE International Symposium* (Washington, DC, USA, 2004), NCA '04, IEEE Computer Society, pp. 356–361.

[63] Sommerville, L. *Software Engineering, Eighth Edition.* Addison Wesley, US, 2007.

[64] Student. The probable error of a mean. *Biometrika 6*, 1 (1908), 1–25.

[65] Temam, R. *Navier-Stokes Equations: Theory and Numerical Analysis, Revised Edition*, vol. 2. Elsevier Science Ltd, 1979.

[66] Wang, A. J. A., and Qian, K. *Component-Oriented Programming.* LNCS. Wiley Inter-Science, 2005.

[67] Westermann, D., Happe, J., Hauck, M., and Heupel, C. The performance cockpit approach: A framework for systematic performance evaluations. In *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications* (Los Alamitos, CA, USA, 2010), vol. 0, IEEE Computer Society, pp. 31–38.

[68] Whittaker, J. A., and Thomason, M. G. A markov chain model for statistical software testing. *IEEE Trans. Softw. Eng. 20* (October 1994), 812–824.

[69] Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., and Gupta, A. The splash-2 programs: Characterization and methodological considerations. *SIGARCH Comput. Archit. News 23*, 2 (May 1995), 24–36.

[70] Xi, H., and Pfenning, F. Eliminating array bound checking through dependent types. *SIGPLAN Not. 33*, 5 (May 1998), 249–257.

Apêndice A

Códigos Fonte

Neste apêndice, apresentam-se os códigos-fonte e conceitos relacionados às seções desenvolvidas ao longo da dissertação.

```
A.  Original

01.  for(k=1;k<=nz2;k++)
02.    for(j=1;j<=ny2;j++)
03.      for(i=1;i<=nx2;i++)
04.        for(m=0;m<=4;m++)
05.          u[m+i*isize1+j*jsize1+k*ksize1] += rhs[m+i*isize1+j*jsize1+k*ksize1];

B.  Intermediário

01.  for(k=1;k<=nz2;k++)
02.    for(j=1;j<=ny2;j++)
03.      for(i=1;i<=nx2;i++)
04.        for(m=0;m<=4;m++)
05.          u[m,i,j,k] +=rhs[m,i,j,k];

C.  Final

01.  for(k=1;k<=nz2;k++)
02.    for(j=1;j<=ny2;j++)
03.      for(i=1;i<=nx2;i++)
04.        for(m=0;m<=4;m++)
05.          u[k,j,i,m] +=rhs[k,j,i,m];
*  isize1, jsize1 e ksize1 representam o limite de cada posição(i,j,k) do vetor
  multidimensional
```

Figura A.1: *Conversão NPB de vetor unidimensional para matriz multidimensional*

$$\begin{vmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{vmatrix} = \begin{vmatrix} a11 & 0 & 0 \\ 0 & a22 & 0 \\ 0 & 0 & a33 \end{vmatrix} + \begin{vmatrix} 0 & 0 & 0 \\ a21 & 0 & 0 \\ a31 & a32 & 0 \end{vmatrix} + \begin{vmatrix} 0 & a12 & a13 \\ 0 & 0 & a23 \\ 0 & 0 & 0 \end{vmatrix}$$

M D Y Z

Figura A.2: Triangular: Diagonal+Lower+Upper.

```
01.  se(P0 ou P1)
02.    para (i=0;i<N/2;i++){
03.      V[i] = 0;
04.        para (j=0;j<N;j++)
05.          V[i] += A[i,j]*X[j]
06.    }
07.  senão se(P2 || P3)
08.    para (i=0;i<N/2;i++){
09.      U[i] = 0;
10.        para (j=0;j<N;j++)
11.          U[i] += B[i,j]*Y[j]
12.    }
13.  redistribuir(U)
14.  redistribuir(V)
15.  para (i=0;i<N/4;i++)
16.    r += V[i]*U[i]
17.  reduce(r)
```

Figura A.3: *Multiplicação paralela de matriz e vetor para quatro processos*

Apêndice B

Tabelas Estatísticas

Neste apêndice, apresentam-se as tabelas de distribuição utilizadas frequentemente na estatística.

B.1 Distribuição z

Exemplo de busca do valor z_p: para um intervalo com 90% de confiança, $\alpha = 0.1$ e $p = 1 - \alpha/2 = 0.95$. $z_{0.95}$ corresponde a 0.000 e 0.95, ou seja, $z_{0.95} = 1,645$.

p	**0.00**	**0.01**	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**
0.5	0.000	0.025	0.050	0.075	0.100	0.126	0.151	0.176	0.202	0.228
0.6	0.253	0.279	0.305	0.332	0.358	0.385	0.412	0.440	0.468	0.496
0.7	0.524	0.553	0.583	0.613	0.643	0.674	0.706	0.739	0.772	0.806
0.8	0.842	0.878	0.915	0.954	0.994	1.036	1.080	1.126	1.175	1.227
p	**0.000**	**0.001**	**0.002**	**0.003**	**0.004**	**0.005**	**0.006**	**0.007**	**0.008**	**0.009**
0.90	1.282	1.287	1.293	1.299	1.305	1.311	1.317	1.323	1.329	1.335
0.91	1.341	1.347	1.353	1.359	1.366	1.372	1.379	1.385	1.392	1.398
0.92	1.405	1.412	1.419	1.426	1.433	1.440	1.447	1.454	1.461	1.468
0.93	1.476	1.483	1.491	1.499	1.506	1.514	1.522	1.530	1.538	1.546
0.94	1.555	1.563	1.572	1.580	1.589	1.598	1.607	1.616	1.626	1.635
0.95	1.645	1.655	1.665	1.675	1.685	1.695	1.706	1.717	1.728	1.739
0.96	1.751	1.762	1.774	1.787	1.799	1.812	1.825	1.838	1.852	1.866
0.97	1.881	1.896	1.911	1.927	1.943	1.960	1.977	1.995	2.014	2.034
0.98	2.054	2.075	2.097	2.120	2.144	2.170	2.197	2.226	2.257	2.290
p	**0.0000**	**0.0001**	**0.0002**	**0.0003**	**0.0004**	**0.0005**	**0.0006**	**0.0007**	**0.0008**	**0.0009**
0.990	2.326	2.330	2.334	2.338	2.342	2.346	2.349	2.353	2.357	2.362
0.991	2.366	2.370	2.374	2.378	2.382	2.387	2.391	2.395	2.400	2.404
0.992	2.409	2.414	2.418	2.423	2.428	2.432	2.437	2.442	2.447	2.452
0.993	2.457	2.462	2.468	2.473	2.478	2.484	2.489	2.495	2.501	2.506

0.994	2.512	2.518	2.524	2.530	2.536	2.543	2.549	2.556	2.562	2.569
0.995	2.576	2.583	2.590	2.597	2.605	2.612	2.620	2.628	2.636	2.644
0.996	2.652	2.661	2.669	2.678	2.687	2.697	2.706	2.716	2.727	2.737
0.997	2.748	2.759	2.770	2.782	2.794	2.807	2.820	2.834	2.848	2.863
0.998	2.878	2.894	2.911	2.929	2.948	2.968	2.989	3.011	3.036	3.062
0.999	3.090	3.121	3.156	3.195	3.239	3.291	3.353	3.432	3.540	3.719

Tabela B.1: Distribuição z

B.2 Distribuição t

Exemplo de busca de valor $t[p; n-1]$ com $n = 14$ e $p = 1 - \frac{0.1}{2}$ para 90%: $t[1 - \frac{0.1}{2}; 14-1] = t[0.95; 13]$. O valor na tabela que representa $t[0.95; 13]$ é 1.771.

n	0.6000	0.7000	0.8000	0.9000	0.9500	0.9750	0.9950	0.9995
1	0.325	0.727	1.377	3.078	6.314	12.706	63.657	636.619
2	0.289	0.617	1.061	1.886	2.920	4.303	9.925	31.599
3	0.277	0.584	0.978	1.638	2.353	3.182	5.841	12.924
4	0.271	0.569	0.941	1.533	2.132	2.776	4.604	8.610
5	0.267	0.559	0.920	1.476	2.015	2.571	4.032	6.869
6	0.265	0.553	0.906	1.440	1.943	2.447	3.707	5.959
7	0.263	0.549	0.896	1.415	1.895	2.365	3.499	5.408
8	0.262	0.546	0.889	1.397	1.860	2.306	3.355	5.041
9	0.261	0.543	0.883	1.383	1.833	2.262	3.250	4.781
10	0.260	0.542	0.879	1.372	1.812	2.228	3.169	4.587
11	0.260	0.540	0.876	1.363	1.796	2.201	3.106	4.437
12	0.259	0.539	0.873	1.356	1.782	2.179	3.055	4.318
13	0.259	0.538	0.870	1.350	1.771	2.160	3.012	4.221
14	0.258	0.537	0.868	1.345	1.761	2.145	2.977	4.140
15	0.258	0.536	0.866	1.341	1.753	2.131	2.947	4.073
16	0.258	0.535	0.865	1.337	1.746	2.120	2.921	4.015
17	0.257	0.534	0.863	1.333	1.740	2.110	2.898	3.965
18	0.257	0.534	0.862	1.330	1.734	2.101	2.878	3.922
19	0.257	0.533	0.861	1.328	1.729	2.093	2.861	3.883
20	0.257	0.533	0.860	1.325	1.725	2.086	2.845	3.850
21	0.257	0.532	0.859	1.323	1.721	2.080	2.831	3.819

22	0.256	0.532	0.858	1.321	1.717	2.074	2.819	3.792
23	0.256	0.532	0.858	1.319	1.714	2.069	2.807	3.768
24	0.256	0.531	0.857	1.318	1.711	2.064	2.797	3.745
25	0.256	0.531	0.856	1.316	1.708	2.060	2.787	3.725
26	0.256	0.531	0.856	1.315	1.706	2.056	2.779	3.707
27	0.256	0.531	0.855	1.314	1.703	2.052	2.771	3.690
28	0.256	0.530	0.855	1.313	1.701	2.048	2.763	3.674
29	0.256	0.530	0.854	1.311	1.699	2.045	2.756	3.659
30	0.256	0.530	0.854	1.310	1.697	2.042	2.750	3.646
60	0.254	0.527	0.848	1.296	1.671	2.000	2.660	3.460
90	0.254	0.526	0.846	1.291	1.662	1.987	2.632	3.402
120	0.254	0.526	0.845	1.289	1.658	1.980	2.617	3.373

Tabela B.2: Distribuição t