UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MANUELA KLANOVICZ FERREIRA

Mapeamento Estático de Processos MPI com Emparelhamento Perfeito de Custo Máximo em Cluster Homogêneo de *Multi-cores*

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Philippe Olivier Alexandre Navaux
Orientador

Porto Alegre, julho de 2012.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Ferreira, Manuela Klanovicz

Mapeamento Estático de Processos MPI com Emparelhamento Perfeito de Custo Máximo em Clusters Homogêneos de *Multi-cores* / Manuela Klanovicz Ferreira. – 2012.

72 f.:il.

Orientador: Phillippe O. A. Navaux

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR, 2012.

1.Mapeamento de Processos. 2.MPI 3.Multi-core 4.Comunicação entre Processos I. Philippe Olivier Alexandre Navaux II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Vladimir Pinheiro do Nascimento
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do PPGC: Prof. Álvaro Freitas Moreira
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

"Liberdade, essa palavra
que o sonho humano alimenta
que não há ninguém que explique
e ninguém que não entenda."

ROMANCEIRO DA INCONFIDÊNCIA – CECÍLIA MEIRELES

AGRADECIMENTOS

Aos meus familiares, agradeço a compreensão pelas diversas vezes em que não pude estar presente durante este período dedicado ao mestrado. Em especial à minha Mãe, Matilde Klanovicz, ao meu irmão, Rodrigo Klanovicz Ferreira, e ao meu amor, Mauricio Machado da Silva, que além de compreensivos também sempre estiveram dispostos a prestar todo o tipo de suporte.

Aos meus companheiros de laboratório e do Grupo de Processamento Paralelo e Distribuído (GPPD): Marco Alves, Felipe Madruga, Eduardo Cruz, Laércio Pilla, Francieli Boito, Eduardo Rocha e Márcia Cera. Por sempre estarem presentes para esclarecer dúvidas, dar conselhos e também servirem de exemplo. Espero conseguir um dia retribuir de alguma forma toda a ajuda recebida. Em especial aos colegas Henrique Freitas, Rodrigo Kassick e Vicente Cruz pelas essenciais revisões feitas ao trabalho.

Aos professores do Programa de Pós-Graduação em Computação da UFRGS e, em especial, ao meu orientador o Prof. Dr. Philippe Olivier Alexandre Navaux por me darem a oportunidade de desfrutar de um dos cursos de pós-graduação mais bem conceituados do país.

A todos os funcionários da UFRGS e colegas de trabalho pela oportunidade de conquistar o meu sonho de seguir estudando.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CMP	*Chip* Multiprocessado
BE	Busca Exaustiva
BT	*Block Tridiagonal solver*
BRD	Biparticionamento Recursivo Dual
CG	*Conjugate Gradient*
DMA	*Direct Memory Access*
EP	*Embarrassingly Parallel*
EPCM	Emparelhamento Perfeito de Custo Máximo
FSB	*Front Side Bus*
FT	*Fast Fourier Transform*
GPP	*General Purpose Processor*
IS	*Integer Sort*
L1	Memória *Cache* Nível 1
L2	Memória *Cache* Nível 2
L3	Memória *Cache* Nível 3
LU	*Lower-Upper solver*
MapEME	Mapeamento Estático MPI com Emparelhamento
MG	MultiGrid
MPE	*MPI Parallel Environment*
MPI	*Massage Passing Interface*
NIC	*Network Interface Card*
NAS	*Numerical Aerodynamics Simulation*
NASA	*National Aeronautics and Space Administration*
NPB	*NAS Parallel Benchmarks*
NP	*Non-deterministic Polynomial time*
RAM	*Random Access Memory*
RDMA	*Remote Direct Memory Access*

SMP	*Symmetric Multi-Processors*
SMT	*Simultâneous Multithreading*
SP	*Scalar Pentadiagonal solver*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Um importante fator que precisa ser considerado para alcançar alto desempenho em aplicações paralelas é a distribuição dos processos nos núcleos do sistema, denominada mapeamento de processos. Mesmo o mapeamento estático de processos é um problema NP-difícil. Por esse motivo, são utilizadas heurísticas que dependem da aplicação e do hardware no qual a aplicação será mapeada. Nas arquiteturas atuais, além da possibilidade de haver mais de um processador por nó do cluster, é possível haver mais de um núcleo de processamento por processador, assim, o mapeamento estático de processos pode considerar pelo menos três níveis de comunicação entre os processos que executam em um cluster *multi-core*: intra-chip, intra-nó e inter-nó. Este trabalho propõe a heurística MapEME (Mapeamento Estático MPI com Emparelhamento) que emprega o Emparelhamento Perfeito de Custo Máximo (EPCM) no cálculo do mapeamento estático de processos paralelos MPI em processadores *multi-core.* Os resultados alcançados pelo mapeamento gerado pela MapEME são comparados aos resultados obtidos pelo mapeamento gerado pela aplicação Scotch, que utiliza o Biparticionamento Recursivo Dual (BRD), já utilizado como heurística para mapeamento estático de processos. Ambas as heurísticas são comparadas à Busca Exaustiva (BE) para verificar o quanto estão próximas do ótimo. Os três métodos têm a complexidade e o ganho no tempo de execução em ralação à distribuição padrão da biblioteca MPICH2 comparados entre si. A principal contribuição deste trabalho é mostrar que a heurística EPCM apresenta ganho de até 40% equivalente a já difundida BRD, e possui uma complexidade menor ao ser aplicado em um cluster *multi-core* que compartilha *cache* nível 2 a cada dois núcleos.

Palavras-Chave: Mapeamento de Processos, MPI, *Multi-core*, Comunicação entre Processos

Static MPI Processes Mapping using Maximum Weighted Perfect Matching at Homogeneous Multi-core Clusters

ABSTRACT

An important factor that must be considered to achieve high performance on parallel applications is the mapping of processes on cores. However, since this is defined as an NP-Hard problem, it requires different mapping heuristics that depends on the application and the hardware on which it will be mapped. On the current architectures we can have more than one multi-core processors per node, and consequently the process mapping can consider three process communication types: intrachip, intranode and internode. This work propose the MapEME (Static Mapping MPI using Matching) that use the Maximum Weighted Perfect Matching (MWPM) to calculate the static process mapping and analyze its performance. The results provided by MapEME are compared with the results of application Scotch. It uses Dual Recursive Bipartitioning (DRB), an already used heuristics for static mapping. Both heuristics are compared with Exhaustive Search (ES) to verify how much the two heuristics are near the optimum. The three methods have theirs complexities analyzed. Also the mapping gain when compared with the standard MPICH2 distribution was measured. The main contribution of this work is to show that the heuristic, EPCM, provides gain up to 40%, close of DRB gain. Furthermore, EPCM has a lower complexity when applied to a multicore cluster that shares L2 cache every two cores.

Keywords: Process Mapping, MPI, Multicore, Processes' Communication, Maximum Weighted Perfect Matching

1 INTRODUÇÃO

O uso de *Chips* Multiprocessados (CMPs), também conhecidos como processadores *multi-core*, está sendo consolidado não somente em servidores de grande porte como também em computadores pessoais. Processadores com oito núcleos estão se tornando comuns e as previsões para os próximos anos são de que o número de núcleos por *chip* vai aumentar drasticamente (ASANOVIC et al., 2006), conforme a Lei de Moore continue valendo nos CMPs (KUMAR et al., 2005). Nesse novo paradigma de construção de máquinas, os núcleos no mesmo *chip* podem compartilhar recursos como, por exemplo, um nível da hierarquia de memória.

A utilização consciente dos recursos compartilhados possibilita o alcance de um melhor desempenho. Melhorias no tempo de comunicação utilizando memória compartilhada, por exemplo, podem resultar em melhorias significativas no desempenho geral da aplicação.

1.1 Problema

Em uma aplicação paralela, o perfil de comunicação entre seus processos nem sempre é uniforme. Nesse caso alguns processos se comunicam mais com uns do que com outros. Mesmo quando essas aplicações são executadas em clusters homogêneos, onde todas as unidades de processamento são iguais, pode haver heterogeneidade na velocidade de comunicação entre as unidades de processamento. Isso porque, mesmo que a rede de interconexão entre os nós seja homogênea, atualmente é comum a presença de mais de um processador por nó e, com o advento dos *multi-cores*, mais de um núcleo de processamento por processador. Neste caso, a velocidade de comunicação entre núcleos de processamento localizados no mesmo processador é maior do que a velocidade de comunicação entre núcleos de processamento que se encontram em nós diferentes do cluster.

Posicionar os processos da aplicação paralela que se comunicam mais em núcleos de processamento com maior velocidade de comunicação entre si tende a contribuir para a diminuição do tempo de comunicação e, consequentemente do tempo de execução das aplicações paralelas.

1.2 Motivação

Atualmente existem diversas bibliotecas que possibilitam a implementação de aplicações paralelas. Para máquinas com memória compartilhada entre todos os núcleos de processamento pode-se utilizar OpenMP (CHANDRA et al., 2001) com comunicação implícita através de variáveis compartilhadas. Outro paradigma de programação paralela é o MPI (*Message Passing Interface*), onde a comunicação é explícita baseada em troca de mensagens, permitindo sua utilização em sistemas com

memória distribuída. Como implementações bastante difundidas de MPI pode-se citar MPICH2 (MPICH2, 2012) e a OPENMPI (OPENMPI, 2011).

Apesar dessa usual classificação baseada em paradigmas, já existem implementações de OpenMP que podem ser utilizadas em um ambiente distribuído, ou seja, sem memória compartilhada, como o *Intel Cluster OpenMP* (INTEL CORPORATION, 2005), mas que não alcança desempenho melhor que o MPI. Em contrapartida, também há implementações MPI, como a MPICH2 que, desde a versão 1.3 (MPICH2 1.3, 2010), utiliza por padrão o método de comunicação chamado *nemesis*. Esse método de comunicação combina a já utilizada troca de mensagens por *socket* com a troca de mensagens por memória compartilhada, levando em consideração se os processos estão compartilhando memória para decidir qual comunicação usar (BUNTINA; MERCIER; GROPP, 2007).

Neste contexto, a programação paralela sozinha não garante o melhor aproveitamento dos recursos. Determinar o melhor local onde cada processo do programa paralelo será executado também é importante. Por padrão os processos são distribuídos automaticamente pelo escalonador de processos da biblioteca utilizada para fazer a paralelização do código fonte.

As bibliotecas de paralelização MPI, já citadas, executam por padrão uma distribuição *round-robin* dos processos paralelos pelos nós do cluster. A Figura 1.1 mostra um exemplo de distribuição de 10 processos em dois nós de um cluster, cada um com 4 unidades de processamento.

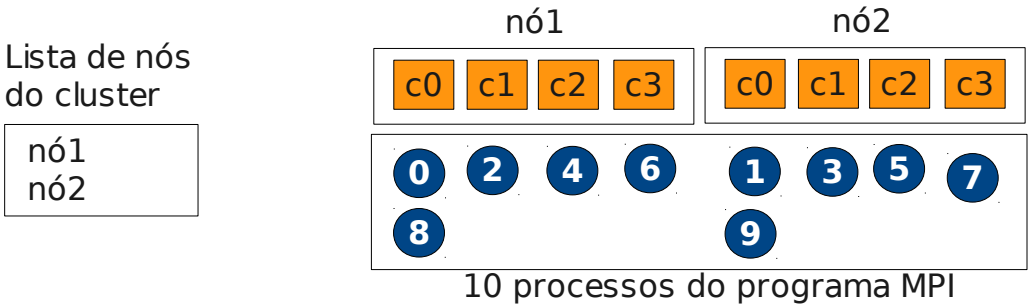


Figura 1.1: Distribuição *round-robin* (padrão) de 10 processos MPI em dois nós com quatro unidades de processamento cada. As unidades de processamento cujos respectivos sistemas operacionais identificaram como c0 receberam dois processos cada.

A distribuição *round-robin* não garante a melhor distribuição dos processos, mesmo no caso de *clusters* homogêneos onde todas as unidades de processamento são iguais, pois ainda pode haver uma heterogeneidade na velocidade de comunicação entre os núcleos do sistema, seja a comunicação através da memória compartilhada no caso de *threads*, ou por troca de mensagens no caso de processos paralelos MPI. A distribuição *round-robin* não garante, por exemplo, que processos que se comunicam mais fiquem mais "próximos", onde próximos significa estar em um local onde a comunicação entre eles seja mais rápida do que com os demais processos.

Ao distribuir conscientemente processos para as unidades de processamento tem-se uma instância do problema do mapeamento (BOKHARI, 1981) que é NP-difícil. Dessa forma, é necessário considerar características da aplicação e do hardware onde ela vai executar para encontrar uma solução específica em um tempo aceitável através de uma

heurística. Mesmo em um cluster homogêneo, a variedade de níveis de comunicação entre processos resulta em uma velocidade de comunicação heterogênea. Isso pode demandar novas heurísticas para a realização do mapeamento de processos visando diminuir o tempo de comunicação e, consequentemente, diminuir o tempo de execução da aplicação alvo.

Para o caso de um cluster *multi-core*, a heterogeneidade de comunicação aumenta, possuindo três níveis. Primeiro pode-se considerar a comunicação dentro do *chip*, entre núcleos que compartilham diferentes níveis de *cache* e, por isso, possuem diferentes tempos de comunicação. Segundo a comunicação entre *chips*, feita através da memória RAM compartilhada. E terceiro a comunicação entre nós diferentes do cluster através da rede de interconexão.

1.3 Objetivos e Contribuições

O objetivo principal deste trabalho é analisar o método Emparelhamento Perfeito de Custo Máximo (EPCM) como heurística para calcular o mapeamento estático de processos paralelos MPI. Para atingir o objetivo principal foram determinados os seguintes objetivos intermediários:

- Alterar o módulo MPE da biblioteca MPI a fim de criar matrizes com o padrão de comunicação entre os processos MPI das aplicações analisadas. Como resultado, para cada aplicação foram geradas duas matrizes que representam a quantidade total de bytes trocados entre os processos paralelos e a quantidade de mensagens trocadas, respectivamente;
- Implementar uma aplicação que utiliza o EPCM para definir um mapeamento para os processos paralelos em tempo polinomial, denominada MapEME (Mapeamento Estático MPI com Emparelhamento);
- Avaliar a MapEME, utilizando o conjunto de *benchmarks* NAS-NPB-MPI-3.3, comparando o ganho do EPCM com ganho obtido pelo mapeamento calculado por outra heurística chamada Biparticionamento Recursivo Dual (BRD), implementado pela aplicação Scotch 5.1 (PELLEGRINI et al., 2010), e também ao ganho obtido pelo mapeamento calculado através da Busca Exaustiva (BE), para ver o quanto próximo do ótimo a MapEME está;

As duas heurísticas comparadas, EPCM e BRD, fazem o cálculo do mapeamento baseadas na quantidade de comunicação entre os processo paralelos e na velocidade de comunicação entre os núcleos do cluster.

A principal contribuição deste trabalho é mostrar que a heurística EPCM apresenta ganho de até 40% equivalente a já difundida BRD, e possui uma complexidade menor ao ser aplicada em um cluster *multi-core* que compartilha *cache* nível 2 a cada dois núcleos. Também concluiu-se que para ambas as heurísticas o mapeamento calculado alcança ganhos próximos à BE. E, principalmente, os resultados mostram que a MapEME é o método em que o ganho intra-chip tem a maior contribuição no total de ganho. Isso se deve ao fato de que ela calcula primeiro o mapeamento dentro do *chip*.

1.4 Organização do Texto

No Capítulo 2, são esclarecidos alguns pontos sobre a comunicação em clusters *multi-core.* Também são descritos os níveis de comunicação e o funcionamento atual da comunicação *nêmesis* da biblioteca MPICH2.

No Capítulo 3, são apresentados os principais conceitos sobre mapeamento de processos paralelos, o funcionamento da heurística de BRD e da técnica de BE para o mapeamento de processos, bem como suas respectivas complexidades. Ao final são citados os trabalhos relacionados ao mapeamento de processos.

No Capítulo 4 é descrita a MapEME bem como sua análise de complexidade. Ao final são especificadas as modificações feitas na biblioteca MPICH2 para obter o padrão de comunicação das aplicações paralelas do NAS-NPB-3.3

O Capítulo 5 traz algumas descrições referentes aos experimentos, tais como a arquitetura onde eles foram executados, de que modo cada processo paralelo foi configurado para ser executado em um núcleo de processamento específico e a análise dos resultados obtidos. Os experimentos fazem uma comparação entre a heurística proposta MapEME com a heurística BRD e a técnica de BE. Também são comparados os resultados obtidos utilizando como entrada para os métodos o total de dados trocados entre os processos paralelos e o número de mensagens trocadas entre esses processos.

Por fim, o Capítulo 6 contém as conclusões referentes às comparações das técnicas de mapeamento utilizadas.

2 COMUNICAÇÃO EM ARQUITETURAS *MULTI-CORE*

É importante conhecer as possibilidades atuais de comunicação em um cluster *multi-core* e como as bibliotecas de paralelização estão utilizando esses níveis de comunicação. Assim, pode-se ilustrar porque um melhor posicionamento dos processos dentro do cluster pode diminuir o tempo gasto pela aplicação paralela na comunicação entre seus processos.

Neste capítulo são apresentados os níveis de comunicação presentes em um cluster *multi-core* para caracterizar a heterogeneidade na velocidade de comunicação em cada nível. Em seguida há uma discussão de como o tamanhos das mensagens, a hierarquia de memória e a contenção podem afetar o tempo de comunicação entre os processos paralelos. Também é descrito o funcionamento atual da comunicação *nemesis* da biblioteca MPICH2 que é capaz de detectar o compartilhamento de um nível na hierarquia de memória entre os processos e utilizar essa memória para a comunicação.

2.1 Níveis de Comunicação em Cluster *Multi-core*

Um cluster é definido como sendo um conjunto de computadores que trabalham interligados por uma rede com um objetivo comum. Nesse contexto, cada computador do cluster é chamado de nó. O objetivo para o cluster é a execução de uma aplicação paralela. Com apenas um processador, um mesmo nó do cluster não pode executar mais de uma tarefa ao mesmo tempo. Em computadores com mais de um processador, cada nó do cluster pode executar tarefas em paralelo e, atualmente, com o surgimento do processadores com múltiplos núcleos (*multi-core*), cada processador é capaz de executar sozinho mais de uma tarefa ao mesmo tempo. A Figura 2.1 faz uma comparação gráfica entre cluster, nó, processador e núcleo.

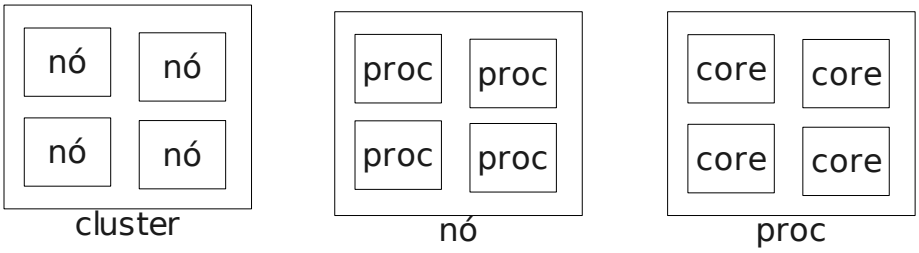


Figura 2.1: Comparação entre um cluster, um nó e um processador (proc) e núcleo (*core*).

É possível distinguir dois tipos de clusters: heterogêneos e homogêneos. O primeiro é caracterizado por possuir dois ou mais nós com diferentes configurações de hardware

ou software. No segundo, abordado neste trabalho, todos os nós possuem as mesmas configurações (LASTOVETSKY et al., 2008). Mas o fato de o cluster possuir nós homogêneos não quer dizer que a comunicação entre os núcleos do cluster ocorra com a mesma velocidade.

Uma vantagem de possuir mais de uma unidade de processamento (núcleo) por processador é que esses núcleos podem compartilhar um nível da hierarquia de memória e utilizar essa memória compartilhada para tornar mais rápida a comunicação entre as tarefas ou processos.

Em clusters *multi-core* é possível identificar uma hierarquia de comunicação de pelo menos três níveis: a comunicação entre diferentes nós de um cluster (inter-nó), a comunicação entre processadores de um mesmo nó (inter-chip) e a comunicação entre diferentes núcleos de um mesmo processador (intra-chip). Nas seções seguintes são caracterizados cada um desses níveis de comunicação presentes nos clusters *multi-core*.

2.1.1 Comunicação inter-nó

A comunicação inter-nó é aquela feita entre processos localizados em diferentes nós do cluster através da rede de interconexão. À exceção de alguns casos abordados na seção 2.2, ela possui a maior latência de comunicação. A topologia mais simples para um cluster é aquela ilustrada na Figura 2.2, onde existe apenas a comunicação inter-nó, ou seja, onde existe apenas um processador *single-core* por nó. Desta forma, a aplicação pode executar em paralelo apenas um processo por nó, onde todos os processos possuem a mesma latência de comunicação entre si, não sendo relevante em qual processador cada processo está executando para diminuir o tempo de comunicação.

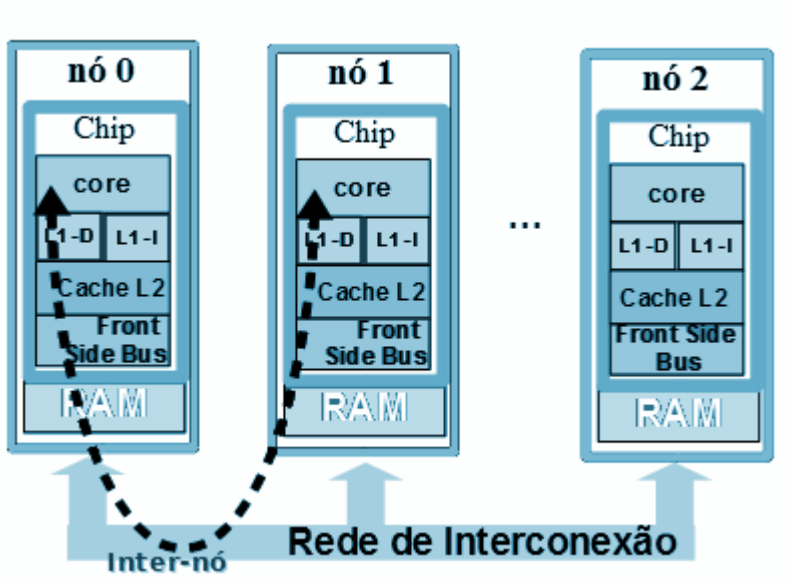


Figura 2.2: Cluster com um núcleo (*core*) por processador (*chip*) e um processador por nó, onde só há comunicação inter-nó para processos executando em paralelo.

2.1.2 Comunicação inter-chip ou intra-nó

A comunicação inter-chip, ilustrada na Figura 2.3, também conhecida como intra-nó, ocorre entre processadores (*chips*) que estão no mesmo nó.

Com a inclusão de mais processadores por nó, as bibliotecas para a paralelização de aplicações são capazes de explorar a memória compartilhada como meio de comunicação. No caso do MPI, é possível realizar a troca de mensagens entre os processos utilizando a memória principal do nó caso os processos estejam executando em processadores no mesmo nó. Uma vez que esse tipo de comunicação possui menor latência comparada à comunicação inter-nó, seria apropriado maximizar a troca de mensagens feita desta forma e, para isso, é necessário posicionar os processos que trocam maior quantidade de mensagens em processadores no mesmo nó.

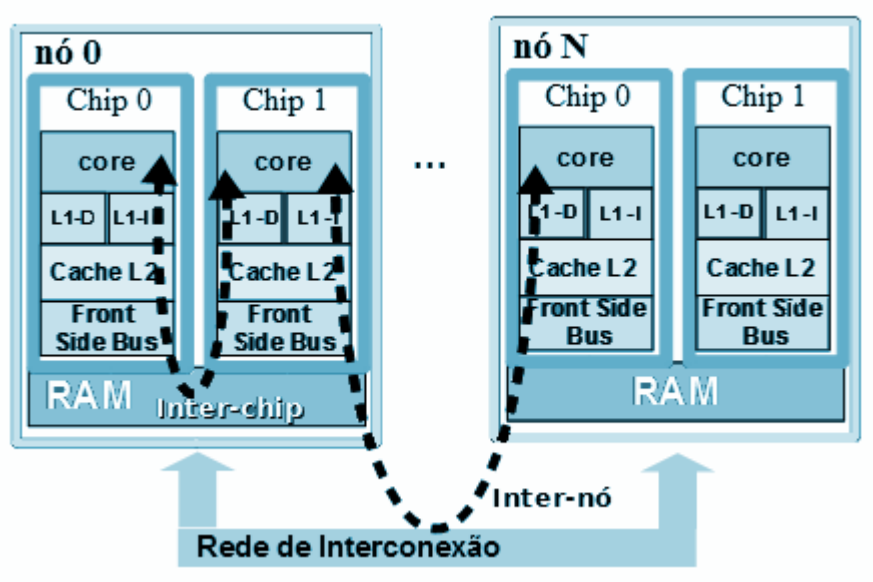


Figura 2.3: Cluster com mais de um processador por nó, onde há comunicação inter-nó e inter-chip.

2.1.3 Comunicação intra-chip

Com o surgimentos do processadores *multi-core*, a comunicação entre processos paralelos pode ocorrer dentro do processador (*chip*), denominada comunicação intra-chip.

Quando há compartilhamento de *cache* entre dois núcleos no mesmo chip, a troca de mensagens MPI pode ocorrer através dessa *cache* compartilhada. Entretanto, nem sempre o compartilhamento de *cache* está presente, pois ainda não é conhecida uma topologia de *cache* ideal para execução de aplicações genéricas (ALVES, 2009). As configurações variam desde o compartilhamento do segundo ou terceiro nível da memória *cache* por diversos núcleos, Figura 2.4, até nenhum compartilhamento, Figura 2.5. Caso nenhum nível de memória *cache* dentro do chip seja compartilhado, a comunicação intra-chip é feita através do *Front Side Bus* (FSB).

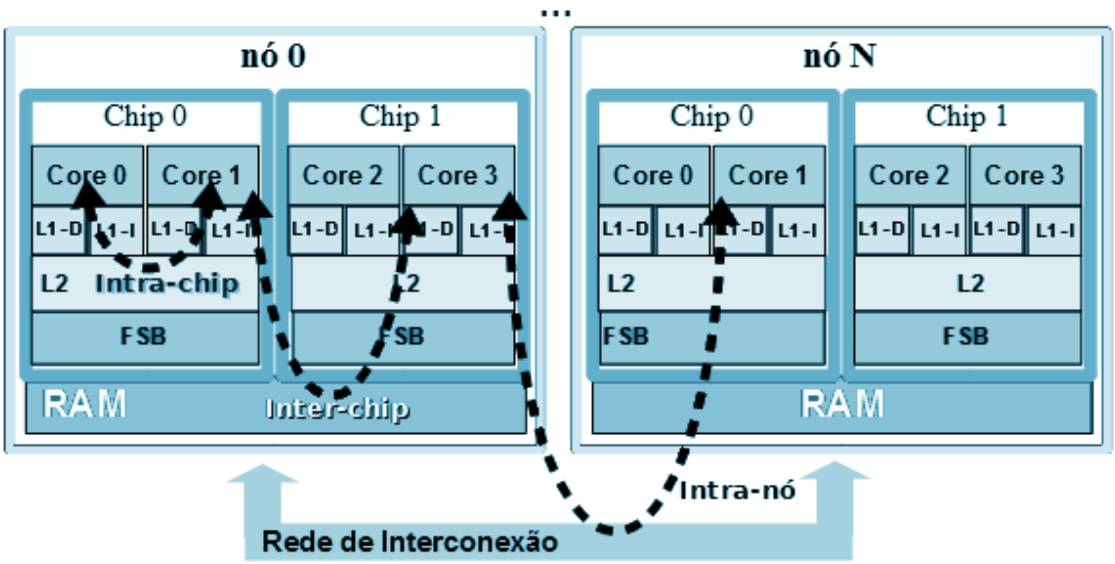


Figura 2.4: Cluster com mais de um núcleo (*core*) por processador (*chip*), com comunicação intra-chip através do compartilhamento de *cache*.

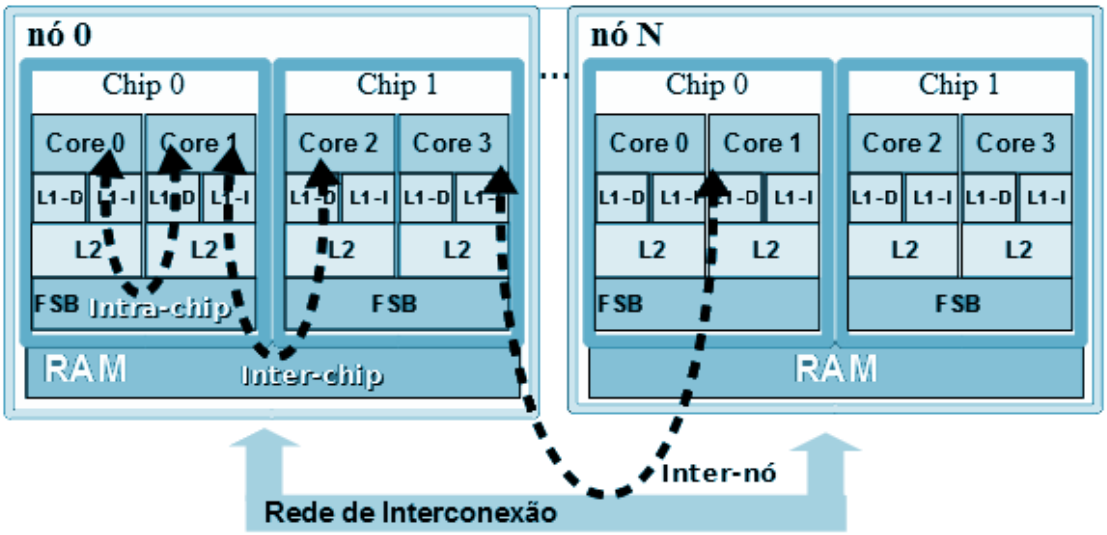


Figura 2.5: Cluster com mais de um núcleo (*core*) por processador (*chip*), com comunicação intra-chip sem compartilhamento de *cache*.

Apesar de se concluir que o nível de menor latência para a troca de mensagens é o intra-chip, nem sempre o seu uso implica em maior ganho de desempenho possível à aplicação como um todo. Algumas aplicações são favorecidas pelo uso deste nível da hierarquia de comunicação, enquanto outras obtém melhor desempenho explorando topologias que não compartilham nenhum nível da memória (MURPHY; KOGGE, 2007), conforme será vista na próxima seção.

2.2 Tempo de comunicação

Processos de aplicações paralelas frequentemente precisam se comunicar para sincronizar suas tarefas ou trocar dados processados. Quanto maior a quantidade e a latência de comunicação, mais tempo levará para a execução encerrar.

Entende-se por latência, o tempo médio necessário para um processo enviar uma mensagem para outro processo da aplicação. O tempo para enviar uma mensagem pode ser afetado por diversos fatores, entre eles: tamanho da mensagem, velocidade de canal de comunicação e contenção do canal por congestionamento.

2.2.1 Tamanho das mensagens

O que ocorre nos processadores atuais é que a comunicação feita através de um nível de hierarquia de memória compartilhada pode ter seu tempo afetado pelo tamanho das mensagens, pois o envio de mensagens grandes não é feito de forma contínua. Mensagens muito grandes são divididas em partes menores que caibam na memória compartilhada que será usada como canal de comunicação. Esse envio de parte por parte pode aumentar consideravelmente o tempo gasto para enviar a mensagem.

A Figura 2.6 ilustra o tempo necessário para completar a troca de mensagens de diferentes tamanhos nos três níveis de comunicação presentes em um cluster *multi-core*. Para os experimentos foram utilizados nós com dois processadores dual-core, onde cada dois núcleos compartilham uma *cache* nível 2 de 4 MB. Os nós do cluster estão conectados por uma rede InfiniBand.

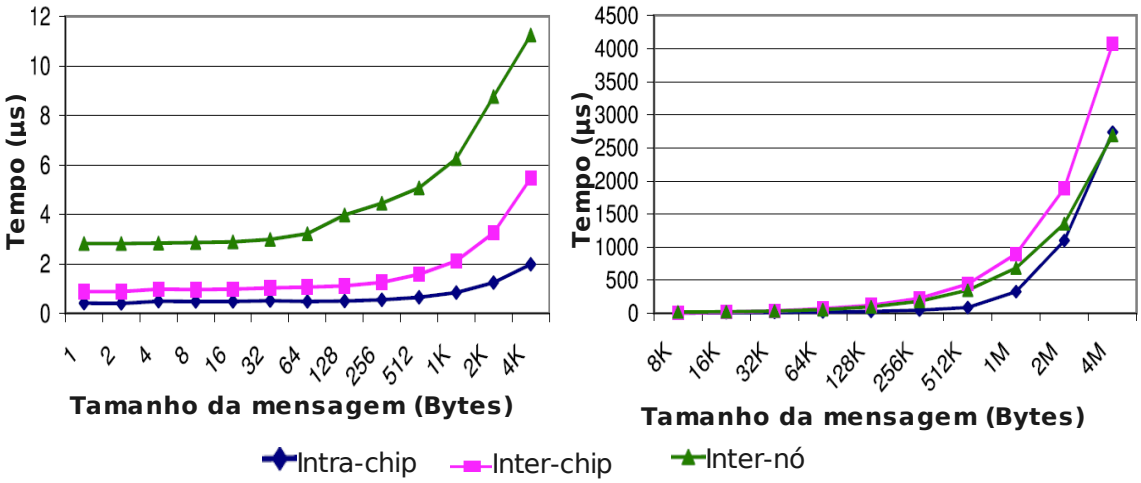


Figura 2.6: Tamanho da Mensagem *versus* comunicação intra-chip, inter-chip e inter-nó (CHAI; GAO; PANDA, 2007).

Para mensagens pequenas e médias, a comunicação intra-chip apresenta o menor tempo de comunicação quando comparada às comunicações inter-chip e inter-nó. Isso é devido ao fato de cada dois processadores compartilharem uma *cache* nível 2. Neste caso a comunicação exige apenas duas operações de *cache* se os *buffers* de comunicação estão na *cache*. Entretanto, para mensagens grandes, a comunicação inter-chip apresenta um tempo de comunicação maior que a inter-nó. Apesar de geralmente a velocidade de comunicação inter-chip ser menor que a inter-nó, a comunicação inter-chip se dá através de um *buffer* compartilhado onde duas cópias na memória principal

são necessárias. Enquanto que a comunicação inter-nó é feita através do *Remote Direct Memory Access* (RDMA) oferecido pela InfiniBand e o protocolo rendezvous (LUI; WU; PANDA, 2003) que forma um canal de cópia-zero. Isso também explica porque para mensagens grandes, maiores que a capacidade da *cache* de nível 2 compartilhada, a comunicação intra-chip e inter-nó apresentam tempos semelhantes (CHAI, 2009).

2.2.2 Canal de Comunicação

O tempo necessário para enviar uma mensagem é proporcional à latência do canal de comunicação. Como discutido previamente na seção 2.1, há pelo menos três níveis de comunicação em um cluster *multi-core*: intra-chip, inter-chip e inter-nó, em ordem crescente de latência para mensagens pequenas e médias, conforme visto na seção 2.2.1. O canal de comunicação utilizado depende do posicionamento do processo que dá origem à comunicação em relação ao processo destino. Se os processos estiverem em núcleos diferentes no mesmo chip, então pode-se utilizar a comunicação intra-chip, geralmente mais rápida. Assim, é importante analisar a aplicação para identificar os processos que possuem a maior quantidade de comunicação e posicioná-los de modo que eles utilizem os canais mais eficientes, minimizando o tempo de troca de mensagens.

2.2.3 Contenção do Canal por Congestionamento

A contenção ocorre quando diversos processos precisam utilizar intensamente o mesmo canal de comunicação, então ocorre um congestionamento. Para todos os níveis de comunicação citados pode haver contenção. Neste caso, a comunicação é tão intensa que não cabe no canal de comunicação. Assim, alguns processos precisam ficar esperando o canal ser liberado.

2.3 Processadores *Multi-core*

Apesar de os processadores com múltiplos núcleos existirem desde os anos 90 em sistemas dedicados, o surgimento de processadores *multi-core* de propósito geral tem alguns anos (FREITAS; ALVES; NAVAUX, 2009).

Antes dos *multi-core*, o ganho de desempenho em processadores se apoiou basicamente em dois pontos:

- aumento do paralelismo em nível de instrução inicialmente através da introdução do pipeline (HENNESSY; PETTERSON, 2007) e posteriormente do pipeline superescalar (SMITH, 1995). Este último além de apresentar o processamento de instruções dividido em estágios realiza uma sobreposição de instruções, utilizando para isso o aumento do número de unidades funcionais e técnicas para solucionar falsas dependências;
- aumento da frequência de trabalho do processador (ciclo de relógio).

Entretanto essas formas tradicionais de aumento de desempenho começaram a apresentar problemas físicos relacionados com o alto grau de integração dos componentes tais como atraso do fio e consumo de potência (ALVES, 2009). Os processadores *multi-core* surgiram como uma nova abordagem para aumentar o desempenho de processadores e ainda, algumas vezes, diminuir a potência dissipada. Para isso os *multi-core* exploram o paralelismo dos processadores em nível de fluxo de

execução. Pesquisas mostram que aplicações paralelas necessitam de um menor tempo de processamento quando executadas em processadores *multi-core* em comparação ao tempo de processamento quando executadas em processadores superescalares, ambos os processadores ocupado a mesma área física (OLOKOTUN et al., 1996).

Atualmente a grande maioria dos processadores de propósito geral são exemplos de arquiteturas *multi-core* com núcleos homogêneos para a execução de aplicações de propósitos gerais (GPP – *General Purpose Processor*) (AMD PHENOM, 2007) (INTEL I7-870, 2010).

2.3.1 Memória Cache em Processadores Multi-core

Assim como os processadores *single-core*, os processadores *multi-core* adotaram a hierarquia de memória como uma forma de melhorar o desempenho. Assim eles escondem a grande latência da memória principal utilizando níveis intermediários menores e mais rápidos entre o processador e a memória principal. Tendo em vista que as memórias rápidas possuem custo muito alto, uma hierarquia de memória é organizada com vários níveis, onde as memórias mais rápidas fica próximas das unidades de processamentos e, então, cada nível de memória abaixo é mais lento. Assim, no topo da hierarquia, mais próximo ao processamento, ficam as memórias de alta velocidade, porém de tamanho limitado e alto custo, e em cada nível abaixo desta hierarquia há memórias menos rápidas, de tamanhos maiores e de custo decrescente.

Apesar de ser um consenso a utilização da hierarquia de memória, sob o ponto de vista da organização desta hierarquia, os fabricantes apostam em diferentes soluções a fim de obter o melhor desempenho. Para verificar essa diferença, vejamos exemplos de organização de *multi-cores* das duas principais fabricantes de processadores de propósito geral: AMD e Intel.

Na Figura 2.7, vê-se exemplos de dois processadores da AMD (AMD, 2012), ambos possuem *cache* de nível 1 (64KB para instruções e 64KB para dados) e nível 2 não compartilhada, ou seja, exclusiva para cada núcleo, e *cache* de nível 3 compartilhada entre todos os núcleos do *chip*.

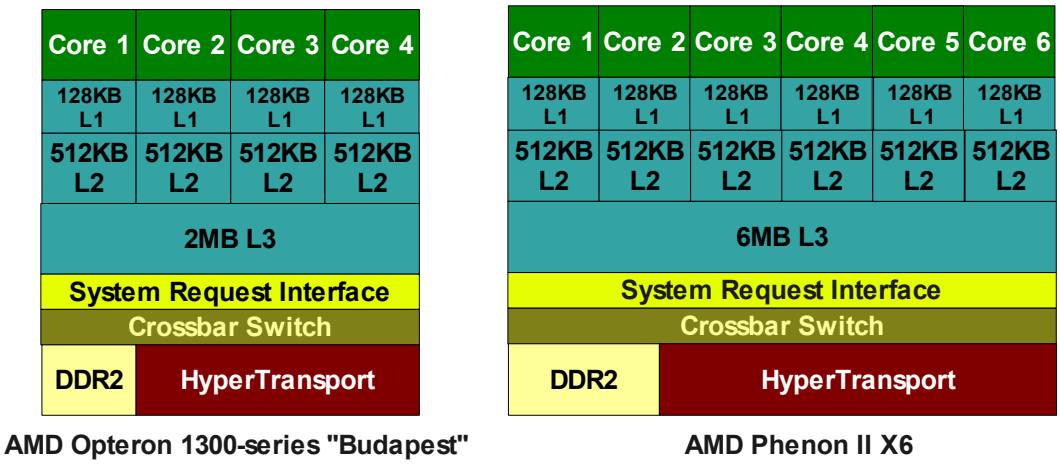


Figura 2.7: Exemplos de organização em processadores *multi-core* da AMD.

Para os processadores Intel (INTEL, 2012), existe a possibilidade ou não de compartilhamentos da *cache* L2, bem como da presença ou não de um terceiro nível de memória *cache*. A Figura 2.8a ilustra um quad-core Xeon, onde existe uma *cache* de nível 1 individual par cada unidade de processamento e uma *cache* de nível 2 compartilhada a cada dois núcleos e não há presença de *cache* nível 3.

Em contrapartida, na Figura 2.8b está a representação de um Core i7, também com quatro unidades de processamento, sendo que cada núcleo possui suporte a SMT (*Simultâneous Multithreading*) de até duas *threads* por núcleos, trabalhando, assim, com até 8 *threads* ativas no total. As memórias *cache* de primeiro e segundo nível desse processador são privadas para cada núcleo, possuindo uma memória *cache* de terceiro nível compartilhada entre todos os núcleos de processamento. Mesmo possuindo memória *cache* de segundo nível privada para cada núcleo, o projeto desse processador determina que qualquer núcleo poderá acessar dados na memória *cache* de nível 2 de outro núcleo a fim de obter melhor desempenho, assim, pode-se considerar essa memória *cache* como sendo de acesso não uniforme.

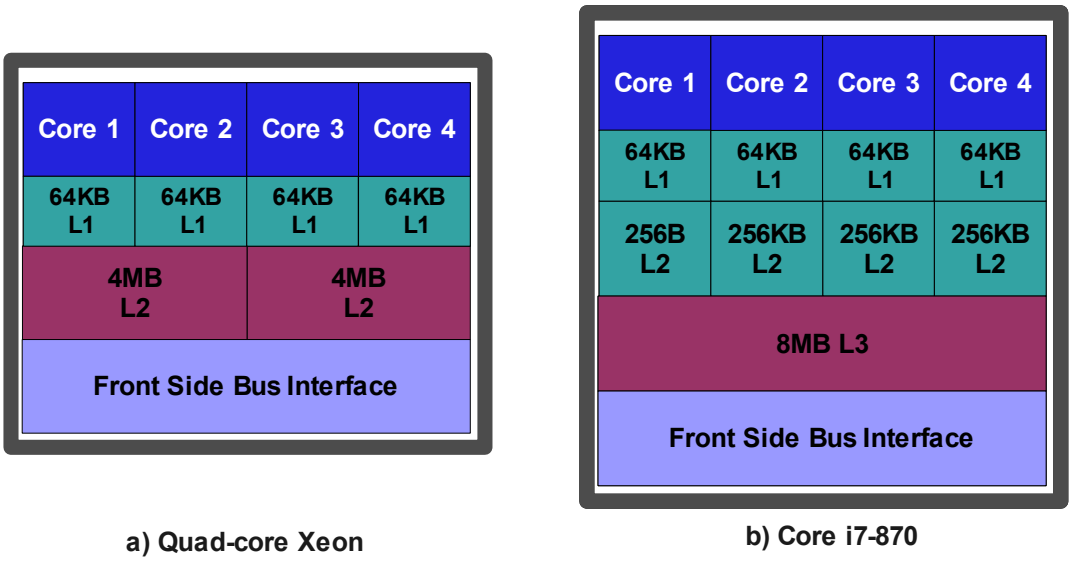


Figura 2.8: Exemplos de organização de processadores *multi-core* da Intel.

Assim, conclui-se que de um fabricante para outro existem variações quanto ao compartilhamento ou não da memória *cache* de nível 2 e, mesmo em processadores de um mesmo fabricantes, existem diferença na presença ou não da *cache* de nível 3. Isso mostra que essa questão ainda não está resolvida.

Em Alves (2009) foi realizado um estudo sobre qual o melhor compartilhamento de cache em processadores *multi-core* considerando o ganho de desempenho nas aplicações executadas, o consumo de potência e ocupação de área. O estudo simula diversas organizações de compartilhamento de cache nível 2 e nível 3 em um processador com 32 núcleos de processamento. Conclui-se que a melhor opção para o compartilhamento de uma cache de nível 2 de até 32MB é aquela em que dois cores compartilham uma cache de nível 2 de 2MB ou de não compartilhamento da *cache* L2. Em relação à *cache* de nível 3, o trabalho conclui que sua utilização traz apenas sobrecusto na ocupação de área e consumo de energia, além de aumentar em 20% o tempo de execução das aplicações.

2.4 Biblioteca de Comunicação MPICH2

MPI é uma especificação para um padrão de uma biblioteca para programação paralela através da troca de mensagens (GROPP; LUSK; SKJELLUM, 1999), ou seja, a comunicação entre os processos deve ser feita de forma explícita. Ao definir um padrão, buscou-se obedecer os seguintes critérios: praticidade, portabilidade, eficiência e flexibilidade.

MPICH2 é uma das diversas implementações disponíveis para o padrão MPI (MPICH2, 2012), que implementa os padrões MPI-1 e MPI-2 (inclusive gerenciamento de processos dinâmicos e I/O paralelo). Ela possui código aberto e é portável, sendo possível instalá-la tanto em ambiente Windows ou GNU/Linux. É uma das implementações mais populares de MPI, servindo de base para outras, tais como IBM MPI (para o Blue Gene), Intel MPI, Cray MPI, Microsoft MPI e Myricom MPI.

A seguir, será apresentado o método de comunicação padrão adotado na versão 1.3 da biblioteca MPICH2 (MPICH2 1.3, 2010), denominado *nemesis*.

2.4.1 Comunicação *Nemesis*

Desde sua versão 1.3, a implementação MPICH2 conta com o modo de comunicação denominado *nemesis* (BUNTINA; MERCIER; GROPP, 2007) como padrão. Deste modo, quando as mensagens são pequenas, menores que o tamanho do nível de hierarquia de memória compartilhada, a troca dessas é feita através de filas que são armazenadas no nível de hierarquia da memória que é compartilhada. Por exemplo, quando dois processadores compartilham uma memória *cache* de nível 2 de 4MB e trocam uma mensagem com tamanho menor que 4MB, ou seja, que cabe na *cache* de nível 2 compartilhada, essa mensagem é enviada por um *buffer* gravado em uma fila em uma região de memória compartilhada dentro desta *cache* (BUNTINA; MERCIER; GROPP, 2006). A Figura 2.9 mostra o envio de uma mensagem através dessas filas compartilhadas: (1) O processo de origem retira um elemento da sua fila *Free,* (2) preenche o elemento com a mensagem, (3) adiciona o elemento na fila de recebimento do processos destino. Então, (4) o processo destino retira o elemento de sua fila de recebimento, (5) processa a mensagem e (6) adiciona o elemento na mesma fila *Free* de onde ele havia sido retirado.

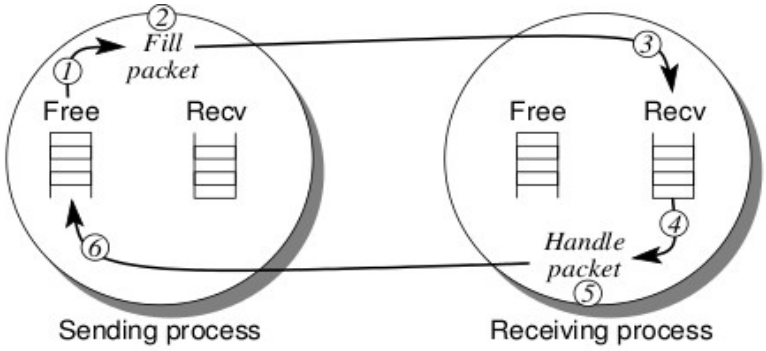


Figura 2.9: Diagrama do modelo de envio e recebimento de uma mensagem na comunicação *nemesis* (BUNTINA; MERCIER; GROPP, 2006).

No caso de mensagens maiores do que o tamanho da memória cache compartilhada, é utilizada a comunicação inter-nó feita com Infini-band, pois esta se mostrou mais eficiente que a comunicação por memória compartilhada. Isso é devido ao fato de mensagens maiores que a *cache* precisarem ser divididas, copiadas e lidas em várias operações, enquanto o RDMA oferecido pela InfiniBand é otimizado para comunicação inter-nó, pois permite que o processo destino acesse dados armazenadas no espaço de endereçamento na memória principal do processo origem no caso em que esses processos estão executando em nós diferentes.

3 MAPEAMENTO DE PROCESSOS

Uma introdução aos principais conceitos referentes ao mapeamento de processos paralelos, quais características do sistema e da aplicação podem ser avaliadas para determinar o mapeamento e como o mapeamento pode ser aplicado é importante para justificar porque esse trabalho trata de mapeamento estático de processos considerando a quantidade de comunicação entre os processos.

Neste capítulo são apresentados os principais conceitos referentes ao mapeamento de processos, o funcionamento da heurística BRD e da técnica de BE para o cálculo do mapeamento estático, bem como suas respectivas complexidades. Por fim, é descrito um levantamento bibliográfico das pesquisas atuais nessa linha.

3.1 Introdução ao Mapeamento de Processos

O mapeamento de processos paralelos é o ato de decidir em qual unidade de processamento um determinado processo irá executar, enquanto o escalonamento de processos determina a sequência de execução das tarefas (LAM et al., 2008). O mapeamento de processos de aplicações paralelas é um problema NP-difícil (BOKHARI, 1981), assim, para calculá-lo em um tempo hábil, são utilizados algoritmos heurísticos que levam em consideração as características da aplicação paralela e da organização do sistema para se aproximar do mapeamento mais adequado (KOPPLER, 1999).

O mapeamento de processos pode ser dividido em dois tipos:

- Mapeamento Estático: onde o mapeamento é feito somente uma vez no início da execução da aplicação e se mantém o mesmo até o final desta execução;
- Mapeamento Dinâmico: quando o mapeamento dos processos pode variar ao longo da execução da aplicação paralela, conforme os padrões de comunicação entre os processos ou a disponibilidade dos recursos se modifique. Neste caso, surge mais um desafio, a migração de processos em execução.

Tanto o mapeamento estático como o mapeamento dinâmico podem considerar diversas características da aplicação paralela e da arquitetura onde se deseja executar essa aplicação, chamada de **arquitetura alvo**. Com o objetivo comum de diminuir o tempo de execução das aplicações paralelas, cada abordagem de mapeamento pode levar em consideração diversas combinações de características de forma a fazer o melhor casamento entre aplicação e arquitetura.

Uma combinação possível em sistemas heterogêneos é considerar quais processos devem ser mapeados para núcleos especializados em executar o conjunto de instruções

dominante nestes processos. Outro desafio do mapeamento de processos presente em sistemas homogêneos e heterogêneos é encontrar uma distribuição que diminua o atraso de comunicação entre os processos. Diminuindo o atraso de comunicação, diminui-se o tempo total de execução da aplicação paralela. Isso porque quanto mais tempo os processos permanecem ociosos aguardando o envio ou recebimento de mensagens de outros processos para continuar o processamento (GROPP; LUSK; SKJELLUM, 1999), mais demorada é a execução da aplicação como um todo.

O mapeamento é um conceito aplicado em sistemas que possuem mais de uma unidade de processamento, ou seja, em sistemas onde os processos paralelos de uma aplicação podem ser enviados para várias opções de unidades de processamento. As principais bibliotecas de paralelização MPI executam por padrão uma distribuição *round-robin* dos processos da aplicação nas unidades de processamento. Como foi dito anteriormente, esse método não leva em consideração nenhuma característica da aplicação ou da arquitetura alvo para distribuir os processos paralelos de forma que a execução da aplicação obtenha melhor desempenho.

Com a popularização dos processadores com múltiplos núcleos, surgiu a possibilidade de realizar o mapeamento de processos também pensando nos diversos núcleos existentes dentro de um processador. O compartilhamento de recursos entre núcleos, tais como memória *cache*, nos CMPs, aparece como uma nova possibilidade de comunicação entre os processos paralelos. Assim, ao realizar o mapeamento de processos que se comunicam frequentemente, é necessário considerar a capacidade de comunicação entre os núcleos para os quais os processos serão mapeados.

3.2 Estado da Arte em Mapeamento de Processos

Há diversos trabalhos de mapeamento de processos em máquinas Simétricas Multiprocessadas (SMP) (BHANOT et al., 2005) (BOLLINGER, 1988), mas esses trabalhos não consideram processadores *multi-core* nem consideram o compartilhamento de recursos entre os núcleos. Sendo assim, técnicas de mapeamento avaliadas nesses trabalhos podem não funcionar bem em processadores *multi-core*.

Em Rodrigues, Madruga, Navaux (2009), é apresentado uma estratégia para reduzir o tempo de comunicação em aplicações paralelas que tenham um padrão de comunicação estático identificado através da análise de uma execução prévia da aplicação. É exigido somente que a aplicação utilize o paradigma de troca de mensagens, não sendo necessária a modificação do código original. Utilizando o algoritmo de Biparticionamento Recursivo Dual, o padrão de comunicação identificado é mapeado em um cluster *multi-core* levando em consideração o compartilhamento de *cache*. Foi obtido um *speed-up* de 9% em média nos experimentos realizados, utilizando apenas uma técnica de mapeamento.

Uma nova heurística para mapeamento de processos é proposta e avaliada em Chen et al. (2006). Os autores consideram os diferentes tempos de interconexões em um nó *multi-core* e utilizam essa informação em seu algoritmo, que também faz o mapeamento de um grafo que representa a comunicação de dados de uma aplicação em um grafo que representa as latências de comunicação entre os núcleos do sistema. Neste trabalho também é necessário uma execução prévia para caracterizar o padrão de comunicação da aplicação a ser mapeada. Entretanto, eles não comparam o mapeamento de processos para nós com o mapeamento para nós e núcleos, estes últimos num mesmo *chip*.

O mapeamento de processos em cluster *multi-core* também é explorado em Dümmler, Rauber, Rünger (2008). Mas essa técnica de mapeamento é desenvolvida com o modelo de programação de *Multiprocessor-tasks* combinado com diferentes equações para fazer o mapeamento. Assim, ela exige que a programação esteja no modelo *Multiprocessor-task*, não sendo empregados nas aplicações fora desse modelo.

O trabalho de Zhang, Yuan, Srinivasan (2010) realiza o mapeamento de processos MPI em cluster*s* Simétricos Multiprocessados (SMP) com nós de *multi-cores* (CMPs). O mapeamento é feito através da minimização do número de comunicações inter-nó. Para isso é feito o cálculo de comunicação inter-nó em todos os mapeamentos possíveis. A principal conclusão deste trabalho é a de que a razão entre comunicação inter-nó e inter-chip deve ser considerada para o cálculo do mapeamento. Nesse trabalho não é considerada a comunicação intra-chip.

Em Broquedis et al. (2010) é apresentado e avaliado o software *Hardware Locality* (hwloc) que colhe informações do hardware relacionadas aos núcleos, *caches* e memórias e disponibiliza estas informações para as aplicações de uma forma abstrata. Utilizando aplicações MPI e OpenMP/MPI (mistas) foram feitas avaliações do hwloc. Com as informações fornecidas pelo software foi realizado o mapeamento das aplicações de forma estática e dinâmica. Comparando o mapeamento estático dos processos feito com a distribuição *round-robin* em relação ao feito considerando as informações fornecidas pelo hwloc também utilizando o algoritmo de Biparticionamento Recursivo Dual conseguiu-se ganhos de até 26% no tempo de execução de aplicações.

O mapeamento dinâmico de processos levando em consideração a largura de banda é abordado em Koukis (2006) e em Xu (2010). Essas abordagens, ao invés de focarem no aproveitamento do compartilhamento de dados entre os processos, têm o objetivo de alocar processos que utilizam mais o canal de comunicação junto com processos que utilizam menos, a fim de evitar a saturação do canal.

Em Kouris, é apresentado um algoritmo que através de informações colhidas dinamicamente sobre a utilização do canal de comunicação – memória compartilhada ou rede de interconexão – escolhe processos para serem executados no mesmo SMP a fim de que o barramento da memória compartilhada e a placa de rede não fiquem nem saturados e nem subutilizados. Apesar de não exigir modificações nas aplicações paralelas, para aplicar o algoritmo é necessário modificar o *firmware* da placa de rede (*Network Inteface Card* - NIC), a fim de colher informações sobre utilização e contenção na NIC. A ideia principal do algoritmo proposto é alocar um processo com grande necessidade de utilização de banda no mesmo SMP de um processo com pequena necessidade, buscando que a capacidade do canal seja ao máximo utilizada, desde que não seja ultrapassada. O cálculo do mapeamento é refeito em determinados intervalos de tempo baseado na informação de utilização dos canais de comunicação colhida no intervalo de tempo anterior.

No trabalho de Xu é constatado que as aplicações paralelas sofrem uma degradação em sua desempenho quando a capacidade máxima do canal de comunicação é atingida. Assim, ao invés de alocar no mesmo SMP processos que juntos atinjam o máximo da

capacidade do canal, é proposta uma alteração do algoritmo de Kouris para que, para um determinado intervalo de tempo, sejam alocados juntos processos que atinjam apenas a média da capacidade de banda necessária para toda a carga de trabalho em todos os intervalos de tempo anteriores. Foi atingido um aumento na taxa de transferência de dados de até 11,7% e de em média 4,1%.

Em Cruz, Alves, Navaux (2010), o Emparelhamento Perfeito de Custo Máximo (EPCM) é utilizando para calcular o mapeamento estático de *threads* OpenMP em um processador *multi-core*. Foi medido o ganho de desempenho alcançado em relação ao mapeamento padrão executado pelo sistema operacional que chegou a 42%. Entretanto, não é feita uma comparação de desempenho do algoritmo com nenhum outro método de mapeamento estático.

3.3 Algoritmos de Mapeamento Estático de Processos

Todos os algoritmos de mapeamento estático utilizados neste trabalho buscam minimizar o custo de comunicação entre os processos paralelos a fim de diminuir o tempo de execução da aplicação. Para isso, é exigida uma execução prévia da aplicação paralela a fim de obter uma caracterização do padrão de comunicação entre os processos. Assim, o mapeamento estático de processos deve ser empregado em aplicações paralelas que possuem um padrão de comunicação constante entre os processos e que são executadas repetidas vezes. Dessa forma, o custo de executar previamente a aplicação e calcular o mapeamento pode ser compensado pela melhoria alcançada no desempenho da aplicação durantes as demais execuções.

Cada processo da aplicação paralela é distribuído para uma unidade de processamento nos algoritmos analisados. Desta forma, todos os processos podem executar em paralelo e não precisam dividir uma unidade de processamento e, por consequência, ficar esperando nos momentos em que outro processo está executando.

A seguir é descrita a heurística BRD utilizada para comparação neste trabalho. Depois é descrita a BE, o método de força bruta utilizado como base para ilustrar o quanto é possível ganhar com o mapeamento.

3.3.1 Biparticionamento Recursivo Dual

O Biparticionamento Recursivo Dual (BRD) é uma heurística que se baseia no método de divisão e conquista para achar uma solução aproximada para o problema do mapeamento (PELLEGRINI; ROMAN, 1996). O objetivo do algoritmo BRD é mapear um grafo que representa a aplicação paralela em um grafo que representa a arquitetura alvo.

O grafo da aplicação, ou grafo $S(V_S, E_S)$, é um grafo valorado não direcionado cujos vértices $(v_S \in V_S)$ representam os processos do programa paralelo e cujas arestas $(e_S \in E_S)$ representam a comunicação entre esses processos. Os valores associados aos vértices representam o custo computacional necessário para executar aquele processo $(c_S(v_S))$. A quantidade de comunicação entre os processos é representada pelos valores associados às arestas ($c_S(e_S)$).

O grafo da arquitetura, ou grafo $T(V_T, E_T)$, representa a topologia da máquina alvo. Sendo a arquitetura alvo homogênea, o grafo T não tem valores atribuídos aos seus vértices $(v_T \in V_T)$. Entretanto, devido ao fato de os *links* de comunicação entre os processadores possuírem velocidades de comunicação diferentes, são associados às arestas $(e_T \in E_T)$ valores inversamente proporcionais à velocidade do *link,* representados por $(c_T(e_T))$. A distância de comunicação entre os nós do grafo da

arquitetura será dada pela distância entre os vértices, contada pelo peso de cada aresta do caminho. Assim, quanto mais arestas demoradas for necessário percorrer para ir de um vértice a outro, mais demorada será a comunicação entre esses vértices.

Um mapeamento do grafo $S(V_S, E_S)$ para o grafo $T(V_T, E_T)$ pode ser representado como duas funções $\varphi : V_S \rightarrow V_T$, onde $\varphi(v_S)=v_T$ se o processo v_S for mapeado para o processador v_T; e $\lambda : E_S \rightarrow E_T$, onde $\lambda(e_S)=\{e1_T, e2_T, \ldots, en_T\}$ se o canal de comunicação e_S é mapeado para os links $e1_T, e2_T, \ldots, en_T$. $|\lambda(e_S)|$ é o número de arestas de E_T usadas para fazer a rota usada pela arestas e_S.

O algoritmo do BRD utiliza o método de divisão e conquista para recursivamente mapear processos em um subconjunto de processadores. O algoritmo pode ser visto abaixo.

```
1.  mapping(VS, VT)
2.  Set_Of_Processes VS;
3.  Set_Of_Processors VT;
4.  {
5.      Set_Of_Processes VS0, VS1;
6.      Set_Of_Processors VT0, VT1;
7.      if (|VS| = = 0)
8.        return;
9.      if (|VT| = = 1)
10.       result(VS,VT);   //VS é mapeado para VT
11.     (VT0, VT1) = Processor_bipartition(VT);
12.     (VS0, VS1) = Process_bipartiotion(VS, VT0, VT1);
13.     list += mapping(VS0, VT0);    //Executa a recursão
14.     list += mapping(VS1, VT1);
15. }
```

Figura 3.1: Algoritmo de BRD (PELLEGRINI, 1994).

No algoritmo BRD ilustrado na Figura 3.1 o biparticionamento recursivo cria uma árvore binária de níveis de recursão. Ao colocar cada novo nível de mapeamento da recursão em um fila de tarefas a ser executada (linhas 13 e 14) e executar as tarefas a partir da primeira inserida na fila até a última, o algoritmo avança na árvore de recursão nível a nível (*breadth-first*), descendo para o próximo nível apenas quando todos os mapeamentos de subconjunto de processos para subconjunto de processadores do nível anterior foram determinados. Esse avanço nível a nível permite que o custo de comunicação entre processos que estão localizados em subconjunto diferentes de processadores seja mais precisamente calculado, uma vez que as informações sobre o mapeamento realizado um nível acima na recursão já estão disponíveis.

A cada nível da recursão do algoritmo descrito na Figura 3.1 ocorre a minimização da função parcial de custo definida como:

$$f'c(\varphi,\lambda)=\sum_{v\in V'_S \wedge \{v,v'\}\in E_S} (c_S(\{v,v'\}).c_T(\lambda(\{v,v'\}))) \quad (3.1)$$

A função acima considera a comunicação entre os processos pertencentes ao subconjunto que está sendo analisado bem como a comunicação dos processos com outros processos, que não pertencem ao conjunto analisado.

A comportamento temporal[1] do método BRD é definido como sendo $O(|E_S|\ |log_2(|V_T|)$ (PELLEGRINI; ROMAN, 1996) onde $|E_S|$ é o número de arestas do grafo que representa a comunicação entre os processos da aplicação e $|V_T|$ representa o número de vértices do grafo que representa a arquitetura alvo. Neste trabalho, o grafo representando a aplicação é sempre um grafo completo e possui o mesmo número de vértices que o grafo da arquitetura alvo, logo esse comportamento temporal pode ser representado relativo apenas ao número de vértices do grafo da aplicação, ou seja, o número de processos da aplicação paralela que iremos representar pela letra *n*, conforme abaixo:

$$\begin{aligned} &O(|E_S|\log_2(|V_T|))= \\ &O(\frac{n^2-n}{2}\log_2(n))= \\ &O(n^2\log_2(n)) \end{aligned} \tag{3.2}$$

Entretanto ao calcular a complexidade temporal do BRD, baseada na função de minimização definida anteriormente em f(3.1) durante a execução do algoritmo ilustrado na Figura 3.1, conclui-se que ela é $O(n^3)$ quando executado sobre um grafo completo.

$$O(n^3)\ para\ BRD$$

O método de BRD está implementado em uma aplicação chamada Scotch, versão 5.1 (PELLEGRINI et al., 2010), utilizada neste trabalho. Com ela, foi calculado o mapeamento estático de processos paralelos levando em consideração a comunicação entre esses processos a fim de comparar o ganho de desempenho em relação à proposta de mapeamento descrita no Capítulo 4.

3.3.2 Busca Exaustiva

A Busca Exaustiva (BE) é um método de força bruta utilizado para que seja possível determinar o ganho máximo que pode ser alcançado pelo mapeamentos estático de processos.

O objetivo do algoritmo BE é, como nas duas heurística comparadas neste trabalho, minimizar o custo da comunicação entre os processos paralelos para, assim, diminuir o tempo de comunicação e de execução da aplicação paralela. Para isso o algoritmo calcula o custo da comunicação para todas as possíveis distribuições dos processos nos núcleos de processamento.

---

[1]O comportamento temporal (*time behavior*) foi definido baseado nos resultados obtidos em experimentos e não no cálculo matemático.

Para este trabalho a BE foi implementada em Python. A escolha da linguagem de programação Python foi exclusivamente pela facilidade de manipulação de listas oferecida por ela.

A partir do padrão de comunicação da aplicação e do custo de comunicação associado a cada nível de compartilhamento de memória, cada possível mapeamento é avaliado. Isso resulta em uma complexidade fatorial em relação ao número de processos da aplicação paralela, referido por *n*.

$$O(n!) \text{ para BE}$$

Sendo esse o algoritmo de força bruta para encontrar o melhor mapeamento de processos considerando o volume de comunicação, ele possui a mesma complexidade atribuída ao problema do mapeamento (BOKHARI, 1981).

Apesar de o algoritmo BE fornecer o melhor mapeamento, a complexidade fatorial faz com que, conforme o número de processos da aplicação aumente, a solução do problema utilizando esse método demore tanto tempo que se torne inviável.

4 PROPOSTA DE MAPEAMENTO ESTÁTICO EMPREGANDO O EMPARELHAMENTO PERFEITO DE CUSTO MÁXIMO

Neste capítulo, é apresentada a MapEME (Mapeamento Estático MPI com Emparelhamento) nossa proposta de heurística para calcular o mapeamento estático de processos MPI que emprega o Emparelhamento Perfeito de Custo Máximo (EPCM).

Inicialmente, são abordados alguns conceitos básicos referentes ao emparelhamento. Em seguida, são descritos o funcionamento do algoritmo de EPCM e a forma como este foi aplicado no padrão de comunicação das aplicações paralelas para obter o mapeamento. A complexidade do método também é descrita para possibilitar uma comparação com os demais algoritmos utilizados neste trabalho.

4.1 Introdução ao EPCM

Para garantir o entendimento do algoritmo EPCM como heurística para o mapeamento são explicados inicialmente alguns conceitos básicos referentes ao emparelhamento de grafos.

Conforme West (2011), o emparelhamento de um grafo é definido como um conjunto de arestas sem vértices em comum, podendo ser também um grafo inteiro composto de arestas sem vértices em comum. Um vértice está emparelhado se estiver ligado a uma das arestas pertencentes ao emparelhamento. Do contrário, o vértice estará desemparelhado. A definição formal para o emparelhamento de um grafo é:

Dado um grafo G=(V,E), o emparelhamento M em G é um conjunto de arestas não adjacentes, ou seja, nenhuma aresta tem vértices em comum com outra.

Um **emparelhamento maximal** é o emparelhamento M de um grafo G com a propriedade de que se qualquer aresta de G que não estiver em M for adicionada a M, ele deixa de ser um emparelhamento, isto é, M é maximal se não for um subconjunto de nenhum outro emparelhamento no grafo G. Em outras palavras, um emparelhamento M de um grafo G é maximal se cada aresta em G tem um vértice em comum com no mínimo uma aresta em M. A Figura 4.1 mostra exemplos de emparelhamentos maximais.

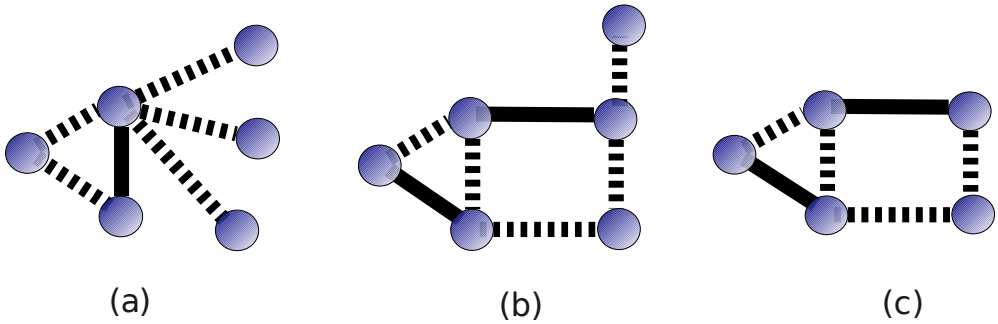


Figura 4.1: Exemplos de emparelhamento maximal de grafos. As arestas contínuas indicam o emparelhamento. As arestas tracejadas não fazem parte do emparelhamento descrito e ilustram as demais arestas do grafo original.

Um **emparelhamento máximo** é um emparelhamento que contem o maior número possível de arestas. É possível que haja muitos emparelhamentos máximos para o mesmo grafo. Note que, todo o emparelhamento máximo é maximal, mas nem todos os emparelhamentos maximais são emparelhamentos máximos. Na Figura 4.2 estão ilustrados exemplos de emparelhamentos máximos.

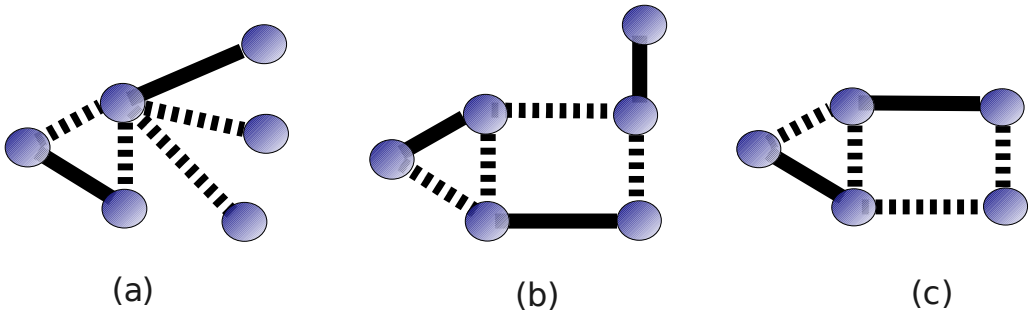


Figura 4.2: Exemplos de emparelhamento máximo de grafos. Apenas o exemplo (b) é um emparelhamento perfeito. As arestas contínuas indicam o emparelhamento. As arestas tracejadas não fazem parte do emparelhamento descrito e representam as demais arestas do grafo original.

O **emparelhamento perfeito** é o emparelhamento Mp que contém todos os vértices no grafo. Cada vértice do grafo é incidente em exatamente uma aresta do emparelhamento. Todo o emparelhamento perfeito é máximo e, consequentemente, maximal. Na Figura 4.2, apenas o exemplo (b) ilustra um emparelhamento perfeito.

Um grafo onde as arestas possuem um peso associado é denominado um grafo valorado. Assim, o valor de um emparelhamento é dado pela soma dos pesos de suas arestas. Um emparelhamento perfeito Mp de G possui o **custo máximo** quando o seu peso é o maior dentre todos os emparelhamentos perfeitos obtidos a partir do grafo G.

4.2 Heurística MapEME

Conforme visto na seção anterior, o emparelhamento forma um conjunto de pares disjuntos de vértices de um grafo. O Emparelhamento Perfeito de Custo Máximo (EPCM) forma pares disjuntos a partir de todos os vértices de um grafo de forma que seja máxima a soma dos valores atribuídos às arestas que ligam esses pares. Desta forma, o EPCM pode ser utilizado para descobrir os pares de processos que se comunicam mais e, assim, definir o mapeamento dos processos em um tempo polinomial através da modelagem dos processos como os vértices do grafo e da quantidade de comunicação entre cada processo como arestas valoradas do grafo.

A MapEME é uma heurística que aplica o EPCM para calcular o mapeamento estático em aplicações MPI em um tempo polinomial considerando a quantidade de comunicação entre os processos e a velocidade de comunicação entre os núcleos de processamento. A informação da organização da arquitetura alvo está implícita em cada vez que a MapEME aplica o EPCM em um grafo valorado que representa o padrão de comunicação da aplicação. A Figura 4.3 mostra os passos executados pela MapEME para o mapeamento de uma aplicação de 8 processos em uma arquitetura de 8 núcleos com três níveis de comunicação. Vê-se que é necessário executar o EPCM uma vez para cada nível de compartilhamento, partindo do mapeamento gerado na execução anterior. Por esse motivo, a heurística MapEME deve ser empregada em aplicações cujo número de processos seja uma potência de 2 e que executem em arquiteturas cujos núcleos de processamento estejam agrupados em números de potências de 2 conforme o nível de comunicação.

Os passos executados pela MapEME, conforme a Figura 4.3, são:

(1) Inicialmente tem-se a matriz que representa o padrão de comunicação entre todos os processos da aplicação paralela, no caso 8. Logo tem-se uma matriz 8x8 onde cada posição da matriz, x_{a1a2}, representa a quantidade de comunicação do processo *a1* para o processos *a2*. Essa matriz é oferecida como entrada para o algoritmo de EPCM;

(2) A primeira aplicação do EPCM divide os processos em pares disjuntos, de forma que a soma da comunicação entre eles, representada pelo valor das arestas, seja máxima;

(3) A aplicação do EPCM gera uma lista de pares de processos;

(4) Com a lista de pares de processos geradas no passo (3) é gerada uma outra matriz 4x4, onde cada posição da matriz representa a quantidade de comunicação entre os pares de processos que indexam a matriz. Com a otimização do algoritmo, essa passo de criação de matriz intermediária foi substituído pelo acesso direto a posições necessárias da matriz original obedecendo a equação abaixo;

$$H(par(a1,a2),par(a3,a8)) = x_{a1a3} + x_{a3a1} + x_{a1a8} + x_{a8a1} + x_{a2a3} + x_{a3a2} + x_{a2a8} + x_{a8a2} \quad \text{f(2)}$$

(5) O EPCM é aplicado novamente sobre os pares de processos gerados no passo anterior;

(6) Então, é gerada uma nova lista, agora com pares de pares de processos;

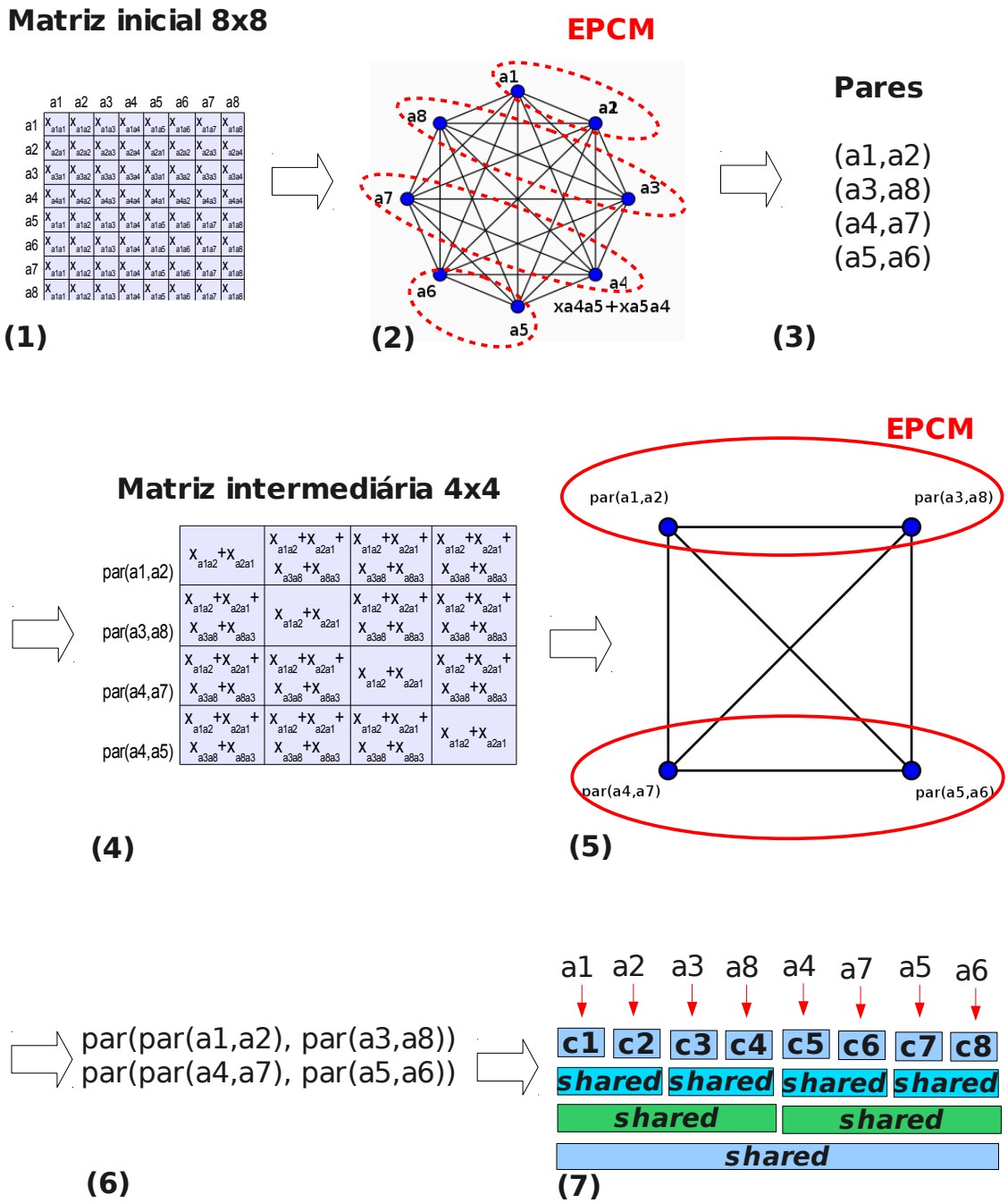


Figura 4.3: Exemplo do fluxo a execução da MapEME para o mapeamento de 8 processos em 8 unidades de processamento que possui três níveis de compartilhamento de memória.

(7) Por se tratar de uma arquitetura alvo com três níveis de comunicação, ou seja, três níveis de compartilhamento, não é necessário aplicar novamente o EPCM, pois já é possível realizar o mapeamento a partir da lista gerada no passo (6). Os pares gerados inicialmente são mapeados para os núcleos que compartilham o primeiro nível de comunicação, mais rápido. Da mesma forma, os pares encontrados nos passos seguintes são mapeados para os núcleos que compartilhem o nível de comunicação seguinte.

4.3 Complexidade da heurística MapEME

Conforme visto na seção anterior, a MapEME aplica diversas vezes o EPCM para encontrar o mapeamento para os processos.

Existe um algoritmo sequencial para o cálculo do Emparelhamento Perfeito de Custo Máximo que possui complexidade temporal de $O(n^3)$ (KOLMOGOROV; BLOSSOM, 2009). Mas há uma versão paralela do algoritmo que calcula o EPCM em grafos completos com uma complexidade temporal $O(n^2 \lg n)$ quando executada com N processos (OSIAKWAN; AKL, 1990). Assim, conforme as equações abaixo, é possível deduzir a complexidade da heurística MapEME como:

$$O\left(\sum_{i=0}^{\sqrt{n}-2} \left(\left(\frac{n}{2^i}\right)^2 + \left(\frac{n}{2^i}\right)^2 . lg\left(\frac{n}{2^i}\right)\right)\right) \rightarrow$$
$$O(n^2 . lg(n)) \; para \; MapEME$$

4.4 Comparação da MapEME com o BRD

A MapEME possui uma complexidade temporal calculada de forma matemática um grau menor que a BRD.

$$O(n^3) \; para \; BRD \qquad O(n^2 . lg(n)) \; para \; MapEME$$

Como visto na Seção 3.3.1, o BRD possui uma abordagem que utiliza a minimização da quantidade de comunicação para calcular o mapeamento. Essa minimização ocorre do nível de comunicação mais demorado para o mais rápido. Por se tratar de um algoritmo guloso, o que é decido à medida que se caminha do nível mais demorado de comunicação para o mais rápido não é revisto.

Da mesma forma, o MapEME é um algoritmo guloso, mas sua abordagem é a de maximizar a comunicação entre os processos que compartilham do nível de comunicação mais rápido para o nível de comunicação mais demorado.

Se não fossem gulosos, esses algoritmos calculariam o mesmo resultado de mapeamento. Mas o fato de as decisões tomadas ao longo da execução não serem revistas faz com que eles possam encontrar soluções diferentes.

Além disso, o fato de o EPCM decidir primeiramente os pares de processos que compartilharão o nível de memória mais rápido pode resultar em um mapeamento mais adequado em processadores *multi-core* que compartilham *caches*.

4.5 Considerações Sobre a Proposta

A MapEME, proposta de mapeamento utilizando o EPCM, apresentada neste trabalho tem o objetivo de realizar o mapeamento estático de processos considerando dois fatores: a heterogeneidade no volume de comunicação entre os processos em aplicações MPI; e a heterogeneidade na velocidade de comunicação entre as unidades de processamento de um cluster *multi-core*.

Os trabalhos de Kouris (2006) e Xu (2010) realizam o mapeamento dinâmico, mas os ganhos são inferiores aos trabalhos de Broquedis et al. (2010), Rodrigues, Madruga, Navaux (2009) e Cruz, Alves, Navaux (2010) que exploram o mapeamento estático. Assim, mesmo considerando que o mapeamento estático exige uma execução prévia da aplicação para ser calculado, acredita-se que o maior ganho alcançado justifica a exploração de novas formas de calculá-lo. Por esse motivo, a proposta apresentada aborda o mapeamento estático de processos.

Apesar dos trabalhos de Rodrigues, Madruga, Navaux (2009), Chen et al. (2006), Zhang, Yuan, Srinivasan (2010), Broquedis et al. (2010) e Cruz, Alves, Navaux (2010) considerarem processadores *multi-core* em seus experimentos, com exceção de Cruz, nenhum dos trabalhos utiliza o EPCM como heurística de mapeamento estático. Considerando que Rodrigues, Broquedis e Chen utilizam o Biparticionamento Recursivo e Zhang utiliza um método de força bruta que analisa todos os mapeamentos possíveis. Em Cruz, Alves, Navaux (2010), onde é utilizado o EPCM, são analisadas aplicações OpenMP executadas em um processador *multi-core*, além disso, o EPCM não é comparado a nenhum outro método de mapeamento.

A MapEME, heurística proposta aqui, utiliza o EPCM e tem o seu desempenho avaliado neste trabalho em aplicações MPI utilizando nós *multi-core*, sendo comparado com o desempenho de outra heurística, a BRD, já difundida.

4.6 Caracterizando o Padrão de Comunicação das Aplicações MPI

Para mensurar o volume de comunicação entre os processos de uma aplicação foi feita uma modificação na ferramenta MPE (MPI *Parallel Environment*) distribuída com a implementação MPICH2 (MPICH2, 2012) do padrão MPI. A ferramenta MPE possui diversas funções para a visualização e depuração do comportamento das aplicações MPI. Apesar de ser distribuída com a biblioteca MPICH2, a ferramenta MPE pode ser utilizada junto com outras implementações do padrão MPI.

O módulo MPE modificado gera duas matrizes durante uma execução prévia da aplicação paralela analisada:

- a primeira matriz é gerada através da soma do tamanho de todas as mensagens trocadas entre cada par de processos, gerando um arquivo com a matriz de comunicação entre os processos. Nessa matriz, cada posição representa o total de *bytes* trocados entre o par de processos dos eixos da matriz. A Tabela 2.1 ilustra a matriz de comunicação gerada para uma execução de classe C do *benchmark* LU do NAS/NPB-3.3-MPI (NASA, 2011) com 8 processos.

Tabela 4.1: Matriz de comunicação para o total de *bytes* enviados no *benchmark* LU com 8 processos.

	0	1	2	3	4	5	6	7
0	172	197575692	172	172	99610892	172	172	172
1	197576952	120	197575640	120	120	100414040	120	120
2	120	197576952	120	197575640	120	120	97964920	120
3	120	120	197576952	120	120	120	120	97161720
4	99611816	120	120	120	120	197575640	120	120
5	120	100414368	120	120	197576952	120	197575640	120
6	120	120	97965240	120	120	197576952	120	197575640
7	120	120	120	97162688	120	120	197576952	120

- a segunda tabela representa a quantidade de mensagens trocadas entre cada processo.

Tabela 4.2: Matriz de comunicação para o número de mensagens trocadas no *benchmark* LU com 8 processos.

	0	1	2	3	4	5	6	7
0	19	40433	19	19	40433	19	19	19
1	40426	10	40424	10	10	40424	10	10
2	10	40426	10	40424	10	10	40424	10
3	10	10	40426	10	10	10	10	40424
4	40426	10	10	10	10	40424	10	10
5	10	40425	10	10	40426	10	40424	10
6	10	10	40425	10	10	40426	10	40424
7	10	10	10	40426	10	10	40426	10

Ao extrair as tabelas de comunicação, constatou-se que para as aplicações paralelas analisadas, quando compiladas para um mesmo tamanho de entrada e um mesmo número de processos, o padrão de comunicação se mantinha constante, independente da arquitetura onde a aplicação executava. Assim, nesses *benchmarks* os processos trocavam a mesma quantidade de *bytes* ou de mensagens sendo executados em um cluster com vários nós, em apenas um nó com diversos processadores ou, até mesmo, em apenas um processador, desde que o número de processos fosse o mesmo.

As tabelas extraídas representando o padrão de comunicação entre os processo das aplicações analisadas foram utilizadas como entrada para todos os algoritmos de mapeamento estático utilizados neste trabalho.

5 EXPERIMENTOS E ANÁLISE DOS RESULTADOS

Inicialmente este Capítulo contém a descrição da arquitetura alvo em que os experimentos foram executados e, em seguida, uma breve introdução a esses experimentos.

Por fim, os resultados obtidos que caracterizam o padrão de comunicação de cada aplicação e o tempo de execução obtido para cada mapeamento calculado são descritos e analisados.

5.1 Ambiente de Execução dos Experimentos

Foram realizados três conjuntos de experimentos:

- o primeiro conjunto utiliza dois processadores *quad-core* Intel Xeon E5405, utilizando tanto a comunicação intra-chip como a comunicação inter-chip. Neste caso, o compartilhamento de cache L2 ocorre entre os núcleos 0-2, 4-6, 1-3, e 5-7;
- o segundo e o terceiro conjuntos utilizam dois nós de processamento de um cluster interconectados por um *switch* Gigabit Ehternet, cada nó com dois processadores *quad-core* Intel Xeon E5405. Esses experimentos exploram também a comunidação inter-nó, além da comunidação intra-chip e inter-chip.

Para verificar os nomes de sistema dados aos núcleos que compartilham *cache* L2 foi utilizado um *micro-benchmarkd ping-pong.* A cada par de núcleos, o *ping-pong* faz o envio de mensagens e mede o tempo em microssegundos. Com esses valores pode-se ver a diferença nos tempos de comunicação entre núcleos no mesmo *chip* (intra-chip) que compartilham L2 e que não compartilham L2, entre núcleos em *chips* diferentes (inter-chip) e entre núcleos em nós diferentes do cluster (inter-nó).

A Figura 5.1 mostra a arquitetura utilizada no segundo e terceiro conjuntos de experimentos. No primeiro conjunto de experimentos foi utilizado apenas um nó da arquitetura representada na Figura.

Cada nó utilizou o sistema operacional Ubuntu 8.04 com kernel versão 2.6.24. Como compiladores foram utilizados o GCC e o Gfortran, ambos com versão 4.4.3.

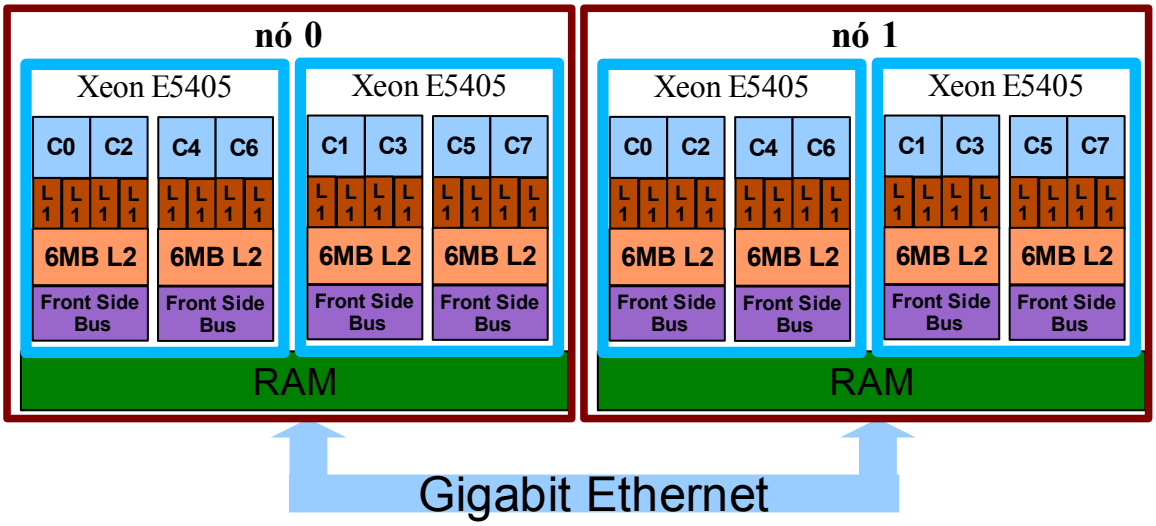


Figura 5.1: Diagrama descrevendo a arquitetura utilizada nos experimentos.

5.2 Metodologia

O objetivo dos experimentos é verificar o ganho de desempenho alcançado pelo mapeamento calculado pela heurística proposta MapEME, que utiliza o EPCM, quando comparado à distribuição padrão feita pela biblioteca MPICH2 *(round-robin)*.

Também são medidos os desempenhos dos mapeamentos calculados pela aplicação Scotch, que utiliza o BRD, e pelo método de BE. O ganho obtido pelo mapeamento resultante da BE representa o ganho máximo, permitindo verificar o quanto a heurística MapEME está próxima do melhor resultado. Já a comparação com a aplicação Scotch é para verificar como a heurística MapEME se sai em relação a uma heurística de mapeamento estático já difundida como a BRD.

Para os experimentos foram utilizados oito *benchmarks* do conjunto NAS-NPB-3.3-MPI (NASA, 2011): BT, CG, IS, LU, MG e SP. Todos foram compilados com classe C, utilizando para sua execução a biblioteca MPICH2-1-3, que, conforme descrito na Seção 2.4, possui o modo de comunicação *nemesis* como padrão. Dentre esses *benchmarks*, existem alguns com padrão de comunicação muito semelhante entre todos os processos paralelos e outros onde alguns processos se comunicam mais.

Como já foi descrito na Seção 4.6, a biblioteca MPE presente em MPICH2 foi alterada para gerar duas matrizes com o volume de comunicação entre os processos: uma referente ao total de *bytes* trocados entre os processos e outra referente ao total de mensagens.

5.2.1 Configurando a Execução dos Processos Paralelos em Núcleos Específicos

Através da função *set_affinity* da biblioteca sched.h, modificou-se os *benchmarks* do NAS-NPB-MPI-3.3 para executar com as afinidades calculadas pelos métodos de mapeamento utilizados nos experimentos. Essas modificações dos *benchmarks* consistiram na inclusão de duas linhas para chamada de funções em cada código alterado, combinada com a compilação de um arquivo a mais onde estavam descritas as

funções chamadas. Apesar da biblioteca sched.h ser codificada em C, ela foi utilizada nos *benchmark* NAS-NPB-MPI-3.3 escritos em Fortran, pois a biblioteca foi compilada utilizando o compilador da linguagem C e "linkada" com o *benchmark* utilizando o compilador Fortran.

Abaixo é possível ver as duas linhas alteradas nos *benchmarks*, que capturam o argumento passado como parâmetro e chamam a função que realiza o mapeamento com esse argumento.

```
1. call get_argument(this+1, parameter)
2. call set_my_affinity(parameter)
```

Em um cluster, cada nó possui a sua própria numeração para os núcleos de processamento, como mostra a Figura 5.1. Assim, a numeração passada para a função *set_affinity* para definir em qual núcleo o processo deve executar só é válida dentro de um mesmo nó. Para realizar o mapeamento de processos para núcleos específicos em nós diferentes foi necessário combinar a distribuição dos processos para os nós realizada pelo arquivo HOST, fornecido como parâmetro para a função *mpirun* da biblioteca MPICH2, com a distribuição dos processos para os núcleos dentro de um mesmo nó, passada por linha de comando ao iniciar a execução do *benchmark*. Essa combinação foi feita através da criação de um *script* que recebe como entrada o mapeamento desejado como uma sequência de dígitos únicos e faz a tradução dessa sequência de dígitos para os respectivos nós e núcleos, configurando o arquivo HOST e invocando o *benchmark* com o parâmetro correto na linha de comando.

5.2.2 Descrição dos Experimentos

Foram realizados três conjuntos de experimentos, todos executados com o mapeamento utilizando a MapEME e com o mapeamento utilizando Scotch e BE para a comparação. Todos os 3 métodos de mapeamento tiveram o seus ganhos calculados sobre a distribuição padrão feita pela biblioteca MPICH2:

1. Experimentos com aplicações paralelas de 8 processos, cada um distribuído para um núcleo de dois processadores *quad-core*, onde também foram considerados a quantidade de *bytes* (representado por 8_b) e de mensagens (representados por 8_m) trocadas. Nesse caso são consideradas a comunicação intra-chip e inter-chip.
2. Experimentos com aplicações paralelas de 16 processos, cada um distribuídos para um núcleo de dois nós com dois processadores *quad-core* cada. Também foram analisados os mapeamentos baseados na quantidade de *bytes* trocados (representados por 16_b) entre os processos e na quantidade de mensagens trocadas (representados por 16_m). Aqui são considerados os três níveis de comunicação: intra-chip, inter-chip e inter-nó.
3. O terceiro conjunto de experimentos é semelhando aos experimentos feitos no item 2, a única diferença é que o mapeamento para os núcleos foi desconsiderado, ou seja, os processos foram distribuídos apenas para os nós e processadores, sendo que sua distribuição para os núcleos dentro dos processadores foi ignorada. O objetivo deste experimento é comparar o quanto o mapeamento intra-chip está contribuindo para o ganho alcançado no mapeamento com 16 núcleos, através da diferença de ganho obtido no experimento do item 2 e no experimento deste item. Os tempos de execução

para esse conjunto de experimentos são identificados pelas siglas 16_b_n e 16_m_n para *bytes* e mensagens respectivamente. Para representar a diferença entre os ganhos dos experimentos 2 e 3 utiliza-se o prefixo "diff_" resultando em diff_16_b_n e diff_16_m_n.

Os experimentos foram executados 100 vezes cada um, calculando médias de tempo cujos intervalos de confiança de 95% de probabilidade estão descritos nos resultados.

5.3 Resultados

Como já dito, para a avaliação dos métodos de mapeamento foram utilizados os *benchmarks* do conjunto NAS-NPB3-MPI (NASA, 2011). Alguns *benchmarks* do conjunto possuem padrões de comunicação semelhantes e, por esse motivo, foram analisados em conjunto. *Benchmarks* com padrões de comunicação distintos foram analisados separadamente.

Os resultados para os *benchmarks* EP e FT não são apresentados, pois devido ao seu padrão de comunicação idêntico entre todos os processos não existem processos que se comunicam mais, o que impede que esses *benchmarks* se beneficiem de qualquer uma das técnicas de mapeamento analisadas.

5.3.1 *Benchmarks* BT, SP e LU

Os *benchmarks* BT, SP e LU buscam solucionar o mesmo problema, simulação de dinâmica de fluídos cuja principal operação é a resolução de equações tridimensionais, mas cada um dos *benchmarks* utiliza um método diferente para solucionar o problema.

Os *benchmaks* BT e SP não foram executados com 8 processos, apenas com 16, porque há uma restrição que diz que esses *benchmarks* só podem ser compilados para um número de processos que seja o quadrado de um número inteiro. Isso se aplica para 16 processos, mas não para 8.

Benckmark BT

O BT (*Block Tridiagonal solver*) busca resolver as equações utilizando uma matriz tridimensional densa. No tamanho C, como ele foi compilado para este trabalho, o BT trabalha sobre uma matriz de dimensões 162x162x162 realizando 200 iterações.

Como pode ser visto na Figura 5.2, no *benchmark* BT com 16 processos alguns processos trocam mais dados entre sí do que outros. Por exemplo, o processo 0 (zero) se comunica com os processos 1, 3, 4, 7, 12 e 13 com 2,08% do total de 77.888 mensagens enviadas (Figura 5.2(b)) e com apenas 0,01% com os demais processos. Em relação aos *bytes* enviados, percebe-se que alguns processos enviam mais *bytes* do que recebem. Como por exemplo o processo 8 (Figura 5.2(a)), que envia para o processo 9 o equivalente a 1,52% do total de 10.628.619.072 *bytes* comunicados e recebe deste mesmo processos apenas 0,57% dos *bytes* comunicados, o que soma 2,09% desse total.

Conforme o gráfico apresentado na Figura 5.3, para todos os métodos de mapeamento o BT apresentou um ganho maior considerando a quantidade de *bytes* comunicados do que a quantidade de mensagens. Isso ocorre devido à distribuição menos uniforme da quantidade de *bytes* que permite um melhor agrupamento dos processos do que a distribuição das mensagens.

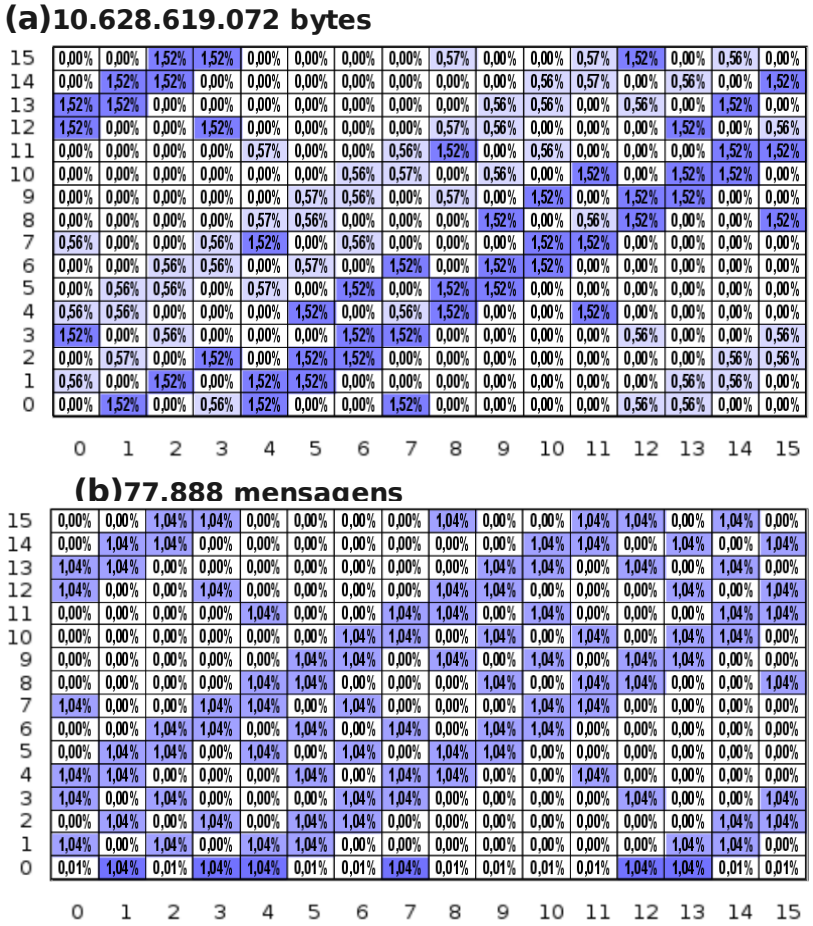


Figura 5.2: Comunicação trocada entre os processos identificados pelas linhas e colunas das matrizes, quanto mais escuro, maior a porcentagem: a) e b) quantidade de *bytes* e mensagens, respectivamente, trocadas em BT com 16 processos.

Observa-se que o ganho total da heurística MapEME (16_b e 16_m) ficou próximo do ganho ótimo obtido pela BE tanto para *bytes* como para mensagens, o que mostra o bom potencial da MapEME. Entretanto, quando comparada a heurística da aplicação Scotch, a MapEME apresentou um ganho menor, mesmo que próximo.

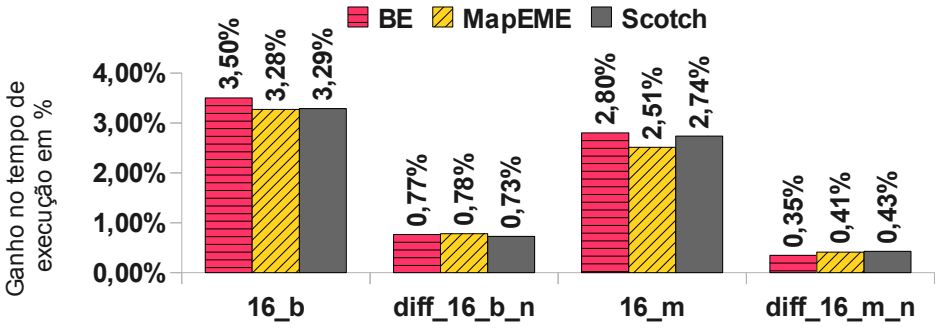


Figura 5.3: Porcentagem de tempo de execução ganho apresentado por cada método em relação à distribuição padrão do MPICH2 no *benchmark* BT.

O ganho atribuído ao mapeamento intra-chip está representado no gráfico da Figura 5.3 por diff_16_b_n e diff_16_m_n em termos de porcentagem de tempo de execução

ganho em relação ao tempo total da distribuição padrão MPICH2. Nesse caso, a MapEME alcançou um ganho maior do mapeamento intra-chip para *bytes* mas não para mensagens.

Já na Tabela 5.1, o ganho está representado em termos de sua porcentagem em relação ao ganho obtido pelo respectivo método quando todos os níveis de comunicação são considerados no mapeamento. Desta forma, é possível notar que, apesar de apresentar um ganho menor que a aplicação Scotch, o ganho obtido pela MapEME tem a maior contribuição do mapeamento intra-chip do que os demais métodos, inclusive maior do que a BE. Sendo que para a MapEME a contribuição intra-chip para o ganho é de 23,96% para *bytes* e 16,4% para mensagens, contra 22,23% e 15,66% da Scotch e 21,87% e 12,42% para a BE, conforme Tabela 5.1.

Tabela 5.1: Proporção do ganho atribuído ao mapeamento intra-chip em relação ao ganho total de cada método para o BT.

BT	BE	MapEME	Scotch
bytes	21,87%	23,96%	22,23%
msgs	12,42%	16,40%	15,66%

Tabela 5.2: Tempos de execução em segundos para *benchmark* BT.

BT		Padrão	Interv. Conf 95%		BE	Interv. Conf 95%		MapEME	Interv. Conf 95%		Scotch	Interv. Conf 95%	
bytes	16_b	373,37	373,04	373,69	360,29	359,68	360,9	361,14	360,87	361,4	361,09	360,79	361,4
	16_b_n	373,37	373,04	373,69	363,5	362,94	364,06	364,07	363,86	364,28	363,82	363,53	364,11
msg	16_m	373,37	373,04	373,69	362,9	362,58	363,22	363,98	341,98	385,98	363,15	362,59	363,71
	16_m_n	373,37	373,04	373,69	364,2	363,92	364,48	365,52	364,98	366,06	364,75	364,42	365,08

A Tabela 5.2 apresenta os tempos de execução para o *benchmark* BT para a distribuição padrão do MPICH2 (Padrão), BE, MapEME e Scotch, com seus respectivos intervalos de confiança.

Benckmark SP

O SP (*Scalar Pentagonal solver*) resolve as equações tridimensionais utilizando estruturas de dados de grades estruturadas. Seu padrão de comunicação é semelhante ao do *benchmark* BT, mas no SP existe quase o dobro de comunicação. Para os experimentos, o SP foi compilado com tamanho C, onde trabalha com uma matriz 162x162x162 executando 400 iterações.

Assim como no BT, na comunicação do *benchmark* SP existem processos que se comunicam mais. E também existe uma heterogeneidade maior na comunicação medida por *bytes* do que na comunicação medida por mensagens. Ao medir por mensagens, todos os processos que se comunicam entre si sempre o fazem com 2,08% do total de 154.656 mensagens trocadas. Apesar do número de mensagens ser quase o dobro do número de mensagens no *benchmark* BT, a distribuição das mensagens entre os processos é muito semelhante. Ao somar o número de *bytes* recebidos e enviados pelos processos, o processo 8 troca com o processo 9 um total de 2,11% do total de 19.433.045.184 *bytes*, enquanto o mesmo processo 8 troca apenas 2,08% do total de *bytes* com o processos 5. Essa maior variedade na quantidade de *bytes* trocados e a maior quantidade de mensagens enviadas contribui para alcançar um ganho maior no mapeamento para os dois casos quando comparado ao BT.

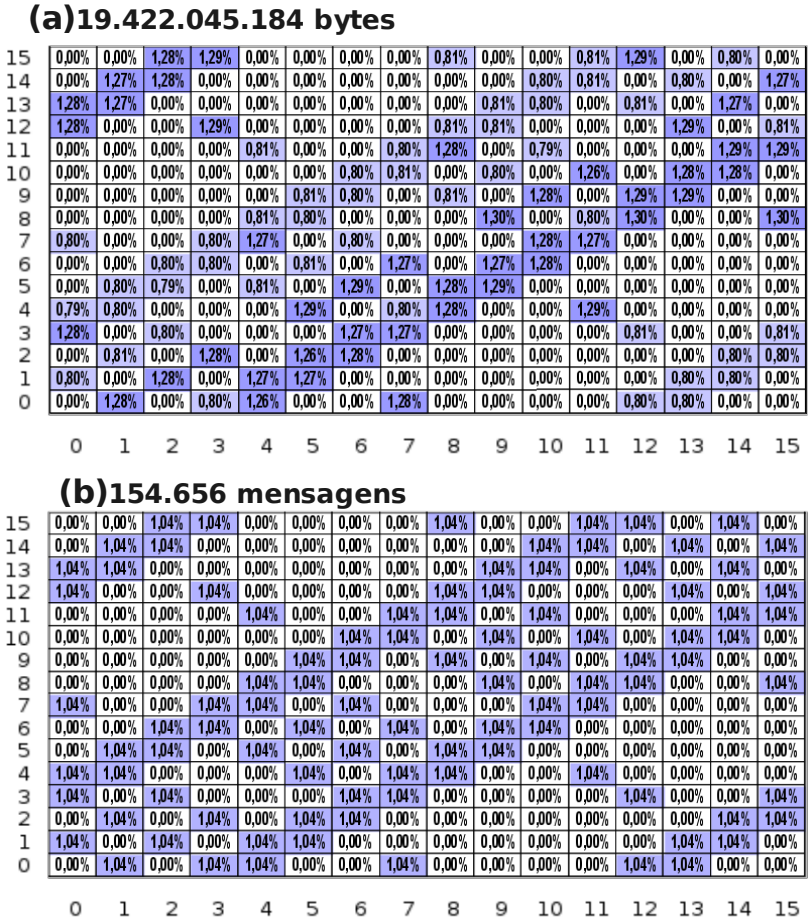


Figura 5.4: Porcentagem de comunicação entre os processos do *benchmark* SP. a) e b) a quantidade de *bytes* e mensagens trocadas em SP com 16 processos.

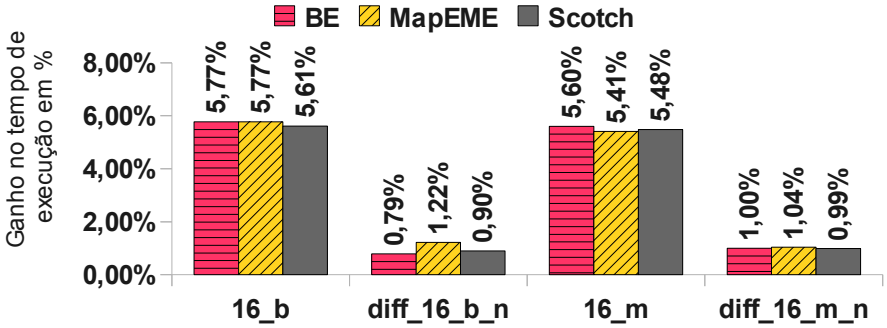


Figura 5.5: Porcentagem de tempo de execução ganho apresentado por cada método em relação à distribuição padrão do MPICH2 no *benchmark* SP.

Como pode ser visto no gráfico da Figura 5.5, no *benchmark* SP também houve um ganho maior quando considerada a quantidade de *bytes* trocados. E, comparado ao BT houve um ganho maior em todos os mapeamentos. Isso porque no SP a quantidade de comunicação é maior, tendo uma maior influência no tempo total de execução da aplicação. Nesse caso, a heurística MapEME se saiu melhor para a comunicação medida com *bytes* do que a aplicação Scotch. E tanto para mensagens como para *bytes* a contribuição do mapeamento intra-chip foi maior para a MapEME, tanto em relação ao

ganho no tempo total da aplicação representado no gráfico, como na proporção para o ganho obtido pelo respectivo método, representado na Tabela 5.3.

Tabela 5.3: Proporção do ganho atribuído ao mapeamento intra-chip em relação ao ganho total de cada método para o SP.

SP	BE	MapEME	Scotch
bytes	13,62%	21,19%	16,12%
msgs	17,82%	19,16%	18,08%

A tabela 5.4 apresenta o tempo de execução em segundo para todos os experimentos realizados com o *benchmark* SP com seus respectivos intervalos de confiança.

Tabela 5.4: Tempo de execução em segundos para o SP.

SP		Padrão	Interv. Conf 95%		BE	Interv. Conf 95%		EPCM	Interv. Conf 95%		BRD	Interv. Conf 95%	
bytes	16_b	521,71	520,13	523,3	491,6	490,92	492,27	491,6	491,13	492,67	492,43	491,85	493,01
	16_b_n	521,71	520,13	523,3	495,7	495,19	496,21	497,98	497,6	498,36	497,15	496,84	497,46
msg	16_m	521,71	520,13	523,3	492,47	491,63	493,31	493,47	492,43	492,43	493,12	492,19	494,05
	16_m_n	521,71	520,13	523,3	497,68	497,17	498,19	498,88	499,46	499,46	498,29	497,81	498,77

Benckmark LU

O LU (*Lower-Upper decomposition*) executa a simulação de fluídos resolvendo as equações através da fatoração LU do sistema de equações em um bloco triangular superior e um bloco triangular inferior. Com o tamanho C, o LU, assim como o BT e SP, também trabalha com uma matriz 162x162x162, mas executa 250 iterações.

O LU não possui a restrição de poder ser compilado apenas para um número de processos que possua uma raiz quadrada inteira. Desta forma, para o *benchmark* LU também foi executado o experimento com 8 processos.

Na Figura 5.6 é possível ver que a comunicação medida por *bytes* no *benchmark* LU é menos uniforme para 8 e 16 processos do que a comunicação medida por mensagens. Assim, o cálculo dos métodos de mapeamento se saem melhor para *bytes* do que para mensagens, conforme gráfico da Figura 5.7. Apesar de trocar um número de *bytes* (4.741.859.776) menor do que os *benchmarks* BT e SP, o fato de a comunicação entre os 16 processos ser mais individualizada, onde cada processo se comunica com um número menor de outros processos, permite um agrupamento melhor feito pelos métodos de mapeamentos analisados, por esse motivo o ganho é maior para o *benchmark* LU, conforme pode ser visto no gráfico da Figura 5.7. O fato também de apresentar um número de mensagens maior do que os *benchmarks* BT e SP, o LU troca 1.942.612 mensagens, mostra que ao trocar uma quantidade de *bytes* menor com um número de mensagens maior leva a uma média de tamanho das mensagens menor de 2.440 *bytes*, o que provoca menor contenção da cache ao mapear para o mesmo chip processos que se comunicam mais.

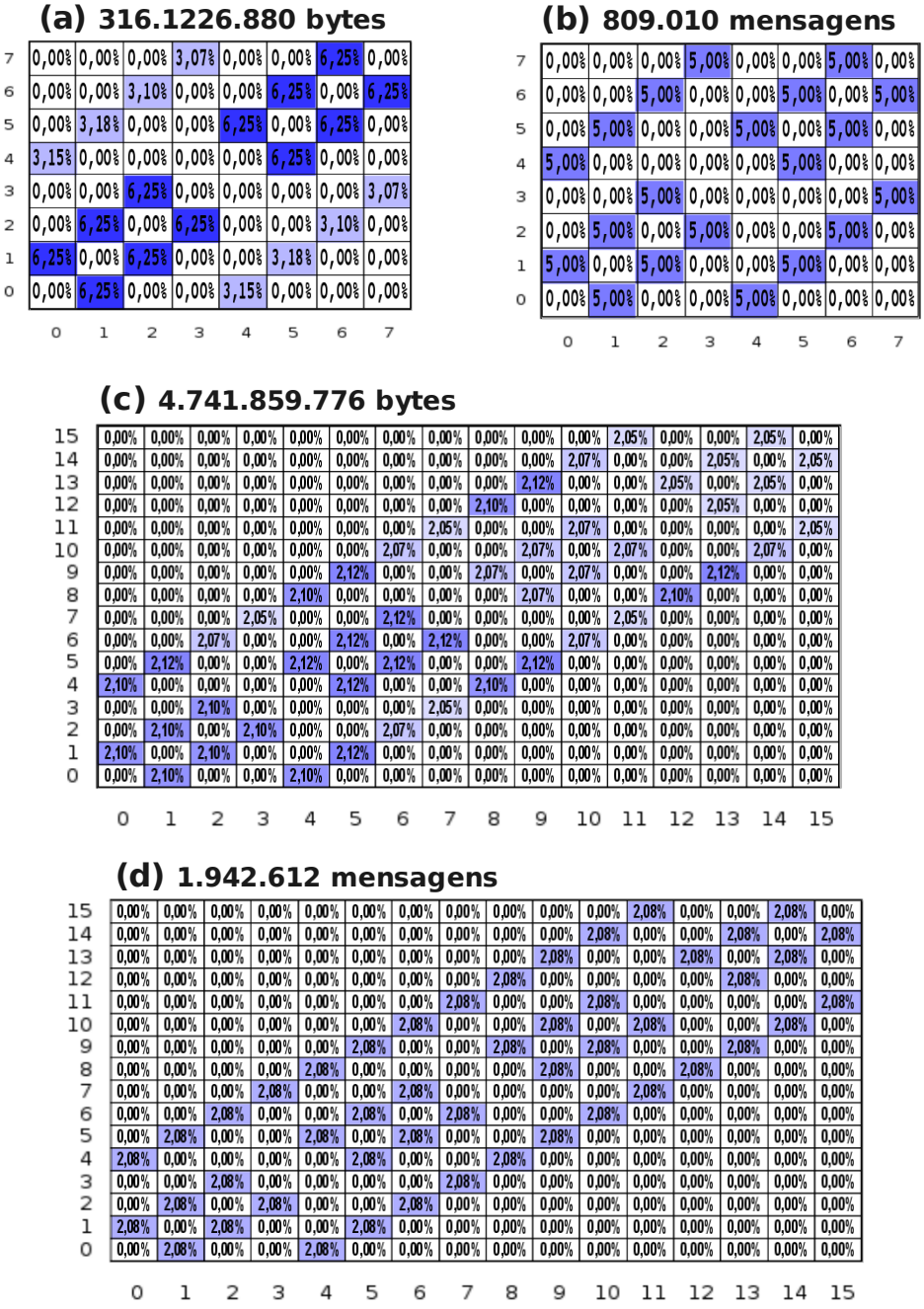


Figura 5.6: Porcentagem de comunicação entre os processos do *benchmark* LU. a) e b) quantidade de *bytes* e mensagens trocadas em LU com 8 processos; c) e d) quantidade de *bytes* e mensagens trocadas em LU com 16 processos.

Analisando o gráfico da Figura 5.7, nota-se que para o *benchmark* LU, o ganho com o mapeamento da heurística MapEME alcança desempenho melhor do que a aplicação Scotch para *bytes*, mas não para mensagens. Entretanto, a contribuição do mapeamento intra-chip é maior tanto para *bytes* como para mensagens. Seguindo o padrão dos demais *benckmarks*, o mapeamento do LU utilizando a heurística MapEME alcançou resultados muito próximos da busca exaustiva, o que prova o seu potencial.

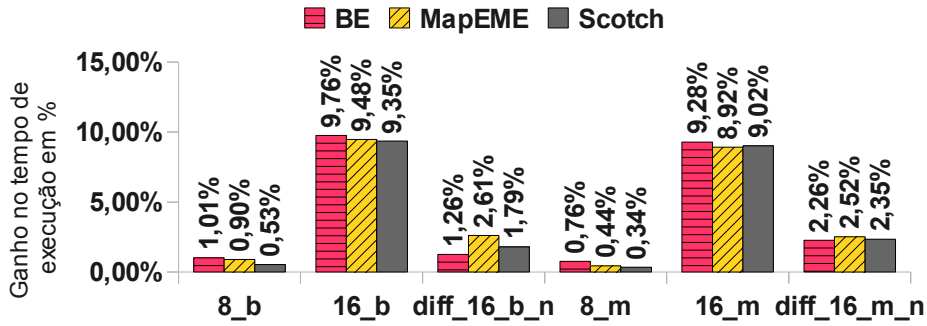


Figura 5.7: Porcentagem de tempo de execução ganho apresentado por cada método em relação à distribuição padrão do MPICH2 no *benchmark* LU.

Na tabela 5.5, nota-se que a contribuição do mapeamento intra-chip em relação ao ganho total atingido também é maior para a MapEME, ultrapassando inclusive o ganho da BE. 27,5% para *bytes* e 28,29% para mensagens são as maiores contribuições da comunicação intra-chip em relação ao ganho.

Tabela 5.5: Proporção do ganho atribuído ao mapeamento intra-chip em relação ao ganho total de cada método para LU.

LU	BE	MapEME	Scotch
bytes	12,88%	27,50%	19,15%
msgs	24,34%	28,29%	26,06%

Os tempos de execução em segundos do *benchmark* LU e seus intervalos de confiança a 95% podem ser vistos na Tabela 5.6.

Tabela 5.6: Tempo de comunicação em segundos para o *benchmark* LU.

LU		Padrão	Interv. Conf 95%		BE	Interv. Conf 95%		EPCM	Interv. Conf 95%		BRD	Interv. Conf 95%	
bytes	8_b	473,7	466,99	480,4	468,91	463,26	474,56	469,45	463,6	475,31	471,2	465,02	477,38
	16_b	239,46	239,02	239,89	216,09	215,7	216,48	216,77	216,46	217,08	217,06	216,84	217,28
	16_b_n	239,46	239,02	239,89	219,1	218,7	219,5	223,01	222,62	223,4	221,35	220,73	221,97
msg	8_m	473,7	466,99	480,4	470,1	465,78	474,42	471,6	467,8	475,4	472,1	467,18	477,02
	16_m	239,46	239,02	239,89	217,23	216,56	217,9	218,11	217,56	218,66	217,86	216,97	218,75
	16_m_n	239,46	239,02	239,89	222,64	221,99	223,29	224,15	223,64	224,66	223,49	223,17	223,81

5.3.2 *Benchmarks* CG

O *benchmark* CG (*Conjugate Gradient*) resolve um sistema de equações lineares esparso, Az=x, através de um método do gradiente conjugado. Ele estima o maior autovalor de uma matriz esparsa simétrica positiva através do métodos de potência inversa. Para os experimentos executados, o CG utilizou o tamanho C, onde o sistema de equações a ser resolvido tem tamanho 150.000 e são executadas 75 iterações.

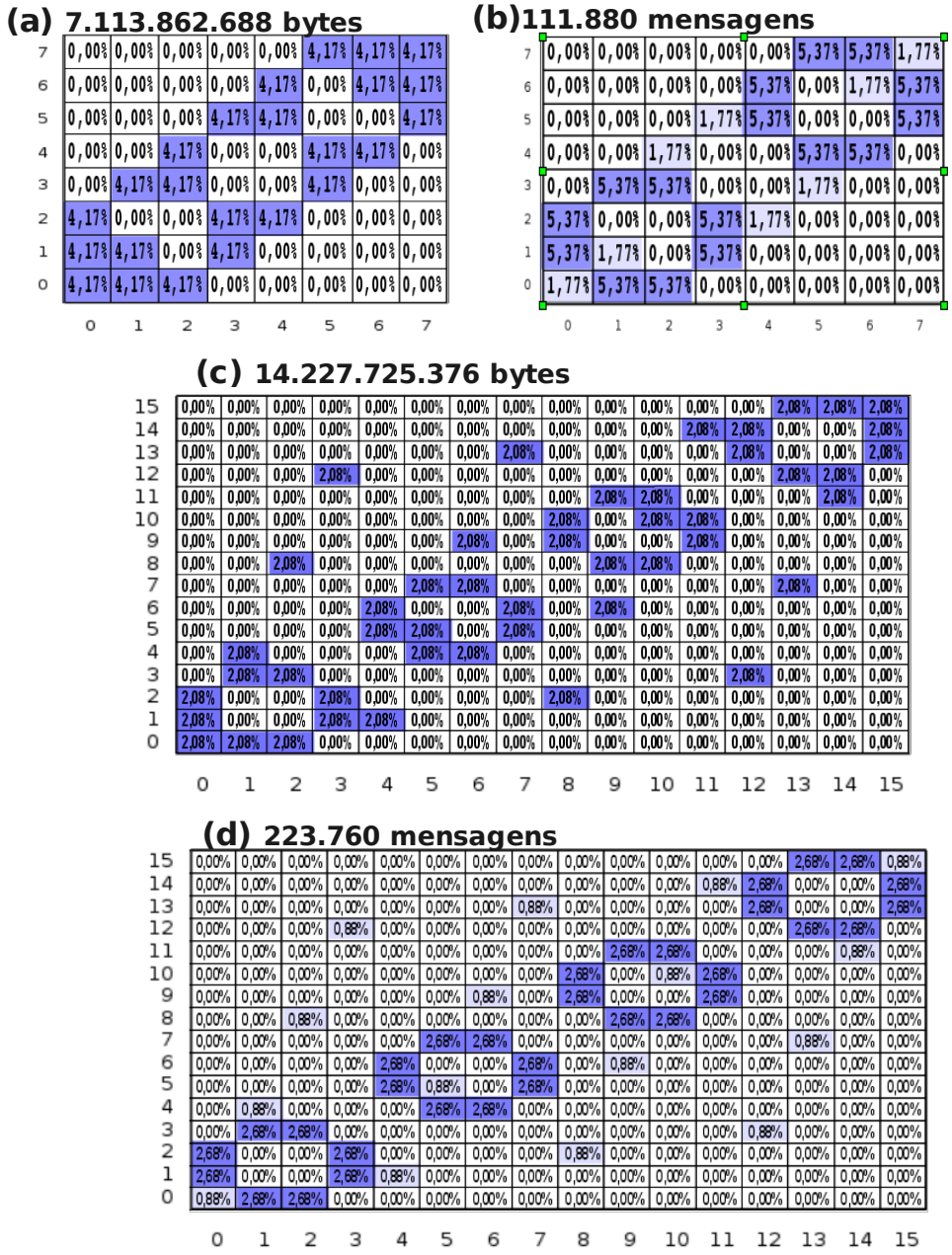


Figura 5.8: Porcentagem de comunicação entre os processos do *benchmark* CG. a) e b) quantidade de *bytes* e mensagens trocadas em CG com 8 processos; c) e d) quantidade de *bytes* e mensagens trocadas em CG com 16 processos.

No CG, a comunicação medida por número de mensagens é menos uniforme do que a comunicação medida pela quantidade de *bytes* tanto para 8 como para 16 processos, conforme pode ser visto na Figura 5.8. Isso faz com que, conforme o gráfico da Figura 5.9, o mapeamento calculado com base no número de mensagens trocadas obtenha um maior ganho. O CG é, dentro os *benckmarks* analisados, aquele onde cada processo se comunica com um número menor de outros processos. Esse fato, que permite que os processos sejam mais facilmente agrupados dois a dois, faz com que os algoritmos de mapeamento analisados obtenham o seu melhor desempenho, chegando a 41% para a BE ao contabilizar a quantidade de mensagens para o mapeamento.

Conforme o gráfico da Figura 5.9, no *benchmark* CG, o ganho no tempo de execução obtido pela heurística MapEME é maior que o da aplicação Scotch, exceto para o teste com 8 processos para *bytes*. A contribuição do mapeamento intra-chip na heurística MapEME é maior tanto para mensagens como para *bytes*.

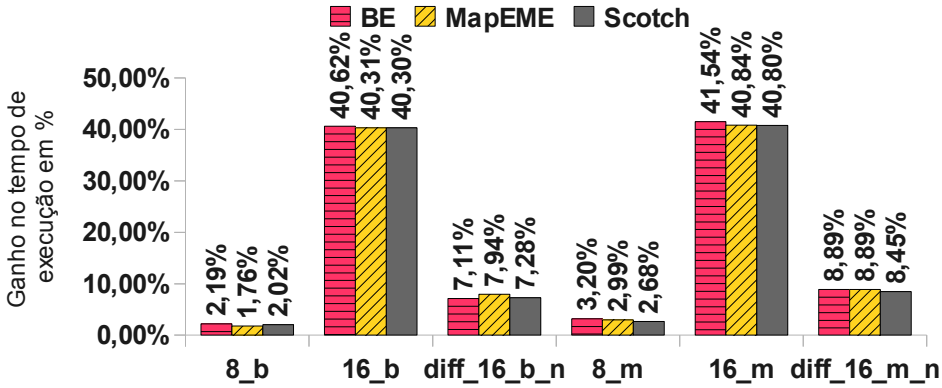


Figura 5.9: Porcentagem de tempo de execução ganho apresentado por cada método em relação à distribuição padrão do MPICH2 no *benchmark* CG.

Apesar de a heurística MapEME ter alcançado uma contribuição maior do mapeamento intra-chip em relação aos outros métodos, conforme a Tabela 5.7, essa contribuição não foi tão grande quanto a do *benckmark* LU. Isso porque é trocada uma quantidade de *bytes* de 14.227.725.376 e de 223.760 mensagens, resultando em uma média de 63.584 *bytes* por mensagens, maior do que a do *benchmark* LU. Esse tamanho maior das mensagens resulta em uma maior contenção da *cache* ao fazer que processos que se comuniquem mais a compartilhem.

Tabela 5.7: Proporção do ganho atribuído ao mapeamento intra-chip em relação ao ganho total de cada método para o CG.

CG	BE	MapEME	Scotch
bytes	17,50%	19,69%	18,05%
msgs	21,39%	21,78%	20,70%

A Tabela 5.8 ilustra os tempos de execução em segundo para o *benchmark* CG e os respectivos intervalos de confiança a 95% de probabilidade.

Tabela 5.8: Tempo de execução em segundo para o *benchmark* CG.

CG		Padrão	Interv. Conf 95%		BE	Interv. Conf 95%		EPCM	Interv. Conf 95%		BRD	Interv. Conf 95%	
bytes	8_b	148,65	148,63	148,66	145,4	145,02	145,78	146,03	145,93	146,13	145,65	145,26	146,05
	16_b	184,42	184,32	184,52	109,5	109,12	109,88	110,08	109,84	110,31	110,09	109,8	110,38
	16_b_n	184,42	184,32	184,52	122,61	122,04	123,18	124,72	124,41	125,03	123,51	123,09	123,93
msg	8_m	148,65	148,63	148,66	143,9	143,53	144,27	144,2	143,77	144,63	144,66	144,38	144,94
	16_m	184,42	184,32	184,52	107,81	107,38	108,24	109,11	108,93	109,29	109,17	108,96	109,38
	16_m_n	184,42	184,32	184,52	124,2	63,2	124,81	125,51	125,08	125,94	124,75	124,36	125,14

5.3.3 *Benchmark* IS

O *benchmark* IS (*Integer Sort*) realiza a ordenação de inteiros pequenos através do algoritmo *bucket sort* (SAPHIR et al., 1997), que faz a ordenação dos números de forma distribuída. Isso permite que o algoritmo seja executado de forma paralela. Com o

tamanho C, utilizado neste trabalho, o IS itera 10 vezes para ordenar 134217728 números inteiros.

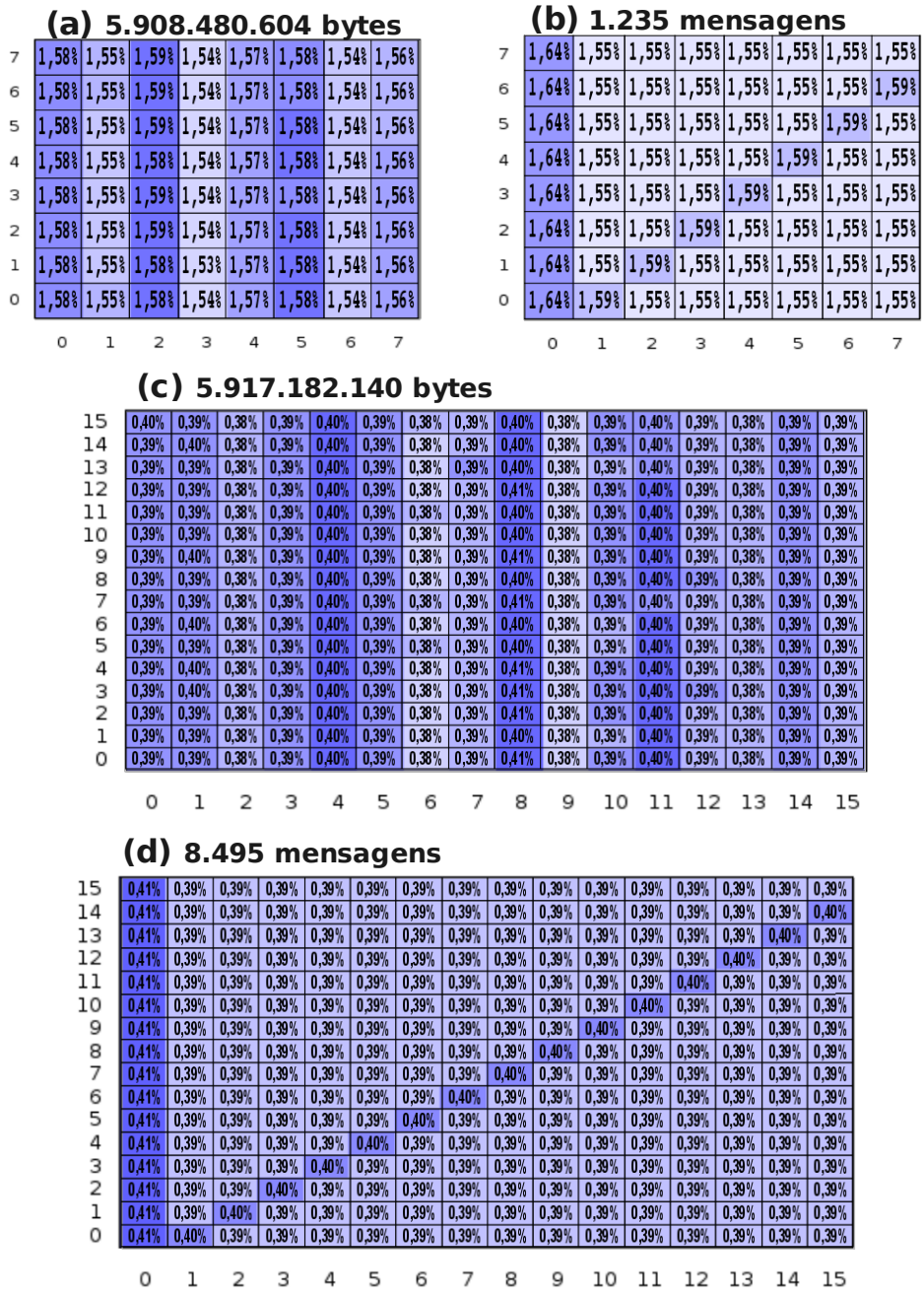


Figura 5.10: Porcentagem de comunicação entre os processos do *benchmark* IS. a) e b) quantidade de *bytes* e mensagens trocadas em IS com 8 processos; c) e d) quantidade de *bytes* e mensagens trocadas em IS com 16 processos.

Conforme a Figura 5.10, o padrão de comunicação observado ao medir a quantidade de *bytes* trocados entre os processos é bastante distinto do padrão observado da quantidade de mensagens. Ao medir os *bytes*, verifica-se que alguns processos recebem mais *bytes* de todos os outros processos, como os processos 0, 2 e 5 para o IS com 8 processos e os processos 4, 8 e 11 para o IS com 16 processos, produzindo um padrão vertical na matriz. Ao analisar a troca de mensagens, no entanto, o processo 0 destaca-se como aquele que recebe mais mensagens de todos os outros processos e há

um padrão transversal na matriz, que indica que todos os processos enviam grande quantidade de mensagens para o processo identificado pelo número acima do seu.

Assim, devido ao padrão menos uniforme de comunicação em *bytes*, ganha-se mais agrupando os processos considerando os *bytes* do que usando a quantidade de mensagens. Isso reflete no maior ganho obtido por todos os métodos na comunicação por *bytes* do que por mensagens, conforme Figura 5.11. Na Figura também é possível notar que no total de tempo ganho, a heurística MapEME apresentou desempenho inferior ao Scotch para 8 e 16 processos considerando *bytes* e mensagens (8_b, 8_m, 16_b e 16_m). Entretanto, ao medir a contribuição do mapeamento intra-chip em relação ao tempo total de execução, observa-se que a MapEME produz um resultado melhor (diff_16_b_n, diff_16_m_n).

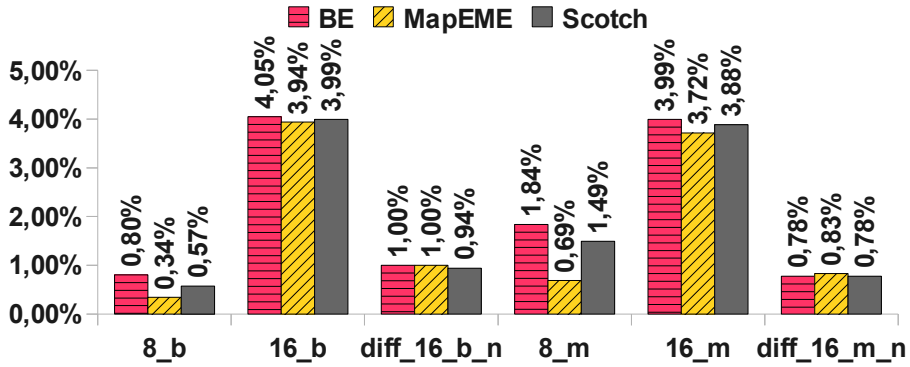


Figura 5.11: Porcentagem de tempo de execução ganho apresentado por cada método em relação à distribuição padrão do MPICH2 no *benchmark* IS.

Em relação à contribuição do mapeamento intra-chip para o ganho produzido por cada método, a MapEME também se saiu melhor, conforme Tabela 5.9. Sendo que 25,35% do ganho obtido pela heurística MapEME na comunicação por *bytes* é atribuído ao mapeamento intra-chip e 22,39% na comunicação por mensagens.

Tabela 5.9: Proporção do ganho atribuído ao mapeamento intra-chip em relação ao ganho total do *benchmark* IS.

IS	BE	MapEME	Scotch
bytes	24,66%	25,35%	23,61%
msgs	19,44%	22,39%	20,00%

Os tempos médios de execução em segundos para cada experimento e os respectivos intervalos de confiança a 95% de probabilidade estão ilustrados na Tabela 5.10.

Tabela 5.10: Tempo de execução em segundo para o *benchmark* IS.

IS		Padrão	Interv. Conf 95%		BE	Interv. Conf 95%		EPCM	Interv. Conf 95%		BRD	Interv. Conf 95%	
bytes	8_b	8,7	8,69	8,71	8,63	8,59	8,67	8,67	8,64	8,71	8,65	8,63	8,68
	16_b	18,03	17,89	18,16	17,3	17,2	17,4	17,32	17,2	17,44	17,31	17,2	17,43
	16_b_n	18,03	17,89	18,16	17,48	17,31	17,65	17,5	17,29	17,71	17,48	17,35	17,61
msg	8_m	8,7	8,69	8,71	8,54	8,44	8,64	8,64	8,48	8,8	8,57	8,45	8,69
	16_m	18,03	17,89	18,16	17,31	17,21	17,41	17,36	17,22	17,5	17,33	17,24	17,42
	16_m_n	18,03	17,89	18,16	17,45	16,93	17,97	17,51	16,82	16,13	17,47	17,04	17,9

5.3.4 *Benchmark* MG

O *benchmark* MG (*MultiGrid)* aproxima a solução de uma equação de Poisson tridimensional discreta utilizando o método multigrid de ciclo V. No tamanho C, o algoritmo itera 20 vezes sobre uma matriz de dimensões 512x512x512.

Conforme a Figura 5.12, a comunicação tanto para 8 como para 16 processos, considerando o número de mensagens ou *bytes* trocados é pouco uniforme. Isso contribui para que o MG esteja entre os *benchmarks* analisados que obtiveram os maiores ganhos no mapeamento, junto com o CG e o LU.

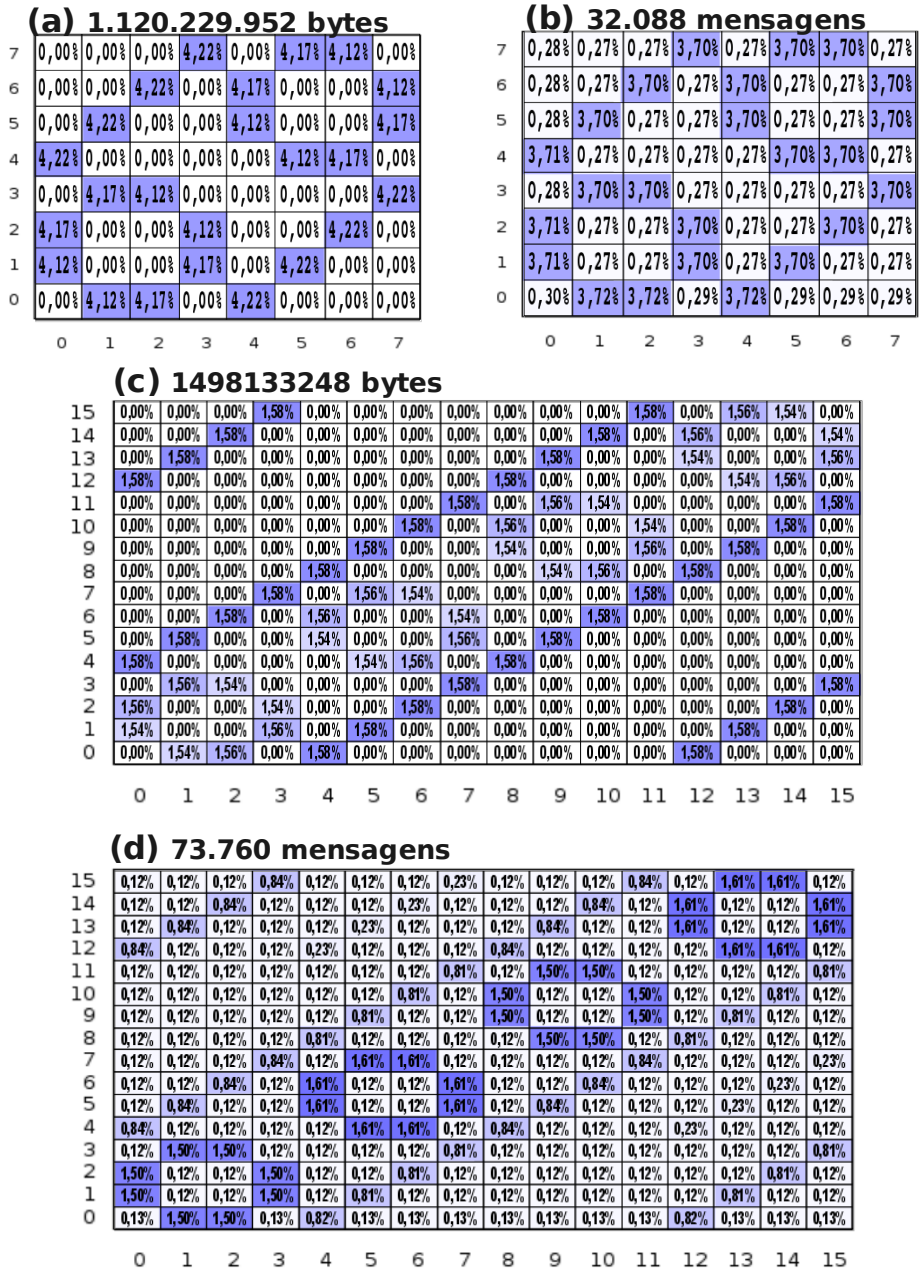


Figura 5.12: Porcentagem de comunicação entre os processos do *benchmark* MG. a) e b) quantidade de *bytes* e mensagens trocadas em MG com 8 processos; c) e d) quantidade de *bytes* e mensagens trocadas em MG com 16 processos.

O mapeamento baseado no número de mensagens trocadas apresenta maior ganho no tempo de comunicação em quase todos os experimentos com o MG, quando comparado ao mapeamento que considera a quantidade de *bytes*, conforme Figura 5.13. Isso ocorre devido a menor uniformidade na comunicação por mensagens, sendo mais fácil agrupar os processos que se comunicam mais nesse caso.

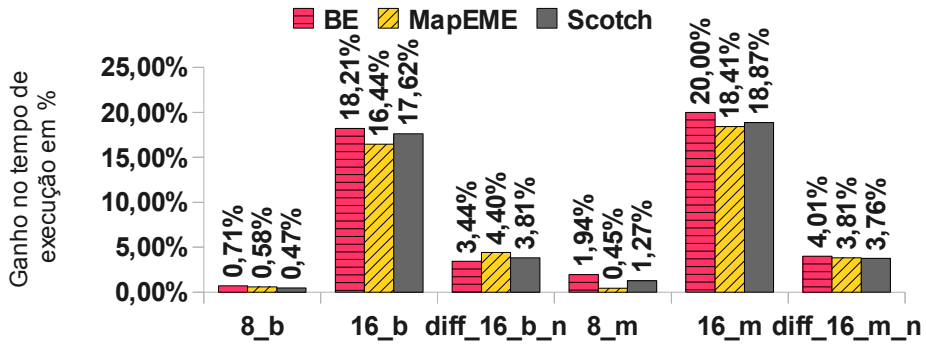


Figura 5.13: Porcentagem de tempo de execução ganho apresentado por cada método em relação à distribuição padrão do MPICH2 no *benchmark* MG.

Assim como no IS, no MG a heurística MapEME apresenta um desempenho inferior ao da aplicação Scoth quando considerado o ganho total no tempo de execução do *benchmark*, mas com uma contribuição maior no mapeamento intra-chip, inclusive na proporção do ganho total atribuído ao método, apresentada na Tabela 5.11.

Tabela 5.11: Proporção do ganho atribuído ao mapeamento intra-chip em relação ao ganho total de cada método para o MG.

MG	BE	MapEME	Scotch
bytes	18,89%	26,76%	21,62%
msgs	20,02%	20,69%	19,92%

A Tabela 5.12 apresenta os tempos médios de execução em segundos dos experimentos executados. Também é apresentado o intervalo de confiança com 95% de probabilidade.

Tabela 5.12: Tempo de execução em segundo para o *benchmark* IS.

MG		Padrão	Interv. Conf 95%		BE	Interv. Conf 95%		EPCM	Interv. Conf 95%		BRD	Interv. Conf 95%	
bytes	8_b	55,27	55,21	55,32	54,88	54,81	54,95	54,95	54,89	55	55,01	54,91	55,12
	16_b	40,69	40,58	40,8	33,28	33,26	33,3	34	33,63	34,37	33,52	33,21	33,82
	16_b_n	40,69	40,58	40,8	34,68	34,47	34,89	35,79	35,52	36,06	35,07	34,75	35,39
msg	8_m	55,27	55,21	55,32	54,2	53,55	54,85	55,02	54,56	55,48	54,57	54,03	55,11
	16_m	40,69	40,58	40,8	32,55	32,23	32,87	33,2	32,81	33,59	33,01	32,73	33,29
	16_m_n	40,69	40,58	40,8	34,18	33,31	35,05	34,75	33,83	35,67	34,54	33,75	35,33

5.4 Considerações Finais

Analisando em conjunto os resultados obtidos para cada *benchmark* fica mais fácil identificar quais são as característica comuns do comportamento da MapEME.

Como pode ser visto no gráfico da Figura 5.14 (FERREIRA et al. 2011) (FERREIRA; CRUZ; NAVAUX, 2011), para os *benchmarks* onde foram executados experimentos com 8 e 16 núcleos, sendo eles LU, CG, IS e MG, o mapeamento calculado para 16 núcleos alcançou um ganho maior do que aquele calculado para 8 núcleos. O mapeamento com 16 núcleos considera a comunicação inter-nó, que é aquela que demora mais tempo. Já o mapeamento com 8 núcleos considera apenas a comunicação inter-chip e intra-chip. Ao calcular o mapeamento com 16 núcleos buscando diminuir o tempo de comunicação ganha-se muito mais minimizando a comunicação inter-nó do que a comunicação inter-chip. Por esse motivo o ganho com o mapeamento em 16 núcleos nos experimentos é sempre maior.

Observa-se também que a heurística MapEME apresentou um ganho próximo ao da BE em todos os experimentos, o que também ocorreu para a heurística BRD utilizada pela aplicação Scoth. Isso demostra o benefício em aplicar as heurísticas para calcular o mapeamento ao invés de utilizar a BE, pois as heurísticas possuem uma complexidade menor e conseguem alcançar ganhos no tempo de execução próximos ao ótimo.

A MapEME obteve um ganho total maior quanto comparado aos ganhos da aplicação Scotch para os *benchmarks* SP 16_b; LU 8_b, 16_b e 8_m; CG 16_b, 8_m e 16_m; e MG 8_b. Apesar de não ter superado o ganho da aplicação Scotch na maioria dos experimentos, para o ganho atribuído apenas ao mapeamento intra-chip o resultado é diferente, como será visto na Figura 5.15.

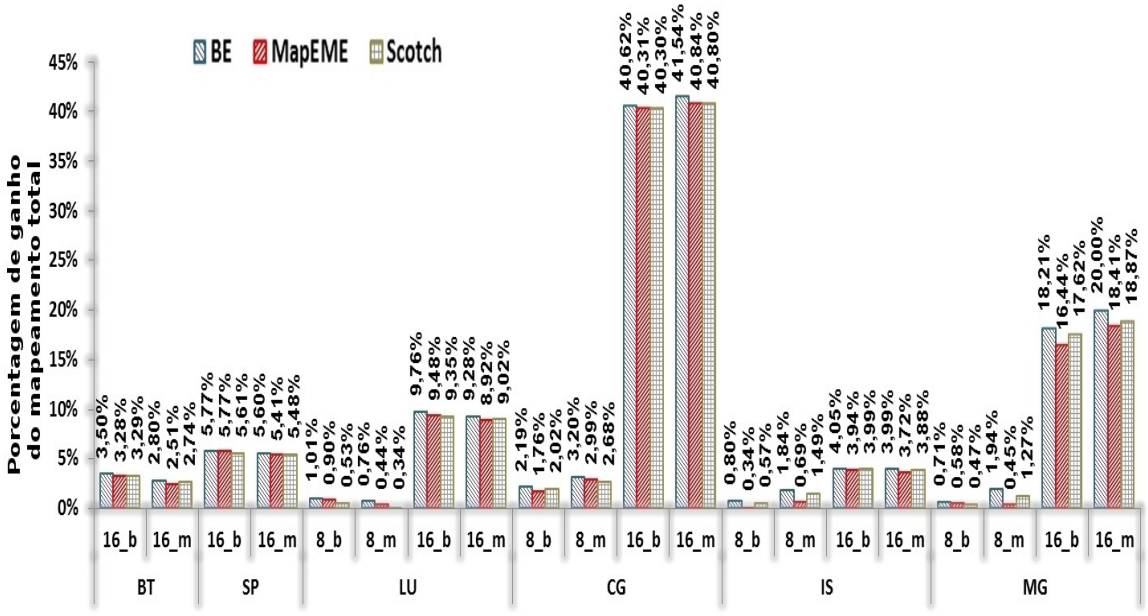
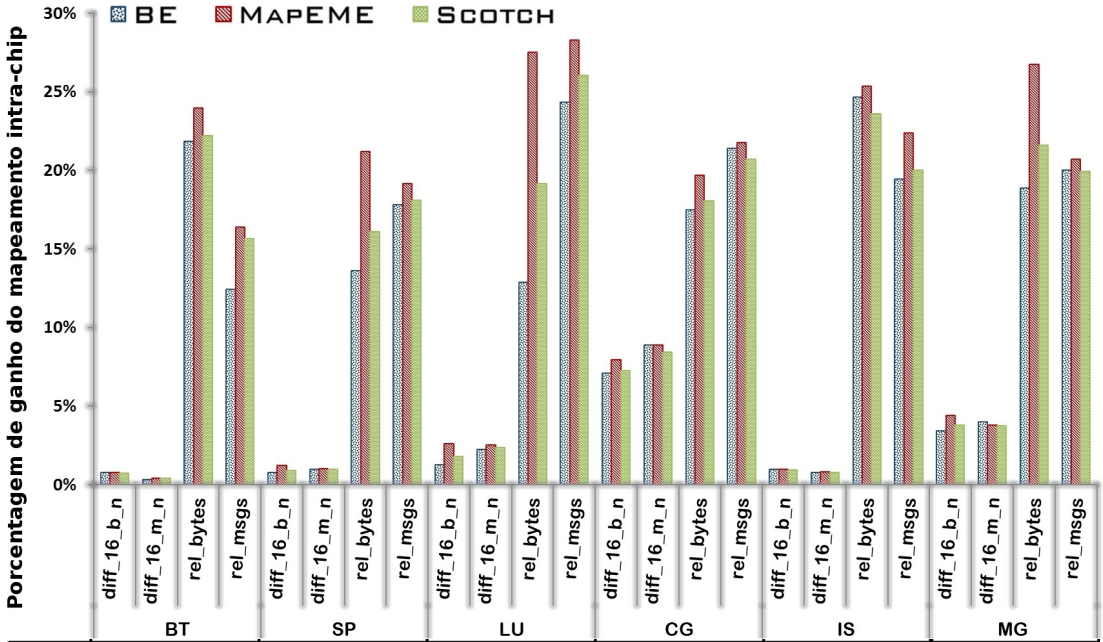


Figura 5.14: Ganho de cada mapeamento em relação ao tempo da distribuição padrão com 8 e 16 processos.

Observa-se também que o mapeamento considerando *bytes* teve um ganho maior para o BT, o SP, o LU e o IS. Isso porque, nesses *benchmarks* a comunicação por *bytes* é menos uniforme que a comunicação por mensagens. Nos *benchmarks* CG e MG ocorre o contrário, a comunicação por mensagens tem uma distribuição menos uniforme. O mapeamento visa diminuir o tempo de comunicação entre os processos durante a execução total do *benchmark*, deixando os processos que se comunicam mais mais próximos. Os melhores ganhos devido ao mapeamento são obtidos nos *benchmarks* em que a comunicação ocorre entre poucos núcleos ou em quantidade maior entre poucos núcleos. Desta forma é possível agrupar de forma mais eficiente os processos que se comunicam mais. Se todos os processos se comunicassem com a mesma quantidade de dados, como ocorre com o EP e FT, não seria possível calcular quais processos se comunicam mais, pois não existiriam tais processos.

O gráfico da Figura 5.15 resume o ganho da comunicação intra-chip calculado de duas formas. Primeiro é expressa a diferença entre o ganho apresentado pela execução com mapeamento intra-chip, inter-chip e inter-nó para 16 núcleos e o ganho apresentado pela execução desconsiderando o mapeamento intra-chip, representada por diff_16_b_n e diff_16_m_n. Para todos os casos, exceto diff_m_n para o BT e MG, a heurística MapEME apresenta um ganho atribuído ao mapeamento intra-chip maior que os demais métodos apresentados nesse trabalho.

$$diff\_16\_b\_n = 16\_b - 16\_b\_n \qquad diff\_16\_m\_n = 16\_m - 16\_m\_n$$



	BT				SP				LU				CG				IS				MG			
BE	0,77	0,35	21,8	12,4	0,79	1,00	13,6	17,8	1,26	2,26	12,8	24,3	7,11	8,89	17,5	21,3	1,00	0,78	24,6	19,4	3,44	4,01	18,8	20,0
MapEME	0,78	0,41	23,9	16,4	1,22	1,04	21,1	19,1	2,61	2,52	27,5	28,2	7,94	8,89	19,6	21,7	1,00	0,83	25,3	22,3	4,40	3,81	26,7	20,6
Scotch	0,73	0,43	22,2	15,6	0,90	0,99	16,1	18,0	1,79	2,35	19,1	26,0	7,28	8,45	18,0	20,7	0,94	0,78	23,6	20,0	3,81	3,76	21,6	19,9

Figura 5.15: Ganho do mapeamento dos núcleos em relação ao tempo da distribuição padrão.

Também é apresentada uma outra medida, que é o quanto o mapeamento intra-chip contribui para o ganho de cada método para a execução com 16 núcleos. Essa medida está representada por rel_bytes e rel_msgs:

$$rel\_bytes = diff\_16\_b\_n / 16\_b \qquad rel\_msg = diff\_16\_m\_n / 16\_m$$

Para essa medida, a heurística MapEME apresenta para todos os *benchmarks* analisados uma maior contribuição do mapeamento intra-chip para o ganho do que os demais métodos utilizados. Essa maior contribuição do mapeamento intra-chip deve-se ao fato de a MapEME dividir inicialmente os processo em pares com maior quantidade de comunicação a fim de mapeá-los para núcleos que compartilham a *cache* L2. A aplicação Scotch faz um biparticionamento dos processos, sendo que o último passo é agrupar processos nos pares que serão mapeados para núcleos com *cache* compartilhada. Ao agrupar os processos que irão utilizar a comunicação intra-chip primeiro, a MapEME encontra uma solução melhor no nível intra-chip, mesmo que essa solução não seja a melhor no geral.

Como exemplo, na Figura 5.16 está ilustrado o funcionamento das heurísticas Scotch, em (a), e MapEME, em (b), para o *benchmark* IS com 8 processos e tamanho C, descrito na Seção 5.3.3, considerando a quantidade de *bytes* trocados entre os processos. Ao calcular a quantidade de comunicação dentro de cada grupo de processos, observa-se que para o mapeamento calculado pela aplicação Scotch a quantidade de comunicação interna para os grupos de quatro processos (1.477.393.276) é menor que a quantidade de comunicação interna para os grupos de quatro processos resultantes do mapeamento calculado pela heurística MapEME (1.477.105.771). Mas a quantidade de comunicação interna nos grupos de dois processos resultantes do mapeamento Scotch é maior (738.834.864) que a quantidade de comunicação interna nos grupos de dois processos mapeados pela MapEME (738.998.056). Por calcular um mapeamento em que a quantidade de comunicação maior no nível intra-chip com compartilhamento de *cache* (grupos de dois processos), a MapEME atinge uma maior contribuição deste nível para o ganho alcançado pela heurística em relação à distribuição padrão MPI.

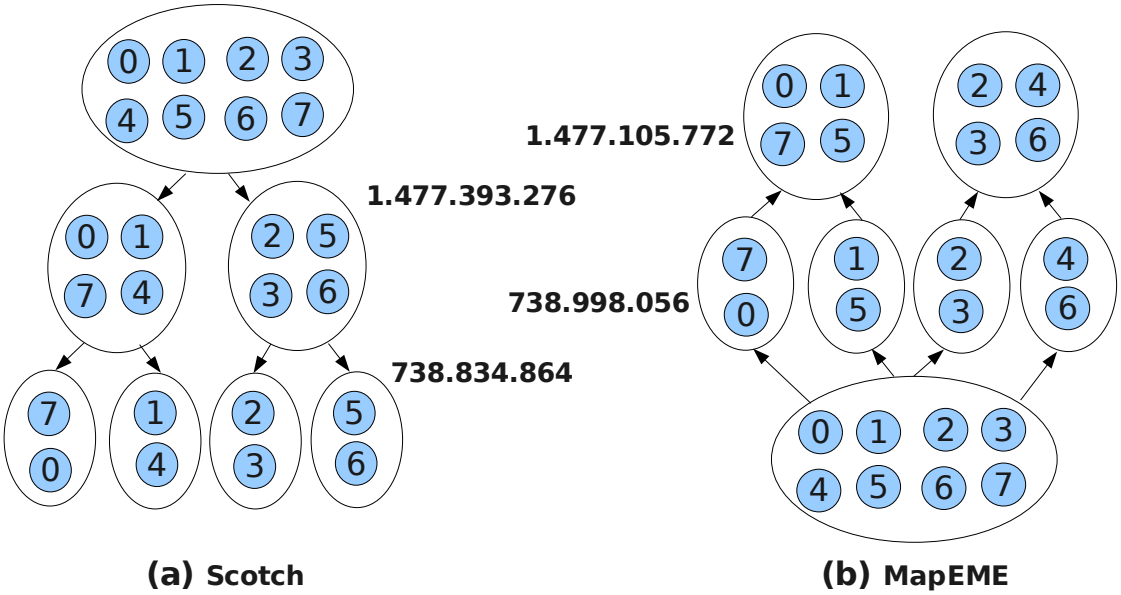


Figura 5.16: Mapeamento do *benchmark* IS com MapEME e Scotch.

6 CONCLUSÃO

Em um cluster de *multi-cores* a comunicação entre os processadores pode ser classificada em pelo menos três níveis: intra-chip, inter-chip e inter-nó, da menor para a maior latência de comunicação. Atualmente a distribuição *default* MPI dos processos não considera esses níveis de comunicação, podendo colocar processos que se comunicam mais em processadores que demoram mais tempo para se comunicar. O mapeamento de processos baseado na comunicação tem o objetivo de distribuir os processos paralelos de forma a aproximar o processos que se comunicam mais diminuindo o tempo de comunicação.

Existem diversos trabalhos tratando de mapeamento estático e dinâmico de processos, mas é naqueles onde o mapeamento estático é empregado em que se alcança o maior ganho no tempo de execução. Os trabalhos que consideram o mapeamento estático de processos geralmente utilizam para calcular o mapeamento um método que tem seu ganho no tempo de execução comparado à distribuição padrão dos processos, mas nunca comparado ao desempenho de outro método de mapeamento.

Esta dissertação propôs e avaliou a MapEME (Mapeamento Estático MPI com Emparelhamento), uma heurística que emprega o Emparelhamento Perfeito de Custo Máximo (EPCM) para calcular o mapeamento de processos MPI em clusters *multi-core*.

A complexidade da MapEME foi comparada a outros dois métodos de mapeamento: BE e à heurística BRD utilizada na aplicação Scotch. A MapEME possui uma complexidade menor que a aplicação Scotch ao considerar sua utilização em aplicações representadas por grafos completos.

$$O(n^3)\ para\ Scotch \qquad O(n^2.lg(n))\ para\ MapEME$$

Nos experimentos, apresentados no Capítulo 5, constatou-se que a contribuição do mapeamento intra-chip para o ganho nos experimentos realizados com a MapEME é sempre superior a dos demais métodos analisados, inclusive em relação à BE.

Essa maior contribuição do mapeamento intra-chip deve-se ao fato de a MapEME calcular o mapeamento com a abordagem de agrupar os processos que se comunicam mais, como foi explicado no Capítulo 4. Desta forma, a MapEME divide inicialmente os processo em pares com maior quantidade de comunicação. Esse processos são mapeados para núcleos de processamento do nível mais baixo de comunicação, ou seja, em núcleos no mesmo chip que aproveitam a comunicação intra-chip através do compartilhamento de *cache* L2.

Ao decidir primeiro quais processos irão compartilhar *cache* L2, a MapEME atribui maior importância a esse nível de comunicação, pois como se trata de um algoritmo

guloso, as decisões tomadas para um nível de comunicação não serão revistas a partir do momento em que se passa para um próximo nível. Ao dar maior relevância à comunicação intra-chip, a MapEME alcança uma maior contribuição para o mapeamento para esse nível de comunicação no ganho geral.

Para o ganho total, a MapEME apresentou resultados melhores quando empregada nos *benchmarks* onde a comunicação ocorre em maior quantidade entre um número menor de processos, embora nem sempre alcance um ganho maior que a aplicação Scotch. Tanto a MapEME quanto a Scotch apresentarem ganhos próximos ao alcançado pela BE.

Assim conclui-se que a MapEME tem um grande potencial como heurística de mapeamento, pois possui uma complexidade menor que a BRD. Além disso, a maior contribuição do mapeamento intra-chip no ganho total sugere que ao aumentar o número de núcleos dentro do chip o mapeamento calculado pela MapEME apresentará um ganho ainda maior.

REFERÊNCIAS

AMD homepage, disponível em http://www.amd.com. Acesso em: fevereiro de 2012.

AMD PHENOM, Family 10h AMD Phenom Processor Product Data Sheet, 2007, disponível em http://support.amd.com/br/Processor_TechDocs/44109.pdf. Acessos em 26 de dezembro de 2011.

ALVES, M. A. Z. **Avaliação do Compartilhamento de Memórias Cache no Desempenho de Arquiteturas *Muli-core***. 2009. 175 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ASANOVIC, R et al. The Landscape of Parallel Computing Researches: A View from Berkeley. **Electrical Engineering and Computer Sciences**, University of California at Berkeley, Technical Report No. UCB/EECS-2006-183, vol. 18, Dezembro 2006.

BHANOT, G. et al. Optimizing task layout on the Blue Gene/L supercomputer. **IBM Journal of Research and Development**, p. 489-500, Março 2005.

BOKHARI, S. H. On the Mapping Problem. **IEEE Transactions on Computers C-30**, 5, 207-214, 1981.

BOLLINGER, S.; Midkiff S. Heuristic Technique for Processor and Link Assignment in Multicomputers. **IEEE Transactions on Computers**, vol. 40, Issue 3, Março 1991.

BROQUEDIS, F. et al. hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications. In: 18th EUROMICRO CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORKBASED PROCESSING, PDP, 2010. **Proceedings...** Piza, Italy: IEEE Computer Society Press, 2010. p. 180-186.

BUNTINA, D.; MERCIER, G.; GROOP, W. Design and evaluation of Nemesis, a scalable low-latency message-passing communication subsystem. In: SIXTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRIG, 2006. **Proceedings...** Singapure. IEEE Computer Society Press, 2006. p.530-540.

BUNTINA, D.; MERCIER, G.; GROPP, W. Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem. **Parallel Computing**, vol. 33, p. 634-644, Issue 9, Setembro 2007.

CHAI, L.; GAO, Q.; PANDA, D. Understanding the Impact of Multi-core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System. In: 7th IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID,

CCGRID, 2007. **Proceedings...** Rio de Janeiro, Brasil: IEEE Computer Society Press, 2007. p. 471-478.

CHAI, L. **High Performance and Scalable MPI Intra-node Communication Middleware for Multi-Core Clusters**, 2009, 155 f. Thesis on The Ohio State University.

CHANDRA, R et al. **Parallel Programming in OpenMP**, San Francisco: Morgan Kaufmann, 2001.

CHEN, H. et al. MPIPP: an automatic profile-guide parallel process placement toolset for SMP clusters and multiclusters. In: 20th ANNUAL INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, ICS '06, 2006. **Proceedings...** Nova York: USA: ACM Press, 2006. p. 353-360.

DÜMMLER, J.; RAUBER, T.; RÜNGER, G. Mapping Algorithms for Multiprocessors Tasks on Multi-core Clusters. In: 37th INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, ICPP '08, 2008. **Proceedings.** Washington: DC: USA: IEEE Computer Society Press, 2008. p. 141-148.

CRUZ, E. H. M., ALVES, M. A. Z, NAVAUX, P. O. A., Process Mapping Based on Memory Access Traces, In: WSCAD-SSC, 2010. **Proceedings of 11th Symposium on Computing Systems**, 2010, p. 72-79.

FERREIRA, M. K. et al. Static Process Mapping Heuristics for MPI Parallel Processes in Homogeneous Multi-core Clusters. In: CONFERENCIA LATINOAMERICANA DE COMPUTACIÓN DE ALTO RENDIMIENTO, CLCAR 2011. **Proceegings...** Colima: México. (Melhor Paper CLCAR 2011)

FERREIRA, M. K.; CRUZ, V. S.; NAVAUX, P. O. A. Static Process Mapping Heuristics Evaluation for MPI Processes in Multi-core Clusters. In: IX WORKSHOP DE PROCESSAMENTO PARALELO E DISTRIBUÍDO, WSPPD 2011. **Proceedings...** Porto Alegre. IX Workshop de Processamento Paralelo e Distribuído, 2011

FREITAS, H. C., ALVES, M. A. Z., NAVAUX, P. O. A. NoC e NUCA: Conceitos e Tedências para Arquiteturas de Processadores Many-core, **Escola Reginal de Alto Desempenho (ERAD)**, Cap. 3, 2009.

GROPP, W.; LUSK, E.; SKJELLUM, A. **Using MPI – Portable Parallel Programming with Message-Passing Interface**. 2.ed. Massachusetts Institute of Technology – Cambridge, Massachusetts, The MIT Press, 1999.

HENNESSY, J. L., PETTERSON, D. A. Computer Architecture: A Quantitative Approach , 4th Edition, **Morgan Kaufmann**, 2007.

INTEL homepage, diponível em http://www.intel.com.br. Acesso em: fevereiro de 2012.

INTEL I7, Intel Core i7-800 Descktop Processor Series, 2010, disponível em http://download.intel.com/design/processor/datashts/320834.pdf. Acesso em dezembro de 2011.

INTEL CORPORATION. Cluster OpenMP User Guide, 2005, disponível em http://software.intel.com/file/6330. Acesso em: outubro de 2011.

KOLMOGOROV, V. BLOSSOM, V: A new implementation of a minimum cost perfect matching algorithm, **Mathematical Programming Computation**, 1(1):43-67, julho de 2009.

KOPPLER, R. Geometry-Aided Rectilinear Partitioning of Unstructured Meshes, **Lecture Notes in Computer Science**. vol. 1557, p.450-459, 1999.

KUMAR, R. et al.. Heterogeneous Chip Multiprocessors. **Computer Journal**, vol. 38, p.32-38, Issues 11, Novembro 2005.

LAM, Y. M. et al. Mapping and Scheduling with Task Clustering for Heterogeneous Computing Systems, In: INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, 2008. **Proceedings.** Heidelberg, Alemanha, 2008, p. 275 – 280.

LASTOVETSKY, A., et al. MPIBlib: Benchmarking MPI Communications for Parallel Computing on Homogeneours and Heterogeneous Cluster, **Lecture Notes in Computer Science**, pp. 227-238, 2008.

LUI, J.; WU, J.; PANDA, D. K. High Performance RDMA-based MPI implementations over InfiniBand, In: 17th ANNUAL ACM INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, ICS '03, 2003. **Proceedings.** Nova York: USA: ACM Press, 2003.

MPICH2 download page, disponível em: <http://www.mcs.anl.gov/research/projects/mpich2/downloads/index.php?s=downloads>. Acesso em: maio de 2012.

MPICH2 1.3 download page, disponível desde outubro de 2010 em: <http://www.mcs.anl.gov/research/projects/mpich2/downloads/tarballs/1.3/> . Acesso em: maio de 2012.

MURPHY, R. C., KOGGE, P. M. On the Memory Access Patterns of Supercomputer Applications: Benchmark Selection and Its Implication, **IEEE Transactions on Computer**, vol. 56, no. 7, pp. 937-945, 2007.

NASA: National Aeronautics and Space Administration – NAS Parallel Benchmarks (NPB-3.3), disponível em: <http://www.nas.nasa.gov/Resources/Software/npb.html>. Acesso em: dezembro de 2011.

OLOKOTUN, K. et al. The Case for a Single-Chip Multiprocessosr. In: ASPLOS: INT. SYMP. ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 1996. **Proceedings …** IEEE, 1996.

OPENMPI homepage, disponível em http://www.open-mpi.org. Acesso em: outubro de 2011.

OSIAKWAN, C., AKL, S. The Maximum Weight Perfect Matching Problem for Complete Weighted Graphs is in PC, PARALLEL AND DISTRIBUTED PROCESSING, 1990, **Proceedings of the Second IEEE Symposium on...** Dallas: TX: USA, 1990, p. 880-887.

PELLEGRINI, F. Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs, SCALABLE HIGH-PERFORMANCE COMPUTING CONFERENCE, 1994, **Proceedings...** Knoxville: TN : USA, 1994, p. 486 – 493.

PELLEGRINI, F.; ROMAN, J. Experimental Analysis of the Dual Recursive Bipartitioning Algorithm for Static Mapping. **Research Report,** pp. 11-38, 1996.

PELLEGRINI, F et al. Scotch and libScotch 5.1 User's Guide, novembro de 2010, disponível em http://gforge.inria.fr/docman/view.php/248/7104/scotch_user5.1.pdf. Acesso em: dezembro de 2011.

RODRIGUES, E. R.; MADRUGA, F. L.; NAVAUX, P. O. A. Multi-core Aware Process Mapping and its Impact on Communication Overhead of Parallels Applications, In: IEEE SYNOISUYN ON COMPUTERS AND COMMUNICATIONS, ICSS2009, 2009. **Proceedings...** Sousse: Tunisia: IEEE Computer Society Press, 2009, p. 811-817.

SAPHIR, W. et al. New Implementations and Results for the NAS Parallel Benchmarks 2, In: 8th SIAM CONFERENCE ON PARALLEL PROCESSIN SCIENTIFIC COMPUTING, 1997. **Proceedings...** Minneapolis: Minnesota: USA, 1997.

WEST, D. B. **Introdution to Graph Theory**, 2.ed. Upper Saddle River : Prentice Hall, 2001. Capítulo 3, p. 107-148.

ZHANG, C.; YUAN, X.; SRINIVASAN, A. Processor Affinity and MPI Performance on SMP-CMP Clusters, In: 11th IPDPS on WORKSHOP ON PARALLEL AND DISTRIBUTED SCIENTIFIC AND ENGINEERING COMPUTING, PDSEC, 2010. **Proceedings...** Atlanta: USA: IEEE Computer Society Press. 2010.