Thiago Marques Soares

HCLogP: Um Modelo Computacional para Clusters Heterogêneos

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional, da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre em Modelagem Computacional.

Orientador: Prof. D.Sc. Marcelo Lobosco

Coorientador: Prof. D.Sc. Rodrigo Weber dos Santos

Juiz de Fora

2017

Thiago Marques Soares,

HCLogP: Um Modelo Computacional para Clusters Heterogêneos/ Thiago Marques Soares. – Juiz de Fora: UFJF/MMC, 2017.

XIV, 98 p.: il.; 29, 7cm.

Orientador: Marcelo Lobosco

Coorientador: Rodrigo Weber dos Santos

Dissertação (mestrado) – UFJF/MMC/Programa de Modelagem Computacional, 2017.

Referências Bibliográficas: p. 94 – 98.

1. Modelos Paralelos. 2. Agregados de Computadores. 3. Ambientes Heterogêneos de Computação. 4. Escalonador. I. , Marcelo Lobosco *et al.*. II. Universidade Federal de Juiz de Fora, MMC, Programa de Modelagem Computacional.

Thiago Marques Soares

HCLogP: Um Modelo Computacional para Clusters Heterogêneos

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional, da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre em Modelagem Computacional.

Aprovada em 9 de Março de 2017.

BANCA EXAMINADORA

Prof. D.Sc. Marcelo Lobosco - Orientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Rodrigo Weber dos Santos - Coorientador
Universidade Federal de Juiz de Fora

Prof. Ph.D. Eugene Francis Vinod Rebello
Universidade Federal Fluminense

Prof. Ph.D. Maria Cristina Silva Boeres
Universidade Federal Fluminense

Dedico este trabalho aos meus pais.

AGRADECIMENTOS

Aos meus pais e parentes, pelo apoio e pela paciência ao longo dos anos.

Aos orientadores Marcelo Lobosco e Rodrigo Weber dos Santos, pela orientação, paciência e apoio oferecido desde o princípio.

Aos meus amigos de curso: Igor Russo, Bruno Marques, Marcelo Marques e Victor Patrocínio, pessoas que tornaram esses dois anos de mestrado divertidos.

Ao colega Tiago Marques do Nascimento pelo código híbrido do HIS, que foi de grande auxílio para este trabalho.

Aos Professores do Programa de Pós Graduação em Modelagem Computacional pelos seus ensinamentos e a todos aqueles que, durante esses dois anos, contribuíram de algum modo para a elaboração deste trabalho.

Finalmente, gostaria de agradecer à CAPES pela bolsa de estudos.

"Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito. Não sou o que deveria ser, mas Graças a Deus, não sou o que era antes".

Martin Luther King

RESUMO

O modelo LogP foi desenvolvido em 1993 para medir os efeitos da latência de comunicação, ocupação dos processadores e banda passante em multiprocessadores com memória distribuída. A ideia era caracterizar multiprocessadores de memória distribuída usando estes parâmetros chave, analisando seus impactos no desempenho. Este trabalho propõe um novo modelo, baseado no LogP, que descreve a influência destes parâmetros no desempenho de aplicações regulares executadas em um agregado (*cluster*) de computadores heterogêneos. O modelo considera que um agregado heterogêneo é composto por diferentes tipos de processadores, aceleradores e controladores de rede. Os resultados mostram que o pior erro nas estimativas feitas pelo modelo para o tempo de execução paralelo foi de $19,2\%$, e, em muitos casos, a execução estimada foi igual ou próxima do tempo real. Além disso, com base neste modelo, foi desenvolvido um escalonador, que baseado nas características da aplicação e do ambiente, escolhe um subconjunto de componentes que minimizem o tempo total de execução paralelo. O escalonador obteve êxito na escolha da melhor configuração para a execução de aplicações com diferentes comportamentos.

Palavras-chave: Modelos Paralelos. Agregados de Computadores. Ambientes Heterogêneos de Computação. Escalonador.

ABSTRACT

The LogP model was proposed in 1993 to measure the effects of communication latency, processor occupancy and bandwidth in distributed memory multiprocessors. The idea was to characterize distributed memory multiprocessor using these key parameters and study their impact on performance in simulation environments. This work proposes a new model, based on LogP, that describes the impacts on performance of regular applications executing on a heterogeneous cluster. The model considers that a heterogeneous cluster is composed of distinct types of processors, accelerators and networks. The results show that the worst error in the estimations of the parallel execution time was about $19,2\%$, and, in many cases, the estimated execution time is equal to or very close to the real one. In addition, based on this model, a scheduler was developed. Based on the applications and computational environment characteristics, the scheduler chooses the subset of processors, accelerators and networks that minimize the parallel execution time. For applications with different behaviors, the scheduler successfully chose the best configuration.

Keywords: Parallel Models. Cluster. Heterogeneous computing. Scheduler.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 Introdução

1.1 Motivação

Simulações científicas de larga escala exigem o uso de sistemas de alto desempenho, que são projetados para lidar com o processamento de uma enorme quantidade de dados. A mais representativa arquitetura dedicada a resolver esta tarefa é o agregado (*cluster*) de computadores. Devido ao rápido desenvolvimento e adoção de novas tecnologias, este tipo de arquitetura tem se tornado cada vez mais heterogênea, combinando, em um único sistema, diferentes processadores, aceleradores, como as unidades de processamento gráfico (GPUs), e controladores de rede. Do ponto de vista econômico, manter os sistemas antigos enquanto novos são incorporados faz sentido. No entanto, tanto da perspectiva do usuário quanto do administrador, lidar com sistemas heterogêneos não é uma tarefa simples. Para um usuário que simula aplicações nesse ambiente utilizando paralelismo de dados, ou seja, que divide os dados para serem processados pelas distintas unidades de processamento, o desafio é manter a eficiência computacional, tendo em vista que o distinto poder computacional dos processadores e aceleradores e as distintas latências e bandas passante das controladoras de rede podem resultar em um desequilíbrio na computação e em atrasos no processo de sincronização. Para os administradores, o desafio é gerenciar de maneira eficiente os diferentes recursos computacionais, de modo que o máximo de usuários possa usufruir destes recursos.

Neste contexto, a estimativa correta do tempo de execução de uma aplicação paralela em um ambiente heterogêneo pode impactar positivamente usuários e administradores de sistema. Com um método que permita realizar, com margem razoável de erro, tal predição, pode-se, por exemplo, escolher *a priori* um conjunto de recursos (processadores, aceleradores e redes) que minimize o tempo de execução de uma aplicação ou aumente a vazão do sistema computacional.

1.2 Objetivos

Os objetivos deste trabalho são a) definir um modelo que permita estimar o tempo total de execução de uma aplicação paralela regular (aplicações em que o tempo de computação para cada iteração no laço paralelo não difere [1]) executada em ambientes de agregados computacionais heterogêneos, e b) utilizar este modelo para implementar um algoritmo escalonador que prediz qual o subconjunto de recursos computacionais (processadores, aceleradores e redes de comunicação) deve ser utilizado para que a aplicação complete sua execução no menor tempo possível.

O modelo proposto estende algumas características do modelo LogP [2], mas considerando que as unidades de processamento possuem poder de processamento distinto e que estas unidades estão interconectadas por controladores de redes com diferentes latências. Considerando as características do ambiente, a ideia é avaliar se o modelo consegue capturar o comportamento de diferentes tipo de aplicações, que podem ter uma predominância de comunicação entre processos (tempo de comunicação), operações lógicas e aritméticas (tempo de computação) ou equilíbrio entre estas duas possibilidades.

Neste trabalho o termo escalonador refere-se a um algoritmo ou otimizador que tem como objetivo encontrar a melhor configuração de recursos dentro de um ambiente heterogêneo, de forma automática, que minimize o tempo de execução de uma aplicação. Dependendo do número e da configuração dos nós e dos controladores de rede esta pode não ser uma tarefa trivial, sendo necessários algoritmos específicos para resolver este tipo de problema.

1.3 Método

Com a finalidade de atingir os objetivos deste trabalho, foi criado um novo modelo computacional, baseado no LogP [2], que estima o tempo de execução total de uma aplicação paralela. O tempo total de execução foi dividido em dois componentes, o tempo de computação e o tempo de comunicação. Para se obter os parâmetros do tempo de computação foram utilizados *benchmarks* para cada tipo de unidade de processamento presente no ambiente. Como as operações que constituem o tempo de comunicação podem variar de acordo com o seu tipo e aplicação, os parâmetros para o tempo de comunicação foram obtidos através de *benchmarks* e de uma análise do código.

Com todos os parâmetros obtidos, a próxima etapa foi validar o modelo com uma diversidade de aplicações paralelas em um ambiente heterogêneo de agregados computacionais. O *benchmark* NAS [3] foi escolhido para esta etapa, pois ele fornece uma gama de aplicações com diferentes características em relação aos tempos de computação e comunicação. Como estas aplicações foram desenvolvidas para serem executadas somente por CPUs, outra aplicação, que simula o sistema imune humano (HIS) [4], foi adotada para avaliar o comportamento do modelo quanto ao uso somente de GPUs e de ambas CPUs e GPUs simultaneamente.

Finalmente foi implementado um escalonador, baseado no modelo desenvolvido, que quando alimentado com as características do ambiente e da aplicação, retorna a melhor configuração de recursos computacionais que minimiza o tempo de execução da aplicação. Todas as aplicações empregadas na etapa anterior foram novamente utilizadas para verificar o comportamento do escalonador.

1.4 Organização

Este trabalho está organizado da seguinte forma: O Capítulo 2 apresenta todos os modelos que serviram de inspiração para a criação do modelo proposto por este trabalho. O Capítulo 3 apresenta o modelo computacional proposto. O Capítulo 4 descreve os passos do algoritmo desenvolvido para encontrar a melhor configuração para a execução de uma aplicação paralela em um ambiente de agregados computacionais. A validação do modelo e os resultados obtidos pelo escalonador para diversas aplicações estão presentes no Capítulo 5. Concluindo este trabalho, o Capítulo 6 traz as considerações finais e trabalhos futuros.

2 Trabalhos Relacionados

2.1 Trabalhos Correlatos ao Modelo HCLogP

2.1.1 LogP

O modelo LogP [2] foi proposto para medir os efeitos da latência de comunicação, taxa de ocupação dos processadores e banda passante em um ambiente de processadores com memória distribuída. Os processadores se comunicam através de uma rede de comunicação utilizando mensagens ponto-a-ponto. A comunicação acontece de forma não coordenada, onde cada unidade de processamento realiza sua comunicação de forma independente. O modelo considera que ambos, rede e processadores, são homogêneos. Os parâmetros do modelo são os seguintes:

a) Latência (**L**), que representa um limite superior da latência ou atraso na troca de mensagens ponto-a-ponto,

b) *Overhead* (**o**), tempo que o processador emprega para transmitir ou receber uma mensagem, não podendo ser usado para executar outra instrução,

c) *Gap* (**g**), que representa o tempo mínimo entre envios ou recebimentos de mensagens consecutivas em um processador, e

d) **P**, número de processadores ou elementos de processamento com sua memória local.

A figura 2.1 mostra uma comunicação onde são trocadas duas mensagens ponto-a-ponto entre dois processos. O processo P_0 inicia o processo de envio da primeira mensagem e, após o tempo **o**, o primeiro *byte* é enviado através da rede de comunicação para o processo P_1. Após um intervalo de tempo L, a primeira mensagem chega ao processo P_1, que por sua vez gasta um tempo **o** para realizar o procedimento para recebê-la. Portanto a primeira mensagem fica disponível para P_1 no tempo de $o + L + o$. Somente após o intervalo g que todo o envio (no processo P_0) e recebimento (no processo P_1) da primeira mensagem é finalizado, de modo que uma segunda mensagem possa ser enviada.

O modelo considera que a rede de comunicação tem uma capacidade finita para envio de mensagens, e que no máximo $[L/g]$ mensagens podem transitar simultaneamente entre

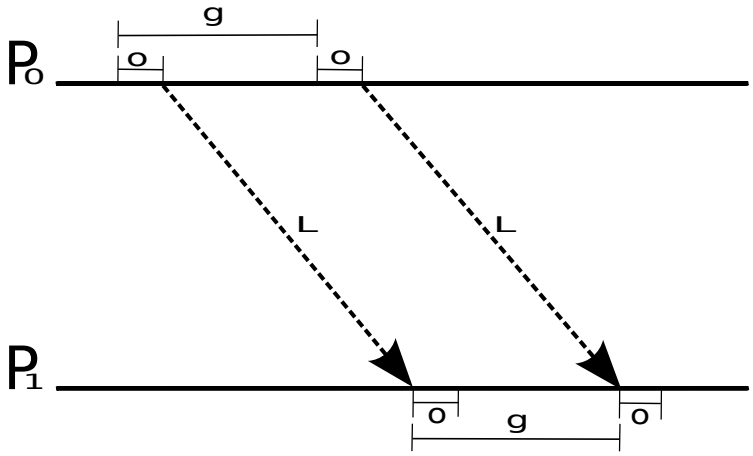

Figura 2.1: Esquema de comunicação do modelo LogP.

qualquer par de processos em qualquer intervalo de tempo. Caso algum processo tente enviar uma mensagem enquanto a capacidade da rede está no máximo, ele é bloqueado até que a mensagem possa ser enviada sem exceder a capacidade da rede. O modelo também considera que todas as mensagens são iguais e de tamanhos pequenos.

2.1.2 LogGP

O modelo LogP funciona bem em predizer o tempo de comunicação quando apenas mensagens pequenas e de tamanho fixo são consideradas. Para resolver este problema o modelo LogGP [5] foi proposto. Este modelo estende o modelo LogP introduzindo um novo parâmetro **G**, que representa o *gap* por *byte* para mensagens maiores. O inverso de G caracteriza a banda passante disponível por processador para mensagens longas.

Para mensagens pequenas o modelo funciona como o modelo LogP, enquanto que para mensagens maiores cada *byte* é enviado consumindo um tempo G. Desta forma, como mostra a figura 2.2, uma mensagem de tamanho k *bytes* demora $o + (k-1) \times G + L + o$ para ser enviada.

Assim como no modelo LogP, este modelo permite a sobreposição de comunicação com computação, uma vez que as operações de comunicação são decompostas em vários passos independentes. Ao considerar pequenas e grandes mensagens, o modelo permite realizar predições mais realistas das aplicações. O modelo também considera que ambos, rede e processadores, são homogêneos.

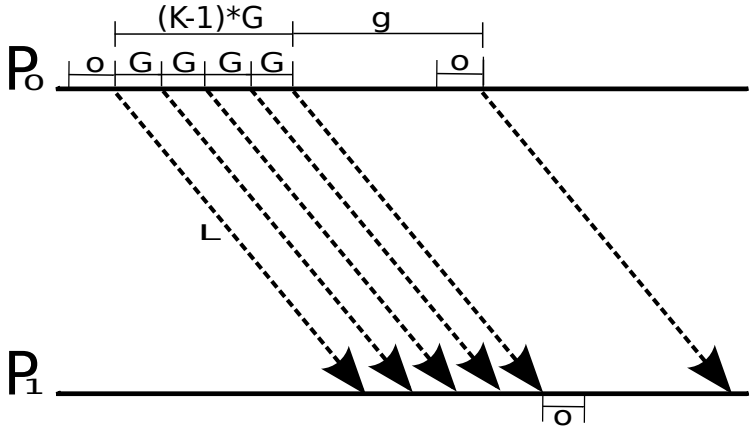

Figura 2.2: Esquema de comunicação do modelo LogGP.

2.1.3 PLogP

O modelo PLogP [6] (modelo LogP parametrizado) define cinco parâmetros, dos quais três deles são funções lineares por partes em relação ao tamanho da mensagem (**m**). O *overhead* foi divido em *overhead* de envio e de recebimento, e o significado dos parâmetros foram minimamente alterados em comparação ao modelo LogP. Assim os seguintes parâmetros são apresentados:

a) Latência (**L**), constante que representa o atraso ponto-a-ponto de processo para processo, combinando todos os fatores como cópia de dados de/para interfaces de rede e a transferência para a camada física da rede;

b) *Overhead* de envio ($\boldsymbol{o_s(m)}$), tempo que o processo passa enviando a mensagem;

c) *Overhead* de recebimento ($\boldsymbol{o_r(m)}$), tempo que o processo passa recebendo a mensagem;

d) *Gap* (**g(m)**), que representa o tempo mínimo entre envios ou recebimentos de mensagens consecutivas em um processador. Seu inverso representa a banda passante ponto-a-ponto entre dois processos para uma dada mensagem de tamanho m. No modelo se assume que o *gap* domina o *overhead*: $g(m) \geq o_s(m)$ e $g(m) \geq o_r(m)$;

e) **P**, número de processadores.

A figura 2.3 ilustra como os parâmetros são utilizados. Foram introduzidos os parâmetros $s(m)$ e $r(m)$ que representam respectivamente os tempos para envio e

recebimento de uma mensagem de tamanho m quando ambos, *sender* e *receiver* iniciam suas operações. O *sender* estará pronto para mandar a próxima mensagem no tempo $s(m) = g(m)$. Como $g(m)$ modela o tempo em que a mensagem ocupa a rede de comunicação, a próxima mensagem não pode ser enviada ou recebida antes de $g(m)$. O tempo em que o *receiver* recebe a mensagem é dado por $r(m) = L + g(m)$. A latência (L) é o tempo que demora para o primeiro *bit* da mensagem ser transferido do *sender* para o *receiver*. O gap da mensagem ($g(m)$) adiciona o tempo entre o recebimento do primeiro e do último *bit* da mensagem. Quando o *sender* envia várias mensagens consecutivas, a latência vai contribuir uma única vez para o tempo total de transferência, mas o *gap* de todas as mensagens é somado. Esta operação pode ser expressa por $r(m_1, ..., r(m_n)) = L + g(m_1 + ... + g(m_n))$.

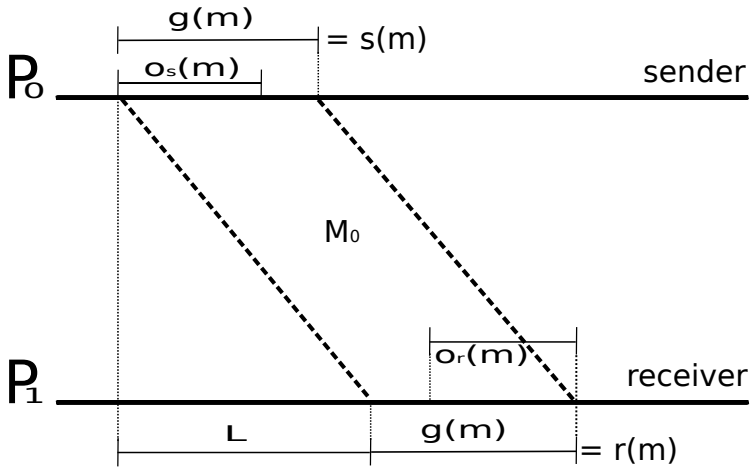

Figura 2.3: Esquema de comunicação do modelo PLogP.

Os autores do modelo PLogP também desenvolveram a biblioteca *logp_mpi*, que é utilizada para mensurar os valores dos parâmetros usados por modelos inspirados no LogP. Esta biblioteca foi implementada como uma aplicação MPI que utiliza mensagens de ida e volta (*roundtrip*) para determinar esses parâmetros sem que seja necessário saturar a rede com mensagens longas. Somente para mensurar o *gap* de mensagens vazias, com zero *bytes* de dados, que a rede deve ser saturada. Outra desvantagem desta biblioteca é que as medidas devem ser feitas com diferentes tamanhos de mensagens, o que implica que um tempo maior é necessário para realizar a coleta dos dados.

Assim como ocorre nos modelos LogP e LogGP, PLogP considera que todas as unidades de processamento e redes de comunicação são homogêneas.

2.1.4 Um modelo acurado para cluster heterogêneo

Para trabalhar com processadores heterogêneos conectados por uma rede Ethernet, Lastvetsky *et al.* [7] estenderam o modelo LogP com o objetivo de verificar o impacto da comunicação no tempo total de execução de uma aplicação.

O modelo é estruturado da seguinte forma: O tempo total de comunicação T, considerando uma mensagem de tamanho M, é composto pelo atraso da transmissão T_{net}, que é o tempo que demora para a mensagem atravessar a rede de comunicação, o atraso no nó de envio L_i, e o atraso no nó destino L_j. No atraso no nó de envio está incluso todo o pré-processamento da mensagem, como por exemplo, a construção do cabeçalho da mensagem e a cópia da mensagem para a camada de rede, portanto dependente do tamanho M da mensagem. Com isso as duas equações abaixo são sumarizadas:

$$T = L_i + L_j + T_{net}, \tag{2.1}$$

$$L_i(M) = C_i + t_i M, \tag{2.2}$$

onde C_i e t_i são respectivamente o tempo para pré-processamento da mensagem e o tempo para processar um *byte* da mensagem no nó de envio i. Analogamente temos que o atraso no nó destino é dado por $L_j(M) = C_j + t_j M$. O atraso na transmissão é dado por:

$$T_{net}(M) = M/\beta_{i,j}, \tag{2.3}$$

onde $\beta_{i,j}$ é a taxa de transmissão da rede conectando os nós i e j.

Com isso temos que o tempo de comunicação ponto-a-ponto em um ambiente heterogêneo do nó i para o nó j é expresso por:

$$T = C_i + t_i M + C_j + t_j M + M/\beta_{i,j}. \tag{2.4}$$

Outros padrões de comunicação também são precisamente representados: um para muitos, muitos para um e muitos para muitos. Em um trabalho posterior Lastvetsky *et al.* [8] também propuseram uma técnica para se estimar os parâmetros de seu modelo.

Em comparação com os modelos PLogP e LogP, este modelo está sendo mais detalhista

mas não necessariamente está propondo algo novo, pois a maioria dos parâmetros já foram propostos anteriormente.

O modelo sugerido nesta dissertação tem como objetivo não somente prever o tempo de comunicação, como também o tempo de computação de aplicações paralelas.

2.1.5 HLoGP

Todos os modelos descritos acima não foram desenvolvidos para lidar com agregados computacionais que misturam diferentes CPUs e aceleradores gráficos, assim como diferentes tipos de redes localizadas no mesmo ambiente.

O HLoGP [9] é um modelo computacional paralelo baseado no LogGP, que captura essas características heterogêneas e possui alta acurácia. A alta precisão do modelo é resultado da substituição dos valores escalares por matrizes e vetores de parâmetros que refletem as diferentes particularidades dos nós de computação. O modelo emprega os seguintes parâmetros:

a) Latência (**L**): onde uma matriz quadrada é definida como $L = \{l_{1,1}, ..., l_{M,M}\}$ em que cada elemento $l_{i,j}$, é associado a latência de uma mensagem enviada do nó N_i para o N_j.

b) *Overhead* (**o**): devido à heterogeneidade do sistema, o *overhead* depende do poder computacional de cada nó. Então ele foi decomposto em um vetor de *overhead* de envio $O_s = \{os_1, ..., os_M\}$, em que cada elemento i representa o *overhead* de envio do nó N_i, e em um vetor de *overhead* de recebimento $O_r = \{or_1, ..., or_M\}$.

c) *Gap* entre as mensagens, (**g**): Semelhante ao *overhead*, este valor depende da interface de rede de cada nó, logo um vetor de *gap* é definido como $g = \{g_1, ..., g_M\}$, onde g_i representa o *gap* do nó i para mensagens de 1 *byte*.

d) *Gap* por *byte* (**G**): O inverso deste parâmetro determina a banda passante envolvida na troca de mensagens entre um par de nós. Desta forma uma matriz de *Gap* é definida como $G = \{G_{1,1}, ..., G_{M,M}\}$, onde cada elemento $G_{i,j}$ representa o *gap* por *byte* de uma mensagem longa enviada do nó N_i para o N_j. O custo de comunicação entre dois nós, englobando rede heterogêneas, deve ser feito com base na menor banda passante dentre as redes envolvidas, pois o *gap* por *byte* total depende do caminho percorrido pela mensagem nos roteadores.

e) Poder Computacional (**P**): Como cada nó pode apresentar características diferentes, um vetor de poder computacional $P_s = \{P_1, ..., P_M\}$ é definido, em que cada elemento P_i corresponde ao poder computacional do nó N_i.

Os vetores e matrizes referentes aos parâmetros podem ser preenchidos através da execução de simples *benchmarks*, como proposto pelos autores do trabalho, que medem o tempo de operações de envio e recebimento entre todos os processos.

No *cluster* onde o modelo foi validado e para a aplicação utilizada para avaliar o modelo, percebeu-se que toda vez que uma mensagem estava pronta para a transmissão, a última mensagem enviada já tinha sido recebida, independente do caminho percorrido. Como consequência, o parâmetro *gap* entre mensagens (g) pode ser ignorado sem afetar a acurácia do modelo.

De acordo com o modelo, o envio de uma mensagem de k *bytes* tem o custo inicial de Os_i no envio do primeiro *byte* para a rede. Envios subsequentes demoram $G_{i,j}$ ciclos. O último *byte* é enviado no tempo $Os_i + (k-1) \times G_{i,j}$. Cada *byte* percorre a rede por L ciclos e finalmente o processo receptor demora Or_j ciclos para receber a mensagem. Portanto o tempo total T para enviar e receber uma mensagem é dado por 2.5:

$$T = Os_i + (k-1) \times G_{i,j} + L_{i,j} + Or_j. \tag{2.5}$$

Os experimentos foram realizados levando em consideração um ambiente heterogêneo e uma aplicação que realiza a compressão de uma imagem volumétrica de ressonância magnética em 6 estágios. Os resultados experimentais mostram um erro percentual entre o tempo estimado e o tempo real de 8,3% no pior caso (primeiro estágio), e de 0,5% no melhor caso (quinto estágio). O erro percentual para o tempo total de execução foi menor que 1%. Estes valores mostram uma alta acurácia do modelo em predizer o tempo de execução de uma aplicação, considerando um ambiente heterogêneo em relação ao poder computacional e comunicação.

No entanto, para que a alta precisão do modelo seja atingida, é necessário realizar a estimativa de um elevado número de parâmetros. Além disso, no inicio da execução da aplicação é preciso realizar uma divisão da carga de trabalho levando em conta o poder computacional de cada nó.

2.1.6 Outros modelos

O modelo de Hockney [12] é o mais simples em termos de custo de comunicação ponto-a-ponto, que é definida como $\alpha + \beta m$, onde α é a latência, β é a taxa de transmissão da rede e m o tamanho da mensagem. O inverso de β é a banda passante do controlador de comunicação e os parâmetros do modelo podem ser medidos diretamente por testes ponto-a-ponto. Baseado nos modelos Hockney, LogP e PLogP, Pjesivac-Grbovic [13] estimou o tempo de comunicação para operações do tipo barreira, redução e muitos para muitos implementadas com diferentes algoritmos e executadas em distintas topologias de rede. Boas predições foram obtidas quando comparadas com os tempos de execução reais.

Outros trabalhos focaram no estudo do impacto da comunicação na execução de uma aplicação paralela. Holt *et al.* [14] utilizaram uma vasta gama de aplicações com o objetivo de avaliar os efeitos de alguns dos parâmetros do modelo LogP em uma máquina de memória compartilhada distribuída (*Distributed Shared Memory* - DSM). Os resultados dos experimentos mostraram que a taxa de ocupação no controlador da comunicação é vital para se conseguir um bom desempenho neste tipo de máquina. Os autores argumentam que modelar a contenção analiticamente é uma tarefa difícil, especialmente em aplicações que tentam esconder a latência. Por esta razão o modelo desenvolvido neste trabalho não se utiliza de um parâmetro específico para descrever os efeitos de contenção de recursos. Consideramos que este efeito é capturado indiretamente pelo *overhead*. Embora algumas aplicações demonstrem uma forte sensibilidade ao *overhead*, como Martin *et al.*[15] mostraram, um dos objetivos deste trabalho é de manter o modelo o mais simples possível, como será descrito no próximo capítulo.

O HLogP [16] foi desenvolvido para ambientes em Grid. Trata-se de um modelo em que o objetivo é capturar as características dos recursos de sistemas distribuídos heterogêneos e não detalhes da arquitetura, tais como topologias de redes de interconexão e características específicas dos *hosts*. Os parâmetros são os mesmos do modelo LogP, em que uma variação é considerada em função do tamanho das mensagens e de quais unidades computacionais se comunicam.

O modelo τ-Lop [17] é um modelo formal que estima o custo de comunicação de operações coletivas em ambientes de memória compartilhada, com foco nas operações definidas pelo padrão MPI. O modelo supera as limitações dos modelos que foram suas inspirações, o log_nP [18] e o $mlog_nP$ [19]. A principal contribuição foi poder estimar

operações concorrentes nos canais que são compartilhados por todos os processadores envolvidos na comunicação. Seguidamente o modelo é estendido para predizer a comunicação em *clusters* com multiprocessadores [20]. O novo modelo τ-Lop oferece uma visão mais global do algoritmo se baseando no conceito de transferências concorrentes e captura a diminuição da banda passante nos canais compartilhados quando várias transmissões estão ocorrendo em paralelo, diminuindo o erro da estimativa do custo de comunicação.

Todos os modelos anteriores podem não funcionar perfeitamente quando considerados:

a) a estimativa do tempo total de execução da aplicação, composto pelo tempo de computação e o tempo de comunicação;

b) *clusters* heterogêneos, que apresentam diferentes CPUs e aceleradores gráficos; e

c) redes heterogêneas.

Apesar do modelo HLogP[9] ser bem acurado para *cluster* heterogêneos, o alto número de parâmetros e o balanceamento de carga necessário para as estimativas do modelo podem ser um problema. O modelo proposto neste trabalho é mais simples e prediz o tempo de execução de uma aplicação paralela independente do ambiente ser homogêneo ou heterogêneo.

2.2 Trabalhos correlatos ao Algoritmo Escalonador

Um trabalho que incorpora as técnicas de predição de execução de uma aplicação paralela e posteriormente usa esta informação para que a execução desta aplicação seja otimizada é o PACE [21]. Este trabalho é parte de um esforço para desenvolver uma análise de desempenho e caracterização do ambiente (*performance analysis and characterization environment* - PACE), que tem como objetivo prover informação a respeito da comunicação, desempenho de execução e escalabilidade de processos sendo executados em sistemas de alto desempenho.

A abordagem adotada pelo PACE não depende da obtenção de dados de aplicações específicas sendo executadas em máquinas específicas; sua principal característica é trabalhar com a análise de informações preditivas, sendo elas: a) tempo de execução; b) *design* de sistemas; c) escalabilidade; e d) estratégias de paralelização. O processo

de avaliação para gerar a informação que será utilizada na etapa de predição é baseado em eventos. Toda vez que uma carga de trabalho é enviada para um processador um evento é criado, e cada modelo de *hardware* produz uma predição de tempo baseada nos parâmetros do evento. O resultado é então gravado em um lista de eventos, que depois de analisada pode predizer o tempo de execução da aplicação.

Resultados mostram que a ferramenta PACE-A [21], desenvolvida utilizando este método, é capaz de produzir informações detalhadas sobre escalabilidade, causas de gargalos de execução e rastreamento de dados. A velocidade com que estas informações são obtidas também permite a otimização dinâmica de aplicações. Exemplos mostraram que tanto a seleção de um algoritmo de otimização quanto de sistemas computacionais podem ser feitas em tempo de execução. A maior desvantagem de se utilizar esta ferramenta é que um novo método deve ser aprendido para que a ferramenta seja incorporada na aplicação paralela.

Um outro *framework* [22] foi desenvolvido com o objetivo de medir e modelar características estáticas e dinâmicas de aplicações e, com base nos resultados, melhorar a interação entre a aplicação e o sistema em que ela será executada. O modelo utilizado tem como foco principal modelar a aplicação independente de qualquer detalhe arquitetural, desta forma um método semi-automático pode ser criado para realizar novas estimativas para novos parâmetros de entrada. O trabalho prediz o tempo de execução da aplicação considerando o tempo de execução de suas instruções e os acessos realizados ao longo da hierarquia de memória, permitindo assim estimar o tempo total de execução da aplicação paralela independente se ela apresenta como característica utilizar mais a CPU, a memória ou ambas. Em relação à hierarquia de memória, o *framework* foi capaz de predizer custos decorrentes de *cache miss*. No entanto o *framework* só funciona em um único nó computacional, ou seja, o *framework* não modela os custos de comunicação relativos a troca de mensagens por um controlador de rede.

Outras referências possuem objetivos principais que divergem um pouco do objetivo do escalonador desenvolvido neste trabalho. Um deles incorpora estimativas acuradas a um escalonador de tarefas em um sistema operacional que utiliza *backfilling* [23]. *Backfilling* é uma técnica que permite que tarefas que são executadas rapidamente passem à frente na fila de tarefas, contanto que o usuário forneça um valor de estimativa para seu tempo de execução. Caso a tarefa tenha um tempo de execução maior que a estimativa do usuário,

a tarefa é excluída da fila de execução.

Para ambientes formados por grades computacionais, uma estimativa baseada no tempo de transferência de arquivo, tempo de espera na fila do escalonador e tempo de execução da aplicação foi desenvolvido [24]. Um algoritmo genético é utilizado para encontrar os valores ótimos para os parâmetros do modelo, testando entre os dados pré-processados diferentes configurações que minimizem o erro da estimativa.

2.3 Resumo do capítulo

Neste capítulo foram apresentados alguns dos principais modelos que seguem a mesma linha de pesquisa e compartilham de um objetivo semelhante ao modelo proposto neste trabalho. Um dos primeiros modelos a surgir foi o modelo LogP [2] e a maioria dos trabalhos posteriores são melhorias e alterações deste modelo. O modelo HCLogP, proposto nesta dissertação, também baseia sua estimativa do tempo de comunicação no modelo LogP. No entanto, o foco é tornar o modelo mais genérico, de forma que resultados acurados possam ser obtidos para ambientes heterogêneos (semelhante ao modelo HLoGP [9]), constituídos por diferentes controladores de rede e arquiteturas computacionais (CPUs e GPUs). A tabela 2.1 apresenta um sumário de todos os parâmetros dos modelos que serviram de base para o modelo HCLogP.

Em relação ao algoritmo escalonador, a principal referência utilizada foi o PACE-A [21], ferramenta capaz de coletar informações detalhadas sobre escalabilidade e causas de gargalos de execução de aplicações. O escalonador proposto neste trabalho compartilha um dos objetivos do PACE-A, a minimização do tempo de execução de uma aplicação em um dado ambiente computacional.

Tabela 2.1: Sumário dos parâmetros de todos os modelos apresentados.

LogP	
Latência (**L**)	Atraso na troca de mensagens ponto-a-ponto.
Overhead (**o**)	Tempo que o processador emprega para transmitir/receber uma mensagem.
Gap (**g**)	Tempo mínimo entre envios/recebimentos de mensagens consecutivas em um processador.
P	Número de processadores.
LogGP	
Latência (**L**)	Atraso na troca de mensagens ponto-a-ponto.
Overhead (**o**)	Tempo que o processador emprega para transmitir/receber uma mensagem.
Gap (**g**)	Tempo mínimo entre envios/recebimentos de mensagens consecutivas em um processador.
P	Número de processadores.
Gap per Byte (**G**)	Tempo associado ao envio de cada *byte* de uma mensagem.
PLogP	
Latência (**L**)	Constante que representa o atraso ponto-a-ponto no envio de mensagem entre pares de processadores.
Overhead de envio ($\boldsymbol{o_s(m)}$)	Tempo que o processador passa enviando a mensagem.
Overhead de recebimento ($\boldsymbol{o_r(m)}$)	Tempo que o processo passa recebendo a mensagem.
Gap (**g(m)**)	Tempo mínimo entre envios ou recebimentos de mensagens consecutivas em um processador.
P	Número de processadores.
HLoGP	
Latência (**L**)	Matriz onde cada elemento é a latência de uma mensagem enviada do nó N_i para o N_j.
Overhead de envio ($\boldsymbol{o_s}$)	Vetor em que cada elemento representa o *overhead* de envio do nó N_i.
Overhead de recebimento ($\boldsymbol{o_r}$)	Vetor em que cada elemento representa o *overhead* de recebimento do nó N_i.
Gap per Byte (**G**)	Matriz onde cada elemento representa o *gap* por *byte* de uma mensagem longa enviada do nó N_i para o N_j.
Gap (**g**)	Vetor em que cada elemento representa o *gap* do nó i para mensagens de 1 *byte*.
Poder Computacional (**P**)	Vetor em que cada elemento representa o poder computacional do nó N_i.

3 HCLogP

O modelo LogP vem sendo usado como um guia para o desenvolvimento de algoritmos paralelos, dado que através dele é possível prever o comportamento de aplicações em um ambiente de processadores homogêneos de memória distribuída. O HCLogP estende, em seu modelo de comunicação, o modelo LogP, com o objetivo de prever, com uma margem moderada de erro, o tempo total de execução de uma aplicação paralela regular em ambientes modernos de *clusters* heterogêneos, composto por diferentes processadores, aceleradores e redes.

A previsão do tempo total de execução leva em conta que esse é composto por duas fases distintas: computação e comunicação. Então, para se prever o tempo de execução é necessário predizer o tempo gasto em ambas as fases. Para se atingir este propósito, o modelo introduz algumas variáveis que serão utilizadas na modelagem da fase computacional, e outras que serão usadas na fase de comunicação.

Para alcançar esse objetivo, três etapas devem ser seguidas:

a) Um modelo matemático que descreve as fases de computação e comunicação da aplicação deve ser escrito;

b) O tempo gasto pela CPU para executar uma parte pequena de passos sequenciais da aplicação deve ser coletado; e

c) Alguns parâmetros devem ser coletados do ambiente heterogêneo.

Enquanto os primeiros dois passos variam de aplicação para aplicação, o último depende somente do *hardware*, podendo assim ser obtido e armazenado para ser usado em todas as aplicações. A descrição detalhada dos passos segue abaixo.

3.1 Estimativa do tempo de computação

Os principais parâmetros utilizados pelo HCLogP para a fase de computação são:

a) $\boldsymbol{R_P}$: é o poder de computação relativo de uma unidade de processamento;

b) **P**: é o número de processos utilizados na execução da aplicação;

c) (**size**): Representa o tamanho do problema;

d) (**I**): é número total de iterações da aplicação paralela regular.

Devido ao distinto poder de processamento dos aceleradores e das CPUs, a normalização do poder de computação relativo ($\boldsymbol{R_P}$) é obtida da seguinte forma: Primeiramente, um *benchmark* é utilizado sequencialmente em cada tipo de unidade de processamento disponível no *cluster*. Por exemplo, se o ambiente é composto por duas CPUs distintas e duas GPUs distintas, o *benchmark* é executado no mínimo quatro vezes, uma em cada tipo de CPU e GPU disponível. Devido às possíveis flutuações no tempo de execução, o *benchmark* deve ser executado em cada unidade individual quantas vezes forem necessárias para se obter um baixo desvio padrão. A própria aplicação que se pretende estimar o tempo pode ser utilizada como *benchmark*, no entanto utilizando um conjunto de dados menor da aplicação original para este propósito. O valor médio do tempo de execução é então usado para normalizar o poder relativo utilizando o valor de desempenho clássico[28]:

$$\frac{\text{Desempenho}_x}{\text{Desempenho}_y} = \frac{\text{Tempo de Execução}_y}{\text{Tempo de Execução}_x}. \tag{3.1}$$

Por exemplo, suponha que a CPU_A execute o *benchmark* em $100s$; CPU_B em $50s$, GPU_A em $10s$ e GPU_B em $5s$. Então a razão de desempenho mostra que a CPU_B é 2 vezes mais rápida que a CPU_A; GPU_A é 10 vezes mais rápida e GPU_B 20 vezes. Se um *cluster* é composto por 4 CPUs_A, 8 CPUs_B, 1 GPU_A and 2 GPUs_B, então considera-se que 70 unidades de processamento estão disponíveis, das quais o poder de processamento é equivalente ao da CPU_A (outros aceleradores ou processadores podem ser escolhidos como referência). Observe que no processo de normalização, cada núcleo de CPU é considerado um único dispositivo, enquanto todos os núcleos disponíveis nos aceleradores são considerados em conjunto como um único dispositivo. O motivo do acelerador ter sido considerado como uma única unidade de processamento foi que normalmente usuários utilizam todo o poder de processamento disponível neste tipo de dispositivo. Então, se a GPU_A é escolhida para ser utilizada, 10 unidades de processamento equivalente a CPU_A serão utilizadas para estimar o tempo de computação da aplicação.

Em um trabalho anterior [26] foram propostas duas maneiras de se calcular o valor de $\mathbf{R_p}$:

a) executar o *benchmark* em cada unidade de processamento para se obter uma métrica, como por exemplo o número de unidades processadas por passo de tempo; ou

b) usar um tempo de computação médio que a unidade de processamento leva para executar uma dada quantidade de iterações sequenciais da aplicação.

No primeiro caso, dado o tamanho do problema (**size**) e o número de unidades processadas por passo de tempo que um dispositivo pode executar ($\boldsymbol{R_p}$), o tempo de computação é dado por: $\mathbf{size}/\mathbf{R_p}$. Nota-se que, neste caso o valor de $\mathbf{R_p}$ pode ser obtido ao se executar o *benchmark* somente uma vez, toda vez que um novo processador/acelerador é incluído no ambiente. No segundo caso, o tempo de computação pode ser obtido ao se multiplicar $\mathbf{R_p}$ pelo número total de iterações. O uso de algumas iterações sequenciais da aplicação em que o tempo de execução paralelo será estimado, ao invés de um *benchmark* genérico, tem a vantagem de gerar resultados mais precisos. A principal desvantagem é que é necessário executar alguns passos sequenciais da aplicação toda vez que é preciso estimar o tempo de execução paralelo da aplicação em um novo tipo de processador/acelerador. Por exemplo, ao se aumentar os tipos de unidades de processamento de dois para três, o $\mathbf{R_p}$ deve ser recalculado para três unidades de processamento. Em ambos os casos, o tempo de computação é estimado considerando que a aplicação escala linearmente. Por esta razão, neste trabalho propõe-se que se estime o tempo de computação da seguinte forma:

$$T_{\text{computação}} = \frac{I}{I_s} \times \left(\frac{I_e}{\text{Soma}_{Rp} + F_r}\right), \tag{3.2}$$

onde $\mathbf{I}$ é o número total de iterações do problema, e $\mathbf{I_s}$ é o número de iterações sequenciais que serão utilizadas para predizer o tempo de computação da aplicação. Quanto maior o valor de $\mathbf{I_s}$, mais acurados serão os valores obtidos para a predição do tempo de computação da aplicação. $\mathbf{I_e}$ é o tempo de execução de $\mathbf{I_s}$ iterações considerando o processador mais lento que será incluído na predição. O processador mais lento é usado como referência porque os processadores mais rápidos devem esperar pelo processador mais lento antes do passo de computação ser finalizado. $\text{Soma}_{\mathbf{Rp}}$ é a soma dos $\mathbf{R_p}$ de todos as unidades de processamento que serão usado na execução paralela, $\mathbf{F_r}$ é um fator de correção. Este fator de correção pode ser utilizado quando não se espera um *speedup* linear na fase de computação. Valores positivos e negativos ou até mesmo o resultado de

um função podem ser utilizados para impor um fator de *speedup* diferente do linear. Por exemplo, dependendo da aplicação este efeito pode ocorrer devido à redução dos custos de acesso à memória, já que uma espaço maior de *cache* fica disponível a medida que mais processadores são utilizados. Apesar de proposto no modelo, este parâmetro não foi utilizado no experimentos computacionais. Nota-se também que a equação 3.2 tem o foco principal em aplicações regulares, isto é, aplicações em que o tempo de computação para cada iteração no laço paralelo não difere [1].

Em teoria sabe-se que ao se utilizar um processador lento e um rápido para a execução de uma aplicação paralela, o tempo de computação fica limitado pelo processador lento, pois em toda etapa de sincronização o processador rápido deve aguardar que todas as tarefas do processador lento estejam finalizadas antes de prosseguir para a próxima etapa. Por este motivo $\mathbf{I_e}$ é calculado tendo como referência o processador lento.

Resultados preliminares e análises empíricas para todas as aplicações que foram utilizadas para validar o modelo mostraram uma melhor aproximação entre o tempo total estimado e o tempo total real ao empregar o valor normalizado de $\mathbf{R_p}$. Acredita-se que isto é um reflexo da sobreposição de comunicação com computação em comunicações assíncronas. Em uma comunicação assíncrona, a função envio começa suas operações mas não as completa, isto é, a função retorna antes dos dados serem copiados para o *buffer*. Da mesma forma, a operação de recebimento é invocada mas retorna antes dos dados serem recebidos e copiados para o *buffer*. Com o suporte de um *hardware* apropriado, o envio e recebimento de mensagens pode acontecer concorrentemente com a computação realizada pela aplicação após o retorno das operações assíncronas [27]. Desta forma o tempo total de execução diminui, como mostra a figura 3.1.

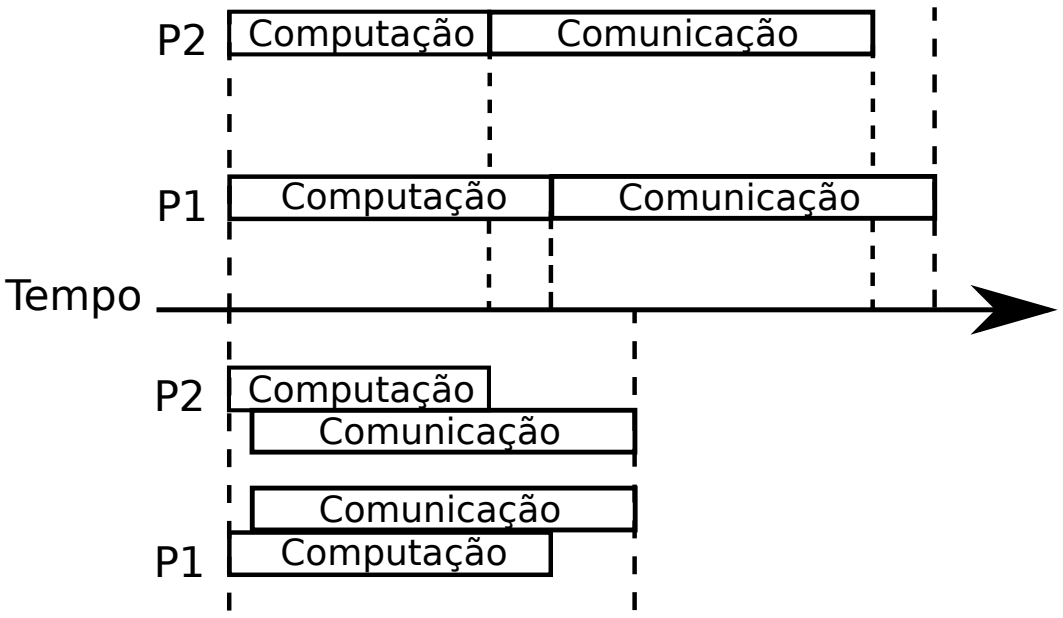

Figura 3.1: Exemplo de sobreposição de comunicação com computação.

3.2 Estimativa do tempo de comunicação

Os principais parâmetros utilizados pelo HCLogP para a fase de comunicação são:

a) Latência ($\boldsymbol{L_d}$): que representa um limite superior da latência ou atraso na troca de mensagens ponto-a-ponto em um controlador de rede (**d**). É o tempo que demora para o primeiro *bit* da mensagem ser transferido do *sender* para o *receiver*;

b) Overhead ($\boldsymbol{o(d, p, m)}$): tempo que o processador (**p**) emprega para transmitir ou receber uma mensagem (**m**) em um controlador de rede (**d**), não podendo ser usado para executar outra instrução;

c) Gap ($\boldsymbol{g_d}$): representa o tempo mínimo entre envios ou recebimentos de mensagens consecutivas em um processo em um controlador de rede (**d**);

d) ($\boldsymbol{N_{op}}$): é número de operações de comunicação por iterações;

e) (**M**): representa o tamanho da mensagem para cada operação de comunicação.

Os valores de $\boldsymbol{L_d}$ e $\boldsymbol{g_d}$ são obtidos antes da execução da aplicação. Um *benchmark* de rede [1] é utilizado para este propósito. O *benchmark* é executado para cada tipo de

[1] `http://mvapich.cse.ohio-state.edu/benchmarks/`

rede disponível no *cluster*, coletando os valores de $\boldsymbol{L_d}$ e $\boldsymbol{g_d}$ para diferentes tamanhos de mensagens, variando de 0 a 4 MB.

Para facilitar a notação vamos descrever o overhead total ($\boldsymbol{o(d,p,m)}$) como $\boldsymbol{o_d}$, ressaltando que o *overhead* varia com o controlador de rede **d**, com o processador **p** e com o tamanho da mensagem **m**. O valor de $\boldsymbol{o_d}$ é computado usando um *benchmark* [25]. Este *benchmark* considera que o *overhead* varia com o tamanho da mensagem. O *overhead* total em uma operação de comunicação ponto-a-ponto é dado por: $o_{\mathbf{d}} = o_{\mathbf{s}} + o_{\mathbf{r}}$, onde $\mathbf{o_s}$ e $\mathbf{o_r}$ são respectivamente o *overhead* de envio e de recebimento. Cada tamanho de mensagem usado por nosso modelo usa o valor correspondente de $\boldsymbol{o_d}$ obtido pelo *benchmark*.

O custo de comunicação depende basicamente de dois fatores: o tipo de mensagem enviada (ponto-a-ponto ou coletiva), e o tamanho da mensagem. Dependendo da biblioteca de troca de mensagens utilizada, o tamanho da mensagem determina os custos de comunicação, e o tipo de mensagem determina o número de mensagens trocadas. Como uma regra geral, o custo de envio de uma única mensagem ponto-a-ponto é igual a $L_d + \frac{M}{B_d} + o_d$, onde **M** é o tamanho da mensagem e $\mathbf{B_d}$ a banda passante do dispositivo **d** que pode ser obtida através de $\mathbf{g_d}$. O custo de comunicação para operações coletivas do mesmo tipo pode variar de acordo com o tamanho da mensagem, topologia de rede e número de processos envolvidos na comunicação. Todas as operações são multiplicadas pelo número de vezes que elas são executadas em cada iteração da aplicação ($\mathbf{N_{op}}$). As equações 3.3 e 3.4 modelam duas operações de comunicação coletiva.

$$T_{AlltoAll} = N_{op} \times (P-1) \times (L_d + \frac{M}{B_d} + o_d), \tag{3.3}$$

$$T_{AllReduce} = N_{op} \times log_2 P \times (L_d + \frac{M}{B_d} + o_d), \tag{3.4}$$

A equação 3.3 refere-se as operações do tipo *All-to-All*, onde todos os processos enviam dados para todos os outros processos. A equação 3.4 modela as operações do tipo *All-Reduce*, onde cada processo tem uma quantidade de dado que deve ser reduzida e uma cópia do resultado deve existir em cada processo. Esta mesma equação também pode ser utilizada para representar as operações do tipo *Scatter* e *Gather*, onde um processo envia/recebe um dado de cada outro processo, e a operação *One-to-All*, em que um processo envia dados para todos os outros.

Contudo, deve-se ressaltar que uma análise do código é obrigatória para determinar a fórmula que deve ser empregada para estimar o custo de comunicação final.

Um dos objetivos deste trabalho é manter o modelo o mais simples possível, resistindo ao ímpeto de sugerir um modelo mais detalhado, algumas considerações foram feitas:

a) incluir os custos de sincronização e comunicação intra-nós (por exemplo, a troca de dados entre GPU e CPU);

b) considerar a topologia da rede.

Caso os custos de comunicação intra-nós sejam relevantes, eles podem ser incluídos no modelo usando $\mathbf{L_d}$, $\mathbf{o_d}$ e $\mathbf{g_d}$, onde o dispositivo é a GPU com a latência, *overhead* e banda passante associados. A topologia de rede não foi considerada por dois motivos: a) algoritmos geralmente não têm sua execução condicionada a um tipo particular de topologia, apesar do protocolo de comunicação usar esta informação para otimizar operações coletivas; e b) a topologia de rede está refletida nos parâmetros $\mathbf{L_d}$ e $\mathbf{g_d}$.

3.3 Estimativa do tempo total

Estimados ambos os componentes, o tempo total de execução da aplicação pode ser calculado da seguinte forma:

$$T_{\text{execução}} = T_{\text{computação}} + T_{\text{comunicação}}. \tag{3.5}$$

Como pode ser observado, as principais diferenças entre o modelo proposto neste trabalho e o LogP original são:

a) o foco em predizer o tempo total de execução da aplicação;

b) o modelo LogP assume um ambiente homogêneo, enquanto que o HCLogP assume um ambiente heterogêneo;

c) devido à heterogeneidade do ambiente, a normalização do número de processos é necessária;

d) o modelo LogP assume que todas as mensagens são de tamanho pequeno, enquanto que o HCLogP não faz nenhuma estimativa quanto ao tamanho da mensagem, que é também um parâmetro do modelo;

e) um novo parâmetro foi introduzido, $\mathbf{R_p}$, com o objetivo de se predizer o tempo de computação de um aplicação.

Benchmarks para coletar os custos de comunicação, *overhead* e o poder computacional relativo de processadores e aceleradores, podem ser executados somente uma vez, ou cada vez que um novo *hardware* ou rede é incluído no ambiente.

Neste trabalho a própria aplicação foi utilizada para se obter os parâmetros necessários para o modelo. Isso se deve ao fato de que muitos *benchmarks* diferenciam o poder de processadores utilizando a métrica de FLOPS (Operações de ponto flutuante por segundo). Esta é uma boa métrica, desde que se saiba exatamente o que está sendo medido. De acordo com o tempo de CPU [28], dado pela seguinte fórmula:

$$\text{Tempo de CPU} = \frac{\text{Número de instruções} \times \text{Média de ciclos de } \textit{clock} \text{ (CPI)}}{\text{Velocidade de } \textit{clock}}, \tag{3.6}$$

um processador que apresenta uma medida de FLOPS maior que o outro, não necessariamente apresenta um desempenho melhor, visto que o CPI das instruções pode ser maior. Esta afirmação também é válida para a velocidade de *clock*. Através do uso da aplicação como *benchmark* é possível simplificar a obtenção de valores (não sendo necessários os valores para se calcular o tempo de CPU), os valores se tornarão mais precisos e com uma única execução é possível obter dois parâmetros do modelo (poder computacional e o tempo de execução das iterações utilizadas como referência para o cálculo do tempo de computação).

3.4 Resumo do capítulo

O modelo HCLogP, proposto neste capítulo, calcula o tempo total de execução de uma aplicação pela soma dos tempos de computação e de comunicação ($T_{\text{execução}} = T_{\text{computação}} + T_{\text{comunicação}}$). O tempo de computação é calculado da seguinte forma:

$$T_{\text{computação}} = \frac{I}{I_s} \times (\frac{I_e}{\text{Soma}_{Rp} + F_r}), \tag{3.7}$$

onde:

- I é o número total de iterações do problema;

- I_s é o número de iterações sequenciais utilizadas como referência;
- I_e é o tempo de execução de I_s iterações;
- R_P é a normalização do poder de computação relativo de cada unidade computacional distinta no ambiente;
- Soma_{Rp} é a soma dos R_p de todos os processos que serão usado na execução paralela;
- F_r é um fator de correção por conta do *speedup* não linear (apesar de proposto, este parâmetro não foi utilizado nos experimentos deste trabalho).

O tempo de comunicação depende do tipo de mensagem enviada (ponto-a-ponto ou coletiva) e do tamanho da mensagem. O custo de comunicação para operações coletivas do mesmo tipo pode variar de acordo com o tamanho da mensagem, topologia de rede e número de processos envolvidos na comunicação. As seguintes fórmulas ilustram como cada operação de comunicação pode ser modelada:

$$T_{SendReceive} = N_{op} \times (L_d + \frac{M}{B_d} + o_d). \tag{3.8}$$

$$T_{AlltoAll} = N_{op} \times (P-1) \times (L_d + \frac{M}{B_d} + o_d), \tag{3.9}$$

$$T_{AllReduce} = N_{op} \times log_2 P \times (L_d + \frac{M}{B_d} + o_d), \tag{3.10}$$

onde:

- N_{op} é o número de chamadas, por iteração, à operação de comunicação;
- P é o número de processos envolvidos na comunicação;
- L_d é a latência no controlador de rede d;
- M é o tamanho da mensagem;
- B_d é a banda passante do controlador de rede d que pode ser obtida através de g_d;
- $o_{\mathbf{d}}$ é o *overhead* total em uma operação de comunicação, que é dado por: $o_{\mathbf{d}} = o_{\mathbf{s}} + o_{\mathbf{r}}$, onde $o_{\mathbf{s}}$ e $o_{\mathbf{r}}$ são respectivamente o *overhead* de envio e de recebimento. O $o_{\mathbf{d}}$ depende

do tamanho da mensagem, do controlador de rede e do tamanho da mensagem envolvida na comunicação.

Assim o custo de comunicação total é a soma do custo de todas as operações de comunicação envolvidas na execução da aplicação.

4 Escalonador

4.1 Introdução

Dado que foi apresentado um modelo para a estimativa do tempo total de execução de uma aplicação paralela, um dos principais objetivos da dissertação é a utilização deste modelo para a obtenção da configuração de nós e redes que devem ser usados para que o tempo de execução paralelo seja o menor possível. Por exemplo, suponha que um ambiente de agregados computacionais seja formado por três tipos diferente de rede e sete nós computacionais organizados da seguinte maneira:

Rede Rápida (R_r)	No_1, No_2, No_3
Rede Intermediária (R_i)	No_4, No_5
Rede Lenta (R_l)	No_1, No_2, No_3, No_4, No_5, No_6, No_7

Tabela 4.1: Exemplo de uma ambiente de agregados computacionais.

Considere também que quanto maior o índice do nó de computação, maior é seu poder de processamento. Nota-se então que uma aplicação que seja embaraçosamente paralela apresenta um melhor desempenho utilizando a rede lenta (R_l), uma vez que os nós de maior capacidade computacional estão interligados por ela. Agora suponha uma aplicação paralela que realize transferências de grandes volumes de dados. Neste caso, o uso das melhores unidades computacionais pode não ser interessante, uma vez que o custo de comunicação empregando a rede mais lenta (R_l) é maior do que o das outras redes. A utilização da rede rápida (R_r), apesar de conter nós com processamento inferior, pode ser interessante dado que o custo de comunicação diminui. Até mesmo a combinação das redes Intermediária (R_i) e Rápida (R_r) pode ser vantajosa para uma dada aplicação, dependendo dos custos de computação e comunicação envolvidos. Os custos de comunicação e de computação equivalem respectivamente aos tempos de comunicação e de computação, ambos descritos no Capítulo 3.

O problema descrito acima tem características de um problema combinatório. Para uma grande quantidade de problemas de otimização combinatória, não pode se esperar

que um algoritmo produza a solução ótima em tempo polinomial para todas as instâncias dos problemas. Uma questão importante que deve ser considerada em problemas de otimização é a classificação quanto a sua dificuldade. Muitos problemas de otimização são pelo menos tão difíceis quanto qualquer problema do tipo NP, sendo classificados como problemas do tipo NP-Difícil, mas não pertencem ao conjunto NP. Problemas que são NP-Difícil e pertencem ao conjunto NP, são considerados NP-Completo [29]. O problema que a equação 4.1 descreve é um problema similar ao problema da soma dos subconjuntos, que é do tipo NP-Completo [30].

$$
\begin{aligned}
\text{Min Tempo Total} = \quad & \sum_{i=1}^{P} \text{CustoComputação}(\text{n}_i) + \\
& \text{CustoComunicação}(R, N) \\
\text{sujeito à} \quad & 0 \leq r \leq 1 \text{ tal que } r \in R \subseteq \mathbb{I} \\
& \sum_{r \in R} r \leq K \\
& 0 \leq n \leq 1 \text{ tal que } n \in N \subseteq \mathbb{I} \\
& \sum_{n \in N} n \leq P
\end{aligned}
\tag{4.1}
$$

onde:

N é conjunto de nós do ambiente,

R é conjunto de redes do ambiente,

P é o número total de nós do ambiente, e

K é o número total de redes do ambiente.

Como o problema P = NP é um problema em aberto, nesta dissertação assume-se que P $\neq$ NP, como é adotado na maioria dos trabalhos de otimização. Assim assume-se que não existe a possibilidade de se encontrar a solução ótima para o problema descrito em tempo polinomial. No entanto, existem abordagens que podem ser utilizadas para contornar esse tipo de problema, e a abordagem clássica para problemas do tipo NP-Completo é utilizar algoritmos que fornecem uma solução aproximada. Um algoritmo que retorna soluções aproximadas é chamado de algoritmo de aproximação. Estes algoritmos são executados em tempo polinomial, mas garantem que a solução aproximada obtida está a uma certa proporção da solução ótima [31, 29].

4.2 Implementação do Escalonador

Para implementar um escalonador que forneça como resultado a melhor configuração de um ambiente para uma dada aplicação, foi escolhido um algoritmo de aproximação baseado em uma modificação do método enumerativo denominado *Branch and Bound*. Métodos enumerativos consistem em investigar vários casos de um dado problema através de uma maneira implícita. Isto significa que a maioria de casos é descartada baseado nas consequências obtidas da análise de um problema numérico particular. O método *Branch and Bound* foi escolhido por poder lidar facilmente com problemas que apresentam variáveis discretas e contínuas, além disso outras técnicas que usam enumeração implícita podem ser facilmente incorporadas no método [32].

Para facilitar o entendimento, utilizaremos um exemplo para descrever a sequência de passos que o escalonador utilizaria para escolher uma configuração que minimiza o tempo de execução de uma aplicação no ambiente computacional descrito na tabela 4.1. Inicia-se com o nó com maior poder de processamento, o nó 7, que tem apenas a rede mais lenta (R_l) conectando-o aos demais nós. Utilizando o método HCLogP, estima-se o tempo de execução da aplicação utilizando-se o nó 7. Suponha que a estimativa seja de um tempo de execução igual a 15 segundos. Na sequência, o escalonador tentará incluir novos nós na solução até que o tempo de execução estimado seja igual ou maior ao melhor tempo estimado até então. Na sequência, tenta-se incluir o segundo nó de maior desempenho, o nó 6. Caso o tempo de execução estimado da aplicação seja reduzido com o uso simultâneo dos nós 6 e 7, empregando a rede R_l para sua comunicação, atualizamos o tempo estimado e o conjunto de nós. Suponha que a estimativa do tempo de execução da aplicação usando estes nós seja de 10 segundos. Como o tempo de execução é reduzido, atualiza-se a solução (6 e 7) e o tempo (10). O algoritmo seguirá sua execução tentando incluir o terceiro nó de maior desempenho, o nó 5, na solução. Considere que a estimativa do tempo de execução da aplicação usando os nós 5, 6 e 7 seja de 11 segundos. Como o novo tempo é maior do que a solução atual, esta é mantida e finaliza-se o passo atual, relativo ao uso apenas de máquinas na rede mais lenta.

As subárvores do primeiro passo são geradas removendo em ordem crescente os nós obtidos na solução 1 e acrescentando os melhores nós da rede mais rápida um nível acima (R_i). Na primeira solução foram obtidos os nós 6 e 7, então é gerada um subárvore removendo o nó 6 e uma outra subárvore removendo os nós 6 e 7, combinando-as com os

nós da rede intermediária (R_i) que possuem melhor desempenho, respeitando a restrição de que o tempo de execução deve se manter menor com a inclusão de um novo nó. Observe que a subárvore removendo apenas o nó 7 e mantendo o nó 6 pode ser descartada, uma vez que o nó 7 é mais rápido do que o nó 6, com custos de comunicação idênticos para ambos. Desta forma, qualquer configuração que tenha apenas um nó conectado pela rede lenta, que não seja o nó 7, pode ser descartada.

No passo 2, o nó 6 é removido e o nó 7 é combinado inicialmente com o nó 5. Suponha que o tempo estimado de execução nesta configuração seja de 9,5 segundos. Na sequência, tenta-se adicionar o nó 4 na configuração. A estimativa do tempo de execução para uma execução da aplicação nos nós 4, 5 e 7 é de 9 segundos, o que permite a inclusão do nó 4 no conjunto solução e a atualização do tempo. Como não restam mais nós nesta rede, finaliza-se esse passo.

Quando o melhor nó de uma rede é removido em uma subárvore, como ocorre no passo 3, todo o processo descrito no passo 1 se repete para esta subárvore, mas desta vez considerando a rede mais rápida um nível acima (R_i). Inicia-se então a estimativa do tempo de execução utilizando-se o nó mais rápido desta rede, o nó 5, acrescentando os demais nós na mesma rede e nas redes inferiores, até que o tempo de execução encontrado não possa ser reduzido ou quando não existirem mais nós a serem incluídos. Ilustramos uma situação em que a segunda situação se aplica. Após a inclusão dos nós 4 e 5 da rede intermediária (R_i) e dos nós 3, 2 e 1 da rede rápida (R_r), não existem mais nós a serem incluídos no passo 3, que tem estimativa de tempo de execução de 6 segundos. Adicionalmente, como o passo 3 já esgotou todas as opções de variação de configurações, nenhuma subárvore é gerada a partir deste passo.

As subárvores do passo 2 são geradas da mesma forma que as geradas no passo 1. Assim o passo 4 descarta inicialmente o nó 4 e verifica a viabilidade de reduzir o tempo estimado de execução da aplicação com a inclusão dos nós 3, 2 e 1 (nesta ordem). O passo 5 repete o procedimento, mas descartando os nós 4 e 5. O passo 6, não apresentado por possuir estimado de execução maior que o tempo da instância superior (passo 2), descarta todos os nós da solução do passo 2 (4, 5 e 7), considerando apenas os nós 1, 2 e 3. Como todas as opções de configuração foram esgotadas, o algoritmo finaliza, selecionando a configuração encontrada no passo 3 como sendo a melhor opção para executar o aplicativo.

Apesar do algoritmo descartar muitas configurações irrelevantes para a solução final

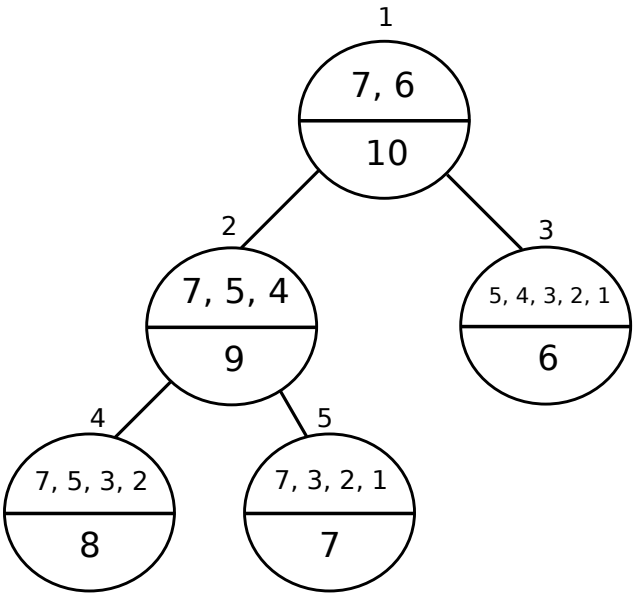

Figura 4.1: Sequência de passos do Escalonador. Cada círculo corresponde a um passo. Os passos são numerados no lado externo dos círculos. No interior destes são apresentados, na parte superior, os nós que compõem uma solução e na parte inferior o tempo para execução da aplicação usando estes nós.

do problema, ele ainda pode ser visto como um algoritmo "exaustivo inteligente". A sua escolha para o escalonador se justifica pelo fato de que muitos ambientes reais apresentam um grande número de nós iguais e um número pequeno de redes diferentes que em muitos casos não se comunicam entre si. Desta forma não é necessária uma solução mais sofisticada envolvendo, por exemplo, a utilização de meta-heurísticas. A escolha da expansão a partir da rede mais lenta é arbitrária, uma vez que o mesmo número de verificações é necessário quando o algoritmo começa a expansão pela rede mais rápida.

Como o principal objetivo do escalonador é a minimização do tempo de execução de uma aplicação para um dado ambiente computacional, o algoritmo utiliza o máximo de recursos disponíveis para que este objetivo seja alcançado. Por exemplo, se uma grande quantidade de recurso for empregada para reduzir minimamente o tempo de execução de uma única aplicação, a escolha do algoritmo será empregar todos os recursos disponíveis. Contudo, quando consideramos que outras aplicações podem estar aguardando recursos em uma fila para execução, esta escolha pode não ser a mais interessante. Neste cenário, as máquinas que contribuem para uma pequena redução no tempo de execução poderiam ser melhor empregadas na execução de outros aplicativos. Ainda que não existam outras aplicações aguardando na fila, poderia-se argumentar que não faz sentido, do ponto de

Algoritmo 1: Pseudo-código do algoritmo do Escalonador

```
1  início
2      conjunto ativos = { raiz R };
3      melhor valor = ∞ ;
4      melhor nó = NULL ;
5      enquanto conjunto ativos ≠ {∅} faça
6          remove o nó R do conjunto ativos;
7          gera as subárvores de R;
8          para subárvore i = 1; i ≤ n_r faça
9              calcula a cota superior de i;
10             se i não possui configuração melhor então
11                 poda a subárvore i;
12             fim se
13             senão se cota superior de i < melhor valor então
14                 melhor valor = cota superior de i;
15                 melhor nó = i;
16             fim se
17             adiciona i ao conjunto ativos;
18         fim para
19     fim enquanto
20 fim
```

vista da eficiência energética, a inclusão de recursos que pouco contribuam para a redução do tempo de execução. Novamente, restrições adicionais poderiam ser implementadas no algoritmo para evitar estas situações.

O pseudo-código 1 representa o algoritmo que implementa o escalonador. As linhas de 2 a 4 representam a inicialização da pilha de ativos utilizada pelo algoritmo, onde a raiz **R** representa a primeira configuração parcial encontrada pelo escalonador. A linha 5 representa o laço principal do algoritmo onde a partir da configuração removida da pilha são acrescentos os novos filhos, correspondendo a novas configurações encontradas pelo escalonador, nesta mesma pilha. As linhas 9, 13 e 14 correspondem ao cálculo do tempo total de execução da aplicação de acordo com o modelo HCLogP utilizando a configuração encontrada pelo algoritmo. A linha 10 é a condição onde nenhum filho do nó corrente é adicionado a pilha de ativos. As linhas de 13 a 15 representam a escolha da melhor configuração dentre todas encontradas até o passo corrente do algoritmo.

4.3 Resumo do capítulo

Para um determinado ambiente heterogêneo constituído de distintas unidades computacionais (CPUs e GPUs) e diferentes controladores de redes, escolher a melhor configuração de forma que o tempo de execução de uma aplicação paralela regular seja o menor possível é uma característica de um problema combinatório. Neste caso o problema é similar ao problema da soma dos subconjuntos, que é do tipo NP-Completo [30].

A abordagem utilizada para implementar o algoritmo escalonador foi um método enumerativo baseado no algoritmo aproximativo *Branch and Bound*. O escalonador estima, a cada passo, o tempo de execução de uma aplicação em uma nova configuração de hardware, que pode ser composta por processadores, aceleradores e redes distintos. A estimativa é baseada na descrição da aplicação seguindo o modelo HCLogP. O algoritmo descarta a análise de algumas configurações que, de um ponto de vista lógico, não poderiam levar a redução do tempo total de execução.

O foco do algoritmo implementado é na minimização do tempo total de execução de uma aplicação paralela regular, ou seja, o algoritmo pode empregar uma grande quantidade de recursos computacionais para obter uma redução mínima no tempo total de execução.

5 Experimentos Computacionais

5.1 Introdução

Os *benchmarks* NAS [3] foram utilizados para validar o modelo HCLogP e o algoritmo empregado no escalonador. Como estes *benchmarks* foram desenvolvidos para serem executados em um ambiente de CPUs, uma outra aplicação, HIS, foi escolhida para avaliar o modelo em um ambiente híbrido formado por CPUs e GPUs. A aplicação HIS simula alguns componentes de sistema imune humano [4]. O objetivo é avaliar o modelo com aplicações que apresentam diferentes características de computação e comunicação e observar se o modelo pode capturar estas características. Para todas as aplicações do *benchmark* NAS, o número de processos deve ser potência de 2, exceto para BT e SP, que devem ser um quadrado perfeito (1, 4, 9, 16, ...). Para a aplicação HIS qualquer número de processos pode ser utilizado.

Para cada ambiente e para cada aplicação, descritos nas próximas seções, serão apresentados os tempos estimados (computação, comunicação e total) e comparados com os tempos de execução real, onde o erro percentual (Equação 5.1) será utilizado para a validação do modelo. Resultados do escalonador para a minimização do tempo de execução também serão apresentados.

$$\text{Erro Percentual} = \frac{|\text{Tempo Estimado} - \text{Tempo Exato}|}{|\text{Tempo Exato}|} \times 100\%. \tag{5.1}$$

Para um melhor entendimento do texto, uma descrição detalhada de cada aplicação avaliada será apresentada em cada uma das seções.

5.2 Ambiente Computacional

Os experimentos foram executados em um *cluster* com 20 máquinas. Dessas, 11 possuem dois processadores AMD 6272, com 32 GB de memória principal, e duas GPUs Tesla M2075, cada uma com 448 núcleos CUDA e 6 GB de memória global. Outras 8 máquinas possuem dois processadores Intel Xeon $E5620$, com 16 GB de memória principal; 6 destas máquinas possuem duas GPUs Tesla C1060 (240 núcleos CUDA e 4 GB de memória global

cada) e as outras 2 com duas GPUs Tesla M2050 (448 núcleos CUDA e 3 GB de memória global). A máquina restante possui quatro processadores AMD 6272, com 128 GB de memória principal e quatro GPUs Tesla M2090 (512 núcleos CUDA e 6 GB de memória global). Linux versão 2.6.32, *driver* CUDA versão 6.0, OpenMPI versão 1.6.2, *nvcc* versão 6.0 e *gcc* versão 4.4.7 foram usados para compilar e executar todos os códigos.

Dois controladores de rede distintos estão disponíveis no *cluster*: Intel 82576 Gigabit Ethernet e InfiniBand Mellanox MT26428 com QDR de 40Gb/s. As máquinas com processadores Intel estão conectadas pelo controlador Gigabit Ethernet, enquanto as máquinas AMD estão conectadas por ambos os controladores. Ambas as redes possuem o modo *full-duplex*, onde dados podem ser enviados e recebidos simultaneamente. Por esta razão, para cada aplicação o modelo considera somente metade do número de mensagens transmitidas, já que elas ocorrem em paralelo. Além disso, os controladores de rede não se comunicam entre si, desta forma somente um único controlador pode ser usado para realizar a comunicação das máquinas. Apesar do número de núcleos disponíveis em cada máquina ser igual a 32 e 64 para AMD (2×16 e 4×16) e 8 para Intel (2×4), em todos os experimentos apresentados na próxima seção somente um processo por máquina foi utilizado.

Três ambientes distintos foram usados nos experimentos. Dois ambientes homogêneos que utilizam somente um único tipo de CPU e um ambiente heterogêneo, que mescla tipos distintos de CPU. O primeiro ambiente homogêneo é composto por processadores AMD, e ambos os controladores de rede são utilizados (Ethernet e Infiniband), enquanto que o segundo ambiente é constituído por processadores Intel utilizando somente o controlador de rede Gigabit Ethernet. No ambiente heterogêneo, metade dos processadores são AMD e metade são Intel. A única exceção são os *benchmarks* BT e SP que utilizam 9 núcleos, dos quais 5 CPUs são Intel e 4 AMD. O modelo também foi avaliado utilizando ambientes homogêneos e heterogêneos para GPUs. O ambiente homogêneo é composto somente pela GPU M2075, enquanto que o ambiente heterogêneo mescla as GPUs C1060, M2050 e M2075.

O problema combinatório que caracteriza o ambiente computacional pode ser descrito pela equação 5.2, onde $P = 20$ e K foi substituído pelo valor 1. O valor de K é igual a 1 porque, apesar do ambiente possuir dois controladores de rede, somente um deles pode ser usado a cada execução das aplicações. Isto ocorre porque a biblioteca de comunicação

MPI não permite o uso simultâneo de redes com tipos diferentes de controladores.

A tabela 5.1 apresenta o valor do parâmetro R_p de cada unidade de processamento do *cluster* normalizado pelo processador AMD (unidade computacional mais lenta). Essa normalização foi feita utilizando a aplicação HIS, dado que os *benchmarks* do NAS foram implementados para serem executados em CPUs. A normalização foi obtida executando uma pequena parte da aplicação (tamanho de malha de $100 \times 100 \times 100$ com 1.000 iterações).

$$
\begin{aligned}
\text{Min Tempo Total} = \quad & \sum_{i=1}^{P} \text{CustoComputação}(\text{n}_i) + \\
& \text{CustoComunicação}(R, N) \\
\text{sujeito a} \quad & 0 \leq r \leq 1 \text{ tal que } r \in R \subseteq \mathbb{I} \\
& \sum_{r \in R} r = 1 \\
& 0 \leq n \leq 1 \text{ tal que } n \in N \subseteq \mathbb{I} \\
& \sum_{n \in N} n \leq P
\end{aligned}
\tag{5.2}
$$

onde:

N é conjunto de nós do ambiente,

R é conjunto de redes do ambiente,

P é o número total de nós do ambiente.

Tabela 5.1: Poder relativo de cada unidade de processamento do *cluster* em que o experimentos foram realizados, normalizados pela unidade mais lenta.

Unidades de Processamento	$\boldsymbol{R_p}$
AMD	1
INTEL	1,78
C1060	131,22
M2050	299,34
M2075	333,73
M2090	364,41

Além disso, os parâmetros do modelo que caracterizam a latência e a banda passante foram obtidos utilizando o MTU (Unidade Máxima de Transmissão), que representa o tamanho do maior pacote que uma camada de um protocolo de comunicação pode transmitir. Se um pacote de dados é maior que o tamanho do MTU, este pacote é

fragmentado em múltiplos pacotes com tamanhos menores que não ultrapassam o máximo referido pelo MTU [33]. Para ambos os controladores de rede o MTU é definido com o valor de 1.500 *bytes*. Portanto, este tamanho de pacote foi utilizado para determinar os valores da latência e banda passante, que são apresentados na tabela 5.2.

Tabela 5.2: Valores de Latência e Banda Passante para os controladores Ethernet e Infiniband, considerando um pacote de 1.500 bytes.

	Latência	**Banda Passante**
Ethernet	$L_{\text{eth}} = 6{,}9 \times 10^{-5}$ s	$B_{\text{eth}} = 93,4$ MB/s
Infiniband	$L_{\text{inf}} = 5{,}1 \times 10^{-6}$ s	$B_{\text{inf}} = 1.030,3$ MB/s

Por fim, para todas as aplicações utilizadas no processo de validação do modelo, nenhum *flag* de otimização foi utilizado na etapa de compilação dos códigos.

5.3 Resultados do Modelo

5.3.1 FT

Um *kernel* de transformada rápida de Fourier 3-D (FT) [3] é usado para resolver numericamente equações diferenciais parciais (EDPs) usando FFTs (Transformadas rápidas de Fourier) diretas e inversas. Este *kernel* testa comunicações de longa distância, considerando que os passos da FFT 3-D requerem uma considerável comunicação para as operações, como por exemplo transposições de *array*. Essas operações são implementadas usando operações coletivas do tipo todos-para-todos. A classe B deste problema foi utilizada para a validação. O algoritmo 2 descreve de maneira simplificada o *kernel* FT.

Algoritmo 2: Pseudo-código do *kernel* FT

```
1 início
2 |  para i ← 1; i ≤ I faça
3 |  |  evolui u0 para u1 (t time steps) no espaço de Fourier;
4 |  |  executa sub-rotina fft em cada processo;
5 |  |  MPI_Alltoall para transferir os resultados;
6 |  |  realiza checagem de resultados;
7 |  fim para
8 fim
```

Para se obter os parâmetros do modelo, uma análise do código fonte é suficiente. No

entanto, como os códigos são de outra autoria e devido a sua complexidade, o *Vampir*[1], uma ferramenta de *profile* para aplicações paralelas, foi utilizado com a intenção de facilitar a coleta de parâmetros.

A fórmula do tempo de computação, definida no capítulo 3, é dada pela equação 5.3. A própria aplicação foi utilizada como *benchmark* para a coleta dos parâmetros: Foram utilizados 10% do total de iterações de cada aplicação para se obter os parâmetros I_s e I_e. O valor de I_e, tempo que a quantidade de iterações I_s demora para executar, foi obtido para cada tipo diferente de unidade computacional presente no ambiente de agregados.

$$T_{\text{computação}} = \frac{I}{I_s} \times (\frac{I_e}{\text{Soma}_{Rp} + F_r}). \tag{5.3}$$

Para os ambientes homogêneos envolvendo um único tipo de processador, os parâmetros para estimar o tempo de computação foram: $I = 20$, $I_s = 2$ (10% do valor de I), $I_e = 13,412$ s para AMD e $I_e = 8,601$ s para Intel, Soma_{Rp} é igual ao número de processos utilizados (como os processadores não apresentam diferenças em um ambiente homogêneo a normalização não é necessária), $F_r = 0$.

Para se estimar o tempo de comunicação, inicialmente deve-se identificar no código os tipos das operações. Duas operações do tipo todos-para-todos foram identificadas no *kernel* FT, sendo descritas pela equação 5.4.

$$T_{\text{Comunicação}} = N_{op} \times (P - 1) \times (L_d + \frac{M}{B_d} + o_d), \tag{5.4}$$

onde $N_{op} = 2$, P é igual ao número de processos utilizados. O custo total de comunicação é resultado da soma de todas as operações identificadas no processo de comunicação. Os valores utilizados para a latência e banda são os mesmos da seção anterior. O tamanho da mensagem M em *bytes* é dado por $M = (((512 \times 256)/P) \times 256) \times 4$, em que 512 e 256 são números referente ao tamanho do problema e 4 se refere ao tamanho de um ponto flutuante de precisão simples em *bytes*. Então temos que $M = 67,1$ MB, $33,5$ MB e $16,7$ MB para respectivos $P = 2$, 4 e 8. Em todas as tabelas, o *overhead* (o_d) representa a soma de todos os *overheads* obtidos para cada tamanho de mensagem e para cada troca de mensagens. O algoritmo de troca de pares foi usado pela comunicação todos-para-todos. As tabelas 5.3, 5.4 e 5.5 mostram os resultados para os

[1] `https://www.vampir.eu/`

ambientes homogêneos.

Tabela 5.3: Resultados para o *kernel* FT usando 2, 4 e 8 nós homogêneos AMD-Ethernet. Todos os tempos estão em segundos (s).

FT AMD-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	64,6	67,1	30,5	33,5	18,6	16,8
Desvio Padrão	1,2%	-	2,3%	-	1,0%	-
Erro Percentual	-	3,7%	-	9,8%	-	9,8%
Tempo de Comunicação	30,4	28,7	40,6	43,1	55,2	50,3
Desvio Padrão	2,3%	-	1,0%	-	1,2%	-
Erro Percentual	-	5,4%	-	6,3%	-	8,9%
Tempo Total de Execução	95,0	97,9	71,1	78,9	73,8	68,7
Desvio Padrão	1,4%	-	0,7%	-	1,0%	-
Erro Percentual	-	3,0%	-	10,9%	-	6,9%
Overhead	2,1	-	2,2	-	1,6	-

Tabela 5.4: Resultados para o *kernel* FT usando 2, 4 e 8 nós homogêneos AMD-Infiniband. Todos os tempos estão em segundos (s).

FT AMD-INF	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	63,4	67,1	30,1	33,5	19,0	16,8
Desvio Padrão	1,5%	-	2,1%	-	1,8%	-
Erro Percentual	-	5,7%	-	11,4%	-	11,7%
Tempo de Comunicação	2,6	2,5	4,0	3,9	4,9	4,6
Desvio Padrão	3,9%	-	4,6%	-	3,7%	-
Erro Percentual	-	0,2%	-	1,1%	-	6,5%
Tempo Total de Execução	66,1	69,8	34,0	37,8	23,9	21,7
Desvio Padrão	1,3%	-	2,3%	-	2,0%	-
Erro Percentual	-	5,6%	-	11,0%	-	9,0%
Overhead	0,2	-	0,3	-	0,4	-

Tabela 5.5: Resultados para o *kernel* FT usando 2, 4 e 8 nós homogêneos Intel-Ethernet. Todos os tempos estão em segundos (s).

FT INTEL-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	42,3	43,0	21,4	21,5	11,4	10,8
Desvio Padrão	1,0%	-	0,7%	-	1,8%	-
Erro Percentual	-	1,7%	-	0,7%	-	1,8%
Tempo de Comunicação	27,0	28,7	41,5	43,1	55,9	50,3
Desvio Padrão	1,6%	-	1,1%	-	0,5%	-
Erro Percentual	-	6,3%	-	3,9%	-	0,6%
Tempo Total de Execução	69,3	73,7	62,8	66,7	67,3	62,6
Desvio Padrão	1,1%	-	0,9%	-	0,5%	-
Erro Percentual	-	6,4%	-	6,2%	-	6,9%
Overhead	2,0	-	2,1	-	1,6	-

Como pode ser observado, todos os erros na estimativa do tempo total de execução ficaram abaixo de 11,0%. A diferença média do tempo total real em relação ao tempo total estimado ficou sempre abaixo de 5 segundos. Para o tempo de comunicação, o modelo conseguiu capturar de forma satisfatória a relação entre o aumento do número de nós e a redução do tamanho da mensagem. O erro máximo observado na estimativa do tempo de comunicação foi de 8,9% para o caso de 8 processadores AMD utilizando a rede Ethernet. O *overhead* para a rede Ethernet sofreu mais influência do tamanho da mensagem do que o número de processadores utilizados, dado que ele diminuiu a medida que foram incluídos mais processos. O contrário foi observado para a rede Infiniband. Nota-se também que o *overhead* no processador Intel é ligeiramente menor do que no processador AMD.

A tabela 5.6 mostra os resultados para o caso heterogêneo, onde Soma_{Rp} é obtido pelo procedimento de normalização descrito no Capítulo 3. Por exemplo, para $P = 8$ onde 4 processadores AMD e 4 Intel foram utilizados na execução paralela da aplicação, $\text{Soma}_{Rp} = (R_p^{\text{AMD}} \times 4) + (R_p^{\text{Intel}} \times 4)$, onde $R_p^{\text{AMD}} = 1$ e $R_p^{\text{Intel}} = 1,78$. Assim temos que os valores de Soma_{Rp} para $P = 2$, 4, 8 e 16 são respectivamente $2,78$, $5,56$, $11,12$ e $22,24$. $M = 8,3$ MB para $P = 16$ e I_e para o ambiente heterogêneo é referente ao processador mais lento incluído no processo de normalização, então $I_e = 13,412$ s, referente ao processador AMD.

Tabela 5.6: Resultados para o *kernel* FT usando 2, 4, 8 e 16 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

FT INTEL-AMD	**2 Nós**		**4 Nós**		**8 Nós**		**16 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado	Real	Estimado
T. de Computação	53,9	48,2	24,6	24,1	14,3	12,1	7,8	6,0
Desvio Padrão	1,0%	-	2,5%	-	3,1%	-	4,2%	-
Erro Percentual	-	10,5%	-	2,0%	-	15,7%	-	22,5%
T. de Comunicação	27,4	28,7	46,7	43,1	52,7	50,3	58,0	53,9
Desvio Padrão	1,6%	-	3,6%	-	2,2%	-	0,7%	-
Erro Percentual	-	4,8%	-	7,6%	-	4,5%	-	7.0%
T. Total de Execução	81,3	79,0	71,4	69,4	67,0	64,0	65,7	61,3
Desvio Padrão	1,1%	-	2,6%	-	1,9%	-	2,0%	-
Erro Percentual	-	2,8%	-	2,6%	-	4,5%	-	6,7%
Overhead	2,0	-	2,2	-	1,6	-	1,4	-

Apesar da combinação dos processadores, o maior erro no tempo total estimado para execução foi de 6,7% para 16 máquinas, representando uma variação de 4,4 segundos em um total de 65,7 segundos. O erro no tempo de computação para 16 nós foi de 22,5%. Isto ocorre porque, para valores pequenos de execução, uma pequena diferença apresentada pode representar um grande erro percentual.

5.3.2 IS

O *Integer Sort* (IS) é um *kernel* que realiza operações de ordenação. Este *kernel* testa tanto a computação de inteiros como o desempenho da comunicação [3]. A classe C deste problema foi utilizada para a validação. O algoritmo 3 apresenta o pseudo-código desta aplicação.

Algoritmo 3: Pseudo-código do *kernel* IS

```
1  início
2      para i ← 1; i ≤ I faça
3          gera a sequência de números aleatórios e chaves subsequentes em todos os
           processos;
4          obtém o tamanho do balde para todo o problema usando MPI_Allreduce;
5          determina a redistribuição das chaves;
6          redistribui as chaves usando MPI_AlltoAll;
7          envia as chaves para os respectivos processos usando MPI_Alltoallv;
8          realiza os cálculos para se obter os valores mínimo e máximo de cada chave
           em cada processo;
9          determina o número total de chaves em todos os outros processos usando
           MPI_Send_Receive;
10     fim para
11 fim
```

Para os ambientes homogêneos envolvendo um único tipo de processador, os

parâmetros para estimar o tempo de computação foram: $I = 10$, $I_s = 1$, $I_e = 2,104$ s para AMD e $I_e = 1,691$ s para Intel, Soma_{Rp} é igual ao número de processos e $F_r = 0$.

O custo de comunicação envolveu 3 equações:

$$T_{AllReduce} = N_{op} \times log_2 P \times (L_d + \frac{M}{B_d} + o_d), \tag{5.5}$$

$$T_{AlltoAll} = N_{op} \times (P-1) \times (L_d + \frac{M}{B_d} + o_d), \tag{5.6}$$

$$T_{SendReceive} = N_{op} \times (L_d + \frac{M}{B_d} + o_d). \tag{5.7}$$

Para a equação 5.5, que representa a primitiva MPI_Allreduce, os parâmetros são: $N_{op} = 1$ e $M = 4 \times 10^{-3}$ MB. A equação 5.6, que representa a primitiva MPI_AlltoAll, foi executada duas vezes com tamanhos de mensagens diferentes e $N_{op} = 1$, a primeira vez com $M = 134,2$ MB, $33,6$ MB e $8,5$ MB para respectivos $P = 2$, 4 e 8 e uma segunda vez com $M = 4 \times 10^{-6}$ MB para todas as configurações. A equação 5.7 se refere a execução da primitiva MPI_Send e MPI_Receive entre todos os processos, portanto ela deve ser multiplicada por $P - 1$, onde $N_{op} = 1$ e $M = 4 \times 10^{-3}$ MB. As tabelas 5.7, 5.8 e 5.9 mostram os resultados para os ambientes homogêneos.

Tabela 5.7: Resultados para o *kernel* IS usando 2, 4 e 8 nós homogêneos AMD-Ethernet. Todos os tempos estão em segundos (s).

IS AMD-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	9,9	10,5	5,0	5,3	3,0	2,6
Desvio Padrão	1,8%	-	3,6%	-	1,8%	-
Erro Percentual	-	6,3%	-	6,2%	-	11,0%
Tempo de Comunicação	13,2	14,4	11,9	10,8	7,0	6,4
Desvio Padrão	3,6%	-	2,4%	-	1,7%	-
Erro Percentual	-	8,8%	-	9,6%	-	9,0%
Tempo Total de Execução	23,1	26,0	16,9	16,6	10,0	9,6
Desvio Padrão	2,4%	-	2,5%	-	1,9%	-
Erro Percentual	-	12,3%	-	1,6%	-	3,4%
Overhead	1,1	-	0,6	-	0,6	-

Tabela 5.8: Resultados para o *kernel* IS usando 2, 4 e 8 nós homogêneos AMD-Infiniband. Todos os tempos estão em segundos (s).

IS AMD-INF	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	9,9	10,5	5,0	5,3	2,9	2,6
Desvio Padrão	1,7%	-	1,6%	-	1,2%	-
Erro Percentual	-	6,4%	-	5,7%	-	9,0%
Tempo de Comunicação	1,5	1,4	1,1	1,0	0,5	0,6
Desvio Padrão	4,3%	-	3,9%	-	4,4%	-
Erro Percentual	-	14,9%	-	7,2%	-	15,5%
Tempo Total de Execução	11,4	11,9	6,0	6,3	3,4	3,3
Desvio Padrão	1,8%	-	1,9%	-	2,6%	-
Erro Percentual	-	4,4%	-	3,4%	-	5,4%
Overhead	0,2	-	0,1	-	0,1	-

Tabela 5.9: Resultados para o *kernel* IS usando 2, 4 e 8 nós homogêneos Intel-Ethernet. Todos os tempos estão em segundos (s).

IS INTEL-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	8,1	8,5	3,9	4,2	2,2	2,1
Desvio Padrão	0,5%	-	2,7%	-	3,4%	-
Erro Percentual	-	5,0%	-	7,8%	-	5,7%
Tempo de Comunicação	13,0	14,4	11,9	10,8	6,6	6,4
Desvio Padrão	1,8%	-	1,3%	-	2,4%	-
Erro Percentual	-	10,4%	-	9,0%	-	6,1%
Tempo Total de Execução	21,1	23,9	15,8	15,6	8,8	9,1
Desvio Padrão	1,2%	-	1,8%	-	3,0%	-
Erro Percentual	-	13,3%	-	1,3%	-	2,6%
Overhead	1,0	-	0,6	-	0,6	-

Todos os erros na estimativa do tempo total de execução ficaram abaixo de 13,3%; a maior diferença entre o tempo real e o estimado foi de 2,8 segundos em um total de 21,1 segundos. Apesar de trabalhar com valores pequenos, o modelo conseguiu bons resultados em termos percentuais, principalmente para a comunicação, onde mais tipos de primitivas de comunicação foram utilizados do que no *kernel* FT. O erro percentual na estimativa do custo de comunicação permaneceu abaixo de 15,5%. Destaca-se que no caso em que foram usados 4 processadores Intel (tabela 5.9), o modelo apresentou uma estimativa precisa, com um erro percentual de apenas 1,3%. Observa-se adicionalmente que o *overhead* nesta aplicação sofre do mesmo comportamento observado na aplicação FT; no entanto, como existem mais operações de comunicação, o *overhead* estagnou em 0,6 segundos ao se aumentar o número de núcleos de 8 para 16.

A tabela 5.10 mostra os resultados para o ambiente heterogêneo. Os valores das estimativas foram obtidos usando-se $\text{Soma}_{Rp} = 2,78$, $5,56$, $11,12$ e $22,24$ para $P = 2$, 4, 8 e 16 respectivamente, $M = 2,1$ MB para $P = 16$ e $I_e = 2,104$ s.

Tabela 5.10: Resultados para o *kernel* IS usando 2, 4, 8 e 16 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

IS INTEL-AMD	**2 Nós**		**4 Nós**		**8 Nós**		**16 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado	Real	Estimado
T. de Computação	9,2	7,6	4,4	3,8	2,0	1,9	1,0	0,9
Desvio Padrão	1,0%	-	2,5%	-	0,1%	-	0,1%	-
Erro Percentual	-	17,5%	-	14,7%	-	6,1%	-	9,6%
T. de Comunicação	14,4	14,4	11,4	10,8	6,7	6,4	3,8	3,4
Desvio Padrão	1,6%	-	0,6%	-	2,4%	-	0,6%	-
Erro Percentual	-	0,2%	-	5,6%	-	5,5%	-	10,1%
T. Total de Execução	23,6	23,0	15,9	15,2	8,8	8,4	4,9	4,5
Desvio Padrão	1,1%	-	0,8%	-	2,4%	-	0,6%	-
Erro Percentual	-	2,5%	-	4,5%	-	3,7%	-	7,8%
Overhead	1,0	-	$5{,}7 \times 10^{-1}$	-	$1{,}7 \times 10^{-1}$	-	$1{,}4 \times 10^{-1}$	-

O error percentual para o tempo total estimado para execução ficou abaixo de 7,8%, e a maior diferença entre o tempo estimado e o real foi de 0,7 segundos em um total de 15,9 segundos para a configuração de 4 nós. Observa-se que o erro para o tempo de comunicação ficou abaixo de 10,1%. Já em relação ao tempo de computação, para 2 e 4 processos esse foi elevado, chegando à 17,5%. Como o custo de comunicação representa uma maior fatia do tempo total de execução, o erro de computação acabou não sendo refletindo em sua integralidade neste erro.

Outro fator que pode fazer com que o erro total de execução se reduza, quando comparado aos erros observados nos tempos de computação e comunicação, ocorre quando as estimativas vão em direções opostas: uma é maior do que o tempo real observado, enquanto a outra é menor. Como o tempo total é a soma dos tempos de computação e comunicação, os erros acabam se anulando, e o resultado final é uma estimativa mais precisa, ainda que por conta do acaso. Observa-se que esta situação ocorreu para o caso de 4 processadores da tabela 5.9.

5.3.3 CG

Este *kernel* utiliza o método dos gradientes conjugados (CG) para calcular uma aproximação para o menor autovalor de uma matriz larga, esparsa e simétrica positiva definida. Este *kernel* é característico de computação em *grids* não estruturadas em que ele testa comunicação irregular empregando multiplicação matriz-vetor não estruturada [3]. A classe C deste problema foi utilizada para a validação. O algoritmo 4 apresenta o pseudo-código desta aplicação.

Para os ambientes homogêneos, os parâmetros para estimar o tempo de computação foram: $I = 75$, $I_s = 7$, $I_e = 51,164$ s para AMD e $I_e = 27,020$ s para Intel, Soma_{Rp} é

Algoritmo 4: Pseudo-código do *kernel* CG

```
1 início
2   para i ← 1; i ≤ I faça
3     Gradientes Conjugados ():
4       obtém ρ através de uma redução e soma usando MPI_Send_Receive;
5       soma as partições da submatriz-vetor A.z entre as linhas usando
        MPI_Send_Receive;
6       realiza a troca de partes de q usando MPI_Send_Receive;
7     normaliza z para obter x;
8   fim para
9 fim
```

igual ao número de processos e $F_r = 0$.

O custo de comunicação envolveu somente operações do tipo MPI_Send and MPI_Receive, dada pela equação 5.8.

$$T_{SendReceive} = N_{op} \times (L_d + \frac{M}{B_d} + o_d). \tag{5.8}$$

Em cada iteração são realizadas 3 operações envolvendo todos os processos, portanto devem ser multiplicadas por $P - 1$. As duas primeiras com $N_{op} = 1$ e $M = 8 \times 10^{-6}$ MB e $1,6 \times 10^{-5}$ MB. A terceira com $N_{op} = 24$, 26 e 28 para os respectivos $P = 2$, 4 e 8, e $M = 0,6$ MB. As tabelas 5.11, 5.12 e 5.13 mostram os resultados para os ambientes homogêneos.

Tabela 5.11: Resultados para o *kernel* CG usando 2, 4 e 8 nós homogêneos AMD-Ethernet. Todos os tempos estão em segundos (s).

CG AMD-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	284,4	274,1	143,5	137,0	62,3	68,5
Desvio Padrão	2,3%	-	1,9%	-	0,9%	-
Erro Percentual	-	3,6%	-	4,5%	-	10,0%
Tempo de Comunicação	12,3	11,7	36,6	38,0	87,9	95,5
Desvio Padrão	3,4%	-	1,7%	-	1,2%	-
Erro Percentual	-	5,2%	-	3,7%	-	8,6%
Tempo Total de Execução	296,7	286,9	180,1	177,4	150,3	169,2
Desvio Padrão	2,3%	-	2,0%	-	1,2%	-
Erro Percentual	-	3,3%	-	1,5%	-	12,6%
Overhead	1,1	-	2,4	-	5,2	-

Tabela 5.12: Resultados para o *kernel* CG usando 2, 4 e 8 nós homogêneos AMD-Infiniband. Todos os tempos estão em segundos (s).

CG AMD-INF	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	284,6	274,1	144,3	137,0	62,5	68,5
Desvio Padrão	2,1%	-	1,2%	-	0,7%	-
Erro Percentual	-	3,7%	-	5,0%	-	9,7%
Tempo de Comunicação	1,1	1,0	3,2	3,4	8,0	8,6
Desvio Padrão	6,5%	-	2,7%	-	2,6%	-
Erro Percentual	-	4,1%	-	7,7%	-	8,0%
Tempo Total de Execução	285,7	275,3	147,5	140,8	70,5	77,9
Desvio Padrão	2,2%	-	1,2%	-	1,0%	-
Erro Percentual	-	3,7%	-	4,5%	-	10,5%
Overhead	0,2	-	0,3	-	0,7	-

Tabela 5.13: Resultados para o *kernel* CG usando 2, 4 e 8 nós homogêneos Intel-Ethernet. Todos os tempos estão em segundos (s).

CG INTEL-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	150,3	144,8	76,1	72,4	40,0	36,2
Desvio Padrão	1,2%	-	1,3%	-	1,7%	-
Erro Percentual	-	3,7%	-	4,9%	-	9,6%
Tempo de Comunicação	12,6	11,7	36,5	38,0	88,1	95,5
Desvio Padrão	1,3%	-	1,0%	-	1,1%	-
Erro Percentual	-	7,2%	-	4,2%	-	8,3%
Tempo Total de Execução	162,9	157,5	112,6	112,1	128,2	136,0
Desvio Padrão	1,2%	-	1,1%	-	1,2%	-
Erro Percentual	-	3,3%	-	0,5%	-	6,1%
Overhead	1,0	-	1,7	-	4,3	-

O maior erro encontrado no tempo total estimado para execução foi de 12,6%, e a maior variação foi de 20 segundos em um total de 150,3 segundos para o mesmo caso. O modelo conseguiu capturar tanto a redução do tempo de computação (erros abaixo de 10,0%) como o aumento do tempo de comunicação (erros abaixo de 8,3%), a medida que novos processos vão sendo incluídos e consequentemente mais mensagens são trocadas. O *overhead* apresentou valores maiores do que os *kernels* anteriores; isso ocorreu devido a maior quantidade de troca de mensagens do *kernel* CG. A tabela 5.14 mostra os resultados para o ambiente heterogêneo onde $N_{op} = 30$ para $P = 16$, $I_e = 51,164$ s e $\text{Soma}_{Rp} = 2,78$, $5,56$, $11,12$ e $22,24$ para $P = 2$, 4, 8 e 16 respectivamente.

O maior erro na estimativa do tempo total de execução foi de 8,5% para 2 processadores, com uma diferença de aproximadamente 20 segundos em um tempo total de 229,8 segundos. Para 8 processadores heterogêneos o modelo previu com grande precisão o tempo de execução total da aplicação. Inversamente ao que aconteceu nos *kernels*

Tabela 5.14: Resultados para o *kernel* CG usando 2, 4, 8 e 16 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

CG INTEL-AMD	**2 Nós**		**4 Nós**		**8 Nós**		**16 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado	Real	Estimado
T. de Computação	217,4	197,2	109,9	98,5	48,7	49,2	25,2	24,6
Desvio Padrão	1,1%	-	0,5%	-	2,6%	-	0,8%	-
Erro Percentual	-	9,3%	-	10,3%	-	1,0%	-	2,4%
T. de Comunicação	12,5	11,7	34,5	38,0	99,7	95,5	237,3	219,1
Desvio Padrão	1,6%	-	1,6%	-	0,7%	-	2,6%	-
Erro Percentual	-	6,1%	-	9,8%	-	4,3%	-	7,6%
T. Total de Execução	229,8	210,4	144,4	138,8	148,4	149,1	262,5	253,7
Desvio Padrão	1,2%	-	0,5%	-	0,9%	-	2,2%	-
Erro Percentual	-	8,5%	-	3,9%	-	0,4%	-	3,2%
Overhead	1,5	-	2,3	-	4,4	-	10,0	-

anteriores, o tempo de computação é o que possui um maior impacto no tempo total. Para duas configurações, os erros foram baixos, variando entre 1,0% e 2,4%, enquanto que para as demais foram um pouco maiores, variando entre 9,3% e 10,3%. Já a comunicação permaneceu com erros abaixo de 10,0%.

5.3.4 MG

MG é um *kernel multigrid* simplificado. As operações executam comunicações altamente estruturadas e testam trocas de dados para curta e longa distância [3]. A classe C deste problema foi utilizada para a validação. O algoritmo 5 apresenta o pseudo-código desta aplicação.

Algoritmo 5: Pseudo-código do *kernel* MG

```
1 início
2     para i ← 1; i ≤ I faça
3         inicializa os coeficientes;
4         computa a solução aproximada na grid mais grossa;
5         efetua a troca dos pontos de borda para cada direção usando
          MPI_Send_Receive;
6         efetua a troca dos dados da borda using MPI_Send_Receive
7     fim para
8 fim
```

Para os ambientes homogêneos, os parâmetros para estimar o tempo de computação foram: $I = 20$, $I_s = 2$, $I_e = 18,730$ s para AMD e $I_e = 12,101$ s para Intel, Soma_{Rp} é igual ao número de processos e $F_r = 0$.

O custo de comunicação envolveu somente operações do tipo MPI_Send e MPI_Receive,

descritas pela equação 5.9.

$$T_{SendReceive} = N_{op} \times (L_d + \frac{M}{B_d} + o_d). \tag{5.9}$$

Em cada iteração são realizadas 6 operações. Todas com $N_{op} = 4$, 8 e 16 para os respectivos $P = 2$, 4 e 8, e $M = 2,097$ MB, $1,049$ MB, $1,048$ MB, $0,532$ MB, $0,264$ MB e $0,260$ MB. As tabelas 5.15, 5.16 e 5.17 mostram os resultados para os ambientes homogêneos.

Tabela 5.15: Resultados para o *kernel* MG usando 2, 4 e 8 nós homogêneos AMD-Ethernet. Todos os tempos estão em segundos (s).

MG AMD-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	88,5	93,7	45,1	46,8	21,4	23,4
Desvio Padrão	0,5%	-	1,3%	-	1,2%	-
Erro Percentual	-	5,8%	-	3,8%	-	9,2%
Tempo de Comunicação	4,3	4,5	8,6	9,1	16,8	18,1
Desvio Padrão	4,2%	-	3,6%	-	0,7%	-
Erro Percentual	-	5,7%	-	5,1%	-	7,9%
Tempo Total de Execução	92,8	98,4	53,7	56,3	38,2	42,3
Desvio Padrão	0,5%	-	1,1%	-	0,7%	-
Erro Percentual	-	6,0%	-	4,7%	-	10,6%
Overhead	0,2	-	0,4	-	0,8	-

Tabela 5.16: Resultados para o *kernel* MG usando 2, 4 e 8 nós homogêneos AMD-Infiniband. Todos os tempos estão em segundos (s).

MG AMD-INF	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	88,6	93,7	44,2	46,8	21,6	23,4
Desvio Padrão	0,3%	-	0,8%	-	1,2%	-
Erro Percentual	-	5,7%	-	6,0%	-	8,5%
Tempo de Comunicação	$3,6 \times 10^{-1}$	$4,1 \times 10^{-1}$	0,7	0,8	1,8	1,6
Desvio Padrão	6,0%	-	3,4%	-	2,0%	-
Erro Percentual	-	14,6%	-	9,9%	-	6,3%
Tempo Total de Execução	88,9	94,1	44,9	47,7	23,3	25,1
Desvio Padrão	0,4%	-	1,0%	-	1,1%	-
Erro Percentual	-	5,8%	-	6,2%	-	7,4%
Overhead	$1,9 \times 10^{-2}$	-	$3,8 \times 10^{-2}$	-	$7,6 \times 10^{-2}$	-

Tabela 5.17: Resultados para o *kernel* MG usando 2, 4 e 8 nós homogêneos Intel-Ethernet. Todos os tempos estão em segundos (s).

MG INTEL-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	62,4	60,5	31,9	30,3	13,8	15,1
Desvio Padrão	0,7%	-	0,6%	-	2,3%	-
Erro Percentual	-	3,0%	-	5,1%	-	10,0%
Tempo de Comunicação	4,0	4,5	9,0	9,1	16,7	18,1
Desvio Padrão	1,5%	-	2,2%	-	0,5%	-
Erro Percentual	-	13,4%	-	0,6%	-	8,8%
Tempo Total de Execução	66,4	65,1	40,9	39,7	30,4	34,0
Desvio Padrão	0,6%	-	0,9%	-	1,2%	-
Erro Percentual	-	1,9%	-	2,9%	-	11,7%
Overhead	0,1	-	0,4	-	0,7	-

O maior erro verificado para a estimativa do tempo total de execução foi de 11,7% para o ambiente com 8 CPUs Intel, e uma diferença máxima de aproximadamente 5,6 segundos em um total de 92,8 segundos para o ambiente com 2 CPUs AMD-Ethernet. Para este *kernel* o tempo de comunicação estimado teve erros mais elevados quando comparado aos *benchmarks* anteriores, com um erro máximo de 14,6%. Novamente, os valores dos tempos são muito pequenos para o ambiente que utiliza a rede Infiniband, e uma pequena variação em seu valor pode resultar em um erro percentual alto. Mesmo diante desta dificuldade, deve-se ressaltar que o modelo obteve uma boa aproximação.

Para esta aplicação, o tempo de comunicação aumenta com o número de processos, já que o tamanho das mensagens se mantém o mesmo, fato que também é capturado pelo modelo. O *overhead* segue o mesmo padrão do custo de comunicação.

A tabela 5.18 mostra os resultados para o ambiente heterogêneo onde $N_{op} = 16$ para $P = 16$, $I_e = 18,730$ s e $\text{Soma}_{Rp} = 2,78,\ 5,56,\ 11,12$ e $22,24$ para $P = 2,\ 4,\ 8$ e 16 respectivamente.

Tabela 5.18: Resultados para o *kernel* MG usando 2, 4, 8 e 16 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

MG INTEL-AMD	**2 Nós**		**4 Nós**		**8 Nós**		**16 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado	Real	Estimado
T. de Computação	77,3	67,4	37,9	33,6	18,5	16,8	10,1	8,4
Desvio Padrão	1,1%	-	0,3%	-	1,8%	-	2,8%	-
Erro Percentual	-	12,9%	-	11,2%	-	9,3%	-	17,0%
T. de Comunicação	4,5	4,5	9,4	9,0	20,5	18,1	41,7	36,2
Desvio Padrão	1,6%	-	2,5%	-	3,9%	-	0,5%	-
Erro Percentual	-	0,1%	-	4,2%	-	11,5%	-	13,2%
T. Total de Execução	81,8	72,1	47,3	43,2	39,0	35,7	51,8	46,1
Desvio Padrão	1,2%	-	0,5%	-	2,3%	-	2,7%	-
Erro Percentual	-	11,9%	-	9,0%	-	8,6%	-	11,1%
Overhead	0,2	-	0,4	-	0,7	-	1,5	-

O erro percentual do tempo total estimado permaneceu flutuando entre 8,6% e 11,9%,

com uma variação máxima de 9,7 segundo em um tempo total de 81,8 segundos. Para o tempo de computação envolvendo 16 nós heterogêneos, o erro percentual foi de 17,0%, no entanto ele representa uma diferença de 1,7 segundos em 10,1 segundos, um erro pequeno em termos absolutos. Para a comunicação o maior erro foi de 13,2% para um número maior de nós e o menor erro foi de 0,1% para um número menor de nós.

5.3.5 EP

O *kernel* embaraçosamente paralelo (EP) gera pares de desvios pseudoaleatórios Gausianos e classifica o número de pares em sucessivos anéis quadráticos, um problema típico de muitas aplicações baseadas no problema de Monte Carlo. A comunicação acontece somente no final da computação, onde uma rotina MPI é usada para combinar a soma gerada em todos os processos. A classe C deste problema foi utilizada para a validação. O algoritmo 6 apresenta o pseudo-código deste *kernel*.

Algoritmo 6: Pseudo-código do *kernel* EP

```
1 início
2 |  gera as sementes para cada processo;
3 |  calcula os contadores e as somas em cada processo;
4 |  uso do MPI_Allreduce para enviar o resultado final para todos os processos;
5 fim
```

Como este *kernel* utiliza uma única iteração, a forma do cálculo do tempo de computação foi alterada para que o modelo realizasse a predição. Um novo método foi utilizado para calcular o R_p: a própria aplicação, considerando um tamanho menor (classe A), foi utilizada como *benchmark* para coletar a capacidade de processamento de dados por passo de tempo para cada tipo de unidade computacional. Com isso, dado o tamanho do problema (**size**) e o R_p para cada dispositivo, tem-se que o tempo de computação é dado por size/R_p.

Para ambos os ambientes homogêneos e heterogêneo, os parâmetros para estimar o tempo de computação foram: $I = 1$, $size = 8.589.934,592$ unidades, $R_p^{\text{AMD}} = 14.518.343,266$ unidades/s, $R_p^{\text{Intel}} = 24.350.979,353$ unidades/s (ambas multiplicadas pelo número de processadores de cada tipo - AMD e Intel - usados) e $F_r = 0$.

O custo de comunicação envolveu somente operações do tipo MPI_AllReduce,

modelada pela equação 5.10.

$$T_{AllReduce} = N_{op} \times log_2 P \times (L_d + \frac{M}{B_d} + o_d). \quad (5.10)$$

Em cada iteração são realizadas 3 operações. As primeiras com $M = 8 \times 10^{-6}$ MB e a última com 8×10^{-5} MB. Todas as três operações com $N_{op} = 1$. As tabelas 5.19, 5.20 e 5.21 mostram os resultados para os ambientes homogêneos.

Tabela 5.19: Resultados para o *kernel* EP usando 2, 4 e 8 nós homogêneos AMD-Ethernet. Todos os tempos estão em segundos (s).

EP AMD-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	295,6	295,8	147,8	147,9	71,3	74,0
Desvio Padrão	0,7%	-	0,2%	-	0,2%	-
Erro Percentual	-	0,1%	-	0,1%	-	3,8%
Tempo de Comunicação	$2{,}0 \times 10^{-4}$	$2{,}1 \times 10^{-4}$	$4{,}1 \times 10^{-4}$	$4{,}2 \times 10^{-4}$	$5{,}9 \times 10^{-4}$	$6{,}4 \times 10^{-4}$
Desvio Padrão	10,3%	-	10,5%	-	9,2%	-
Erro Percentual	-	6,9%	-	3,3%	-	7,0%
Tempo Total de Execução	295,6	295,8	147,8	147,9	71,3	74,0
Desvio Padrão	0,7%	-	0,2%	-	0,2%	-
Erro Percentual	-	0,1%	-	0,1%	-	3,8%
Overhead	$2{,}2 \times 10^{-6}$	-	$8{,}8 \times 10^{-6}$	-	$1{,}8 \times 10^{-5}$	-

Tabela 5.20: Resultados para o *kernel* EP usando 2, 4 e 8 nós homogêneos AMD-Infiniband. Todos os tempos estão em segundos (s).

EP AMD-INF	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	297,2	295,8	145,9	147,9	71,2	74,0
Desvio Padrão	1,0%	-	0,8%	-	1,1%	-
Erro Percentual	-	0,5%	-	1,4%	-	3,9%
Tempo de Comunicação	$1{,}4 \times 10^{-5}$	$1{,}6 \times 10^{-5}$	$3{,}8 \times 10^{-5}$	$3{,}5 \times 10^{-5}$	$8{,}7 \times 10^{-5}$	$9{,}4 \times 10^{-5}$
Desvio Padrão	11,9%	-	12,5%	-	12,8%	-
Erro Percentual	-	14,5%	-	7,5%	-	8,9%
Tempo Total de Execução	297,2	295,8	145,9	147,9	71,2	74,0
Desvio Padrão	1,0%	-	0,8%	-	1,1%	-
Erro Percentual	-	0,5%	-	1,4%	-	3,9%
Overhead	$3{,}7 \times 10^{-7}$	-	$1{,}4 \times 10^{-6}$	-	$2{,}8 \times 10^{-6}$	-

Tabela 5.21: Resultados para o *kernel* EP usando 2, 4 e 8 nós homogêneos Intel-Ethernet. Todos os tempos estão em segundos (s).

EP INTEL-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	176,0	176,4	87,9	88,2	40,8	44,1
Desvio Padrão	0,9%	-	1,2%	-	1,3%	-
Erro Percentual	-	0,2%	-	0,3%	-	8,0%
Tempo de Comunicação	$2{,}4 \times 10^{-4}$	$2{,}1 \times 10^{-4}$	$4{,}8 \times 10^{-4}$	$4{,}2 \times 10^{-4}$	$5{,}9 \times 10^{-4}$	$6{,}4 \times 10^{-4}$
Desvio Padrão	13,3%	-	9,7%	-	11,6%	-
Erro Percentual	-	10,5%	-	12,4%	-	8,0%
Tempo Total de Execução	176,0	176,4	87,9	88,2	40,8	44,1
Desvio Padrão	0,9%	-	1,2%	-	1,3%	-
Erro Percentual	-	0,2%	-	0,3%	-	8,0%
Overhead	$2{,}1 \times 10^{-6}$	-	$8{,}3 \times 10^{-6}$	-	$1{,}7 \times 10^{-5}$	-

O modelo previu, com boa precisão, o comportamento do *kernel* EP, sendo o valor do maior erro no tempo total estimado para execução igual a 8,0%. Como o custo de comunicação é inferior ao custo de computação, o erro do tempo total é um reflexo do erro de computação. Apesar de valores mínimos para o tempo de comunicação, o modelo conseguiu prever resultados com a mesma ordem de grandeza dos valores mensurados, sendo o maior erro igual à 14,5%. Como o tamanho de cada mensagem é pequeno, o *overhead*, que também apresentou valores baixos, teve pouca influência no erro final.

A tabela 5.22 mostra os resultados para o ambiente heterogêneo.

Tabela 5.22: Resultados para o *kernel* EP usando 2, 4, 8 e 16 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

EP INTEL-AMD	**2 Nós**		**4 Nós**		**8 Nós**		**16 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado	Real	Estimado
T. de Computação	235,6	221,0	118,0	110,5	52,0	55,2	28,5	27,6
Desvio Padrão	1,2%	-	0,4%	-	0,5%	-	2,8%	-
Erro Percentual	-	6,2%	-	6,4%	-	6,3%	-	3,2%
T. de Comunicação	$2{,}6 \times 10^{-4}$	$2{,}1 \times 10^{-4}$	$4{,}5 \times 10^{-4}$	$4{,}2 \times 10^{-4}$	$7{,}7 \times 10^{-4}$	$6{,}4 \times 10^{-4}$	$8{,}9 \times 10^{-4}$	$8{,}5 \times 10^{-4}$
Desvio Padrão	10,7%	-	14,8%	-	10,4%	-	8,2%	-
Erro Percentual	-	17,1%	-	6,3%	-	17,3%	-	4,6%
T. Total de Execução	235,6	221,0	118,0	110,5	52,0	55,2	28,5	27,6
Desvio Padrão	1,2%	-	0,4%	-	0,5%	-	2,8%	-
Erro Percentual	-	6,2%	-	6,4%	-	6,3%	-	3,2%
Overhead	$4{,}3 \times 10^{-6}$	-	$8{,}6 \times 10^{-6}$	-	$1{,}7 \times 10^{-5}$	-	$3{,}4 \times 10^{-5}$	-

Os resultados para o ambiente heterogêneo mostram uma flutuação do erro em torno de 6,4% quando até 8 máquinas são utilizadas, e uma queda para 3,2% ao se empregar 16 nós. Trata-se portanto de um erro médio maior se comparado ao ambiente homogêneo. A estimativa para a comunicação exibiu um comportamento semelhante ao da configuração com máquinas homogêneas. Observa-se que em todos os cenários o desvio padrão relativo à comunicação apresentou valores elevados, mesmo aumentando-se o número de execuções deste *kernel*. O principal motivo para o desvio padrão percentual ser alto é o fato dos

tempos de comunicação serem muito pequenos, de modo que qualquer pequena flutuação nos tempos causa grande variação nos desvios.

5.3.6 LU

Esta aplicação resolve um sistema de equações diferenciais parciais (EDPs) usando o método *Lower-Upper (LU) symmetric Gauss-Seidel*. A aplicação LU encontra a solução para um sistema triangular inferior e superior, por bloco (5×5) e esparso. Comparado com os outros dois métodos usados para o mesmo fim, BT e SP, este apresenta uma quantidade limitada de paralelismo [3]. A classe B deste problema foi utilizada para a validação. O algoritmo 7 apresenta o pseudo-código desta aplicação.

Algoritmo 7: Pseudo-código da aplicação LU

```
1 início
2   para i ← 1; i ≤ I faça
3     define os valores de borda do grid para a variáveis dependentes de cada
      processo;
4     computa o termo de crescimento baseado na solução exata prescrita;
5     envia/recebe duas linhas de dados com os valores de borda;
6     troca de informações referente às direções norte, sul, leste e oeste usando
      MPI_Send_Receive;
7   fim para
8   computa o erro da solução e a integral de superfície;
9 fim
```

Para os ambientes homogêneos, os parâmetros para estimar o tempo de computação foram: $I = 250$, $I_s = 25$, $I_e = 44,535$ s para AMD e $I_e = 32,525$ s para Intel, Soma_{Rp} é igual ao número de processos e $F_r = 0$.

O custo de comunicação envolveu somente operações do tipo MPI_Send e MPI_Receive, modeladas pela equação 5.11.

$$T_{SendReceive} = N_{op} \times (L_d + \frac{M}{B_d} + o_d). \tag{5.11}$$

Em cada iteração são realizadas 2 operações. A primeira com $N_{op} = 2$, 4 e 8, e a segunda com $N_{op} = 100$, 200 e 400 para os respectivos $P = 2$, 4 e 8. O número de operações de comunicação pode parecer alto para a segunda operação, no entanto ele se justifica devido à necessidade de operações adicionais serem necessárias para se encontrar os vizinhos, uma vez que quando os dados são linearizados é preciso saber onde

a informação para o próximo passo está localizada. Operações de envio/recebimento são realizadas para cada bloco da matriz e para cada direção (norte, sul, leste e oeste). Além disso, o tamanho das mensagens é de $M = 0,462$ MB para a primeira operação e 2×10^{-3} MB para a segunda operação. Para a segunda operação, o valor transmitido é ligeiramente superior ao valor do MTU das redes, o que implica que cada mensagem tenha de ser divida em dois pacotes para ser transmitida pelas redes. As tabelas 5.23, 5.24 e 5.25 mostram os resultados para os ambientes homogêneos.

Tabela 5.23: Resultados para a aplicação LU usando 2, 4 e 8 nós homogêneos AMD-Ethernet. Todos os tempos estão em segundos (s).

LU AMD-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	221,6	222,6	110,0	111,3	60,0	55,7
Desvio Padrão	0,6%	-	1,0%	-	0,7%	-
Erro Percentual	-	0,5%	-	1,2%	-	7,3%
Tempo de Comunicação	6,6	6,8	8,2	9,1	17,0	18,2
Desvio Padrão	1,9%	-	3,1%	-	0,5%	-
Erro Percentual	-	2,9%	-	11,6%	-	7,1%
Tempo Total de Execução	228,2	229,6	118,1	120,8	77,0	74,7
Desvio Padrão	0,6%	-	1,2%	-	0,7%	-
Erro Percentual	-	0,6%	-	2,3%	-	3,0%
Overhead	0,2	-	0,4	-	0,9	-

Tabela 5.24: Resultados para a aplicação LU usando 2, 4 e 8 nós homogêneos AMD-Infiniband. Todos os tempos estão em segundos (s).

LU AMD-INF	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	222,1	222,6	109,3	111,3	60,3	55,7
Desvio Padrão	0,6%	-	0,8%	-	0,8%	-
Erro Percentual	-	0,2%	-	1,9%	-	7,7%
Tempo de Comunicação	0,7	0,6	0,8	0,9	1,7	1,5
Desvio Padrão	3,4%	-	7,6%	-	1,9%	-
Erro Percentual	-	10,8%	-	11,1%	-	7,8%
Tempo Total de Execução	222,8	223,2	110,1	112,1	62,0	57,2
Desvio Padrão	0,7%	-	0,8%	-	0,9%	-
Erro Percentual	-	0,2%	-	1,8%	-	7,7%
Overhead	$8,7 \times 10^{-3}$	-	$1,7 \times 10^{-2}$	-	$3,5 \times 10^{-2}$	-

Tabela 5.25: Resultados para a aplicação LU usando 2, 4 e 8 nós homogêneos Intel-Ethernet. Todos os tempos estão em segundos (s).

LU INTEL-ETH	**2 Nós**		**4 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	165,6	162,6	84,1	81,3	37,9	38,2
Desvio Padrão	0,3%	-	0,5%	-	0,9%	-
Erro Percentual	-	1,8%	-	3,3%	-	0,6%
Tempo de Comunicação	6,4	6,8	8,1	9,1	17,0	18,2
Desvio Padrão	1,8%	-	2,5%	-	0,9%	-
Erro Percentual	-	5,5%	-	12,8%	-	6,9%
Tempo Total de Execução	172,0	169,6	92,1	90,7	55,0	56,7
Desvio Padrão	0,3%	-	0,5%	-	0,9%	-
Erro Percentual	-	1,4%	-	1,5%	-	3,7%
Overhead	0,2	-	0,3	-	0,6	-

O maior erro percentual na estimativa do tempo total de execução em ambientes homogêneos foi de 7,7%. Quando os componentes do tempo de execução são analisados individualmente, percebe-se que na estimativa do tempo de comunicação o maior erro foi de 12,8% e na estimativa do tempo de computação foi de 7,7%. Apesar da quantidade de operações de comunicação, seu custo é pequeno, de modo que o erro de computação é o que apresenta uma maior influência no erro total. O valor do *overhead* se manteve baixo; isso ocorreu devido ao fato do *overhead* para o tamanho de mensagem igual à 2×10^{-3} MB ser de $3,5 \times 10^{-7}$ s para envio e $2,7 \times 10^{-7}$ s para recebimento.

A tabela 5.26 mostra os resultados para o ambiente heterogêneo, onde $N_{op} = 800$ para $P = 16$, $I_e = 44,535$ s e $\text{Soma}_{Rp} = 2,78$, $5,56$, $11,12$ e $22,24$ para $P = 2$, 4, 8 e 16 respectivamente.

Tabela 5.26: Resultados para a aplicação LU usando 2, 4, 8 e 16 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

LU INTEL-AMD	**2 Nós**		**4 Nós**		**8 Nós**		**16 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado	Real	Estimado
T. de Computação	193,7	160,2	97,6	80,1	43,3	40,0	23,0	20,0
Desvio Padrão	0,7%	-	0,7%	-	0,3%	-	0,3%	-
Erro Percentual	-	17,3%	-	17,9%	-	7,6%	-	13,0%
T. de Comunicação	6,7	6,8	7,9	9,0	16,9	18,1	39,7	36,3
Desvio Padrão	3,5%	-	0,4%	-	1,4%	-	3,0%	-
Erro Percentual	-	0,9%	-	14,0%	-	7,0%	-	8,4%
T. Total de Execução	200,4	167,2	105,6	89,5	60,2	58,9	62,7	57,9
Desvio Padrão	0,8%	-	0,6%	-	1,4%	-	1,9%	-
Erro Percentual	-	16,6%	-	15,1%	-	2,0%	-	7,4%
Overhead	0,2	-	0,4	-	0,8	-	1,7	-

O maior erro apresentado para o tempo total estimado foi de 16,6%, representando uma variação de 33,2 segundos em um tempo de 200,4 segundos. Erros mais elevados para o tempo de computação foram apresentados para as configurações iniciais, sofrendo uma queda para as configurações com mais máquinas. Para 8 nós heterogêneos o modelo

obteve uma boa aproximação, com um erro de 2,0%. Os erros de comunicação ficaram abaixo de 14,0%, resultado de uma diferença de 1,1 segundos de um total de 7,9 segundos.

5.3.7 BT

Esta aplicação resolve um sistema de equações diferenciais parciais (EDPs) usando o método tri-diagonal por bloco (BT). A aplicação BT resolve um sistema de equações independentes não dominantes diagonalmente, tri-diagonais por bloco usando tamanho de bloco 5 $\times$ 5 [3]. A classe C deste problema foi utilizada para a validação. O algoritmo 8 apresenta o pseudo-código desta aplicação.

Algoritmo 8: Pseudo-código da aplicação BT

```
1 início
2   para i ← 1; i ≤ I faça
3     envia as faces superior, inferior, leste, oeste, norte e sul usando
      MPI_Send_Receive;
4     executa a solução de linha para a direção x;
5     executa a solução de linha para a direção y;
6     executa a solução de linha para a direção z;
7     atualiza o vetor u;
8   fim para
9 fim
```

Para os ambientes homogêneos, os parâmetros para estimar o tempo de computação foram: $I = 200$, $I_s = 20$, $I_e = 231,620$ s para AMD e $I_e = 133,560$ s para Intel, Soma_{Rp} é igual ao número de processos e $F_r = 0$.

O custo de comunicação envolveu somente operações do tipo MPI_Send e MPI_Receive, modeladas pela equação 5.12.

$$T_{SendReceive} = N_{op} \times (L_d + \frac{M}{B_d} + o_d). \tag{5.12}$$

Em cada iteração são realizadas 3 operações com $N_{op} = 6$, 3 e 3, e $M = 0,525$ MB, $0,202$ MB e $0,269$ MB para as respectivas operações. Em todas as operações existe a troca mensagens entre todos os processos, portanto elas devem ser multiplicadas por $P - 1$. Estes valores são válidos para todas as configurações. As tabelas 5.27, 5.28 e 5.29 mostram os resultados para os ambientes homogêneos. Como o *benchmark* exige que o número de processos utilizados seja um quadrado perfeito e o ambiente possui somente

8 nós com processadores Intel, o ambiente homogêneo para Intel foi executado somente com 4 nós.

Tabela 5.27: Resultados para a aplicação BT usando 4 e 9 nós homogêneos AMD-Ethernet. Todos os tempos estão em segundos (s).

BT AMD-ETH	**4 Nós**		**9 Nós**	
	Real	Estimado	Real	Estimado
Tempo de Computação	532,8	579,1	285,1	257,4
Desvio Padrão	0,5%	-	0,6%	-
Erro Percentual	-	8,7%	-	9,7%
Tempo de Comunicação	34,0	29,8	86,0	79,5
Desvio Padrão	1,2%	-	0,9%	-
Erro Percentual	-	12,4%	-	7,6%
Tempo Total de Execução	566,8	610,2	371,1	340,5
Desvio Padrão	0,5%	-	0,7%	-
Erro Percentual	-	7,7%	-	8,3%
Overhead	1,4	-	3,7	-

Tabela 5.28: Resultados para a aplicação BT usando 4 e 9 nós homogêneos AMD-Infiniband. Todos os tempos estão em segundos (s).

BT AMD-INF	**4 Nós**		**9 Nós**	
	Real	Estimado	Real	Estimado
Tempo de Computação	533,3	579,1	286,7	257,4
Desvio Padrão	0,9%	-	0,8%	-
Erro Percentual	-	8,6%	-	10,2%
Tempo de Comunicação	3,2	2,7	8,0	7,2
Desvio Padrão	2,4%	-	1,5%	-
Erro Percentual	-	15,9%	-	10,2%
Tempo Total de Execução	536,5	581,8	294,7	264,5
Desvio Padrão	1,0%	-	1,0%	-
Erro Percentual	-	8,4%	-	10,2%
Overhead	0,1	-	0,2	-

Tabela 5.29: Resultados para a aplicação BT usando 4 nós homogêneos Intel-Ethernet. Todos os tempos estão em segundos (s).

BT INTEL-ETH	**4 Nós**	
	Real	Estimado
Tempo de Computação	369,1	333,9
Desvio Padrão	0,4%	-
Erro Percentual	-	9,5%
Tempo de Comunicação	34,2	29,8
Desvio Padrão	0,8%	-
Erro Percentual	-	12,8%
Tempo Total de Execução	403,3	364,9
Desvio Padrão	0,6%	-
Erro Percentual	-	9,5%
Overhead	1,2	-

O maior erro percentual no tempo total de execução foi de 10,2% para a configuração de 9 nós AMD-Infiniband. O maior erro percentual das operações de comunicação foi de 15,9% para a mesma configuração, porém usando 4 nós, mas tal erro representa uma variação de 0,5 segundos em um total de 3,2 segundos. O erro de computação de todas as configurações ficou próximo de 10,2%.

A tabela 5.30 mostra os resultados para o ambiente heterogêneo, onde $I_e = 231,620$ s e $\text{Soma}_{Rp} = 5,56$, $12,9$ e $22,24$ para $P = 4$, 9 e 16 respectivamente.

Tabela 5.30: Resultados para a aplicação BT usando 4, 9 e 16 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

BT INTEL-AMD	**4 Nós**		**9 Nós**		**16 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	460,4	416,5	157,0	179,5	89,5	104,1
Desvio Padrão	2,6%	-	0,1%	-	0,2%	-
Erro Percentual	-	9,5%	-	14,3%	-	16,2%
Tempo de Comunicação	34,5	29,8	87,3	79,4	156,3	149,0
Desvio Padrão	0,5%	-	0,5%	-	0,5%	-
Erro Percentual	-	13,8%	-	8,9%	-	4,6%
Tempo Total de Execução	494,9	447,6	244,3	262,3	245,8	259,5
Desvio Padrão	2,4%	-	0,2%	-	0,3%	-
Erro Percentual	-	9,5%	-	7,3%	-	5,5%
Overhead	1,3	-	3,4	-	6,4	-

Boas aproximações foram obtidas pelo modelo para o tempo total de execução no ambiente heterogêneo, onde o maior erro percentual foi de 9,5% para 4 nós. Para a comunicação, o maior erro foi observado para 4 nós (13,8%), enquanto que o maior erro para a computação foi observado para a configuração de 16 nós (16,2%). Apesar do erro na estimativa do tempo de computação aumentar com o acréscimo de nós, o erro na

estimativa do tempo total de execução diminui. Isto ocorre porque, com o aumento no número de nós envolvidos na execução, o tempo de comunicação aumenta e o tempo de computação diminui. Consequentemente, o tempo de comunicação passa a representar uma maior parcela do tempo total de execução, enquanto o tempo de computação reduz sua parcela de contrubição. Deste modo, mesmo com um aumento crescente no erro de estimativa do tempo de computação, este acaba se tornando cada vez menos expressivo no erro do tempo total de execução.

5.3.8 SP

Esta aplicação resolve um sistema de equações diferenciais parciais (EDPs) usando o método escalar penta-diagonal (SP). A aplicação SP resolve um sistema de equações independentes, escalar penta-diagonal e diagonalmente não-dominadas usando o esquema de multi-partição [3]. A classe C deste problema foi utilizada para a validação. O algoritmo 9 apresenta o pseudo-código desta aplicação.

Algoritmo 9: Pseudo-código da aplicação SP

```
1  início
2      para i ← 1; i ≤ I faça
3          executa a multiplicação do vetor pela matriz diagonal por blocos;
4          MPI_Isend e MPI_Ireceive são usados para enviar/receber o buffer;
5          executa a fatoração aproximada no plano x;
6          MPI_Isend e MPI_Ireceive são usados para enviar/receber o buffer;
7          executa a fatoração aproximada no plano y;
8          MPI_Isend e MPI_Ireceive são usados para enviar/receber o buffer;
9          executa a fatoração aproximada no plano z;
10         MPI_Isend e MPI_Ireceive é usado para enviar/receber o buffer;
11         adiciona o vetor u;
12     fim para
13 fim
```

Para os ambientes homogêneos, os parâmetros para estimar o tempo de computação foram: $I = 400$, $I_s = 40$, $I_e = 230,680$ s para AMD e $I_e = 165,48$ s para Intel, Soma_{Rp} é igual ao número de processos e $F_r = 0$.

O custo de comunicação envolveu somente operações do tipo MPI_Send e MPI_Receive, que foram modeladas pela equação 5.13.

$$T_{SendReceive} = N_{op} \times (L_d + \frac{M}{B_d} + o_d). \tag{5.13}$$

Para esta aplicação, cada P apresenta um padrão de troca de mensagens diferentes que são sumarizados da seguinte maneira:

a) Para $P = 4$, temos as seguintes mensagens:

- $N_{op} = 3$ e $M = 0,512$ MB, entre todos os processos, portanto a operação é multiplicada por $P - 1$,
- $N_{op} = 1$ e $M = 0,525$ MB,
- $N_{op} = 3$ e $M = 0,134$ MB.

b) Para $P = 9$:

- $N_{op} = 1$ e $M = 0,503$ MB, entre todos os processos, portanto a operação é multiplicada por $P - 1$,
- $N_{op} = 6$ e $M = 0,466$ MB,
- $N_{op} = 6$ e $M = 0,494$ MB,
- $N_{op} = 6$ e $M = 0,229$ MB,
- $N_{op} = 6$ e $M = 0,225$ MB,
- $N_{op} = 6$ e $M = 0,513$ MB,
- $N_{op} = 6$ e $M = 0,233$ MB.

c) Para $P = 16$ temos o seguinte padrão de mensagens:

- $N_{op} = 1$ e $M = 0,2816$ MB, entre todos os processos, portanto a operação é multiplicada por $P - 1$,
- $N_{op} = 10$ e $M = 0,289$ MB,
- $N_{op} = 10$ e $M = 0,274$ MB,
- $N_{op} = 10$ e $M = 0,394$ MB,
- $N_{op} = 10$ e $M = 0,396$ MB,
- $N_{op} = 10$ e $M = 0,281$ MB,
- $N_{op} = 10$ e $M = 0,397$ MB,
- $N_{op} = 10$ e $M = 0,396$ MB,

- $N_{op} = 10$ e $M = 0,390$ MB,
- $N_{op} = 10$ e $M = 0,391$ MB,
- $N_{op} = 10$ e $M = 0,295$ MB,
- $N_{op} = 10$ e $M = 0,268$ MB,
- $N_{op} = 10$ e $M = 0,128$ MB.

As tabelas 5.31, 5.32 e 5.33 mostram os resultados para os ambientes homogêneos.

Tabela 5.31: Resultados para a aplicação SP usando 4 e 9 nós homogêneos AMD-Ethernet. Todos os tempos estão em segundos (s).

SP AMD-ETH	**4 Nós**		**9 Nós**	
	Real	Estimado	Real	Estimado
Tempo de Computação	485,0	576,7	228,5	256,3
Desvio Padrão	0,7%	-	0,9%	-
Erro Percentual	-	18,9%	-	12,1%
Tempo de Comunicação	28,5	24,1	80,4	74,0
Desvio Padrão	3,3%	-	0,4%	-
Erro Percentual	-	15,6%	-	8,0%
Tempo Total de Execução	513,5	601,7	309,0	334,9
Desvio Padrão	0,6%	-	1,0%	-
Erro Percentual	-	17,2%	-	8,4%
Overhead	0,9	-	4,6	-

Tabela 5.32: Resultados para a aplicação SP usando 4 e 9 nós homogêneos AMD-Infiniband. Todos os tempos estão em segundos (s).

SP AMD-INF	**4 Nós**		**9 Nós**	
	Real	Estimado	Real	Estimado
Tempo de Computação	483,5	576,7	227,3	256,3
Desvio Padrão	0,7%	-	0,8%	-
Erro Percentual	-	19,3%	-	12,8%
Tempo de Comunicação	1,9	2,2	11,4	10,1
Desvio Padrão	9,5%	-	3,9%	-
Erro Percentual	-	14,0%	-	6,8%
Tempo Total de Execução	485,4	578,9	238,7	266,5
Desvio Padrão	0,7%	-	0,9%	-
Erro Percentual	-	19,3%	-	12,7%
Overhead	3,9 $\times 10^{-2}$	-	0,2	-

Tabela 5.33: Resultados para a aplicação SP usando 4 nós homogêneos Intel-Ethernet. Todos os tempos estão em segundos (s).

SP INTEL-ETH	**4 Nós**	
	Real	Estimado
Tempo de Computação	364,6	414,2
Desvio Padrão	0,6%	-
Erro Percentual	-	13,6%
Tempo de Comunicação	28,6	24,1
Desvio Padrão	1,2%	-
Erro Percentual	-	15,9%
Tempo Total de Execução	393,2	439,1
Desvio Padrão	0,7%	-
Erro Percentual	-	11,7%
Overhead	0,9	-

Esta aplicação apresentou um erro no tempo estimado de execução de 19,3% para o ambiente que utiliza processadores AMD. A comunicação é pouco expressiva para todos os ambientes homogêneos, então a maior influência deste erro foi o tempo de computação, cujo erro chegou a 19,3%. Quando o número de processos aumenta para 9, o erro de computação diminui para 12,8%. O maior erro de comunicação foi de 15,6%, também para configuração de 4 nós AMD-Ethernet. Para o ambiente homogêneo Intel, o erro de computação foi de 13,6%, menor do que o observado no ambiente AMD, enquanto que o erro de comunicação foi um pouco maior, 15,9%. O erro na estimativa do tempo total de execução foi de 11,7%.

A tabela 5.34 mostra os resultados para o ambiente heterogêneo, onde $I_e = 230,680$ s e $\text{Soma}_{Rp} = 5,56$, $12,9$ e $22,24$ para $P = 4$, 9 e 16 respectivamente.

Tabela 5.34: Resultados para a aplicação SP usando 4, 9 e 16 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

SP INTEL-AMD	**4 Nós**		**9 Nós**		**16 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado
Tempo de Computação	414,4	414,9	181,5	178,8	130,2	103,7
Desvio Padrão	3,2%	-	2,1%	-	3,1%	-
Erro Percentual	-	0,1%	-	1,5%	-	20,3%
Tempo de Comunicação	28,0	24,1	83,5	74,0	213,0	188,8
Desvio Padrão	1,6%	-	1,2%	-	1,0%	-
Erro Percentual	-	14,2%	-	11,4%	-	11,4%
Tempo Total de Execução	442,4	439,0	265,0	257,4	343,2	305,1
Desvio Padrão	3,1%	-	1,6%	-	1,7%	-
Erro Percentual	-	0,8%	-	2,9%	-	11,1%
Overhead	0,9	-	4,5	-	13,0	-

Ao contrário do que aconteceu com o ambiente homogêneo, o maior erro para o tempo

total é apresentado para a configuração que utiliza 16 nós, 11,1%. No entanto, observa-se que o erro foi bem menor do que no caso homogêneo. Para 4 nós o modelo foi preciso em suas estimativas de tempo total de execução, apresentando um erro de 0,8%. O erro referente a comunicação permaneceu flutuando entre 11,4% e 14,2%. Para a estimativa do tempo de computação, o erro foi maior do que no ambiente homogêneo, chegando a 20,3% para 16 nós. Este erro não foi refletido no tempo total porque para 16 nós a comunicação passa a ser o item de maior impacto.

5.3.9 HIS GPU

Um simulador de três dimensões do Sistema Imune Humano (HIS) [4] foi usado para a validação do modelo utilizando GPUs. O simulador implementa um modelo matemático que usa um conjunto de equações diferenciais parciais (EDPs) para descrever como algumas células e moléculas envolvidas na resposta imune inata reagem a um patógeno. A implementação é baseada no método das diferenças finitas [34] para a discretização espacial e o método de Euler explícito para a evolução no tempo [35, 36]. O código foi implementado em C e utiliza CUDA para resolver as equações diferenciais parciais nas GPUs. Ao fim de cada passo temporal, faz-se necessária a troca de bordas entre GPUs, para que a próxima etapa de computação comece. As CPUs são responsáveis pela troca de bordas, e primitivas MPI foram utilizadas para este fim. Uma malha de tamanho $200 \times 200 \times 200$ foi usada na validação do modelo. Para esta aplicação, a comunicação entre CPU e GPU não foi considerada, uma vez que o tempo de comunicação pelo controlador de rede é maior do que a troca de dados utilizando o soquete PCIe. O algoritmo 10 da uma visão geral do simulador HIS.

Os parâmetros para estimar o tempo de computação foram: $I = 10000$, $I_s = 100$, $I_e = 11,160$ s para C1060 e $F_r = 0$.

O custo de comunicação envolveu somente operações do tipo MPI_Send e MPI_Receive, modeladas pela equação 5.14.

$$T_{SendReceive} = N_{op} \times (L_d + \frac{M}{B_d} + o_d), \tag{5.14}$$

onde $N_{op} = 1$ e $M = 1,92$ MB para todas as configurações. Como a comunicação acontece entre todos os processos, o custo de comunicação deve ser multiplicado por

Algoritmo 10: Pseudo-código da aplicação HIS GPU

```
1 início
2   define tamanho de malha que vai ser computado por cada unidade
    computacional;
3   inicializa as sub-malhas de acordo com as condições inciais;
4   para t ← 1; t ≤ I faça
5     executa os kernels (GPUs) para computar os pontos;
6     usa MPI_Isend e MPI_Receive para efetuar a troca de borda entre as
      máquinas;
7     sincroniza todas as máquinas;
8   fim para
9 fim
```

$P-1$. A tabela 5.35 mostra os resultados para 4 (2 M2075 e 2 C1060), 5 (3 M2075 e 2 C1060), 7 (4 M2075 e 3 C1060) e 8 (4 M2075 e 4 C1060) nós.

Tabela 5.35: Resultados para a aplicação HIS-GPU usando 4, 5, 7 e 8 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

HIS INTEL-AMD	**4 Nós**		**5 Nós**		**7 Nós**		**8 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado	Real	Estimado
T. de Computação	149,6	157,5	132,6	115,9	109,7	84,7	95,4	78,7
Desvio Padrão	0,6%	-	0,8%	-	0,2%	-	0,9%	-
Erro Percentual	-	5,3%	-	12,6%	-	22,7%	-	17,4%
T. de Comunicação	689,8	618,7	833,9	825,0	1260,6	1237,5	1384,6	1443,7
Desvio Padrão	0,3%	-	0,5%	-	0,2%	-	0,6%	-
Erro Percentual	-	10,3%	-	1,1%	-	1,8%	-	4,3%
T. Total de Execução	839,4	810,5	966,6	983,9	1370,2	1390,8	1479,9	1591,1
Desvio Padrão	0,7%	-	0,6%	-	0,2%	-	0,6%	-
Erro Percentual	-	3,4%	-	1,8%	-	0,9%	-	7,5%
Overhead	34,0	-	43,0	-	60,0	-	69,0	-

O maior erro no tempo total estimado foi de 7,5%, correspondente a uma variação de 111,2 segundos de um total de 1.479,9 segundos. O maior erro de computação foi de 22,7%, no entanto, este erro não é expressivo no erro total pois a comunicação representa a maior fatia no tempo total. O maior erro de comunicação foi de 10,3% para a configuração inicial, e reduzido para 4,3% nas configurações subsequentes. O *overhead* é alto devido ao número de iterações do problema, sendo que para 16 nós, ele representa quase 73% do tempo de computação e cerca de 5% do tempo de comunicação.

5.3.10 HIS CPU-GPU

Esta aplicação implementa o mesmo simulador descrito na seção 5.3.9, com a diferença de que as CPUs são também incluídas na etapa de computação, e não ficam responsáveis somente pela comunicação. Desta forma, tanto CPUs quanto GPUs realizam o

processamento dos dados através de uma balanceamento de carga dinâmico, onde a aplicação começa com uma distribuição inicial de dados, e a medida que o laço principal avança, é feito o monitoramento do tempo de computação das CPUs e GPUs, e de acordo com este tempo, uma nova distribuição de dados é realizada de maneira a tentar melhorar o tempo de computação total. O algoritmo 11 dá uma visão geral do simulador HIS CPU-GPU.

Algoritmo 11: Pseudo-código da aplicação HIS CPU-GPU

```
1  início
2      define tamanho de malha que vai ser computado por cada unidade
       computacional;
3      inicializa as sub-malhas de acordo com as condições inciais;
4      para t ← 1; t ≤ I faça
5          se t % iterações para realizar balanceamento = 0 então
6              realiza balanceamento de carga dinâmico;
7          fim se
8          executa os kernels (GPUs) e as funções (CPUs) para computar os pontos;
9          usa MPI_Isend e MPI_Receive para efetuar a troca de borda entre as
           máquinas;
10         sincroniza todas as máquinas;
11     fim para
12 fim
```

Os parâmetros para estimar o tempo de computação foram: $I = 1000$, $I_s = 100$, $I_e = 1708,24$ s a CPU AMD e $F_r = 0$.

O custo de comunicação envolveu somente operações do tipo MPI_Send e MPI_Receive, descritas pela equação 5.15.

$$T_{SendReceive} = N_{op} \times (L_d + \frac{M}{B_d} + o_d), \tag{5.15}$$

onde $N_{op} = 1$ e $M = 1,28$ MB para todas as configurações. Como a comunicação acontece entre todos os processos, o custo de comunicação deve ser multiplicado por $P - 1$.

A tabela 5.36 mostra os resultados para 2 (1 AMD - M2075 - e 1 Intel - C1060), 3 (1 AMD - M2075 - e 2 Intel - C1060), 5 (2 AMD - M2075 - e 3 Intel - 1 C1060 e 2 M2050) e 7 (3 AMD - M2075 - e 4 Intel - 2 C1060 e 2 M2050) nós.

Tabela 5.36: Resultados para a aplicação HIS CPU-GPU usando 2, 3, 5 e 7 nós heterogêneos Intel-AMD-Ethernet. Todos os tempos estão em segundos (s).

HIS INTEL-AMD	**2 Nós**		**3 Nós**		**5 Nós**		**7 Nós**	
	Real	Estimado	Real	Estimado	Real	Estimado	Real	Estimado
T. de Computação	31,2	36,5	25,9	28,4	11,6	12,2	11,1	9,1
Desvio Padrão	0,7%	-	0,7%	-	0,3%	-	0,3%	-
Erro Percentual	-	17,0%	-	17,9%	-	4,5%	-	17,6%
T. de Comunicação	15,9	13,8	31,5	27,6	64,5	55,1	96,7	82,7
Desvio Padrão	3,5%	-	0,4%	-	1,4%	-	3,0%	-
Erro Percentual	-	0,9%	-	12,6%	-	14,6%	-	14,5%
T. Total de Execução	47,2	51,2	57,4	57,4	76,2	69,6	107,8	95,1
Desvio Padrão	0,8%	-	0,6%	-	1,4%	-	1,9%	-
Erro Percentual	-	8,6%	-	0,2%	-	8,6%	-	11,8%
Overhead	0,9	-	1,4	-	2,4	-	3,3	-

O maior erro para o tempo total de execução estimado foi de 11,8%, uma diferença de 12,7 segundos em um total de 107,8 segundos. O erro na estimativa do tempo de computação permaneceu flutuante ao redor de 17,0%, com uma redução para 4,5% utilizando 5 máquinas, o que é uma boa aproximação considerando o nível de heterogeneidade do ambiente (2 CPUs e 3 GPUs diferentes). O custo de comunicação foi estimado de forma precisa para 2 nós e ficou estagnado entre 12,6% e 14,5% para as outras configurações.

Para esta aplicação, o erro nas estimativas do tempo de computação médio, se comparado as outras aplicações, foi elevado. Isto se deve ao balanceamento de carga dinâmico, que tem como principal objetivo distribuir a carga entre as unidade de processamento de modo a minimizar a diferença na etapa de computação entre elas. Por exemplo, uma unidade de processamento duas vezes mais rápida que uma lenta, recebe uma quantidade de dados duas vezes maior do que a lenta, desta forma inibindo a diferença entre o poder de processamento com uma carga de dados maior. Para os casos heterogêneos o modelo considera Soma_{Rp}, como sendo a soma do valor normalizado para cada unidade computacional, o que prejudica sua precisão. Deve-se recordar que o modelo foi criado para lidar com aplicações regulares, e o escalonamento dinâmico usando por esta aplicação faz com que suas características de execução lembrem a de aplicações irregulares.

5.4 Resultados do Escalonador

O escalonador desenvolvido neste trabalho baseia-se nas características de uma aplicação paralela regular, descritas com o uso do modelo HCLogP, e do ambiente que será utilizado

para sua execução para identificar um subconjunto de processadores e de controladores de rede que minimizem o seu tempo total de execução. Esta seção tenta verificar experimentalmente se o escalonador cumpre o seu objetivo, usando para isso o mesmo conjunto de aplicações apresentado ao longo da seção anterior.

Algumas considerações devem ser feitas em relação aos resultados que serão apresentados neste capítulo:

- Para a criação dos gráficos, o algoritmo original do escalonador foi modificado de forma a considerar todas as configuração possíveis de execução da aplicação. Por exemplo, suponha que uma aplicação possa ser executada em 2, 4, 8 e 16 nós. Neste caso, o algoritmo obtém a configuração de hardware que minimiza o tempo de execução empregando 2 nós, 4 nós e assim sucessivamente até 16 nós. A execução original do algoritmo do escalonador, como apresentado no capítulo 4, iria apresentar somente uma configuração de hardware: aquela na qual o menor tempo total de execução pudesse ser obtido.

- O tempo total de execução pode ser diferente dos resultados apresentados nas tabelas da seção anterior, pois a configuração de execução da seção anterior foi escolhida *a priori* com o objetivo de validar o modelo. Já o objetivo do escalonador é encontrar o subconjunto de recursos que minimize o tempo total de execução de uma aplicação. Assim, os tempos de execução podem ser diferentes por conta dos recursos de hardware utilizados serem diferentes, ainda que o número total de recursos empregados na execução seja igual.

- Os tempos obtidos pelo escalonador não incluem o *overhead*, pois seria necessária a inclusão de um parâmetro que varia com o tamanho da mensagem e com as unidades de processamento, se tornando um processo exaustivo a sua inclusão no arquivo de entrada do escalonador.

5.4.1 FT

A figura 5.1 apresenta o gráfico com os tempos de execução estimados para a execução de FT usando as melhores configurações obtidas pelo escalonador. O escalonador encontrou a melhor configuração ao utilizar 8 processadores AMD, com a rede Infiniband, com o tempo estimado de 21,236 segundos. Como o custo de comunicação do *kernel* é alto,

uma rede mais rápida possui um impacto maior na escolha da melhor configuração, como refletidas nas tabelas da subseção 5.3.1, onde as melhores configurações utilizam a rede Infiniband. Para todas as execuções e todos os ambientes, o escalonador conseguiu capturar o comportamento da aplicação.

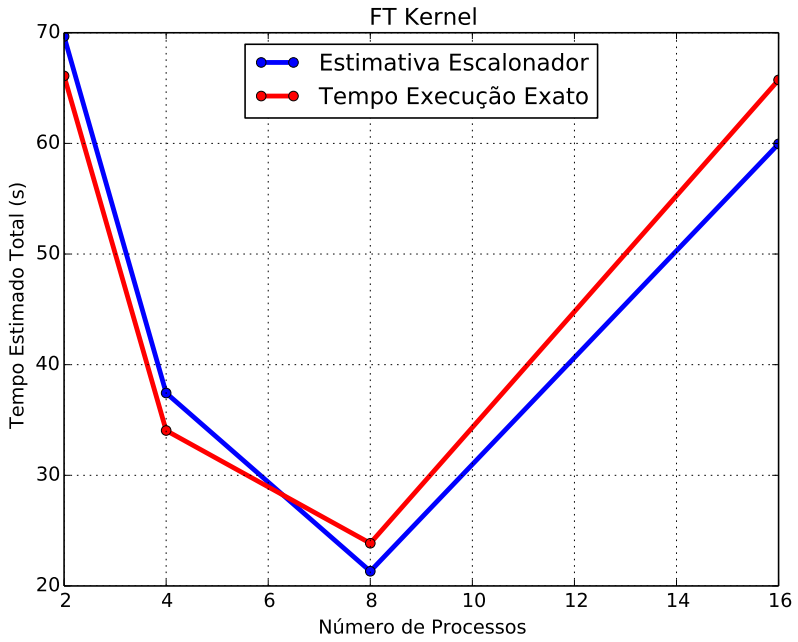

Figura 5.1: *Kernel* FT - Estimativa do escalonador comparado com o tempo real para vários nós.

5.4.2 IS

A figura 5.2 mostra que o escalonador obteve boas aproximações em suas estimativas para os tempos de execução de IS. Como o tempo de comunicação representa a maior parte do tempo total de execução, as escolhas para 2, 4 e 8 processos foram feitas utilizando a rede Infiniband. Os resultados são próximos aos melhores tempos mensurados para o *kernel* IS. A melhor configuração estimada é a que emprega 8 processadores AMD interconectados pela rede Infiniband, exatamente a mesma que verificou-se obter o menor tempo total de execução.

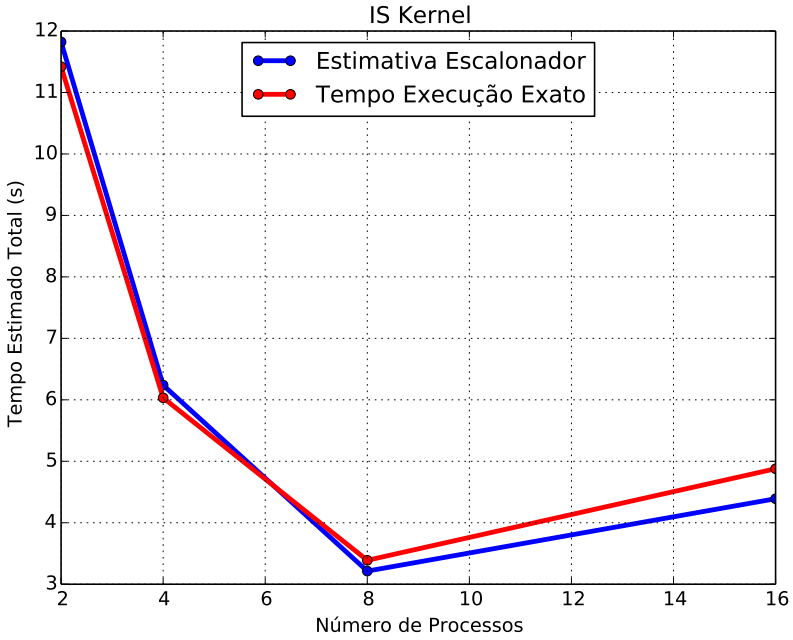

Figura 5.2: *Kernel* IS - Estimativa do escalonador comparado com o tempo real para vários nós.

5.4.3 CG

A figura 5.3 mostra que o escalonador teve sucesso em suas estimativas para os tempos de execução de CG. Para este *kernel*, como o custo de computação predomina para as configurações de 2 e 4 nós, a melhor configuração obtida foi a que utiliza processadores Intel interligados pela rede Ethernet. Somente para 8 processos que o custo de comunicação se torna significativo, o que compensou retirar um pouco do poder computacional (passando a utilizar as CPUs AMD) e usar a rede Infiniband. A melhor configuração obtida foi para 8 processadores AMD, interconectados pelo controlador de rede Infiniband, com estimativa de execução em aproximadamente 77 segundos. Deve-se destacar que o *overhead* não é considerado nas estimativas feitas pelo escalonador.

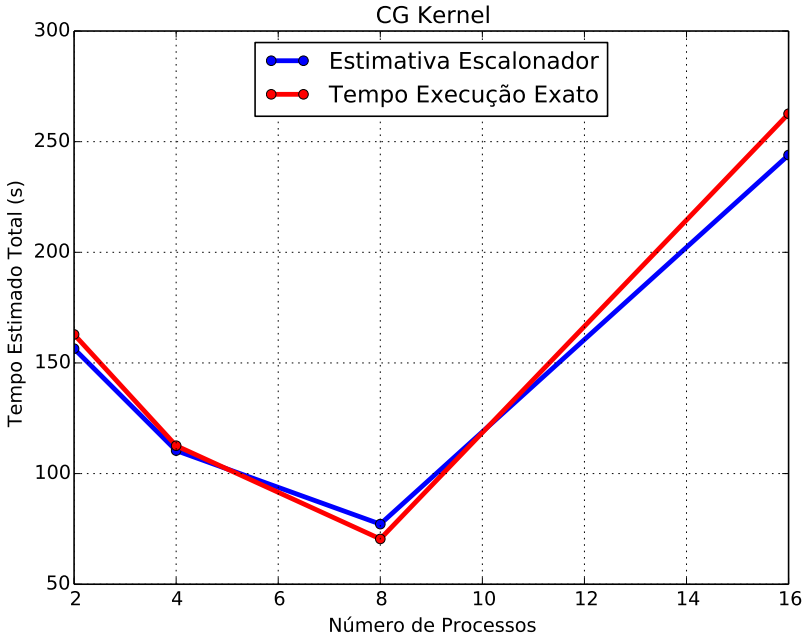

Figura 5.3: *Kernel* CG - Estimativa do escalonador comparado com o tempo real para vários nós.

5.4.4 MG

A figura 5.4 mostra as estimativas feitas pelo escalonador para o tempo total de execução de MG, considerando apenas as melhores configurações, quando executadas em 2, 4, 8 e 16 nós. Verifica-se que o erro sofre um pequeno aumento a medida que novos nós vão sendo incluídos, fruto do aumento do erro da estimativa do tempo de comunicação.

De forma semelhante ao que se verificou no *kernel* CG, para este *kernel* as melhores configurações para 2 e 4 nós foram as que empregaram o processador Intel combinados com a rede Ethernet. Como o custo de comunicação aumenta rapidamente para este *kernel*, a melhor configuração foi obtida com 8 processadores AMD interconectados pela rede Infiniband, com um tempo total estimado de 25,1 segundos.

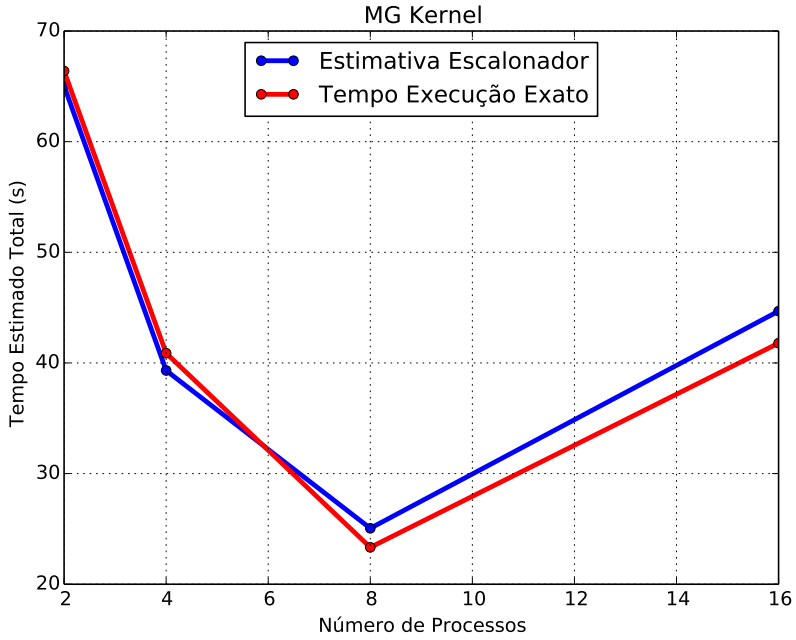

Figura 5.4: *Kernel* MG - Estimativa do escalonador comparado com o tempo real para vários nós.

5.4.5 EP

Por tratar-se de um *kernel* embaraçosamente paralelo, o tempo de comunicação de EP não interfere de forma expressiva no tempo total de execução, portanto o tempo total tem a tendência de diminuir linearmente a medida que novos processos vão sendo incorporados na execução deste *kernel*. Este comportamento foi capturado pelo escalonador (figura 5.5), e como esperado a melhor configuração foi para 16 processadores (8 AMD e 8 Intel) interligados pela rede Ethernet com um tempo de 28,6 segundos.

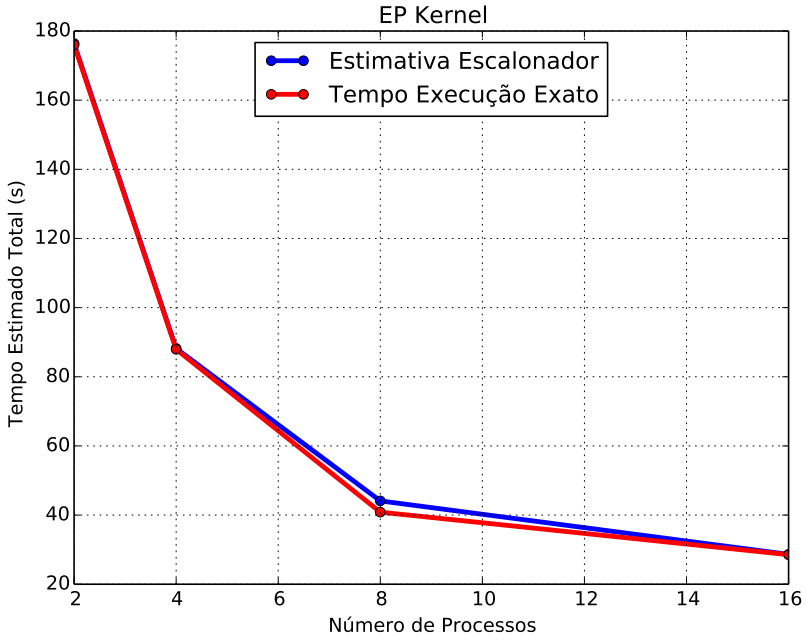

Figura 5.5: *Kernel* EP - Estimativa do escalonador comparado com o tempo real para vários nós.

5.4.6 LU

A figura 5.6 mostra os resultados com os tempos de execução estimados para a execução de LU usando as melhores configurações obtidas pelo escalonador. Verifica-se que os valores são precisos para os melhores casos das configurações iniciais. Somente para 16 nós que o erro sofre um aumento. Como o tempo de comunicação não apresenta uma grande influência no tempo total estimado, para as configurações iniciais de 2, 4 e 8 nós, o escalonador escolhe como melhores configurações as que utilizam processadores Intel interligados pela rede Ethernet. A configuração que possui a estimativa de ter o menor tempo de execução é a que utiliza 8 nós Intel, interligados pelo controlador Ethernet. Estima-se que nesta configuração a aplicação execute em aproximadamente 57 segundos.

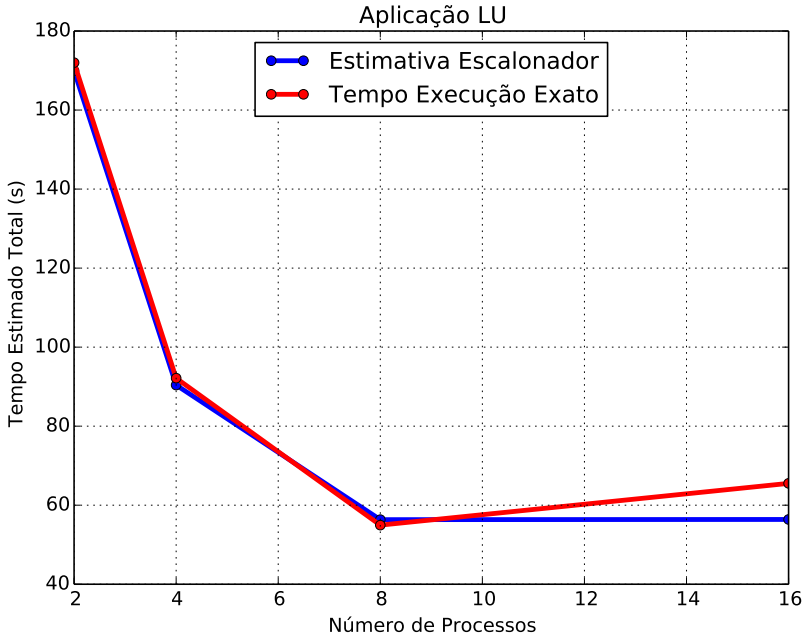

Figura 5.6: Aplicação LU - Estimativa do escalonador comparado com o tempo real para vários nós.

5.4.7 BT

De forma contrária ao que acontece com a aplicação LU, a figura 5.7 mostra melhores aproximações, por parte do escalonador, para as configurações que empregam uma quantidade maior de nós na execução da aplicação BT. A configuração que o escalonador estima que obterá o menor tempo de execução é a que utiliza 8 processadores Intel e 1 processador AMD, ambos interligados pela rede Ethernet, com um tempo de 231,5 segundos. Deve-se destacar que o valor obtido na tabela 5.30 para 9 nós é referente à configuração utilizando 5 CPUs Intel e 4 CPUs AMD, o que explica a diferença entre os tempos da tabela e do gráfico.

Para esta aplicação, onde o tempo de computação possui a maior parte da influência no tempo total, o escalonador executou poucos passos, dado que para 9 nós o escalonador deve comparar se a configuração de 9 processadores AMD interligados pela rede Infiniband é mais rápida do que utilizar 8 processadores Intel e 1 AMD interligados pela rede Ethernet. E para 16 nós somente uma configuração é possível, utilizar todos os 8 processadores Intel disponíveis (considerados rápidos) e posteriormente incorporando os processadores AMD (considerados lentos) até que o número de nós seja igual ao exigido pela aplicação. Relembrando que o critério de parada foi alterado para que o gráfico com todas as

configurações do *cluster* fosse gerado; o escalonador em sua versão original retornaria somente a melhor configuração encontrada.

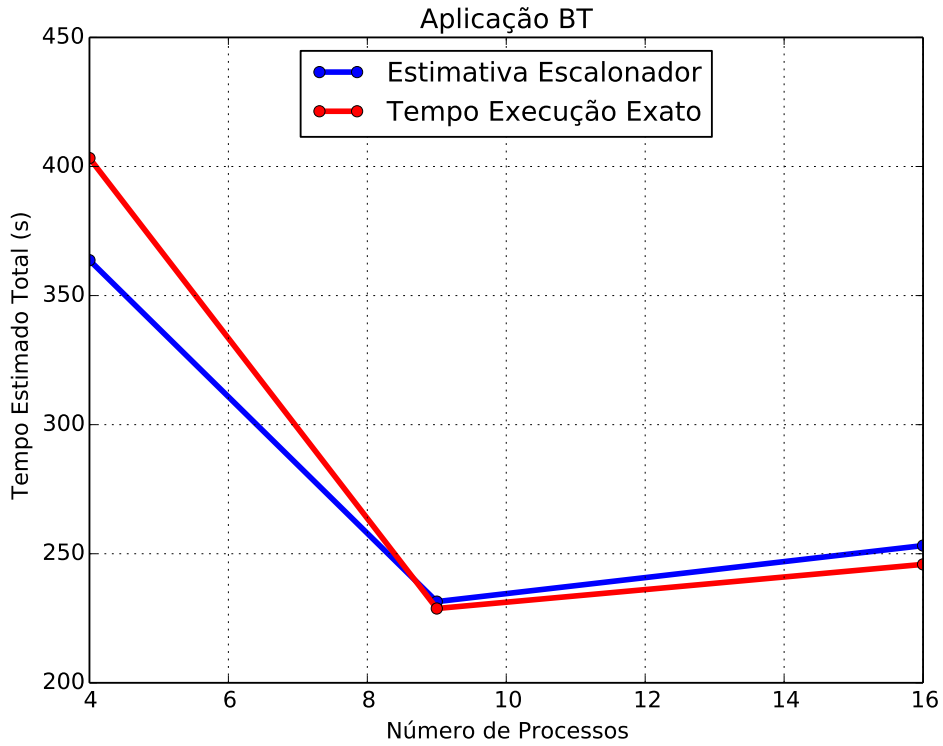

Figura 5.7: Aplicação BT - Estimativa do escalonador comparado com o tempo real para vários nós.

5.4.8 SP

A figura 5.8 apresenta as estimativas do tempo de execução de SP nas melhores configurações encontradas pelo escalonador. Novamente são apresentados, para cada número de nós, as estimativas para o tempo total de execução usando a melhor combinação de processador e rede. Quando essa figura é comparada com as das outras aplicações, verifica-se um erro mais elevado, sendo esse causado pelo erro na estimativa do tempo de computação. No entanto o modelo conseguiu capturar o comportamento da aplicação para a execução paralela. A melhor configuração obtida pelo escalonador foi utilizando 8 CPUs Intel e 1 CPU AMD interligadas pela rede Ethernet, com estimativa de executar a aplicação com um tempo de 225,4 segundos.

Da mesma forma que aconteceu com a aplicação BT, o escalonador realizou poucos passos para encontrar a melhor configuração. Novamente deve-se destacar que a diferença

observada entre o valor estimado no tempo de execução apresentado na tabela 5.34 para 9 nós e o valor para a mesma configuração apresentado na figura 5.8 é referente ao uso de uma configuração distinta para realizar as estimativas, já que na tabela estimou-se o tempo de execução com o uso de 5 CPUs Intel e 4 CPUs AMD.

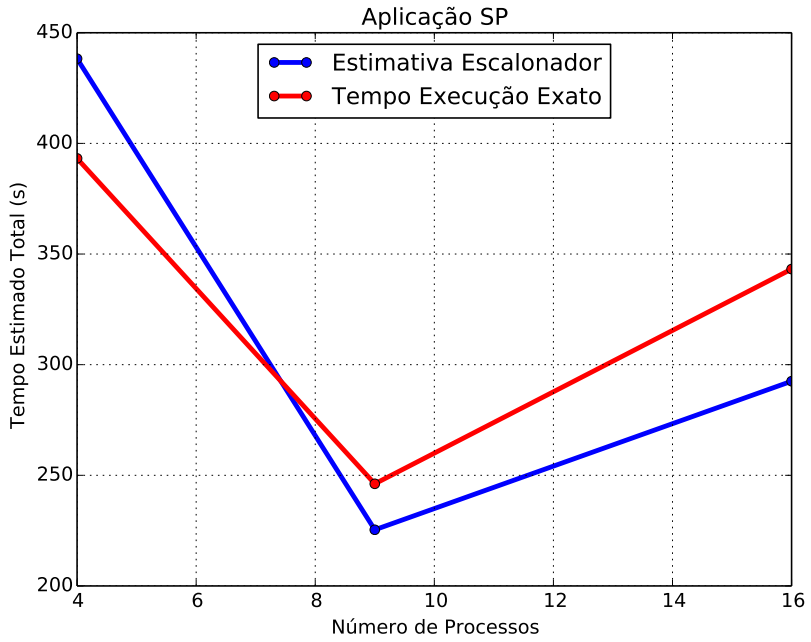

Figura 5.8: Aplicação SP - Estimativa do escalonador comparado com o tempo real para vários nós.

5.4.9 HIS GPU

A figura 5.9 apresenta as estimativas para o tempo de execução de HIS em GPUs, encontradas pelo escalonador para as melhores configurações de hardware empregando 4, 5, 6, 7 e 8 nós. Para este caso em que somente GPUs são utilizadas o resultado é trivial, considerando que as melhores unidades de processamento estão interligadas pelo controlador de rede mais rápido. Deve-se ainda destacar que os valores apresentados são diferentes dos encontrados na tabela 5.35 porque os ambientes usados nas estimativas são diferentes. A tabela 5.35 apresenta estimativas do tempo total de execução em um ambiente heterogêneo pré-definido, enquanto que o escalonador estima o tempo na melhor configuração encontrada para a execução da aplicação, usando para isso o modelo HCLogP.

O gráfico da figura 5.9 mostra que as estimativas feitas pelo escalonador para o tempo total de execução, usando por base o modelo HCLogP, foram inferiores aos tempos reais. É possivel verificar que a estimativa feita para o menor número de processos, quatro, teve maior erro do que as estimativas feitas para as configurações com maior número de processos. A melhor configuração encontrada pelo escalonador reduz significativamente o melhor tempo estimado de execução da aplicação paralela encontrado na tabela 5.35: de 810,5 segundos utilizando 4 nós (2 M2075 e 2 C1060), reduz para 153,3 segundos utilizando 1 M2090 e 4 M2075 interligadas pela rede Infiniband.

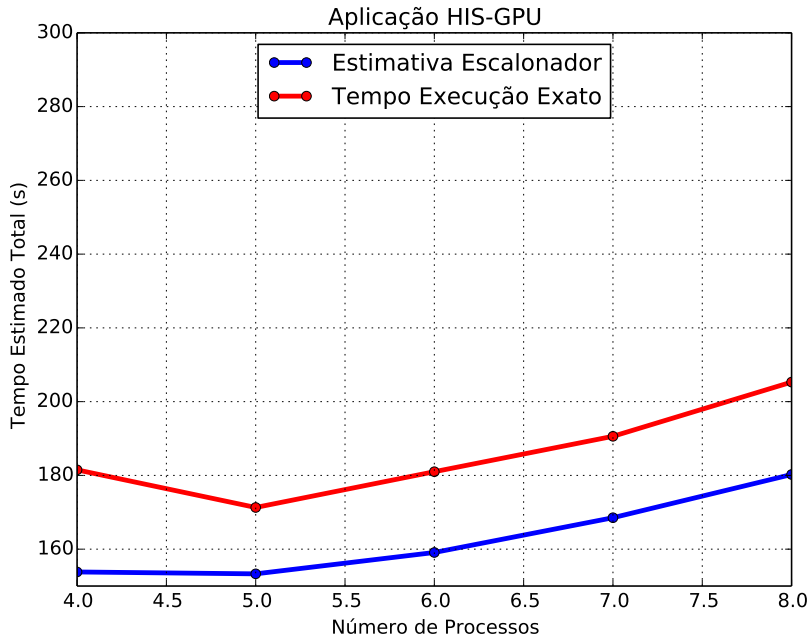

Figura 5.9: Aplicação HIS GPU - Estimativa do escalonador comparado com o tempo real para vários nós.

5.4.10 HIS CPU-GPU

De maneira análoga a aplicação HIS GPU, os valores estimados pelo escalonador para a execução de HIS, quando executada simultaneamente em GPUs e CPUs, são diferentes dos encontrados na tabela 5.36, que apresenta o resultado do modelo para algumas configurações pré-definidas, e que não necessariamente são as que reduzem mais o tempo de execução. Observa-se que boas aproximações foram obtidas para as duas configurações iniciais; a partir da configuração com 6 processos o erro aumenta. Isto ocorre devido à redução no tempo de computação ser mínima com a inclusão de novos dispositivos a partir

de 5 nós (uma redução de 0,5 segundos), ou seja, a quantidade de dados é insuficiente para ocupar todas as unidades de processamento, comportamento esse não capturado pelo modelo. A melhor configuração obtida pelo escalonador foi a que utilizou 6 CPUs AMD (1 GPU M2090 e 5 M2075) interligadas pela rede Infiniband, com um tempo de 14,6 segundos, muito inferior ao melhor tempo da tabela 5.36 (51,2 segundos).

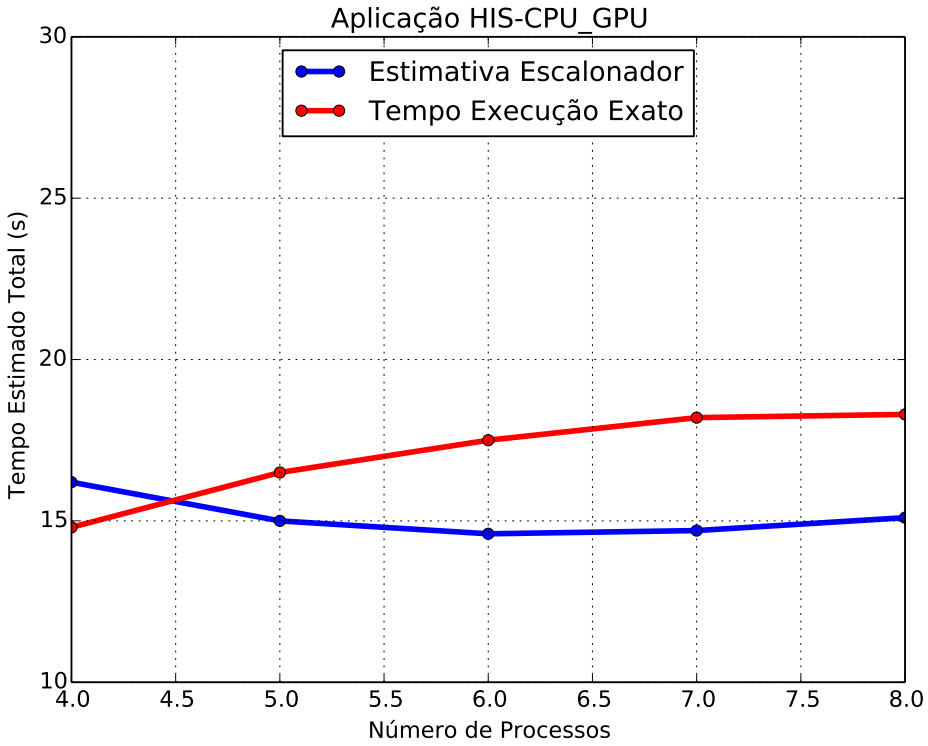

Figura 5.10: Aplicação HIS CPU-GPU - Estimativa do escalonador comparado com o tempo real para vários nós.

5.5 Resumo do capítulo

Este capítulo apresentou os resultados de uma avaliação experimental do modelo HCLogP e do escalonador que o utiliza para encontrar a configuração de hardware que minimiza o tempo de execução paralelo de uma aplicação regular. Para esta avaliação, foi utilizado o *benchmark* NAS e a aplicação HIS, que simula o funcionamento do sistem imune. Os resultados mostram que o modelo obteve bons resultados, com erros abaixo de 19,2% na estimativa do tempo total de execução de todas as aplicações regulares em um ambiente heterogêneo. O modelo também conseguiu capturar o comportamento característico de

cada aplicação para diferentes configurações de ambientes de execução. Além disso, uma boa acurácia foi observada nas estimativas dos componentes (tempo de comunicação e de computação) do tempo total de execução.

O escalonador encontrou a melhor configuração de execução para todas as aplicações regulares avaliadas, mesmo para aquelas (LU e HIS-GPU) em que o erro proveniente do modelo poderia afetar seus resultados por conta da pequena diferença no tempo total de execução entre algumas configurações.

Como o *kernel* EP executa somente uma iteração, a fórmula do tempo de computação apresentada no capítulo 3 foi alterada para:

$$T_{\text{computação}} = \frac{\text{size}}{R_p}, \tag{5.16}$$

onde:

- $size$ é o tamanho do problema;
- R_p foi substituído por uma métrica que representa o processamento de dados por passo de tempo de uma unidade computacional.

Neste trabalho o modelo foi avaliado utilizando uma aplicação que realiza balanceamento de carga dinâmico (HIS-GPU-CPU), e que por isso possui características de uma aplicação irregular. O modelo foi concebido para lidar apenas com aplicações regulares. Apesar do baixo erro na estimativa do tempo total de execução, o escalonador não conseguiu identificar a melhor configuração de execução para esta aplicação.

6 Conclusão

Neste trabalho foi avaliado um novo modelo que prediz o tempo de execução de aplicações paralelas regulares em ambientes heterogêneos. Os resultados mostraram que o modelo pode prever o tempo total de aplicações com diferentes características, sendo executadas por dispositivos diferentes e interconectadas por controladores de rede distintos. Os erros encontrados para o tempo de computação ficaram abaixo de 22,7% em todos os experimentos. Para a comunicação o maior erro foi de 17,3%, e para o tempo total de execução 19,2%. Em alguns casos, o valor estimado foi próximo ou igual ao tempo de execução real. Além disso, o modelo conseguiu capturar o comportamento de aplicações com distintas características para diferentes configurações de ambientes. Os experimentos foram realizados em um pequeno *cluster* heterogêneo, composto por dois tipos de CPU, três tipos de GPUs e dois tipos de redes.

O modelo apresenta para o cálculo do tempo de computação um parâmetro que representa um fator de correção quando não se espera um *speedup* linear na fase de computação. Isto pode ocorrer, por exemplo, devido à redução dos custos de acesso à memória pelo incremento do tamanho da *cache*, decorrente de mais processadores serem agregados ao processamento. Valores positivos e negativos ou até mesmo o resultado de um função podem ser utilizados para impor um fator de *speedup* diferente do linear. Como trabalho futuro, pretende-se realizar experimentos para verificar se a acurácia do modelo aumenta quando valores diferentes de zero são usados para este parâmetro. Outra alternativa seria realizar uma análise de sensibilidade do modelo para este parâmetro. Os resultados dos testes podem, eventualmente, levar a eliminação do fator de correção caso o melhor valor nas estimativas seja sempre obtido com este parâmetro igual a zero.

Este modelo foi utilizado como base para o desenvolvimento de um escalonador que, apresentadas as características do ambiente e da aplicação, encontra a melhor configuração de nós que minimiza o tempo de execução total da aplicação. O algoritmo do escalonador é baseado no algoritmo aproximativo *Branch and Bound* que emprega um método enumerativo para encontrar a solução. Para todas as aplicações o escalonador encontrou a configuração que apresenta o menor tempo de execução total. A única exceção foi a aplicação HIS, quando executada simultâneamente em CPUs e GPUs. Como esta

aplicação tem características de uma aplicação irregular, o escalonador não conseguiu encontrar a configuração que minimiza seu tempo de execução. O escalonador também se adaptou bem à medida que novos nós são incorporados na execução da aplicação, considerando as influências do tempo de computação e comunicação no tempo total.

Como trabalho futuro, o principal objetivo é avaliar o modelo em situações em que todos os recursos de um nó computacional sejam usados. Nos experimentos realizados, um único processo por nó foi considerado, enquanto que cada nó possuía um número maior de núcleos disponíveis. Desta forma vai ser possível verificar se o modelo será capaz de prever o tempo de execução de aplicações paralelas que se beneficiam do máximo poder computacional que um ambiente pode oferecer. Outro fator que deve ser considerado no modelo é a comunicação que pode ocorrer entre os distintos controladores de redes: Todos os experimentos foram conduzidos usando ou a rede Ethernet, ou a rede Infiniband, mas nenhum usando as duas redes em conjunto. Identificar e aprimorar o modelo para que os erros sejam reduzidos, bem como avaliá-lo com mais aplicações, também são pontos importantes para a evolução deste trabalho.

No escalonador seria interessante, como trabalho futuro, considerar uma heurística, algoritmos genéticos, por exemplo, que avalie a relação custo $\times$ benefício de se incluir uma nova unidade computacional. Isto evitaria que fosse acrescida à solução uma nova unidade computacional que trouxesse ganhos inexpressivos no tempo total de execução. Para melhor ilustrar a situação, imagine o seguinte cenário. A execução de uma aplicação com 3 CPUs foi de 40 segundos, divididos entre 20 segundos de computação e 20 segundos de comunicação. O escalonador testa então a inclusão de mais uma CPU. O tempo de execução passaria, com esta inclusão, a ser de 39 segundos, divididos entre 17 segundos de computação e 22 segundos de comunicação. Como o ganho foi de apenas 1 segundo, não compensaria incluir este recurso, que poderia estar livre para ser melhor utilizado por outra aplicação.

Outros aspectos que podem ser objeto de trabalhos futuros são: a) comparar os resultados obtidos pelo escalonador com os obtidos por outras técnicas, como por exemplo as que usem uma heurística gulosa, b) avaliá-lo em ambientes maiores, e c) integrar o *overhead* na predição do escalonador.

REFERÊNCIAS

[1] CARIÑO, R. L., BANICESCU, I., "Dynamic load balancing with adaptive factoring methods in scientific applications", *The Journal of Supercomputing*, v. 44, n. 1, pp. 41–63, 2008.

[2] CULLER, D., KARP, R., PATTERSON, D., SAHAY, A., SCHAUSER, K. E., SANTOS, E., SUBRAMONIAN, R., VON EICKEN, T., "LogP: Towards a Realistic Model of Parallel Computation", *SIGPLAN Not.*, v. 28, n. 7, pp. 1–12, July 1993.

[3] BAILEY, D. H., BARSZCZ, E., BARTON, J. T., BROWNING, D. S., CARTER, R. L., DAGUM, L., FATOOHI, R. A., FREDERICKSON, P. O., LASINSKI, T. A., SCHREIBER, R. S., OTHERS, "The NAS parallel benchmarks", *International Journal of High Performance Computing Applications*, v. 5, n. 3, pp. 63–73, 1991.

[4] SOARES, T. M., XAVIER, M. P., PIGOZZO, A. B., CAMPOS, R. S., DOS SANTOS, R. W., LOBOSCO, M., "Performance Evaluation of a Human Immune System Simulator on a GPU Cluster", In: *Parallel Computing Technologies*, pp. 458–468, Springer, 2015.

[5] ALEXANDROV, A., IONESCU, M. F., SCHAUSER, K. E., SCHEIMAN, C., "LogGP: incorporating long messages into the LogP modelâone step closer towards a realistic model for parallel computation". In: *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pp. 95–105, 1995.

[6] KIELMANN, T., BAL, H. E., VERSTOEP, K., "Fast measurement of LogP parameters for message passing platforms", In: *Parallel and Distributed Processing*, pp. 1176–1183, Springer, 2000.

[7] LASTOVETSKY, A., MKWAWA, I.-H., O'FLYNN, M., "An accurate communication model of a heterogeneous cluster based on a switch-enabled ethernet network". In: *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, v. 2, pp. 6–pp, 2006.

[8] LASTOVETSKY, A., RYCHKOV, V., "Building the communication performance model of heterogeneous clusters based on a switched network". In: *Cluster Computing, 2007 IEEE International Conference on*, pp. 568–575, 2007.

[9] BOSQUE, J. L., PASTOR, L., "A parallel computational model for heterogeneous clusters", *IEEE Transactions on Parallel and Distributed Systems*, v. 17, n. 12, pp. 1390, 2006.

[10] VALIANT, L. G., "A bridging model for parallel computation", *Communications of the ACM*, v. 33, n. 8, pp. 103–111, 1990.

[11] WILLIAMS, T. L., PARSONS, R. J., "The heterogeneous bulk synchronous parallel model". In: *International Parallel and Distributed Processing Symposium*, pp. 102–108, 2000.

[12] HOCKNEY, R. W., "The communication challenge for MPP: Intel Paragon and Meiko CS-2", *Parallel Computing*, v. 20, n. 3, pp. 389–398, 1994.

[13] PJEŠIVAC-GRBOVIĆ, J., ANGSKUN, T., BOSILCA, G., FAGG, G. E., GABRIEL, E., DONGARRA, J. J., "Performance analysis of MPI collective operations", *Cluster Computing*, v. 10, n. 2, pp. 127–143, 2007.

[14] HOLT, C., HEINRICH, M., SINGH, J. P., ROTHBERG, E., HENNESSY, J., *The effects of latency, occupancy, and bandwidth in distributed shared memory multiprocessors*. Computer Systems Laboratory, Stanford University, 1995.

[15] MARTIN, R. P., VAHDAT, A. M., CULLER, D. E., ANDERSON, T. E., *Effects of communication latency, overhead, and bandwidth in a cluster architecture*. v. 25. ACM, 1997.

[16] DE AMORIM MENDES, H., "HLogP: Um Modelo de Escalonamento para a Execução de Aplicações MPI em Grades Computacionais", *Master's Thesis, Universidade Federal Fluminense*, 2004.

[17] RICO-GALLEGO, J.-A., DÍAZ-MARTÍN, J.-C., "τ-Lop: Modeling performance of shared memory MPI", *Parallel Computing*, v. 46, pp. 14–31, 2015.

[18] CAMERON, K. W., GE, R., SUN, X.-H., " $log_m P$ and $log_3 P$: Accurate Analytical Models of Point-to-Point Communication in Distributed Systems", *IEEE Transactions on Computers*, v. 56, n. 3, 2007.

[19] TU, B., FAN, J., ZHAN, J., ZHAO, X., "Performance analysis and optimization of MPI collective operations on multi-core clusters", *The Journal of Supercomputing*, v. 60, n. 1, pp. 141–162, 2012.

[20] RICO-GALLEGO, J.-A., DÃAZ-MARTÃN, J.-C., LASTOVETSKY, A. L., "Extending τ-Lop to model concurrent MPI communications in multicore clusters", *Future Generation Computer Systems*, v. 61, pp. 66 – 82, 2016.

[21] NUDD, G. R., KERBYSON, D. J., PAPAEFSTATHIOU, E., PERRY, S. C., HARPER, J. S., WILCOX, D. V., "PACE-A toolset for the performance prediction of parallel and distributed systems", *The International Journal of High Performance Computing Applications*, v. 14, n. 3, pp. 228–251, 2000.

[22] MARIN, G., MELLOR-CRUMMEY, J., "Cross-architecture performance predictions for scientific applications using parameterized models". In: *ACM SIGMETRICS Performance Evaluation Review*, v. 32, n. 1, pp. 2–13, 2004.

[23] TSAFRIR, D., ETSION, Y., FEITELSON, D. G., "Backfilling Using System-Generated Predictions Rather Than User Runtime Estimates", *IEEE Trans. Parallel Distrib. Syst.*, v. 18, n. 6, pp. 789–803, June 2007.

[24] SMITH, W., "Prediction services for distributed computing". In: *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–10, 2007.

[25] DOERFLER, D., BRIGHTWELL, R., "Measuring MPI send and receive overhead and application availability in high performance network interfaces". In: *European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*, pp. 331–338, 2006.

[26] SOARES, T. M., DOS SANTOS, R. W., LOBOSCO, M., "A Parallel Model for Heterogeneous Cluster". In: *Algorithms and Architectures for Parallel Processing - ICA3PP 2016 Collocated Workshops: SCDT, TAPEMS, BigTrust,*

UCER, DLMCS, Granada, Spain, December 14-16, 2016, Proceedings, pp. 76–90, 2016.

[27] CARDELLINI, V., FANFARILLO, A., FILIPPONE, S., "Overlapping communication with computation in MPI applications", *Universita di Roma Tor Vergata, Tech. Rep. DICII RR-16.09*, 2016.

[28] PATTERSON, D. A., HENNESSY, J. L., *Computer Organization and Design, Fourth Edition, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. 4th ed. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2008.

[29] NEUMANN, F., WITT, C., *Combinatorial Optimization and Computational Complexity*. Springer Berlin Heidelberg: Berlin, Heidelberg, 2010.

[30] MARTELLO, S., TOTH, P., "Knapsack problems: Algorithms and computer interpretations", *Hoboken, NJ: Wiley-Interscience*, 1990.

[31] LEISERSON, C., CORMEN, T., RIVEST, R., STEIN, C., *Algoritmos: teoria e prática*. CAMPUS - RJ, 2002.

[32] BRASSARD, G., BRATLEY, P., *Fundamentals of Algorithmics*. Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1996.

[33] MURRAY, D., KOZINIEC, T., LEE, K., DIXON, M., "Large MTUs and internet performance". In: *High Performance Switching and Routing (HPSR), 2012 IEEE 13th International Conference on*, pp. 82–87, 2012.

[34] LEVEQUE, R., *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems (Classics in Applied Mathematics Classics in Applied Mathemat)*. Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2007.

[35] PIGOZZO, A. B., MACEDO, G. C., SANTOS, R. W., LOBOSCO, M., "On the computational modeling of the innate immune system", *BMC Bioinformatics*, v. 14 Suppl 6, pp. S7, 2013.

[36] ROCHA, P. A. F., XAVIER, M. P., PIGOZZO, A. B., QUINTELA, B. D. M., MACEDO, G. C., DOS SANTOS, R. W., LOBOSCO, M., "A three-dimensional computational model of the innate immune system". In: *International Conference on Computational Science and Its Applications*, pp. 691–706, 2012.