Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Proposta de um sistema de modelagem e predição analítica de desempenho para uma plataforma de processamento paralelo

Autor: Roberto Hirochi Herai
Orientador: Prof. Marco Aurélio Amaral Henriques

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: **Engenharia de Computação.**

Banca Examinadora

Prof. Dr. Marco Aurélio Amaral Henriques (Presidente) . DCA/FEEC/Unicamp
Prof. Dr. Léo Pini Magalhães ..... DCA/FEEC/Unicamp
Prof. Dr. Mario Jino ..... DCA/FEEC/Unicamp
Prof. Dr. Wagner Meira Jr. ..... DCC/UFMG

Campinas, SP - dezembro/2005

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Herai, Roberto Hirochi
H412p Proposta de um sistema de modelagem e predição analítica de desempenho para uma plataforma de processamento paralelo / Roberto Hirochi Herai. -- Campinas, SP: [s.n.], 2005.

Orientador: Marco Aurélio Amaral Henriques
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Processamento paralelo (Computadores). 2. Previsão. 3. Modelagem de dados. I. Marco Aurélio Amaral Henriques. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Proposal of a performance modeling and analytical prediction system for a parallel processing system
Palavras-chave em Inglês: Parallel processing, Compilers, Analytical performance prediction, Parallel application modeling
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Léo Pini Magalhães, Mário Jino e Wagner Meira Junior
Data da defesa: 16/12/2005

Resumo

A predição de desempenho é um importante mecanismo para avaliar a utilização de recursos e estimar o tempo de execução de aplicações em sistemas paralelos. Este trabalho apresenta uma ferramenta que permite criar modelos que representam características da aplicação e dos computadores utilizados para processá-la. Tais modelos são combinados para gerar um modelo de desempenho mais abrangente, cuja análise permite obter estimativas de tempo de execução que contemplem fatores de atraso tais como operações aritméticas sobre diferentes tipos de dados e efeitos de contenção, causados por concorrência. As estimativas podem ser geradas em poucos segundos e permitem analisar o impacto causado na aplicação pela utilização de diferentes configurações do sistema de processamento paralelo. A ferramenta foi implementada para um sistema de processamento paralelo baseado em Java, chamado JoiN, e ela mostrou ser possível obter estimativas satisfatórias de tempos de execução para diversos tipos de aplicações paralelas.

Palavras-chave: processamento paralelo, compiladores, predição analítica de desempenho, modelagem de aplicações paralelas.

Abstract

Performance prediction is an important mechanism to evaluate the use of resources and predict the execution time of applications in parallel systems. This work presents a tool for the creation of models that represent applications and computers characteristics. Such models are combined to generate a performance model, whose analysis derives execution time estimates that include delay factors, such as arithmetic operations on different data types and contention effects caused by process concurrence. The estimates can be generated in a few seconds and allow the analysis of the impact caused in the application by changes in the parallel system configuration. The tool was implemented for a Java based parallel processing system called JoiN and it showed that satisfactory execution time estimates can be obtained for several types of parallel applications.

Keywords: parallel processing, compilers, analytical performance prediction, parallel application modeling.

À minha mãe Setsu e ao meu irmão Renato, que sempre farão falta em minha vida.

Agradecimentos

Ao meu orientador, Prof. Marco Aurélio Amaral Henriques, pela oportunidade, amizade, orientação e principalmente pelo fantástico espírito de pesquisa do qual sempre terei respeito e muita admiração. Ética e competência fazem parte dele, e me orgulho de ter tido a oportunidade de conviver nesses poucos anos como seu aluno. Agradeço pelas longas conversas e histórias a respeito do Japão, as quais fizeram-me lembrar mais de minhas origens e de parte da minha família que está lá.

Aos colegas e amigos do grupo de pesquisa e desenvolvimento da plataforma de processamento paralelo JoiN, pelas discussões, críticas, sugestões e motivações dadas desde o início do desenvolvimento deste trabalho. Vocês foram muito importantes para a concretização desta dissertação e de todo trabalho resultante dela.

Aos colegas e verdadeiros amigos que fiz na Faculdade de Engenharia Elétrica e de Computação, na Unicamp.

Ao amigo Eduardo Nicola, pelos conselhos e oportunidades profissionais que me permitiram dedicar ao trabalho de forma mais tranquila.

Aos amigos Marcos Hamanaka, Maurício Duarte, Edson Donizete, Fabiano Pádua e José Carlos, pela amizade, conselhos e por me ajudar expontaneamente nos momentos em que precisei de ajuda. Agradeço a amizade de todos e serei eternamente grato por isso.

À família de meus amigos Helmann e Rafael, que ajudaram a tornar a cidade de Campinas minha segunda casa.

Aos membros de minha família que, apesar das perdas, jamais mediram esforços para ajudar uns aos outros.

Ao meu irmão Paulo Yoshitomo, que abdicou de parte de sua vida para ajudar a família em momentos de dificuldades.

À CAPES, pelo apoio financeiro.

Sumário

Lista de Figuras

Lista de Tabelas

Siglas utilizadas no texto

Sigla	Significado
SPMD	Single Program Multiple Data
FIFO	First In First Out
PeMPSys	Performance Modeling and Prediction System
ARC	Analytical Representation Compiler
CRES	Computational Resource Evaluation Service
APET	Analytical Performance Estimation Tool
PAS	Parallel Application Specification
SCAn	Source Code Analyser
PASC	Parallel Application Specification Compiler
AMGen	Application Model Generator
ASA	Árvore de Sintaxe Abstrata
AMC	Application Model Compiler
TPT	Time Prediction Tool
SSE	Streaming SIMD Extensions
MMX	Multimedia Extensions

Artigos derivados deste trabalho

1. R. H. Herai, M. A. A. Henriques. "Ferramentas de Modelagem e Predição de Performance Analítica em uma Plataforma de Processamento Paralelo.". *V Workshop em Sistemas Computacionais de Alto Desempenho* (WSCAD'2004), Foz do Iguaçu, Paraná, Brasil, pg. 3-10, Outubro 2004.

2. R. H. Herai, M. A. A. Henriques. "Uma Ferramenta para Predição Analítica de Desempenho de Aplicações Java Paralelas.". *VI Workshop em Sistemas Computacionais de Alto Desempenho* (WSCAD'2005), Rio de Janeiro, Rio de Janeiro, Brasil, pg. 73-80, Outubro 2005.

Capítulo 1

Introdução

As plataformas de processamento paralelo que operam sobre a infra-estrutura disponibilizada pela Internet são exemplos de uma tecnologia em ampla expansão. Neste capítulo serão vistos alguns exemplos destas plataformas, como elas funcionam e a forma com que alguns sistemas fornecem diferentes funcionalidades a elas. Entre estes sistemas, estão os de predição de desempenho, os quais utilizam alguma técnica para estimar o tempo que uma aplicação levará para ser executada em uma plataforma paralela.

1.1 A Internet como fonte de recursos na execução de aplicações

O grande desenvolvimento científico e tecnológico das últimas décadas permitiu o surgimento e aperfeiçoamento de inúmeras linhas de pesquisa de caráter teórico e prático. Estudos que no passado eram realizados estritamente na teoria, ou em pequenos laboratórios, hoje são realizados com fins comerciais e desenvolvidos por empresas, centros de pesquisa ou universidades, tanto estatais como privados.

Uma das áreas que mais têm se destacado nos últimos anos é a de comunicação de dados, principalmente com a popularização da Internet. Ela permite interconectar uma grande quantidade de computadores geograficamente distribuídos para formar uma grande rede de âmbito mundial. Nos dias de hoje não é difícil encontrar uma escola, um centro de pesquisa, uma empresa, um estabelecimento comercial ou residências com pelo menos um computador conectado à Internet. Em função disso, muitos recursos e esforços vêm sendo investidos para a construção e aperfeiçoamento de tecnologias que façam uso de toda ou parte da infra-estrutura fornecida pelos computadores interligados através da Internet.

Os grids computacionais, também conhecidos como plataformas de processamento paralelo sobre a Internet ou simplesmente plataformas de processamento paralelo, são conseqüência disso. Alguns deles são propostos com o objetivo de aproveitar, por meio de softwares, a ociosidade de milhares de computadores distribuídos pelo mundo e interligados por diferentes tipos de rede [BB00].

A Fig. 1.1 exemplifica a estrutura de uma plataforma de processamento paralelo fictícia, a qual é composta por computadores de vários tipos conectados pela Internet. Nota-se que há computadores que dispõem de grande capacidade para armazenamento de dados (a), uma rede formada por máquinas homogêneas (b), um computador com grande capacidade de processamento (c) e um palmtop (d), dispositivo com capacidades de processamento e armazenamento limitados.

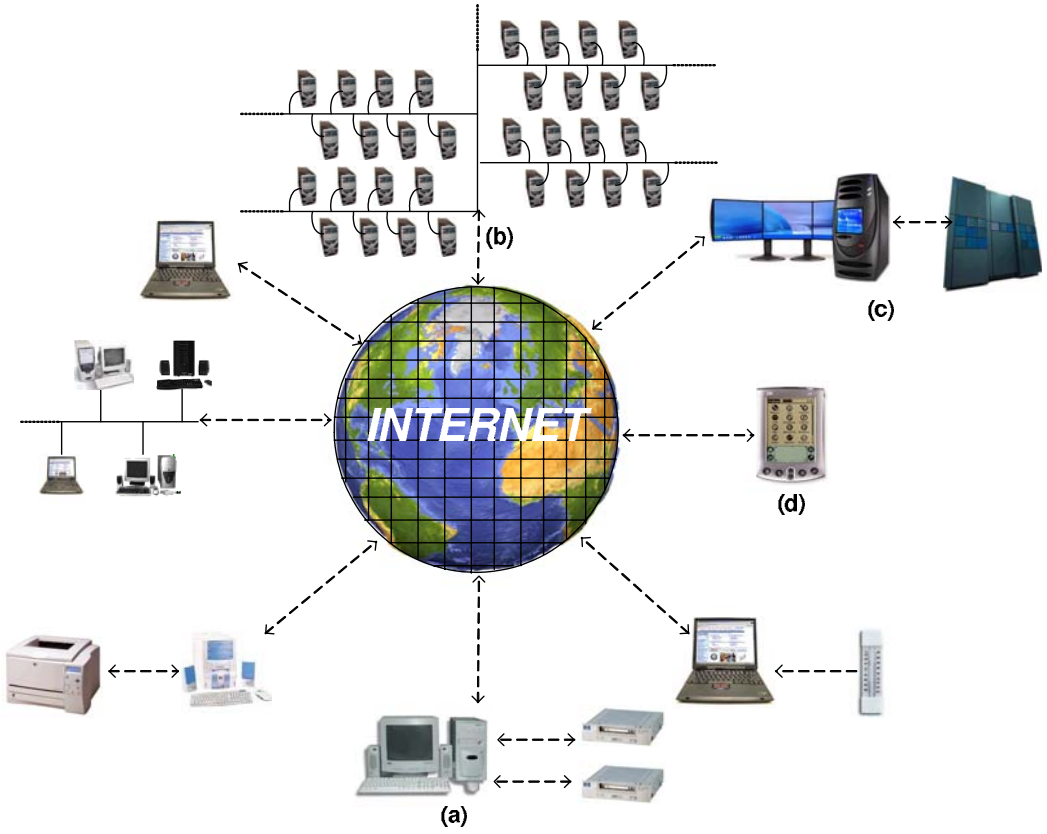

Fig. 1.1: Recursos de uma plataforma de processamento paralelo

O uso de plataformas de processamento paralelo permite aproveitar os computadores ociosos disponíveis, que poderiam estar sendo sub-utilizados, para diminuir o tempo de execução de um programa paralelo, cujas tarefas, as quais são partes de um programa que é dividido em múltiplos fragmentos, possam ser executadas simultaneamente em computadores distintos. Com as plataformas de processamento paralelo vários benefícios são possíveis, como, por exemplo, uma empresa no Brasil utilizar os computadores na madrugada do Japão, onde estes computadores poderiam estar ociosos e vice-versa.

Atualmente, o uso de plataformas de processamento paralelo é feito em diversas áreas para a solução de uma grande quantidade de problemas [FK04]. Exemplos de utilização de tais plataformas podem ser encontrados em áreas como cinema (criação de filmes e animações em 3D), criptografia (quebra de código criptografado), meteorologia (modelagem do clima), biologia (seqüenciamento de

cadeias de DNA), jogos (cenários e animações), engenharia (simulação de processos), matemática (resolução de problemas complexos) e economia (simulação de mercados).

Apesar da ampla aplicabilidade das plataformas de processamento paralelo, seu uso na prática não é uma tarefa tão simples, pois é difícil dividir um programa em múltiplos fragmentos, chamados de tarefas, de forma que computadores distintos executem diferentes porções do mesmo, sem que ocorra alguma perda de eficiência. Além disso, a alta latência de comunicação na Internet restringe as aplicações de suas plataformas de processamento paralelo àquelas que não necessitam de muita comunicação entre as tarefas durante a execução. Desta forma, um software baseado em uma plataforma de processamento paralelo sobre a internet precisa ser bastante sofisticado para controlar a distribuição e a execução de aplicações sobre os computadores que, via de regra, são distintos entre si.

1.2 Plataformas de processamento paralelo

Em função das grandes vantagens apresentadas com o uso de plataformas de processamento paralelo, existem atualmente várias pesquisas no desenvolvimento de softwares baseados em tais plataformas com diferentes características e estratégias na utilização dos computadores para a execução de aplicações [BBL02].

A plataforma de processamento paralelo Condor [LLM88] [FK99] [FMM00] [TTL02] é uma das mais antigas. Desenvolvida na Universidade de Wisconsin, sua origem data de 1985 e ela foi derivada de outro projeto chamado *Remote Unix*. Seu foco principal é o gerenciamento e monitoramento dos recursos disponíveis em uma plataforma de processamento paralelo para que possam ser utilizados na resolução de algum problema de maior porte.

Legion [GWF+94] [GWT97] [leg05], desenvolvido na Universidade de Virginia em 1993, objetiva construir uma plataforma de processamento paralelo baseada em objetos na qual todos os elementos da plataforma são representados por objetos, sejam eles dados ou objetos reais como computadores, redes, microscópios, telescópios ou outros dispositivos físicos.

A plataforma de processamento paralelo Globus [FK99] [MHHK99] [ACF+01], desenvolvida em conjunto pelo Argonne National Laboratory e pela Universidade Southern California no ano de 1997, é um projeto caracterizado por um conjunto de serviços para plataformas de processamento paralelo, permitindo que aplicações para tais plataformas sejam desenvolvidas de maneira incremental pela possibilidade de inserção de novos serviços.

A plataforma ESG (*Earth System Grid*) [FGA+02] [PCD+03], desenvolvida em 1999 e patrocinada pelo Departamento de Energia dos Estados Unidos, é um projeto de pesquisa para a construção de uma plataforma de processamento paralelo e de dados a fim de prover um ambiente poderoso e transparente para permitir uma nova geração de pesquisas climáticas. Trata-se de uma plataforma distribuída para armazenamento, acesso e localização de dados previamente processados resultantes de simulações climáticas.

SETI@home [ACK+01] [ACK+02] [Gro05] [Com05], desenvolvido na Universidade de Berkeley em 1999, é uma das plataformas de processamento paralelo mais conhecidas da atualidade. Tem o objetivo de realizar a busca de inteligência extra-terrestre pela análise de ondas de rádio, nas quais realizam-se buscas por padrões que possam evidenciar atividades de vida inteligente. A demanda de processamento do SETI@home é suprida pela utilização de uma pequena parcela dos milhões de computadores distribuídos ao redor do globo, que juntos fornecem alto poder de processamento.

A plataforma de processamento paralelo BOINC (*Berkeley Open Infrastructure for Network Computing*) [And04a] [And04b] foi projetada em 2002 pelo mesmo grupo da plataforma SETI@home. Ela foi desenvolvida para sanar uma grande limitação do SETI@home, que é só poder executar um tipo de aplicação. BOINC, pelo contrário, provê um sistema que pode ser utilizado para executar diversos tipos de aplicações.

Inicialmente chamada de MyGrid, a plataforma OurGrid [OBV+02] [CPC+03] [CFA+04] é um ambiente que distribui e controla a execução de aplicações em vários computadores. Desenvolvida pela parceria entre a Universidade Federal de Campina Grande e a empresa Hewlett-Packard (HP), ela busca ser uma proposta aberta (não exige uma infra-estrutura específica), extensível (facilita a inserção de novas funcionalidades) e completa (cobre todo o ciclo de produção que uma plataforma de processamento paralelo deve apresentar), procurando tratar aplicações do tipo *bag-of-tasks*.

JoiN [Yer98] [LH99] [Hen99] [Luc02] [PvZH03] [Yer03] é uma plataforma de processamento paralelo nascida como fruto de pesquisas iniciadas em 1996, na Universidade Estadual de Campinas. Ela possui uma estrutura baseada em componentes e se propõe a fornecer portabilidade no desenvolvimento e na execução de aplicações paralelas. Tal estrutura foi desenvolvida de forma que novos componentes e ferramentas sejam facilmente incorporados à plataforma, para incrementar suas funcionalidades. A plataforma JoiN permite que aplicações sejam executadas de forma automatizada aproveitando a ociosidade dos computadores, exigindo apenas que um programa cliente da plataforma permaneça em execução no computador disponibilizado a ela. O programa cliente é responsável por receber requisições para o processamento de uma ou mais aplicações, a partir de um conjunto de dados e do código que são escalonados e enviados por um módulo chamado coordenador.

Como pode ser observado, há inúmeras implementações de plataformas de processamento paralelo que tratam o gerenciamento dos recursos de uma plataforma paralela de diferentes maneiras para executar aplicações. Todas elas consideram aspectos semelhantes quanto ao projeto da infra-estrutura da plataforma e das aplicações. Tais aspectos existem devido às próprias características da plataforma paralela, como ampla distribuição, grande escala, alta heterogeneidade, múltiplos domínios administrativos, entre outros.

Para tratar de tais aspectos, uma plataforma de processamento paralelo normalmente é provida de um conjunto de funcionalidades expansíveis, que podem ser chamadas de serviços ou também de sistemas de uma plataforma paralela (Fig. 1.2). Cada sistema é responsável por tratar de diferentes aspectos quanto às aplicações e quanto à infra-estrutura da plataforma de processamento paralelo. A Fig. 1.2 procura ilustrar a independência entre os sistemas que compõem uma plataforma de processamento paralelo, a existência de uma camada responsável por gerenciar a execução e interação entre eles e a capacidade de incremento de novos sistemas para o caso de uma plataforma de processamento paralelo expansível. Alguns sistemas são descrito resumidamente a seguir.

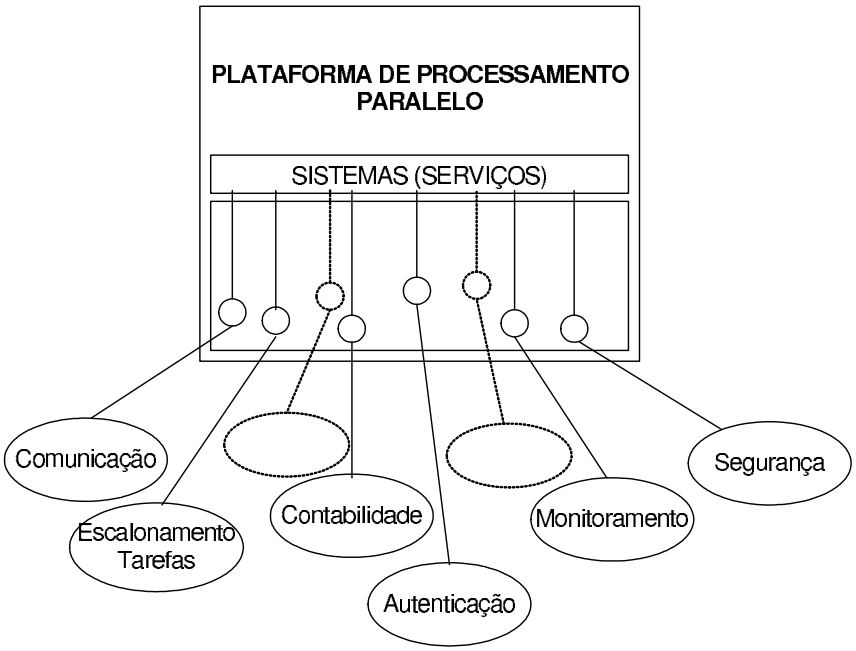

Fig. 1.2: Serviços de uma plataforma de processamento paralelo

O sistema de escalonamento de tarefas é responsável por alocar tarefas de uma aplicação aos computadores da plataforma paralela de forma a equilibrar a carga de trabalho entre eles.

O sistema de autenticação é responsável por gerenciar a entrada de usuários e aplicações, e o acesso aos recursos de uma plataforma de processamento paralelo.

O sistema de segurança visa garantir a integridade dos dados transmitidos pelos computadores de uma plataforma de processamento paralelo, evitando que informações importantes sejam acessadas por usuários indevidos.

O sistema de comunicação é responsável pelo controle de tráfego e entrega de mensagens entre os computadores que compõem uma plataforma de processamento paralelo, informando também quando uma mensagem não pôde ser entregue ao destinatário.

O sistema de tolerância a falhas visa garantir que, quando algum tipo de problema previsível ocorra, a plataforma de processamento paralelo possa se recuperar do problema preservando, se possível, algum estado anterior à ocorrência da falha.

O serviço de contabilidade é responsável pelo registro dos serviços prestados pelos computadores. Este sistema é importante porque pode motivar, por meio de alguma forma de pagamento, proprietários de computadores a disponibilizar suas máquinas à plataforma paralela.

O sistema de monitoramento permite acompanhar a utilização e a disponibilidade dos recursos de uma plataforma paralela, de maneira localizada ou global, em tempo real ou por gráficos construídos estatisticamente. Por meio desse sistema é possível identificar a ocorrência de falhas ou possíveis

melhorias que podem ser realizadas na plataforma de processamento paralelo.

Mesmo considerando inúmeros aspectos de projeto da infra-estrutura, o grande dinamismo dos ambientes nos quais as plataformas de processamento paralelo operam acaba degradando seu desempenho. Além disso, muitas aplicações são inviáveis porque requerem um poder de processamento que vai muito além daquele disponibilizado pelas plataformas paralelas atuais.

Sistemas de predição de desempenho permitem estimar o tempo de execução de uma aplicação antes mesmo de executá-la nos computadores de uma plataforma de processamento paralelo. Isso permite verificar se é viável o tempo de execução de uma aplicação e a quantidade mínima de computadores que seriam necessários para viabilizar o uso de uma plataforma de processamento paralelo na execução de uma aplicação dentro de um prazo específico [Gau04].

Plataformas de processamento paralelo que possuam um sistema de predição, por mais simples que seja, podem ser mais eficientes quanto à utilização dos recursos, além de permitir conhecer o tempo que é necessário aguardar para que o processamento de uma aplicação seja concluído. Essa informação pode ser muito útil, pois o proprietário de uma aplicação poderá decidir entre executar uma aplicação na plataforma atual, utilizar outra configuração de computadores, ou até mesmo deixar de executá-la.

A obtenção de estimativas de tempo leva em conta algumas características da aplicação e das máquinas da plataforma paralela que participarão do seu processamento. Tais estimativas podem ser feitas de várias formas e, entre elas, destacam-se duas: as baseadas em técnicas dinâmicas e as baseadas em técnicas estáticas (também conhecidas como analíticas).

Técnicas analíticas são uma boa alternativa se comparadas às técnicas dinâmicas, pois fornecem boas estimativas de tempo a um custo baixo e de forma bastante rápida, pois não exigem o uso efetivo dos computadores e de informações além daquelas que já estão disponíveis na plataforma de processamento paralelo. As ferramentas existentes que implementam alguma técnica de predição analítica, o fazem de diversas maneiras, exigindo pelo menos um dos itens listados a seguir:

- utilização de uma linguagem de programação não-padrão, pouco utilizada e pouco conhecida: requer tempo disponível para seu aprendizado. Além disso, a documentação e exemplos de sua utilização são, muitas vezes, escassos devido à sua pouca procura e baixa utilização;

- inserção de marcas especiais no código-fonte da aplicação: a partir do código-fonte que já se encontra pronto para execução em um sistema paralelo, é necessário aprender e entender a forma com que se deve inserir marcas especiais no mesmo, tais como comentários com uma notação específica. Isto serve para indicar ao sistema que fará a predição de desempenho da aplicação quais as partes do código que indicam, por exemplo, se a execução será feita de maneira seqüencial ou paralela;

- modificação na arquitetura da plataforma de processamento paralelo: dependendo do caso, pode ser necessário implementar ou modificar parte do sistema paralelo para que um sistema de predição de desempenho possa ser utilizado ou desenvolvido.

Além disso, há técnicas e/ou ferramentas que não levam em consideração, na análise do código-fonte, atrasos causados por operações aritméticas realizadas sobre diferentes tipos de dados. Outro fator de atraso que algumas ferramentas não tratam é a contenção causada quando um recurso (rede, CPU) é utilizado de forma compartilhada e concorrente.

1.3 Objetivos

Em função das vantagens apresentadas pelos sistemas de predição de desempenho, é proposto neste trabalho o desenvolvimento de um sistema de predição baseado em uma técnica analítica para a plataforma de processamento paralelo JoiN. Desta forma, os objetivos são claramente listados a seguir:

- verificar a qualidade das estimativas geradas pela técnica PAMELA Estendida quando implementada por uma ferramenta;
- obter experiência com o desenvolvimento de sistemas de predição de desempenho;
- conhecer de forma detalhada a implementação e o funcionamento de uma plataforma de processamento paralelo para desenvolver um sistema de predição;
- estudar tecnologias relacionadas à predição de desempenho utilizadas em outras plataformas de processamento paralelo;
- propor um sistema de predição para a plataforma de processamento paralelo JoiN.

O sistema proposto, batizado de PeMPSys (*Performance Modeling and Prediction System*), utiliza a técnica de predição analítica PAMELA Estendida[Yer03] para a geração de estimativas de tempo. As estimativas baseiam-se na construção e análise de modelos que apresentam informações de atraso de uma aplicação e o desempenho dos computadores de uma plataforma de processamento paralelo.

Para produzir os modelos e as estimativas, o sistema possui três ferramentas: CRES (*Computational Resource Evaluation Service*), ARC (*Analytical Representation Compiler*) e APET (*Analytical Performance Estimation Tool*).

Além da necessidade de um sistema de predição de desempenho para a plataforma de processamento paralelo JoiN, o desenvolvimento deste trabalho foi motivado também por outros fatores que são peculiares no desenvolvimento de sistemas deste tipo. Alguns deles são listados a seguir:

- verificar a qualidade das estimativas geradas pela técnica PAMELA Estendida quando implementada por uma ferramenta;
- obter experiência com o desenvolvimento de sistemas de predição de desempenho, ou de outros sistemas semelhantes a este;

- conhecer de forma detalhada a implementação e o funcionamento de uma plataforma de processamento paralelo como o JoiN para o desenvolvimento de um sistema de predição de desempenho;
- fornecer uma contribuição científica na forma de proposta de um sistema de predição para uma plataforma de processamento paralelo com as características similares às de JoiN.

1.4 Contribuições

As principais contribuições deste trabalho são:

- a disponibilização de uma ferramenta de predição de desempenho capaz de estimar de forma confiável tempos de execução para determinados tipos de aplicações aparalelas, mesmo que usando computadores heterogêneos e não dedicados como aquelas da plataforma JoiN de processamento paralelo;
- o levantamento dos tipos de aplicação nos quais a linguagem PAMELA Estendida é eficiente para modelagem e obtenção de estimativas de tempo de execução.

1.5 Organização do trabalho

Em vista do que é proposto, o restante desta dissertação está organizado da seguinte forma:

- ***Capítulo 2:*** apresenta alguns conceitos básicos quanto à construção de modelos para técnicas de predição analítica de desempenho e alguns trabalhos relacionados a este que introduzem a terminologia e os conceitos necessários ao entendimento do texto e a literatura correlata;
- ***Capítulo 3:*** apresenta a técnica de modelagem e predição analítica de desempenho PAMELA Estendida e algumas características específicas da plataforma de processamento paralelo JoiN. A primeira é apresentada de forma simplificada, com os principais operadores da notação e a forma com que os modelos são representados. Em relação a JoiN, são apresentadas as principais características da plataforma de processamento paralelo quanto a sua estrutura e como suas aplicações são executadas;
- ***Capítulo 4:*** descreve a construção do sistema de predição PeMPSys, de acordo com as características da plataforma de processamento paralelo JoiN, utilizando a técnica PAMELA Estendida para a geração das estimativas de tempo. É visto como as ferramentas CRES, ARC e APET criam seus modelos de forma automática e como estes são analisados para obter estimativas de tempo da execução de uma aplicação na plataforma de processamento paralelo considerada. Ao final do capítulo são discutidas algumas limitações do sistema PeMPSys e possíveis alternativas de uso para que melhores estimativas possam ser obtidas;

- ***Capítulo 5:*** apresenta vários experimentos que foram realizados utilizando diferentes tipos de aplicações e também a configuração de cada um dos computadores utilizados. Nos experimentos são modificados alguns parâmetros para que os tempos de execução possam ser comparados com os tempos produzidos pelo sistema de predição. Os resultados produzidos por tais experimentos são analisados e discutidos para verificar a qualidade das estimativas fornecidas pelo sistema de predição de desempenho PeMPSys;

- ***Capítulo 6:*** apresenta algumas conclusões e trabalhos futuros em relação ao sistema desenvolvido e aos resultados obtidos. Também discute a melhoria do sistema de predição e outros trabalhos que podem ser desenvolvidos a partir deste.

Capítulo 2

Conceitos básicos e trabalhos relacionados

2.1 Introdução

Este trabalho apresenta a proposta de um sistema de modelagem e predição analítica de desempenho e, para que ele seja descrito, é importante que antes sejam definidos alguns conceitos básicos relacionados à construção de tais sistemas e também algumas técnicas de predição que podem ser implementadas e utilizadas para a obtenção de estimativas de tempo.

Em relação às técnicas, serão apresentadas as dinâmicas e as estáticas (ou analíticas), com suas peculiaridades, bem como vantagens e desvantagens para que, ao final do capítulo, seja feito um levantamento de alguns trabalhos relacionados às mesmas. São também discutidos sistemas semelhantes ao proposto para que se possa relacionar as características apresentadas por cada um deles, e que foram úteis para a definição de algumas funcionalidades da proposta apresentada aqui.

2.2 Conceitos básicos

2.2.1 Modelos para a predição de desempenho

Modelos podem ser utilizados para representar uma variedade de entidades computacionais para prever a forma da utilização de computadores em função das aplicações. A criação de tais modelos permite conhecer as características fundamentais de um determinado sistema, representando-o de forma organizada e facilitando estudos de análise e predição de desempenho.

Mullender [Mul93] define modelos como coleções de atributos e regras que regem as interações sobre estes atributos. Para Schopf [Sch98], um modelo é uma representação matemática de um processo, dispositivo ou conceito.

Independentemente da definição, em geral modelos são uma forma eficiente de representar informações de desempenho dos computadores de uma plataforma de processamento paralelo ou de uma aplicação. Quando utilizados para representar os computadores de uma plataforma paralela, modelos possuem informações que podem definir, por exemplo, a velocidade de computação e de comunicação de um computador na execução de alguma operação de cálculo matemático ou de envio de dados, respectivamente. Tais modelos são conhecidos como modelos de máquinas. Para representar uma aplicação, modelos costumam apresentar informações de atraso extraídos do código-fonte. Tais informações podem representar, por exemplo, a quantidade de operações matemáticas realizadas, o tipo de dados analisados e trechos de troca de mensagens com outros computadores. Neste último caso, o modelo é conhecido como modelo da aplicação.

As informações presentes nos modelos podem ser úteis para realizar diferentes tipos de análise da aplicação em função dos dados processados por ela.

2.2.2 Compiladores

As informações de atraso presentes em um modelo da aplicação podem ser coletadas a partir da análise do código-fonte de uma aplicação de duas formas: manual ou por meio de uma ferramenta.

Extrair informações de atraso de forma manual consiste em uma tarefa árdua, pois consome grande tempo por parte do responsável por realizar essa atividade. Em função disso, esta forma de criação do modelo da aplicação costuma ser pouco utilizada.

Ferramentas que criam automaticamente o modelo da aplicação a partir de seu código-fonte são uma boa alternativa para sistemas que utilizam informações presentes no modelo da aplicação. Entretanto, a construção de tais ferramentas é bastante complexa e exige o conhecimento de uma grande quantidade de conceitos e técnicas relacionadas para que a ferramenta possa funcionar adequadamente [Gau04].

Compiladores são um tipo de ferramenta que podem ser utilizados para a criação de modelos de aplicação. De forma geral, um compilador é um programa que recebe como dado de entrada um programa escrito em uma linguagem de programação (a linguagem fonte) e produz como saída de seu processamento um programa escrito em outra linguagem, simplificada ou não, que pode ser analisado posteriormente por outro compilador.

A arquitetura convencional de um compilador divide-se em Análise e Síntese, conforme a ilustração da Fig. 2.1. A Análise pode ser subdividida ainda em Análise Léxica, Análise Sintática e Análise Semântica. A Síntese é mais variada, podendo ser composta pelas etapas de Geração de Código Intermediário, Otimização de Código e Geração de Código Final. Somente esta última etapa da Síntese é obrigatória [ASU86].

O principal objetivo do Analisador Léxico é identificar seqüências de caracteres que constituem unidades léxicas, que são chamadas de tokens. Para isso, o analisador lê o programa fonte, caractere por caractere, verificando se os caracteres lidos pertencem ao alfabeto da linguagem, identificando os tokens e desprezando comentários e espaços em branco desnecessários.

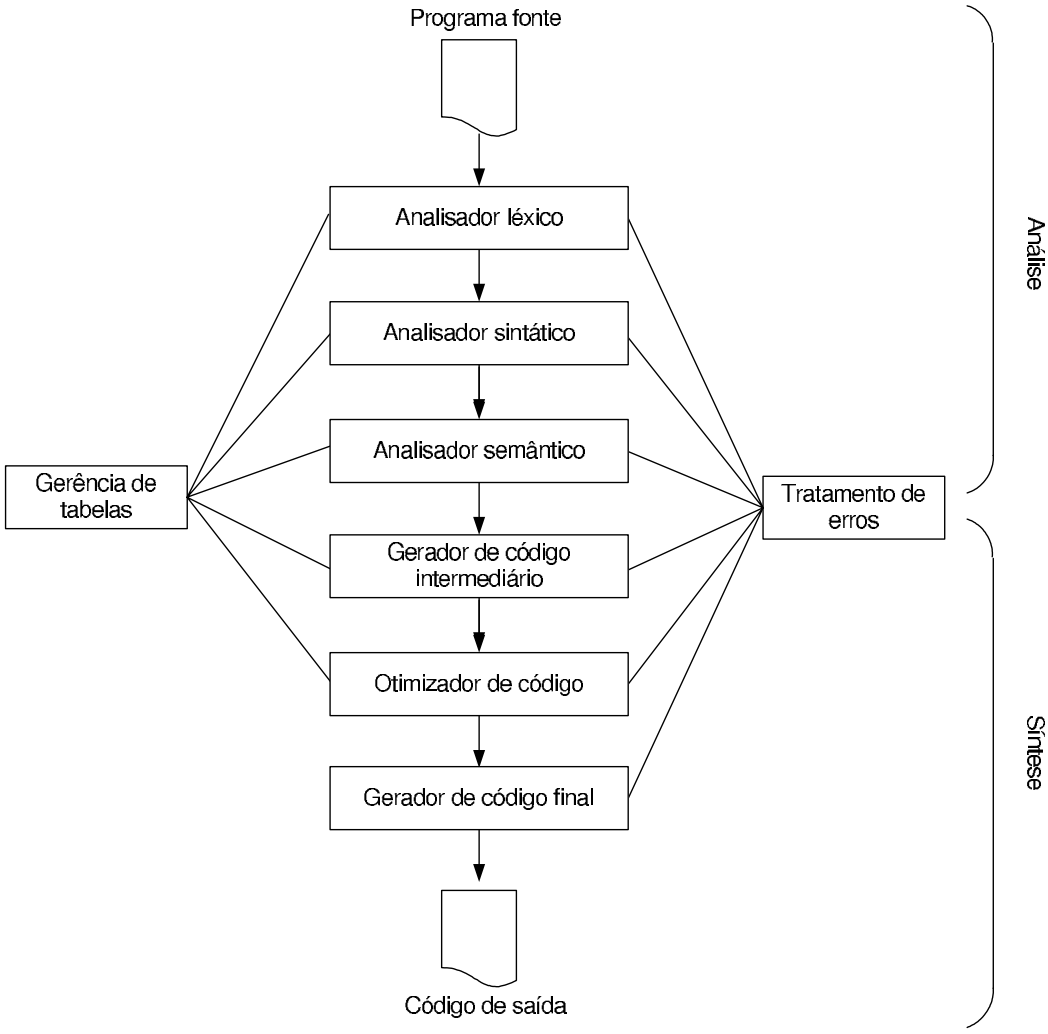

Fig. 2.1: Fases de desenvolvimento de um compilador para análise do código-fonte de uma aplicação

Os tokens são classes de símbolos tais como: palavras reservadas, delimitadores, identificadores etc. Estes elementos podem ser representados internamente através do próprio símbolo, como no caso das palavras reservadas e delimitadores, ou por um par ordenado, onde o primeiro elemento indica a classe do símbolo, e o segundo, é um índice para uma área onde o próprio símbolo foi armazenado.

Além da identificação de tokens, o Analisador Léxico inicia a construção da Tabela de Símbolos, e também envia mensagens de erro caso identifique unidades léxicas não aceitas pela linguagem. A saída do Analisador Léxico é uma cadeia de tokens que é passada para a próxima fase do compilador, a Análise Sintática. Geralmente, o Analisador Léxico é implementado como uma subrotina que funciona sob o comando do Analisador Sintático.

JLex é um gerador de analisadores léxicos que pode ser utilizado para produzir analisadores para aplicações escritas em várias linguagens de programação, como Java, por exemplo. Para geração do

Analisador Léxico pela ferramenta JLex, deve ser especificado um arquivo contendo as expressões regulares que definem os tokens da linguagem que será analisada. Este arquivo é analisado pela ferramenta JLex, criando um programa em Java que, após compilado, representa o Analisador Léxico.

O Analisador Sintático identifica seqüências de símbolos que constituem estruturas sintáticas, como por exemplo, expressões e comandos, por meio de uma varredura ou *parsing* da representação interna (cadeia de tokens) do programa fonte. Outra função do Analisador Sintático é a detecção de erros de sintaxe, com o objetivo de identificar de forma clara a posição e o tipo de erro ocorrido. Mesmo que erros sejam encontrados, o Analisador Sintático deve tentar recuperá-los prosseguindo a análise do texto restante. Freqüentemente, o Analisador Sintático trabalha em conjunto com o Analisador Semântico, cuja principal atividade é determinar se as estruturas sintáticas analisadas fazem sentido, ou seja, verificar se um identificador declarado como variável é usado como tal; se existe compatibilidade entre operandos e operadores em expressões, por exemplo.

Cup é um gerador de analisadores sintáticos para o qual deve ser especificado um arquivo contendo a gramática da linguagem que será analisada. A gramática projetada para a ferramenta CUP mapeia as regras de organização dos tokens da linguagem de programação analisada. Em função disso, analisadores léxicos e sintáticos costumam ser utilizados em conjunto, no qual os primeiros fornecem os tokens para que se realize a análise sintática do código.

O Gerador de Código Intermediário utiliza a representação interna produzida pelo Analisador Sintático e gera uma seqüência de código final (código de saída), ou gera um código intermediário, sendo que este último possibilita algumas vantagens, tal como a otimização do código intermediário. Esta última fase é considerada uma fase opcional no desenvolvimento de um compilador, podendo ou não existir.

A Gerência de Tabelas compreende um conjunto de tabelas e rotinas associadas que são utilizadas por quase todas as fases do compilador. Algumas tabelas usadas são fixas para cada linguagem, tais como: a tabela de palavras reservadas, tabela de delimitadores etc. Entretanto, a estrutura de maior importância é aquela que é montada durante a análise do programa fonte, com informações sobre: declarações de variáveis, declarações dos procedimentos ou subrotinas , parâmetros etc. Essas informações são armazenadas na Tabela de Símbolos. A cada ocorrência de um identificador no programa fonte, a tabela é acessada, e o identificador é procurado na tabela. Caso encontrado, as informações associadas a ele são comparadas com as informações obtidas no Programa Fonte, sendo que qualquer nova informação é inserida na tabela.

O Tratamento de Erros tem por objetivo tratar erros que são detectados em todas as fases de Análise e Síntese do programa fonte. Qualquer fase analítica deve prosseguir em sua análise, mesmo que erros tenham sido detectados.

Os compiladores desenvolvidos neste trabalho abordarão, na fase de síntese, apenas a geração do código final, sem passar pelas etapas opcionais de geração de código intermediário e de otimização de código.

2.2.3 Intervalos

As informações presentes em um modelo de máquinas costumam ser utilizadas em sistemas de predição de desempenho para estimar o tempo de execução de uma aplicação em um determinado conjunto de computadores.

A construção do modelo das máquinas costuma ser feita com o uso de algoritmos de *benchmark* para saber o tempo aproximado que determinados tipos de operações, também presentes em uma aplicação, podem levar para serem executados.

Um problema na utilização das informações de benchmark é que elas nem sempre representam de forma eficiente o desempenho dos computadores de uma plataforma de processamento paralelo heterogênea e compartilhada, pois este desempenho costuma variar bastante devido à execução de outros programas de usuários, os quais são executados em horários aleatórios, sendo praticamente impossível prever com exatidão o desempenho de um computador em determinado horário.

O uso de intervalos é uma alternativa para criar o modelo das máquinas como proposto por Schopf e Berman [SB97] . Eles permitem expressar possíveis variações no tempo de execução de uma aplicação, estabelecendo desta forma um tempo mínimo e máximo esperado, na forma de um intervalo fechado de valores do tipo $[t_{min}, t_{max}]$.

Os intervalos são construídos tipicamente de um conjunto de valores pontuais representando diferentes medidas de tempo. A geração dos dados pode ser feita por algoritmos de *benchmark* que apresentam um certo grau de eficiência para representar o comportamento de uma determinada máquina na execução, por exemplo, de operações matemáticas. Para que vários valores sejam gerados para um mesmo experimento, o *benchmark* é executado várias vezes, sendo que a cada medição, é gerado um tempo que representa o desempenho do computador para executar o experimento naquele instante. Após a obtenção de um conjunto com vários resultados para o mesmo *benchmark*, pode-se então criar um intervalo do tipo $[min, max]$ que representa o comportamento do recurso medido para as operações realizadas pelo *benchmark*. A geração do intervalo é feita através de uma análise estatística dos dados em relação à amplitude dos valores para determinar o comportamento provável do recurso analisado.

Definição de intervalos

A caracterização de um intervalo X é definido como sendo uma tupla

$$X = [\underline{x}, \overline{x}]; \;\; \{x \in X | \underline{x} \leq x \leq \overline{x}\} \tag{2.1}$$

onde $\underline{x}$ é o limite inferior do intervalo, e $\overline{x}$ o limite superior.

A partir da sua sua definição, podem ser realizadas algumas operações básicas sobre eles [SB97]. Dados os intervalos X e Y de valores, e um valor escalar p, tem-se que:

$X+Y$	$[\underline{x}+\underline{y}, \overline{x}+\overline{y}]$
$p+X$	$[p+\underline{x}, p+\overline{x}]$
$p*X$	$[p*\underline{x}, p*\overline{x}]$
$max(X,Y)$	$[max(\underline{x},\underline{y}), max(\overline{x},\overline{y})]$

Criação de intervalos

Existem várias abordagens para a construção de um intervalo a partir de valores absolutos medidos. A mais simples delas seria usando os valores extremos (maior e menor tempo) gerados pelos algoritmos de *benchmark*. Outra forma seria excluir os pontos extremos e gerar o intervalo com o segundo e o penúltimo maior valor do intervalo. Outra técnica, interessante para dados com distribuição normal, seria a utilização do desvio padrão na determinação dos pontos do intervalo.

A abordagem utilizada aqui para criar os intervalos procura basicamente excluir pontos das extremidades que estejam muito distantes dos demais calculados pelo algoritmo de *benchmark* executado em uma determinada máquina. Tal abordagem foi proposta por Huerta [Yer03] e permite que seja indicada uma porcentagem C de pontos que devem permanecer dentro de um intervalo selecionado.

O método de seleção segue a seguinte notação:

- $S = x_1, x_2, ..., x_N$: seqüência ordenada de valores pontuais produzidos por um mesmo algoritmo de benchmark, em diferentes instantes de tempo;
- $X_{i,j} = [x_i, x_j]$: é um sub-intervalo definido pelos elementos x_i e x_j de S.

A idéia é descartar D pontos, mantendo $C\%$ de um determinado intervalo S, onde D é calculado pela Eq. 2.2.

$$D = \left\lfloor \frac{(100 - C) * N}{100} \right\rfloor \tag{2.2}$$

Após definida a porcentagem C de pontos mantidos, varre-se o intervalo em busca de um sub-intervalo com a menor amplitude de $A(X_{i,j}) = X_j - X_i$. A meta, desta forma, é encontrar um sub-intervalo da forma:

$$X_{i,j}; 1 \leq i \leq D+1 \; e \; j = N - D + i - 1 \, , \tag{2.3}$$

o qual descarta $i-1$ pontos do início e $N-j$ pontos do fim do intervalo original e resulta no sub-intervalo de melhor amplitude possível.

A abordagem apresentada permite obter o comportamento do sistema analisado de forma mais precisa, excluindo valores obtidos pelos *benchmarks* que colocariam as estimativas dentro de um intervalo muito grande e, desta forma, muito ineficiente para a predição. Assim, o uso de intervalos pode fornecer tempos que refletem de forma mais adequada a variação de desempenho dos recursos analisados [Yer03].

2.3 Técnicas de predição de desempenho

Para realizar estimativas de tempo, um sistema de predição deve adotar alguma técnica que forneça bons resultados e que seja feita de forma rápida, pois a utilização de sistemas de predição que demoram para realizar as estimativas de tempo pode desencorajar seu uso de forma prática. Entretanto, é difícil construir um sistema de predição e torná-lo eficiente para avaliar o desempenho de plataformas de processamento paralelo sobre cargas de trabalho reais, pois diversos fatores que afetam no tempo de execução de uma aplicação devem ser considerados.

Para a maioria das aplicações com troca de mensagens, a velocidade de comunicação é crucial para alcançar boas estimativas de desempenho. Além disso, é necessária alguma técnica que permita analisar efeitos de contenção no uso de recursos compartilhados quando estes são limitados. Se a aplicação estiver sendo executada em um ambiente não dedicado, ela poderá ter que compartilhar a utilização do processador com outros processos que estão em execução. Huerta [Yer03] chama isso de contenção externa, na qual os processos concorrentes com a aplicação podem ser de usuários ou do sistema operacional e estão fora do controle da plataforma de processamento paralelo. Este tipo de contenção é muito difícil de ser modelado, mas apresenta algumas alternativas de tratamento, tais como a abordagem de intervalos proposta por Schopf e Berman[SB97] e estendida por Huerta [Yer03].

Outro tipo de contenção é a interna que, ao contrário da externa, pode ser modelada mais facilmente, pois quem induz a contenção é a própria plataforma de processamento paralelo ou a aplicação executada por ela.

Como visto, há várias considerações importantes na obtenção de estimativas de tempo de execução de uma aplicação. Para isso, as técnicas de predição subdividem-se em dois tipos de abordagens: dinâmicas e estáticas (ou analíticas).

2.3.1 Técnicas de predição dinâmica

Técnicas de predição dinâmica baseiam suas estimativas de tempo em informações relativas à execução simplificada da aplicação na plataforma de processamento paralelo. Tais técnicas realizam a análise da contenção interna sobre os componentes paralelos que formam a aplicação. Devido à natureza dinâmica de execução, tal análise deve considerar todos os possíveis fluxos de execução, que podem implicar em uma complexidade de ordem exponencial.

As técnicas dinâmicas são baseadas principalmente em simulação e instrumentação. Na simulação a predição é obtida por meio da análise de dados relativos ao tempo de execução de todo ou parte do código de uma aplicação sobre um conjunto reduzido de dados, como fazem as técnicas das referências [ZKK04], [DJSW96], [SCW+02] e [BDCW92]. Na instrumentação, as estimativas de tempo são obtidas por meio de medições durante uma execução simplificada da aplicação para localizar blocos do código executadas na maioria do tempo pela aplicação. Técnicas como as propostas nas referências [MC95], [MCC+95], [Sar95], [HMG+97] e [AR01] baseiam-se nesta forma

de estimativa.

A qualidade dos resultados fornecidos por técnicas dinâmicas apresentam, em sua maioria, boa precisão. Entretanto, há um custo envolvido devido à necessidade de utilização dos computadores da plataforma de processamento paralelo no momento da obtenção da estimativa de tempo.

Portanto, um fator limitante do uso de técnicas dinâmicas é que, para que a predição seja possível, é necessário que os computadores que serão utilizados para o processamento da aplicação estejam disponíveis no momento da obtenção da estimativa de tempo. Outro fator é o tempo envolvido, que pode ser relativamente elevado. Este último fator pode, em alguns casos, desmotivar o uso de tais ferramentas. Usuários podem optar por executar diretamente a aplicação na plataforma de processamento paralelo, em vez de ficar aguardando que a estimativa de tempo fique pronta [Gau04].

2.3.2 Técnicas de predição estáticas ou analíticas

Técnicas de predição analíticas, ao contrário das dinâmicas, são baseadas em informações estáticas da plataforma de processamento paralelo e da aplicação. Elas armazenam tais informações na forma de modelos configuráveis para determinar a sensibilidade de um algoritmo para as características de um conjunto de computadores que farão seu processamento. As informações dos modelos são representadas de forma a facilitar estudos de análise e predição de desempenho de aplicações.

O modelo de máquinas contém características relativas ao desempenho de comunicação e de computação dos computadores na execução de operações, tais como aritméticas (soma, subtração, multiplicação e divisão) sob diferentes tipos de dados (como inteiro, inteiro longo, ponto flutuante de precisão simples e dupla).

O modelo da aplicação contém características da aplicação quanto a forma com que os dados e as tarefas serão distribuídas e processadas nos computadores da plataforma de processamento paralelo, e quanto ao tipo de operações de computação e de comunicação executadas sobre os dados.

A análise da contenção interna nas técnicas de predição analíticas baseiam-se apenas nas informações presentes nos modelos mencionados. Nela, os modelos são analisados para coletar informações sobre a estrutura do paralelismo da aplicação, o código executado pelas tarefas e os computadores que farão o seu processamento. Sistemas de predição como PerPret, SUIF, PACE e PAMELA obtêm suas estimativas de tempo baseados em técnicas analíticas. Eles serão descritos em mais detalhes na próxima seção.

De forma geral, técnicas analíticas fornecem boas precisões, porém inferiores se comparadas às dinâmicas. Entretanto, o custo da utilização de técnicas analíticas é significativamente menor, pois não é necessária a utilização dos computadores e de outros recursos para a geração das estimativas de tempo. Além disso, o tempo gasto na obtenção das estimativas é da ordem de segundos, ao contrário das técnicas dinâmicas que, devido a natureza dinâmica das informações utilizadas, podem levar minutos ou até mesmo horas para que as estimativas de tempo sejam obtidas, dependendo das características da aplicação. Em função disso, técnicas analíticas têm se apresentado como uma boa alternativa na construção de sistemas de predição de desempenho. Um fator limitante é que

as ferramentas que utilizam tais técnicas são mais complexas, pois requerem uma análise bastante minuciosa do código da aplicação e das características da plataforma de processamento paralelo.

2.4 Sistemas de predição analíticos de desempenho

Mesmo apresentando bons resultados práticos em uma certa implementação, uma técnica de predição não apresentará a mesma qualidade nos resultados para qualquer plataforma de processamento paralelo. Isto ocorre porque há uma grande quantidade de plataformas de processamento paralelo, cada uma com suas peculiaridades, que tratam dos recursos e das aplicações de diferentes maneiras.

Há sistemas de predição, como aqueles apresentados na referências [vG03] e [BMSD95], baseados em técnicas analíticas, que, na coleta de informações de uma aplicação, consideram que todas as operações realizadas são, por exemplo, de ponto flutuante, mesmo que a aplicação realize operações sobre outros tipos de dados. Grande parte dos sistemas de predição impõe algumas restrições quanto ao seu uso de forma prática. Alguns exigem a inserção de marcas específicas no código-fonte da aplicação para que a predição funcione [BMSD95]. Outros requerem a utilização de uma linguagem de programação especial (não padrão) no desenvolvimento das aplicações para a plataforma de processamento paralelo, com o intuito de dispensar o uso de marcas no código e a intervenção manual para que a predição funcione [JSKN03]. Há casos que, mesmo com a utilização de uma linguagem não padrão, o uso de marcas é obrigatório [KRR04]. Há ainda sistemas de predição que requerem modificações na arquitetura da plataforma de processamento paralelo para que possam coletar dados para obter as estimativas de tempo [Gau04].

Há sistemas de predição de desempenho que requerem a reescrita manual de parte ou todo o código da aplicação em uma linguagem de programação não padrão, na forma de um modelo equivalente que possua informações de atraso a serem consideradas na predição [KRR04]. Esta exigência de intervenção manual pode desencorajar o uso das ferramentas que utilizam tais modelos.

A seguir são apresentados alguns sistemas de análise e predição de desempenho baseados em técnicas analíticas, que se propõem a tratar do problema de estimar o tempo de execução de uma aplicação em uma plataforma de processamento paralelo utilizando diferentes abordagens de predição. As ferramentas procuram simplificar o processo de geração das estimativas de tempo, baseando-se em técnicas e funcionalidades específicas. Para calcular a precisão das estimativas, os sistemas baseiam-se na relação entre o tempo de execução e o tempo estimado.

2.4.1 PerPreT

O sistema PerPreT (*PERformance PREdiction Tool*), desenvolvido na linguagem de programação Fortran, utiliza um simples modelo de predição estática para obtenção de informações de aplicações,

como speedup [1], eficiência [2], tempo de execução e de comunicação de aplicações escritas no modelo de programação SPMD (*Single Program Multiple Data*). Neste modelo, o mesmo código é carregado em todas as unidades de processamento para realizar a mesma tarefa sobre diferentes conjuntos de dados [BMSD95]. Especialmente para casos onde o número de processadores é grande (sistemas maciçamente paralelos), o modelo usado pela ferramenta PerPreT possui maior aplicabilidade que técnicas de modelagem tradicionais baseados em Modelos de Markov ou Redes de Petri, pois estes não são adequados para sistemas com alta escalabilidade [vG96].

Para a predição, a ferramenta PerPreT utiliza uma descrição parametrizada do sistema e da aplicação, onde o tamanho do problema e o número de processadores usados para executar um programa são parâmetros livres que podem ser alterados no modelo de predição gerado pela ferramenta.

Para representar as aplicações paralelas, o sistema cria equações analíticas com parâmetros de comunicação e computação livres. Para cada uma das equações, há uma descrição do sistema de predição relacionado à comunicação e à computação:

a. comunicação: as rotinas do processo de comunicação presentes na ferramenta PerPreT pertencem a uma biblioteca de comunicação da qual não se têm detalhes sobre sua especificação. Isso impossibilita adaptá-la a um sistema específico que necessite a incorporação de novas funções e inviabiliza uma análise detalhada sobre o comportamento da comunicação;

b. computação: o modelo de computação da aplicação é obtido simplesmente pela análise do número de declarações aritméticas que serão executados em cada máquina do sistema. A medição dos valores referente a cada tipo de operação de computação é feita em flops (*FLoating point Operations Per Second*), isto é, considera-se que todas as operações realizadas pela aplicação são em ponto flutuante.

A criação das equações é feita de forma direta pela ferramenta sem a geração de uma representação intermediária da aplicação paralela.

Apesar da similaridade com outras ferramentas na geração do modelo da aplicação, a ferramenta PerPreT não possui a capacidade de realizar a predição de tempo considerando fatores de contenção pelo uso de recursos (rede, CPU), bem como aplicações que não estejam em conformidade com o modelo SPMD; mesmo assim ela apresenta resultados de precisão na faixa de 85% [BMSD95]. Caso algumas dessas restrições ocorram, os autores propõem a utilização de técnicas convencionais mais apropriadas.

Um dos fatores limitantes do sistema é que as aplicações paralelas devem ser desenvolvidas em uma linguagem especial chamada LOOP. Aplicações escritas nesta linguagem podem ser automaticamente analisadas pelo sistema de predição, sem que haja a necessidade de inserção manual de marcadores no código-fonte para que a geração das estimativas de tempo funcione.

[1] aumento de velocidade observado quando se executa um determinado processo em P processadores em relação à execução deste processo em um (1) processador

[2] relação entre o *speedup* e o número de processadores P para identificar a fração de tempo médio em que os processadores estão sendo utilizados

2.4.2 SUIF

SUIF é um sistema de processamento paralelo que se propõe a traduzir automaticamente programas científicos seqüenciais em código paralelo para máquinas paralelas escaláveis. A ferramenta do ambiente SUIF é composta por um compilador com três fases de paralelização na análise e tradução do código seqüencial: análise local de comunicação, análise de sincronização e geração de código. Essas fases são explicadas detalhadamente na referência [AALT95].

Kühnemann et al. apresentam em [KRR04] uma ferramenta para a predição de desempenho das aplicações paralelas geradas pela ferramenta SUIF. A ferramenta, desenvolvida na linguagem de programação C++, adota uma abordagem estática de predição de desempenho, com precisão na escala de 93%, e se adequa a aplicações desenvolvidas em Fortran, C, C++ e Java.

A técnica de modelagem utilizada para a criação de modelos de desempenho que resultarão no tempo de predição, é composta por duas fases: modelagem das comunicações em MPI (*Message Passing Interface*) e do esforço computacional, descritas a seguir:

a. modelagem da fase de comunicação: utiliza-se de funções pré-definidas no código que serão mapeadas no modelo de comunicação. Tais funções são relativas a comunicações do tipo *unicast* (um para um) e *multicast* (um para muitos) de uma biblioteca MPI. Os dados considerados na análise são o tamanho da mensagem e o número de processadores participantes na operação de comunicação. Os valores específicos dos coeficientes de comunicação das funções mapeadas são coletadas de um arquivo referente às propriedades da máquina que envolve a velocidade do enlace de comunicação;

b. modelagem do custo computacional: realiza a criação direta (isto é, sem uma representação intermediária) de um modelo equivalente à aplicação paralela. Ele contabiliza o tempo de execução de operações de baixo nível sobre dados do tipo inteiro e ponto flutuante. Esses dados são avaliados de forma isolada para cada máquina do ambiente paralelo, somente uma vez. A forma da análise é feita por programas de testes de desempenho específicos (programas de *benchmark*) que reproduzem o poder computacional de cada máquina do sistema paralelo. Os valores medidos pelos *benchmarks* e os dados referentes ao custo computacional da aplicação coletados são armazenados em arquivo juntamente com as informações necessárias ao modelo de comunicação.

Os modelos gerados são analisados por um compilador que cria funções correspondentes ao custo computacional da aplicação paralela. As funções obtidas apresentam-se de forma parametrizada representando o tempo de execução da aplicação, permitindo ainda estimar o comportamento e o desempenho de diferentes variantes da aplicação na mesma plataforma paralela.

Apesar da automatização no processo de modelagem da aplicação, é necessária a inserção de marcadores no código para separar diferentes tipos de blocos no programa, como blocos de comunicação, blocos de inicialização de variáveis e blocos computacionais. Caso nenhuma marcação seja inserida no código, a ferramenta considerará a aplicação como sendo um bloco computacional único, e operações de comunicação serão desconsideradas pelo sistema de predição.

2.4.3 PACE

A estratégia de criação de modelos referentes às características da aplicação e dos computadores da plataforma de processamento paralelo também é feita pelo sistema de predição PACE (*Performance Analysis and Characterization Environment*). O sistema, desenvolvido na linguagem de programação Fortran, possui ferramentas independentes para criar os modelos mencionados [JSKN03]:

a. modelagem da aplicação: a análise estática do código fonte da aplicação para a geração do modelo respectivo não requer a inserção de marcadores no código e forma a base do processo de predição, onde são verificadas as estruturas de controle de fluxo (chamadas de funções), freqüência de realização das operações e estrutura da comunicação entre as unidades de processamento da aplicação paralela. A análise do código fonte da aplicação (com suporte a C, Fortran 77 e 90 que utiliza MPI ou PVM - *Parallel Virtual Machine*) cria uma representação do modelo da aplicação na linguagem de *script* [3] PSL (*Parallel Specification Language*) que é compilado para gerar um modelo real da aplicação. O *script* pode ser livremente alterado para modificação e inserção de parâmetros no sistema de predição;

b. modelagem dos recursos computacionais: os recursos computacionais disponíveis são modelados pelas ferramentas de recurso do sistema. Essas ferramentas utilizam uma linguagem de configuração, HMCL (*Hardware Modeling and Configuration Language*), para fazer a modelagem do hardware e definir o desempenho do sistema paralelo que será utilizado para a execução da aplicação.

As ferramentas possuem inúmeros programas de *benchmark* para avaliar o desempenho de CPU, rede e memória para operações de baixo nível sobre tipos de dados em ponto flutuante de uma variedade de plataformas de hardware. A partir disso tem-se um modelo estático em HMCL que pode ser reutilizado para diferentes análises e predições de tempo.

Após a construção dos modelos, eles são analisados por um mecanismo de avaliação da ferramenta PACE que disponibilizará ao usuário:

- tempos de predição (para diferentes sistemas e aplicações) para serem avaliados na forma de equações analíticas pela própria ferramenta;
- escalabilidade dos recursos e da aplicação para que possam ser explorados;
- utilização dos recursos do sistema para a predição;
- visualização do estado atual de execução das aplicações por mecanismos de rastreamento.

A análise dos resultados fornecidos pelo sistema PACE indica uma precisão média de 95% nas estimativas.

[3] lista de comandos, baseada numa determinada linguagem, que podem ser executados sem interação do usuário para executar determinadas operações de maneira automática

2.4.4 PAMELA

Para as predições nesta abordagem, são criados dois modelos. O primeiro é referente ao modelo da aplicação paralela e o segundo, referente ao modelo das máquinas e da rede que compõem a plataforma paralela. A abordagem proposta pelo modelo PAMELA (*PerformAnce ModEling LAnguage*), desenvolvida em C, é esquematizada na Fig. 2.2.

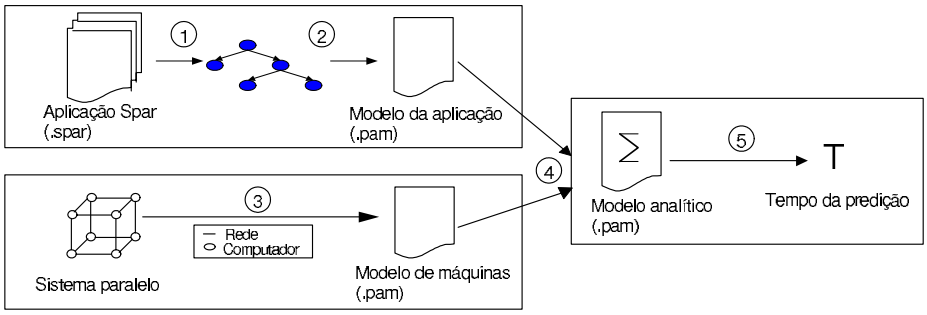

Fig. 2.2: Etapas do processo de predição da abordagem do modelo PAMELA

Geração do modelo da aplicação

As aplicações para a ferramenta PAMELA são escritas na linguagem de programação Spar [vR01], que é uma extensão da linguagem de programação Java incrementada com primitivas de paralelismo no código. Por meio da extensão criada, pode-se explicitar quais blocos de código serão executadas de forma paralela ou seqüencial.

A geração do modelo da aplicação é formada pelas etapas 1 e 2 da Fig. 2.2. Essas etapas consistem basicamente na conversão do código fonte da aplicação em Spar, para uma representação intermediária na linguagem de programação Vnus IR(*Intermediate Representation*) [vR00]. Esta representação intermediária, criada por um compilador específico chamado Timber [vRKS01], é então analisada por um outro compilador para a geração do modelo da aplicação na representação em PAMELA.

A representação do modelo da aplicação verifica a execução de operações de baixo nível (aritméticas, lógicas etc.) existentes na representação intermediária para contabilizar o número de vezes que cada tipo de operação é realizada sobre determinados tipos de dados da linguagem. No modelo da aplicação também são avaliados os efeitos causados pela contenção no uso de recursos compartilhados (CPU, rede).

Geração do modelo das máquinas

A geração do modelo das máquinas, representada pela etapa 3 na Fig. 2.2, consiste na execução de programas de *benchmark* simples, capazes de analisar fatores como efeitos da memória cache,

operações de comunicação e computação para diversos tipos de operações lógicas, aritméticas e relacionais.

Os dados obtidos são armazenados em arquivo e fornecem informações do sistema paralelo que permitem analisar o seu poder computacional em *flops*, como nas abordagens apresentadas nas referências [vG01], [BMSD95], [AALT95] e [JSKN03].

Geração das predições

A predição é gerada por meio da análise do modelo analítico (etapa 5 na Fig. 2.2), obtida com a combinação do modelo das máquinas com o modelo da aplicação (etapa 4 na Fig. 2.2), como ocorre também em outras abordagens similares [BMSD95] [SC96] [KRR04] [FJ95] e [Fah95].

Os possíveis atrasos causados pela contenção no uso de recursos compartilhados, velocidade do enlace de comunicação e outros fatores são avaliados nesta fase, pois foram representados por parâmetros no modelo da aplicação.

A técnica de predição apresenta uma boa precisão nos resultados, tratando efeitos causados pela contenção no uso de recursos compartilhados de uma forma simples, porém eficiente. Os modelos gerados podem ser reutilizados para diferentes configurações de sistema e de tamanho da aplicação para analisar a eficiência dos mecanismos de escalonamento e particionamento de dados. Um dos fatores que vêm limitar o uso prático da ferramenta é que ela exige o uso de uma linguagem de programação não padronizada, além da colocação de marcadores especiais no programa.

A análise dos resultados fornecidos pelo sistema PAMELA indica que sua precisão situa-se na faixa de 80% a 90% [vG03].

2.5 Considerações

A predição de desempenho é um instrumento importante para a análise de aplicações paralelas. Foram vistas neste capítulo as técnicas de predição de desempenho e a forma com que cada uma delas trata do problema de geração de estimativas de tempo.

Foi visto ainda que técnicas dinâmicas realizam algum tipo de simulação ou medição para estimar o tempo de execução de aplicações enquanto que técnicas analíticas baseiam-se em informações estáticas da aplicação e dos computadores da plataforma de processamento paralelo. Essas informações, na maioria das vezes, são armazenadas em modelos que seguem uma determinada notação. Algumas propostas de ferramentas de predição baseadas em técnicas analíticas abordam o problema da criação dos modelos dividindo-o em duas partes: modelo da aplicação e modelo das máquinas.

Um dos problemas envolvidos na geração do modelo da aplicação é a necessidade da inserção de marcadores explícitos no código-fonte da mesma. Sistemas como PACE e PAMELA exigem que isso seja feito. As aplicações para PACE podem ser escritas em Fortran (77 ou 90) ou C. PAMELA

pressupõe o uso de uma nova linguagem de programação, chamada Spar, que procura modelar efeitos de atraso causados pela contenção no uso de recursos. A ferramenta PerPreT também propõe o uso de uma nova linguagem de programação, chamada LOOP, que limita as aplicações àquelas descritas pelo modelo de programação SPMD (*Single Program Multiple Data*). Porém, PerPreT não exige marcação explícita no código.

Como a proposta deste trabalho é o desenvolvimento de um sistema de predição para a plataforma JoiN, foi estudada a possibilidade de integração das ferramentas mencionadas com esta plataforma; porém várias limitações foram encontradas. Pelo fato da plataforma JoiN adotar uma filosofia de ser portável, ela foi totalmente desenvolvida em Java, e os sistemas que fazem parte de sua arquitetura atual ou que estão em desenvolvimento para posterior integração, adotam esta mesma filosofia. Tanto os sistemas PerPreT, SUIF, PACE e PAMELA foram desenvolvidos em diferentes linguagens de programação, como Fortran, C++, Fortran e C, respectivamente, o que diminui as possibilidades de integração direta com a plataforma de processamento paralelo JoiN. Além disso, um outro problema é que os sistemas de predição de desempenho são, de uma forma geral, construídos para plataformas de processamento paralelo específicas, as quais possuem particularidades, tais como a linguagem em que as aplicações são desenvolvidas, sistema de escalonamento de tarefas e a arquitetura dos computadores que compõem a plataforma paralela, responsáveis pelo processamento da aplicação.

Em função do custo envolvido na geração de predições e da necessidade de ferramentas automatizadas que exijam o mínimo de esforço do programador para realizar predições, este trabalho propõe uma nova ferramenta de predição utilizando uma abordagem de predição analítica, para a plataforma de processamento paralelo JoiN.

O próximo capítulo dedica-se a apresentar algumas características da plataforma mencionada que serão importantes para o sistema e a técnica de predição adotada para que a ferramenta gere tempos de predição automaticamente.

Capítulo 3

Predição de desempenho em JoiN

Este capítulo dedica-se a apresentar uma técnica de modelagem e predição analítica de desempenho, chamada PAMELA Estendida, e algumas características de uma plataforma de processamento paralelo, chamada JoiN.

O intuito é introduzir os conceitos necessários para a apresentação, no próximo capítulo, de um sistema de predição de desempenho para a plataforma de processamento paralelo JoiN, utilizando a técnica de predição mencionada para a geração de modelos e para a obtenção de estimativas de tempo.

3.1 Introdução

No desenvolvimento de um sistema de predição de desempenho, estão envolvidos dois fatores importantes. O primeiro é a técnica de modelagem utilizada para a construção de modelos e para a obtenção de estimativas de tempo, e o segundo é relativo às características da plataforma de processamento paralelo na qual esta técnica será aplicada.

Quanto à técnica de modelagem, é importante adotar uma que trate de diversos fatores de atraso que influenciem no tempo de execução de uma aplicação, que disponibilize resultados de boa qualidade e que o tempo e o custo envolvidos para a obtenção da predição sejam baixos.

Quanto à plataforma de processamento paralelo, é necessário conhecer, por exemplo, a forma com que as aplicações são fragmentadas em tarefas e distribuídas entre os computadores da plataforma. Isso permite identificar de que forma as tarefas de uma aplicação serão executadas.

Em função disso, as próximas seções deste capítulo dedicam-se a discutir mais detalhadamente os dois fatores mencionados. Primeiro é apresentada uma técnica de modelagem e predição de desempenho baseada em uma abordagem analítica chamada PAMELA Estendida. Em seguida, algumas características da plataforma de processamento paralelo JoiN, úteis para a técnica de predição, serão ressaltadas e analisadas.

3.2 Técnica de modelagem e predição de desempenho

Definir formalmente uma técnica de modelagem e predição de desempenho que leve em consideração todas as características descritas até aqui não é trivial. Há diversas técnicas com diferentes abordagens quanto à criação de modelos e obtenção de tempos de predição, cada uma com suas vantagens e peculiaridades em relação às demais.

Uma técnica que pode ser utilizada para a construção de modelos e realizar predições é aquela adotada pela ferramenta PAMELA [vG01]. Esta ferramenta implementa um modelo de mesmo nome [vG96].

PAMELA é uma técnica de modelagem e predição de desempenho baseada em uma abordagem analítica. Ela propõe um mecanismo de modelagem que considera efeitos de atraso causados por processos de computação e de comunicação e apresenta uma forma nova, mas simples, de tratar a contenção interna sobre a utilização de recursos compartilhados de uma plataforma de processamento paralelo. Os recursos considerados são a rede e os computadores que compõem a plataforma paralela.

A contenção interna sobre os computadores do sistema paralelo permite analisar efeitos causados pela utilização da memória e processos de computação de uma aplicação que são executados de forma concorrente. A análise da contenção interna sobre a rede é feita partindo-se do mesmo princípio com o que é feito nos computadores, mas desta vez analisa-se apenas quando o recurso de rede é utilizado de forma concorrente.

Além da contenção interna, as operações de atraso presentes na aplicação e as características dos computadores da plataforma de processamento paralelo são tratadas por PAMELA na forma de 3 modelos (Fig. 3.1). O primeiro é referente ao modelo da aplicação paralela, o segundo referente ao modelo das máquinas e da rede que compõem a plataforma paralela e o terceiro equivale à combinação dos dois anteriores para produzir tempos de predição.

Para permitir que a técnica proposta por PAMELA trate também de efeitos de contenção externa, além da interna, Huerta [Yer03] propôs uma extensão da técnica apresentada por PAMELA, chamando-a de PAMELA Estendida. Nela, são adotados intervalos para a criação do modelo de máquinas e os tempos de predição passam a ser representados na forma de intervalos de valores, o que permite melhorar a qualidade dos dados tratados pela abordagem original do modelo PAMELA. Além disso, o uso de intervalos permite representar de forma mais adequada a variação de desempenho dos recursos analisados, os quais são causados pela contenção externa.

Os três modelos podem ser criados, por meio de ferramentas ou de forma manual, para representar as características da aplicação, da plataforma de processamento paralelo e a combinação dos modelos para a geração de intervalos de tempo de predição. Para isso, é feita uma breve descrição da notação PAMELA Estendida utilizada para a geração de cada modelo.

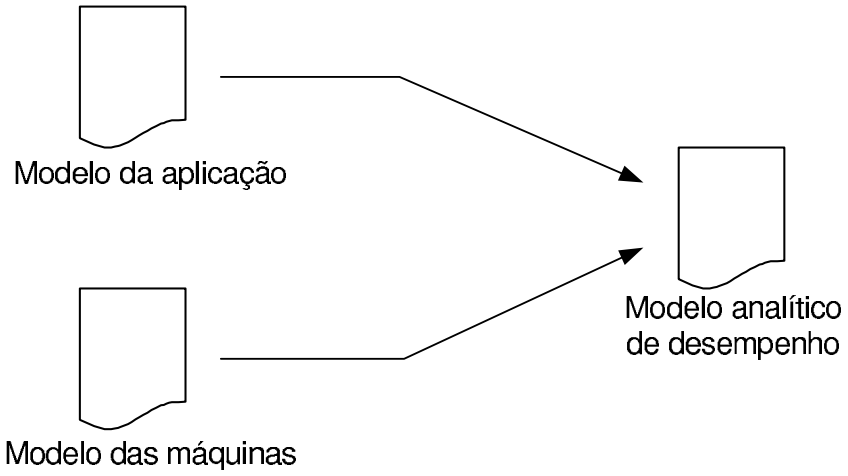

Fig. 3.1: Modelo de desempenho

3.2.1 Modelo da aplicação

O modelo da aplicação visa representar as características de uma aplicação paralela e contém informações sobre a forma com que a aplicação será executada em uma plataforma de processamento paralelo. A quantidade de tarefas por computador, operações de computação e de comunicação e o fluxo de dados entre as tarefas farão parte do modelo que deve conter todas as informações necessárias para prever como uma aplicação poderá ser executada em um conjunto de computadores de uma plataforma de processamento paralelo.

Para identificar a forma com que os recursos (computador e rede) de uma plataforma paralela são utilizados é necessário conhecer a quantidade de tarefas que são executadas pelos computadores da plataforma. Tal informação, presente no modelo da aplicação, é fundamental para analisar efeitos de contenção interna.

PAMELA possui um conjunto de operadores que permitem modelar uma aplicação paralela em relação a operações de atraso de tempo gerados por processos de comunicação e de computação. A seguir é apresentado um conjunto reduzido de operadores básicos da linguagem. Uma relação completa com a descrição de cada operador pode ser encontrada de forma detalhada na referência [vG96].

- $delay(\tau)$: representa o atraso de tempo τ para algum tipo de operação realizada pela aplicação;
- ; : operador binário de execução de processos seqüenciais;
- ||: operador binário de execução de processos em paralelo;
- $seq(j = a, b)Op_j$: operador de redução seqüencial, equivalente a $Op_a; ...; Op_b$;
- $par(j = a, b)Op_j$: operador de redução paralelo, equivalente a $Op_a||...||Op_b$;

- $if\{Op_a\}else\{Op_b\}$: operador de execução condicional, que equivale a execução exclusiva de um bloco Op_a ou um Op_b. Este operador não possui uma condição associada, pois possui o intuito de modelar apenas a possibilidade da execução exclusiva de diferentes blocos;
- $use(r, \tau)$: operador que indica o uso de um determinado recurso r por um tempo τ, para representar a contenção no uso do recurso no modelo da aplicação. O uso de um recurso r está associado a um fator de contenção (F_r), denotando a quantidade de processos que utilizam-no simultaneamente (em paralelo).

Os operadores PAMELA definidos são suficientes para modelar uma grande quantidade de aplicações paralelas e efeitos de atraso que afetam o seu tempo de execução [vG96]. Para ilustrar de forma simplificada como esses operadores podem ser utilizados, consideremos as seguintes variáveis:

- um computador p e outro q ($p, q \in \{1, 2, ..., P\}$, onde P é a quantidade de computadores);
- uma rede $net_{p \rightarrow q}$, a qual representa o enlace de comunicação entre os computadores p e q, sendo $p \neq q$.
- uma tarefa $tk_{i,j}$ (tarefa j de lote i);
- um atraso de tempo $\tau_{j,p}$ que representa o tempo associado ao processamento de uma tarefa $tk_{i,j}$ em um computador p;
- um atraso de tempo $\tau_{p \rightarrow q}$ que representa o tempo associado ao envio de dados de um computador p para um q.

Para modelar a contenção interna de um computador p, que recebe uma quantidade M de tarefas independentes entre si para processamento em paralelo, utiliza-se a instrução use, a qual produz um modelo simplificado equivalente ao da Eq. (3.1).

$$L = par(j = 1, M)\{use(comp_p, \tau_{j,p});\} \tag{3.1}$$

Na equação, as informações de τ_j estão presentes no código-fonte da aplicação, o qual precisa ser analisado para obtê-las. As informações envolvem as operações realizadas sobre os diferentes tipos de dados, os quais podem ser de adição, subtração, multiplicação ou divisão, por exemplo. As operações podem estar dentro de algum tipo de estrutura da linguagem de programação utilizada para a implementação da aplicação, como em blocos de execução contendo estruturas aninhadas de repetição ou de condição, ou em algum objeto. Desta forma, τ_j pode representar uma composição de outras instruções de atraso de tempo.

Em outro caso, desta vez para ilustrar o processamento de uma quantidade de M tarefas de forma seqüencial em um computador p, é produzido um modelo L equivalente ao da Eq. (3.1), porém com o uso da instrução seq, a qual é utilizada para representar a execução de instruções de maneira seqüencial. No modelo, ilustrado na Eq. (3.2), uma tarefa $tk_{i+1,j}$, cujo tempo de processamento é τ_{i+1},

só pode iniciar seu processamento depois que a tarefa anterior $tk_{i,j}$, cujo tempo de processamento é τ_i, for concluída.

$$L = seq(j = 1, M)\{use(comp_p, \tau_{j,p});\} \tag{3.2}$$

A modelagem da rede é feita de forma semelhante ao que é feito na modelagem das tarefas que são processadas nos computadores. Para isso, ao invés de considerar no modelo o atraso de tempo relacionado a uma tarefa, considera-se o atraso de envio de dados $\tau_{p \rightarrow q}$. Este atraso de tempo pode estar associado ao envio de uma tarefa, de um conjunto delas ou do respectivo resultado de seu processamento. A Eq. (3.3) ilustra um modelo no qual são enviadas sequencialmente M tarefas de um computador p para um q.

$$L = seq(j = 1, M)\{use(net_{p \rightarrow q}, \tau_{p \rightarrow q});\} \tag{3.3}$$

Para que a modelagem da contenção interna seja possível, é necessário que as informações da estrutura do fluxo de execução da aplicação (quantidade de tarefas e fluxo de dados entre elas) estejam disponíveis de alguma forma, seja no próprio código-fonte da aplicação ou em algum outro local.

3.2.2 Modelo de máquinas

O modelo das máquinas procura descrever as características de computação e de comunicação dos computadores e interconexões de rede que compõem a plataforma de processamento paralelo, sejam eles homogêneos ou heterogêneos, dedicados ao processamento de aplicações paralelas ou compartilhados com outras.

O modelo procura refletir a variação de desempenho de cada computador da plataforma paralela na execução de operações básicas sobre diferentes tipos de dados e na velocidade de comunicação com outros computadores da plataforma paralela. Em várias abordagens utilizadas para representar as características dos computadores de uma plataforma de processamento paralelo, as informações são armazenadas na forma de valores pontuais, o que também é feito pela abordagem original da técnica de modelagem PAMELA.

Entretanto, um dos grandes problemas presentes na representação das características dos computadores e da rede de uma plataforma paralela é modelar a contenção externa. Uma das formas com que isso pode ser feito é com o uso de intervalos. Conforme apresentado na Seção 2.2.3, é possível representar as características dos computadores na execução de algum tipo de operação na forma de intervalos de valores do tipo $[t_{min}, t_{max}]$, ao invés de valores pontuais, estabelecendo assim um tempo mínimo (t_{min}) e um tempo máximo (t_{max}) na execução de algum tipo de operação de computação ou de comunicação. Sendo assim, o modelo de máquinas pode ser definido pelos parâmetros:

$op\_data_p = delay(inter\_op\_data_p)$

e

$send_{p \to q}(dataSize) = delay(inter\_send_{p \to q}(dataSize))$.

O parâmetro $inter\_op\_data_p$ é um intervalo de valores, que representa o tempo gasto na execução de operações *op* (adição, subtração, multiplicação, divisão etc.) sobre determinados tipos de dados *data* (inteiro, ponto flutuante, caracter etc.) no computador p, $1 \leq p \leq P$. O parâmetro $inter\_send_{p \to q}(dataSize)$ indica um intervalo de valores que representa o tempo gasto no envio de dados de tamanho $dataSize$ de um computador p para um q.

Para representar o desempenho de cada computador na forma de um intervalo, é importante que sejam realizadas várias medições para cada tipo de operação a ser analisada. A partir do conjunto de medições, é calculado um intervalo que representa o desempenho esperado do computador para o tipo de operação analisada.

Definidos formalmente os modelos de máquinas e da aplicação, pode-se criar um novo modelo a partir deles, o qual representa a estimativa de tempo da aplicação na forma simbólica (ou analítica). Este formato simbólico permite utilizar o modelo para estimar tempos de execução de uma aplicação sob diferentes condições de processamento.

A próxima seção discute a geração do modelo analítico de desempenho.

3.2.3 Modelo analítico de desempenho

O modelo analítico de desempenho representa o modelo do qual é obtido o tempo de predição de uma aplicação para alguma configuração de computadores e de rede. Ele é composto por diversas expressões matemáticas que representam um tempo T de execução de uma aplicação na forma simbólica.

A construção das expressões matemáticas, que compõem o modelo analítico de desempenho, é baseada na análise do modelo da aplicação sob duas suposições, sendo ambas relacionadas à quantidade de recursos da plataforma de processamento paralelo, como faz PAMELA:

- recursos são ilimitados: não há contenção interna nos recursos da plataforma paralela;
- recursos são limitados: há possibilidade de contenção interna nos recursos da plataforma paralela.

Na análise do primeiro caso, onde inexiste contenção, a utilização de um determinado recurso de forma paralela implica que o tempo gasto na sua utilização equivale à parte do bloco paralelo que utiliza o recurso por mais tempo. No modelo, isso é representado com o uso do operador matemático max, o qual substitui as instruções paralelas do modelo da aplicação. Na análise deste primeiro caso, o modelo L produz uma única expressão que pode ser representada por T_0, a qual indica o tempo de execução do modelo sem levar em conta a contenção. Para produzir a expressão de T_0, é feita uma substituição das instruções PAMELA a partir do modelo L, que são:

- $a; b$ é substituído por $\tau_a + \tau_b$;
- $a||b$ é substituído por $max(\tau_a, \tau_b)$;
- $par(j = a, b)Op_j$ é substituído por $max_{j=a,b}(\tau_j)$;
- $seq(j = a, b)Op_j$ é substituído por $sum_{j=a,b}(\tau_j)$;
- $use(r, \tau)$ é substituído por $delay(\tau)$.

Na análise do segundo caso, onde há contenção, a utilização de um determinado recurso de forma paralela implica que o tempo gasto pela sua utilização equivale ao somatório do tempo gasto por cada um dos blocos que tentam utilizar o recurso de forma paralela. No modelo, isso é representado com o uso do operador de somatório sum. Para este caso, PAMELA considera a carga imposta em cada recurso separadamente. Nesta fase, o modelo L produz uma expressão T_r para cada recurso r ($r \in \{1, 2, ..., R\}$, onde R é a quantidade de recursos) da plataforma paralela. Para isso, são aplicadas as seguintes substituições no modelo de aplicação L:

- $a; b$ é substituído por $\tau_a + \tau_b$;
- $a||b$ é substituído por $\tau_a + \tau_b$;
- $par(j = a, b)Op_j$ é substituído por $sum_{j=a,b}(\tau_j)$;
- $seq(j = a, b)Op_j$ é substituído por $sum_{j=a,b}(\tau_j)$;
- $use(r, \tau)$ é substituído por $delay(F_r \times \tau)$.

Para a geração das expressões T_0 e T_r, os parâmetros op_data_p e $send_{p \to q}(dataSize)$ do modelo da aplicação L são convertidos em suas representações de atraso de tempo na forma simbólica, equivalentes aos parâmetros $inter\_op\_data_p$ e $inter\_send_{p \to q}$ do modelo de máquinas, respectivamente.

Após a geração das expressões de T_0 e T_r, pode-se representar o tempo T de execução de uma aplicação como sendo o maior valor entre eles, conforme a Eq. (3.4).

$$T = max_{j=0,R}(T_j) \tag{3.4}$$

É importante mencionar que a geração das expressões que irão compor o modelo analítico de desempenho depende muito da carga de trabalho alocada a cada computador da plataforma de processamento paralelo. Esta alocação é realizada pelo sistema de escalonamento de tarefas adotado e é fundamental conhecê-lo profundamente para que seja possível analisar efeitos de atraso causados pela execução da aplicação na plataforma paralela, como também para analisar efeitos dos dois tipos de contenção verificados por esta abordagem: a interna e a externa.

3.2.4 Tempos de predição

A fase de geração do intervalo de tempo de predição de uma aplicação consiste na última etapa da técnica de predição apresentada. A partir dela é possível identificar a qualidade dos resultados fornecidos pela abordagem em relação a tempos de execução real para alguma aplicação paralela analisada.

Para obter o tempo de predição, as expressões matemáticas que compõem a expressão de T (Eq. (3.4)) são avaliadas. Para isso, elas sofrem um processo de substituição simbólica, na qual todos os parâmetros do tipo $t\_op\_data_p$ e $t\_send_{p \rightarrow q}(dataSize)$ das expressões são substituídas pelos seus respectivos intervalos numéricos de valores $[t_{min}, t_{max}]$, presentes no modelo de máquinas. No processo de substituição, cada um dos parâmetros (representando algum tipo de operação) das expressões de T é substituído pelo respectivo intervalo do computador responsável pelo processamento da operação. Após a substituição de todos os parâmetros simbólicos das expressões por um determinado intervalo, é necessário calcular o valor de T por meio da avaliação das expressões matemáticas que o compõem.

O valor expressado por T, na forma de um intervalo, equivale ao tempo de execução esperado para a aplicação analisada. Nesta estimativa, são verificados os efeitos da contenção interna com o uso de PAMELA por meio da análise do modelo da aplicação, e também efeitos da contenção externa com o uso de intervalos para expressar o desempenho dos computadores da plataforma de processamento paralelo na forma de um modelo de máquinas.

Devido à natureza analítica do modelo de desempenho, a etapa de obtenção de T pode ser realizada com diferentes modelos de máquinas, cada uma com uma determinada quantidade de computadores e diferentes desempenhos. Com isso, pode-se criar ou modificar, de forma manual ou por meio de alguma ferramenta, inúmeras configurações de computadores, reais ou fictícias, que formarão o modelo de máquinas com o intuito de verificar algumas questões como:

- o impacto causado pela adição de novos computadores no modelo de máquinas;
- a viabilidade prática de execução de uma determinada aplicação a partir do tempo estimado;
- a necessidade de reprojetar uma parte ou toda a aplicação paralela para que ela forneça resultados com tempos de execução satisfatórios.

O intervalo de tempo calculado também pode ser utilizado para fazer análises de estimativas de speedup e de eficiência da plataforma paralela na execução de uma aplicação paralela, dispensando a necessidade de efetivamente executá-la em uma plataforma paralela, tarefa que pode ter custo alto.

Esta seção apresentou a técnica de modelagem e predição analítica de desempenho baseada em PAMELA, chamada PAMELA Estendida. Ela é base do trabalho pois representa a abordagem utilizada pelo sistema de predição proposto para a obtenção de tempos de predição. Para isso, é importante definir algumas características da plataforma de processamento paralelo que são utilizadas para definir como os computadores da plataforma paralela executarão as tarefas de uma aplicação.

Em vista disso, a próxima seção apresenta a plataforma de processamento paralelo JoiN e algumas de suas características que são importantes na implementação de um sistema de predição.

3.3 A Plataforma de processamento paralelo JoiN

A plataforma de processamento paralelo JoiN propõe-se a ser um computador virtual maciçamente paralelo baseado na Internet. Ela nasceu como fruto de pesquisas iniciadas em 1996 [Yer98] [Hen99] e, desde então, vem sendo constantemente desenvolvida e aperfeiçoada.

Formada por uma estrutura baseada em componentes e inteiramente desenvolvida em Java, a plataforma JoiN tem como principais características sua grande portabilidade, já que pode ser executada em qualquer computador que dispõe de uma implementação da Máquina Virtual Java [LY99]. Dentre outras características da plataforma está sua flexibilidade de operação e de configuração.

Os componentes em JoiN foram desenvolvidos na forma de serviços, também chamados de subsistemas de uma plataforma paralela. Cada sub-sistema fornece alguma funcionalidade para permitir que a plataforma paralela execute as aplicações submetidas a ela de forma segura e eficiente. Subsistemas de comunicação, autenticação e controle de acesso, segurança na execução de aplicações, recuperação de falhas, controle de aplicações, escalonamento de tarefas etc. são exemplos que oferecem algumas funcionalidades à plataforma JoiN.

A estrutura da plataforma JoiN é baseada em um modelo de plataforma paralela sobre o qual as aplicações são escalonadas e executadas. A forma com que isso é feito é muito importante para definir como uma aplicação projetada para tal plataforma será executada de maneira eficiente nos computadores.

Esta seção apresenta de forma simplificada o modelo de processamento paralelo da plataforma JoiN, a estrutura e a característica de suas aplicações, e o funcionamento dos sistemas de escalonamento de tarefas e de seleção de um conjunto de máquinas para executar alguma aplicação.

3.3.1 Modelo de processamento paralelo

JoiN é um sistema composto por quatro tipos distintos de componente: servidor, coordenador, trabalhador e de administração (jack). Cada computador participante na plataforma JoiN executa pelo menos um destes componentes que, dependendo de sua função, poderá estar presente em um ou muitos computadores simultaneamente. A Fig. 3.2 apresenta um diagrama que ilustra esta organização. Brevemente descritas, as atribuições de cada um destes componentes são:

- um nó de entrada (servidor): atua como primeiro ponto de contato de novos computadores que entram no sistema para atuar como nós trabalhadores;

- um ou mais nós de administração: viabiliza a administração da plataforma, enviando, por exemplo, requisições de execução de aplicações ao nó servidor;
- vários nós de coordenação de grupo: responsáveis pelo monitoramento, divisão, distribuição e coleta do resultado das tarefas dentro de um grupo;
- vários nós de processamento (nós trabalhadores): dedicados exclusivamente ao processamento das tarefas enviadas pelo nó coordenador.

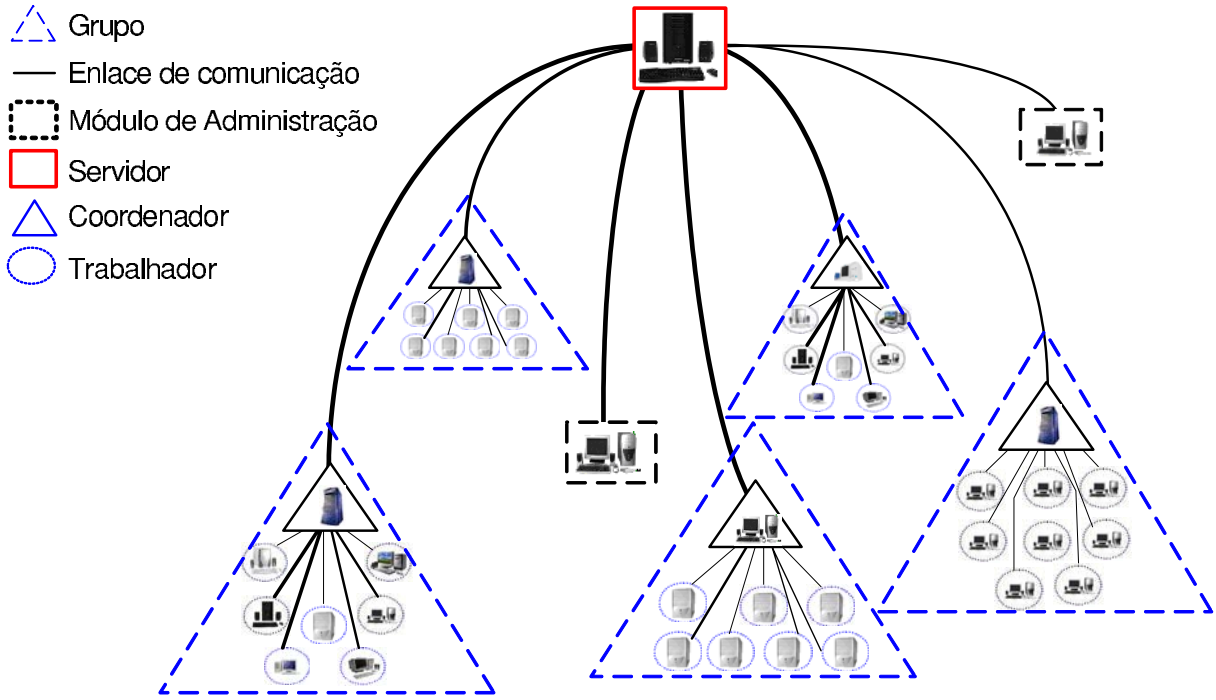

Fig. 3.2: Estrutura do sistema paralelo JoiN

Um grupo na plataforma paralela é formado por um computador coordenador e por vários trabalhadores. Os grupos podem ser independentes, e seus nós têm a capacidade de executar diferentes aplicações sobre diferentes blocos de dados em linhas de execução (*threads*) independentes. Computadores trabalhadores só podem se comunicar com seus respectivos coordenadores, os quais são capazes de se comunicar com outros coordenadores e com o servidor. O módulo de administração comunica-se apenas com o servidor.

3.3.2 Características das aplicações

A estrutura das aplicações, que segue a definição apresentada na referência [Luc02], não se restringe ao modelo SPMD. Ela é formada por um conjunto de A lotes de tarefas e por um conjunto de B relações de dependência de dados entre os lotes.

A cada lote de tarefas B_i, está associado um índice i ($i = 1..A$) e uma cardinalidade c_i, definida como sendo o número de tarefas do lote de índice i. Uma tarefa de um lote é definida como uma seqüência ordenada de operações aritméticas, matemáticas, de desvio condicional ou de atribuição, entrada ou saída de dados, que implementa algum tipo de processamento finito e determinista sobre um bloco de dados de entrada a fim de produzir um bloco de dados de saída [Luc02].

Define-se também que os lotes de tarefas B_1 e B_A possuem cardinalidade igual a um ($c_1 = c_A = 1$) e são responsáveis pelo processamento do bloco inicial e final da aplicação, respectivamente. Um bloco de dados de saída de uma tarefa só pode fazer parte de um bloco de dados de entrada de uma tarefa de índice maior.

Para especificar o paralelismo da aplicação, é utilizada a linguagem PASL (*Parallel Application and Specification Language*) [Luc02]. Ela é composta por 2 seções funcionais, como mostra a Fig. 3.3.

```
//seção de declarações
path=aplicacoes/numerosPrimos
B1=DivideDados.class;
B2=ProcessaDados.class;
B3=ColetaResultados.class;

//seção de ligações
T1=B1(1)«dados.dat;
T2=B2(100)«T1;
T3=B3(1)«T2»resultado.dat;
```

Fig. 3.3: Especificação de uma aplicação paralela em PASL

- declarações: contêm os caminhos das classes e dos dados que serão utilizados pela aplicação. No exemplo, a variável $path$ indica o diretório e $B1$, $B2$, $B3$ as classes utilizadas pela aplicação;
- ligações: indicam os fluxos de dados entre os lotes de tarefas, sua cardinalidade e os arquivos de entrada e saída da aplicação. No exemplo tem-se que $T1$, que compõe o lote $B1$ com cardinalidade 1, recebe de entrada o arquivo $dados.dat$ para processamento. A tarefa $T2$, que compõe o lote $B2$ com cardinalidade 100, lê a saída produzida pelo processamento de $T1$. Finalmente a tarefa $T3$, que faz parte do lote $B3$ com cardinalidade 1, lê as saídas de $T2$ e escreve o resultado final no arquivo $resultado.dat$.

3.3.3 Mecanismo de escalonamento de tarefas

Define-se escalonador de tarefas como sendo a estrutura, em uma plataforma de processamento paralelo, responsável pela alocação das tarefas de uma aplicação aos computadores trabalhadores de forma a equilibrar a carga de trabalho entre eles e assim obter o melhor desempenho possível.

Lotes de tarefas que apresentam cardinalidade igual a um são alocadas diretamente no computador coordenador, enquanto que os demais lotes são alocados para os computadores trabalhadores, de forma proporcional ao seu fator de desempenho W_i ($i = 1..P$, onde P é o número de computadores trabalhadores), fator este que indica a capacidade de processamento do computador em relação aos demais do seu grupo.

Para calcular o fator de desempenho, entre os computadores trabalhadores e seu respectivo coordenador, executam-se programas de *benchmark* em cada um deles. Sendo App uma aplicação com vários testes de *benchmark*, $T(App, M_i)$ o tempo de execução da App no computador trabalhador M_i, e $min_{j=1,P}T(App, M_j)$ o menor tempo de execução da App entre os computadores trabalhadores de um grupo, o fator de desempenho W_i de cada computador trabalhador será definido pela Eq. (3.5).

$$W_i = \frac{min_{j=1,P}T(App, M_j)}{T(App, M_i)} \tag{3.5}$$

A análise da Eq. (3.5) indica que o fator de desempenho de um computador trabalhador é relativo ao mais veloz na execução da aplicação de testes App e assume um valor sempre menor ou igual a um.

A classificação a partir do fator de desempenho W_i, possibilita calcular a quantidade S_i de tarefas alocadas estaticamente para cada computador trabalhador. Para um total de P computadores trabalhadores, a quantidade de tarefas alocada a cada um deles, de um lote de tarefas B_k de cardinalidade c_k, será definido pela Eq. (3.6).

$$S_i = \left\lceil \frac{f \times c_k \times W_i}{\sum_{j=1}^{P} W_j} \right\rceil \tag{3.6}$$

Nesta equação, f denota a fração do total de tarefas c_k que será alocada inicialmente pela alocação estática.

Para ilustrar como é feita a alocação de tarefas, considere M_1, M_2 e M_3 três computadores executando um componente trabalhador com fatores de desempenho equivalentes a $W_1 = 1,0$, $W_2 = 0,4$ e $W_3 = 0,2$, respectivamente. Sendo B_A um lote A de tarefas com cardinalidade $c_A = 128$ e $f = 1$, indicando que 100% das tarefas serão alocadas de forma estática, tem-se que a quantidade de tarefas alocada a M_1, M_2 e M_3 é de $S_1 = 80$, $S_2 = 32$ e $S_3 = 16$, respectivamente.

Quando o valor atribuído a f está entre 0 e 1, o escalonamento das tarefas é feito de forma mista: uma parte das tarefas é alocada de forma estática e a outra de forma dinâmica. Por exemplo, se o valor de f for igual a $0,3$, então 30% do total de tarefas é alocado estaticamente e as demais serão alocadas dinamicamente durante o processamento. Quando uma tarefa é concluída, o escalonador submete uma nova tarefa (que ainda não foi alocada) para o respectivo computador trabalhador. Depois da alocação de todas as tarefas de um lote pelo escalonador, inicia-se um processo de replicação de tarefas. A replicação é a alocação de cópias das tarefas de um determinado lote B_k em computadores trabalhadores que tenham capacidade para receber mais tarefas para processamento. A alocação

dinâmica das réplicas é feita da mesma forma que as tarefas originais. Para cada tarefa concluída, é alocada uma réplica no computador trabalhador. O funcionamento do sistema de escalonamento apresentado pode ser visto de forma detalhada na referência [Luc02].

O sistema de replicação de tarefas fornece tolerância a falhas no nível da aplicação, pois ameniza atrasos decorrentes de possíveis mudanças repentinas de desempenho nos computadores trabalhadores, ou também pela saída ou exclusão de um ou mais computadores de um grupo devido a algum motivo desconhecido. Com este esquema, tem-se a garantia de que a aplicação será concluída, desde que haja pelo menos um trabalhador em atividade.

3.3.4 Sistema de seleção de máquinas

O sistema de seleção de máquinas da plataforma de processamento paralelo JoiN fornece um mecanismo, na forma de uma interface gráfica, que permite selecionar uma determinada quantidade de computadores para o processamento de uma aplicação na plataforma paralela.

A seleção de computadores pode levar em conta o fator de desempenho descrito pela Eq. (3.5), isto é, a seleção de uma quantidade de P computadores poderá escolher, por exemplo, os mais velozes segundo esse fator.

A disponibilidade de tal sistema de seleção na plataforma JoiN permite variar a quantidade de computadores sem que seja necessário remover manualmente computadores até que a quantidade desejada seja atingida.

3.4 Considerações

Este capítulo apresentou brevemente alguns conceitos relacionados à construção de modelos para armazenar e representar informações de desempenho de computadores ou de atrasos presentes em uma aplicação paralela.

Foi visto como a notação da técnica de modelagem e predição analítica de desempenho PAMELA Estendida pode ser utilizada para modelar operações de atraso (comunicação e computação) presentes em uma aplicação paralela considerando efeitos de contenção interna. Apresentou-se como a modelagem das características de desempenho dos computadores de uma plataforma de processamento paralelo pode ser feita com o uso de intervalos, permitindo expressar de forma simples a variação de desempenho dos computadores de uma plataforma paralela e, conseqüentemente, atrasos que podem ser causados pela contenção externa.

Como a proposta do trabalho é o desenvolvimento de uma ferramenta de modelagem e predição de desempenho para a plataforma JoiN, foram apresentadas as características mais importantes de tal plataforma para a criação de uma ferramenta baseada no modelo PAMELA Estendida.

O próximo capítulo apresenta a proposta do trabalho de forma detalhada. Será visto como é feita a geração dos modelos de desempenho, por meio de ferramentas construídas especialmente para tal finalidade, para a obtenção de estimativas de tempo de execução de uma aplicação.

Capítulo 4

Proposta de um sistema de modelagem e predição analítica de desempenho

Neste capítulo é apresentada a proposta de um sistema de modelagem e predição analítica de desempenho, chamado PeMPSys (*Performance Modeling and Prediction System*), para a plataforma de processamento paralelo JoiN. O capítulo descreve as fases de desenvolvimento do sistema de predição proposto, o qual é baseado na técnica de modelagem e predição de desempenho PAMELA Estendida.

4.1 Introdução ao sistema de predição PeMPSys

Sistemas de predição analítica de desempenho, conforme descrito nos Caps. 1 e 2, permitem estimar o tempo de execução de uma aplicação nos computadores de uma plataforma de processamento paralelo de forma rápida e com baixo custo, até mesmo quando o desempenho ou a quantidade destes varia.

Foi visto que tais sistemas utilizam alguma técnica de modelagem e de predição de desempenho que procura contemplar várias informações que possam influenciar no tempo de execução de uma aplicação. Há sistemas que coletam as informações necessárias para a predição de forma automática e outros de forma manual. Entretanto, em ambos os casos, há fatores que limitam o uso de tais ferramentas de forma prática. Em função disso, este trabalho propõe o desenvolvimento de um sistema de modelagem e predição analítica de desempenho, baseada na técnica PAMELA Estendida, para a plataforma de processamento paralelo JoiN.

O sistema de predição proposto (PeMPSys) não exige o uso de marcas especiais no código-fonte da aplicação, ou o uso de uma linguagem de programação não padrão para que a predição seja possível.

Para a coleta das informações, o sistema de predição faz uso de ferramentas específicas. Isso

permite que o sistema contemple efeitos de atraso causados pela contenção no uso de recursos, tanto interna quanto externa, e que operações aritméticas realizadas sobre diferentes tipos de dados sejam tratadas de forma diferenciada. As informações coletadas são representadas na forma de modelos que são construídos e analisados para que forneçam uma estimativa do tempo de execução de uma aplicação.

As próximas seções descrevem de forma detalhada o sistema de predição proposto PeMPSys, a forma com que ele constrói modelos e também como estes modelos são analisados para produzir uma estimativa de tempo na forma de um intervalo de valores.

4.2 Estrutura do sistema de predição

O sistema de predição PeMPSys é responsável pela criação de 3 modelos: modelo de máquinas, modelo da aplicação e modelo analítico de desempenho. A construção de tais modelos é feita por ferramentas desenvolvidas especialmente para este fim, de maneira que sejam independentes entre si e funcionem em etapas distintas, conforme a estrutura apresentada na Fig. 4.1.

As 3 ferramentas, ARC (*Analytical Representation Compiler*), CRES (*Computational Resources Evaluation Service*) e APET (*Analytical Performance Estimation Tool*), apresentam funcionalidades bem definidas: a primeira (quadro A da figura) é responsável pela geração de um modelo da aplicação, a segunda (quadro B) pela geração de um modelo de máquinas da plataforma paralela e a terceira (quadro C) pela geração de um modelo analítico de desempenho e de tempos de predição. A geração dos modelos envolve as seguintes etapas (marcadas por números na figura):

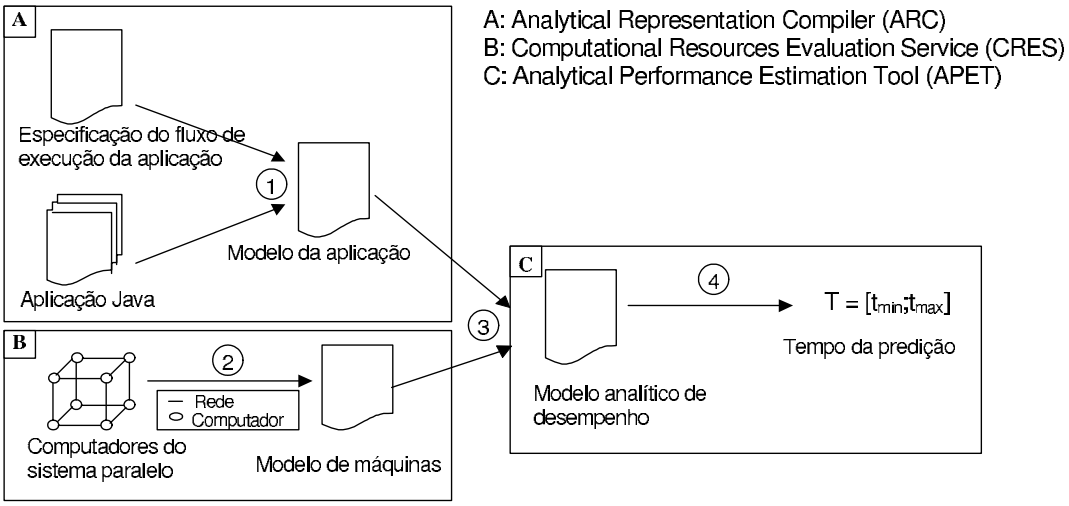

Fig. 4.1: Etapas do processo de geração de modelos para a predução de desempenho

1. análise do código-fonte da aplicação em Java e da especificação do fluxo de execução em PASL por meio da ferramenta ARC, para a geração de um modelo da aplicação;

2. avaliação dos recursos computacionais dos computadores da plataforma paralela com o uso de algoritmos de *benchmark* (algoritmos de testes) executados pela ferramenta CRES em cada computador, para a geração de um modelo de máquinas;

3. combinação simbólica do modelo de máquinas com o modelo da aplicação usando a ferramenta APET, para a geração de um modelo analítico de desempenho, composto de um conjunto de expressões matemáticas;

4. avaliação das expressões matemáticas do modelo analítico de desempenho usando a ferramenta APET para a obtenção de estimativas de tempo.

As próximas seções apresentam em detalhes o desenvolvimento de cada uma das ferramentas citadas. Será vista a forma com que cada uma delas cria um modelo coletando informações disponíveis na plataforma de processamento paralelo JoiN e nas aplicações executadas nela.

4.3 Geração do modelo de máquinas

A ferramenta CRES (*Computational Resource Evaluation Service*) é a parte do sistema PeMPSys responsável pela geração automática do modelo de máquinas da plataforma de processamento paralelo JoiN (quadro B da Fig. 4.1). A geração do modelo é aquela descrita na Seção 3.2.2, baseada na execução de *benchmarks* para avaliar a capacidade de computação e de comunicação do computador coordenador e dos computadores trabalhadores. A avaliação é feita sobre operações aritméticas básicas em diferentes tipos de dados e sobre troca de mensagens, definindo intervalos de valores para os parâmetros op_data_p e $send_{p \rightarrow q}$ para cada computador analisado.

Baseado no modelo da plataforma de processamento paralelo JoiN, no qual os computadores trabalhadores só podem comunicar-se com seus respectivos computadores coordenadores, e estes com outros computadores coordenadores e com o servidor, é apresentada na Fig. 4.2 a estrutura de funcionamento de CRES em um computador trabalhador. Os passos envolvidos são:

1. execução de algoritmos de *benchmark* para operações de computação e de comunicação;

2. armazenamento dos resultados de cada operação em um arquivo do sistema de arquivos local;

3. leitura do arquivo com os valores medidos para determinação de um intervalo de valores para cada um dos *benchmarks* realizados;

4. envio dos intervalos ao computador coordenador p;

5. atualização dos intervalos que estão armazenados no modelo de máquinas.

Na Fig. 4.3 é ilustrado o funcionamento da ferramenta CRES quando os benchmarks são executados em um computador coordenador. Para este caso, não são realizados *benchmarks* de comunicação

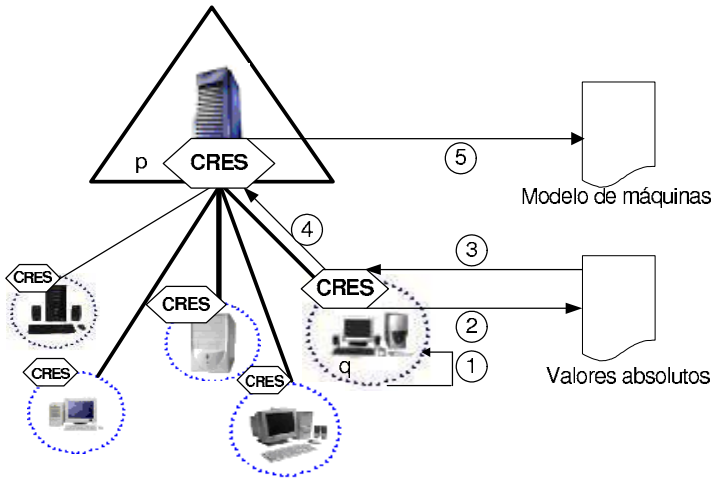

Fig. 4.2: Estrutura funcional de CRES em um computador trabalhador

pois, para executar uma aplicação, o computador coordenador não necessita se comunicar com outros computadores além dos trabalhadores, que já fizeram os testes de comunicação. Desta forma, dado um computador coordenador p e um trabalhador q, é assumido que o parâmetro $send_{p \rightarrow q}$ que define o tempo de comunicação entre p e q será o mesmo que $send_{q \rightarrow p}$. A ordem de execução dos *benchmarks* em um computador coordenador (indicados por números na Fig. 4.3) é a seguinte:

1. execução de algoritmos de *benchmark* para operações de computação;
2. armazenamento dos resultados de cada operação em um arquivo do sistema de arquivos local;
3. leitura do arquivo com os valores medidos para determinação de um intervalo de valores para cada um dos testes realizados;
4. atualização do modelo de máquinas com o intervalo determinado.

A execução dos *benchmarks* para a geração do modelo de máquinas é realizada sempre que o computador, coordenador ou trabalhador, fica ocioso. Os novos dados, para cada teste, são armazenados seguindo uma política de filas FIFO (*First In First Out*) com um tamanho limitado. Isto é, quando a quantidade de medições atinge um limite, os valores mais antigos vão sendo substituídos pelos mais novos.

Para uma determinada máquina $comp_p$ de um grupo, ($0 \leq p \leq P$, onde P é o número de computadores trabalhadores e $comp_0$ representa o computador coordenador), com K medições para cada tipo de operação realizada pelo *benchmark*, tem-se um conjunto de dados de tamanho K na forma $t\_op\_data_p = [cp_1; cp_2; ...; cp_K]$ para operações de computação e um conjunto $t\_send_{p \rightarrow q} = [cm_1; cm_2; ...; cm_K]$ para operações de comunicação.

Estes valores são armazenados no sistema de arquivos de cada computador analisado, como valores absolutos de todas as medições. A partir destes dados, é possível construir intervalos de valores

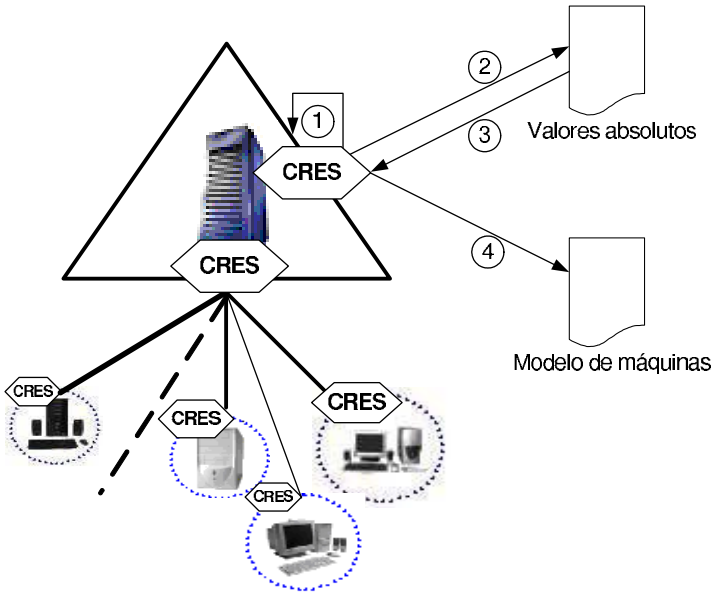

Fig. 4.3: Estrutura funcional do CRES em um computador coordenador

do tipo $inter\_op\_data_p = [cp_{min}; cp_{max}]$ e $inter\_send_{p \to q} = [cm_{min}; cm_{max}]$, para operações de computação e de comunicação, respectivamente. Os intervalos construídos farão parte do modelo de máquinas, o qual é único em cada grupo da plataforma de processamento paralelo. O modelo de máquinas armazena informações de desempenho de todos os computadores que formam um grupo na plataforma JoiN.

Os algoritmos de *benchmark* utilizados na realização dos *benchmarks* são específicos para a avaliação de plataformas de processamento paralelo na execução de aplicações científicas de grande porte [SBO01] e fazem parte do pacote Java Grande Benchmark Suite, disponível em [jav05]. Seu tempo de execução depende de cada computador, mas normalmente se situa abaixo de 3 minutos.

4.4 Geração do modelo da aplicação

A geração do modelo da aplicação é feita pela ferramenta ARC (*Analytical Representation Compiler*) e corresponde ao quadro A da Fig. 4.1. ARC foi desenvolvida especialmente para analisar e extrair informações de atraso presentes em uma aplicação paralela.

A estrutura da ferramenta ARC é representada na Fig. 4.4. Ela é formada por três blocos: PASC (*Parallel Application and Specification Compiler*), SCAn (*Source Code Analyser*), e AMGen (*Application Model Generator*). Este último possui informações da plataforma de processamento paralelo JoiN em relação a sua estrutura paralela e ao seu escalonador de tarefas e tem por objetivos combinar os resultados fornecidos pelos outros dois blocos. PASC e SCAn foram projetados e desenvolvidos na forma de 2 compiladores, seguindo a estrutura da Fig. 2.1, com o auxílio das ferramentas JLex e

CUP para a criação de analisadores léxicos e sintáticos, respectivamente.

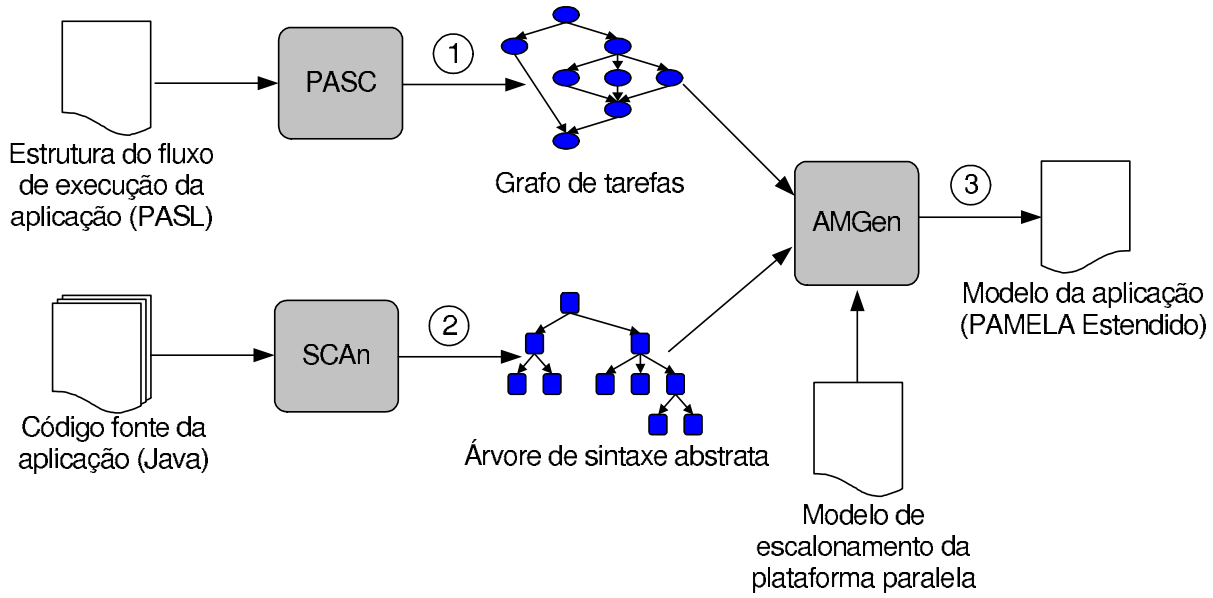

Fig. 4.4: Estrutura da ferramenta ARC

O resultado de cada análise descrita (léxica e sintática) é uma representação intermediária na forma de uma estrutura de dados manipulável, de fácil acesso, eficiente e na qual toda a hierarquia das operações realizadas pelo código analisado é mantida. As informações presentes nas estruturas de dados (representações intermediárias), são acessadas pela ferramenta AMGen, para a geração de uma representação que equivale ao modelo da aplicação. A notação utilizada para representar as instruções em tal modelo é a da PAMELA Estendida, isto é, baseada em intervalos.

Desta forma, para que ARC construa um modelo da aplicação, os três blocos atuam na seguinte ordem:

1. PASC (*Parallel Application and Specification Compiler*): analisa a estrutura do fluxo de execução de uma aplicação escrito na linguagem PASL (linguagem de especificação de fluxo de JoiN) para produzir um grafo de tarefas;

2. SCAn (*Source Code Analyser*): analisa o código-fonte de uma aplicação escrita na linguagem Java para produzir uma árvore de sintaxe abstrata;

3. AMGen (*Application Model Generator*): gera o modelo da aplicação a partir da análise dos nós do grafo e da árvore, utilizando informações da plataforma JoiN, para representar simbolicamente a forma com que as tarefas de uma aplicação são distribuídas e processadas entre os computadores da plataforma paralela.

Estas etapas da ferramenta ARC serão descritas de forma detalhada a seguir.

4.4.1 PASC: análise do fluxo de execução da aplicação

A primeira etapa de desenvolvimento da ferramenta ARC foi a construção do compilador PASC, que é responsável por analisar a estrutura do fluxo de execução de uma aplicação, escrita na linguagem PASL, e criar uma representação intermediária equivalente.

A construção do compilador, dividida em 3 fases (léxica, sintática e geração de uma representação intermediária), é descrita a seguir.

Na fase de análise léxica foi definida a estrutura léxica da linguagem PASL. Tal estrutura possibilita identificar um conjunto de caracteres que formam um token válido para a linguagem (palavras reservadas, identificadores, números, operadores etc.). Para a fase de análise sintática foi definida a estrutura sintática da linguagem PASL, de forma a identificar combinações válidas dos tokens da linguagem que foram reconhecidos inicialmente pelo analisador léxico. Para cada token ou conjunto deles provenientes do analisador léxico, o analisador sintático permite identificar se o token representa um lote de tarefas, a localização de um arquivo com código executável, a cardinalidade de um lote, o fluxo de dados entre as tarefas de cada lote e a origem e o destino final dos dados. Essas informações são utilizadas para a fase de criação de uma representação intermediária na forma de uma estrutura de dados do tipo grafo, chamada de grafo de tarefas.

Um grafo de tarefas consiste num conjunto de nós (ou vértices) e num conjunto de arcos (ou arestas). Cada arco num grafo é especificado por um par de nós ordenados, indicando uma origem e um destino. Tanto nós como arcos de um grafo podem armazenar uma série de informações, dependendo de sua finalidade.

A criação de cada nó do grafo é feita durante a fase de análise sintática. Para tokens que representam um novo lote, um novo nó do grafo é criado. As ligações no grafo são feitas por tokens que representam o fluxo de dados entre os lotes. Neste caso, um conjunto de tokens é lido para identificar a origem e o destino dos dados e fazer as conexões necessárias entre os nós do grafo. Outras informações como cardinalidade e código executado pelas tarefas de cada lote são armazenadas como atributos de um nó do grafo.

A Fig. 4.6 traz um exemplo de uma representação gráfica de um grafo de tarefas com seus respectivos atributos, no qual é ilustrado o fluxo de dados entre os lotes de tarefas, representado pelo código PASL da Fig. 4.5.

O grafo representado na figura apresenta as informações necessárias que serão utilizadas pela ferramenta AMGen para produzir uma parte do modelo da aplicação referente às características do paralelismo da aplicação. Supondo que a cardinalidade dos lotes B_1, B_2, B_3, B_4, B_5 e B_6 seja $c_1 = 1$, $c_2 = 4$, $c_3 = 4$, $c_4 = 1$, $c_5 = 1$ e $c_6 = 1$, respectivamente, tem-se o grafo expandido da Fig. 4.7, o qual ilustra o fluxo de dados entre as tarefas que compõem cada um dos lotes da aplicação. No grafo, $tk_{i,j}$ representa uma tarefa j de um lote i.

```
//seção de declarações

path = "join.applications.sample.apptest";
B1 = "DistribuiDados.class";
B2 = "Processa1.class";
B3 = "Processa2.class";
B4 = "Processa3.class";
B5 = "Processa4.class";
B6 = "ColetaResultados.class";
//seção de ligações
T1 = B1(c1) « dados.dat;
T2 = B2(c2) « T1;
T3 = B3(c3) « T1;
T4 = B4(c4) « T3;
T5 = B5(c5) « T3;
T6 = B6(c6) « T2,T4,T5 »resultado.dat;
```

Fig. 4.5: Código PASL

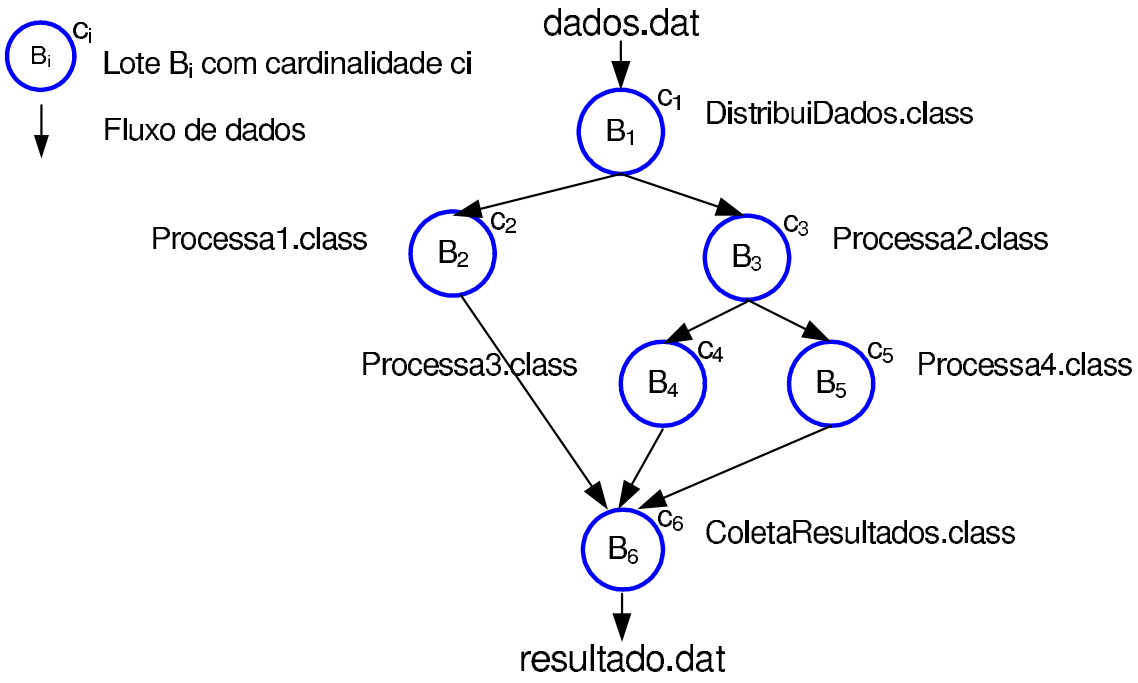

Fig. 4.6: Grafo de tarefas

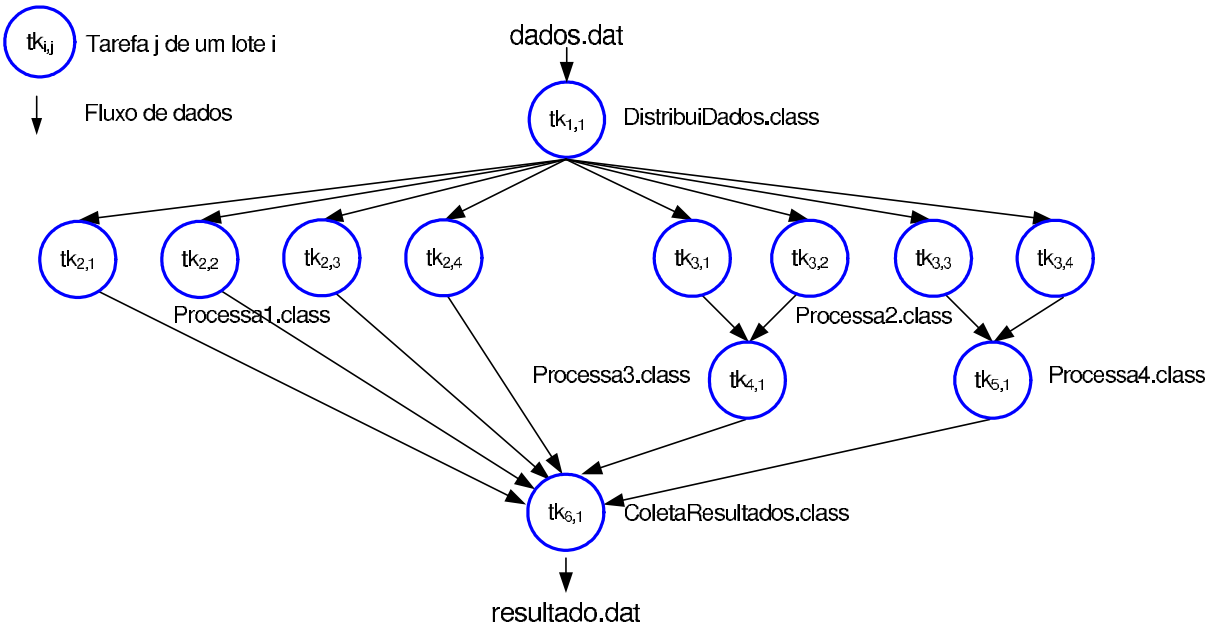

Fig. 4.7: Grafo de tarefas expandido

4.4.2 SCAn: análise do código-fonte da aplicação

Além das informações da estrutura do fluxo de execução da aplicação presentes no grafo de tarefas, a geração do modelo da aplicação requer também informações de tempo de execução de cada uma das tarefas que compõem os lotes de uma aplicação. Desta forma, para cada nó do grafo, é feita uma análise do código-fonte para a geração de uma representação intermediária na forma de uma árvore de sintaxe abstrata (ASA) (etapa 2 da Fig. 4.4). A geração da representação intermediária é feita pelo compilador SCAn (*Source Code Analyser*), e sua construção seguiu as mesmas fases de desenvolvimento do compilador PASC descrito na seção 4.4.1.

Na primeira fase, análise léxica, foram definidas as expressões regulares que formam os padrões de tokens da linguagem Java. Por ser uma linguagem de programação mais complexa que a PASL, a definição das expressões regulares que formam a estrutura léxica da linguagem é bem mais extensa e trabalhosa. A definição de tal estrutura foi necessária para que o código de uma aplicação seja devidamente analisado e reconhecido, indicando se um token é uma palavra reservada da linguagem, algum operador, um comentário, um identificador, um número, entre outros.

Todas as expressões regulares foram construídas sem se basear em outras já existentes. Isso foi feito para garantir total domínio sobre o projeto do compilador para que, caso existam necessidades de eventuais modificações ou adaptações do mesmo para outras atividades, o tempo gasto não seja elevado. Além disso, houve o objetivo de desenvolver um analisador léxico utilizando uma notação específica, sem efetuar o reaproveitamento de partes de outros, para obter domínio no desenvolvimento de tais analisadores. Após definir todas as expressões regulares, foi gerado um analisador léxico a partir da estrutura léxica definida com o uso da ferramenta JLex.

A próxima etapa foi a construção de um analisador sintático. Para isso, foi preciso definir a estrutura gramatical da linguagem Java para a geração do analisador sintático. Assim como na definição da estrutura léxica da linguagem Java, definir a sua estrutura gramatical foi uma tarefa complexa já que se trata de uma linguagem de programação de alto-nível, orientada a objetos e com uma grande quantidade de recursos e instruções. Para a definição da estrutura gramatical da linguagem, baseou-se em uma estrutura gramatical fornecida pela empresa desenvolvedora da linguagem Java, a SUN, disponível em [GJS96]. Durante esta fase de desenvolvimento, verificou-se que a gramática fornecida por tal empresa apresenta falhas e não é capaz de analisar todo tipo de código válido para um compilador Java. Desta forma, foi necessário reescrever e modificar parte dela para que a análise de qualquer código que siga as especificações da linguagem seja possível.

Foi verificada a existência de ferramentas que realizassem a análise sintática de código Java; porém, não foi encontrada nenhuma de código aberto e que fornecesse uma estrutura adaptável e manipulável de acordo com as características do trabalho proposto. Além disso, devido à necessidade de total controle das partes de algum código analisado, foi necessária a construção de tal analisador.

Concluída a definição da estrutura gramatical da linguagem, foi gerado automaticamente um analisador sintático com o uso da ferramenta CUP, que utiliza a gramática definida como informação de entrada. O analisador sintático criado é utilizado em conjunto com o analisador léxico e verifica o que cada token proveniente da análise léxica representa no código-fonte de uma aplicação. Cada token, depois de identificado pelo analisador sintático, é inserido em uma representação intermediária na forma de uma árvore de sintaxe abstrata.

Uma árvore de sintaxe abstrata (ASA) é uma estrutura de dados hierárquica, com um nó raiz, que representa a definição de uma classe (é o ponto de partida para alguma consulta na árvore), nós filhos, que representam estruturas internas de uma classe, e ligações, que conectam dois nós da árvore. Um nó da árvore é uma estrutura simples, com atributos que identificam o valor e o tipo de cada token proveniente da análise léxica.

A Fig. 4.8 ilustra um exemplo de código-fonte em Java, e a Fig. 4.9 sua respectiva representação na forma de uma ASA, que é obtida através do compilador SCAn. Ele permite analisar qualquer aplicação escrita na linguagem Java que se restrinja à gramática padrão [GJSB05] e requer que todo o código-fonte das classes utilizadas esteja disponível.

```
class  ProcessData{
   void  metodo_M(){
     int  a;
     float  res=0;
     for(a=0;a<100;a++){
       res = res+a;
     }
   }
}
```

Fig. 4.8: Código-fonte de uma classe simplificada

Para cada referência a uma classe presente em um nó do grafo de tarefas, é criada uma represen-

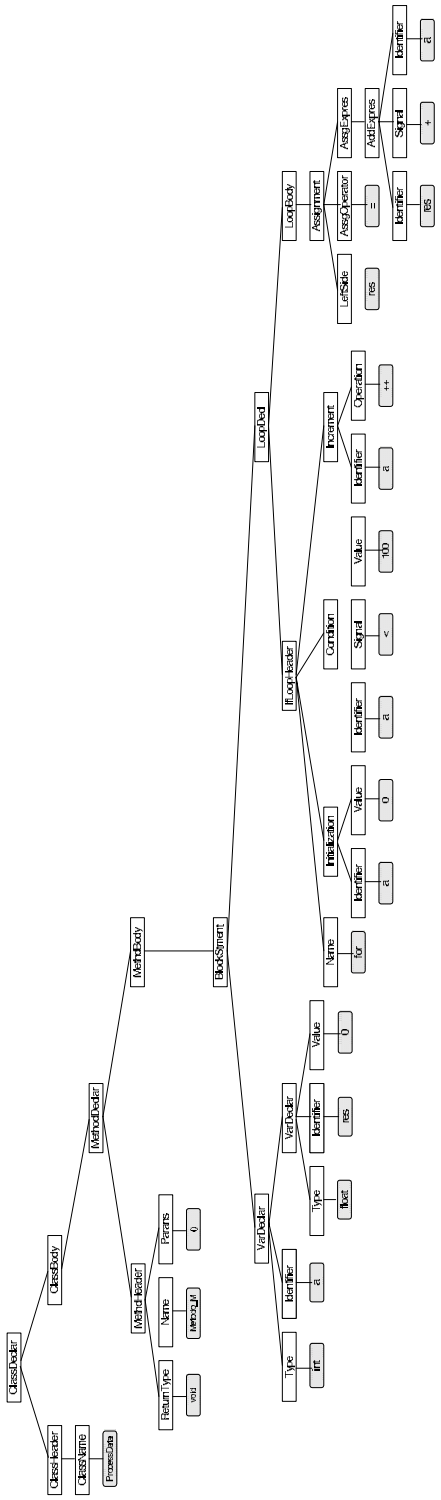

Fig. 4.9: Árvore de sintaxe abstrata correspondente ao código da Fig. 4.8

tação intermediária na forma de uma nova ASA que mapeia as informações do código necessárias para a predição de desempenho.

Este processo permite que o programa de entrada seja totalmente analisado, gerando para cada classe da aplicação uma representação na forma de uma ASA. Isso torna possível uma navegação na hierarquia da estrutura criada e, a partir de qualquer parte identificada do código presente na árvore, pode-se capturar informações relativas ao tempo de execução de blocos de computação da aplicação.

4.4.3 AMGen: geração do modelo da aplicação

Após a geração das representações intermediárias do fluxo de execução da aplicação e do código-fonte executado pelas tarefas, é feita a geração do modelo da aplicação (etapa 3 da Fig. 4.4), por meio da ferramenta AMGen (*Application Model Generator*).

Inicialmente AMGen analisa o grafo de tarefas para identificar o fluxo de dados entre as tarefas que compõem os lotes de uma aplicação. Em seguida, com o uso de informações do sistema de escalonamento de tarefas da plataforma paralela, a ferramenta identifica a quantidade de tarefas e de dados que são enviados para cada computador participante da plataforma.

No sistema paralelo JoiN o escalonamento de tarefas é feito de forma híbrida: inicialmente parte das tarefas é distribuída estaticamente entre os computadores e num segundo momento o restante delas é distribuído dinamicamente (sob demanda). Neste trabalho considera-se que todas as tarefas são distribuídas estaticamente entre os computadores, de acordo com o poder de processamento de cada um (isto equivale a termos $f = 1$ na Eq. 3.6). A predição baseada em um escalonamento misto[1] é deixada para um trabalho futuro. Entretanto, os resultados experimentais (Cap. 5) indicam que não há introdução de erros significativos decorrentes desta simplificação.

A análise estática da quantidade de tarefas que cada computador trabalhador receberá permite calcular os atrasos causados no ambiente da plataforma paralela em relação ao uso da rede no envio dos dados, e no uso da CPU de cada computador, trabalhador ou coordenador, na execução das tarefas. Nesta etapa, cada nó do grafo de tarefas é analisado por AMGen para a geração do modelo da aplicação, considerando dois tipos de análise sobre o grafo de tarefas.

Análise da execução quanto à relação de dependência entre os lotes de uma aplicação

No primeiro tipo de análise, é produzida a parte do modelo da aplicação que define se a execução de um lote é feita de forma paralela ou seqüencial em relação aos demais. Para isso, verifica-se, a partir do nó inicial do grafo, os nós que dependem dos dados produzidos por ele. Para cada um dos nós que recebem dados de outro nó, é feito o mesmo tipo de análise até que todos os nós do grafo tenham sido verificados.

[1] parte das tarefas é alocada de forma estática e a outra de forma dinâmica

Para a geração de um modelo, todo nó ou conjunto de nós produz um sub-modelo $\mathbb{L}_i$ que indica o custo envolvido no processamento dos nós verificados a partir de um lote B_i. O modelo da aplicação, indicado por L, representa o atraso de tempo para o processamento de todos os lotes de tarefas, e é representado por $L = \mathbb{L}_1$, que equivale ao custo do sub-modelo produzido pelo lote de índice 1, que é o lote inicial da aplicação.

Neste tipo de análise, são considerados 3 tipos de nó, independentemente de suas cardinalidades (Fig. 4.10):

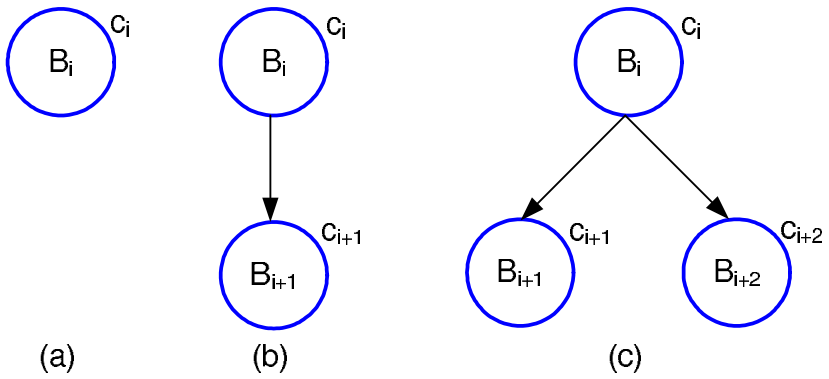

Fig. 4.10: Fluxo de dados entre lotes de tarefas

(a) lote B_i que não produz dados para outro lote;

(b) lote B_i que produz dados para um lote B_{i+1};

(c) lote B_i que produz dados para dois ou mais lotes;

No primeiro tipo de nó (Fig. 4.10 (a)), é produzido um sub-modelo $\mathbb{L}_i$ simples, equivalente ao processamento do próprio lote, representado por $\mathbb{L}_i = \mathbb{B}_i$, onde $\mathbb{L}_i$ representa o sub-modelo do lote B_i e $\mathbb{B}_i$ o seu custo de processamento.

No segundo tipo de nó (Fig. 4.10 (b)), há o processamento seqüencial do lote B_i com o lote B_{i+1}. O sub-modelo produzido é equivalente a $\mathbb{L}_i = \mathbb{B}_i; \mathbb{L}_{i+1}$.

O terceiro tipo de nó (Fig. 4.10 (c)) expressa a execução dos lotes B_{i+1} e B_{i+2} de forma paralela, após o processamento do lote B_i, pois os lotes são independentes entre si e dependem dos dados produzidos pelo processamento deste último lote. O modelo $\mathbb{L}_i$ produzido para representar esse tipo de processamento é equivalente a $\mathbb{L}_i = \mathbb{B}_i; (\mathbb{L}_{i+1} || \mathbb{L}_{i+2})$. Quando um lote produz dados para dois ou mais lotes independentes entre si, como é o caso da Fig. 4.10 (c), representa-se a execução dos sub-modelos equivalentes aos lotes de forma paralela.

Exemplo: Para ilustrar como é feita a análise da execução de cada lote em relação aos demais, é utilizado o grafo de tarefas da Fig. 4.6. O grafo apresenta a estrutura de uma aplicação paralela, com

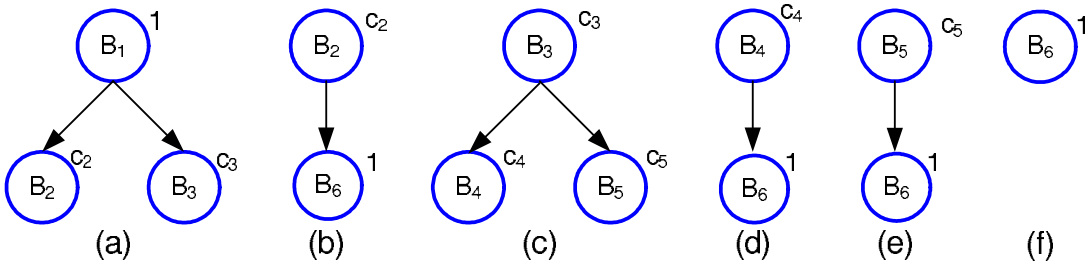

Fig. 4.11: Sub-grafos considerados para geração do modelo da aplicação

um lote inicial e outro final. Na análise de tal grafo, AMGen considera os 6 sub-grafos representados na Fig. 4.11.

A análise dos sub-grafos de (a) a (f) da Fig. 4.11 produz, respectivamente, as linhas 2 a 7 do modelo da aplicação parcial da Fig. 4.12, no qual a linha 2 equivale ao sub-grafo da Fig. 4.11 (a), a linha 3 equivale ao sub-grafo da Fig. 4.11 (b), a linha 4 equivale ao sub-grafo da Fig. 4.11 (c), a linha 5 equivale ao sub-grafo da Fig. 4.11 (d), a linha 6 equivale ao sub-grafo da Fig. 4.11 (e) e a linha 7 equivale ao sub-grafo da Fig. 4.11 (f). Na linha 1 é representado o modelo da aplicação L em função do sub-modelo $\mathbb{R}_1$.

1 $L = \mathbb{L}_1$
2 $\mathbb{L}_1 = \mathbb{B}_1; (\mathbb{L}_2 || \mathbb{L}_3)$
3 $\mathbb{L}_2 = \mathbb{B}_2; \mathbb{L}_6$
4 $\mathbb{L}_3 = \mathbb{B}_3; (\mathbb{L}_4 || \mathbb{L}_5)$
5 $\mathbb{L}_4 = \mathbb{B}_4; \mathbb{L}_6$
6 $\mathbb{L}_5 = \mathbb{B}_5; \mathbb{L}_6$
7 $\mathbb{L}_6 = \mathbb{B}_6$

8 $L = \mathbb{B}_1; (\mathbb{L}_2 || \mathbb{L}_3)$
9 $L = \mathbb{B}_1; ((\mathbb{B}_2; \mathbb{L}_6) || \mathbb{L}_3)$
10 $L = \mathbb{B}_1; ((\mathbb{B}_2; \mathbb{L}_6) || (\mathbb{B}_3; (\mathbb{L}_4 || \mathbb{L}_5)))$
11 $L = \mathbb{B}_1; ((\mathbb{B}_2; \mathbb{L}_6) || (\mathbb{B}_3; ((\mathbb{B}_4; \mathbb{L}_6) || \mathbb{L}_5)))$
12 $L = \mathbb{B}_1; ((\mathbb{B}_2; \mathbb{L}_6) || (\mathbb{B}_3; ((\mathbb{B}_4; \mathbb{L}_6) || (\mathbb{B}_5; \mathbb{L}_6))))$
13 $L = \mathbb{B}_1; ((\mathbb{B}_2; \mathbb{B}_6) || (\mathbb{B}_3; ((\mathbb{B}_4; \mathbb{B}_6) || (\mathbb{B}_5; \mathbb{B}_6))))$
14 $L = \mathbb{B}_1; ((\mathbb{B}_2; \mathbb{B}_6) || (\mathbb{B}_3; (\mathbb{B}_4 || \mathbb{B}_5); \mathbb{B}_6))$
15 $L = \mathbb{B}_1; (\mathbb{B}_2 || (\mathbb{B}_3; (\mathbb{B}_4 || \mathbb{B}_5))); \mathbb{B}_6$

Fig. 4.12: Modelo da aplicação parcial produzido a partir da análise de um conjunto de sub-grafos

Após a criação do modelo dos sub-grafos da Fig. 4.11, são feitas algumas simplificações no mesmo. Na primeira etapa de simplificação, o sub-modelo $\mathbb{L}_1$ da linha 1 foi substituído pelo seu equivalente representado na linha 2, produzindo a linha 8. Na linha 9, o sub-modelo $\mathbb{L}_2$ da linha 8 foi substituído pelo seu conteúdo representado na linha 3 e assim sucessivamente. Finalmente na linha 15, está o modelo resultante de outras duas simplificações realizadas nas linhas 13 e 14, em que $\mathbb{B}_6$ que está contido em dois blocos paralelos, passa a fazer parte de um bloco seqüencial logo após o

bloco paralelo sem ele.

Análise das tarefas quanto à cardinalidade de cada uma

No segundo tipo de análise feita por AMGen, é criada a parte do modelo da aplicação que representa o custo para o processamento e o envio de cada uma das tarefas que compõem os lotes. Para isso, classifica-se pares de nós do grafo de tarefas quanto a cardinalidade de cada lote, conforme representação da Fig. 4.13:

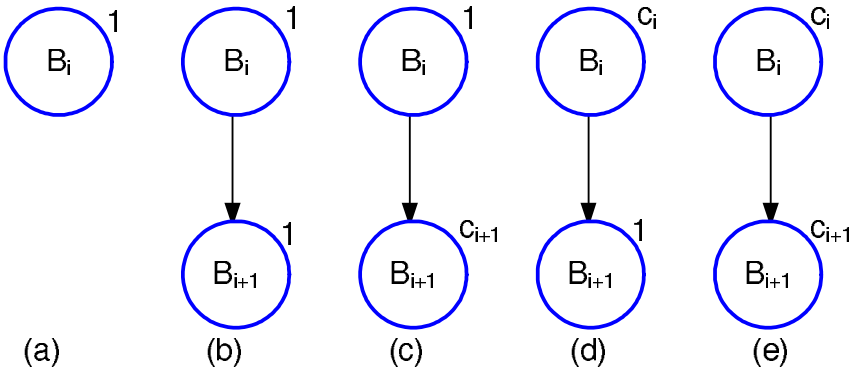

Fig. 4.13: Fluxo de dados entre lotes de tarefas com diferentes cardinalidades

A descrição de cada nó no grafo da Fig. 4.13 é apresentado a seguir:

(a) lote B_i com cardinalidade unitária que não produz dados para outro lote;

(b) lote B_i com cardinalidade unitária e que produz dados para um lote B_{i+1} de mesma cardinalidade;

(c) lote B_i com cardinalidade unitária e que produz dados para um lote B_{i+1} de cardinalidade c_{i+1} maior que um;

(d) lote B_i com cardinalidade c_i maior que um e que produz dados para um lote B_{i+1} de cardinalidade unitária;

(e) lote B_i com cardinalidade c_i maior que um e que produz dados para um lote B_{i+1} de cardinalidade c_{i+1} maior que um.

Um outro tipo de lote, com cardinalidade maior que um que não produz dados para outro lote não é considerado, pois no sistema JoiN o lote final de uma aplicação possui cardinalidade unitária.

Para analisar cada tipo de par de lotes, são modelados o processamento e a transmissão de dados. São utilizadas no modelo as instruções $dataSizeIn(tk_{i,j})$ e $dataSizeOut(tk_{i,j})$, para representar o

tamanho dos dados que são processados e que são produzidos por uma tarefa $tk_{i,j}$, respectivamente. Além disso, é presumido que lotes com cardinalidade igual a um são processados no computador coordenador ($comp_0$), que é o local onde encontram-se o escalonador de tarefas e os dados a serem processados pelo lote. Isto implica que, neste caso, não há envio de dados pela rede. Cada modelo produzido pela análise dos cinco tipos de lotes é descrito a seguir.

Para a análise de um lote B_i de cardinalidade unitária e que não produz dados para outro lote (Fig. 4.13 (a)), AMGen produz um custo $\mathbb{B}_i$ no qual há a utilização do recurso representado pelo computador $comp_0$ para processar a única tarefa do lote, representada no modelo por $tk_{i,1}$. O custo do processamento da única tarefa do lote B_i no computador coordenador é equivalente a:

$$\mathbb{B}_i := use(comp_0, tk_{i,1})$$

Para o tipo de lote B_i com cardinalidade unitária e que produz dados para um lote B_{i+1} de mesma cardinalidade (Fig. 4.13 (b)), é produzido um modelo em que a tarefa de cada um dos lotes é processada de maneira seqüencial no computador $comp_0$. O custo resultante é equivalente a:

$$\mathbb{B}_i := use(comp_0, tk_{i,1}); use(comp_0, tk_{i+1,1})$$

Nele, conforme enfatizado, há ausência de comunicação, pois os dois lotes serão processados no mesmo computador.

Para um lote B_i com cardinalidade igual a um e que produz dados a outro lote B_{i+1} com cardinalidade maior do que um (Fig. 4.13 (c)), é produzido um modelo que representa o processamento da tarefa do lote B_i e o envio dos dados para as tarefas do lote B_{i+1}, no qual há a utilização dos recursos $comp_0$ e net_0 para processar o lote de tarefas e para enviar os dados das tarefas para os computadores trabalhadores, respectivamente. A Fig. 4.14 ilustra de forma simplificada o cálculo do custo produzido que representa o processamento e o envio das tarefas de um lote para um conjunto de P computadores trabalhadores. A linha 1 declara o custo do lote B_i, que equivale ao processamento $use(comp_0, tk_{i,1})$ da tarefa $tk_{i,1}$ no computador $comp_0$ (linha 2), e ao envio dos dados das tarefas do lote B_{i+1} (linhas 3 a 7) para os computadores trabalhadores. Na linha 3, para cada computador p, é alocada (linha 4) uma quantidade $b-a$ de tarefas, onde $a = id\_Inicial_p$ e $b = id\_Inicial_p - 1 + S_p$, com $id\_Inicial_p = 1 + \sum_{i=1}^{p} S_i$. Para cada uma das S_p tarefas alocadas a um computador p, é utilizada a rede net_0 (linha 5) do computador $comp_0$ para enviar os dados de entrada ($dataSizeIn$) para a tarefa $tk_{i+1,j}$.

```
1   B_i :={
2       use(comp_0, tk_i,1);
3       par(p=1,P){
4         par(j = a,b){
5           use(net_0, send_0→p(dataSizeIn(tk_i+1,j)));
6         }
7       }
8   }
```

Fig. 4.14: Custo do processamento de um lote de tarefas em um computador coordenador e da distribuição de dados para outro lote

Para um lote B_i com cardinalidade maior do que um e que produz dados para um lote B_{i+1} de cardinalidade unitária (Fig. 4.13 (d)), é produzido um modelo que representa apenas o processamento e o envio dos dados das tarefas do lote B_i para o computador coordenador. O lote B_i é formado por uma quantidade de c_i tarefas que já foram distribuídas e estão prontas para processamento nos computadores trabalhadores da plataforma paralela. A Fig. 4.15 ilustra de forma simplificada o cálculo do custo do processamento e envio dos resultados produzidos pelas tarefas do lote indicado por B_i. A linha 1 declara o custo $\mathbb{B}_i$, que equivale ao processamento das tarefas do lote B_i entre os computadores trabalhadores. Na linha 2, para cada computador p da plataforma, é feito o processamento (linha 4) e o envio do resultado (linha 5) de cada uma das $b - a$ tarefas para o computador $comp_0$. Neste modelo, a e b são calculados da mesma forma que foram para a Fig. 4.14.

```
1   B_i := {
2         par(p=1,P){
3           par(j = a,b){
4             use(comp_p, tk_i,j);
5             use(net_p, send_p→0(dataSizeOut(tk_i,j)));
6           }
7         }
8   }
```

Fig. 4.15: Custo do processamento e do envio dos resultados de tarefas a partir de computadores trabalhadores

No último caso, onde há um lote B_i com cardinalidade maior que um e que produz dados para um lote B_{i+1} de cardinalidade também superior a um (Fig. 4.13 (e)), é produzido um modelo seqüencial que representa o processamento das tarefas que compõem o lote B_i e o envio dos dados resultantes para os computadores que farão o processamento das tarefas do lote seguinte. O lote B_i é representado no modelo por um conjunto de instruções que descrevem o processamento de cada tarefa do lote em um computador trabalhador. Para o lote B_{i+1} é representado apenas o envio das tarefas, já que no momento da análise das tarefas de tal lote será gerado o modelo correspondente ao seu processamento. O custo deste último caso é ilustrado de forma simplificada na Fig. 4.16. Na linha 1, o custo $\mathbb{B}_i$ equivale ao processamento das tarefas do lote entre os computadores trabalhadores (linhas 2 a 7) e ao envio dos dados para o lote B_{i+1} (linhas 8 a 12).

Quando um lote B_i produz dados para dois ou mais lotes, a ferramenta AMGen produz um modelo para o processamento das tarefas do lote e outro para o envio dos dados para cada um dos lotes dependentes dele.

Após a geração do modelo equivalente ao processamento e ao envio das tarefas que compõem uma aplicação, é necessário gerar um outro modelo que corresponde à estimativa de custo computacional do código executado pelas tarefas.

```
1    B_i = {
2          par(p=1,P){
3            par(j=a,b){
4              use(comp_p, tk_{i,id(p)});
5              use(net_p,send_{p→0}(dataSizeOut(tk_{i,j})));
6            }
7          }
8          par(p=1,P){
9            par(j=a,b){
10             use(net_0,send_{0→p}(dataSizeIn(tk_{i+1,j})));
11           }
12         }
13   }
```

Fig. 4.16: Modelo da aplicação para representar o processamento e a distribuição de dados de lotes com cardinalidade maior que um

Obtenção da estimativa de custo computacional

A estimativa de custo computacional para uma tarefa j de um lote i é representada pela quantidade de operações aritméticas de cada tipo executadas pelo código.

Para obter informações de custo do código executado por uma tarefa, é utilizado o compilador SCAn, que cria uma representação intermediária na forma de uma árvore de sintaxe abstrata a partir do código-fonte de uma aplicação Java. Essas informações de custo são armazenadas como atributos dos nós do grafo de tarefas gerado.

A partir da árvore de sintaxe abstrata, a ferramenta SCAn permite localizar um nó base que representa a declaração de algum método específico do código-fonte. O código executado por um método, como declaração de variáveis, chamada de funções, estruturas de repetição e de condição, operações aritméticas e lógicas são representadas na estrutura intermediária como nós-filho do nó base.

Para que a análise seja precisa, é necessário verificar o nó base e todos os seus nós-filho para identificar se há alguma informação de atraso. Para cada nó analisado, a ferramenta SCAn verifica se ele representa a execução de algum tipo de operação no código para produzir uma representação de atraso equivalente no modelo da aplicação.

As operações aritméticas e os tipos de dados considerados pela ferramenta estão listados na Tab. 4.1 e 4.2, respectivamente. O tempo gasto pelas operações aritméticas sobre diferentes tipos de dados é representado no modelo da aplicação pelo parâmetro op_data_p, discutido na seção 3.2.1. Com relação às operações de atribuição, todas são consideradas como sendo de atribuição em ponto flutuante de simples precisão. Isso foi feito devido a falta de algoritmos de *benchmark* para avaliar o desempenho dos computadores participantes do sistema paralelo para realizar tal operação sob uma grande variedade de dados.

A Fig. 4.17 ilustra de forma simplificada como é feita a representação no modelo da aplicação considerando um computador p que executará o trecho de código ilustrado na figura.

Tab. 4.1: Operações aritméticas

Símbolo da operação	Descrição
+	adição
-	subtração
*	multiplicação
/	divisão
%	resto da divisão
+=	adição e atribuição
-=	subtração e atribuição
*=	multiplicação e atribuição
/=	divisão e atribuição
++	incremento
–	decremento
==	igual
!=	diferente
<	menor
>	maior
<=	menor ou igual
>=	maior ou igual

Tab. 4.2: Tipos de dados

Tipo de dado	Descrição
byte	inteiro de 8 bits
short	inteiro curto de 16 bits
int	inteiro de 32 bits
long	inteiro de 64 bits
float	ponto flutuante com precisão simples de 32 bits
double	ponto flutuante com precisão dupla de 64 bits
char	caracter de 16 bits
boolean	assume valor lógico

```
1   ...
2   double  w=10000;
3   int  x=10;
4   int  y=20;
5   int  z=x+y;
6   ...
```

(a) Código-fonte

```
|1  ...
|2  assign_double_p;
|3  assign_int_p;
|4  assign_int_p;
|5  access_p;
|6  access_p;
|7  sum_int_p;
|8  assign_int_p;
|9  ...
```

(b) Modelo da aplicação

Fig. 4.17: Modelo da aplicação para um código com operações aritméticas

O trecho de código da Fig. 4.17 (a), representado internamente em uma árvore, é analisado para produzir o modelo da aplicação equivalente ao da Fig. 4.17 (b). Na representação, o código da linha 2 produz a linha 2 do modelo, referente a uma operação de atribuição de um número em ponto flutuante de precisão dupla. O mesmo ocorre nas linhas 3 e 4 do código analisado e do modelo gerado. A linha 5 do código produz as linhas 5 a 8 do modelo, que equivalem a duas operações de acesso ao valor de uma variável, a uma operação de adição e a uma operação de atribuição respectivamente.

Com relação às estruturas de repetição consideradas pela ferramenta, elas são listadas na Tab. 4.3. Os três tipos de estruturas podem ser analisados pela ferramenta AMGen, que detecta informações de atraso presentes nos blocos que são executados e também as operações realizadas pelas variáveis de controle presentes em tais estruturas.

Tab. 4.3: Estruturas de repetição

Estrutura de repetição	**Descrição**
for(inicializacao;condicao;incremento) <comando>	repete um conjunto de instruções controlando o número de repetições com um contador
while(condição)<comando>	repete um conjunto de instruções enquanto uma certa condição for satisfeita
do<comando>while(condição)	repete um conjunto de instruções enquanto uma certa condição for satisfeita, sendo executado pelo menos uma vez

O laço do tipo $for(inicializacao; condicao; incremento)$ é a estrutura de repetição exemplificada a seguir. A declaração do laço é convertida no modelo da aplicação com o uso da instrução seq, e os parâmetros do laço são utilizados para definir o valor dos parâmetros da instrução. Além disso, todas as operações de computação executadas pelos parâmetros da declaração do laço de repetição são convertidos no modelo da aplicação como operações de atraso na forma op_data_p. A Fig. 4.18 ilustra um trecho de código composto de uma estrutura de repetição e o modelo da aplicação produzido a

partir dele.

```
1   ...
2   for(int  i=0;i<50000;i++){
3      ...
4      bloco1;
5      ...
6   }
7   ...
8
```

(a) Código-fonte

```
|1   ...
|2   assign_int_p;
|3   access_p;
|4   compare_p;
|5   seq(i=1,50000){
|6      ...
|7      custo_bloco1;
|8      access_p;
|9      sum_int_p;
|10     assign_int_p;
|11     access_p;
|12     compare_p;
|13     ...
|14  }
|15  ...
```

(b) Modelo da aplicação

Fig. 4.18: Modelo da aplicação para um código que contém uma estrutura de repetição

Na Fig. 4.18 (a), a iniciação $i = 0$ da estrutura de repetição, na linha 2 do código, produz a linha 2 do modelo da aplicação da Fig. 4.18 (b). Os parâmetros de iniciação $i = 0$, de condição $i < 50000$ e de incremento $i++$, presentes na linha 2 do código são utilizados para definir os valores 1 e 50000 do parâmetro da instrução seqüencial seq, na linha 5 do modelo. O custo da operação de incremento é representado nas linhas 8 a 10 do modelo. A condição é representada nas linhas 11 e 12 do modelo, e também nas linhas 3 e 4. A linha 4 do código produz uma representação equivalente ao da linha 7 do modelo.

Com relação às estruturas de condição consideradas, elas são listadas na Tab. 4.4. Os três tipos de estruturas podem ser analisados pela ferramenta AMGen, que detecta informações de atraso presentes nos blocos que são selecionados e também o custo da comparação realizada para determinar qual bloco será executado.

Uma estrutura condicional do tipo if(condicao){bloco1}else{bloco2}, por exemplo, é representada no modelo pela instrução condicional if{custo_bloco1}else{custo_bloco2}. A condição da estrutura if presente no código da aplicação é convertida no modelo para instruções do tipo $\mathrm{op\_data_p}$ em função das operações aritméticas realizadas por ela. A condição condicao é a parte da estrutura if que é sempre executada independentemente do valor da condição ser verdadeiro ou falso. A Fig. 4.19 traz uma ilustração da conversão de uma estrutura de condição em uma instrução de atraso equivalente no modelo da aplicação.

O código da figura ilustra a estrutura condicional if utilizada de forma aninhada. A condição $x > 10$ da linha 2 do código produz as linhas 2 a 4 do modelo. As linhas 3 a 5 do código produzem as linhas 5 a 7 do modelo. A condição da linha 6 do código produz as linhas 8 a 10 do modelo. As linhas 7 a 13 do código produzem no modelo as linhas 11 e 17. Os blocos bloco1, bloco2, bloco3 e

Tab. 4.4: Estruturas de condição

Estrutura de condição	**Descrição**
if(condição)<comando>	comando é executado se condição possuir o valor true
<variavel>==<condicao>?<valor1>:<valor2>	variavel recebe valor1 se a condição for verdadeira, caso contrário recebe o valor2
switch(<controle>) case <$valor_i$>: <$comando_i$>; break; default: <$comando_{i+1}$>;	escolhas múltiplas de comandos a serem executados, dependendo do valor da variável controle

```
1   ...
2   if(x>10){
3      bloco1;
4   }
5   else{
6      if(x>20){
7        bloco2;
8      }
9      else{
10       bloco3;
11     }
12     bloco4;
13  }
14  ...
15
```

(a) Código-fonte

```
|1  ...
|2  access_p;
|3  compare_p;
|4  if{
|5     custo_bloco1_p;
|6  }
|7  else{
|8     access_p;
|9     compare_p;
|10    if{
|11      custo_bloco2_p;
|12    }
|13    else{
|14      custo_bloco3_p;
|15    }
|16    custo_bloco4_p;
|17 }
|18 ...
|19
```

(b) Modelo da aplicação

Fig. 4.19: Modelo da aplicação para um código que contém estruturas condicionais

bloco4 presentes nas linhas 3, 7, 10 e 12 do código, são convertidos em suas representações de atraso nas linhas 5, 11, 14 e 16 do modelo, respectivamente.

Para o caso de chamadas de métodos de algum objeto, a ferramenta SCAn cria uma árvore equivalente à classe do objeto instanciado e, em seguida, AMGen analisa-o para obter informações de atrasos de computação realizados pelo método chamado, com seus respectivos parâmetros, caso existam. Considerar o valor dos parâmetros na chamada de algum método é importante pois em alguns casos eles podem definir o número de vezes que uma estrutura de repetição é executada.

Na Fig. 4.20 é ilustrado de forma simplificada um trecho de código que chama um método da mesma classe a partir de uma estrutura de condição e, ao lado, sua forma equivalente no modelo da aplicação.

```
1    ...
2    {
3       double  y=1000;
4       if(condicao){
5         metodo_M(y);
6       }else{
7         bloco1;
8       }
9    }
10   ...
11
12
13   void  metodo_M(double  y){
14      double  i;
15      for(i=0;i<y;i++){
16        bloco2;
17      }
18   }
19   ...
```

(a) Código-fonte

```
|1   ...
|2   {
|3      assign_double_p;
|4      custo_condicao_p;
|5      if{
|6        metodo_M;
|7      }
|8      else{
|9        custo_bloco1_p;
|10     }
|11  }
|12  ...
|13  metodo_M={
|14     assign_double_p;
|15     assign_double_p;
|16     access_p;
|17     compare_p;
|18     seq(i=1,1000){
|19       assign_double_p;
|20       access_p;
|21       sum_double_p;
|22       compare_p;
|23       custo_bloco2_p;
|24     }
|25  }
|26  ...
```

(b) Modelo da aplicação

Fig. 4.20: Modelo da aplicação para um código que contém uma chamada de função

Na linha 5 do código da Fig. 4.20 (a) é ilustrado a chamada do método metodo_M(y), em que é passado como parâmetro o valor da variável y, que é 1000. No código, o método é representado pelas linhas 13 a 18. Tal chamada é representada no modelo da aplicação da Fig. 4.20 (b) pela linha

6, e o custo da função é ilustrado pelas linhas 13 a 25. Os demais trechos do código da figura são representados no modelo através das seguintes linhas: a linha 3 do código produz a linha 3 do modelo, a condição da estrutura if da linha 4 do código produz as linhas 4 e 5 do modelo, e as linhas 4 a 9 do código produzem as linhas 5 a 11 do modelo.

A próxima etapa para a predição de desempenho de uma aplicação é a combinação do modelo de máquinas com o modelo da aplicação, substituindo os parâmetros op_data_p e $send_{p \to q}$ pelos operadores de atraso de tempo $inter\_op\_data_p$ e $inter\_send_{p \to q}$ respectivamente. A partir disso pode ser feita uma análise sobre os efeitos causados pela contenção de recursos com o uso do operador use (seção 3.2.1), permitindo estimar o tempo total de execução das aplicações paralelas sob várias condições de forma estática.

4.5 Geração do modelo e da predição analítica de desempenho

A geração do modelo analítico de desempenho e de tempos de predição na forma de um intervalo de valores corresponde ao quadro C da Fig. 4.1 e é feita pela ferramenta APET (*Analytical Performance Estimation Tool*).

A estrutura da ferramenta APET é apresentada na Fig. 4.21, sendo composta por dois blocos: AMC (*Analytical Model Compiler*) e TPT (*Time Prediction Tool*). O primeiro é responsável pela geração de um modelo analítico de desempenho a partir de um modelo de máquinas e outro da aplicação (número 1 na figura). O segundo avalia matematicamente o modelo analítico de desempenho produzido para obter um intervalo de tempo de predição (número 2 na figura), que representa a aplicação sendo executada em uma determinada quantidade e configuração de computadores. É importante destacar que, além desta ferramenta, não há nenhuma outra documentada na literatura que realize a análise de algum modelo escrito na linguagem PAMELA Estendida.

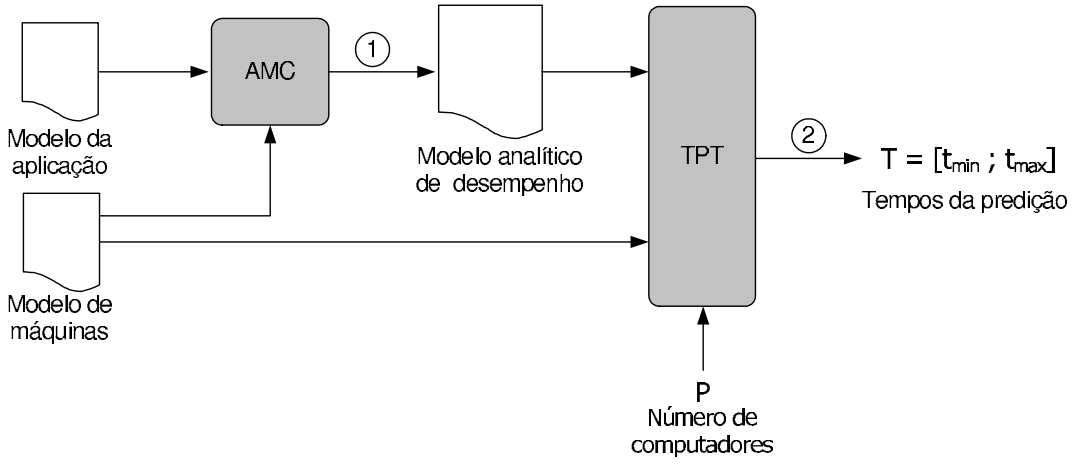

Fig. 4.21: Estrutura da ferramenta APET

A seguir será descrita cada uma das ferramentas que compõem APET.

4.5.1 Geração do modelo analítico de desempenho

A geração do modelo analítico de desempenho (número 1 da Fig. 4.21) é baseada na técnica conhecida por PAMELA Estendida, e realizada por AMC (*Application Model Compiler*), que foi desenvolvido na forma de um compilador que transforma o modelo da aplicação, representado na linguagem PAMELA Estendida, em expressões matemáticas na forma simbólica, isto é, analítica, as quais apresentam diferentes valores numéricos.

O compilador foi desenvolvido em 3 fases, como os compiladores da ferramenta ARC, e gera uma representação intermediária do modelo da aplicação na forma de uma árvore de sintaxe abstrata, a qual é analisada e convertida em expressões matemáticas.

A primeira fase do compilador é a de análise léxica. Nela é definida a estrutura léxica da linguagem PAMELA Estendida para que, com o uso da ferramenta JLex, seja criado um analisador léxico para reconhecer as cadeias de caracteres ou tokens da linguagem. Os tokens representam as instruções utilizadas para modelar efeitos de atraso no modelo, que é o caso de operações de computação e de comunicação.

A segunda fase é a criação de um analisador sintático para a linguagem utilizada no modelo da aplicação. Nesta fase é definida a gramática da linguagem PAMELA Estendida para reconhecer a estrutura organizacional dos tokens da linguagem. A partir da gramática, é utilizada a ferramenta CUP para produzir um analisador sintático.

A terceira fase é a geração de uma representação intermediária na forma de uma árvore de sintaxe abstrata. A árvore é gerada pelo analisador sintático que, a partir dos tokens reconhecidos pelo analisador léxico, cria os nós da árvore de acordo com a representação de cada token.

A Fig. 4.22 ilustra um modelo de aplicação e a Fig. 4.23 a árvore de sintaxe abstrata correspondente produzida por AMC. Tal árvore possuitem o mesmo tipo de estrutura de dados criada pela ferramenta ARC, onde é feita a análise do código-fonte de aplicações Java.

```
ProcessData0(p)={
   assign_intp;
   assign_floatp;
   sum_intp;
   for(i=1,400000){
     assign_intp;
     sum_intp;
     mpy_floatp;
     mpy_doublep;
   }
}
```

Fig. 4.22: Modelo da aplicação para um código com operações aritméticas simples

Após a criação da árvore mencionada, o compilador realiza uma combinação simbólica dos modelos de máquinas e da aplicação, produzindo um conjunto de expressões em que os parâmetros do tipo op_data_p e $send_{p \to q}$ representados na árvore são convertidos, respectivamente, para os parâmetros $inter\_op\_data_p$ e $inter\_send_{p \to q}$ do modelo de máquinas. As expressões representam o tempo de execução do modelo em duas situações: quando não há contenção e quando há contenção no uso de recursos de forma compartilhada.

Na ausência de contenção é gerada a expressão parcial T_0 (ver seção 3.2.3). Para isso, a ferramenta analisa a árvore nó a nó, fazendo as transformações necessárias. Nós que representam instruções seqüenciais transformam-se em adição e nós que indicam instruções paralelas são representados pelo operador max, que resulta no maior tempo de todos os blocos executados de forma paralela.

Na presença de contenção, é gerada uma expressão para cada recurso utilizado de forma compartilhada na plataforma de processamento paralelo. Desta forma, para um conjunto de R recursos utilizados de forma paralela no modelo da aplicação, é gerado um total de R expressões, dadas por T_r ($r = 1..R$) (ver seção 3.2.3). Para a criação de uma expressão T_r, para um recurso r qualquer, o compilador AMC verifica os nós da árvore e busca todos aqueles que fazem uso do recurso analisado. Quando um recurso é utilizado de forma paralela por um conjunto de tarefas, o tempo gasto para processá-las equivale a soma dos tempos de todas as tarefas que utilizam o recurso de forma compartilhada.

Na presença de alguma estrutura condicional do tipo $if\{blc_1\}else\{blc_2\}$ na árvore, o sistema produz uma expressão matemática equivalente à média aritmética entre os tempos dos blocos blc_1 e blc_2 da estrutura.

A estrutura condicional if pode também aparecer com vários comandos do tipo $else$ aninhados no modelo da aplicação para uma mesma instrução condicional. Neste caso, é considerado a maior média aritmética entre os tempos gerados em cada bloco exclusivo do modelo da aplicação. Tal tempo é calculado na forma simbólica, é conhecido como pior caso.

O pior caso é considerado na produção das expressões matemáticas quando há estruturas condicionais no modelo analisado, para tentar evitar que a avaliação de um modelo analítico gere estimativas de tempo muito distantes dos tempos de execução. Outras formas que poderiam ser consideradas para o cálculo das estimativas são o caso médio e o melhor caso. No primeiro, é considerado a média entre os tempos do bloco da estrutura condicional e no segundo considera-se o bloco da estrutura condicional com o menor tempo.

Desta forma, se a predição considerar o caso médio, e a aplicação executar poucas vezes o código deste bloco, a qualidade das estimativas será ruim. Por outro lado, considerar o melhor caso pode gerar um problema similar, mas desta vez, o tempo estimado pode ser um intervalo com tempos muito inferiores aos tempos de execução da aplicação. A alternativa adotada é conservadora, pois garante que os tempos estimados jamais sejam inferiores aos tempos de execução.

Uma alternativa para melhorar a qualidade das estimativas fornecidas por APET, seria configurá-la para que, na presença de estruturas condicionais no código, fosse informada uma taxa indicando o percentual Q de vezes que cada bloco da estrutura condicional seria executado. Este percentual pode

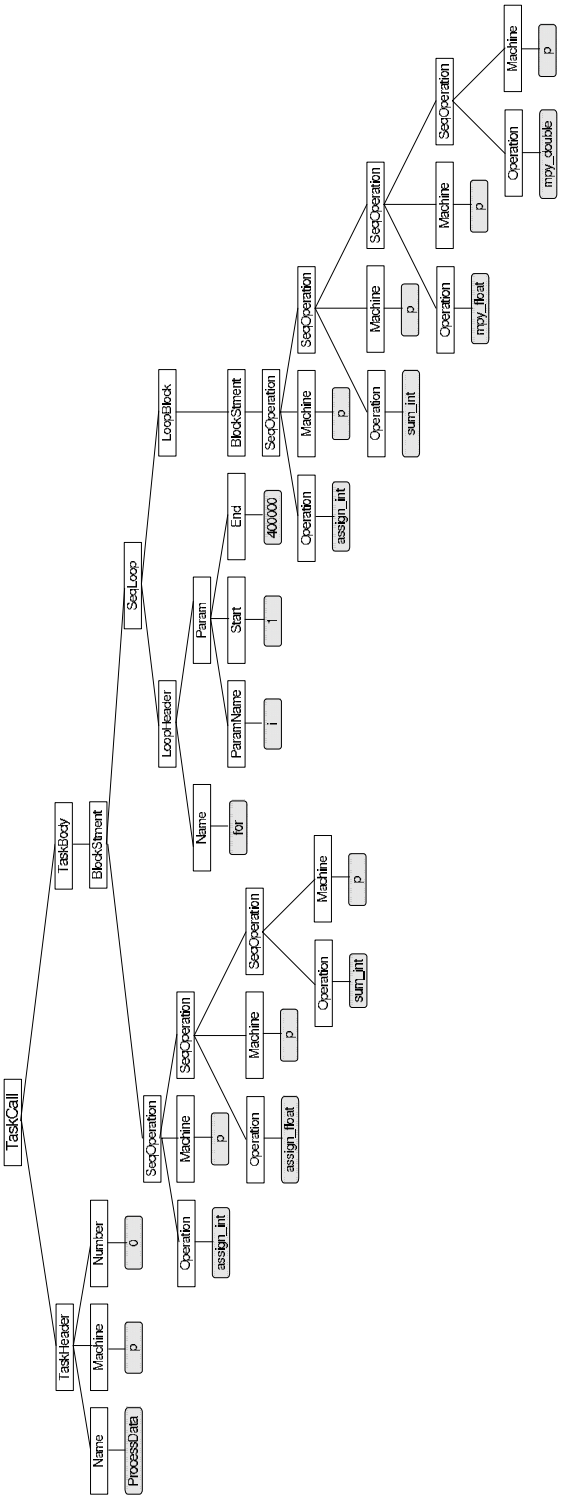

Fig. 4.23: Árvore de sintaxe abstrata de um modelo de aplicação correspondente ao modelo da Fig. 4.22

então ser utilizado para calcular o intervalo de tempo da predição da aplicação de forma mais precisa.

Desta forma, considerando uma estrutura condicional do tipo $if(cond)\{blc_1\}else\{blc_2\}$ que verifica, por meio da condição $cond$, a validade de alguma instrução e que os blocos blc_1 e blc_2 realizam algum tipo de operação se $cond$ for verdadeiro e falso, respectivamente, é possível calcular um intervalo de tempo esperado, chamado T_e, da aplicação. Para calcular T_e, basta somar o atraso de tempo τ_{cond} da condição $cond$ com o resultado da multiplicação de Q com o atraso de tempo τ_{blc_1} do bloco blc_1 e o resultado da multiplicação de $(1-Q)$ com o atraso de tempo τ_{blc_2} do bloco blc_2. Assim, T_e é expressado por $T_e = \tau_{cond} + Q \times \tau_{blc_1} + (1-Q) \times \tau_{blc_2}$. Para demonstrar como isso pode ser feito, será ilustrado no Cap. 5 como as estimativas fornecidas pelo sistema de predição proposto podem ser melhoradas com a adoção deste mecanismo.

Um problema desta abordagem é que há a necessidade de se ter informações detalhadas do algoritmo que será executado para que se possa estimar o percentual Q de vezes que um bloco de uma estrutura condicional será executado.

Uma outra alternativa para se obter estimativas de tempo mais precisas seria realizar uma análise abstrata das expressões condicionais do código. Tal abordagem é complexa para ser implementada e necessita que parte do algoritmo da aplicação seja executado por alguma ferramenta para que informações extras sejam coletadas.

Uma terceira alternativa de melhoria de precisão seria executar parte do código utlizando um conjunto reduzido dos dados da aplicação para que se obtenham informações precisas da freqüência estimada de execução de blocos que compõem estruturas condicionais. Tal abordagem envolve simulação e não é adequado para a ferramenta proposta, pois esta utiliza uma abordagem analítica apenas, a qual não envolve o uso de informações produzidas por meio de simulação ou de execução na própria plataforma paralela.

Após a criação das expressões, com ausência e presença de contenção, em que é considerado o pior caso, é feita a geração da expressão final T (ver seção 3.2.3) do modelo analítico de desempenho. Esta expressão é representada em função das expressões parciais T_0 e T_r, e assume o maior valor entre elas. Tais expressões não possuem ainda um valor determinado em relação à quantidade e ao desempenho de cada computador que será utilizado para estimar o tempo de execução de uma aplicação. Isto é tarefa para o próximo módulo (TPT) apresentado a seguir.

4.5.2 TPT: Geração de tempos de predição

A geração das estimativas de tempo na forma de um intervalo de valores é feita pela ferramenta TPT (*Time Prediction Tool*), representada pela etapa 2 da Fig. 4.21. Nela, é necessário informar uma quantidade P de computadores e fornecer um modelo de máquinas, os quais serão utilizados para realizar a predição a partir de um modelo analítico de desempenho.

TPT utiliza tais informações para substituir os parâmetros do tipo $inter\_op\_data_p$ e $inter\_send_{p \rightarrow q}$ de todas as expressões parciais que fazem parte da expressão final T do modelo analítico de desem-

penho, em que o custo de tarefas alocadas para um determinado computador tem seus parâmetros substituídos pelos respectivos valores contidos no modelo das máquinas informado.

Em seguida, a expressão final T é avaliada por TPT, o qual fornece como resultado um intervalo de valores do tipo $[t_{min}; t_{max}]$, que representa os tempos mínimo e máximo esperados na execução da aplicação. Para avaliar as expressões, TPT realiza operações aritméticas básicas sobre intervalos, como adição, subtração, divisão, multiplicação, e outras como max e sum, que retornam o maior e o somatório dos elementos de uma lista, respectivamente.

Para estimar o tempo que a mesma aplicação poderá demorar para que seja executada em uma quantidade de computadores inferior ao disponível no modelo das máquinas, basta informar no momento da geração de uma estimativa um novo valor para P, que não ultrapasse a quantidade de computadores representados no modelo de máquinas. Isso faz com que TPT selecione os P computadores mais velozes do modelo das máquinas usando o fator de desempenho relativo de cada um.

Além da variação da quantidade de computadores para a geração de estimativas, pode-se utilizar um modelo de máquinas que possua informações fictícias de computadores utilizados para a predição. Desta forma, é possível também verificar o impacto de tempo causado quando uma aplicação for executada em um conjunto de computadores homogêneos com desempenho igual, por exemplo, ao do computador mais rápido da plataforma de processamento paralelo.

Como apresentado, a partir de um único modelo analítico de desempenho, pode-se utilizar diferentes valores para P. Além disso, pode-se modificar o modelo das máquinas para estimar o intervalo de tempo que o modelo analítico demorará quando for utilizado um conjunto de computadores, homogêneos ou heterogêneos, com desempenho similiar ao do computador mais rápido, mais lento, ou algum outro em particular. Desta forma, a predição pode ser feita utilizando modelos de máquinas variados, para analisar o comportamento de uma aplicação com o uso de plataformas de processamento paralelo formados por computadores fictícios.

Para verificar a validade dos resultados produzidos pelo sistema de predição proposto, em todas as suas fases de desenvolvimento, foram comparados os resultados obtidos por cálculos manuais com os gerados pelo sistema. Desta forma, tanto os modelos quanto os cálculos realizados para obter estimativas de tempo foram comparados para diferentes tipos de aplicações. Uma análise detalhada a respeito da validade dos modelos criados pela linguagem PAMELA Estendida pode ser encontrada no trabalho de Yero [Yer03].

4.6 Considerações

Este capítulo apresentou uma proposta de um sistema de modelagem e predição analítica de desempenho, chamado PeMPSys, desenvolvido especialmente para o sistema de processamento paralelo JoiN.

PeMPSys cria modelos utilizando uma notação de modelagem de desempenho chamada PAMELA Estendida. Os três modelos gerados pelo sistema de predição - máquinas, aplicação e analítico de de-

sempenho - podem ser utilizados para analisar o intervalo de tempo esperado na execução de uma aplicação considerada, sob diferentes aspectos em relação a: quantidade e desempenho dos computadores, código da aplicação e viabilidade no tempo para executar uma aplicação. Para a criação dos três modelos citados, foram apresentadas três ferramentas.

Para a geração do modelo de máquinas, a ferramenta CRES desenvolvida foi integrada à plataforma de processamento paralelo JoiN para que a criação de tal modelo seja realizada no momento em que nenhuma aplicação estiver sendo executada na plataforma. Outra parte do sistema de predição que corresponde à criação de um modelo da aplicação é feita pela ferramenta ARC. Ela possui os módulos SCAn, PASC e AMGen. O primeiro foi desenvolvido especialmente para analisar o código-fonte de uma aplicação em Java e é capaz de representar na forma de uma árvore de sintaxe abstrata toda a estrutura interna de um programa e permitir, desta forma, que as informações de atraso sejam coletadas e utilizadas em conjunto com outras informações. O segundo foi desenvolvido especialmente para analisar o código-fonte de uma especificação do fluxo de uma aplicação paralela em PASL e é capaz de representar na forma de um grafo de tarefas toda a estrutura do fluxo de execução de uma aplicação e, desta forma, que as informações de paralelismo de uma aplicação sejam coletadas. O terceiro módulo faz uso das informações dos dois primeiros para produzir o modelo da aplicação.

A partir disso, a ferramenta chamada APET é responsável pela combinação simbólica do modelo de máquinas com o da aplicação e pela geração das predições. Ela possui uma estrutura de funcionamento baseada na substituição de instruções do modelo de máquinas no modelo da aplicação e na avaliação de expressões matemáticas simples, produzindo um intervalo de tempo esperado.

Para a criação dos modelos não é necessário realizar qualquer tipo de mudança na estrutura das aplicações. Todo o sistema foi desenvolvido com o uso de informações disponíveis no sistema de processamento paralelo JoiN. Dessa forma, não são feitas exigências quanto à marcação do código-fonte ou utilização de uma nova linguagem de programação não padrão para permitir o uso do sistema de predição na versão atual da plataforma JoiN. Além disso, os modelos criados pelas ferramentas podem ser modificados e utilizados para analisar o desempenho quando variam a quantidade e a configuração dos computadores que serão utilizados para processar uma aplicação.

Capítulo 5

Resultados experimentais

Este capítulo apresenta e discute resultados de experimentos realizados com o sistema de predição PeMPSys, para avaliar suas principais características.

Para os experimentos foram utilizadas algumas aplicações típicas, implementadas e executadas na plataforma de processamento paralelo JoiN. A seguir, são descritos a configuração dos computadores utilizados e das aplicações consideradas e os resultados obtidos em cada um dos experimentos.

5.1 Confi guração dos computadores

Um conjunto de 33 computadores foi empregado para realizar o processamento das aplicações, sendo um coordenador/servidor e os demais trabalhadores, com diferentes configurações, tanto de hardware quanto de software, conforme Tab. 5.1. Os computadores fazem parte de 4 sub-redes distintas (A, B, C e D) e se conectam por meio de switches. A velocidade dos processadores varia de 850 MHz a 1.8 GHz, e a quantidade de memória entre 128 e 512 MB. Em relação à velocidade da rede, com exceção dos computadores da sub-rede B, que possuem Ethernet de 10 Mbps, todos os demais estão conectados por redes Fast Ethernet de 100 Mbps.

É importante destacar que os computadores listados na tabela não fazem parte de um ambiente dedicado exclusivamente ao processamento das aplicações da plataforma de processamento paralelo e que não há informações suficientes para determinar os tipos de aplicativos instalados e o padrão de utilização das mesmas. Uma alternativa para tentar determinar isso seria obter o perfil de utilização das máquinas para que a previsão das aplicações seja feita de forma mais eficiente, o qual é deixado para um trabalho futuro.

O fator de desempenho relativo é utilizado pelo sistema de predição para identificar a quantidade de tarefas que será alocada a cada computador da plataforma. Tal fator pode sofrer variação caso seja medido variadas vezes devido a ausência de controle do ambiente de experimentos, computadores e rede, considerado. A quinta coluna da Tab. 5.1 lista os fatores de desempenho de cada um dos 32

Tab. 5.1: Computadores utilizados nos experimentos e suas respectivas configurações

Sub-Rede: Computador	**Índice(p)**	**Arquitetura**	**Memória(Mb)**	**Fator de desempenho(W_p)**
A:trancoso	0	Pentium III 1133 MHz	384	-
B:itu	1	Pentium III 850 MHz	128	1.0000
B:ilhabela	2	Pentium III 850 MHz	128	0.9994
B:maceio	3	Pentium III 850 MHz	128	0.9232
B:buzios	4	Pentium III 850 MHz	128	0.9219
B:hortolandia	5	Pentium III 850 MHz	128	0.9215
B:ourofino	6	Pentium III 850 MHz	128	0.9209
B:fortaleza	7	Pentium III 850 MHz	128	0.9205
B:ubatuba	8	Pentium III 850 MHz	128	0.9198
B:guaruja	9	Pentium III 850 MHz	128	0.9163
C:variac	10	Pentium 4 1.8 GHz	256	0.8595
C:bobina	11	Pentium 4 1.8 GHz	256	0.8587
C:campinas	12	Pentium 4 1.8 GHz	256	0.8585
C:transistor	13	Pentium 4 1.8 GHz	256	0.8584
C:varicap	14	Pentium 4 1.8 GHz	256	0.8581
C:indutor	15	Pentium 4 1.8 GHz	256	0.8579
C:varistor	16	Pentium 4 1.8 GHz	256	0.8578
C:transformador	17	Pentium 4 1.8 GHz	256	0.8576
C:baudelaire	18	Pentium 4 1.8 GHz	256	0.8575
C:zener	19	Pentium 4 1.8 GHz	256	0.8568
C:resistor	20	Pentium 4 1.8 GHz	256	0.8568
D:img	21	Pentium 4 1.5 GHz	512	0.6365
D:png	22	Pentium 4 1.5 GHz	512	0.6364
D:gif	23	Pentium 4 1.5 GHz	512	0.6358
D:qtm	24	Pentium 4 1.5 GHz	512	0.6356
D:dxf	25	Pentium 4 1.5 GHz	512	0.6355
D:wmf	26	Pentium 4 1.5 GHz	512	0.6354
D:mpeg	27	Pentium 4 1.5 GHz	512	0.6354
D:pict	28	Pentium 4 1.5 GHz	512	0.6352
D:tga	29	Pentium 4 1.5 GHz	512	0.6350
D:mov	30	Pentium 4 1.5 GHz	512	0.6348
D:eps	31	Pentium 4 1.5 GHz	512	0.6345
D:rib	32	Pentium 4 1.5 GHz	512	0.6288

computadores utilizados nos experimentos. Na tabela, os índices (segunda coluna) relativos a cada computador foram atribuídos em ordem decrescente de fator de desempenho relativo. O computador 0, representado pelo computador que atua simultaneamente como servidor e coordenador (ver seção 3.3.1), não possui um fator de desempenho relativo associado, pois tal fator é utilizado exclusivamente para o escalonamento de lotes de tarefas cuja cardinalidade é superior a um. No computador coordenador de JoiN são alocadas apenas as tarefas dos lotes cuja cardinalidade é unitária.

Uma rápida análise dos fatores de desempenho relativos apresentados na Tab. 5.1 mostra que os computadores mais velozes da plataforma paralela têm configuração de hardware considerada mais lenta (menor quantidade de memória e clock mais lento).

Os computadores das sub-redes C e D são equipados com o processador Pentium 4 que, apesar de apresentar freqüência de clock superior ao do seu antecessor Pentium III, é baseado em uma arquitetura conhecida como NetBurst [Pab00]. Tal arquitetura é composta por vários componentes que implementam tecnologias distintas, entre elas a Hyper Pipelined Technology, cuja principal característica é a capacidade de dividir o processador para que opere em 20 estágios diferentes de pipeline, aos invés dos 10 estágios apresentados pelo Pentium III [Die99].

O uso de mais estágios de pipeline permite que o processador opere a freqüências maiores, pois cada estágio é mais simples. Apesar da maior capacidade em executar instruções de forma simultânea, boa parte das aplicações é composta por sequência de instruções com desvios, o que dificulta o aproveitamento de vários estágios. Para otimizar o processamento de tais aplicações, o processador utiliza um recurso de execução especulativa que visa adiantar parte do processamento, mesmo que ela não seja executada. Caso não seja executada, os ciclos gastos para otimização do processamento são perdidos e, desta forma, quanto maior a quantidade de estágios de pipeline de um processador, maior será a chance de se desperdiçar trabalho [PSS05].

Para compensar este problema, uma unidade de execução de operações inteiras, que já existia em versões anteriores do processador, sofreu modificações e tornou-se mais rápida. Porém, uma outra unidade, responsável pela execução de instruções complexas (ponto flutuante) não sofreu a mesma evolução. Além disso, o co-processador aritmético também não teve avanços [PSS05]. Houve ainda a eliminação de duas das unidades de execução, uma delas responsável pelo processamento de instruções MMX (Multimedia Extensions) [ea97] e a outra responsável pelo processamento de instruções do tipo SSE (Streaming SIMD Extensions) [Die99].

Desta forma, apesar dos computadores Pentium III apresentarem freqüências inferiores às do Pentium 4, eles podem, em alguns casos, executar aplicações mais rapidamente, como demonstra o trabalho publicado em [PSS05].

5.2 Configuração das aplicações

As aplicações experimentadas foram divididas em duas categorias: quanto ao determinismo e quanto à razão computação/comunicação ($r_{cc} = tempo\_computacao/tempo\_comunicacao$).

Uma aplicação é dita determinista no contexto deste trabalho se é possível determinar pela análise do seu código a quantidade de operações executadas. Uma característica de tal aplicação é que, na maioria das vezes, não há estruturas condicionais no código. Determinar a quantidade de operações executadas por uma aplicação é fundamental para a qualidade das predições. Por outro lado, as aplicações ditas não deterministas não permitem obter a quantidade de operações executadas, tornando as estimativas menos precisas. Uma das características peculiares em tais aplicações é a presença de estruturas condicionais, as quais podem determinar que a aplicação siga por diferentes fluxos a cada execução, podendo alterar significativamente a carga computacional.

Quanto à razão r_{cc}, as aplicações podem ser divididas em 3 classes:

(i) $r_{cc} >> 1$ (aplicação ideal para JoiN pois obtém uma maior eficiência na utilização dos computadores)

(ii) $r_{cc} > 1$ (aplicação pode ser usada em JoiN, mas não será muito eficiente pelo fato da plataforma usar redes heterogêneas e compartilhadas)

(iii) $r_{cc} \leq 1$ (aplicação não considerada, por não sere adequada para se executar em JoiN ou em outras plataformas paralelas)

Nos experimentos foram consideradas apenas as classes (i) e (ii).

Com base nas categorias descritas, a Tab. 5.2 apresenta as aplicações experimentadas e as características de cada uma quanto ao determinismo do código e quanto à classe a que pertencem.

Tab. 5.2: Aplicações utilizadas nos experimentos

Nome da aplicação	**Código determinista?**	**Razão computação/comunicação (r_{cc})**
Cálculo do número π	sim	$r_{cc} >> 1$
Busca de números primos	não	$r_{cc} >> 1$
Operações completas em matrizes	sim	$r_{cc} > 1$
Operações parciais em matrizes	não	$r_{cc} > 1$

Para os experimentos das aplicações da Tab. 5.2, foram feitas parametrizações quanto a:

- quantidade de computadores utilizados para executar a aplicação;
- quantidade de tarefas;
- quantidade de dados enviados pela rede para processamento.

O algoritmo utilizado para o cálculo do número π é baseado no método de Monte Carlo [Bec77], no qual é utilizado um conjunto de N pontos que representam coordenadas x e y geradas aleatoriamente. Quanto maior a quantidade de pontos gerados, maior é a precisão do número π encontrado. Esta aplicação foi experimentada com 4 versões, nas quais variava-se a quantidade de tarefas e de pontos verificados (Tab. 5.3). Cada tarefa recebe um mesmo número N/M (onde M representa o total de tarefas) indicando a quantidade de pontos a serem gerados e tratados por ela.

Tab. 5.3: Versões da aplicação de cálculo do número π

Cálculo do número π (Monte Carlo)		
Versão	**Tarefas (M)**	**Pontos totais (N)**
π_A	100	1×10^9
π_B	100	2×10^9
π_C	200	1×10^9
π_D	200	2×10^9

Para a aplicação de busca de números primos, o algoritmo utilizado é baseado em um método conhecido como Crivo de Eratóstenes [CP01], no qual são identificados todos os números primos de um determinado intervalo numérico. Esta aplicação foi experimentada com 4 versões, conforme listagem da Tab. 5.4, nas quais variou-se a quantidade de tarefas e o tamanho total do intervalo de busca. Cada tarefa recebeu uma parcela do intervalo, que foi dividido em partes iguais.

Tab. 5.4: Versões da aplicação de busca de números primos

Busca de números primos		
Versão	**Quantidade de tarefas (M)**	**Intervalo de busca**
prim_A	100	0 a 5×10^5
prim_B	100	0 a 1×10^6
prim_C	200	0 a 5×10^5
prim_D	200	0 a 1×10^6

Para a aplicação que realiza operações completas em matrizes, foi desenvolvido um algoritmo fictício, no qual é considerado um conjunto de matrizes quadradas com dimensões iguais entre elas. Para cada matriz, é calculado o seu quadrado e, em seguida, para cada um dos seus elementos, são realizadas várias operações matemáticas básicas de adição, subtração, multiplicação e divisão. Esta aplicação também foi experimentada com 4 versões, conforme a listagem da Tab. 5.5. Cada tarefa recebeu uma matriz do tamanho especificado pela aplicação e fez os cálculos sobre ela.

Para a aplicação que realiza operações parciais em matrizes, foi desenvolvido um algoritmo fictício (baseado na aplicação que realiza operações completas em matrizes), no qual é considerado um conjunto de matrizes quadradas com dimensões iguais entre elas. Para cada matriz, é calculado o seu quadrado e, em seguida, para elementos selecionados aleatoriamente da matriz, são realizadas várias

Tab. 5.5: Versões da aplicação que realiza operações completas em matrizes

Operações completas em matrizes		
Versão	**Quantidade de tarefas e de matrizes (M)**	**Dimensão das matrizes**
mat_A	100	128
mat_B	100	256
mat_C	200	128
mat_D	200	256

operações matemáticas básicas de soma, subtração, multiplicação e divisão. Logo o volume de processamento em cada tarefa é variável e imprevisível. Esta aplicação também foi experimentada com 4 versões, conforme a listagem da Tab. 5.6. Cada tarefa recebeu uma matriz do tamanho especificado pela aplicação e fez os cálculos sobre ela.

Tab. 5.6: Versões da aplicação que realiza operações parciais em matrizes

Operações parciais em matrizes		
Versão	**Quantidade de tarefas e de matrizes (M)**	**Dimensão das matrizes**
rand_A	100	128
rand_B	100	256
rand_C	200	128
rand_D	200	256

5.3 Cálculo do erro das estimativas

Analisar o erro das estimativas fornecidas por um sistema de predição de desempenho é importante para verificar sua precisão.

Um método utilizado para calcular o erro em sistemas de predição é baseado na Eq. 5.1, em que o erro relativo E de uma estimativa equivale à subtração dos tempos de execução T_X e estimado T_E, dividido pelo tempo estimado T_E. Quanto mais próximo de zero for o valor do erro E, melhor será a estimativa. Na presença de erro, o sinal de E indicará se o tempo de execução T_X foi maior (sinal positivo) ou menor (sinal negativo) do que o intervalo de tempo estimado T_E. Tal método calcula o erro dos tempos de execução com relação a estimativa produzida, pois este é fixo e os tempos de execução variam de um para outro, para um mesmo T_E.

$$E = \frac{T_X - T_E}{T_E} \tag{5.1}$$

Optou-se por calcular o erro com uma fração (percentual) da estimativa T_E pelo fato desta ser constante durante várias execuções dos programas, facilitando a comparação dos erros relativos.

Sistemas de predição como PAMELA, PACE, PerPret e SUIF, descritos no Cap. 2, utilizam o método descrito pela Eq. 5.1 para calcular o erro nas suas estimativas. Em tais sistemas, a estimativa de tempo que representa a predição é indicada na forma de um valor pontual.

Entretanto, no sistema de predição PeMPSys, os tempos de predição são calculados na forma de intervalos de valores do tipo $[t_{min}; t_{max}]$. Desta forma, é importante adotar algum método que, para o cálculo do erro de uma estimativa, leve em consideração todo o intervalo calculado.

Para que isso seja feito, é utilizado um cálculo simples que se baseia na mesma idéia da Eq. 5.1. Considerando que o tempo da estimativa T_{E_P} para uma quantidade de P computadores é um intervalo do tipo $[t_{min}; t_{max}]$, que $\overline{T_{E_P}}$ é a média aritmética de t_{min} e t_{max} para P e que o tempo da i-ésima execução em P computadores é $T_{X_{P,i}}$, o cálculo do erro $E_{P,i}$ pode ser feito de acordo com a Eq. 5.2. Nela, o sinal de $E_{P,i}$ indica se o erro do tempo de execução $T_{X_{P,i}}$ ficou acima (sinal positivo) ou abaixo (sinal negativo) do intervalo de tempo estimado T_{E_P}.

$$E_{P,i} = \begin{cases} 0 & \text{se } t_{min} \leq T_{X_{P,i}} \leq t_{max} \\ \frac{T_{X_{P,i}} - t_{max}}{\overline{T_{E_P}}} & \text{se } T_{X_{P,i}} > t_{max} \\ \frac{T_{X_{P,i}} - t_{min}}{\overline{T_{E_P}}} & \text{se } T_{X_{P,i}} < t_{min} \end{cases} \tag{5.2}$$

Para calcular o erro médio $\overline{E}_P$ a partir de cada $E_{P,i}$ para P computadores e k execuções, utiliza-se a Eq. 5.3.

$$\overline{E}_P = \frac{1}{k} \sum_{i=1}^{k} |E_{P,i}| \tag{5.3}$$

Vale ressaltar que as aplicações consideradas neste trabalho servem apenas para exemplificar a utilizadade do sistema de predição de desempenho proposto. Desta forma, em hipótese alguma definem os limites dos métodos empregados para produzir estimativas de tempo.

5.4 Resultados dos experimentos

Os experimentos das 4 versões de cada aplicação foram feitos em diferentes horários do dia e da noite utilizando um mesmo conjunto de computadores, independentemente se estavam ou não compartilhados com outros usuários.

Para uma mesma configuração de uma aplicação foram realizadas 4 execuções, e variou-se a quantidade de computadores, iniciando com os 32 computadores disponíveis e reduzindo, em cada experimento, a quantidade pela metade. Removeram-se sempre os mais lentos primeiro, até que restassem

apenas os dois mais velozes. Para uma determinada quantidade de computadores, independentemente do tipo ou da versão da aplicação experimentada, utilizou-se sempre as mesmas máquinas.

Para obtenção das estimativas, foi utilizado um modelo das máquinas atualizado no momento da realização de cada experimento. Os resultados de cada estimativa foram comparados com os tempos de execução da aplicação, para efeito de avaliação do erro $E_{P,i}$ (calculado por meio da Eq. 5.2) das predições de desempenho e da média do erro $\overline{E}_P$. Estes dois erros são representados nas tabelas em termos de porcentagem, o qual é obtido pela multiplicação do valor calculado por 100.

Para se obter cada estimativa de desempenho para as aplicações, foram gastos em média 7 segundos, quando o sistema de predição foi executado em um computador Pentium III, 800 MHz, 384 Mb RAM, Windows 2000 e versão 1.4.2 da Máquina Virtual Java.

5.4.1 Cálculo do número π

Esta aplicação, conforme apresentada na seção 5.2, caracteriza-se por ser determinista e possuir razão $r_{cc} >> 1$. Em função de tais características é esperada que as estimativas fornecidas pelo sistema de predição sejam boas, pois sabe-se a quantidade de operações realizadas pelo código e o uso da rede não afetará o tempo de execução da aplicação, pois o tempo de processamento é muito maior do que o tempo gasto para enviar os dados das tarefas pela rede. A seguir, são apresentados resultados obtidos com as 4 versões experimentadas desta aplicação. É visto que elas apresentam bons resultados, pois os tempos de execução obtidos, em todos os casos, estiveram dentro ou bem próximos do intervalo de tempo estimado.

Os resultados obtidos com a versão π_A são apresentados na Tab. 5.7. Nesta versão, a maioria das estimativas apresentaram erros iguais ou bastante próximos de zero. Um dos motivos que contribuiu para isso foi porque esta versão realiza bastante processamento sobre poucos dados que são destinados a ela, fazendo com que a velocidade da rede não seja fator determinante no tempo de execução da aplicação. Um outro motivo é o determinismo do código, o qual permite determinar a quantidade de operações executadas pelo código para se estimar com melhor eficiência, por meio de tempos fornecidos por algoritmos de *benchmark*, o tempo gasto para executar as operações em determinada arquitetura.

Em 12 das 20 medições o tempo de execução se situou dentro do intervalo de tempo estimado ($E_{P,i}$). Apesar do erro diferente de zero em 8 execuções, em geral eles foram pequenos. O maior erro ocorreu quando foram utilizados 16 computadores e foi de $E_{16,2} = 15,02\%$ na segunda execução. Este elevado valor do erro pode ser atribuído à instabilidade do conjunto de computadores utilizado, o qual, em algum momento da execução da aplicação, sofreu perda de desempenho causando uma elevada contenção externa que não foi possível considerar na geração do intervalo da predição. Nesta mesma quantidade de computadores também foi identificada a maior freqüência de erros, com 3 das 4 estimativas erradas, e também a maior média de erro da versão, que foi de $\overline{E}_{16} = 7,27\%$. A menor média de erro foi de $\overline{E}_4 = 0,06\%$, quando foram utilizados 4 computadores. A média neste caso foi pequena porque apenas uma das 4 estimativas apresentou erro. Além disso, ele corresponde ao menor valor entre as 8 estimativas com erro da versão. Nota-se que, pela análise dos resultados para

todos os valores de P, que pelo menos uma das quatro estimativas apresentou erro. Em função da grande quantidade de acertos para os valores de P, a média de erro foi próxima de zero quando foram utilizados até 8 computadores, e não ultrapassou 8% quando a quantidade aumentou.

Tab. 5.7: Resultados da aplicação de cálculo do número π: versão π_A

Cálculo do número π: versão π_A					
P	i	$T_{X_{p,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ **(%)**	$\overline{E_P}$ **(%)**
2	1	726	[714;807]	0	0,26
	2	706		-1,05	
	3	774		0	
	4	758		0	
4	1	401	[364;411]	0	0,06
	2	380		0	
	3	412		0,26	
	4	389		0	
8	1	202	[196;221]	0	0,72
	2	190		-2,88	
	3	215		0	
	4	207		0	
16	1	125	[98;115]	9,39	7,27
	2	131		15,02	
	3	120		4,69	
	4	108		0	
32	1	75	[66;73]	2,88	2,88
	2	73		0	
	3	60		-8,63	
	4	71		0	

É importante verificar que, de acordo com a Tab. 5.7, com o aumento da quantidade de computadores o erro das estimativas apresentou um comportamento irregular. O aumento de 2 para 4 computadores reduziu o erro, mas o aumento de 4 para 8 e posteriomente para 16, fez com que o erro aumentasse gradativamente e, de 16 para 32 computadores, o erro voltou a diminuir. Pressupõe-se que a irregularidade nos erros ocorreu por influência de efeitos de contenção interna e externa, descritos na seção 3.2, os quais podem fazer com que o tempo de execução aumente ou diminua.

O gráfico da Fig. 5.1 ilustra a proximidade dos tempos de execução com relação às estimativas de tempo. Por meio dele é possível verificar que os tempos de execução que apresentaram erros em algumas estimativas permaneceram próximos dos intervalos. Nota-se que o aumento ou a diminuição na quantidade de computadores não fez com que a diferença das estimativas com relação aos tempos de execução piorassem. Com $P = 2$, 4 e 8, por exemplo, é visível que apenas um dos 4 tempos de execução ficou fora dos limites do intervalo estimado. Com $P = 16$ e 32, apesar do tamanho reduzido do intervalo, os tempos de execução fora dos intervalos permaneceram muito próximos a

estes, mostrando que a qualidade das estimativas é satisfatória. Isto demonstra, para este tipo de aplicação experimentada, que o uso de intervalos para tratar efeitos de contenção foi eficiente.

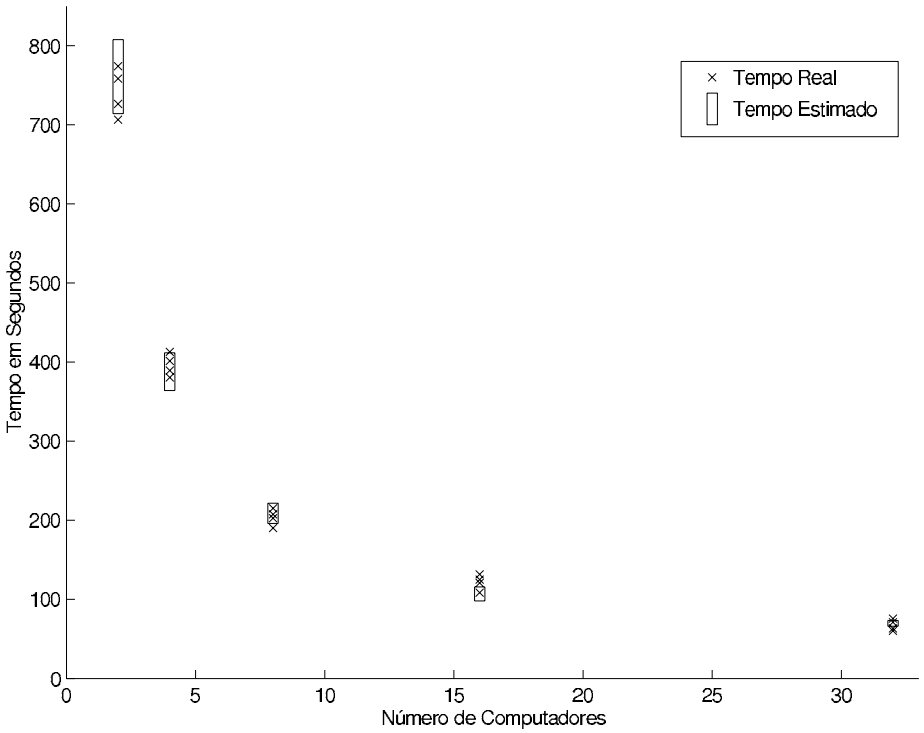

Fig. 5.1: Tempo para cálculo do número π: versão π_A

As outras 3 versões desta aplicação apresentaram resultados bastante similares a esta. Os resultados obtidos podem ser verificados no Apêndice C.1 deste trabalho.

Uma análise dos resultados gerados a partir das 4 versões da aplicação de cálculo do número π: π_A, π_B, π_C e π_D, mostrou que a maioria dos tempos de execução ficou entre os limites dos intervalos calculados. De um total de 80 tempos de execução (20 para cada versão), 55 se enquadraram dentro das estimativas. Isso representa quase 70% de todas as execuções realizadas pelas 4 versões da aplicação. O gráfico da Fig. 5.2 ilustra a distribuição de frequência da quantidade de estimativas que não apresentaram erro em cada uma das versões. Um dado importante é que a frequência de acertos das 4 versões é similar, mesmo quando a quantidade de computadores é modificada. Além disso, os resultados demonstraram que o aumento da quantidade de tarefas ou da carga computacional da aplicação não causa nenhum tipo de impacto aparente no tempo de execução que leve a uma degradação da qualidade das estimativas de uma versão para outra.

Desta forma, apesar das versões apresentarem erros de valor variado, as médias de erro (Tab. 5.8) ficaram próximas umas das outras. Nota-se na tabela que a diferença do maior e do menor erro médio é pequena e que os erros permaneceram em todos os casos abaixo de $2,3\%$. A versão com maior erro médio foi a π_A, com $\overline{E} = 2,238\%$, e a versão com o menor erro médio foi a π_D, com $\overline{E} = 0,95\%$. Para esta aplicação, o erro médio nas estimativas foi baixo porque grande parte dos

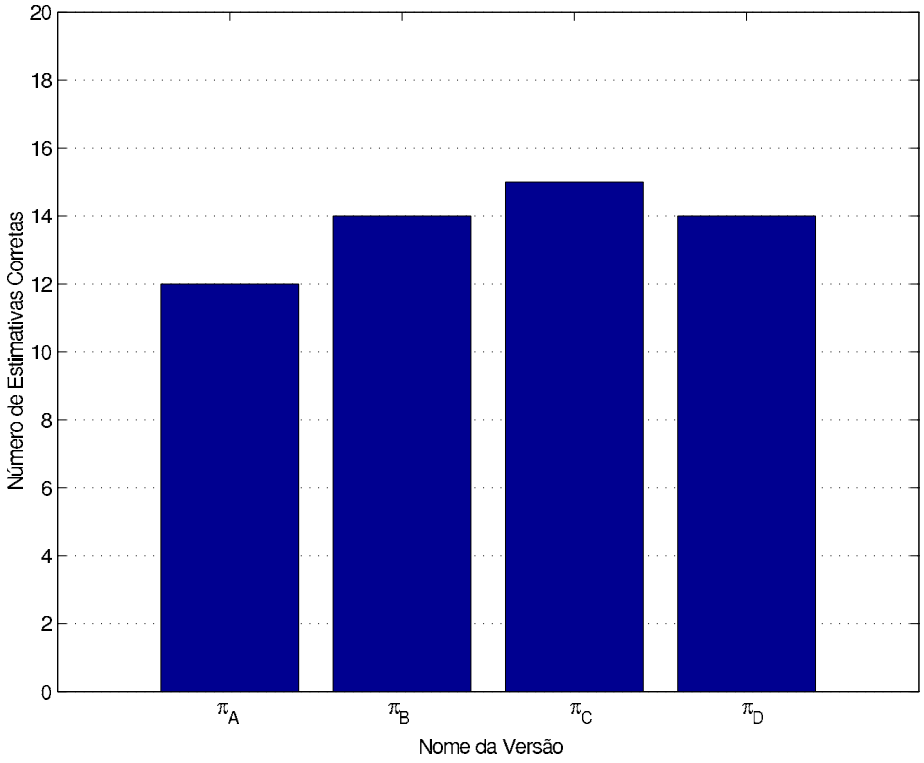

Fig. 5.2: Frequência de acertos de cada versão da aplicação do cálculo do número π

tempos de execução permaneceram bastante próximos dos intervalos das estimativas calculadas.

Tab. 5.8: Erro médio de cada versão da aplicação de cálculo do número π

Cálculo do número π: erro médio de cada versão	
Nome da Versão	**Erro Médio ($\overline{E}$) (%)**
π_A	2,238
π_B	1,212
π_C	1,18
π_D	0,95

Uma particularidade deste tipo de aplicação é o seu aparente determinismo, pois é possível determinar, mesmo com a presença de uma estrutura condicional no código, a quantidade aproximada de operações que será executada. Outra característica é o volume de dados enviado para a aplicação. Para que ela funcione, basta enviar para cada tarefa um único número que representa a quantidade de pontos a serem avaliados pelo método o que a torna mais imune a variações de desempenho na rede. Um outro tipo de aplicação, multiplicação de matrizes, apresenta características semelhantes à aplicação experimentada no que diz respeito à quantidade de operações executadas pelo algoritmo. Porém, o volume de dados enviados depende diretamente das dimensões das matrizes que serão multiplicadas e é bem superior ao existente neste exemplo. Neste caso o desempenho da predição é mais dependente da avaliação do desempenho da rede, como pode ser percebido nos dados presentes no Ap. C.

5.4.2 Busca de números primos

A segunda aplicação experimentada, busca de números primos, apresenta $r_{cc} >> 1$ e uma característica diferente da aplicação de cálculo do número π, que é a ausência de determinismo do código. As 4 versões da aplicação experimentada apresentaram, na maioria dos casos, tempos de execução inferiores aos intervalos calculados. A seguir serão apresentados os resultados obtidos.

Na primeira versão experimentada, chamada $prim_A$, cujos resultados são apresentados na Tab. 5.9, apenas 2 dos 20 tempos de execução coincidiram com as estimativas. Eles correspondem à segunda e à terceira execuções quando foram utilizados 16 e 2 computadores, respectivamente. Os demais tempos de execução permaneceram sempre abaixo do limite inferior dos respectivos intervalos de tempo estimados. Um motivo que contribuiu para que isso acontecesse foi a ausência de determinismo do código da aplicação, a qual não permite estimar de forma precisa a quantidade de operações executadas pelo código e, conseqüentemente, o tempo que será gasto para processar tais operações.

Nos experimentos com 8 computadores observou-se que 3 dos 4 erros são os maiores entre todos da versão, sendo o maior deles igual a $E_{8,2} = 25,29\%$, o qual corresponde à segunda execução. Conseqüentemente, isso contribuiu para que o erro médio para este valor de P também fosse o maior

Tab. 5.9: Resultados da aplicação de busca de números primos: versão $prim_A$

Busca de números primos: versão $prim_A$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,exec}$ **(%)**	$\overline{E}_P$ **(%)**
2	1	849	[885;1036]	-3,75	3,44
	2	802		-8,64	
	3	893		0	
	4	872		-1,35	
4	1	428	[470;550]	-8,24	8,82
	2	431		-7,65	
	3	442		-5,49	
	4	399		-13,92	
8	1	245	[324;356]	-23,24	18,90
	2	238		-25,29	
	3	269		-16,18	
	4	287		-10,88	
16	1	141	[158;185]	-9,91	5,40
	2	168		0	
	3	144		-8,16	
	4	152		-3,50	
32	1	90	[96;109]	-5,85	7,07
	2	83		-12,68	
	3	95		-0,98	
	4	87		-8,78	

de todos, que foi de $\overline{E_8} = 18,90\%$. Esta média foi muito maior do que a segunda maior média de erro, que foi de apenas $\overline{E_4} = 8,82\%$ com 4 computadores. A menor média de erro obtida entre todos foi de $\overline{E_2} = 3,44\%$, quando foram utilizados 2 computadores. Pressupõe-se que a grande variação no erro das estimativas para cada um dos valores de P foi causada por efeitos de contenção, mesmo quando são feitas parametrizações na mesma.

Apesar dos erros encontrados nas estimativas, a visualização do gráfico da Fig. 5.3 permite verificar que todos os intervalos das estimativas conseguiram acompanhar parcialmente a variação dos tempos de execuçao quando P variou. Além disso, é possível encontrar os dois únicos tempos de execução que ficaram dentro dos intervalos estimados, sendo um com 2 e outro com 16 computadores. Com 32 computadores, apesar do aparente acerto (apenas visual no gráfico) todas as 4 estimativas apresentaram erro. Nota-se também que, apesar da grande freqüência de erros (em 18 das 20 estimativas) verificada, os tempos de execução estão sempre próximos dos intervalos, o que demonstra uma qualidade razoável das estimativas, mesmo com a presença de estruturas condicionais no código.

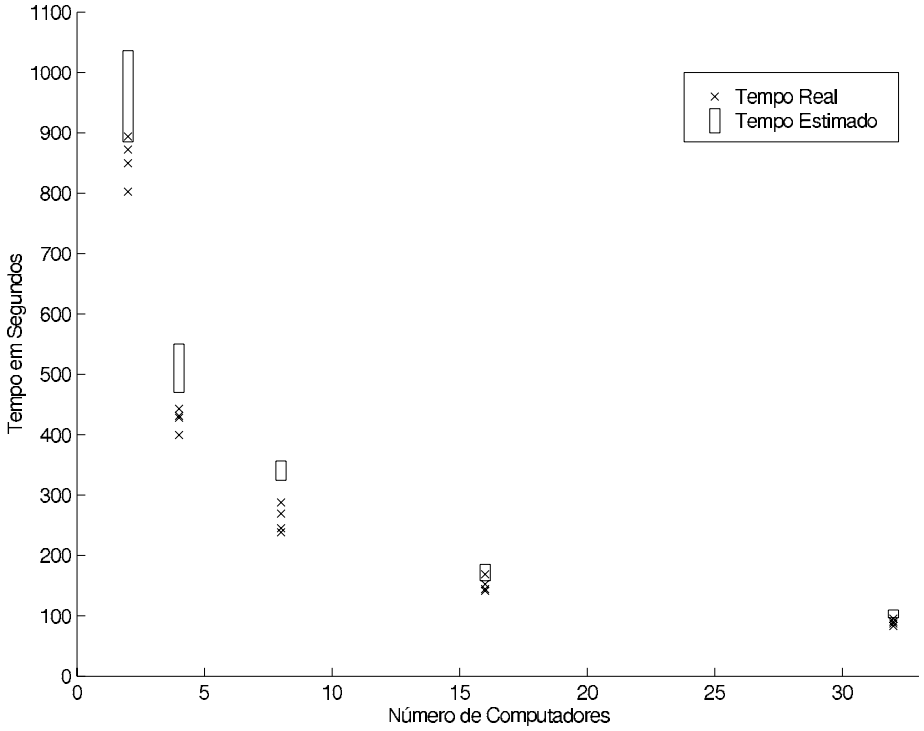

Fig. 5.3: Tempo para busca de números primos: versão $prim_A$.

As outras 3 versões desta aplicação apresentaram resultados bastante similares a esta. Os resultados obtidos podem ser verificados no Apêndice C.2 deste trabalho.

Uma análise dos resultados produzidos a partir das 4 versões da aplicação de busca de números primos mostrou que a maior parte das estimativas apresentou erro: das 80, 74 apresentaram erro. Isso representa uma taxa de acerto de apenas $7,5\%$. As 6 estimativas corretas ocorreram em versões diferentes e, em apenas uma delas, para um mesmo valor de P.

O gráfico da Fig. 5.4 ilustra o comparativo da freqüência de acertos nas estimativas das 4 versões da aplicação de números primos. A maior frequência de acertos, como pode ser visualizado, ocorreu na versão $prim_B$ com 3 acertos, ao passo que na versão $prim_D$ todas as estimativas apresentaram erro. Nota-se no gráfico que a quantidade de acertos não predominou em nenhuma versão.

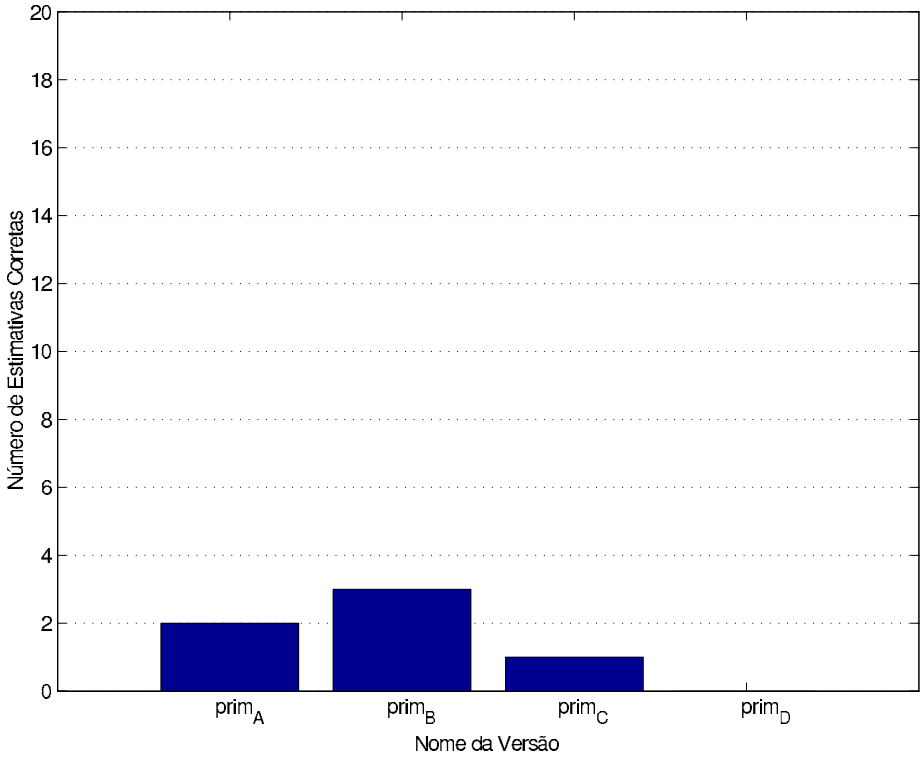

Fig. 5.4: Freqüência de acertos de cada versão da aplicação de busca de números primos

Em relação ao erro médio de cada versão, conforme pode ser observado na Tab. 5.10, a versão $prim_B$ foi a que apresentou o menor erro médio, que foi de $\overline{E} = 8,046\%$. Nota-se na tabela que a maior diferença de erro entre as versões voi relativamente grande, de aproximadamente 7%. Pelos resultados, nota-se que a mudança da quantidade de tarefas e da carga de computação não trouxeram evidências que fizessem com que as estimativas passassem a acertar ou errar mais, pois em nenhum dos casos ocorreu alguma variação constante nas estimativas e nos tempos obtidos.

O erro em praticamente todas as estimativas ocorreu porque elas são calculadas considerando o pior caso entre os distintos blocos a serem executados em uma estrutura condicional. O problema dessa abordagem nesta predição é que o bloco que faz parte da estrutura condicional (e que verifica se um número é primo) é executado poucas vezes. Tal bloco é composto por diversas operações que são responsáveis pela manipulação do número primo encontrado. A predição considera que tal bloco é executado uma quantidade muito maior de vezes do que na execução real da aplicação, motivo pelo qual as estimativas de tempo encontram-se sempre maiores do que os tempos de execução da aplicação. Se fosse conhecida a probabilidade do fluxo de execução utilizar um determinado bloco da

Tab. 5.10: Erro médio de cada versão da aplicação de busca de números primos

Busca de números primos: erro médio de cada versão	
Nome da versão	**Erro médio ($\overline{E}$) (%)**
$prim_A$	8,726
$prim_B$	8,046
$prim_C$	12,688
$prim_D$	15,712

estrutura condicional, esta informação poderia trazer maior precisão para as predições. Este aspecto é tratado na próxima seção.

5.4.3 Busca de números primos baseada na execução de um bloco condicional

Uma alternativa para melhorar a qualidade das estimativas fornecidas por APET seria informar uma taxa indicando o percentual Q de vezes que o bloco da estrutura condicional seria executado. Para a aplicação de busca de números primos, sabe-se que a quantidade $prim([a,b])$ de números primos que podem ser encontrados em um intervalo do tipo $[a,b]$ é dado por $prim([a,b]) = prim(b) - prim(a)$, em que $prim(x) \approx \frac{x}{\log x - 1}$. Em função disso, é possível determinar uma quantidade aproximada de vezes em que é verdadeira a condição que verifica se um número é primo. Para determinar isso, considera-se que a quantidade de números verificados em um intervalo $[a,b]$ é $q = b - a$, e que o percentual Q de números que podem ser primos no intervalo é dado pela relação $Q = \frac{prim([a,b])}{q}$. Este percentual pode então ser utilizado para calcular o intervalo de tempo da predição da aplicação de forma mais precisa, pois sabe-se a quantidade aproximada de vezes que um bloco da estrutura condicional será executado (vide seção 4.5.1).

Para demonstrar como isso pode ser feito, as estimativas da aplicação de busca de números primos foram recalculadas para verificar o erro de cada uma das 4 versões.

Na versão $prim_A$, após o cálculo das estimativas em função de Q, a maioria dos tempos ficaram dentro dos limites dos intervalos calculados. Os resultados desta versão, baseada na probabilidade de execução de um bloco condicional, são apresentados na Tab. 5.11. Das 20 estimativas de tempo, apenas 6 apresentaram algum erro, ao passo que no cálculo das estimativas sem considerar a probabilidade de execução de um bloco condicional, 18 das 20 estimativas apresentaram erro. O maior erro, que era de $E_{8,2} = 25,29\%$ passou a ser de $E_{8,2} = 12,82\%$. Com 4 computadores, todas as estimativas que apresentavam erro passaram a ser corretas, pois os erros não foram tão grandes como foram com 8 computadores, o que fez com que apenas uma das 4 estimativas que estavam erradas passasse a ser correta. O grande motivo desta melhora na qualidade das estimativas é que a quantidade aproximada de operações executadas pelo código é informada, permitindo assim estimar o tempo gasto para processar tais operações.

O gráfico da Fig. 5.5 ilustra os tempos de execução da versão $prim_A$ com os novos valores dos

Tab. 5.11: Resultados da aplicação de busca de números primos baseada na probabilidade de execução de um bloco condicional: versão $prim_A$

Busca de números primos: versão $prim_A$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ **(%)**	$\overline{E}_P$ **(%)**
2		849	[752;880]	0	0,40
		802		0	
		893		1,59	
		872		0	
4		428	[399;467]	0	0
		431		0	
		442		0	
		399		0	
8		245	[275;302]	-10,40	6,33
		238		-12,82	
		269		-2,08	
		287		0	
16		141	[134;157]	0	1,89
		168		7,56	
		144		0	
		152		0	
32		90	[81;92]	0	0,87
		83		0	
		95		3,47	
		87		0	

intervalos calculados em função da probabilidade de execução de um bloco condicional. Nota-se claramente que os limites de tempo de cada intervalo foram reduzidos em relação aos da Fig. 5.3, o que fez com que os tempos de execução, para todos os valores de P ficassem muito próximos das estimativas. Com 2 e 16 computadores, por exemplo, os 3 erros e 1 acerto passaram a ser de 3 acertos e apenas um erro. Com 4 e 8 computadores, todas as estimativas que estavam bastante distantes dos tempos de execução passaram a ficar muito próximas destes ou coincidentes. O mesmo ocorreu com 32 computadores, em que o intervalo ficou mais próximo ainda dos tempos de execução.

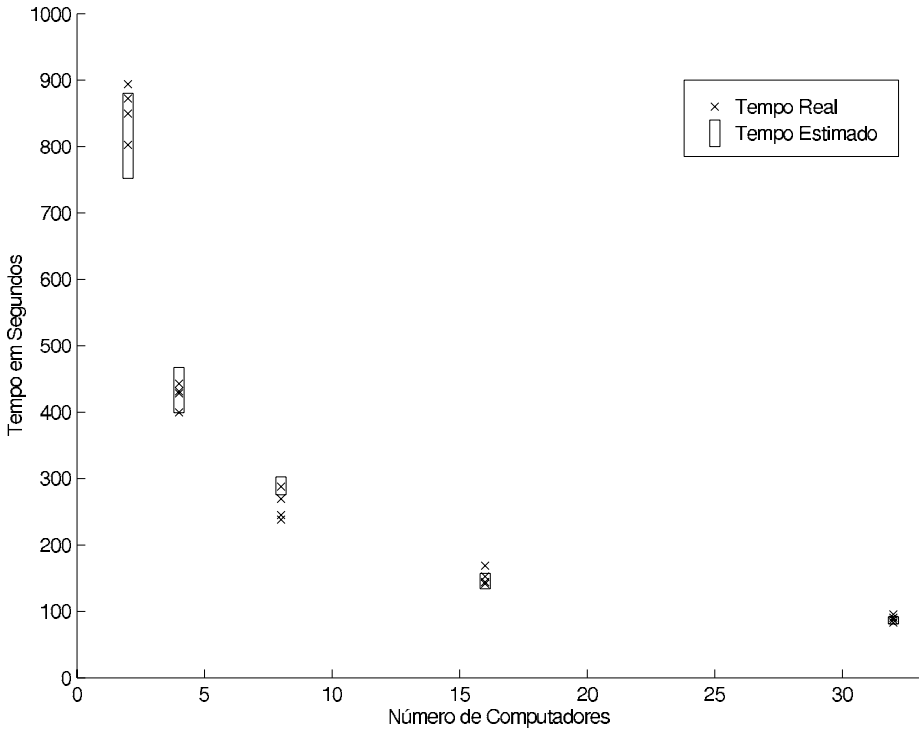

Fig. 5.5: Tempo para busca de números primos baseada na probabilidade de execução de um bloco condicional: versão $prim_A$

As outras 3 versões desta aplicação apresentaram resultados bastante similares a esta. Os resultados obtidos podem ser verificados no Apêndice C.3 deste trabalho.

Uma análise dos resultados fornecidos pelas 4 versões da aplicação demonstrou que tal alternativa permitiu melhorar bastante a qualidade das estimativas da aplicação, reduzindo muito os erros que foram obtidos sem o seu uso. Dos 80 tempos de execução, o acerto das estimativas aumentou de apenas 6 para 41. Isso representa uma melhora considerável na quantidade de acertos, que passou de 7,5% para 51,25%. Além disso, dos 39 tempos que ainda apresentam erros, praticamente todos tiveram uma redução bastante significativa no valor do erro, pois no cálculo das novas estimativas os intervalos aproximaram-se bastante dos tempos de execução.

O gráfico da Fig. 5.6 ilustra um comparativo da freqüência de acertos nas estimativas das 4 versões experimentadas. A maior freqüência de acertos aconteceu na versão $prim_A$, com 14 das

20 estimativas. A pior freqüência foi na versão $prim_D$, em que apenas 6 das 20 estimativas foram corretas. Nota-se que, para este caso, o aumento da quantidade de tarefas ou da carga computacional de cada uma delas fez com que a freqüência de acertos diminuisse.

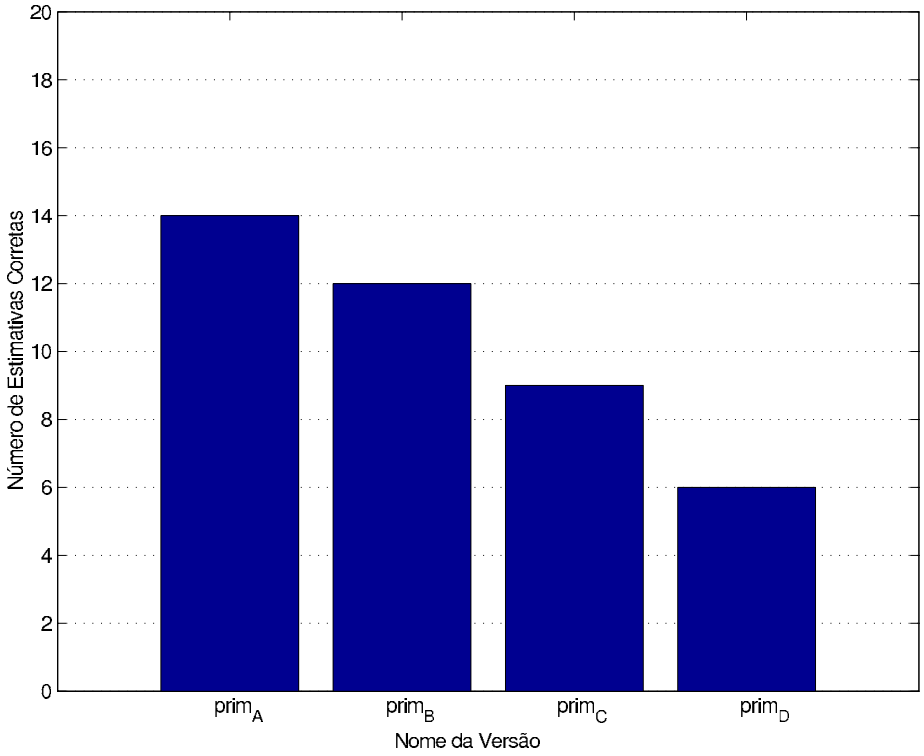

Fig. 5.6: Frequência de acertos de cada versão da aplicação do cálculo do número π baseada na probabilidade de execução de um bloco condicional

Na Tab. 5.12 são listados os erros médios de cada versão. Com exceção da versão $prim_D$ que apresentou picos de erros com 8 e 16 computadores o erro médio das demais versões não ultrapassou 3%.

Apesar desta forma de se calcular as estimativas de tempo fornecer melhores resultados para aplicações que possuem estruturas condicionais do tipo if, é necessário possuir informações detalhadas do algoritmo que será executado para que se possa estimar o percentual Q de vezes em que um bloco de uma estrutura condicional poderá ser executado.

5.4.4 Operações completas em matrizes

Esta aplicação, conforme apresentada na seção 5.2, caracteriza-se por ser determinista e possuir razão $r_{cc} > 1$. É esperado que, devido a esta última característica, que a qualidade das estimativas seja inferior com relação às aplicaçoes com $r_{cc} >> 1$, pois o sistema de predição foi desenvolvido para tratar de aplicações ditas ideais para JoiN.

Tab. 5.12: Erro médio de cada versão da aplicação de busca de números primos baseada na probabilidade de execução de um bloco condicional

Busca de números primos: erro médio de cada versão	
Nome da versão	**Erro médio ($\overline{E}$) (%)**
$prim_A$	1,898
$prim_B$	1,89
$prim_C$	2,956
$prim_D$	4,218

A seguir, será visto que, nos experimentos desta aplicação, os resultados foram, em geral, muito bons, mesmo quando a quantidade de dados processados pela aplicação foi relativamente grande se comparada à das aplicações de busca de números primos e à de cálculo do número π. A seguir, são apresentados e discutidos alguns resultados dos experimentos realizados.

Na versão mat_A, metade das 20 estimativas não apresentou erro. Apesar do determinismo do código, o tempo gasto para envio dos dados para processamento influi diretamente no tempo de execução da aplicação. Em função disso, a qualidade das estimativas torna-se diretamente dependente da valocidade da rede, a qual costuma sofrer grandes variações de desempenho. Os resultados desta versão são apresentados na Tab. 5.13. Entre os 10 erros encontrados, os dois maiores foram, na ordem, de $\overline{E}_{32,3} = 27,35\%$ e $E_{32,2} = 21,05\%$ quando foram utilizados 32 computadores, e correspondem à terceira e à segunda execução da aplicação, respectivamente. Em função disso, o maior erro médio para esta versão foi com 32 computadores, com $\overline{E}_{32} = 13,16\%$. Para os demais valores de P, nota-se que o erro permaneceu bastante inferior e, na grande maioria deles, entre 1% e 6%. Para $P = 16$, por exemplo, o erro médio foi de $\overline{E}_{16} = 4,49\%$ e não ultrapassou 1,5% com 2, 4 e 8 computadores. A menor média de erro, que foi de $\overline{E}_4 = 0,64\%$, ocorreu com $P = 4$, o qual foi o único da versão em que apenas um dos 4 tempos de execução apresentou erro.

É importante destacar que um provável motivo para o valor elevado do erro quando foram utilizados 32 computadores é alguma instabilidade ocorrida no sistema paralelo durante a execução da versão, fazendo com que o tempo gasto na execução fosse muito maior do que o esperado.

O gráfico da Fig. 5.7 ilustra os tempos de execução e as estimativas da versão mat_A. Apesar de haver erros em metade das estimativas realizadas, pode-se visualizar claramente que, para todos os valores de P, as estimativas estão sempre muito próximas do tempo real. Além disso, pode ser percebido que todos os tempos de execução que resultaram em algum erro ficaram acima do limite superior do intervalo de tempo estimado, mas muito próximos dele. Nota-se também que, em todas as estimativas, pelo menos um dos tempos de execução não acusou erro e que, com 4 computadores, ocorreu a maior frequência de acertos. Independentemente dos erros e acertos, o gráfico mostra que os resultados obtidos para esta versão foram satisfatórios.

As outras 3 versões desta aplicação apresentaram resultados bastante similares a esta. Tais resultados obtidos podem ser verificados no Apêndice C.4 deste trabalho.

Tab. 5.13: Resultados da aplicação que realiza operações completas em matrizes: versão mat_A

Operações completas em matrizes: versão mat_A					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}(\%)$	$\overline{E}_P$ **(%)**
2	1	602	[576;634]	0	1,03
	2	643		1,49	
	3	589		0	
	4	650		2,64	
4	1	311	[288;340]	0	0,64
	2	322		0	
	3	348		2,55	
	4	327		0	
8	1	161	[154;181]	0	1,49
	2	189		4,78	
	3	183		1,19	
	4	178		0	
16	1	89	[81;97]	0	4,49
	2	102		5,62	
	3	95		0	
	4	108		12,36	
32	1	48	[43;52]	0	13,16
	2	62		21,05	
	3	65		27,37	
	4	54		4,21	

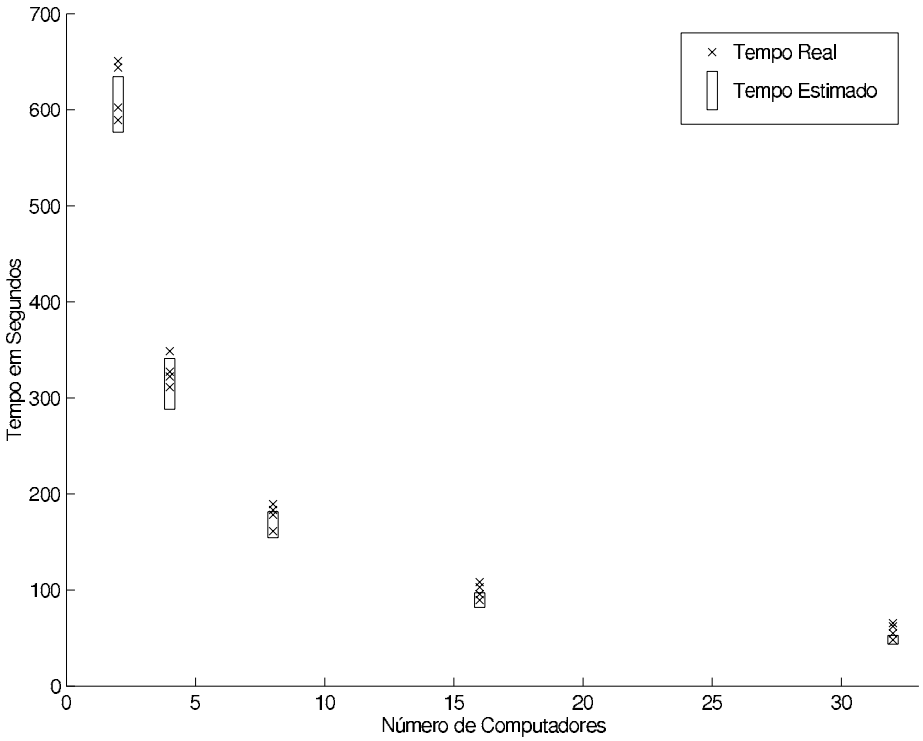

Fig. 5.7: Tempo para operações completas em matrizes: versão mat_A

Uma análise dos resultados gerados pelas 4 versões demonstram que, para a aplicação de operações completas em matrizes, o sistema de predição foi eficiente para o cálculo das estimativas. Das 80 estimativas, 30 não apresentaram erro. Conforme o gráfico de freqüência da Fig. 5.8, a versão mat_A foi a que apresentou a maior quantidade de acertos, com 10 ao todo, ao passo que a versão mat_D foi a que obteve menos sucesso em suas estimativas, com apenas 6 acertos. Nas outras duas versões, mat_B e mat_C, as estimativas acertaram 6 vezes. Por meio da figura, nota-se que a quantidade de acertos entre as 4 versões distribui-se quase igualmente entre elas. Isso garante que modificações na aplicação quanto ao número de tarefas e a carga de trabalho de cada uma, para os casos testados, não fazem com que as estimativas produzidas pelo sistema de predição apresentem alguma considerável melhora. No entanto, pode-se notar que, para esta aplicação, um aumento na quantidade de tarefas e da carga computacional para cada uma fez com que a quantidade de acertos diminuisse.

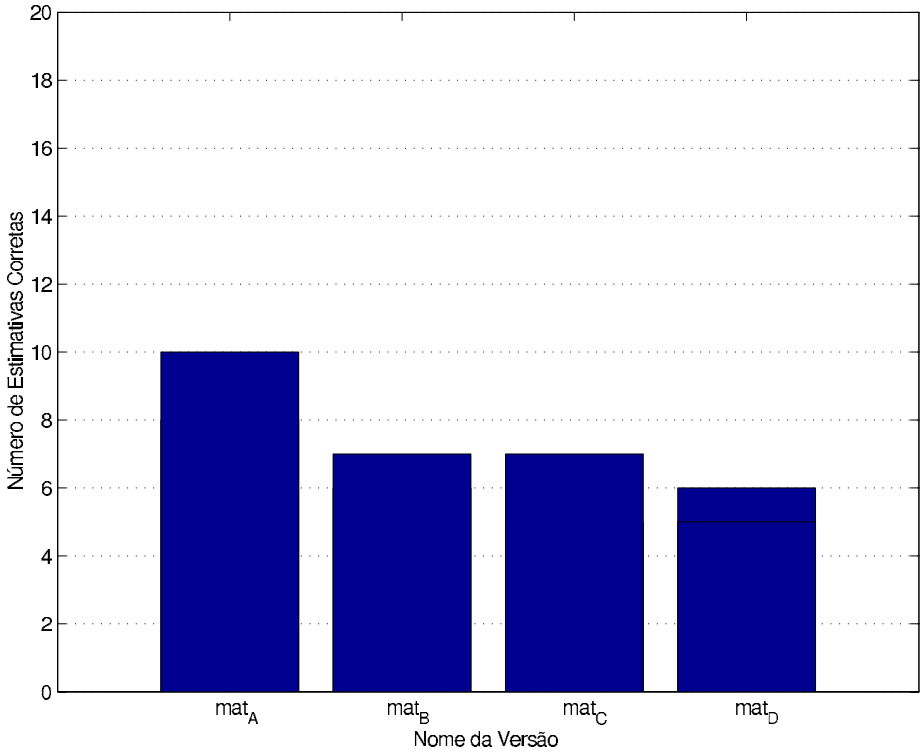

Fig. 5.8: Frequência de acertos de cada versão da aplicação que realiza operações completas em matrizes

Na Tab. 5.14 são apresentadas as médias de erro de cada uma das 4 versões da aplicação que realiza operações sobre matrizes. Pode-se verificar que os erros médios, em todos os casos, estão sempre abaixo de 4,2%, e que a diferença entre eles é reduzida.

Tab. 5.14: Erro médio de cada versão da aplicação que realiza operações completas em matrizes

Operações completas em matrizes: erro médio de cada versão	
Nome da versão	**Erro médio ($\overline{E}$) (%)**
mat_A	4,162
mat_B	3,672
mat_C	2,822
mat_D	2,394

5.4.5 Operações parciais em matrizes

Esta aplicação, conforme apresentada na seção 5.2, caracteriza-se por não ser determinista e possuir $r_{cc} > 1$. Em função destas características, já é de se esperar que o sistema de predição apresente resultados piores com relação às aplicações determinista e com razão $r_{cc} >> 1$.

As 4 versões experimentadas apresentaram resultados ruims, pois os tempos de execução em praticamente todos os casos estiveram fora e distantes dos intervalos de tempo estimados. A seguir, são apresentados os resultados e algumas análises dos experimentos feitos com cada uma das versões.

Os experimentos da versão $rand_A$, conforme dados da Tab. 5.15, apresentaram resultados ruins para quase todos os valores de P. Dos 20 tempos de execução, apenas uma não acusou erro, que foi na primeira execução utilizando 32 computadores. Para esta quantidade de computadores, também ocorreu o menor erro, que foi de $E_{32,2} = 3,6\%$. Ela é várias vezes menor do que o maior erro encontrado, que foi $E_{4,4} = 22,45\%$ com 4 computadores. Nota-se, na tabela, que a grande maioria dos erros, com exceção do caso com 32 computadores, apresentaram valores maiores do que 10%, o que é muito ruim para a qualidade das estimativas. Em função disso, a média de erro para 2, 4, 8 e 16 computadores variou de 9,77% e 20,32%. A exceção ocorreu com 32 computadores, em que a média de erro foi de $\overline{E}_{32} = 4,05\%$. Uma causa provável destes erros é a presença de contenção nos computadores do sistema paralelo, os quais fizeram com que o tempo de execução da versão sofresse um aumento considerável. Além disso, a utilização da rede para envio de dados às tarefas pode ter influenciado nos tempos de execução com relação às estimativas calculadas, pois o tipo de rede utilizado pode facilmente sofrer modificações de desempenho.

No gráfico da Fig. 5.9 observa-se claramente que os tempos de execução e as estimativas estão distantes entre si para os casos em que foram utilizados 2 e 4 computadores. Com 8 e 16 computadores, nota-se que os tempos de execução estiveram próximos dos intervalos, porém todos possuem tempos de execução inferiores ao do intervalo. Com 32 computadores ocorreu o inverso do que foi verificado com 2 e 4 computadores, pois os tempos, para este caso, estiveram muito próximos da estimativa.

As outras 3 versões desta aplicação apresentaram resultados bastante similares a esta. Tais resultados obtidos podem ser verificados no Apêndice C.5 deste trabalho.

Tab. 5.15: Resultados da aplicação que realiza operações parciais em matrizes: versão $rand_A$

Operações parciais em matrizes: versão $rand_A$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}(\%)$	$\overline{E}_P$ (%)
2	1	652	[734;780]	-10,83	16,25
	2	587		-19,42	
	3	612		-16,12	
	4	593		-18,63	
4	1	321	[380;395]	-15,23	20,32
	2	304		-19,61	
	3	287		-24,00	
	4	293		-22,45	
8	1	186	[188;201]	-1,03	9,77
	2	167		-10,80	
	3	164		-12,34	
	4	159		-14,91	
16	1	95	[104;108]	-8,49	12,03
	2	91		-12,26	
	3	92		-11,32	
	4	87		-16,04	
32	1	57	[54;57]	0	4,05
	2	52		-3,60	
	3	52		-3,60	
	4	49		-9,01	

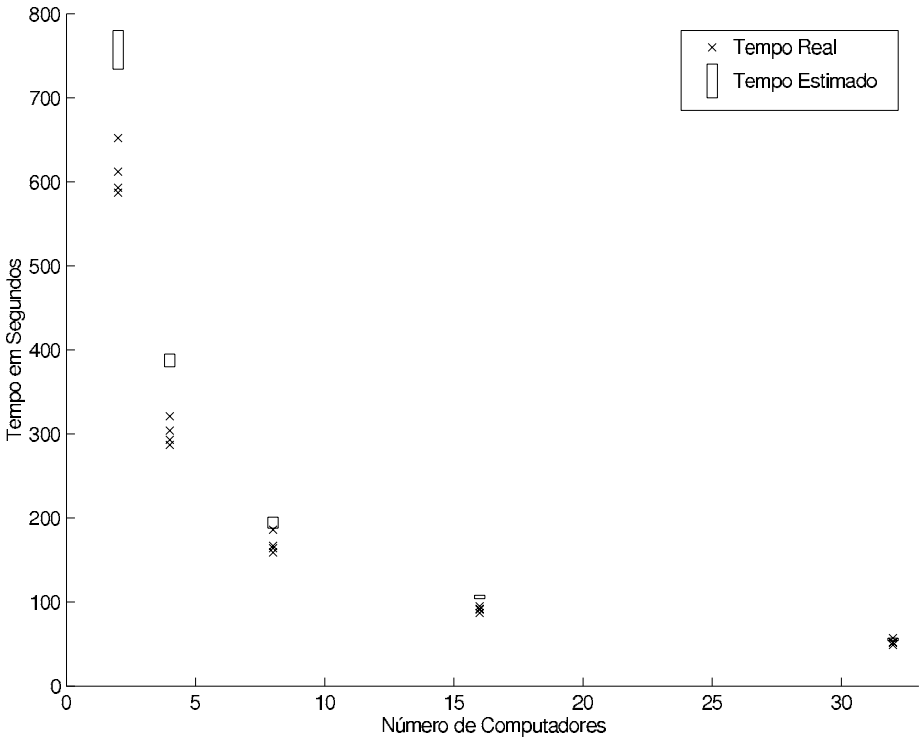

Fig. 5.9: Tempo para operações parciais em matrizes: versão $rand_A$

Uma análise dos resultados das 4 versões da aplicação que realiza operações parciais em matrizes mostrou que os resultados apresentaram erros bastante elevados e todos ficaram abaixo do limite inferior do intervalo de tempo estimado. Dos 80 tempos de execução, os 2 únicos acertos ocorreram nas versões $rand_A$ e $rand_B$, ambos quando foram utilizados 32 computadores. A freqüência de acertos de cada versão é ilustrada no gráfico da Fig. 5.10.

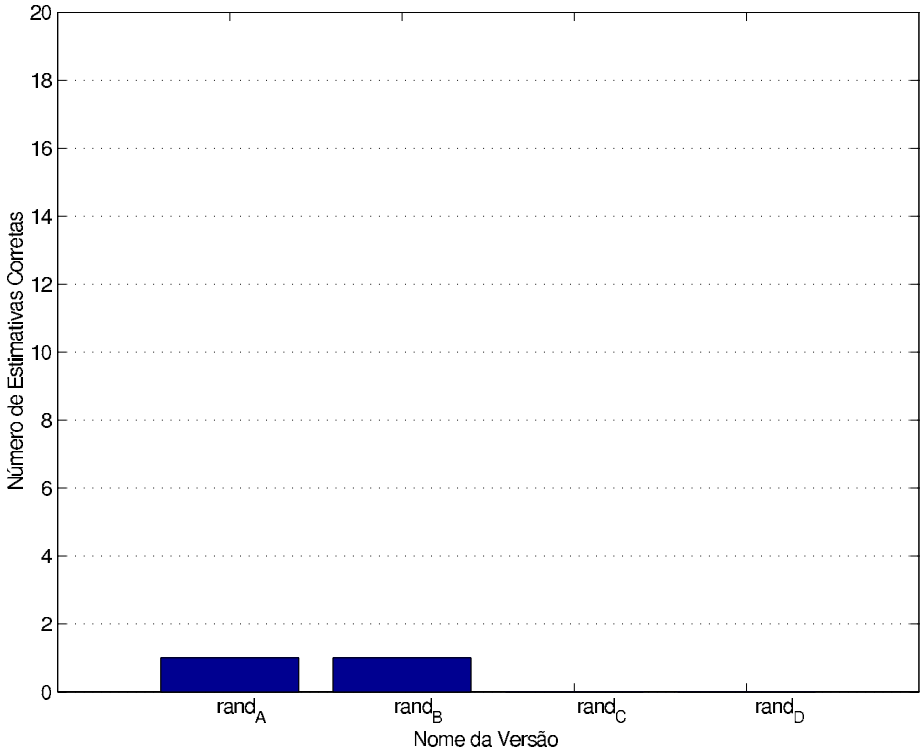

Fig. 5.10: Frequência de acertos de cada versão da aplicação que realiza operações parciais em matrizes

Em função dos elevados erros apresentados pela versão $rand_D$, conforme dados apresentados na Tab. 5.16, esta versão foi a que apresentou o maior erro médio em relação às outras, e foi de $\overline{E} = 21,426\%$. Pressupõe-se que os elevados erros apresentados pela versão $rand_D$ tenham sido causados pela contenção externa, pois não há outros motivos que comprovem algum indício que justifique os erros elevados.

Tab. 5.16: Erro médio de cada versão da aplicação que realiza operações parciais em matrizes

Operações parciais em matrizes: erro médio de cada versão	
Nome da Versão	**Erro Médio ($\overline{E}$) (%)**
$rand_A$	12,484
$rand_B$	13,53
$rand_C$	14,472
$rand_D$	21,426

De um modo geral, apesar dos erros das estimativas serem elevados, o sistema de predição foi capaz de gerar estimativas que pudessem acompanhar o aumento dos tempos de execução da aplicação, porém mantendo um erro que não sofreu grandes modificações para cada valor de P. Apesar disso, os resultados para esta aplicação não foram satisfatórios porque o erro médio das versões, conforme a Tab. 5.16, em nenhum dos casos apresentados, foi inferior a 12%.

Além disso, uma característica importante de tal aplicação é que boa parte da carga computacional dela faz parte de um bloco de execução de uma estrutura condicional, a qual seleciona apenas uma parte dos elementos das matrizes para realizar operações sobre eles. Desta forma, uma alternativa para melhorar as estimativas e diminuir o erro, é identificar a probabilidade aproximada que a estrutura condicional é executada e, então, realizar as mesmas considerações feitas na aplicação de busca de números primos, em que é possível identificar a quantidade aproximada de números que são primos de um determinado intervalo numérico.

5.5 Considerações

Os resultados preliminares apresentados neste capítulo demonstraram que o sistema de predição proposto gerou estimativas precisas quando o tipo de aplicação verificado foi determinista. As aplicações de cálculo do número π e operações completas sobre matrizes, que são do tipo determinista, foram analisadas pelo sistema de predição. A maioria dos tempos de execução permaneceu bastante próximo de ou entre os limites dos intervalos estimados. Os erros médios, quando existiram, foram sempre baixos e em nenhum dos casos verificados ultrapassaram 4,2 %.

Em contrapartida, aplicações que se caracterizam por ter um algoritmo não determinista apresentam estimativas de tempo ruins devido à incerteza sobre a quantidade de operações executadas pelo código da aplicação. As duas aplicações que se caracterizam desta forma, busca de números primos e operações parciais em matrizes, apresentaram erros médios bastante elevados. O menor erro, que foi de aproximadamente 8 % na aplicação de busca de números primos, já é considerável. O maior erro, que foi próximo de 22 % para a aplicação de operações parciais em matrizes, é inaceitável por ser muito elevado. Foi discutido que uma alternativa que pode ser empregada para tentar reduzir o erro das estimativas é informar ao sistema de predição uma quantidade aproximada de vezes que as estruturas condicionais do código seriam executadas. Isso foi demonstrado na aplicação de busca

de números primos, na qual calculou-se a quantidade aproximada de vezes que as estruturas condicionais são executadas. Os resultados mostraram que, o erro da predição, que estava entre 8 e 16%, sofreu grande redução e passou a ficar entre 1 e 4% aproximadamente. No entanto, tal abordagem requer que se conheça em detalhes as características do algoritmo para que seja possível identificar a probabilidade de execução das estruturas condicionais presentes no código.

Os resultados preliminares apresentados permitiram antever a utilidade do sistema de predição para alguns tipos de aplicações que possuem as mesmas características daquelas experimentadas neste trabalho.

Além disso, foi possível verificar que modificações na quantidade de computadores ou na estrutura de uma aplicação, como a quantidade de tarefas ou a quantidade de dados processados por ela, não afetam de maneira aparente a qualidade das estimativas do sistema de predição. Esta é uma característica fundamental de sistemas de predição de desempenho, como os apresentados no Cap. 2, e não poderia estar ausente no sistema proposto neste trabalho. Entretanto, é importante ressaltar que as aplicações utilizadas nos experimentos são exemplos genéricos mas não esgotam todas as possibilidades de experimentos.

Capítulo 6

Conclusão e trabalhos futuros

6.1 Conclusão

Este trabalho abordou o desenvolvimento de um sistema de modelagem e predição analítica de desempenho, chamado PeMPSys (Performance Modeling and Prediction System) para a plataforma paralela JoiN. Foi visto que, conforme apresentado no Cap. 1, sistemas de predição de desempenho são ferramentas muito importantes para estimar o tempo de execução das aplicações em uma plataforma de processamento paralelo. A ausência de tal sistema em uma plataforma paralela pode fazer com que a utilização dos recursos disponíveis na plataforma seja ruim. Isso pode ocorrer devido ao fato de que algumas aplicações podem demorar um tempo muito maior do que o máximo exigido para a obtenção de sua solução.

Para a obtenção das estimativas de tempo por meio de um sistema de predição é necessário o uso de alguma técnica eficiente que permita estimar o tempo de execução de alguma aplicação sem que seja necessário realizar modificações na estrutura da aplicação e da plataforma paralela. Além disso, é necessária uma técnica que forneça estimativas em pouco tempo, pois ferramentas que demoram em demasia podem desmotivar seu uso.

Em função disso, o Cap. 2 abordou dois tipos de técnicas de predição que podem ser utilizadas para a geração de estimativas de tempo: as dinâmicas e as estáticas (ou analíticas). Técnicas dinâmicas, apesar de fornecerem estimativas com maior precisão em relação às analíticas, são baseadas em simulação, exigem o uso efetivo dos recursos disponíveis na plataforma de processamento paralelo e o tempo gasto para a obtenção das estimativas, dependendo do caso, pode ser elevado. Foi visto que técnicas analíticas, apesar da precisão inferior em relação às técnicas dinâmicas, são uma boa alternativa para realizar estimativas de tempo, pois não é necessário o uso efetivo dos recursos da plataforma paralela e o tempo gasto para a obtenção das estimativas é baixo.

Algumas ferramentas que implementam tais técnicas baseiam-se na criação de dois modelos: de aplicação e de máquinas, para representar as características da aplicação e dos computadores que farão o seu processamento, respectivamente. Os problemas envolvidos na geração do modelo de

aplicação são a necessidade de inserção de marcadores explícitos no código-fonte da mesma e o uso de linguagens de programação especiais. Ferramentas como PACE e PAMELA exigem tais marcadores. As aplicações para PACE podem ser escritas em Fortran (77 ou 90) ou C. No caso de PAMELA propõe-se a criação de uma nova linguagem de programação, chamada Spar, que procura modelar efeitos de atraso causados pela contenção interna no uso de recursos. A ferramenta PerPreT também propõe o uso de uma nova linguagem de programação, chamada LOOP, que limita as aplicações àquelas descritas pelo modelo de programação SPMD. Porém, PerPreT não exige marcação explícita no código.

O processo de geração do modelo de máquinas para cada ferramenta mencionada é baseado na execução de algoritmos de *benchmark* (algoritmos de testes) no sistema paralelo, com o intuito de avaliar o desempenho da rede e dos computadores.

Em função do custo envolvido na geração de predições e da necessidade de ferramentas automatizadas que minimizem os esforços para realizar predições, este trabalho propôs um novo sistema de predição utilizando uma abordagem de predição analítica. O sistema de predição, batizado de PeMPSys, faz uso da técnica de predição PAMELA Estendida, descrita no Cap. 3, para a criação de 3 modelos (máquinas, aplicação e analítico de desempenho), que combinados irão gerar as estimativas de tempo de execução.

A criação do modelo de máquinas é feita pela ferramenta CRES (Computational Resource Evaluation Service). Ela baseia-se na execução de *benchmarks* nos computadores da plataforma paralela e utiliza intervalos de valores para modelar os atrasos.

Para a criação do modelo de aplicação, foi desenvolvida a ferramenta ARC (Analytical Representation Compiler), composta por três partes: PASC, SCAn e AMGen. A primeira é um compilador para analisar o código PASL que define a especificação do fluxo de execução da aplicação paralela, a segunda é um compilador que analisa o código Java executado pelas tarefas da aplicação e a terceira utiliza os resultados das análises feitas pelas duas anteriores para gerar um modelo de aplicação.

Para a geração das estimativas de tempo, foi desenvolvida a ferramenta APET (Analytical Performance Estimation Tool), composta pelas ferramentas AMC e PTT. Em uma primeira fase, AMC (Analytical Model Compiler) combina o modelo de máquinas com o de aplicação e gera um modelo analítico que é composto por expressões matemáticas parametrizadas pela quantidade e desempenho dos computadores que serão utilizados para executar a aplicação. A partir do modelo analítico, TPT (Time Prediction Tool) avalia-o para obter na forma de um intervalo de valores uma estimativa de tempo da aplicação analisada.

A vantagem do sistema de predição PeMPSys com relação a outras ferramentas apresentadas na literatura é que, para a geração do modelo da aplicação com o uso de ARC, ela não requer marcação no código-fonte da aplicação, nem o uso de uma linguagem de programação não padronizada e nem modificações na estrutura da plataforma de processamento paralelo considerada. Para a geração das estimativas a ferramenta APET considera efeitos de atraso causados por contenção interna e por contenção externa. Além disso, operações aritméticas sob diferentes tipos de dados são tratadas de forma diferenciada.

É importante destacar que a natureza analítica dos modelos permite que eles sejam modificados e as estimativas podem ser realizadas sobre diferentes condições.

Os experimentos realizados com o uso do sistema de predição PeMPSys, conforme Cap. 5, demonstraram que, dependendo das características do algoritmo da aplicação, os resultados podem variar. Aplicações como a de cálculo do número π e de operações completas em matrizes apresentaram resultados satisfatórios, pois é possível identificar a quantidade aproximada de operações que serão executadas pelo algoritmo e, conseqüentemente, o tempo estimado que a aplicação levará para que seja executada.

Entretanto, para as aplicações como a de busca de números primos e a de operações parciais em matrizes, os resultados demonstraram que os tempos de execução podem ficar fora dos intervalos de tempo estimado, pois em tais aplicações não é possível determinar a quantidade aproximada de operações que serão executadas devido à presença de estruturas condicionais no código. Uma alternativa que permite melhorar a qualidade das estimativas quando há estruturas condicionais no código é, conforme demonstrado com a aplicação de busca de números primos, informar uma taxa indicando o percentual de vezes que o bloco da estrutura condicional será executado. Um problema de tal alternativa é que ela exige um profundo conhecimento da forma com que o algoritmo da aplicação é executado para que seja possível determinar o percentual de vezes que as estruturas condicionais da aplicação são executadas. Sem este percentual, o sistema de predição se comporta de maneira conservadora, oferencedo estimativas de tempo de pior caso.

De forma geral, o desenvolvimento deste trabalho permitiu atingir alguns objetivos, os quais são listados a seguir de forma resumida:

- verificar a qualidade das estimativas geradas pela técnica PAMELA Estendida quando implementada por uma ferramenta;
- obter experiência com o desenvolvimento de sistemas de predição de desempenho;
- conhecer de forma detalhada a implementação e o funcionamento de uma plataforma de processamento paralelo para desenvolver um sistema de predição;
- estudar tecnologias relacionadas à predição de desempenho utilizadas em outras plataformas de processamento paralelo;
- propor um sistema de predição para a plataforma de processamento paralelo JoiN.

6.2 Trabalhos futuros

Este trabalho não tem a pretensão de propor um sistema definitivo para o problema de geração de estimativas de tempo para plataformas de processamento paralelo. No entanto, podem ser propostas melhorias visando a qualidade das estimativas, assim como novas propostas podem surgir baseadas nos conceitos apresentados e nas ferramentas que foram desenvolvidas.

Dentre os principais pontos que podem ser desenvolvidos futuramente, destacam-se os seguintes:

1. testes com algoritmos de benchmark que retratem melhor as características dos computadores com relação à execução de operações de computação e de comunicação;
2. extensão dos algoritmos de benchmark para coletar informações de desempenho dos computadores para operações não-básicas, tais como criação de objetos, chamada de métodos, chamada de métodos de bibliotecas matemáticas da API Java, entre outros;
3. estudo da utilização do sistema de predição com um sistema de escalonamento de tarefas para auxiliar na estratégia de equilíbrio de carga entre os computadores utilizados para o processamento das tarefas de uma aplicação;
4. estudo de utilização do compilador SCAn para outras finalidades além da geração de parte do modelo da aplicação, tais como otimização de código, análise estatística dos tipos de operações predominantes no código, entre outros;
5. realização de experimentos com quantidades maiores de computadores e com aplicações que demandam grande quantidade de dados para processamento para se obter informações mais abrangentes sobre o comportamento do sistema proposto;
6. estudo da possibilidade de adaptação do sistema de predição em outras plataformas paralelas que apresentem características similares às de JoiN.

Bibliografi a

[AALT95] S. P. Amarasinghe, J. M. Anderson, M. S. Lam, and C. W. Tseng. The suif compiler for scalable parallel machines. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, California, USA, fevereiro, 1995.

[ACF+01] W. Allcock, A. Chervenak, I. Foster, L. Pearlman, V. Welch, and M. Wilde. Globus toolkit support for distributed data-intensive science. In *Proceedings of Computing in High Energy Physics (CHEP '01)*, Beijing, China, setembro, 2001.

[ACK+01] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home-massively distributed computing for seti. In *Computing in Science and Engineering*, pages 78–83. IEEE Computational Science and Engineering, 2001.

[ACK+02] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. In *Communications of the ACM*, pages 56–61, novembro, 2002.

[And04a] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, novembro, 2004.

[And04b] D. P. Anderson. Public computing: Reconnecting people to science. In *Conference on Shared Knowledge and the Web*, Madrid, Spain, novembro, 2004.

[AR01] Matthew Arnold and Barbara G. Ryder. A framework for reducing the cost of instrumented code. In *PLDI '01: Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, pages 168–179, Snowbird, Utah, United States, junho, 2001.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, Principles, Techniques, and Tools*. Addison Wesley, 1986. ISBN 0-201-10088-6.

[BB00] R. Buyya and M. A. Baker. Grid computing. In *Grid 2000: First IEEE/ACM International Workshop (Lecture Notes in Computer Science)*, Bangalore, India, dezembro, 2000.

[BBL02] M. A. Baker, R. Buyya, and D. Laforenza. The grid: International efforts in global computing. In *International Journal of Software Practice and Experience*, julho, 2002.

[BDCW92] Eric A. Brewer, Chrysanthos N. Dellarocas, Adrian Colbrook, and William E. Weihl. Proteus: a high-performance parallel-architecture simulator. In *SIGMETRICS '92/PERFORMANCE '92: Proceedings of the 1992 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 247–248, New York, NY, USA, 1992. ACM Press. http://doi.acm.org/10.1145/133057.133146. Último acesso: 10/2005.

[Bec77] P. Beckmann. *A History of Pi*. Golem Press, 1977.

[BMSD95] J. Brém, M. Madhukar, E. Smirni, and L. Dowdy. Perpret - a performance prediction tool for massively parallel systems. In *Proceedings of the Joint Conference Performance Tools / MMB*, Heidelberg, Germany, setembro, 1995.

[CFA+04] L. B. Costa, L. Feitosa, E. Araújo, G. Mendes, R. Coelho, W. Cirne, and D. Fireman. Mygrid: A complete solution for running bag-of-tasks applications. In *Anais do SImpósio Brasileiro de Redes de Computadores (SBRC) 2004 - Salão de Ferramentas*, maio, 2004.

[Com05] Using Distributed Computing. Folding@home and genome@home. Página na internet, 2005. citeseer.lcs.mit.edu/589744.html.Último acesso: 11/2005.

[CP01] R. Crandall and C. Pomerance. *Prime numbers: a computational perspective*. Springer-Verlag, New York, 2001.

[CPC+03] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauv, F. Alves Barbosa da Silva, C. Osthoff Barros, and C. Silveira. Running bag-of-tasks applications on computational grids: The mygrid approach. In *Proceedings of the ICCP'2003 - International Conference on Parallel Processing*, outubro, 2003.

[Die99] K. Diefendorff. Pentium iii = pentium ii + sse. In *Microprocessor Report*, 1999.

[DJSW96] T. Delaitre, G. Justo, F. Spies, and S. Winter. Simulation modelling of parallel systems. In *DAPSYS'96: 1st Austrian-Hungarian Workshop on Distributed and Parallel Systems (Miskolc, Hungary, 1996)*, pages 25–32, outubro, 1996. citeseer.ist.psu.edu/delaitre96simulation.html. Último acesso: 10/2005.

[ea97] O. Lempel et al. Intel's mmx technology - a new instruction set extension. In *Proceedings of COMPCON*, 1997.

[Fah95] T. Fahringer. Estimating and optimising performance from parallel programs. In *special issue IEEE Computer 28*, 1995.

[FGA+02] I. Foster, Argonne National Laboratory Group, E. Alpert, A. Chervenak, B. Drach, C. Kesselman, V. Nefedova, D. Middleton, A. Shoshani, A. Sims, and D. Williams. The earth system grid ii: Turning climate datasets into community resources. In *18th International Conference on Interactive Information and Processing Systems (IIPS)*, Orlando, USA, 2002.

[FJ95] A. Ferscha and J. Johnson. N-map: a virtual processor discrete event simulation tool for performance prediction in the capse environment. In *28th Hawaii International Conference on System Sciences (HICSS'95)*, Hawaii, USA, janeiro, 1995.

[FK99] I. Foster and K. Kesselman. *Computational Grids*. Morgan-Kaufmann Publishers, 1999.

[FK04] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann Publishers, 2004.

[FMM00] M. C. Ferris, M. P. Mesnier, and J. J. Moré. Neos and condor: solving optimization problems over the internet. In *ACM Trans. Math. Softw.*, volume 26, pages 1–18. ACM Press, 2000. http://doi.acm.org/10.1145/347837.347842. Último acesso: 10/2005.

[Gau04] H. Gautama. *A Probabilistic Approach to Symbolic Performance Modeling of Parallel Systems*. PhD thesis, dezembro, 2004.

[GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, 1st edition, 1996.

[GJSB05] J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification*. Addison-Wesley, 3rd edition, 2005.

[Gro05] SETI@home Developer Group. Seti@home. Página na internet, 2005. http://setiathome.ssl.berkeley.edu. Último acesso: 06/2005.

[GWF+94] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P.F. Reynolds. Legion: The next logical step toward a nationwide virtual computer. Relatório Técnico CS-94-21, Department of Computer Science, University of Virginia, USA, 1994.

[GWT97] A. S. G., Wm. A. Wulf, and Legion Team. The legion vision of a worldwide virtual computer. In *Communications of ACM*, volume 40, pages 39–45. ACM Press, 1997. http://doi.acm.org/10.1145/242857.242867. Último acesso: 10/2005.

[Hen99] M. A. A. Henriques. A proposal for java based massively parallel processing on the web. In *Proceedings of The First Annual Workshop on Java for High-Performance Computing*, Rhodes, Greece, junho, 1999.

[HMG+97] J. K. Hollingsworth, B. P. M., M. J. R. Goncalves, Oscar Naim, Zhichen Xu, and L. Zheng. MDL: A language and compiler for dynamic program instrumentation. In *IEEE PACT*, 1997. citeseer.ist.psu.edu/97288.html. Último acesso: 10/2005.

[jav05] The benchmark suite. Página na internet, 2005. http://www.epcc.ed.ac.uk/javagrande/javag.html. Último acesso: 06/2005.

[JSKN03] S. A. Jarvis, D. P. Spooner, L. C. Keung, and G. R. Nudd. Performance prediction and its use in parallel and distributed computing systems. In *17th IEEE International Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice, France, abril, 2003.

[KRR04] M. Kühnemann, T. Rauber, and G. Rünger. A source code analyzer for performance prediction. In *Proc. of the Fourth Workshop on Massively Parallel Processing at the IPDPS04*, Santa Fe, New Mexico, USA, abril, 2004.

[leg05] Legion. Página na internet, 2005. http://www.cs.virginia.edu/ legion. Último acesso: 06/2005.

[LH99] F. O. Lucchese and M. A. A. Henriques. Aplicação de um computador massivamente paralelo virtual no seqüenciamento de cadeias de dna. Relatório Técnico DCA-RT 02/99, Departamento de Engenharia de Computação e de Automação Industrial, Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas - Unicamp, Campinas, São Paulo, Brasil, 1999.

[LLM88] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[Luc02] F. O. Lucchese. Um mecanismo para distribuição de carga em ambientes virtuais de computação maciçamente paralela. Dissertação de mestrado, Departamento de Engenharia de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas - Unicamp, Campinas, São Paulo, Brasil, 2002.

[LY99] T. Lindholm and F. Yellin. *The Java(TM) Virtual Machine Specification (2nd Edition)*. Sun Microsystems Inc., 1999.

[MC95] Wagner Meira Jr. Mark Crovella, Thomas J. LeBlanc. Parallel performance prediction using the lost cycles toolkit. Relatório Técnico TR 580, Computer Science Department, University of Rochester, USA, maio 1995. `ftp://ftp.cs.rochester.edu/pub/papers/`. Último acesso: 10/2005.

[MCC+95] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. In *IEEE Computer*, volume 28, pages 37–46, 1995. citeseer.ist.psu.edu/miller95paradyn.html. Último acesso: 10/2005.

[MHHK99] G. Mahinthakumar, Forrest M. Hoffman, William W. Hargrove, and N. T. Karonis. Multivariate geographic clustering in a metacomputing environment using globus. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, page 5, Portland, Oregon, United States, novembro, 1999. ACM Press. http://doi.acm.org/10.1145/331532.331537. Último acesso: 10/2005.

[Mul93] S. Mullender. *Distributed Systems*. ADDISON-WESLEY, 1993.

[OBV+02] C. Osthoff, P. Barros, C. Veronez, F. Agostin, W. Cirne, E. L. Santos-Neto, L. B. Costa, F. Silva, P. Pascutti, and P. Bisch e A. Silva. Utilização do software mygrid para adaptar uma aplicação de dinâmica molecular em um grid. Relatório Técnico LNCC-55-2002, Laboratório Nacional de Computação Científica - LNCC, Brasil, 2002.

[Pab00] Thomas Pabst. Intel's new pentium 4 processor. Página na internet, 2000. http://www.tomshardware.com/cpu/20001120/p4-03.html. Último acesso: 07/2005.

[PCD+03] L. Pouchard, L. Cinquini, B. Drach, , and the ESG team. Exploring ontologies in earth system grid (esg). In *Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, maio, 2003.

[PSS05] F. Völkel P. Schmid, B. Töpelt and U. Scheffel. Dothan over netburst: Is the pentium 4 a dead end? Página na internet, 2005. http://www.hardfeed.com/srv/ramp/article/1546.html. Último acesso: 07/2005.

[PvZH03] O. Prado, F. J. von Zuben, and M. A. A. Henriques. Ptp and join as software packages for phylogenetic inference. In *1st International Conference on Bioinformatics and Computational Biology*, Ribeirão Preto, SP, Brasil, maio, 2003.

[Sar95] T. Saridakis. Arraytracer: A parallel performance analysis tool. Relatório Técnico TR95-0136, European Research Consortium for Informatics and Mathematics, 1995. citeseer.ist.psu.edu/saridakis96arraytracer.html. Último acesso: 10/2005.

[SB97] J. M. Schopf and F. Berman. Performance prediction using intervals. Relatório Técnico CS-97-541, University of California, San Diego, USA, 1997.

[SBO01] L. A. Smith, J. M. Bull, and J. Obdrzálek. A parallel java grande benchmark suite. In *Proceedings of the ACM/IEEE conference on Supercomputing*, Denver, Colorado, USA, novembro, 2001.

[SC96] M. R. Steed and M. J. Clement. Performance prediction of pvm programs. In *Proceedings of the 10th International Parallel Processing Symposium*, Honolulu, HI, abril, 1996.

[Sch98] J. M. Schopf. *Performance Prediction and Scheduling for Parallel Applications on Multi-User Clusters*. Tese de doutorado, University of California, San Diego, USA, 1998.

[SCW+02] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. A framework for performance modeling and prediction. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 16–22, Baltimore, Maryland, USA, novembro, 2002.

[TTL02] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley and Sons Inc., dezembro 2002.

[vG96] A. J. C. van Gemund. *Performance Modeling of Parallel Systems*. Tese de doutorado, Department of Information Technology and Systems, Delft University of Technology, Netherlands, Sweden, 1996.

[vG01] A. J. C. van Gemund. Automatic cost estimation of data parallel programs. Relatório Técnico 1-68340-44(2001)09, Faculty of Information Technology and Systems, Delft University of Technology, Netherlands, Sweden, 2001.

[vG03] A. J. C. van Gemund. Symbolic performance modeling of parallel systems. In *IEEE Transactions on Parallel and Distributed Systems*, volume 14, pages 154–165, Heidelberg, Germany, Fevereiro 2003.

[vR00] C. van Reeuwijk. Vnus language specification version 2.1. Relatório Técnico PDS-2000-002, Faculty of Information Technology and Systems, Delft University of Technology, Netherlands, Sweden, 2000.

[vR01] C. van Reeuwijk. Spar language specification. Relatório Técnico PDS-2001-003, Faculty of Information Technology and Systems, Delft University of Technology, Netherlands, Sweden, 2001.

[vRKS01] C. van Reeuwijk, F. Kuijlman, and H. J. Sips. Spar: an extension of java for scientific computation. In *ACM Java Grande - ISCOPE Conference*, California, USA, junho, 2001.

[Yer98] E. J. Huerta Yero. Um sistema para o processamento maciçamente paralelo na world wide web. Dissertação de mestrado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas - Unicamp, Campinas-SP, Brasil, 1998.

[Yer03] E. J. Huerta Yero. *Estudo sobre Processamento Maciçamente Paralelo na Internet*. Tese de doutorado, Departamento de Engenharia de Computação e de Automação Industrial, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas - Unicamp, Campinas, São Paulo, Brasil, ago/2003.

[ZKK04] G. Zheng, G. Kakulapati, and Laxmikant V. Kalé. Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In *18th International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, New Mexico, abril, 2004.

Apêndice A

Um exemplo detalhado de aplicação: cálculo do número π

Esta seção lista de forma simplificada dois modelos gerados a partir da análise da estrutura do fluxo de execução de uma aplicação em PASL e do código Java executado pelas tarefas. Primeiro será listado o código PASL da aplicação, em seguida o código Java executado pelas tarefas, em seguida o modelo da aplicação e, no final, o modelo analítico resultante em função deste último.

A.1 Especificação do fluxo de execução na linguagem PASL

A listagem do arquivo $monteCarlo.pas$ possui a especificação do fluxo de execução da aplicação do cálculo do número π. Na seção de declarações é feito o mapeamento do código executado pelas tarefas de cada lote, enquanto que na seção de ligações é feito a indicação do fluxo dos dados entre os lotes de tarefas da aplicação. Uma explicação detalhada do código PASL foi apresentada na seção 3.3.2. No código a seguir, tem-se que $T1$, que compõe o lote $B1$ com cardinalidade 1, não recebe dados de entrada para processamento. A tarefa $T2$, que compõe o lote $B2$ com cardinalidade 100, lê a saída produzida pelo processamento de $T1$. Finalmente a tarefa $T3$, que faz parte do lote $B3$ com cardinalidade 1, lê as saídas de $T2$ para produzir o resultado final da aplicação.

```
// secao de declaracoes
path =  "PIMonteCarlo";
B1 =    "DistributeData.class";
B2 =    "ProcessData.class";
B3 =    "GatherResults.class";

%%
// secao de ligacoes
T1 = B1(1) << NULL;
T2 = B2(100) << T1;
T3 = B3(1) << T2;
```

O código executado pelas tarefas dos lotes B1, B2 e B3 são DistributeData.class, ProcessData.class e GatherResults.class, respectivamente. O código de tais lotes serão listados nas seções seguintes.

A.2 Código-fonte Java da aplicação

A.2.1 Código da classe DistributeData

A classe DistributeData é a responsável pela distribuição da quantidade de pontos que será verificada entre as tarefas do lote seguinte. É importante destacar que este lote, indicado por B_0 e cuja cardinalidade é 1, produz dados iguais para todas as tarefas do lote seguinte. A indicação dos dados que serão enviados para cada tarefa deve estar indicada diretamente no código, sem que haja necessidade de haver algum tipo de pré-processamento. O uso de estruturas de repetição para preparação dos dados, por exemplo, deve ser feito de forma que tais dados sejam diretamente atribuídos a algum tipo de arranjo, como um vetor. O código da classe DistributeData listado abaixo foi desenvolvido desta forma. Em função, é deixada como trabalho futuro a extensão do compilador SCAn para que seja capaz de identificar os dados que serão enviados às tarefas, mesmo que para a sua organização seja necessário realizar algum tipo de pré-processamento do código analisado.

```
package PIMonteCarlo;
import java.io.Serializable;
import br.unicamp.fee.dca.join.framework.app.AppCode;

/**
* DistributeData.java
*
* Created: Thu Mar 21 14:01:09 2005
*
* @author Roberto H. Herai
* @version 1.0
*/

public class DistributeData implements AppCode{
   public Serializable taskrun(Serializable par) {
      int number_tasks = 100;
      int iteration_size = 10000000;
      int limits[][] = new int[number_tasks][1];
      for(int i=0;i<number_tasks;i++){
         limits[0][0]=iteration_size;
      }
      return limits;
   }
}
```

A.2.2 Código da classe ProcessData

A classe ProcessData é a responsável pelo processamento dos dados produzidos pelo lote anterior B_0, enviados pelo computador coordenador, representa o algoritmo executado pelas tarefas do lote B_1, de cardinalidade 100.

```
package PIMonteCarlo;
import java.io.Serializable;
import br.unicamp.fee.dca.join.framework.app.AppCode;

/**
* ProcessData.java
*
* Created: Thu Mar 21 17:20:15 2005
*
* @author Roberto H. Herai
* @version 1.0
*/

public class ProcessData implements AppCode{
   public Serializable taskrun(Serializable par) {
      int bound[] = (int[])par;
      int limit = bound[0];
      int total = 0;
      double x,y;
      double r=10;
      for(int i=0; i<=limit; i++) {
         x = Math.random();
         y = Math.random();
         if((x*x+y*y< 1))  total++;
      }
      int result[] = new int[1];
      result[0]=total;
      return result;
   }
}
```

A.2.3 Código da classe GatherResults

A classe GatherResults é a responsável pela coleta e organização dos resultados gerados pelo processamento das tarefas nos computadores trabalhadores. Esta classe representa o algoritmo executado pelo lote B_2, de cardinlidade 1.

```
package PIMonteCarlo;
import java.io.Serializable;
```

```
import br.unicamp.fee.dca.join.framework.app.AppCode;

/**
* GatherResults.java
*
* Created: Thu Mar 21 17:00:30 2005
*
* @author Roberto H. Herai
* @version 1.0
*/

public class GatherResults implements AppCode{
   public Serializable taskrun(Serializable par) {
      Object[] results = (Object[]) par;
      long total = 0;
      int number_tasks = results.length;
      for(int i=0; i<number_tasks; i++) {
         total += ((int[]) results[i])[0];
      }
      int iteration_size = 10000000;
      long number_of_iterations = number_tasks *  iteration_size;
      double pi[] =new double[1];
      pi[0]= 4 *  total/number_of_iterations;
      return pi;
   }
}
```

A.3 Modelo PAMELA simplificado da aplicação Java

O modelo da aplicação, produzido pela ferramenta AMGen, é representado nesta seção de maneira simplificada. A parte do modelo que representa o código da tarefa i do lote B_i em um computador p $ProcessData(p, i)$ ($p \in 1, 2, ..., P$, $i \in 0, 1, ..., 99$), é a mesma para todas as tarefas do lote. Desta forma, é ilustrado apenas o modelo $ProcessData(p, 0)$ da tarefa 0 associada a um computador p. Os modelos que representam o código executado pelos demais lotes, B_0 e B_2, são ilustrados de forma completa. Nas expressões foi utilizado o parâmetro COORD que representa o coordenador em JoiN.

```
COORD=0;
L=L_0;L_1;L_2
L_0={
   use(COORD,task[0,0](COORD));
   seq(p=1,P){
      use(net,send[COORD,p](codeSize(task[1,0])));
   }
   id=0;
```

```
   par(p=1,P){
      par(i=0,100*W[p]-1){
         use(net,send[COORD,p](dataSizeIn(task[1,id])));
         id=id+1;
      }
   }
}

task[0,0](COORD)=DistributeData(COORD,0);

DistributeData(COORD,0)={
   taskrun{
      assign_int;
      assign_int;
      100 * assign_int;
      seq(i=1,100){
         array_access;
         assign_int;
         access;
      }
      access;
   }
}

//-----------------------------------------------
L_1={
   id=0;
   par(p=1,P){
      par(i=0,100*W[p]-1){
         use(p,task[1,id](p));
         use(net,send[p,COORD](dataSizeOut(task[1,id])));
         id=id+1;
      }
   }
}

task[1,taskNumber](p)=ProcessData(p,taskNumber);

ProcessData(p,0)={
   taskrun{
      assign_int;
      cast;
      access;
      assign_obj;
      assign_int;
      array_access;
      assign_int;
```

```
      assign_double;
      assign_int;
      seq(i=1,10000000){
         access;
         compare;
         access;
         access;
         assign_int;
         sum_int;
         assign_double;
         assign_double;
         sum_double;
         mpy_double;
         access;  access;
         mpy_double;
         access;
         access;
         compare;
         if{
            access;
            assign_int;
            sum_int;
         }
      }
      array_access;
      assign_obj;
      access;
      access;
   }
}

codeSize(task[2,0])=879;

L_2={
   use(COORD,task[2,0](COORD));
}

task[2,0](COORD)=GatherResults(COORD,0);

GatherResults(COORD,0)={
   taskrun{
      assign_obj;
      cast;
      access;
      assign_long;
      assign_int;
      seq(i=0,100){
```

```
            access;
            compare;
            access;
            assign_int;
            sum_int;
            assign_long;
            sum_long;
            cast;
            array_access;  access;
         }
      }
}
```

A.4 Modelo analítico de desempenho

Este modelo, produzido pela ferramenta AMC, é representado de maneira simplificada. As expressões matemáticas que representam o modelo da aplicação da tarefa i de um lote B_i em um computador p ($ProcessData(p, i)$ ($p \in 1, 2, ..., P$, $i \in 0, 1, ..., 99$)) é a mesma para todas as tarefas do lote. Desta forma, é ilustrado apenas a expressão $ProcessData(p, 0)$ da tarefa 0 em um computador p. As expressões das tarefas dos demais lotes, B_0 e B_2, são ilustradas de forma completa. Nas expressões foi utilizado o parâmetro COORD que representa o coordenador em JoiN.

```
//Task map

task[1,taskNumber](p)=ProcessData(p,taskNumber)
task[0,0](COORD)=DistributeData(COORD,0)
task[2,0](COORD)=GatherResults(COORD,0)

//Task cost
ProcessData(p,0)=2*inter_assign_obj+4*inter_assign_int
   +inter_assign_double+2*inter_array_access+3*inter_access
   +inter_cast+10000000*(7*inter_access+2*inter_compare
   +2*inter_assign_double+inter_assign_int+inter_sum_int
   +inter_sum_double+2*inter_mpy_double
   +max(inter_access+inter_assign_int+inter_sum_int,0))

GatherResults(COORD,0)=1*inter_assign_obj+1*inter_cast
+1*inter_access_obj+1*inter_assign_long+1*inter_assign_int
+101*(1*inter_access+1*inter_compare
   +2*inter_access+1*inter_assign_int+1*inter_sum_int)
+2*inter_assign_int+3*inter_assign_int+2*inter_assign_long
+1*inter_mpy_int+1*inter_access+2*inter_access
+1*inter_assign_double+1*inter_div_long
+1*inter_mpy_long+1*inter_access_double
+2*inter_access_double+1*inter_access_type
```

```
DistributeData(COORD,0)=1*inter_assign_int+2*inter_assign_int
+1*(100*1)*inter_assign_int+100*inter_array_access
+100*inter_assign_int+100*inter_access+inter_access

T=max(sum[p=0,P-1](inter_send[COORD,p](
   codeSize(task[1,0])))[id=0]
+sum[p=0,P-1](sum[i=0,100*W[p]-1](
   inter_send[COORD,p](dataSizeIn(task[1,id]))
[id=id+1]))+[id=0]sum[p=0,P-1](sum[i=0,100*W[p]-1](
   inter_send[p,COORD](dataSizeOut(task[1,id]))[id=id+1])),
task[0,0](COORD)+sum[p=0,P-1](0)[id=0]+
sum[p=0,P-1](sum[i=0,100*W[p]-1]([id=id+1]0))
+[id=0]
sum[p=0,P-1](sum[i=0,100*W[p]-1]([id=id+1]0))
+task[2,0](COORD),

sum[p=0,P-1](0)[id=0]
+max[p=0,P-1](sum[i=0,100*W[p]-1]([id=id+1]0))
+[id=0]max[p=0,P-1](
   sum[i=0,100*W[p]-1](task[1,id](p)[id=id+1])),

task[0,0](COORD)
+sum[p=0,P-1](inter_send[COORD,p](codeSize(task[1,0])))
[id=0]
+max[p=0,P-1](max[i=0,100*W[p]-1](
   inter_send[COORD,p](dataSizeIn(task[1,id]))[id=id+1]))
+[id=0]
max[p=0,P-1](max[i=0,100*W[p]-1](task[1,id](p)
+inter_send[p,COORD](dataSizeOut(task[1,id]))[id=id+1]))
+task[2,0](COORD))
```

Apêndice B

Desempenho dos computadores

B.1 Desempenho dos computadores de índice 0, 1, 9, 10, 20, 21 e 32

O modelo das máquinas ilustrado lista os computadores utilizados nos experimentos com seus desempenhos na execução de operações básicas de computação e de comunicação, indicados pelos parâmetros $inter\_op\_data_p$ e $inter\_send_{p \rightarrow q}$, respectivamente. O desempenho dos computadores é representado pela execução de uma unidade de cada uma das operações básicas por segundo.

Por motivos de simplicação, são apresentados os desempenhos do computador 0 que faz parte da sub-rede A, dos computadores 1 e 9 que fazem parte da sub-rede B, dos computadores 10 e 20 da sub-rede C e dos computadores 21 e 32 da sub-rede D. Os dois computadores selecionados de cada uma das sub-redes mencionadas representam o computador com o maior e o menor fator de desempenho da sub-rede cosiderada. Nota-se que, além do único computador da rede A, são apresentados o desempenho de apenas dois computadores de cada uma das outras sub-redes, pois o desempenho dos computadores que pertencem a uma mesma sub-rede é bastante semelhante. Isso pode ser comprovado pelo desempenho dos computadores que foi apresentado na Tab. 5.1 do Cap. 5.

```
//desempenho do computador coordenador (indice 0): trancoso
inter_div_int[0]=[3.927165354330709E-8;7.829724409448821E-8]
inter_mpy_double[0]=[9.881889763779529E-8;2.0206692913385824E-7]
inter_sum_int[0]=[5.610236220472442E-9;5.905511811023622E-9]
inter_mpy_int[0]=[8.070866141732285E-9;8.316929133858268E-9]
inter_div_float[0]=[1.081692913385827E-7;2.2741141732283467E-7]
inter_sum_float[0]=[1.0162401574803148E-7;1.9945866141732284E-7]
inter_assign_int[0]=[1.968503937007874E-9;1.968503937007874E-9]
inter_mod_float[0]=[4.2913385826771657E-8;8.494094488188976E-8]
inter_mpy_float[0]=[8.90748031496063E-9;9.891732283464566E-9]
inter_div_double[0]=[1.0757874015748032E-7;2.2253937007874015E-7]
inter_assign_float[0]=[1.3779527559055119E-9;1.3779527559055119E-9]
```

```
inter_loop_iteration[0]=[6.81193920527376E-9;1.413660501739608E-8]
inter_sum_double[0]=[1.0132874015748033E-7;2.09744094488189E-7]
inter_access[0]=[3.789370078740158E-9;3.789370078740158E-9]
inter_assign_double[0]=[1.4763779527559054E-9;1.4763779527559054E-9]
inter_cast[0]=[5.906770505349727E-8;1.1611138522569391E-7]
inter_array_access[0]=[5.081200787401574E-8;9.972933070866142E-8]

//desempenho do computador trabalhador (indice 1): itu
inter_assign_double[1]=[1.8700787401574807E-9;1.9685039370078745E-9]
inter_access[1]=[5.16732283464567E-9;5.265748031496064E-9]
inter_loop_iteration[1]=[7.037783067814198E-9;7.056094732344505E-9]
inter_mpy_float[1]=[1.186023622047244E-8;1.1958661417322836E-8]
inter_mod_float[1]=[5.6496062992125984E-8;5.659448818897638E-8]
inter_sum_float[1]=[1.3469488188976377E-7;1.3479330708661416E-7]
inter_sum_int[1]=[7.332677165354331E-9;7.431102362204726E-9]
inter_assign_float[1]=[2.6082677165354333E-9;2.7066929133858275E-9]
inter_div_float[1]=[1.4163385826771654E-7;1.419291338582677E-7]
inter_assign_int[1]=[2.165354330708661E-9;2.263779527559055E-9]
inter_mpy_double[1]=[1.3208661417322836E-7;1.3223425196850396E-7]
inter_send[1][0](64)=[8.0;17.0]
inter_sum_double[1]=[1.3459645669291335E-7;1.3474409448818895E-7]
inter_div_int[1]=[5.236220472440945E-8;5.246062992125985E-8]
inter_mpy_int[1]=[1.0777559055118112E-8;1.0875984251968505E-8]
inter_div_double[1]=[1.4148622047244094E-7;1.4266732283464567E-7]
inter_cast[1]=[7.717293247526734E-8;7.736281958056117E-8]
inter_array_access[1]=[6.7347440944881885E-8;6.739665354330708E-8]

//desempenho do computador trabalhador (indice 9): guaruja
inter_assign_double[9]=[2.017716535433071E-9;2.1161417322834645E-9]
inter_access[9]=[5.659448818897638E-9;5.708661417322834E-9]
inter_loop_iteration[9]=[7.648171885491059E-9;7.690899102728439E-9]
inter_mpy_float[9]=[1.2893700787401574E-8;1.2992125984251969E-8]
inter_mod_float[9]=[6.151574803149607E-8;6.161417322834645E-8]
inter_sum_float[9]=[1.4660433070866142E-7;1.4749015748031495E-7]
inter_sum_int[9]=[8.021653543307087E-9;8.120078740157481E-9]
inter_assign_float[9]=[2.85433070866141 7E-9;2.903543307086614E-9]
inter_div_float[9]=[1.5418307086614173E-7;1.550688976377953E-7]
inter_assign_int[9]=[2.362204724409449E-9;2.4606299212598425E-9]
inter_mpy_double[9]=[1.4379921259842518E-7;1.4394685039370078E-7]
inter_send[9][0](64)=[8.0;20.0]
inter_sum_double[9]=[1.4650590551181103E-7;1.4660433070866142E-7]
inter_div_int[9]=[5.698818897637795E-8;5.713582677165355E-8]
inter_mpy_int[9]=[1.1761811023622047E-8;1.186023622047244E-8]
inter_div_double[9]=[1.5403543307086614E-7;1.5428149606299212E-7]
inter_cast[9]=[8.39341519328322E-8;8.527992214086682E-8]
inter_array_access[9]=[7.330216535433071E-8;7.3745078740157475E-8]
```

```
//desempenho do computador trabalhador (indice 10): variac
inter_assign_double[10]=[3.5433070866141735E-9;3.690944881889764E-9]
inter_access[10]=[2.362204724409449E-9;2.4606299212598425E-9]
inter_loop_iteration[10]=[2.624671916010499E-9;2.6979185741317222E-9]
inter_mpy_float[10]=[1.5895669291338583E-8;1.609251968503937E-8]
inter_mod_float[10]=[1.2234251968503935E-7;1.2396653543307087E-7]
inter_sum_float[10]=[5.665846456692912E-7;5.72244094488189E-7]
inter_sum_int[10]=[8.56299212598425E-9;9.05511811023622E-9]
inter_assign_float[10]=[3.5433070866141735E-9;3.690944881889764E-9]
inter_div_float[10]=[6.109251968503938E-7;6.156496062992125E-7]
inter_assign_int[10]=[1.1318897637795275E-9;1.1811023622047246E-9]
inter_mpy_double[10]=[5.562992125984253E-7;5.603346456692913E-7]
inter_send[10][0](64)=[7.0;12.0]
inter_sum_double[10]=[5.654527559055117E-7;5.705708661417324E-7]
inter_div_int[10]=[4.4488188976377963E-8;4.493110236220473E-8]
inter_mpy_int[10]=[1.8159448818897637E-8;1.8307086614173228E-8]
inter_div_double[10]=[6.101870078740158E-7;6.149114173228347E-7]
inter_cast[10]=[6.394301200918403E-8;6.442298119494067E-8]
inter_array_access[10]=[2.832923228346456E-7;2.861220472440945E-7]

//desempenho do computador trabalhador (indice 20): resistor
inter_assign_double[20]=[3.5433070866141735E-9;3.6417322834645676E-9]
inter_access[20]=[2.362204724409449E-9;2.4606299212598425E-9]
inter_loop_iteration[20]=[2.636879692364036E-9;2.7650613440761762E-9]
inter_mpy_float[20]=[1.594488188976378E-8;1.6092519685039375E-8]
inter_mod_float[20]=[1.222933070866141 7E-7;1.2327755905511812E-7]
inter_sum_float[20]=[5.674212598425196E-7;5.716535433070866E-7]
inter_sum_int[20]=[8.56299212598425E-9;9.104330708661416E-9]
inter_assign_float[20]=[3.5433070866141735E-9;3.690944881889764E-9]
inter_div_float[20]=[6.110236220472441E-7;6.156003937007874E-7]
inter_assign_int[20]=[1.1318897637795275E-9;1.1811023622047246E-9]
inter_mpy_double[20]=[5.56496062992126E-7;5.605807086614173E-7]
inter_send[20][0](64)=[7.0;11.0]
inter_sum_double[20]=[5.666830708661418E-7;5.709645669291339E-7]
inter_div_int[20]=[4.4537401574803145E-8;4.498031496062993E-8]
inter_mpy_int[20]=[1.8159448818897637E-8;1.8356299212598426E-8]
inter_div_double[20]=[6.109744094488189E-7;6.154527559055118E-7]
inter_cast[20]=[6.398024232965589E-8;6.443927271543004E-8]
inter_array_access[20]=[2.837106299212598E-7;2.858267716535433E-7]

//desempenho do computador trabalhador (indice 21): img
inter_assign_double[21]=[4.232283464566929E-9;4.3799212598425194E-9]
inter_access[21]=[2.854330708661417E-9;5.118110236220473E-9]
inter_loop_iteration[21]=[3.2045412928035156E-9;3.6867484587682355E-9]
inter_mpy_float[21]=[1.934055118110236E-8;1.9488188976377955E-8]
inter_mod_float[21]=[1.5137795275590552E-7;1.5255905511811022E-7]
inter_sum_float[21]=[8.100393700787402E-7;8.109744094488189E-7]
```

```
inter_sum_int[21]=[1.0679133858267718E-8;1.186023622047244E-8]
inter_assign_float[21]=[4.281496062992125E-9;4.3799212598425194E-9]
inter_div_float[21]=[8.611220472440944E-7;8.6373031496063E-7]
inter_assign_int[21]=[1.3779527559055119E-9;1.4763779527559054E-9]
inter_mpy_double[21]=[7.943897637795276E-7;7.99212598425197E-7]
inter_send[21][0](64)=[62.0;372.0]
inter_sum_double[21]=[8.065944881889764E-7;8.116633858267715E-7]
inter_div_int[21]=[5.4183070866141725E-8;5.5068897637795294E-8]
inter_mpy_int[21]=[2.1998031496062995E-8;2.2293307086614176E-8]
inter_div_double[21]=[8.596948818897638E-7;8.625000000000001E-7]
inter_cast[21]=[7.83042917607147E-8;7.844453464035853E-8]
inter_array_access[21]=[4.050196850393701E-7;4.0548720472440945E-7]

//desempenho do computador trabalhador (indice 32): rib
inter_assign_double[32]=[4.232283464566929E-9;4.3799212598425194E-9]
inter_access[32]=[2.903543307086614E-9;3.001968503937008E-9]
inter_loop_iteration[32]=[3.216749069157053E-9;3.3083073918085823E-9]
inter_mpy_float[32]=[1.934055118110236E-8;1.9488188976377955E-8]
inter_mod_float[32]=[1.514271653543307E-7;1.517224409448819E-7]
inter_sum_float[32]=[8.09251968503937E-7;8.104330708661418E-7]
inter_sum_int[32]=[1.0925196850393701E-8;1.1466535433070865E-8]
inter_assign_float[32]=[4.232283464566929E-9;4.3799212598425194E-9]
inter_div_float[32]=[8.616141732283463E-7;8.625E-7]
inter_assign_int[32]=[1.3779527559055119E-9;1.4271653543307086E-9]
inter_mpy_double[32]=[7.969980314960631E-7;7.97982283464567E-7]
inter_send[32][0](64)=[62.0;655.0]
inter_sum_double[32]=[8.089566929133859E-7;8.110236220472441E-7]
inter_div_int[32]=[5.423228346456694E-8;5.447834645669292E-8]
inter_mpy_int[32]=[2.2047244094488193E-8;2.2145669291338586E-8]
inter_div_double[32]=[8.60531496062992E-7;8.616633858267717E-7]
inter_cast[32]=[7.830021702758653E-8;7.837871758404678E-8]
inter_array_access[32]=[4.046259842519685E-7;4.052165354330709E-7]
```

Apêndice C

Resultados de experimentos complementares

C.1 Cálculo do número π

Esta aplicação, dividida nas versões π_A, π_B, π_C e π_D, realiza o cálculo do número π utilizando uma técnica conhecida como método de Monte Carlo. As 4 versões, descritas na seção 5.2 deste trabalho, variam quanto a quantidade de tarefas e quanto ao tamanho do problema, listados na Tab. 5.3.

Os resultados obtidos das versões π_B, π_C e π_D da aplicação de cálculo do número π são apresentados a seguir. Experimentos com a versão π_A foram apresentados co Cap. 5

C.1.1 Versão π_B

Os resultados obtidos nos experimentos desta versão são apresentados na Tab. C.1. Nela, das 20 medições realizadas, em apenas 6 as estimativas fornecidas pelo sistema de predição apresentaram erro. Além disso, todos os erros verificados foram pequenos. Os dois maiores erros ocorreram quando foram utilizados 16 computadores, e foram iguais a $E_{16,2} = 8,32\%$ e $E_{16,1} = 7,88\%$ na segunda e na primeira execução da versão com 16 computadores, respectivamente. Também nessa quantidade de computadores que foi obtida a maior freqüência de erros, onde em 2 das 4 o sistema de predição errou. Além disso, por apresentarem os dois maiores erros entre todos os valores de P, o erro médio também foi o maior, sendo igual a $\overline{E}_{16} = 4,05\%$. Para as 5 variações de computadores utilizados nos experimentos da versão, com exceção de 16 computadores, todos os demais apresentaram apenas um dos 4 tempos de execução com erro.

O gráfico da Fig. C.1 ilustra de a proximidade dos tempos de execução com relação as estimativas geradas. Por meio dele, é fácil visualizar que, mesmo apresentando erros em algumas estimativas (conforme apresentado na Tab. C.1), todos os tempos de execução estão muito próximos das estimativas produzidas pelo sistema de predição. Nesta versão, também é fácil visualizar que o aumento da

Tab. C.1: Resultados da aplicação de cálculo do número π: versão π_B

Cálculo do número π: versão π_B					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	1388	[1378;1515]	0	0,33
	2	1359		1,31	
	3	1410		0	
	4	1451		0	
4	1	851	[748;843]	1,01	0,25
	2	770		0	
	3	813		0	
	4	803		0	
8	1	415	[392;443]	0	0,42
	2	410		0	
	3	385		1,68	
	4	427		0	
16	1	259	[216;241]	7,88	4,05
	2	260		8,32	
	3	230		0	
	4	223		0	
32	1	134	[112;136]	0	1,01
	2	127		0	
	3	129		0	
	4	141		4,03	

quantidade de computadores não fez com que a diferença das estimativas com o tempo de execução piorasse.

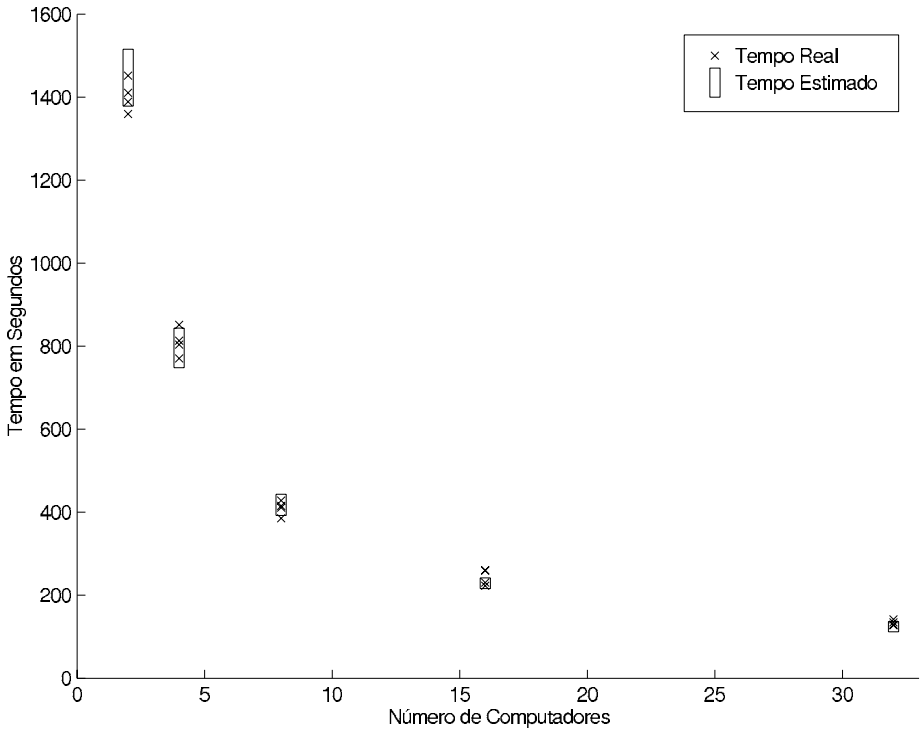

Fig. C.1: Tempo para cálculo do número π: versão π_B

C.1.2 Versão π_C

Os resultados obtidos nos experimentos desta versão são apresentados na Tab. C.2. Nela, dos 20 tempos de execução, em apenas 5 delas o sistema de predição não acertou as estimativas. Destas, apenas uma delas apresentou erro maior que 4%, que foi de $E_{32,4} = 15,13\%$ na quarta execução quando foram utilizados 32 computadores. Para esta quantidade de computadores, também foi registrado o segundo maior erro de uma das execuções da versão, que foi de $E_{32,1} = 3,96\%$ na primeira execução, os quais foram responsáveis pelo maior erro médio da versão, igual a $\overline{E}_{32} = 4,62\%$. Nas outras estimativas, quando foram utilizados 2, 4, 8 e 16 computadores, o erro jamais foi maior que 2,5%. Excepcionalmente quando foram utilizados 4 computadores, a estimativa acertou todos os tempos de execução. Em função disso, com exceção do uso de 4 computadores, o erro médio com 2, 8 e 16 computadores não ultrapassou 0,6%.

A grande quantidade de acertos fez com que o erro médio para todos os valores de P experimentados fosse pequeno, sendo que em nenhum deles o erro ultrapassou 5%.

No gráfico da Fig. C.2 é ilustrada a relação dos tempos de execução com os tempos estimados

Tab. C.2: Resultados da aplicação de cálculo do número π: versão π_C

Cálculo do número π: versão π_C					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	691	[687;759]	0	0,55
	2	671		2,21	
	3	727		0	
	4	721		0	
4	1	398	[364;411]	0	0
	2	378		0	
	3	410		0	
	4	383		0	
8	1	197	[189;213]	0	0,50
	2	185		1,99	
	3	209		0	
	4	203		0	
16	1	110	[98;117]	0	0,23
	2	97		0,93	
	3	107		0	
	4	112		0	
32	1	65	[56;63]	3,36	4,62
	2	58		0	
	3	63		0	
	4	72		15,13	

quando a quantidade de computadores variou. Através dele, pode-se visualizar claramente o caso excepcional da versão π_C em que, com 4 computadores, todos os tempos de execução permaneceram sobre os limites de tempo do intervalo da estimativa calculada. Pode-se visualizar também que, apesar do erro $E_{32,4} = 15,13\%$ ser relativamente grande em relação ao tamanho do intervalo para $P = 32$, o tempo representado permitiu estimar de forma eficiente o tempo provável que a versão demorará para que seja executada. Uma consideração importante é que, como em outros casos, a variação de P não afetou a qualidade dos resultados fornecidos pelo sistema de predição. Mesmo dobrando a quantidade de tarefas e reduzindo a carga computacional de cada uma delas em relação a versão π_B, os resultados fornecidos pelo sistema de predição para as versões da aplicação considerada foram satisfatórios.

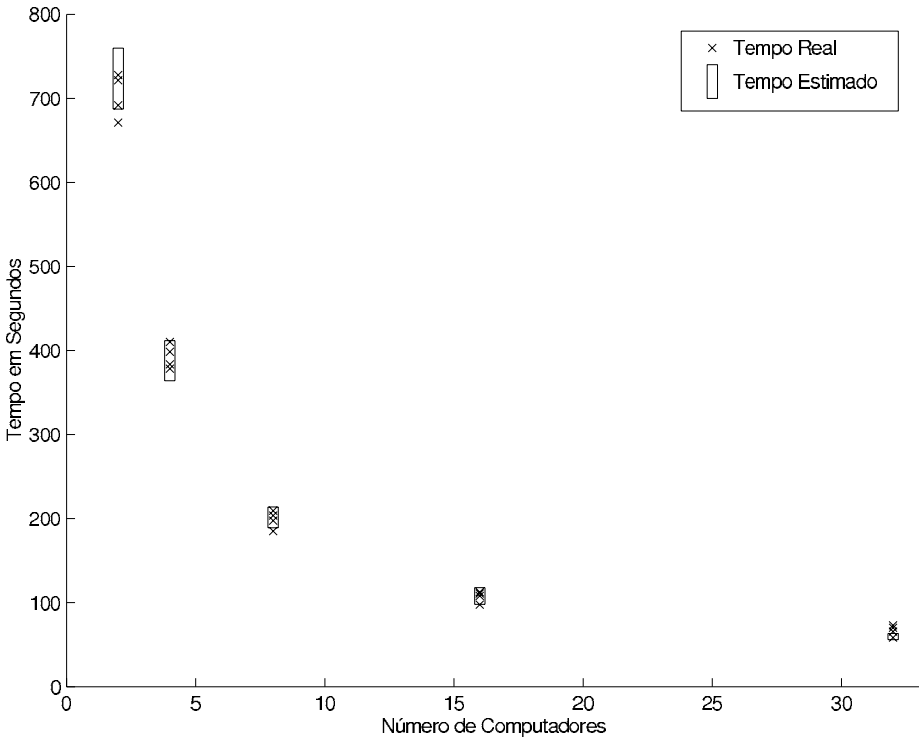

Fig. C.2: Tempo para cálculo do número π: versão π_C

C.1.3 Versão π_D

Os resultados obtidos nos experimentos desta versão são apresentados na Tab. C.3. Nela, apenas 6 das 20 estimativas apresentaram erro. Com exceção dos erros $E_{16,3} = 6,24\%$ e $E_{32,3} = 5,04\%$, todas as demais apresentaram erros inferiores a 2,6%. O menor erro foi de $E_{4,2} = 1,29\%$, verificada na segunda execução para $P = 4$. Com $P = 32$, observou-se a maior freqüência de estimativas com erro, com 3 dos 4 tempos de execução. Isso fez com que o erro médio para esse valor de P fosse o maior de todos. De forma totalmente contrária, com $P = 8$ observou-se que todos os tempos de

execução ficaram entre os limites do intervalo estimado, isto é, nenhum erro foi observado. Para as estimativas quando foram utilizados 2 e 4 computadores, apenas um dos 4 tempos de execução de cada valor de P apresentou erro. Em função disso, o erro médio de tais quantidades de computadores foi inferior a 1%.

Tab. C.3: Resultados da aplicação de cálculo do número π: versão π_D

Cálculo do número π: versão π_D					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	1443	[1414;1599]	0	0,35
	2	1530		0	
	3	1393		1,39	
	4	1568		0	
4	1	734	[728;823]	0	0,32
	2	718		1,29	
	3	743		0	
	4	781		0	
8	1	407	[378;427]	0	0
	2	426		0	
	3	391		0	
	4	411		0	
16	1	221	[196;221]	0	1,56
	2	205		0	
	3	234		6,24	
	4	211		0	
32	1	121	[112;126]	0	2,52
	2	109		2,52	
	3	132		5,04	
	4	129		2,52	

O gráfico da Fig. C.3 ilustra os tempos de execução e os tempos das estimativas obtidas pelo sistema de predição. Através dele é fácil identificar que, para P =2, 4 e 16 computadores, apenas um dos 4 tempos de execução ficou fora dos limites do intervalo estimado. Nota-se também, pelo gráfico, que a diferença dos tempos de execução em relação aos intervalos é pequena, pois todos estão visualmente entre ou bastante próximos dos limites estabelecidos pelo intervalo. O único caso em que todos os tempos estão entre os limites do intervalo ocorreu com $P = 32$. De forma geral, os resultados para todas as versões desta aplicação foram satisfatórios.

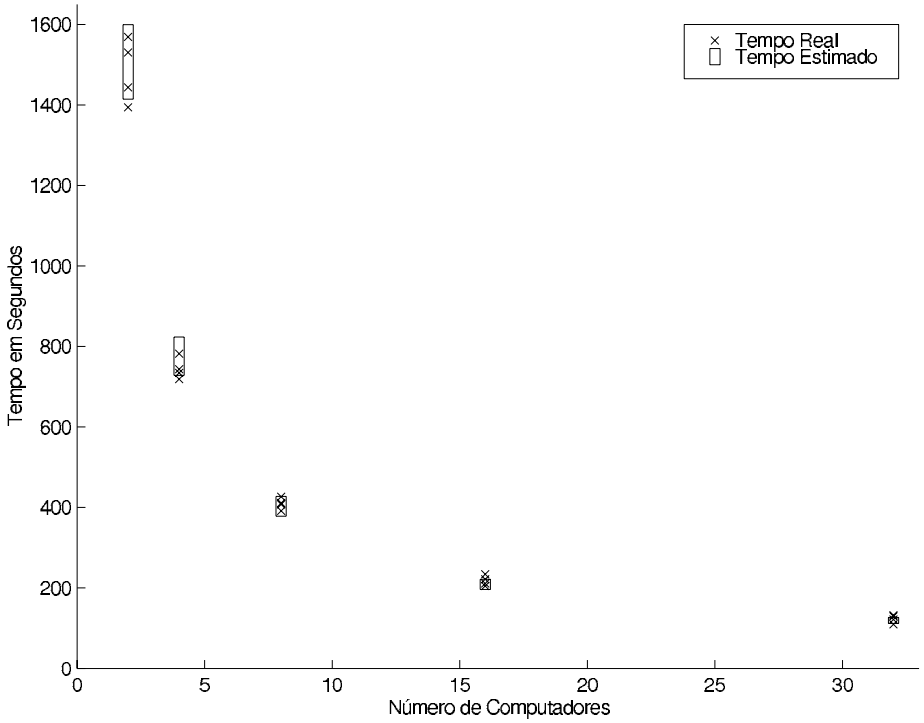

Fig. C.3: Tempo para cálculo do número π: versão π_D

C.2 Busca de números primos

Esta aplicação, dividida nas versões $prim_A$, $prim_B$, $prim_C$ e $prim_D$, realiza a busca de números primos utilizand uma técnica conhecida como Crivo de Eratóstenes. As 4 versões, descritas na seção 5.2 deste trabalho, variam quanto a quantidade de tarefas e quanto ao tamanho do problema, listados na Tab. 5.4.

Os resultados obtidos das versões $prim_B$, $prim_C$ e $prim_D$ da aplicação de busca de números primos são apresentados a seguir. Experimentos com a versão $prim_A$ foram apresentados co Cap. 5

C.2.1 Versão $prim_B$

Os resultados obtidos nos experimentos desta versão são apresentados na Tab. C.4. Nela, apenas 3 das 20 execuções não apresentaram erro em suas estimativas. Elas correspondem a terceira execução com 2 e 16 computadores, e a segunda execução com 32 computadores. Os 17 tempos que apresentaram erro em suas estimativas, ficaram todos abaixo do tempo do limite inferior dos intervalos calculados. O maior erro verificado foi de $E_{16,2} = 20,99\%$ na segunda execução com 16 computadores, e foi também para esta quantidade de computadores que observou-se a maior média de erro, que foi de $\overline{E}_{16} = 11,37\%$. A menor média de erro ocorreu quando foram utilizados 2 computadores,

com $\overline{E}_2 = 3,52\%$, e é quase 4 vezes menor do que a maior média de erro observada.

Tab. C.4: Resultados da aplicação de busca de números primos: versão $prim_B$

Busca de números primos: versão $prim_B$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	3542	[3740;4122]	5,04	3,52
	2	3475		6,74	
	3	3773		0	
	4	3650		2,29	
4	1	1526	[1880;2201]	17,35	10,45
	2	1760		5,88	
	3	1679		9,85	
	4	1702		8,72	
8	1	941	[1118;1426]	13,92	8,67
	2	974		11,32	
	3	1093		1,97	
	4	1023		7,47	
16	1	547	[632;740]	12,39	11,37
	2	488		20,99	
	3	663		0	
	4	549		12,10	
32	1	343	[374;438]	7,64	6,22
	2	389		0	
	3	322		12,81	
	4	356		4,43	

Pelo gráfico da Fig. C.4 pode-se observar claramente os casos em que, com 2, 16 e 32 computadores, apenas um dos 4 tempos de execução permaneceu sobre as estimativas. Nota-se também que, com o aumento da quantidade de computadores, os tempos de execução sofreram uma leve aproximação com relação aos intervalos estimados. Com 32 computadores, apesar de apenas 1 dos tempos estar entre os valores do intervalo calculado, a diferença com os demais tempos de execução é tão pequena que, na ilustração do gráfico, causam a impressão de que estão entre os limites do intervalo calculado.

C.2.2 Versão $prim_C$

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.5. Nela, dos 20 tempos de execução, apenas 1 não apresentou erro, o qual corresponde a primeira execução com 2 computadores. Dos 19 erros ocorridos, 13 deles são maiores do que 11%. Isso fez com que a média de erro para esta versão, para diferentes valores de P, fosse alta. Apenas com 2 computadores a média

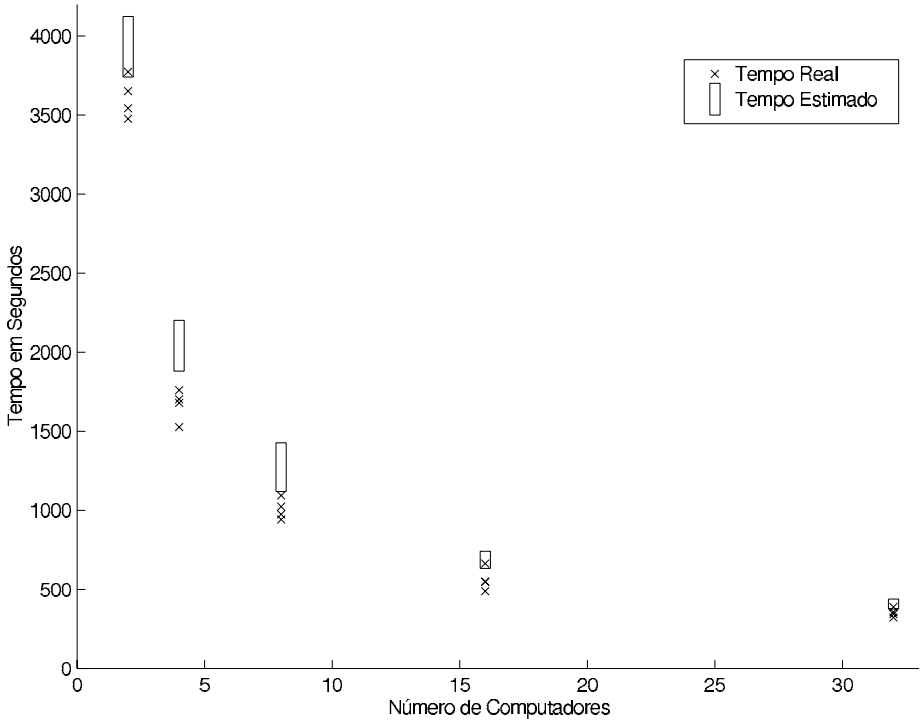

Fig. C.4: Tempo para busca de números primos: versão $prim_B$.

de erro foi inferior a 10% ($E_2 = 8,2\%$). Para os demais valores de P, o erro variou aproximadamente entre 11% e 15%.

Os dados do gráfico da Fig. C.5 apresentam os tempos de execução e os intervalos estimados para a versão. Pelo gráfico, é fácil identificar que, com P=4, 8, 16 e 32 computadores, todas as estimativas apresentaram erro. Apenas para $P = 2$ é que foi verificado um acerto entre as 20 estimativas. Nota-se também pelo gráfico que na metade das estimatvas com P=2, 4, 8 e 16 a diferença dos tempos de execução em relação aos intervalos é pequena. Apenas com P=32 que todos os tempos de execução estão bastante próximos dos intervalos estimados.

Nota-se que as estimativas para esta versão, que apresenta o dobro da quantidade de tarefas da versão $prim_B$, apresentaram um comportamento esperado, em que a maioria dos tempos de execução ficaram abaixo do limite inferior dos intervalos de tempo estimado pelo sistema de predição. Além disso, é fácil verificar que com 32 computadores, os tempos de execução ficaram mais próximos entre si.

C.2.3 Versão $prim_D$

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.6. Nela, das 20 execuções realizadas, todas as estimativas apresentaram erro. Além disso, na grande maioria das

Tab. C.5: Resultados da aplicação de busca de números primos: versão $prim_C$

Busca de números primos: versão $prim_C$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$(%)
2	1	926	[890;1042]	0	8,20
	2	870		2,07	
	3	749		14,60	
	4	734		16,15	
4	1	395	[470;550]	14,71	11,47
	2	450		3,92	
	3	372		19,22	
	4	429		8,04	
8	1	226	[283;331]	18,57	14,90
	2	207		24,76	
	3	254		9,45	
	4	262		6,84	
16	1	118	[158;185]	23,32	15,01
	2	139		11,08	
	3	125		19,24	
	4	147		6,41	
32	1	74	[93;109]	18,81	13,86
	2	81		11,88	
	3	77		15,84	
	4	84		8,91	

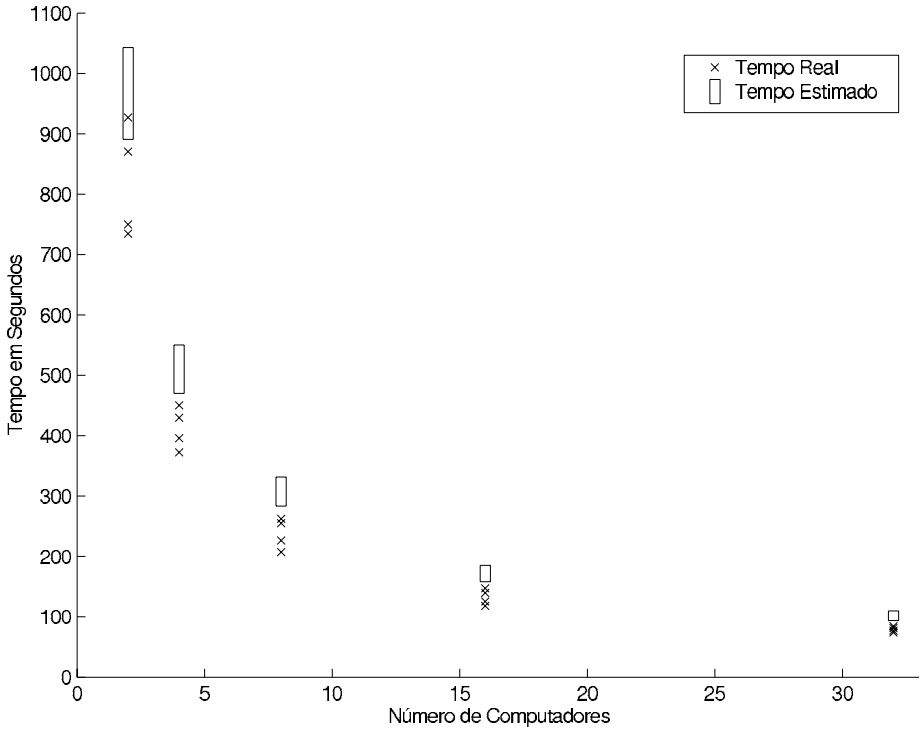

Fig. C.5: Tempo para busca de números primos: versão $prim_C$.

estimativas o erro foi maior do que 9%, e isso contribuiu para que o erro médio com P=4, 8, 16 e 32 computadores fosse maior do que 10%. Apenas para P=2 que o erro médio foi inferior a 10%.

Tab. C.6: Resultados da aplicação de busca de números primos: versão $prim_D$

Busca de Números Primos: versão $prim_D$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	3088	[3271;3530]	5,38	5,64
	2	3119		4,47	
	3	2975		8,70	
	4	3135		4	
4	1	1386	[1726;2021]	18,15	16,21
	2	1332		21,03	
	3	1543		9,77	
	4	1428		15,91	
8	1	749	[1040;1217]	25,79	24,08
	2	690		31,01	
	3	732		27,29	
	4	902		12,23	
16	1	401	[580;680]	28,41	22,30
	2	453		20,16	
	3	467		17,94	
	4	437		22,70	
32	1	268	[324;378]	15,95	10,33
	2	288		10,26	
	3	290		9,69	
	4	305		5,41	

O gráfico da Fig. C.6 permite visualizar de forma mais clara alguns dos dados contidos na Tab. C.6. Pode-se verificar que nenhum dos intervalos apresentou algum acerto. Além disso, com exceção da utilização de 32 computadores, todos os tempos de execução permaneceram bastante distantes, e sempre abaixo dos limites inferiores dos intervalos calculados.

C.3 Busca de números primos baseada em probabilidade

Os resultados obtidos das versões $prim_B$, $prim_C$ e $prim_D$ da aplicação de busca de números primos baseada em probabilidade são apresentados a seguir.

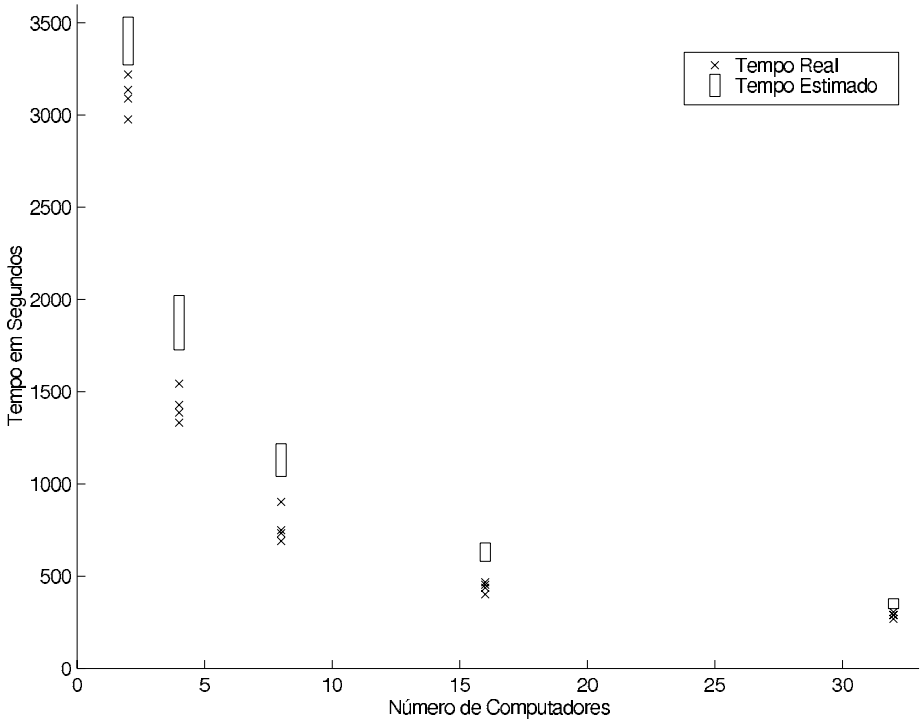

Fig. C.6: Tempo para busca de números primos: versão $prim_D$.

C.3.1 Versão $prim_B$

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.7. Nela, dos 20 tempos de execução, de 3 passaram-se a ser 12 acertos com o uso da probabilidade de execução de um bloco condicional. Além disso, os tempos que apresentavam erros elevados sofreram uma redução bastante significativa desse valor. Com 4 e 8 computadores, em que todos apresentavam erros, passaram-se a ser, para cada valor de P, 3 dos 4 tempos de execução dentro dos intervalos estimados. Além disso, a freqüência de acertos que antes era bem pequena, passou a ser grande e ficou distribuída quase que igualmente entre os valores de P. O maior erro, que era de $E_{16,2} = 20,99\%$ passou a ser de $E_{16,2} = 8,40\%$. Com isso, os erros médios foram substancialmente reduzidos, e variaram entre 0,2% e 3,6%.

No gráfico da Fig. C.7 são apresentadas as novas estimativas para a versão $prim_B$. Por meio dele, é visível a melhora nas estimativas para todos os valores de P, se comparadas com às do gráfico da Fig. C.4. É visível também que, com $P = 2$, apenas um dos 4 tempos de execução não apresentou erro. Porém, com 4 e 8 computadores ocorreu o inverso, pois apenas um dos 4 tempos apresentou erro em relação as estimativas.

Tab. C.7: Resultados da aplicação de busca de números primos baseada na probabilidade de execução de um bloco condicional: versão $prim_B$

Busca de números primos: versão $prim_B$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	E_P (%)
2	1	3542	[3179;3503]	1,17	3,41
	2	3475		0	
	3	3773		8,08	
	4	3650		4,40	
4	1	1526	[1598;1870]	4,15	1,04
	2	1760		0	
	3	1679		0	
	4	1702		0	
8	1	941	[950;1212]	0,83	0,21
	2	974		0	
	3	1093		0	
	4	1023		0	
16	1	547	[537;629]	0	3,56
	2	488		8,40	
	3	663		5,83	
	4	549		0	
32	1	343	[317;372]	0	1,23
	2	389		4,93	
	3	322		0	
	4	356		0	

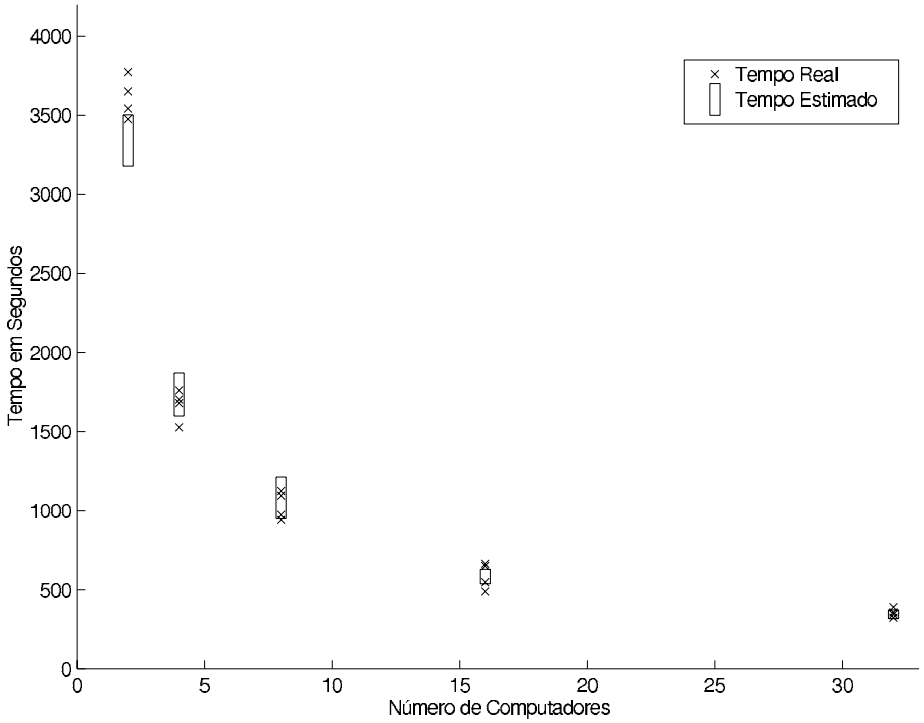

Fig. C.7: Tempo para busca de números primos baseada na probabilidade de execução de um bloco condicional: versão $prim_B$

C.3.2 Versão $prim_C$

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.8. Nela, ao invés de apenas 1 estimativa correta, agora são 9, com 11 ainda apresentando erros. Apesar disso, todas as estimativas com erro sofreram uma considerável redução e, pelos dados da tabela, a freqüência de acertos ficou distribuída quase que igualmente para todos os valores de P. Outro fator importante é que, de todas as estimativas com erro, apenas duas delas são maiores do que 10%, quando foram utilizados 8 e 16 computadores. Em função disso, o erro médio para cada valor de P foi baixo, e jamais foi maior do que $4,6\%$.

Tab. C.8: Resultados da aplicação de busca de números primos, baseada na probabilidade de execução de um bloco condicional: versão $prim_C$

Busca de números primos: versão $prim_C$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	926	[756;885]	5	2,13
	2	870		0	
	3	749		0,85	
	4	734		2,68	
4	1	395	[399;467]	0,92	1,79
	2	450		0	
	3	372		6,24	
	4	429		0	
8	1	226	[240;281]	5,37	4,51
	2	207		12,67	
	3	254		0	
	4	262		0	
16	1	118	[134;157]	11	4,30
	2	139		0	
	3	125		6,19	
	4	147		0	
32	1	74	[79;92]	5,85	2,05
	2	81		0	
	3	77		2,34	
	4	84		0	

Por meio do gráfico da Fig. C.8, é possível visualizar que os intervalos e os tempos de execução estão se sobrepondo ou que estão muito próximos entre si. Com 4, 8 e 16 computadores, nota-se que metade dos tempos estão dentro do intervalo, e a outra metade está logo abaixo do limite inferior dos intervalos. Com 2 computadores, nota-se que há o único tempo de execução entre todos da versão que ficou logo acima do limite superior do intervalo. Com 32 computadores, nota-se que o intervalo foi capaz de estimar para todos os tempos de execução para essa quantidade de computadores. Com-

parando esses dados com os do gráfico da Fig. C.5, pode-se perceber o quanto as estimativas para esta versão melhoraram.

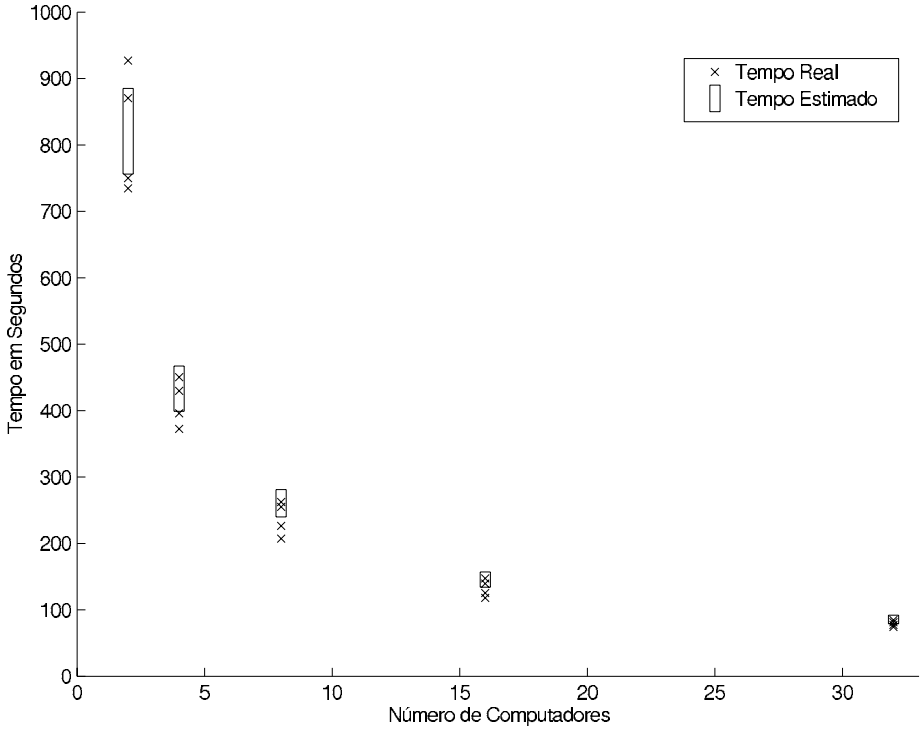

Fig. C.8: Tempo para busca de números primos baseada na probabilidade de execução de um bloco condicional: versão $prim_C$

C.3.3 Versão $prim_D$

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.9. Nela, após o cálculo dos novos intervalos, foram identificadas 6 estimativas com acerto, sendo que nos intervalos antigos, sem considerar a probabilidade de execução de um bloco condicional, nenhum dos 20 tempos de execução apresentou acerto. Além disso, apesar da grande quantidade de erros, vários deles sofreram uma considerável redução devido a aproximação dos valores com os intervalos. O valor do erro que antes oscilava entre 4% e 31% passou a variar de 2% a 21%. Em função disso, o maior erro médio que antes era de $\overline{E}_8 = 24,08\%$ passou a ser de $\overline{E}_8 = 12,54\%$.

A diminuição dos erros torna-se mais aparente através da visualização do gráfico da Fig. C.9. Por meio dele, é fácil identificar que, com o novo intervalo calculado, os tempos de execução ficaram muito próximos se comparados aos intervalos da Fig. C.6, calculados sem considerar a probabilidade de execução de um bloco condicional. Uma característica deste gráfico em relação aos das outras versões desta aplicação, é que as estimativas com erro ainda predominam, apesar de serem pequenas.

Tab. C.9: Resultados da aplicação de busca de números primos, baseada na probabilidade de execução de um bloco condicional: versão $prim_D$

Busca de números primos: versão $prim_D$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	E_P (%)
2	1	3088	[2780;3000]	3,04	2,96
	2	3119		4,12	
	3	2975		0	
	4	3135		4,67	
4	1	1386	[1467;1715]	5,09	4,01
	2	1332		8,49	
	3	1543		0	
	4	1428		2,45	
8	1	749	[884;1034]	14,08	12,54
	2	690		20,23	
	3	732		15,85	
	4	902		0	
16	1	401	[493;578]	17,18	9,99
	2	453		7,47	
	3	467		4,86	
	4	437		10,46	
32	1	268	[275;321]	2,35	0,59
	2	288		0	
	3	290		0	
	4	305		0	

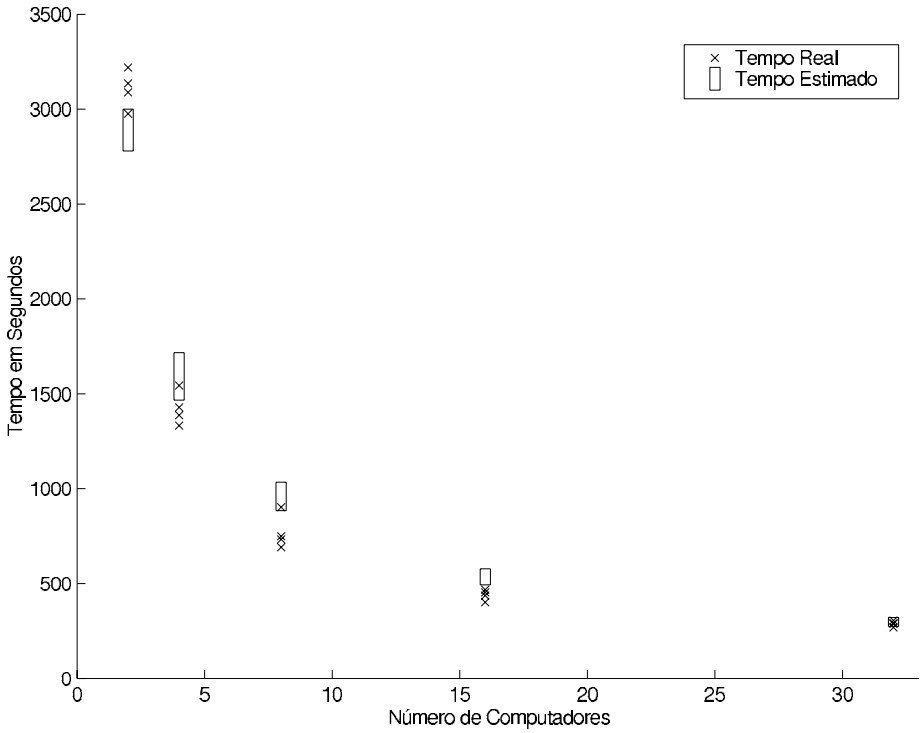

Fig. C.9: Tempo para busca de números primos baseada na probabilidade de execução de um bloco condicional: versão $prim_D$

C.4 Operações completas em matrizes

Esta aplicação, dividida nas versões mat_A, mat_B, mat_C e mat_D, realiza operações completas em matrizes. As 4 versões, descritas na seção 5.2 deste trabalho, variam quanto a quantidade de tarefas e quanto ao tamanho do problema, listados na Tab. 5.5.

Os resultados obtidos das versões mat_B, mat_C e mat_D da aplicação de operações completas em matrizes são apresentados a seguir. Experimentos com a versão mat_A foram apresentados co Cap. 5

C.4.1 Versão mat_B

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.10. Nela, apenas 7 dos 20 tempos de execução não apresentaram erro. O maior erro verificado foi de $E_{32,2} = 18,6\%$ na segunda execução com 32 computadores, bem maior do que o erro médio para este mesmo valor de P, que foi de $\overline{E}_{32} = 6,98\%$. Para todos os valores de P, o único que não apresentou nenhum acerto nas estimativas foi para $P = 16$. Cosequentemente, foi para este mesmo valor de P que foi encontrado o segundo maior erro médio da versão, com $\overline{E}_{16} = 5,89\%$. As demais médias de erro para 2, 4 e 8 computadores foram bem menores, e variaram entre 1% e 2,4%.

Por meio do gráfico da Fig. C.10, pode-se observar que praticamente todos os tempos de execução estão muito próximos dos intervalos calculados. Os tempos que apresentam erro estão todos acima do limite superior dos intervalos e podem ter sido causados pelo aumento da carga de computação, principalmente de comunicação, que também envolve o uso de uma quantidade de memória considerável para manipulação de dados. Excepcionalmente para P=2, nota-se que um dos tempos de execução foi muito maior que os demais. Para outros valores de P, os tempos de execução e estimados permaneceram sempre próximos entre si.

C.4.2 Versão mat_C

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.11. Nela, dos 20 tempos de execução, em 7 deles as estimativas não apresentaram erro. Os acertos foram divididos quase que igualmente entre todos os valores de P, sendo que para 4 e 16 computadores, metade das estimativas foram corretas. Para outros valores de P, apenas uma das 4 estimativas foram corretas. Com relação aos tempos que apresentaram erros em suas estimativas, apenas 5 foram maiores que 5%. Em função disso, a média de erro para todos os valores de P foram pequenos, sendo $\overline{E}_{32} = 3,55\%$ o maior deles, e $\overline{E}_{16} = 1,90\%$ o menor, para 32 e 16 computadores, respectivamente.

A ausência de grandes variações entre os tempos de execução fica mais nítida pela visualização do gráfico da Fig. C.11, em que todos os tempos, para um mesmo valor de P, estiveram sempre bastante próximos entre si. Além disso, nota-se que apenas com P=2 que um dos tempos de execução ficou abaixo do limite inferior da estimativa. Nota-se também que, pelo gráfico, a medida que a quantidade

Tab. C.10: Resultados da aplicação que realiza operações completas em matrizes: versão mat_B

Operações completas em matrizes: versão mat_B					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	2302	[2111;2370]	0	2,34
	2	2342		0	
	3	2580		9,37	
	4	2298		0	
4	1	1296	[1096;1230]	5,67	2,15
	2	1254		2,06	
	3	1240		0,86	
	4	1205		0	
8	1	651	[568;638]	2,16	1
	2	625		0	
	3	649		1,82	
	4	632		0	
16	1	402	[338;392]	2,74	5,89
	2	429		10,14	
	3	396		1,10	
	4	427		9,59	
32	1	206	[172;215]	0	6,98
	2	251		18,60	
	3	232		8,79	
	4	216		0,52	

Tab. C.11: Resultados da aplicação que realiza operações completas em matrizes: versão mat_C

Operações completas em matrizes: versão mat_C					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	1242	[1063;1178]	5,71	3,24
	2	1215		3,30	
	3	1019		3,93	
	4	1148		0	
4	1	591	[562;639]	0	2,46
	2	640		0,17	
	3	697		9,66	
	4	629		0	
8	1	347	[297;328]	6,08	2,96
	2	335		2,24	
	3	339		3,52	
	4	312		0	
16	1	188	[158;184]	2,34	1,90
	2	182		0	
	3	193		5,26	
	4	167		0	
32	1	92	[85;98]	0	3,55
	2	105		7,65	
	3	101		3,28	
	4	101		3,28	

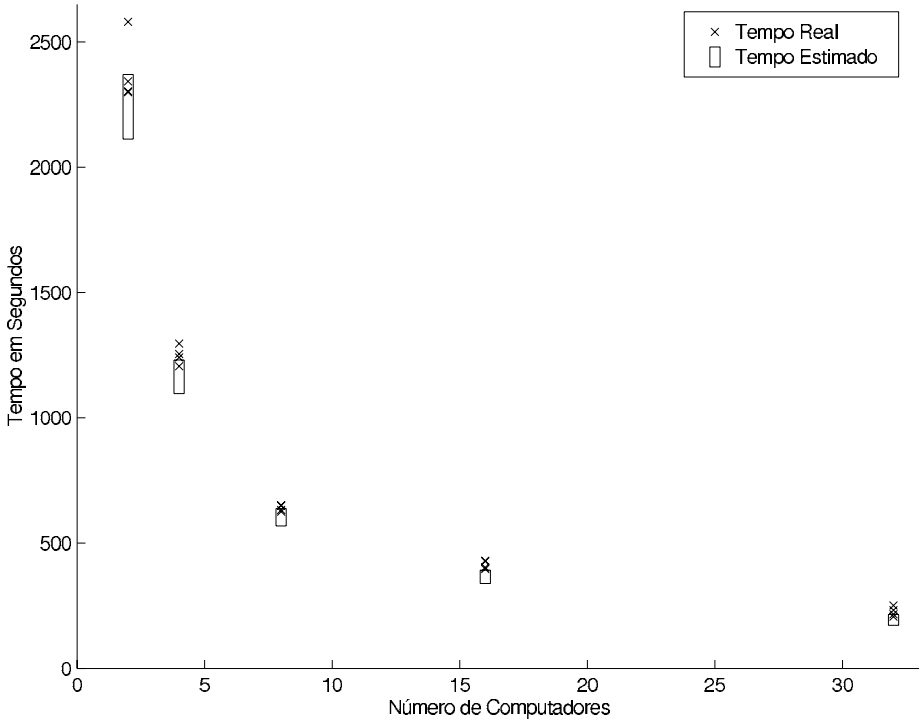

Fig. C.10: Tempo para operações completas em matrizes: versão mat_B

de computadores dobrou, a diminuição dos tempos de execução foram acompanhadas de maneira eficiente pelas estimativas de tempo.

C.4.3 Versão mat_D

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.12. Nela, apesar da pouca quantidade de acertos, em apenas 5 dos 20 tempos de execução, os 15 tempos de execução com erro ficaram sempre muito próximos do tempo estimado. Em função disso, apenas duas estimativas apresentaram erros maiores que 4,6% e foram $E_{16,2} = 7,24\%$ e $E_{32,1} = 8,55\%$, com 16 e 32 computadores, respectivamente. As demais estimativas com erro variaram entre 0,1% e 4,6%. Em função disso, a maior média de erro foi pequena, e corresponde a $\overline{E}_{32} = 4,2\%$ com 32 computadores. A menor média de erro, que foi de $\overline{E}_8 = 1,67\%$ com 8 computadores, foi bastante próxima dos valores do erro para 2, 4 e 16 computadores, e variou em aproximadamente 0,7% para mais e para menos.

No gráfico da Fig. C.12 pode-se comprovar que os baixos valores do erro e do desvio padrão fizeram com que todos os tempos de execução no gráfico ficassem sempre próximos um dos outros. Além disso, nota-se que, com 2 computadores, 3 dos 4 tempos estão fora do intervalo, e um está muito próximo do limite superior do mesmo. Para todos os valores de P, é nítido que há pelo menos uma estimativa sem erro e as que apresentam erro estão agrupados com os demais tempos de execução.

Tab. C.12: Resultados da aplicação que realiza operações completas em matrizes: versão mat_D

Operações completas em matrizes: versão mat_D					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	4870	[4385;4689]	3,99	1,78
	2	4746		1,26	
	3	4690		0,02	
	4	4773		1,85	
4	1	2531	[2283;2449]	3,47	1,97
	2	2479		1,27	
	3	2523		3,13	
	4	2417		0	
8	1	1324	[1163;1296]	2,28	1,67
	2	1284		0	
	3	1270		0	
	4	1350		4,39	
16	1	722	[647;735]	0	2,35
	2	785		7,24	
	3	737		0,29	
	4	748		1,88	
32	1	404	[328;374]	8,55	4,2
	2	387		3,70	
	3	372		0	
	4	390		4,56	

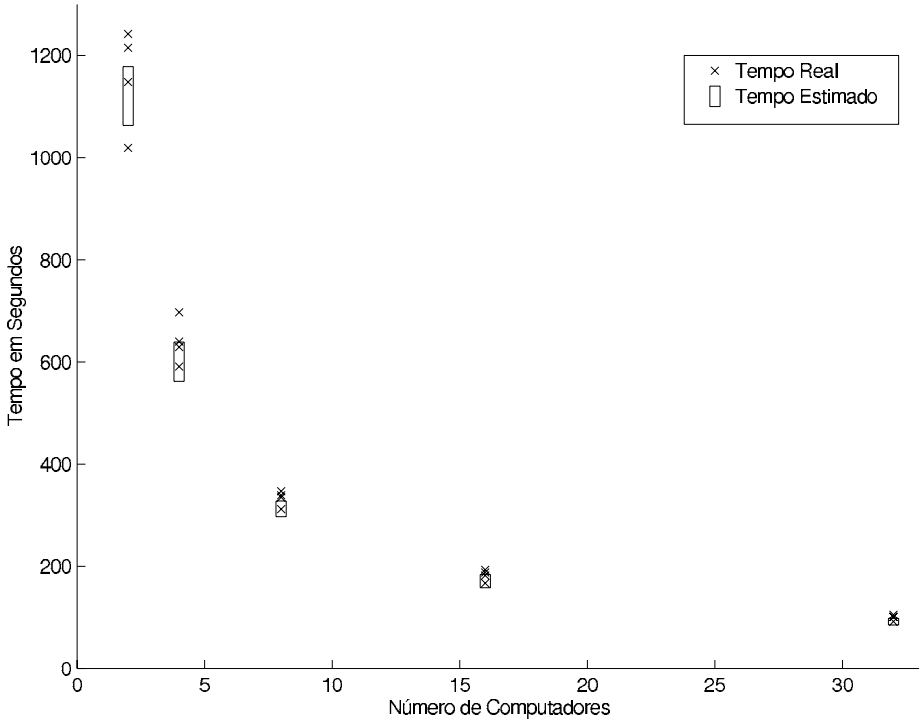

Fig. C.11: Tempo para operações completas em matrizes: versão mat_C

C.5 Operações parciais em matrizes

Esta aplicação, dividida nas versões $rand_A$, $rand_B$, $rand_C$ e $rand_D$, realiza operações parciais em matrizes. As 4 versões, descritas na seção 5.2 deste trabalho, variam quanto a quantidade de tarefas e quanto ao tamanho do problema, listados na Tab. 5.5.

Os resultados obtidos das versões $rand_B$, $rand_C$ e $rand_D$ da aplicação de operações parciais em matrizes são apresentados a seguir. Experimentos com a versão $rand_A$ foram apresentados co Cap. 5

C.5.1 Versão $rand_B$

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.13. Nela, dos 20 tempos de execução, em apenas um deles a estimativa acertou, quando foram utilizados 32 computadores. Para esta mesma quantidade de computadores também foi verificado o menor erro médio, de $\overline{E}_{32} = 2,91\%$, pois todos os erros foram baixos. Este erro foi várias vezes inferior em relação aos demais, sempre maior do que 10%.

Por meio do gráfico da Fig. C.13, é fácil visualizar que com 2, 4 e 8 computadores as estimativas em relação aos tempos de execução ficaram bastante distantes. Porém, com 16 e 32 computadores, os erros são menores, pois as estimativas estão próximas dos tempos de execução. Pelo gráfico, pode-se

Tab. C.13: Resultados da aplicação que realiza operações parciais em matrizes: versão $rand_B$

Operações parciais em matrizes: versão $rand_B$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	2401	[3018;3159]	19,98	18,32
	2	2605		13,37	
	3	2387		20,43	
	4	2416		19,49	
4	1	1255	[1569;1632]	19,62	17,60
	2	1326		15,18	
	3	1278		18,18	
	4	1290		17,43	
8	1	649	[804;841]	18,84	18,63
	2	657		17,87	
	3	616		22,86	
	4	681		14,95	
16	1	345	[408;446]	14,75	10,19
	2	362		10,77	
	3	377		7,26	
	4	374		7,96	
32	1	210	[219;228]	4,03	2,91
	2	217		0,89	
	3	221		0	
	4	243		6,71	

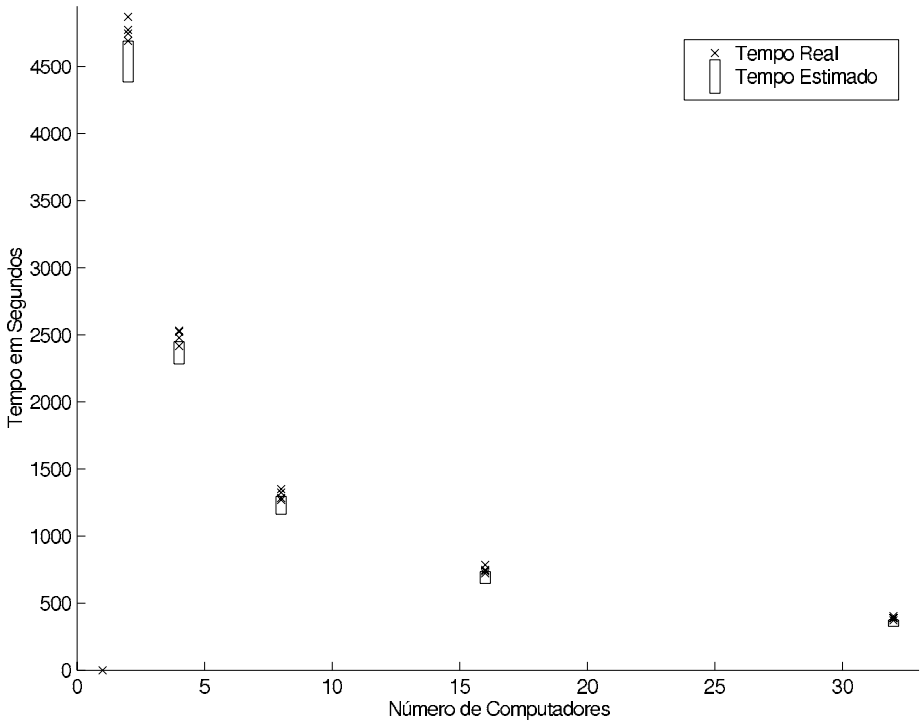

Fig. C.12: Tempo para operações completas em matrizes: versão mat_D

verificar que os tempos de execução não estiveram tão dispersos entre si, pois houve pouca variação de tempo entre eles. A exceção ocorreu com 2 computadores, onde é nítida a grande distância de um dos tempos em relação aos demais.

C.5.2 Versão $rand_C$

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.14. Nela, todas as estimativas apresentaram erro e, das 20 realizadas, em 14 delas o erro foi maior do que 10%. Em função disso, o erro médio para 4, 8, 16 e 32 computadores ficou acima desse valor. A exceção foi com 2 computadores, em que o erro médio foi de $\overline{E}_2 = 7,06\%$, pois todas as estimativas ficaram com erro entre 4,3% e 9,3%. O maior erro médio ocorreu com o uso de 16 computadores e foi de $\overline{E}_{16} = 23,63\%$. Os erros das estimativas para este valor de P foram os maiores da versão.

No gráfico da Fig. C.14, são ilustrados os tempos de execução e as estimativas para a versão $rand_C$. Nele, é fácil visualizar que os tempos com os menores erros são aqueles gerados com 32 computadores. Para outros valores de P, os erros foram grandes, pois a diferença de tempo que existe entre as estimativas e os tempos de execução é nítida. Com 8 e 16 computadores, as quais apresentam as duas maiores médias de erro, nota-se que o tamanho do intervalo é pequeno e isso fez com que no cálculo dos erros os valores fossem maiores, mesmo estando mais próximos do intervalo do que os tempos com 2 e 4 computadores.

Tab. C.14: Resultados da aplicação que realiza operações parciais em matrizes: versão $rand_C$

Operações parciais em matrizes: versão $rand_C$					
P	i	$T_{X_{P,i}}$ **(seg)**	T_{E_P} **(seg)**	$E_{P,i}$ (%)	$\overline{E}_P$ (%)
2	1	1325	[1389;1560]	4,34	7,06
	2	1286		6,99	
	3	1277		7,60	
	4	1252		9,29	
4	1	653	[741;817]	11,30	14,63
	2	630		14,25	
	3	628		14,51	
	4	597		18,49	
8	1	351	[403;422]	12,61	16,18
	2	338		15,76	
	3	333		16,97	
	4	323		19,39	
16	1	149	[206;215]	27,08	23,63
	2	164		19,95	
	3	167		18,53	
	4	145		28,98	
32	1	89	[108;113]	17,19	10,86
	2	98		9,05	
	3	103		4,52	
	4	94		12,67	

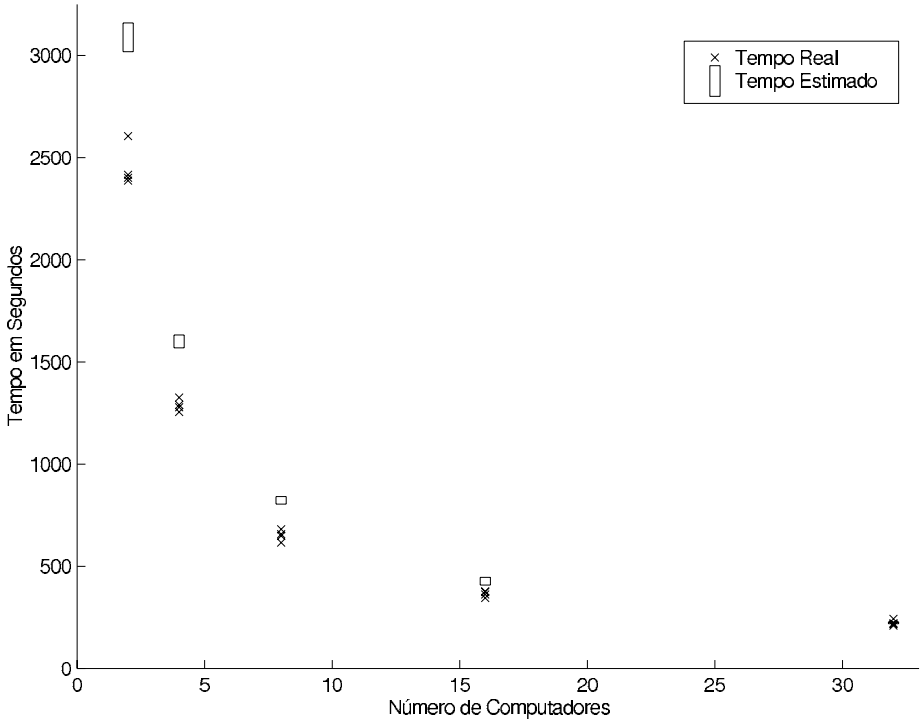

Fig. C.13: Tempo para operações parciais em matrizes: versão $rand_B$

C.5.3 Versão $rand_D$

Os resultados obtidos com os experimentos desta versão são apresentados na Tab. C.15. Nela, todas as estimativas apresentaram erros bastante elevados, nunca inferiores a 9% e que chegaram a atingir 33,4% em relação aos tempos de execução. Em função disso, as médias de erro para diferentes valores de P permaneceram elevados. Com 32 computadores verificou-se a menor de todas, que foi de $\overline{E}_{32} = 13,93\%$. De forma contrária, com 2 computadores foi registrada a maior média, que atingiu um valor bastante elevado, igual a $\overline{E}_2 = 30,39\%$.

No gráfico da Fig. C.15 são ilustrados os tempos de execução e as estimativas da versão $rand_D$. Nota-se que para 2, 4 e 8 computadores a diferença entre eles foi grande. Em conseqüência disso, os erros também assumiram valores elevados. Com 16 e 32 computadores, apesar das estimativas estarem próximas dos intervalos calculados, os erros foram grandes porque os intervalos para essas quantidades de computadores foram pequenos. Outro dado importante e que é visível no gráfico, é que os tempos de execução, em todos os casos, apresentaram uma variação muito pequena para um mesmo valor de P, e acompanharam parcialmente a variação dos intervalos, pois a medida que estes diminuíam, os tempos de execução sofreram menores variações.

Tab. C.15: Resultados da aplicação que realiza operações parciais em matrizes: versão $rand_D$

Operações parciais em matrizes: versão $rand_D$					
P	i	T_X **(seg)**	T_E **(seg)**	$E_{p,i}$ (%)	$\overline{E_p}$ (%)
2	1	3955	[6037;6419]	33,43	30,39
	2	4147		30,35	
	3	4295		27,97	
	4	4179		29,83	
4	1	2217	[3115;3259]	28,18	25,40
	2	2340		24,32	
	3	2290		25,89	
	4	2375		23,22	
8	1	1310	[1616;1702]	18,44	20,33
	2	1233		23,09	
	3	1290		19,65	
	4	1282		20,13	
16	1	719	[847;883]	14,80	17,08
	2	738		12,60	
	3	681		19,19	
	4	659		21,73	
32	1	396	[438;456]	9,40	13,93
	2	354		18,79	
	3	375		14,09	
	4	378		13,42	

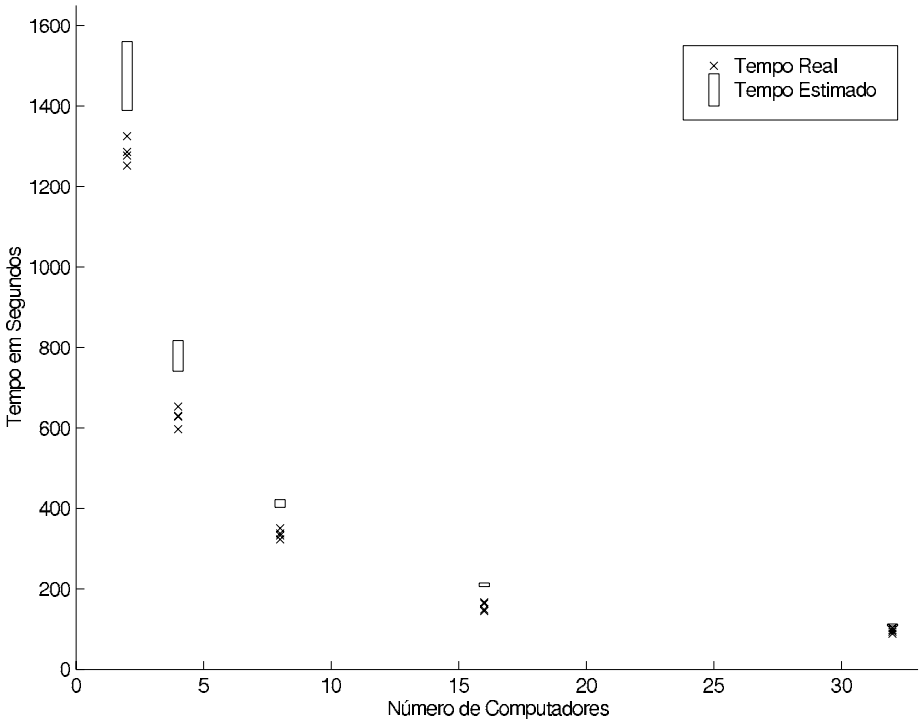

Fig. C.14: Tempo para operações parciais em matrizes: versão $rand_C$

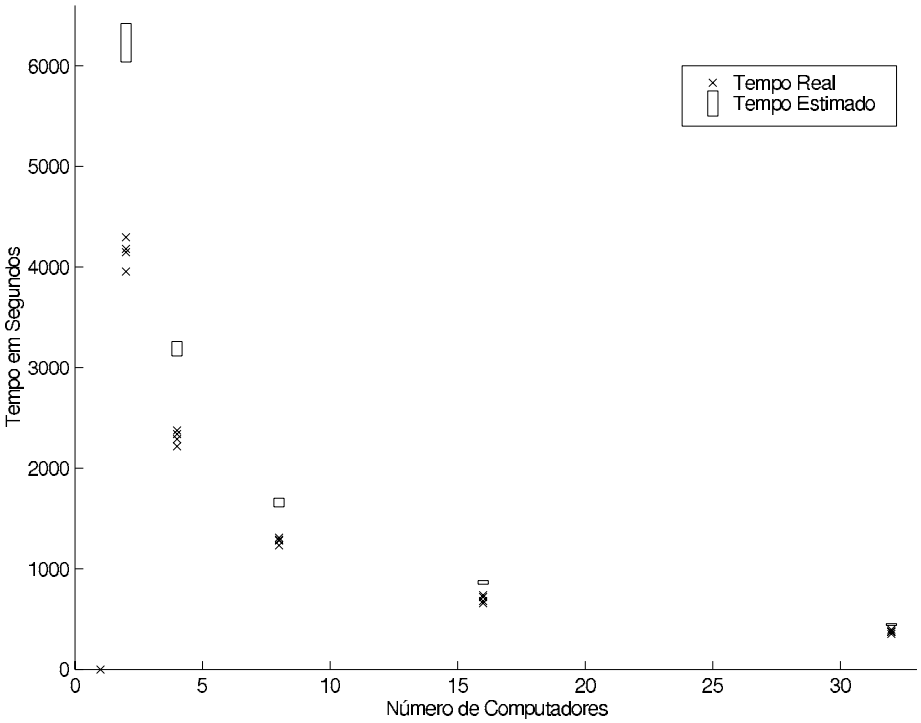

Fig. C.15: Tempo para operações parciais em matrizes: versão $rand_D$