PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Programa de Pós-Graduação em Informática

Cíntia Pinto Avelar

AVALIAÇÃO DE ABORDAGENS DE MAPEAMENTO DE PROCESSOS EM REDES-EM-CHIP PARA APLICAÇÕES PARALELAS

Belo Horizonte

08 de Setembro de 2014

Cíntia Pinto Avelar

AVALIAÇÃO DE ABORDAGENS DE MAPEAMENTO DE PROCESSOS EM REDES-EM-CHIP PARA APLICAÇÕES PARALELAS

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Henrique Cota de Freitas

Belo Horizonte

08 de Setembro de 2014

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Avelar, Cíntia Pinto
A949a Avaliação de abordagens de mapeamento de processos em redes-em-chip para aplicações paralelas / Cíntia Pinto Avelar. Belo Horizonte, 2014.
84 f. : il.

Orientador: Henrique Cota de Freitas
Dissertação (Mestrado) - Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática.

1. Sistemas programáveis em chip. 2. Sistemas de comunicação sem fio. 3. Processamento paralelo (Computadores). 4. Arquitetura de redes de computador. I. Freitas, Henrique Cota de. II. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática. III. Título.

CDU: 681.3-11

Dissertação apresentada ao Programa de Pós-Graduação em Informática como requisito parcial para qualificação ao Grau de Mestre em Informática pela Pontifícia Universidade Católica de Minas Gerais.

Cíntia Pinto Avelar

AVALIAÇÃO DE ABORDAGENS DE MAPEAMENTO DE PROCESSOS EM REDES-EM-CHIP PARA APLICAÇÕES PARALELAS

Prof. Dr. Henrique Cota de Freitas –
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

Prof. Dr. Mark Alan Junho Song –
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

Prof. Dr. Márcio Bastos Castro –
Universidade Federal de Santa Catarina (UFSC)

Belo Horizonte, 08 de Setembro de 2014

AGRADECIMENTOS

Agradeço, primeiramente, o meu orientador, Prof. Dr. Henrique Freitas, pela oportunidade, pelos conhecimentos transmitidos à mim, pela paciência e dedicação! Muitíssimo obrigada, Henrique! Obrigada a todos os brilhantes Doutores que compõem o corpo docente do Mestrado em Informática da PUC Minas. E obrigada Giovana por me aguentar (rs)!

Aos meus pais e minha irmã por sempre me apoiarem e me ajudarem diante de obstáculos que enfrentei e que ainda enfrentarei e por compreenderem a minha ausência em alguns momentos pela dedicação ao mestrado. Obrigada pela dedicação de sempre. Obrigada por serem meus exemplos de vida e por me ensinarem a viver.

Não posso deixar de citar e agradecer os colegas e amigos que fiz no mestrado, principalmente aos mestrandos que dividiam os laboratórios: Matheus Queiroz, Matheus Souza, Agripino, Faber, Carolina, Paulo. Obrigada pela troca de conhecimento, pelas ideias e experiências compartilhadas. E, não podemos esquecer, das conversas fiadas para descontrair e que sem elas não teria sido tão bom! Um agradecimento especial ao Pedro Henrique Penna, que além de ter contribuído para este trabalho, foi um excelente amigo. Obrigada à Poliana (Mestre pela PUC Minas) pela contribuição, dicas e conversas.

Aos colegas e amigos do grupo de pesquisa CArT (*Computer Architecture and Parallel Processing Team*) pelos trabalhos e contribuições, mesmo que indiretas. Que a parceiria não se encerre com essa dissertação.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela bolsa integral concedida à mim durante um período dos anos de 2013 e 2014. A Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) pelos recursos computacionais utilizados.

Obrigada à todos os meus amigos e parentes que me apoiaram nessa caminha. As festas, as reuniões, as conversas, as piadas e risadas (que foram de chorar.. rs)... Cada momento (bom ou ruim) foi e é essencial para mim! Obrigada à todos por estarem sempre comigo!

Agradeço, também, a todos os meus colegas e amigos do TCEMG que me incentivaram e acreditaram em mim. Obrigada, especialmente, à minha equipe de trabalho pela compreensão, companheirismo e amizade.

"O primeiro dever da inteligência é desconfiar dela mesma."

Albert Einstein

RESUMO

As Redes-em-*Chip*, ou NoCs (do inglês *Networks-on-Chip*) surgiram para minimizar os problemas nas arquiteturas *many-core* relacionados à atraso na transferência dos dados, escalabilidade, perda de pacote devido à baixa resistência do barramento, dentre outros. Nessa arquitetura existem roteadores responsáveis pelas comunicações entre os núcleos, o que aumenta a capacidade de execução de aplicações paralelas que se comunicam via troca de mensagens. Porém, devido à alguns problemas físicos de projeto, diversas técnicas estão sendo propostas para alcançarem melhores desempenhos com a utilização dessa arquitetura. Uma dessas técnicas é a de mapeamento de processos, onde o foco é a distribuição dos processos nos núcleos de processamento. Este trabalho apresenta e avalia três heurísticas de mapeamento estático, onde a análise da comparação é feita com base no desempenho de redes-em-chip simuladas. Além disso, este trabalho apresenta um estudo do algoritmo de clusterização K-means como estratégia de mapeamento de processos em redes-em-chip, antes não apresentado em trabalhos do estado da arte, em comparação com as heurísticas BRD e Gulosa. Os resultados foram obtidos por meio de simulações e mostram que, para alguns padrões de comunicação, o K-means obteve melhores resultados, e em outras situações, o BRD é a melhor opção. O estudo comprova, também, a importância de obter conhecimento da carga de trabalho para apontar a melhor estratégia de mapeamento de processos para a carga em questão.

Palavras-chave: Redes-em-chip, mapeamento de processos, heurísticas de mapeamento de processos.

ABSTRACT

Networks-on-Chip (NoCs) have emerged to minimize problems in many-core related to the delay in the transfer of data, scalability, packet loss due to the low resistance of the bus, and so on. In this architecture there are routers responsible for the communication between cores, which increase the ability to execute parallel applications that communicate via messaging. However, because of some physical design problems, several techniques have been proposed to achieve a better performance using this architecture. One of these techniques is the mapping of processes, which focus on the distribution of processes in processor cores. This master thesis presents and evaluates three static mapping heuristics, in which an analysis of the comparison is made based on the performance of networks-on-chip simulated. Furthermore, this work is the first to present a study of the K-means clustering algorithm as a means to perform process mapping on networks-on-chip, comparing it to the BRD and Greedy heuristics. The results were obtained through simulations and showed that for some communication patterns, the K-means obtained better results, and in other situations, the BRD is the best option. The study also demonstrates the importance of gathering information from the workload to choose the best process mapping strategy to be applied.

Keywords: Networks-on-chip, process mapping, process mapping heuristics.

LISTA DE FIGURAS

FIGURA 1 – MPPA 256 30
FIGURA 2 – Ilustração de uma NoC 31
FIGURA 3 – Topologias 32
FIGURA 4 – Ilustração de um roteador 33
FIGURA 5 – Algoritmo BRD 40
FIGURA 6 – Algoritmo Guloso 41
FIGURA 7 – Algoritmo Kmeans 42
FIGURA 8 – Exemplo de Trace 45
FIGURA 9 – Comunicação via memória 46
FIGURA 10 – Gráfico de Tempo de Execução dos Algoritmos 51
FIGURA 11 – Gráficos de latência do *buffer* da aplicação CG 52
FIGURA 12 – Gráficos de latência das mensagens na rede da aplicação CG 53
FIGURA 13 – Gráficos de Latência total das mensagens da aplicação CG 53
FIGURA 14 – Gráficos de distância média da aplicação CG 54
FIGURA 15 – Gráficos de latência dos *buffers* da aplicação EP 56
FIGURA 16 – Gráficos de latência das mensagens na rede da aplicação EP 57
FIGURA 17 – Gráficos de Latência total das mensagens da aplicação EP 58
FIGURA 18 – Gráficos de distância média da aplicação EP 58
FIGURA 19 – Gráficos de latência dos *buffers* da aplicação FT 60
FIGURA 20 – Gráficos de latência das mensagens na rede da aplicação FT 61
FIGURA 21 – Gráficos de Latência total das mensagens da aplicação FT 62
FIGURA 22 – Gráficos de distância média da aplicação FT 63
FIGURA 23 – Gráficos de latência dos *buffers* da aplicação IS 64
FIGURA 24 – Gráficos de latência das mensagens na rede da aplicação IS 65

FIGURA 25 – Gráficos de Latência total das mensagens da aplicação IS 66
FIGURA 26 – Gráficos de distância média da aplicação IS 67
FIGURA 27 – Gráficos de latência dos *buffers* da aplicação MG 68
FIGURA 28 – Gráficos de latência das mensagens na rede da aplicação MG 69
FIGURA 29 – Gráficos de Latência total das mensagens da aplicação MG 70
FIGURA 30 – Gráficos de distância média da aplicação MG 71
FIGURA 31 – Gráfico de mensagens geradas na aplicação CG. 79
FIGURA 32 – Gráficos de mensagens entregues na aplicação CG. 80
FIGURA 33 – Gráfico de mensagens geradas na aplicação EP. 80
FIGURA 34 – Gráficos de mensagens entregues na aplicação EP. 81
FIGURA 35 – Gráfico de mensagens geradas na aplicação FT. 81
FIGURA 36 – Gráficos de mensagens entregues na aplicação FT. 82
FIGURA 37 – Gráfico de mensagens geradas na aplicação IS. 82
FIGURA 38 – Gráficos de mensagens entregues na aplicação IS. 83
FIGURA 39 – Gráfico de mensagens geradas na aplicação MG. 83
FIGURA 40 – Gráficos de mensagens entregues na aplicação MG. 84

LISTA DE ABREVIATURAS E SIGLAS

BRD Biparticionamento Recursivo Dual

CDCM *Communication dependence and computation model*

CG *Conjugate Gradient*

CMP *Chip Multiprocessor*

CPU *Central Processing Unit*

EP *Embarrassingly Parallel*

FLIT *Flow Control Unit*

FPGA *Field Programmable Gate Array*

FT *Fast Fourier Transform*

IS *Integer Sort*

JPEG *Joint Photographic Experts Group*

MG *Multi-Grid*

MPE MPI *Parallel Environment*

MPEG *Moving Picture Experts Group*

MPI Interface de passagem de mensagem, do inglês *Message Passing Interface*

MPICH *Message Passing Interface Chameleon*

MPPA *Multi-Purpose Processor Array*

NAS NASA *Advanced Supercomputing*

NoCs Redes-em-Chip, do inglês *Networks-on-Chip*

NPB NAS *Parallel Benchmark*

NP-difícil do inglês *Non-Deterministic Polynomial-time hard*

OpenMP Multi-processamento aberto, do inglês *Open Multi-Processing*

QoS Qualidade de Serviço, do inglês *Quality of Service*

RAM *Random Access Memory*

UMA *Uniform Memory Access*

WNoCs Redes-em-Chip Sem Fio, do inglês *Wireless Networks-on-Chip*

SUMÁRIO

1 INTRODUÇÃO 25

1.1 Problema 26

1.2 Objetivos 27

1.3 Justificativa e Contribuição 27

1.4 Organização do texto 28

2 REFERENCIAL TEÓRICO 29

2.1 Arquiteturas *Many-Core* 29

2.2 NoC: Conceitos e Estruturas 30

2.3 Mapeamento de Processos 34

2.4 Trabalhos Correlatos 35

3 HEURÍSTICAS AVALIADAS 39

3.1 Biparticionamento Recursivo Dual - BRD 39

3.2 Algoritmo Guloso 40

3.3 *K-means* 41

4 MÉTODO DE AVALIAÇÃO PROPOSTO 43

4.1 Cargas de Trabalho 43

4.1.1 Arquivos de teste baseados no NPB 44

4.1.2 Custo das Comunicações 47

4.2 Ambiente de simulação 47

4.3 Decisões de Simulação e Avaliação 48

5 AVALIAÇÃO DOS RESULTADOS 50

5.1 Tempo de execução dos algoritmos 50

5.2 Aplicação CG 51

5.3 Aplicação EP 55

5.4 Aplicação FT 59

5.5 Aplicação IS 63

5.6 Aplicação MG 67

5.7 Considerações Finais 71

6 CONCLUSÃO **73**

APÊNDICE A – RESULTADOS COMPLEMENTARES **79**

1 INTRODUÇÃO

Em uma arquitetura de processadores *multi-core* a conexão de núcleos de processamento é realizada por barramento ou por chaves *crossbar* (KUMAR; ZYUBAN; TULLSEN, 2005). Embora sejam soluções simples, as duas abordagens enfrentam problemas de escalabilidade devido às restrições físicas da interconexão (fio), quando do aumento da quantidade de núcleos e consequente aumento do fio, tais como: resistência na propagação dos dados, roteamento físico da interconexão e maior competição pelo canal de comunicação. Devido ao crescente número de aplicações requisitando um desempenho melhor do *hardware*, houve a necessidade de inserir mais núcleos dentro do chip. Surgia, então, uma nova categoria: os processadores *many-core*. Para diminuir o impacto dos problemas relacionados aos fios surgiram as redes-em-chip (do inglês *Networks-on-Chip* - NoCs). Elas reduzem o comprimento do fio devido a inserção de roteadores responsáveis pela comunicação entre os núcleos.

As NoCs permitem integrar diversos núcleos de processamento em um único chip e são compostas por um conjunto de roteadores interligados. Estudos mostram que as NoCs podem trazer um ganho de desempenho considerável em comunicações intra chip para aplicações paralelas que se comunicam via troca de mensagens ou compartilhamento de memória (FREITAS; NAVAUX, 2009) (BENINI; MICHELI, 2002) (BENINI; MICHELI, 2005). Apesar de apresentarem diversas vantagens, ainda existem limitações de *hardware* e de projeto relevantes que prejudicam o desempenho de uma NoC. Pode-se citar como problemas causadores de perda de desempenho: alta latência, consumo de energia, tráfego intenso causador de gargalos na rede, estratégias e ambientes de programação, dentre outros. Com base nisso, ao longo dos anos foram propostas várias soluções a fim de resolver ou, pelo menos, diminuir o impacto desses problemas.

Uma rede-em-chip possui algumas características semelhantes às de uma rede de computadores. Essa arquitetura permite a implementação de protocolos/algoritmos de roteamento, controle de fluxo, canal virtual, QoS, interface de rede, dentre outros (AGARWAL; ISKANDER; SHANKAR, 2009). A maioria das pesquisas de NoCs focam seus estudos em um ou mais desses parâmetros com o objetivo de melhorar o desempenho significativamente.

Existem vários tipos de NoCs, cada uma com um objetivo, mas a maioria busca melhores taxas de desempenho. Dentre os tipos de NoCs existentes para solucionar as questões envolvidas anteriormente, pode-se destacar: NoCs Tridimensionais, NoCs Nanofotônicas, as WNoCs (NoCs sem fio) e NoCs reconfiguráveis (CARLONI; PANDE; XIE, 2009). Além dos variados tipos de NoCs, também existem técnicas que são utilizadas para ganho de desempenho. Uma delas consiste em um mapeamento de processos, no qual será

o tema principal abordado neste trabalho.

A tarefa de mapear processos consiste em definir a distribuição dos processos nos núcleos com o objetivo de reduzir o tempo de execução de uma aplicação paralela. Em um mapeamento de processos podem ser adotados vários critérios para criar/utilizar uma estratégia: número de comunicações entre os processos, concorrência, disponibilidade de recursos, número de saltos entre o núcleo origem e o núcleo destino, tempo de execução de cada processo, tamanho do pacote de dados que será transmitido, dentre outros.

Uma situação que leva a utilização de técnicas de mapeamento é quando há muita troca de mensagens entre processos que talvez possam estar em núcleos distantes. Para mapear os processos da melhor forma possível é fundamental conhecer a carga de trabalho que será executada e/ou o padrão de comunicação dessas cargas, caso contrário o mapeamento pode não ser eficaz o suficiente. Por esse motivo e por ser um problema da classe NP-Difícil (BOKHARI, 1981), a tarefa de mapear processos não é trivial, pois cargas de trabalho podem apresentar comportamentos distintos. Neste contexto é de grande importância o conhecimento das aplicações, pois assim é possível apontar a melhor estratégia de mapeamento. Vale ressaltar que o mapeamento de processos pode ser realizado em qualquer um dos tipos de NoCs citados e em qualquer outro projeto que haja comunicação entre periféricos, porém este trabalho irá abordar o mapeamento de processos em uma NoC convencional (e.g. topologia *mesh*, roteamento XY).

1.1 Problema

As oportunidades de pesquisas existentes em NoCs são várias: latência de comunicação, consumo de energia, gargalos na rede, esquemas de roteamento, estratégias de programação, limitações na capacidade dos fios de metal que interligam os núcleos/roteadores, dentre outros. Para todas as oportunidades existem problemas, que mesmo de formas diferentes, causam uma redução no desempenho de uma NoC.

O problema abordado neste trabalho pode ser resumido na seguinte pergunta: Como estratégias de mapeamento de processos podem obter ganho no desempenho em NoCs na execução de aplicações paralelas?

O mapeamento de processos é importante em cargas paralelas com comunicações coletivas devido à dependência ou à interação entre os processos e, considerando que cada processo execute em um núcleo separado, a possibilidade de criar um gargalo na rede é alta, logo pode haver uma queda considerável no desempenho.

1.2 Objetivos

Considerando o problema exposto na Seção 1.1, o objetivo deste trabalho é avaliar três abordagens de mapeamento de processos em uma NoC e identificar a melhor para as cargas utilizadas. As aplicações que serão usadas em ambiente de simulação para mapeamento e avaliação são oriundas do *benchmark* do NPB ((NAS),2010) e possuem comunicações coletivas.

Dentre os objetivos específicos pode-se citar:

a) Modelar uma NoC em um ambiente de simulação que permite a visualização de diversos parâmetros.
b) Avaliar e definir estratégias de mapeamento.
c) Implementar e testar as estratégias de mapeamento definidas.
d) Avaliar o desempenho de cada estratégia considerando cada carga de trabalho separadamente.

1.3 Justificativa e Contribuição

No mapeamento de processos, várias estratégias podem ser aplicadas para solucionar problemas de desempenho. Isso torna o problema ainda mais complexo, pois encontrar uma estratégia ideal para a execução de uma carga de trabalho específica requer análise e conhecimento mais extenso da aplicação e do padrão de comunicação entre os processos.

A abordagem deste trabalho é diferente dos demais trabalhos, ou seja, trabalhos que abordam os temas de redes-em-chip e mapeamento de processos e ainda comparam diferentes estratégias de mapeamento são pouco encontrados na literatura. Essa abordagem pode trazer novas possibilidades de estudos para quem busca desempenho em NoCs. Além disso, garantindo o ganho de desempenho, a indústria de processadores poderá se beneficiar com este estudo fabricando chips considerando a técnica de mapeamento de processos, uma vez que, conforme a metodologia adotada, além da rede-em-chip, decisões de mapeamento também são influenciadas pela organização da memória. Outro ponto interessante do estudo é a possibilidade de implementar técnicas de mapeamento nos próprios Sistemas Operacionais de aplicações específicas, por exemplo, já que é crescente a quantidade de processadores com a tecnologia de redes-em-chip.

Portanto, é importante ressaltar que este estudo está voltado para a análise de três algoritmos em cargas paralelas, sendo que esta análise ainda não foi encontrada nos trabalhos correlatos.

1.4 Organização do texto

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta o referencial teórico com os conceitos mais importantes e trabalhos relacionados à este estudo; as heurísticas de mapeamento estudadas e utilizadas neste trabalho podem ser encontradas no Capítulo 3; no Capítulo 4 são descritos os métodos/procedimentos realizados para testar os algoritmos apresentados; a análise dos resultados é mostrada no Capítulo 5; por fim, no Capítulo 6 contém as conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo é possível entender melhor a evolução dos processadores e como surgiram as NoCs. Apresenta também alguns conceitos importantes relacionados à Redes-em-Chip e Mapeamento de Processos. Ao final encontram-se os Trabalhos Correlatos.

2.1 Arquiteturas *Many-Core*

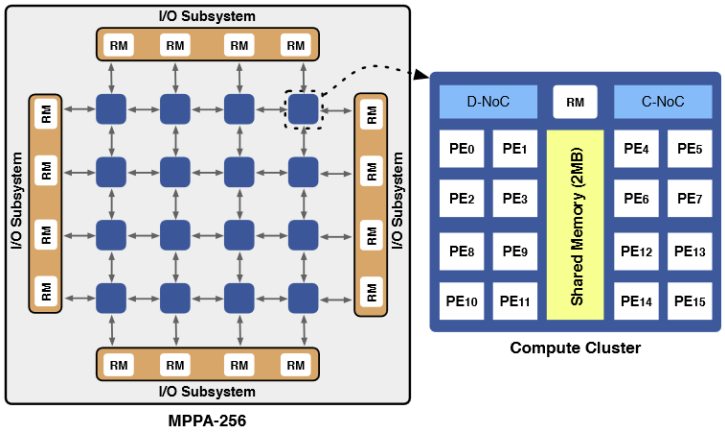
A Lei de Moore (MOORE, 1965) diz que o número de transistores nos chips de processamento dobraria a cada dois anos, aproximadamente. Isso ocorreria devido à redução do tamanho dos transistores e um custo de fabricação relativamente baixo. Essa redução do tamanho desse componente viabilizou o aumento da frequência dos chips, elevando o desempenho da computação. Porém, chegou a um ponto em que o aumento dessa frequência não era recomendável devido a alta potência dissipada. Diante desse cenário, surgiram novos projetos de arquitetura: os *multi-cores*. Essa nova arquitetura propõe a integração entre núcleos (por meio de um único barramento) no lugar do aumento de frequência, ou seja, os núcleos, individualmente, não necessitam de altos níveis de frequência, pois o processamento é feito em paralelo, onde se ganha desempenho.

Com o passar do tempo surgiu a necessidade de aumentar o número de núcleos integrados para elevar ainda mais a *performance*. Então, surgiu uma nova classe: os *many-cores*. Essa arquitetura permite integrar um número elevado de núcleos de processamento em um único chip. Dessa forma é possível alcançar um desempenho maior e um consumo de energia reduzido (BORKAR, 2007).

O uso de uma arquitetura *many-core* justifica-se pela execução de cargas de trabalho que exigem processamento paralelo em nível de instrução (FREITAS; ALVES; NAVAUX, 2009). O projeto dos *many-cores* começa a ter problemas quando há um número elevado de núcleos que executam cargas paralelas, pois a tendência é aumentar o número de comunicações entre esses núcleos. Logo, o barramento que interliga os núcleos tende a ficar sobrecarregado, prejudicando o tráfego dos dados. Uma solução proposta para esses problemas são as redes-em-chip (BENINI; MICHELI, 2002) (BENINI; MICHELI, 2005) (BJERREGAARD; MAHADEVAN, 2006) discutidas na próxima seção.

A Kalray é um exemplo de companhia que possui um processador *many-core* chamado MPPA-256 (CASTRO et. al., 2013) da família MPPA *MANYCORE* cujo projeto está representado na Figura 1. O MPPA-256 é composto por *clusters*, cada um contendo 16 núcleos e memória compartilhada, conectados por uma NoC. A NoC do MPPA-256 está estruturada em uma topologia *Torus*-2D com um canal full duplex de 3,2 GB/s entre *clusters* adjacentes e possui implementação do recurso de Qualidade de Serviço.

Figura 1 – Visão geral do MPPA 256 da família MPPA MANYCORE da Kalray



Fonte: (CASTRO et. al., 2013)

Além do desempenho, os processadores *many-core* trazem outras vantagens importantes, como a possibilidade de utilizar somente os núcleos necessários, o balanceamento da carga de trabalho entre os núcleos, a possibilidade de substituir núcleos com defeito, dentre outros (FREITAS; NAVAUX, 2009). Vale ressaltar que a escolha de uma arquitetura depende, em grande parte, da carga de trabalho que será executada. Ou seja, a partir de características da carga é possível saber se uma NoC é aplicável e qual será sua implementação, ou se um processador *many-core* com interconexões de barramentos é suficiente, ou se um *multi-core* atende às necessidades, ou até mesmo um *single-core*.

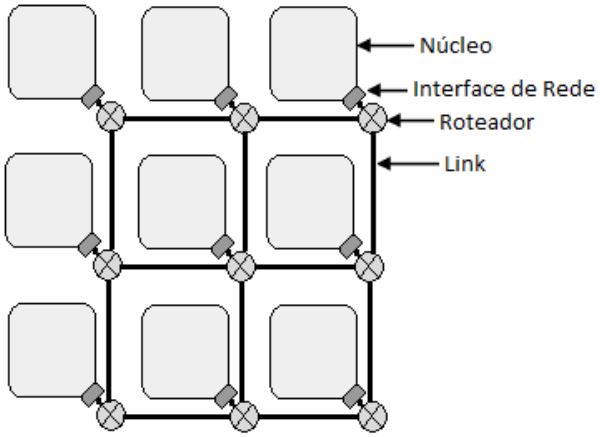
2.2 NoC: Conceitos e Estruturas

O conceito de uma rede de comunicação surgiu para conectar um grande número de sistemas, substituindo conexão dedicada ponto-a-ponto e de outras formas de interligação em pequena escala. Cada rede deve fornecer uma solução de custo eficaz para um grande número de exigências conflitantes, como a flexibilidade, escalabilidade, confiabilidade e desempenho (CIDON,2009). Portanto, fica a cargo de arquitetos e *designers* de rede solucionar vários pontos de um problema de otimização, resultando em soluções diversificadas e numerosas de rede.

Uma NoC é composta por três elementos básicos: roteador, *links* de dados e interface de rede, como mostra a Figura 2. O roteador é responsável pela comunicação entre o núcleos (feita por meio de troca de pacotes de dados), logo ele é encarregado por definir as rotas dos pacotes, controlar o fluxo de dados e garantir qualidade de serviço. Os *links* de dados correspondem aos caminhos por onde os pacotes irão passar, portanto são eles quem interligam os roteadores, formando a topologia da rede. Por último e não menos importante, a interface de rede é responsável por garantir a comunicação entre o

roteador e o núcleo, ou seja, deve fazer a comunicação correta entre protocolos diferentes (FREITAS; NAVAUX, 2009).

Figura 2 – Ilustração de uma NoC e seus componentes

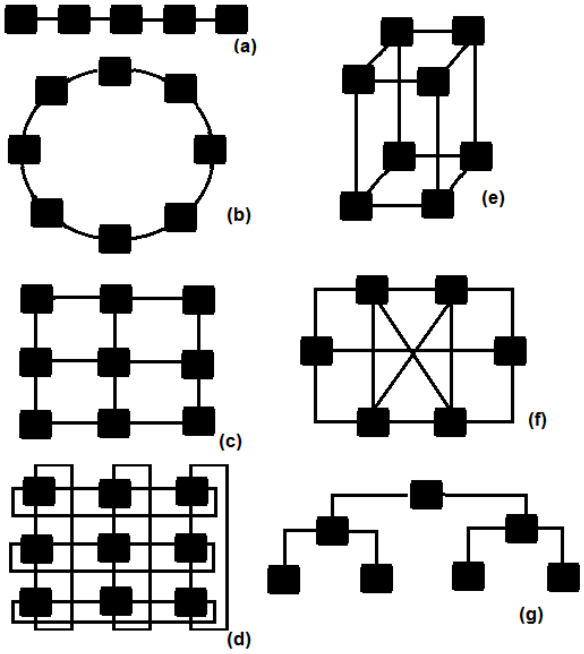


Fonte: do Autor.

Argumenta-se que a otimização do desempenho de uma NoC pode conduzir à várias soluções para minimizar os impactos de possíveis problemas da arquitetura (CIDON, 2007). O desafio dos estudiosos é buscar uma solução para problemas de redes tradicionais, tais como roteamento, qualidade de serviço, fluxo e controle de congestionamento e confiabilidade. O trabalho de (AGARWAL; ISKANDER; SHANKAR, 2009) apresenta vários trabalhos relacionados à essas questões de redes que podem influenciar no desempenho de uma rede-em-chip.

Uma rede-em-chip pode ser caracterizada pela sua topologia e pelos mecanismos (protocolos) de comunicação utilizados. Os protocolos de redes definem como são efetuadas as transferências de dados e a topologia de uma rede significa a organização sob a forma de um grafo, onde as arestas são os canais de comunicação e os vértices correspondem aos núcleos. Ou seja, a topologia indica a estrutura na qual os roteadores serão interligados. A escolha de uma topologia está relacionada com a carga de trabalho que será executada e influencia diretamente no desenho da rede (FREITAS; NAVAUX, 2009). Por esse motivo, a tarefa de escolher uma topologia ideal não é fácil. A topologia mais comum é a *Mesh* e sua representação pode ser vista na Figura 3(c).

Figura 3 – Topologias de redes de interconexão: Rede Linear (a), Rede Anel (b), Rede Grelha (c), Rede Toróide (d) Rede Cubo de Grau 3 (e), Rede Completamente Conectada (f) e Rede em Árvore (g)



Fonte: (GONÇALVES, 2010)

A Figura 3 mostra alguns tipos de topologias de rede (GONÇALVES, 2010). A mais simples de todas e talvez a menos utilizada devido ao baixo desempenho é a Rede Linear (Figura 3(a)). Nessa topologia, os núcleos são interligados aos núcleos ao lado (esquerdo e direito), sendo que os que se localizam nas extremidades são interligados apenas com um núcleo. A rede Anel (Figura 3(b)) nada mais é que a Linear com os extremos interconectados. Ela consegue um desempenho superior à Linear, pois o pior caso da Linear (envio de mensagem entre os extremos) é solucionado. Na rede Mesh (Figura 3(c)) os núcleos são interligados com os seus adjacentes formando uma matriz. A ligação entre os extremos dessa rede forma uma rede Torus (Figura 3(d)). Na Figura 3(e) pode-se observar a rede Hipercubo, onde cada nó é interligado a outro dependendo do grau da rede. No caso da figura, a rede tem grau 3, portanto cada nó é interligado a 3 outros nodos. Na rede Totalmente Conectada (Figura 3(f)) os núcleos estão interconectados com todos os outros. Essa rede seria a mais ideal, dentre as citadas, por ser mais rápida, porém o que a torna difícil de ser implantada é o alto custo de implementação, pois a adição de um novo nó requer outro projeto. Por fim, na rede Árvore Binária (Figura 3(g)) cada núcleo está conectado a outros dois núcleos com a presença de um nó raiz, formando uma árvore binária.

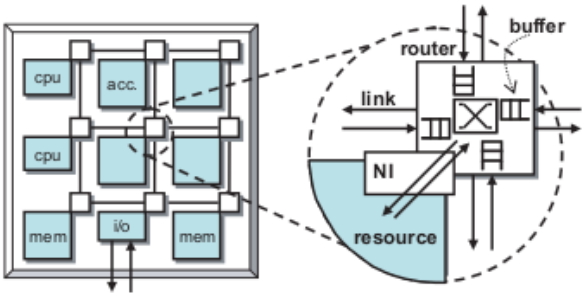
Em (FREITAS; NAVAUX; SANTOS, 2008) são apresentadas decisões de projeto de arquiteturas NoC *Multi-Cluster* com o objetivo de apoiar padrões de comunicação coletiva por meio de topologias reconfiguráveis em um chip baseado em FPGA. Os resultados

apresentam a viabilidade da proposta, pois reduziu o consumo de energia quando comparada com uma NoC convencional. Além disso, com a reconfiguração de topologias melhora a taxa de transferência de dados e o desempenho em 86% para padrões de comunicação coletiva e melhora, também, a escalabilidade e flexibilidade da rede.

Com o objetivo de avaliar e melhorar ainda mais os estudos sobre NoCs, (CARLONI; PANDE; XIE, 2009) apresenta quatro abordagens de redes-em-chip: Tridimensionais (CIDON, 2009), Nanofotônica (CARLONI; PANDE; XIE, 2009), *Wireless* NoC (ANGULY et. al., 2009) (AMORIM; OLIVEIRA; FREITAS, 2012) e NoCs reconfiguráveis (FREITAS; NAVAUX; SANTOS, 2008). As Nanofotônicas possuem propriedades únicas de comunicação óptica e espera-se atingir níveis de desempenho que não podem ser combinados em NoCs convencionais. A *Wireless* NoC é uma alternativa radical ao metal que interliga os núcleos e consiste na utilização de transmissão de sinais via Radiofrequência/ *Wireless* (CARLONI; PANDE; XIE, 2009) (ANGULY et. al., 2009).

A comunicação entre os núcleos é realizada por meio de roteadores conectados a uma interface de rede. A arquitetura dos roteadores pode variar de acordo com o projeto de cada NoC. Projetos muito complexos de roteadores não são muito recomendados se o objetivo é ganhar desempenho, ou seja, quanto mais simples o projeto melhor, pois o gasto de energia será menor e o encaminhamento pode ser feito mais rapidamente. A Figura 4 mostra um projeto de arquitetura simples de um roteador para uma NoC, ilustrando os *buffers*, os *links* de entrada e saída, a ligação com a interface de rede (que interliga ao núcleo). No caso das NoCs, o tamanho dos *buffers* pode influenciar no desempenho final: quanto maior os *buffers*, maior será a NoC.

Figura 4 – Ilustração de um roteador



Fonte: (SALMINEN; KULMALA; HAMALAINEN, 2008)

Os protocolos de roteamento podem variar de acordo com o projeto da NoC. Existem dois tipos de roteamento: roteamento determinístico e roteamento adaptativo. No roteamento determinístico, os pacotes saem da origem já com o caminho pré-determinado até o destino. No roteamento adaptativo, os pacotes saem da origem sem um caminho determinado, ou seja, utilizam de informações da rede para encaminhar os pacotes. Como exemplos de algoritmos de roteamento pode-se citar: XY, *West-First*, *North-Last*,

Negative-First, *Odd-Even*, *Fully-Adaptive* (RENTALA; LEHTONEN; PLOSILA, 2006) (TEDESCO, 2006). Dentro do contexto de NoC, algoritmos de roteamento são utilizados para diversas finalidades: tolerar falhas físicas e lógicas, balancear e controlar o fluxo de dados, escolher o menor caminho para ganhar desempenho, garantir a entrega de pacotes, dentre outras.

Por se tratar de uma rede, a NoC também permite algumas implementações afim de melhorar os serviços que ela oferece, como: a implementação de canais virtuais para diminuir congestionamento na rede; a garantia de QoS que garante a entrega de pacotes, a integridade e a latência; o controle de fluxo no que diz respeito a largura de banda da rede, a capacidade dos *buffers*, dentre outros (AGARWAL; ISKANDER; SHANKAR, 2009).

Estudos sobre NoC e suas implementações ainda devem ser realizados, pois a integração entre *hardware* e *software* ainda é uma questão sem muitas soluções reais. Acredita-se que as NoCs podem ser promissoras no que diz respeito ao desempenho, considerando o aumento de aplicações paralelas.

2.3 Mapeamento de Processos

Mapear processos consiste em definir a distribuição de processos para os processadores com o objetivo de reduzir o tempo de execução de uma aplicação paralela. Em um mapeamento de processos pode ser adotado vários critérios para montar uma estratégia: número de comunicações entre os processos, disponibilidades e necessidade de recursos, tempo de execução de cada processo, dentre outros. No contexto de processadores *many-core*, o mapeamento explora a escalabilidade de uma NoC para reduzir o tráfego na rede e, com isso, aumentar o desempenho e reduzir o consumo de energia.

Mapeamento de processos é uma técnica complexa e que requer atenção diferenciada por se tratar de um problema da classe NP-Difícil (BOKHARI, 1981). Problemas pertencentes à esta classe não possuem solução com tempo computacional polinomial. Por isso, a adoção de heurísticas/algoritmos que minimizem esse problema é fundamental para encontrar a solução ótima ou o mais próximo possível da ótima.

O mapeamento pode ser estático ou dinâmico. O mapeamento estático é aplicado à carga de trabalho antes da execução desta carga. Por isso é importante conhecer o padrão de comunicação da aplicação executada para melhor aplicar o mapeamento. Já o mapeamento dinâmico é realizado ao longo da execução da carga. Pode ser uma boa opção quando a carga de trabalho não é conhecida, porém o tempo de execução da aplicação tende a aumentar e, se for necessário a execução da mesma carga diversas vezes, o tempo de execução pode ser maior, pois o mapeamento será aplicado todas as vezes em que a

carga for executada.

Neste trabalho adotou-se o mapeamento estático, uma vez que tem-se o conhecimento prévio da carga de trabalho e o objetivo é analisar o comportamento da rede para avaliar o quão válido é mapear processos afim de ganhar desempenho

Em (DUMMLER; RAUBER; RUNGER, 2008) explora-se o uso de multiprocessadores de tarefas (M-tarefas). Considera-se diferentes algoritmos de mapeamento para M-tarefas para avaliar a eficiência e a escalabilidade usando diferentes aplicações e diferentes sistemas *multicore*. Como exemplos de aplicações, grandes programas de computação científica foram considerados. Com os resultados foi possível observar que o mapeamento ideal depende geralmente da relação entre as comunicações dentro da M-tarefas.

Outra abordagem para mapeamento de processos é mostrada em (CARLONI; PANDE; XIE, 2009) onde o foco é na redistribuição dos dados. Ele mostra que a redistribuição de dados representa a maior sobrecarga em programas de computação quando os dados são trocados na memória do processador. A partir disso, é apresentada uma técnica que reduz a quantidade de trocas de dados, onde essa técnica consiste em manipular dados para o mapeamento de processo lógico. O método proposto para mapeamento mostra melhoras na redistribuição de dados e oferece maior flexibilidade na especificação de quais elementos de dados são distribuídos.

De acordo com (CRUZ; ALVES; NAVAUX, 2010), o mapeamento de processos é uma abordagem muito utilizada em máquinas paralelas para alcançar ganhos de desempenho por meio de melhores usos de recursos como a memória cache e interconexões de rede. Nesse mesmo artigo, os autores buscam melhorar o desempenho de aplicações paralelas que utilizam memória compartilhada que são executadas em máquinas UMA, com a proposta de um método para analisar a memória cache sem a necessidade de conhecer a aplicação. Os resultados foram positivos, obtendo-se 42% de melhora em relação ao mapeamento do sistema operacional.

Considerando a busca constante por alto desempenho e, na maioria das vezes, a preocupação com o consumo elevado de energia, este trabalho reúne os dois conceitos acima: NoCs e Mapeamento de Processos. Na próxima seção é possível encontrar trabalhos que relacionam esses dois conceitos.

2.4 Trabalhos Correlatos

Os trabalhos correlatos mostram os problemas e a importância de fazer um mapeamento de processos em NoCs para alcançar um desempenho desejável. Ou seja, o mapeamento explora a escalabilidade de uma NoC para reduzir o tráfego na rede e, com isso, aumentar a *performance* e reduzir o consumo de energia.

A topologia da rede é um fator importante para o desempenho de uma rede. Por esse motivo, Murali e Micheli (2004) apresentam uma ferramenta chamada SUNMAP para selecionar a melhor topologia para uma determinada aplicação que produz um mapeamento de processos adequado para a combinação, ou seja, automatiza etapas importantes de uma rede-em-chip que são a escolha da melhor topologia e o mapeamento mais adequado à topologia escolhida. A ferramenta tem como objetivos principais minimizar os atrasos nas comunicações, diminuir o consumo de energia com a redução da área de comunicação e minimizar as restrições de largura de banda. O SUNMAP também suporta diferentes funções de roteamento para melhorar a comunicação.

Considerando que mapeamento no contexto NoC é um problema, Ascia, Catania e Palesi (2004) apresentam mapeamentos de dados para NoC baseadas em CMPs (*Chip Multiprocessors*), cujo objetivo é reduzir a distância entre o núcleo de origem e o núcleo de destino. Nessa abordagem, leva-se em consideração que mapeamentos estáticos podem não funcionar bem para aplicações de vários segmentos que passam por diferentes fases de exceção e, em cada fase, com padrões de acesso de dados potencialmente diferentes. Nas experiências realizadas, foi possível analisar o mapeamento proposto e o de dados de forma isolada, bem como de forma integrada.

O mapeamento do espaço em uma arquitetura de rede on-chip pode ser explorado com vários objetivos. Em (ASCIA; CATANIA; PALESI, 2004), a abordagem é uma busca para obter os mapeamentos que otimizam o desempenho e o consumo de energia. Foi utilizado um simulador orientado a eventos de rastreamento que torna possível obter efeitos dinâmicos importantes com impactos significativos sobre o mapeamento. A avaliação foi realizada com tráfego sintetizado e aplicações reais (um codificador/decodificador MPEG-2) que confirmam a precisão e escalabilidade da abordagem.

De acordo com (CALAZANS, et. al., 2005), o problema de mapeamentos para arquiteturas com n núcleos, consiste em encontrar uma associação de cada núcleo de uma rede de tal forma que o custo seja minimizado. Então, apresenta-se uma dependência de comunicação e um modelo de computação (CDCM) para capturar o tempo e o volume de comunicação de uma aplicação. Os resultados são comparados com um modelo proposto em (HU; MARCULESCU, 2003) onde explica-se o volume de cada canal testado. Nesta comparação, chegou-se a conclusão que o modelo CDCM é mais eficiente, reduzindo o tempo médio de execução em 40% e o consumo de energia em 20%.

Em (CARVALHO, E. C.; N. MORAES, 2007), os autores apresentam cinco heurísticas para o mapeamento de tarefas dinâmicas. Aplicações sintéticas foram modeladas pela teoria de grafos usando SystemC para realizar o experimento em uma topologia em malha 8x8. A estratégia é baseada em um processador Manager, e a NoC é dividida em dois tipos: tarefas de hardware e tarefas de software. O principal resultado mostra uma

redução da ocupação dos links interno da NoC.

O impacto da contenção de rede no mapeamento de aplicativo é apresentado por Chou e Marculescu (CHOU; MARCULESCU, 2008). A ideia é usar uma formulação de programação linear com base no mapeamento de contenção conhecida a fim de minimizar a disputa intertelha. Diferentes tamanhos de NoCs (de 3x3 para 5x5 topologias de malha) foram verificados com uma heurística de mapeamento usando a linguagem de programação C. As cargas de trabalho foram baseadas em aplicações reais e sintéticas, como o decodificador MPEG4, e os resultados mostraram melhoria da taxa de transferência para diversos casos.

O trabalho de Tornero e outros (2008) apresenta um mapeamento topológico baseado na comunicação consciente. A estratégia é correlacionar modelo de rede com o desempenho real da rede. Os resultados mostraram melhor desempenho em termos de latência e consumo de energia que outras técnicas.

Outra abordagem de mapeamento em NoCs pode ser vista em (CHEN et. al., 2008), onde é proposto um compilador para mapeamentos de aplicações em arquiteturas de multi-processadores em chip (CMP). O mapeamento proposto tenta otimizar a localização dos acessos de dados, e pode ser benéfico nas perspectivas de desempenho. O compilador tem quatro etapas principais: o agendamento de tarefas, o mapeamento do processador, o mapeamento de dados e roteamento de pacotes. Na primeira etapa, o código do aplicativo é paralelizado e as linhas paralelas resultantes são atribuídas aos processadores virtuais. A segunda etapa implementa um mapeamento virtual de processador para processador físico com objetivo de garantir que os segmentos que já se espera uma frequente comunicação uns com os outros sejam atribuídos ao processador mais próximo possível. Na terceira etapa, os elementos de dados são mapeados para memórias associadas aos nós CMP. O principal objetivo desse mapeamento é colocar um determinado item de dados em um nó que está perto de nós que irão acessá-lo. O último passo determina os caminhos (entre memórias e processadores) para os dados de maneira eficiente. O resultado experimental do compilador mostra que este modelo reduz claramente o consumo de energia das aplicações testadas de forma significativa.

Uma proposta de mapeamento de tarefas baseado em uma comunicação conhecida foi proposta em (LEE; C; HWANG, 2010). A abordagem é focada em maior prioridade de acesso à memória compartilhada quando a tarefa tem uma grande quantidade de comunicação. As cargas de trabalho são H.264, Motion-JPEG e decodificadores de MP3, e transformações 2D. O algoritmo proposto reduziu o número de ciclos de *clock*, o consumo de energia e custos de comunicação para transferência de dados.

Alguns algoritmos de mapeamento são propostos por Marcon e outros (2010), onde os autores avaliam o desempenho em relação ao consumo de energia. Entre os algorit-

mos apresentados estão métodos de busca exaustiva e heurísticas estocásticas e realiza algumas combinações destes. Os algoritmos apresentados são: Busca Exaustiva (ES - do inglês *Exhaustive Search*), *Simulated Annealing* (SA), *Tabu Search* (TS) e duas heurísticas gulosas: *Largest Communication First* (LCF) e *Incremental* (GI). A partir desses, os autores ainda criaram três abordagens mistas. Com os resultados, os autores comprovaram que o uso da busca exaustiva é inviável no caso de NoCs grandes e que os melhores resultados, considerando tempo de execução e consumo de energia, foram alcançados com a combinação entre os algoritmos LCF, GI e a implementação modificada do SA.

Outro trabalho que considera o mapeamento uma questão importante a ser relevada em NoCs é o de Ost e outros(2011). Os autores apresentam uma integração entre três heurísticas de mapeamento dinâmico: *Nearest Neighboor* (NN), *Dependencies-Neighborhood* (DN) e *Lower Energy Consumption* baseado na *Dependencies-Neighborhood* (LEC-DN). A primeira heurística considera apenas a proximidade de um recurso disponível para executar a tarefa, a segunda considera todas as dependências entre as tarefas e a terceira considera a proximidade, em números de saltos, e o volume de comunicação entre as tarefas. A modelagem proposta forneceu resultados precisos, porém a avaliação da heurística é limitada pelo grande tempo de simulação. Portanto é necessário uma estrutura de mais alto nível para a implementação e validação de novas heurísticas de mapeamento dinâmico. Vale lembrar que o mapeamento dinâmico é realizado em tempo de execução, o que pode aumentar o tempo de processamento ou de simulação de uma aplicação.

Diferente da maioria dos trabalhos, esta dissertação propõe a avaliação de três heurísticas de mapeamento de processos em redes-em-chip com intuito de avaliar como diferentes abordagens de mapeamento influenciam no desempenho final da aplicação e, que dada uma carga de trabalho, é possível analisar e encontrar a melhor técnica possível para melhorar o desempenho. Por meio de estudos anteriores (AVELAR et. al., 2011) (OLIVEIRA et al., 2011), pode-se verificar que tal questão procede, ou seja, a carga tem grande importância no momento de escolha de uma heurística e de uma técnica de mapeamento. Ou seja, quanto mais conhecimento da aplicação, o resultado pode se aproximar mais da solução ótima.

Outro ponto apresentado neste trabalho que diferencia dos demais trabalhos e que contribui para os estudos de mapeamento de processos é a implementação e utilização do algoritmo de clusterização K-means como estratégia de mapeamento. Esse algoritmo é comumente utilizado em mineração de dados para obter a divisão do espaço de dados que deseja trabalhar. No processo de geração do mapeamento, o K-means foi associado à uma função de balanceamento para melhor distribuição dos processos nos *clusters* (explicada no Capítulo 4). A apresentação desse algoritmo como possível alternativa para minimizar o problema de mapeamento é considerada relevante para a melhoria e continuidade dos estudos de mapeamento de processos.

3 HEURÍSTICAS AVALIADAS

Os três algoritmos avaliados neste trabalho são: BRD (Biparticionamento Recursivo Dual), Guloso e K-means. O BRD é um algoritmo que utiliza o conceito de divisão-e-conquista para gerar o mapeamento. O Guloso considera a melhor solução local para o problema, o que não garante a melhor solução global. E o K-means é um algoritmo de clusterização que foi adaptado para gerar um mapeamento. Após a aplicação desses algoritmos nas cargas de trabalho, serão avaliados os resultados de cada um no Capítulo 5.

3.1 Biparticionamento Recursivo Dual - BRD

O algoritmo heurístico BRD (Biparticionamento Recursivo Dual) é baseado na técnica de divisão-e-conquista para gerar, recursivamente, uma solução de mapeamento, dado um grafo que representa a aplicação e outro representando a arquitetura (PELLEGRINI,2008). O objetivo do algoritmo é minimizar o custo de comunicação entre os processos.

A aplicação é representada por um grafo valorado e não direcionado cujos vértices indicam os processos da aplicação paralela e as arestas representam as comunicações entre os processos. Os pesos dos vértices indicam a capacidade de processamento para executar aquele processo e os pesos das arestas representam o custo de comunicação entre os processos. Neste trabalho, os vértices não serão valorados e o peso das arestas indicam o custo de comunicação (descrito no Capítulo 4) entre dois processos. A arquitetura também é representada por um grafo cujos vértices representam os núcleos de processamento e as arestas representam os *links* de comunicação entre os núcleos. Nota-se que o grafo da arquitetura também simboliza a topologia da rede, para este caso, uma *Mesh*.

O algoritmo é descrito em (PELLEGRINI,2008) e pode ser visualizado na Figura 5. No início, ele considera todos os núcleos de processamento, ou seja, a arquitetura da rede (no caso deste trabalho) destinada ao mapeamento. A cada passo o algoritmo divide o domínio (conjunto de núcleos) ainda não processado em dois subdomínios separados e chama o algoritmo de biparticionamento para separar os subconjuntos de processos associados com o domínio por meio dos subdomínios.

Esse método está implementado em uma ferramenta chamada Scotch, desenvolvida por (PELLEGRINI,2008) e utilizada neste trabalho para gerar e avaliar o mapeamento de processos resultante da ferramenta.

Figura 5 – Algoritmo de Biparticionamento Recursivo Dual - BRD.

```
function BRD-MAPPING(D, P)
    if |P| = 0 then
        return
    if |D| = 1 then
        map P on D
        return
    (D0, D1) ← PROCESSOR-BIPARTITION(D)
    (P0, P1) ← PROCESS-BIPARTITION(P, D0, D1)
    BRD-MAPPING(D0, P0)
    BRD-MAPPING(D1, P1)
```

Fonte: (PELLEGRINI,2008)

3.2 Algoritmo Guloso

Um algoritmo Guloso se baseia na busca da melhor solução local para ser usada na busca de uma solução global. De acordo com (CHOU; MARCULESCU, 2008), um algoritmo guloso, dado um critério de escolha, a cada passo, analisa e seleciona um subconjunto de soluções sem considerar escolhas anteriores. Portanto, a solução global dependerá das subsoluções encontradas a cada passo.

A proposta de uma heurística gulosa para gerar o mapeamento de processos utilizada e comparada neste trabalho foi implementada com base na proposta de (OLIVEIRA et al., 2011). A modelagem do problema baseou-se na teoria dos grafos e foi necessário um breve conhecimento das cargas de trabalho. Assim, foram modelados dois grafos: i) um representa a topologia da rede, onde os núcleos de processamento são representados pelos vértices e os links que interligam os roteadores são representados pelas arestas, ii) o outro representa as comunicações entre os processos, sendo que os vértices simbolizam os processos, as arestas as comunicações entre dois processos e o peso das arestas representa o custo de comunicação entre o processo origem e o destino. Para a análise do algoritmo, o critério considerado é o custo das comunicações (peso das arestas) descrito posteriormente.

A ideia central do algoritmo é identificar o processo que se comunica mais e atribuí-lo ao núcleo que possui o maior número de *links*. Feito isso, os passos seguintes são: escolher o processo vizinho que possui o maior número de comunicações com o que foi mapeado e que ainda não está mapeado e escolher o núcleo, ainda não mapeado, mais próximo do último mapeado. O algoritmo pode ser visualizado na Figura 6.

Figura 6 – Algoritmo Guloso.

```
function GREDY-MAPPING(G_p, G_n)
    p ← VERTEX-HIGHEST-WEIGHT(G_p)
    n ← VERTEX-HIGHEST-DEGREE(G_n)
    map[n] ← p
    last-p ← p
    last-n ← n
    while there are vertexes to be mapped do
        p ← ADJACENT-VERTEX-HIGHEST-WEIGHT(G_p)
        n ← ADJACENT-VERTEX-HIGHEST-DEGREE(G_n)
        map[n] ← p
        last-p ← p
        last-n ← n
    return map
```

Fonte: (OLIVEIRA et al., 2011)

A heurística gulosa tem como base a busca da melhor solução local, considerando o critério de custo das comunicações, já que o objetivo é minimizar a latência de comunicação. Porém, não se pode assegurar que é a melhor solução global para o problema de mapeamento de processos.

3.3 *K-means*

Essa solução baseia-se no algoritmo de clusterização *K-means*, adotando como estratégia central para a redução do custo de comunicação na NoC, o agrupamento dos processos que se comunicam muito entre si em conjuntos de núcleos adjacentes denominados *clusters*. Para tanto, o algoritmo sucessivamente reagrupa os processos utilizando o conceito de distância euclidiana mínima. Assim como o Guloso, o K-means é um algoritmo heurístico e que não há garantias que irá convergir para uma solução global ótima e o resultado pode ser influenciado pelo método de inicialização dos *clusters*.

Esse algoritmo, mostrado na Figura 7, recebe como entrada uma matriz C, que discrimina o custo de comunicação de cada processo para todas as demais; o número de *clusters* k; e a distância de comunicação mínima d, desejável entre os processos de um mesmo *cluster*. A saída do algoritmo, por sua vez, é um vetor M que indica o mapeamento dos processos nos núcleos de processamento. O funcionamento do algoritmo é explicado a seguir.

Inicialmente, os processos são distribuídos de maneira aleatória entre os k *clusters* (CLUSTERS-RANDOM-POPULATE()). Em seguida, calcula-se o centroide de cada *cluster* a partir da média dos custos de comunicação dos processos que o constituem (CLUSTERS-COMPUTE-MEANS()). Depois, os processos são reagrupadas considerando sua distância euclidiana para os centroides de cada *cluster*, sendo atribuídas ao *cluster* mais próximo

(CLUSTERS-REPOPULATE()).

Enquanto todos os processos encontrarem-se a uma distância maior que d do centroide do seu respectivo *cluster* e o conjunto de processos que constitui um *cluster* mudar, o algoritmo continua a recalcular os centroides de cada *cluster* e reagrupar todas os processos. No entanto, quando este critério não for mais verdadeiro o processo de clusterização termina.

Nesse momento, cada *cluster* agrupa os processos que mais se comunicam entre si, mas podem não refletir uma clusterização física dos núcleos de processamento. Por exemplo, considere uma topologia *mesh* 4×8 e 4 *clusters*. Ao final do processo de clusterização pode-se obter *clusters* de tamanho 3, 3, 3 e 23. Entretanto, fisicamente um *cluster* representa um conjunto de núcleos próximos, mas é evidente que um *cluster* de 23 núcleos não expressa essa característica para essa NoC.

Para resolver esse impasse, os *clusters* devem ser balanceados de forma que, ao final do processo, as seguintes condições sejam satisfeitas: (I) todos os *clusters* possuam uma população de mesmo tamanho e (II) o balanceamento seja ótimo, isto é, não exista nenhum outro balanceamento que implique em um custo de comunicação menor que o balanceamento obtido. Tal balanceamento pode ser alcançado baseando-se na observação de que ele corresponde, em essência, ao problema de *matching* mínimo em um grafo bipartido valorado onde: os vértices do grupo A correspondem aos processos; os vértices do grupo B aos *clusters*; e o valor das arestas a distância dos processos aos *clusters*. Sendo assim, utilizou-se o algoritmo proposto por Bertserkas (BERTSEKAS, 1988) para efetuar o balanceamento dos *clusters* (AUCTION-BALANCING())

Figura 7 – Algoritmo de clusterização K-means.

```
function KMEANS-MAPPING(C, k, d)
    CLUSTERS-RANDOM-POPULATE()
    CLUSTERS-COMPUTE-MEANS()
    repeat
        CLUSTERS-REPOPULATE()
        CLUSTERS-COMPUTE-MEANS()
    until distance > d and CLUSTERS-CHANGED()
    map ← AUCTION-BALANCING(c, k)
    return map
```

Fonte: Do autor

4 MÉTODO DE AVALIAÇÃO PROPOSTO

Neste capítulo são apresentados os procedimentos realizados antes da execução dos testes, a descrição das cargas de trabalho utilizadas e do ambiente de simulação e a descrição das decisões que foram necessárias para realizar as simulações.

Os algoritmos apresentados no Capítulo 3 foram utilizados como heurísticas de mapeamento de processos com alocação de cada processo em cada núcleo da arquitetura testada. As cinco aplicações testadas possuem 32, 64, 128 e 256 processos e as arquiteturas simuladas possuem 32, 64, 128 e 256 núcleos. Logo o mapeamento é 1:1, o que significa que cada processo será alocado em um núcleo ou cada núcleo terá um processo alocado.

4.1 Cargas de Trabalho

Para melhor representar o comportamento de uma aplicação de propósito geral, o requisito para a escolha das cargas de trabalho foi obter cargas que tivessem padrões de comunicação diferentes e que representassem algumas situações/problemas comuns na computação. Para melhor exploração e visualização do comportamento da rede, cargas paralelas seriam ideais.

Dadas essas características, escolheu-se aplicações paralelas do *NAS Parallel Benchmark* (NPB) ((NAS), 2010) para ser a base dos *traces* utilizados neste trabalho. Cada aplicação utilizada representa um problema diferente de computação. Com isso espera-se uma diversidade de características entre elas. O NPB é baseado em diferentes modelos de programação: Multi-processamento aberto (*OpenMP - Open Multi-processing*), para memória compartilhada, interface de passagem de mensagem (*MPI - Message Passing Interface*), para memória distribuída, e sequencial. A escolha de um dos modelos pode depender de um grande número de variáveis, em alguns casos não controladas pelo sistema em geral, que exercem grande influência no desempenho da aplicação. Portanto, neste trabalho não foi avaliado questões de memória compartilhada e considerou-se apenas as NoCs que utilizam passagem de mensagem.

O *NAS Parallel Benchmark* utilizado consiste de *kernels* simulados e são aplicações baseadas em MPI. Para a análise de diferentes contextos, esse *benchmark* possui diferentes classes ou tamanhos diferentes de aplicações (por exemplo, S, W, A, B, C e D). Neste trabalho, escolheu-se a classe A que requer menos recursos a serem simulados, em que o tempo de simulação é uma restrição para obter os resultados. As aplicações avaliadas neste trabalho são: CG (*Conjugate Gradient*), EP (*Embarassingly Parallel*), FT (*Fast Fourier Transform*), IS (*Integer Sort*), MG (*Multigrid*), cujas principais funcionalidades são descritas abaixo ((NAS), 2010):

- CG (*Conjugate Gradient*): possui comunicações irregulares entre os processos. Utiliza um método de gradiente conjugado para encontrar uma estimativa do maior valor de uma matriz simétrica positiva com padrões randômicos diferentes de zero;

- EP (*Embarassingly Parallel*): não possui uma comunicação significante entre os processos. Essa aplicação gera pares gaussianos randômicos e tabula-os em sucessivos anéis e exige, ao final, a combinação de somas de vários processos;

- FT (*Fast Fourier Transform*): essa aplicação soluciona, numericamente, uma equação diferencial utilizando transformadas de Fourier em 3 dimensões. Existem muitas comunicações entre os processos deste método com o objetivo de efetuar operações como transposição de vetores. Realiza, também, um teste de desempenho de comunicações de longas distâncias;

- IS (*Integer Sort*): com o objetivo de testar tanto a velocidade da computação de inteiros quanto o desempenho das comunicações, essa aplicação utiliza um algoritmo para gerar chaves *pseudo* aleatórias e depois efetua operação de ordenação de números inteiros. Essa aplicação é bastante utilizada em códigos de métodos de partículas;

- MG (*Multigrid*): essa aplicação mostra uma simplificação do *kernel* 3D *multigrid* para obter uma solução discreta aproximada para o problema de Poisson. Realiza muitas comunicações entre os processos, pois executa testes de comunicação de dados entre grades de pequenas e longas distâncias.

4.1.1 Arquivos de teste baseados no NPB

Para caracterizar o comportamento da NoC e testar sua escalabilidade, as aplicações paralelas do NPB descritas anteriormente serviram de base para a criação de outros arquivos para a realização dos testes. Essas aplicações foram executadas em uma plataforma real, gerando arquivos de rastro usando a biblioteca MPE (*Multi-Processing Environment*) do MPICH (distribuição MPI) para redes de 32, 64, 128 e 256 processos, com o objetivo de obter todas as informações referentes à comunicação entre os núcleos. Os arquivos de rastro gerados são as entradas de dados para NoCs de 32 (4x8), 64 (8x8), 128 (8x16) e 256 (16x16) núcleos simuladas em um simulador de NoC, chamado TOPAZ (ABAD, 2012), descrito mais adiante.

Para gerar os arquivos de rastro citados foi utilizado um *cluster* homogêneo do Programa de Pós-Graduação em Informática da PUC Minas. Ele é composto por 10 nós interconectados por um *Switch Gigabit Ethernet*. Cada nó do *cluster* contém um *chip* Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz e 4GB de memória RAM. Esse

processador é composto por 4 núcleos de processamento que compartilham uma memória *cache* de 4MB.

Considerando o objetivo desta dissertação de avaliar os mapeamentos de processos e por questão de adequação ao simulador, os *traces* fornecidos pelo NPB possuem algumas características que foram modificadas para a realização dos testes. Para exemplificar, a Figura 8 mostra um trecho de um *trace* gerado pelo NAS antes das modificações.

Figura 8 – Trecho de *trace* original gerado pelo NAS.

```
2.093937 47 [11 0] 8
2.093937 47 [11 1] 8
2.093937 47 [11 2] 8
2.093937 47 [11 3] 8
2.093937 47 [11 4] 8
2.093937 47 [11 5] 8
2.093937 47 [11 6] 8
2.093937 47 [11 7] 8
2.093937 47 [11 8] 8
2.093937 47 [11 9] 8
2.093937 47 [11 10] 8
2.093937 47 [11 11] 8
2.093937 47 [11 12] 8
2.093937 47 [11 13] 8
2.093937 47 [11 14] 8
2.093937 47 [11 15] 8
```

Fonte: Dados da pesquisa.

Cada linha do arquivo do NPB possui cinco campos:

- 1º campo: representa o instante de tempo em que o envio ocorre após o início da execução da aplicação. Esse tempo é dado em segundos;
- 2º campo: identificador do tipo de padrão de comunicação coletiva;
- 3º e 4º campos: representados entre colchetes ([ ]), indicam os processos origem e destino da mensagem, respectivamente;
- 5º campo: representa o tamanho da mensagem enviada (dado em *bytes*).

A primeira modificação foi desconsiderar o identificador do tipo de comunicação, uma vez que todas as comunicações serão consideradas e avaliadas independente do seu tipo. Ou seja, julgou-se que este campo não é necessário para as avaliações.

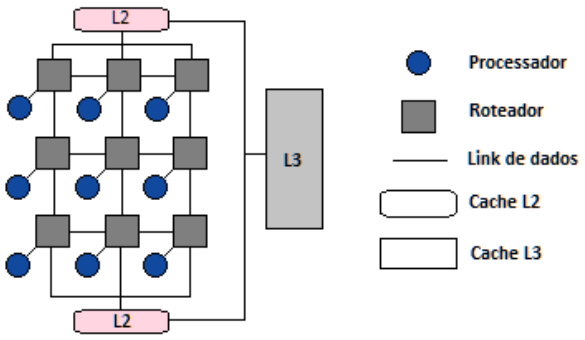
A segunda alteração é feita com os instantes de tempo em que o envio de cada mensagem ocorre. A contagem de tempo no simulador utilizado (TOPAZ (ABAD, 2012)) é feita em ciclos. Portanto foi necessária a alteração deste campo. Para isso, os instantes

dados em segundos foram transformados em ciclos, dada a frequência dos processadores que geram os traços de execução. Logo após, os ciclos foram normalizados para iniciarem no instante 101, que é o primeiro instante disponível após um *warmup* de 100 ciclos. O *warmup* representa uma etapa de funcionamento do simulador para que ele entre em um regime de funcionamento estável.

A terceira modificação, também relacionada aos instantes de tempo, foi nos atrasos permitidos entre os envios das mensagens. Entre um envio e outro, o atraso máximo é de 50 ciclos, ou seja, para iniciar uma nova transmissão em relação à última comunicação, a inatividade é ajustada para 50 ciclos. O novo prazo foi calculado com a premissa de que um período grande de ociosidade pode ser irrelevante para a avaliação dos mapeamentos em um ambiente de simulação.

Outra questão que deve ser avaliada é sobre o tamanho das mensagens enviadas em cada comunicação. Pacotes de dados muito grandes podem gerar gargalos, perdas ou atrasos na entrega. Para evitar um tráfego intenso na rede é interessante que os pacotes tenham limite de tamanho. Para pacotes que ultrapassem o limite suportado pela rede, o núcleo de origem enviará ao destino um pacote contendo o endereço de memória (e.g. L2) de onde o dado se encontra. Assim, o nó destino poderá buscar a informação da qual necessita na memória. Essa interação com a memória originaria uma rede secundária de dados e a arquitetura resultante é ilustrada na Figura 9. Tem-se algumas vantagens ao possuir tamanhos fixos e reduzidos dos pacotes de dados: com pacotes menores é possível utilizar a memória local do núcleo, já que a troca de dados de pacotes muito grandes não são realizadas na rede e sim via memória compartilhada; os *buffers* terão seus projetos facilitados, ou seja, podem haver menos gargalos e a gerência de filas será mais simplificada com menos/menores pacotes; pode-se ter como hipótese que o consumo de energia também será minimizado pela redução de pacotes ou do tamanho dos mesmos trafegando pelos roteadores.

Figura 9 – Ilustração de uma Rede-em-Chip com Memória Compartilhada.



Fonte: Do autor.

No cálculo de custo das comunicações foram utilizados os tamanhos originais dos

pacotes, pois considera-se que processos que trocam mensagens grandes também devam ficar próximos para compartilhar a mesma memória e garantir um desempenho melhor.

Vale ressaltar que o tamanho limite de um pacote depende da rede-em-chip utilizada. Cada projeto de NoC contém seus elementos com suas próprias restrições. Portanto, é importante que uma análise prévia da arquitetura alvo seja feita para estipular esse limite, caso seja necessário e caso haja um projeto que permita a utilização de memória compartilhada.

4.1.2 Custo das Comunicações

Para cada comunicação foi calculado um custo para que a mensagem saia da origem e chegue ao seu destino final. O cálculo foi feito considerando o tamanho do pacote de dados a ser enviado, a quantidade de comunicações entre os dois processos e o número de saltos entre os núcleos de origem e destino. Para calcular o número de saltos, considerou-se núcleos dispostos em uma rede de topologia *Mesh 2D*.

O custo de cada comunicação foi representado na seguinte expressão:

$$\text{Cost}_{o,d} = ((NMsg_{(o,d)} * \sum_{n=0}^{n} Size_{(o,d)}) + (NMsg_{(d,o)} * \sum_{n=0}^{n} Size_{(d,o)})) * Hop_{(o,d)}$$

onde $NMsg$ representa o número de comunicações entre o processo de origem (o) e o destino (d), $Size$ indica o tamanho real da mensagem transmitida e Hop simboliza o número de saltos entre os dois processos.

O número de mensagens enviadas e tamanho da mensagem considera todas as comunicações entre os núcleos de origem e destino. Por exemplo: tem-se uma comunicação entre os núcleos 5 e 10, então os cálculos de quantidade de comunicações e tamanho das mensagens são feitos considerando a comunicação do nó 5 para o nó 10 e do nó 10 para o nó 5.

4.2 Ambiente de simulação

A simulação foi o método escolhido de avaliação dos resultados de mapeamento de processos. O ambiente de simulação de Redes-em-Chip escolhido foi o TOPAZ (ABAD, 2012). Desenvolvido no Departamento de Eletrônica e Computação na Universidade de Catanbria, na Espanha, o TOPAZ é um simulador de rede de propósito geral de interconexão que permite a modelagem de uma ampla variedade de roteadores com diferentes combinações de velocidade e precisão. Para facilitar a compreensão, a extensão e a reutilização, o projeto é orientado a objetos e a sua aplicação é na linguagem C++ e pode ser usado em qualquer plataforma UNIX com um compilador padrão de C++ (ABAD, 2012).

Essa ferramenta também possibilita a integração com o GEM5, um simulador completo de arquitetura (BINKERT,2011).

O TOPAZ possui uma estrutura diversificada e contém componentes que oferecem várias possibilidades de testes. Dentre os componentes fornecidos pode-se citar:

a) Topologias: possui as topologias *Ring*, *Mesh* (2D e 3D), *Torus* (2D e 3D), *Midimew* (2D) e *Square Midimew* (2D).
b) Controle de fluxo: possui métodos como o Bolha, *Wormhole*, Canal Virtual.
c) Padrões de Tráfego: dentre os padrões de tráfego estão o Randômico, Tornado, Bit-Reverso, *Hot-Spot* e *Trace-Based*.
d) Roteadores: dentre os roteadores estão o *Adaptive Bubble*, *Deterministic Bubble*, *Rotary Router*, *Buffered Crossbar* e *Bidirecional Router*.
e) Parâmetros de Configuração: é possível determinar parâmetros como tamanho do *buffer*, tamanho do pacote de dados, limite de atrasos, número de canais virtuais, dentre outros.

O TOPAZ oferece diversas métricas para avaliar o desempenho da rede. Dentre eles estão: quantas mensagens foram geradas, recebidas e injetadas, latência da rede, dos *buffers* e latência total, *throughput*, consumo de memória, desempenho dos canais virtuais, dentre outros.

4.3 Decisões de Simulação e Avaliação

Para avaliar o comportamento da NoC simulada, algumas decisões foram tomadas antes da simulação efetiva das cargas.

Para gerar o mapeamento considerou-se o custo de comunicação entre os processos descrito na seção anterior como critério de decisão. Tanto no algoritmo BRD quanto no algoritmo Guloso, os pesos das arestas representam justamente esse custo de comunicação.

Conforme dito na Seção 4.1.1, é importante que os pacotes tenham tamanhos fixos na execução da aplicação. Portanto, na realização da simulação, os pacotes possuem tamanhos fixos para melhor avaliar o comportamento da rede diante das três heurísticas de mapeamento. Os pacotes variam entre 4, 6, 8 e 10 *flits**. Pode-se obter maiores variações de tamanho de pacotes e alterações nos resultados das simulações quando há a integração entre o TOPAZ e o GEM5. Como este trabalho não aborda esta integração, os testes ficaram restritos à variação citada.

Considerando que as aplicações testadas foram executadas em plataformas *multi-core* variando de 32 a 256 processos, as simulações foram realizadas em NoCs de tamanho

* *Flit - Flow Control Unit*

4x8 (32 núcleos), 8x8 (64 núcleos), 8x16 (128 núcleos) e 16x16 (256 núcleos), sendo um processo para cada núcleo. O TOPAZ possui arquivos de configuração onde é possível definir características das redes-em-chip simuladas. Para os testes, criou-se, nesses arquivos de configuração, redes com os tamanhos citados, com topologia mesh e *buffers* de 11 *flits*. Foi utilizado um roteador simples para a topologia mencionada, que usa uma política de roteamento chamada DOR (*Dimension OrdereRouting*) e o comportamento do *buffer* é controlado pela política CT (*Virtual Cut Through*). Os tamanhos dos pacotes (citados anteriormente) são definidos no momento da simulação, assim como os arquivos de rastro que definem o tráfego da rede. Das métricas fornecidas, as avaliadas serão as latências de *buffer*, de rede e total e a distância média percorrida pelos pacotes.

5 AVALIAÇÃO DOS RESULTADOS

O objetivo principal de um mapeamento de processos é melhorar o desempenho da aplicação uma vez que os processos são alocados nos núcleos considerando algumas características. Essa alocação pode fazer com que a comunicação entre dois processos seja mais rápida, ou seja, o mapeamento pode reduzir o número de saltos com a redução de latência das mensagens e, consequentemente, a latência total da rede. A latência é o tempo que a mensagem leva para sair da origem e chegar à seu destino final. Considerando esses fatos, as métricas avaliadas nesta dissertação são a latência da rede e dos *buffers*, latência total das mensagens e a distância média percorrida pelos pacotes na execução de cada algoritmo.

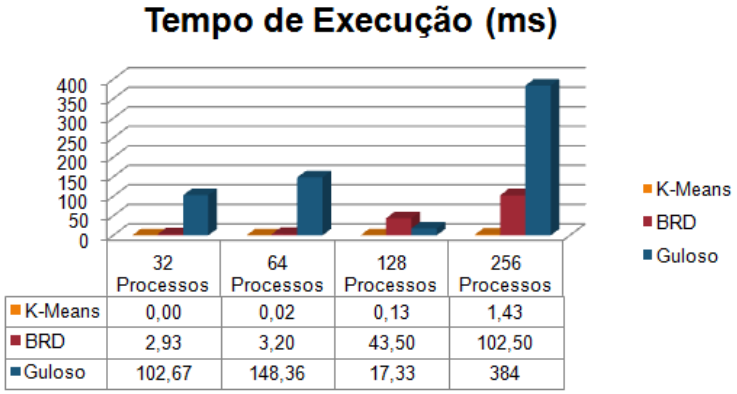
Como as aplicações variam de quantidade de processos (32, 64, 128 e 256 processos), para cada quantidade de processo/núcleo existe um mapeamento diferente. O mapeamento não é o mesmo para cada um devido ao cálculo de custo de comunicação que considera variáveis que possuem valores diferentes em cada arquivo de teste: são considerados o número de saltos, quantidade de comunicação entre os processos e tamanho dos pacotes, variando de acordo com o *trace*.

Este capítulo mostra os resultados dos testes, em forma de gráficos, e a avaliação do comportamento/desempenho da rede para as três heurísticas e as cinco cargas de trabalho utilizadas. Para simplificar a visualização dos resultados, cada figura contém quatro gráficos referentes aos testes com os diferentes tamanhos de pacotes e quantidade de processos. Este capítulo também contém a avaliação dos tempos de execução de cada algoritmo para exemplificar a complexidade de tempo de cada um.

5.1 Tempo de execução dos algoritmos

Na computação paralela e de alto desempenho, uma questão fundamental para ser analisada é o tempo de execução das aplicações. Por isso, pensando na *performance* geral de uma solução, ou seja, considerando que o mapeamento estático é executado antes da execução efetiva da carga de trabalho, calculou-se a média de tempo de execução dos algoritmos de geração dos mapeamentos para cada quantidade de núcleo/processos. Este tempo médio de execução dos três algoritmos, medido em milissegundos, é apresentado no gráfico da Figura 10.

Figura 10 – Gráfico de tempo de execução dos algoritmos



	32 Processos	64 Processos	128 Processos	256 Processos
K-Means	0,00	0,02	0,13	1,43
BRD	2,93	3,20	43,50	102,50
Guloso	102,67	148,36	17,33	384

Fonte: Dados da pesquisa.

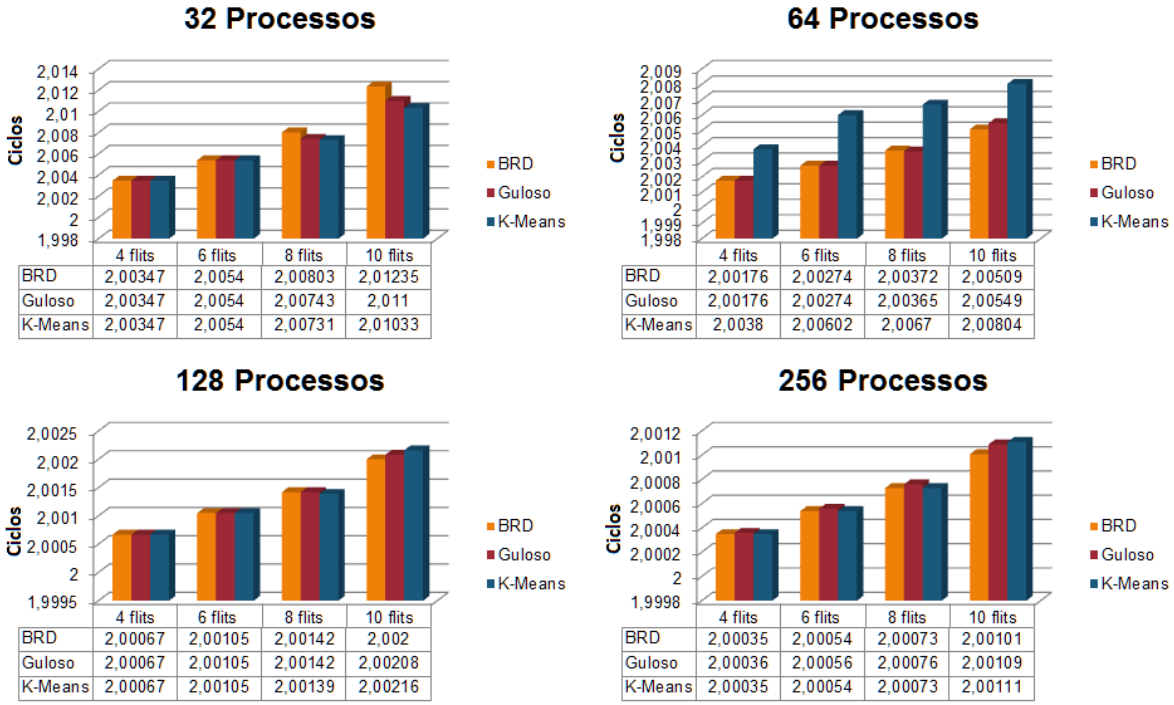
Pode-se notar o desempenho superior do K-means com relação aos demais algoritmos. Porém, esse fato não inviabiliza o uso dos demais algoritmos. É importante observar também que a quantidade de dados a serem processados pode ser de grande importância nessa avaliação: quanto mais dados, mais tempo será necessário para a execução do algoritmo.

5.2 Aplicação CG

A aplicação CG possui um padrão de comunicação *unicast*, ou seja, as comunicações coletivas, de modo geral, são 1:1 e grande parte das comunicações são realizadas entre nós adjacentes (OLIVEIRA, 2012), ou seja, essa aplicação possui, em sua maioria, comunicações ponto-a-ponto de forma irregular. A desvantagem de ter comunicações entre nós adjacentes antes do mapeamento é que após a realização do mapeamento, os processos que já se comunicavam com nós próximos podem ser alocados em núcleos não muito próximos como antes. Isso pode acontecer devido ao cálculo de custo de comunicação feito antes dos mapeamentos que, além de considerar o número de saltos, considera a quantidade de comunicações e o tamanho dos pacotes enviados em cada transmissão.

Os gráficos da Figura 11 mostram os resultados considerando a latência dos *buffers*. Para essa métrica não houve um algoritmo que pudesse ser destacado, pois as diferenças dos resultados não passam de 1%. Ou seja, mesmo variando o tamanho da rede (e quantidade de processos) e o número de *flits*, não existe uma discrepância nos resultados. Isso ocorre devido ao elevado número de transmissões que a aplicação CG possui, que resulta em um comportamento homogêneo de ocupação da rede e consequentemente dos *buffers*, após o mapeamento. O tempo que os pacotes ficaram no *buffer* tende a ser parecido mesmo com diferentes mapeamentos.

Figura 11 – Gráficos de latência do *buffer* da aplicação CG



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	2,00347	2,0054	2,00803	2,01235
Guloso	2,00347	2,0054	2,00743	2,011
K-Means	2,00347	2,0054	2,00731	2,01033

64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	2,00176	2,00274	2,00372	2,00509
Guloso	2,00176	2,00274	2,00365	2,00549
K-Means	2,0038	2,00602	2,0067	2,00804

128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	2,00067	2,00105	2,00142	2,002
Guloso	2,00067	2,00105	2,00142	2,00208
K-Means	2,00067	2,00105	2,00139	2,00216

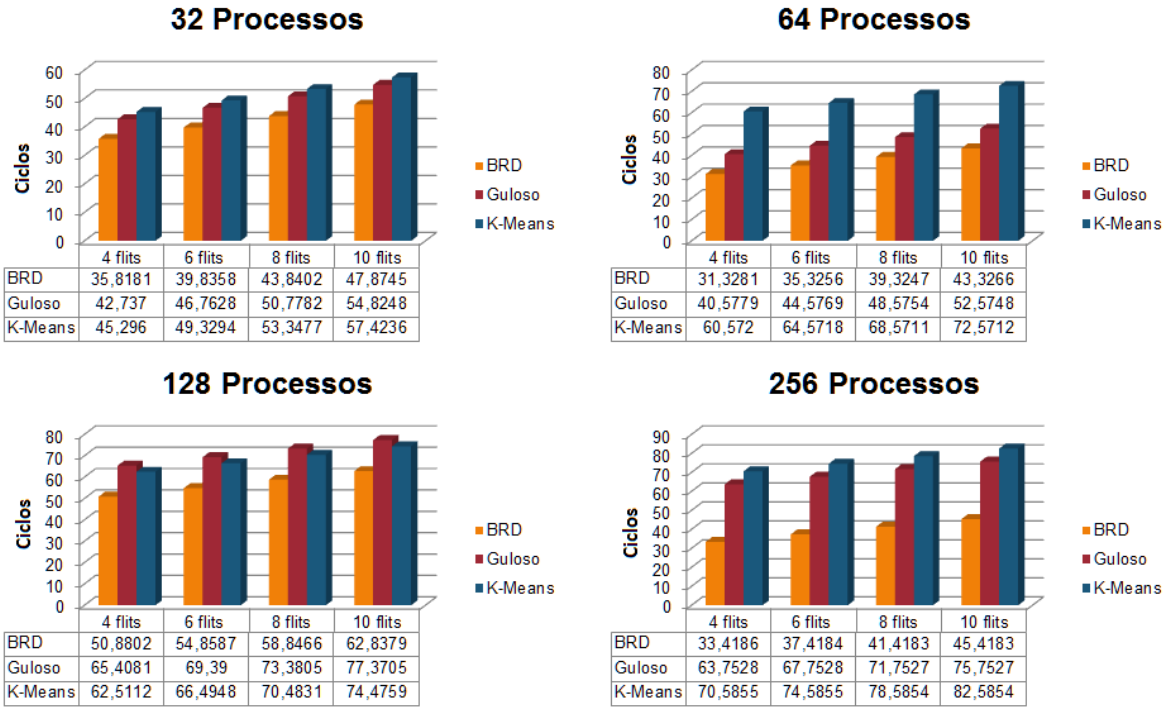
256 Processos	4 flits	6 flits	8 flits	10 flits
BRD	2,00035	2,00054	2,00073	2,00101
Guloso	2,00036	2,00056	2,00076	2,00109
K-Means	2,00035	2,00054	2,00073	2,00111

Fonte: Dados da pesquisa.

Conforme é apresentado nos gráficos da Figura 12, a latência da rede aumenta com o crescimento do tamanho do pacote. Isso ocorre pois o aumento do tamanho do pacote inunda ainda mais a rede, podendo gerar mais gargalos. A distinção entre os resultados de latência para cada algoritmo e tamanho de rede deve-se ao fato de mesmo com a rede saturada, a diferença no tempo é maior, pois existe a influência da quantidade de saltos, ou seja, o caminho percorrido pelos pacotes são divergentes.

De maneira geral, o algoritmo que apresentou melhores resultados foi o BRD, em seguida o Guloso, sendo esse cenário diferente apenas para a rede com 128 núcleos/processos. Nos testes realizados com 32 processos, o melhor resultado foi obtido utilizando o algoritmo BRD, sendo ele 14,51% e 19,94% melhor que utilizando o Guloso e o K-means, respectivamente, no caso de pacotes de 10 *flits*, por exemplo. O contexto é semelhante para os resultados em uma rede de 64 processos. O BRD atingiu melhores resultados (também na simulação de pacotes com 10 *flits*): 21,43% melhor que o Guloso e 67,49% superior com relação ao K-means. Os resultados para os testes com 128 processos mostram uma situação um pouco diferente: a latência da rede ainda foi menor com a utilização do BRD, porém em segundo o K-means, com diferença de 18,52% e depois o Guloso com uma desigualdade de 23,03% comparado com o BRD, no caso de pacotes de 10 *flits*. Por fim, com 256 processos o BRD mantém o melhor resultado, em seguida o Guloso com diferença de 66,78% e o K-means com 81,83%, nos testes realizados com 10 *flits*. Vale lembrar que a situação é semelhante para as demais variações de *flits*.

Figura 12 – Gráficos de latência das mensagens na rede da aplicação CG



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	35,8181	39,8358	43,8402	47,8745
Guloso	42,737	46,7628	50,7782	54,8248
K-Means	45,296	49,3294	53,3477	57,4236

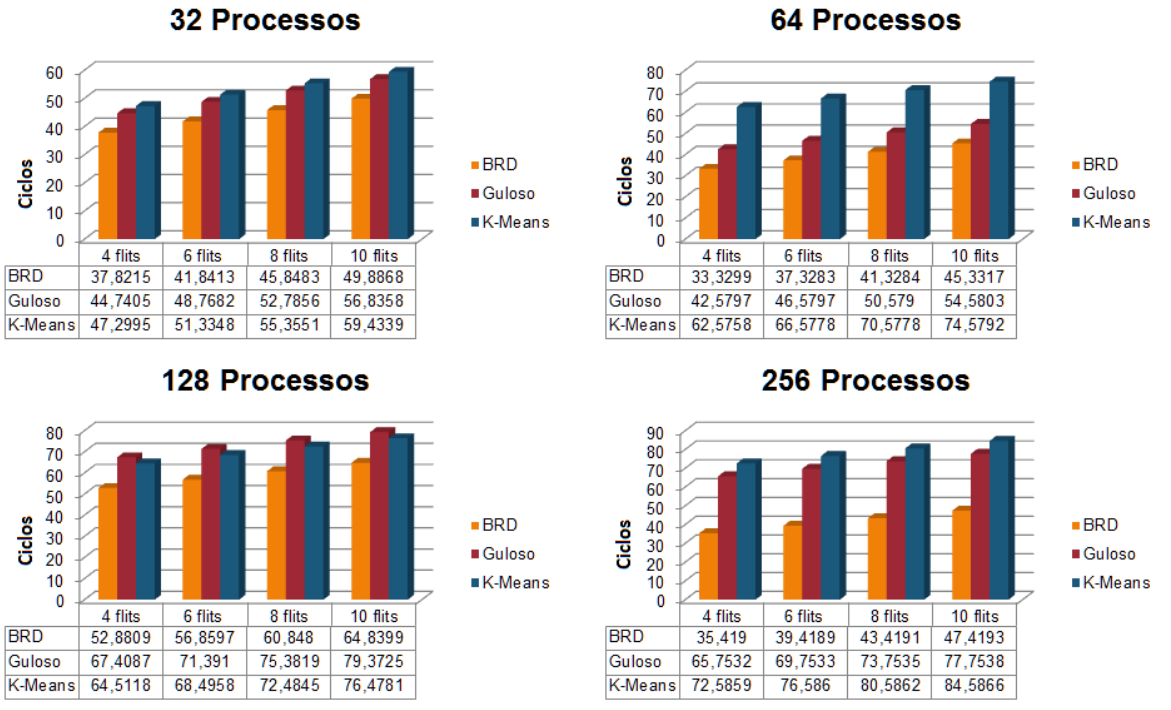
64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	31,3281	35,3256	39,3247	43,3266
Guloso	40,5779	44,5769	48,5754	52,5748
K-Means	60,572	64,5718	68,5711	72,5712

128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	50,8802	54,8587	58,8466	62,8379
Guloso	65,4081	69,39	73,3805	77,3705
K-Means	62,5112	66,4948	70,4831	74,4759

256 Processos	4 flits	6 flits	8 flits	10 flits
BRD	33,4186	37,4184	41,4183	45,4183
Guloso	63,7528	67,7528	71,7527	75,7527
K-Means	70,5855	74,5855	78,5854	82,5854

Fonte: Dados da pesquisa.

Figura 13 – Gráficos de latência total das mensagens da aplicação CG



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	37,8215	41,8413	45,8483	49,8868
Guloso	44,7405	48,7682	52,7856	56,8358
K-Means	47,2995	51,3348	55,3551	59,4339

64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	33,3299	37,3283	41,3284	45,3317
Guloso	42,5797	46,5797	50,579	54,5803
K-Means	62,5758	66,5778	70,5778	74,5792

128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	52,8809	56,8597	60,848	64,8399
Guloso	67,4087	71,391	75,3819	79,3725
K-Means	64,5118	68,4958	72,4845	76,4781

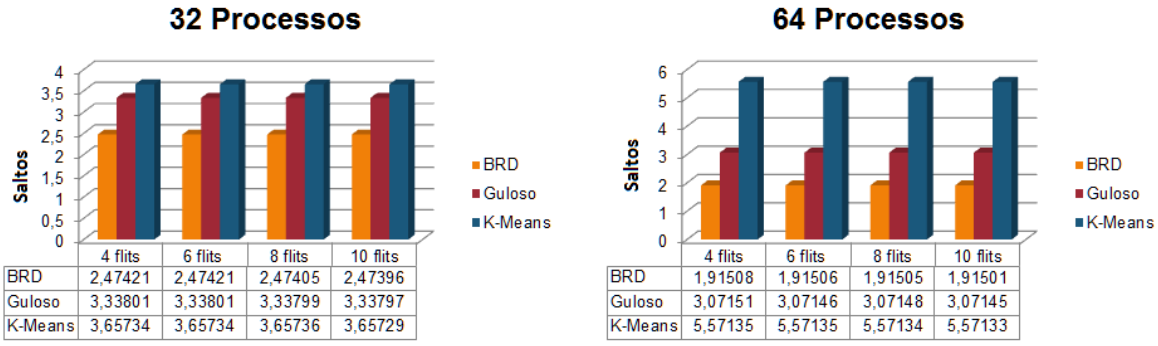
256 Processos	4 flits	6 flits	8 flits	10 flits
BRD	35,419	39,4189	43,4191	47,4193
Guloso	65,7532	69,7533	73,7535	77,7538
K-Means	72,5859	76,586	80,5862	84,5866

Fonte: Dados da pesquisa.

A latência total é dada pela soma da latência do *buffer* e a latência da rede, cujos resultados podem ser visualizados nos gráficos da Figura 13. Com isso, considerando que

a avaliação de latência do *buffer* mostrou uma similaridade entre os resultados, pode-se constatar que a dispersão dos resultados de latência total deve-se pela latência da rede. É importante ressaltar que a latência está relacionada diretamente no mapeamento gerado em cada algoritmo. A partir do custo das comunicações, cada algoritmo gerou um mapeamento diferente para cada quantidade de processos. Isso mostra que o algoritmo BRD conseguiu distribuir melhor os processos para a aplicação CG, diminuindo a quantidade de saltos entre as comunicações. O Guloso, por ser um algoritmo que objetiva a solução local do problema, pode encontrar dificuldades em mapear uma carga irregular. Outro fator do programa CG é que tem-se, originalmente, um número elevado de comunicações entre nós adjacentes. Isso faz com que o número de saltos seja menor entre os núcleos antes do mapeamento. Após o mapeamento, essa questão pode não ser mais verdadeira, pois no cálculo de custo o peso da quantidade de saltos será menor.

Figura 14 – Gráficos de distância média da aplicação CG



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	2,47421	2,47421	2,47405	2,47396
Guloso	3,33801	3,33801	3,33799	3,33797
K-Means	3,65734	3,65734	3,65736	3,65729

64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	1,91508	1,91506	1,91505	1,91501
Guloso	3,07151	3,07146	3,07148	3,07145
K-Means	5,57135	5,57135	5,57134	5,57133

128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	4,35099	4,35082	4,35093	4,35089
Guloso	6,16839	6,16819	6,1683	6,16825
K-Means	5,80606	5,80576	5,80585	5,80574

256 Porcessos	4 flits	6 flits	8 flits	10 flits
BRD	2,17728	2,17727	2,17727	2,17727
Guloso	5,96907	5,96907	5,96907	5,96907
K-Means	6,82314	6,82315	6,82316	6,82316

Fonte: Dados da pesquisa.

A distância percorrida pelos pacotes pode influenciar na latência das mensagens na rede, pois quanto maior a distância, maior pode ser o tempo gasto pelos pacotes para que cheguem até o destino final. Os resultados apresentados nos gráficos da Figura 14 reforçam a relação entre essas duas métricas. Nos testes com 32 processos, o Guloso e o K-means obtiveram resultados inferiores ao BRD: 34,92% e 47,83%, respectivamente. O cenário é o mesmo para os resultados com 64 processos, ou seja, o BRD atingiu uma distância média menor que os demais: com o Guloso a distância foi 60,38% maior e com o K-means foi 190,92% maior. Nos resultados com 128 processos, o K-means apresentou

6,24% de melhora com relação ao Guloso, porém ainda ficando 33,43% atrás do BRD. O cenário volta a se repetir com os testes de 256 processos, sendo o BRD superior aos demais: 174,15% em comparação ao Guloso e 213,38% com relação ao K-means. A avaliação anterior foi feita com os pacotes de 10 *flits* uma vez que não houve alteração significativa nos resultados com a variação do tamanho do pacote.

De modo geral, para a aplicação CG, o mapeamento resultante do uso do BRD obteve melhores resultados que as demais soluções. Isso aconteceu porque neste programa cada processo se comunica com um número reduzido de processos. Tal fato possibilita que os processos sejam agrupados dois a dois de maneira mais fácil, o que pode contribuir para os melhores resultados obtidos com o BRD. Outro fator importante é que grande parte da comunicações já ocorria entre processos adjacentes antes do mapeamento, então o uso dos algoritmos possivelmente aumentaram a distância entre os processos. A alocação de processos feita com o Guloso considera sempre o maior custo de comunicação, porém a estratégia pode não favorecer a alocação dos demais processos e alocando-os em núcleos mais distantes que antes do mapeamento. A heurística por K-means encontrou dificuldades na alocação dos processos na parte do balanceamento dos processos nos *clusters*. Por ser uma carga em que os processos se comunicam com poucos processos, os processos podem ter se concentrado em apenas alguns *clusters*, assim, após o balanceamento, a distância entre eles aumentaram.

5.3 Aplicação EP

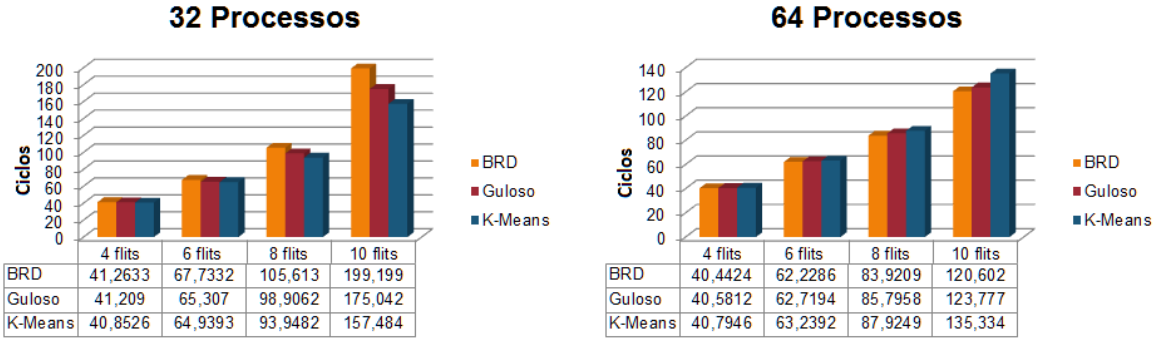
Diferente da aplicação CG, a EP possui um padrão de comunicação *broadcast*, ou seja, possui um padrão regular e é uma aplicação considerada pequena com relação às demais aplicações (OLIVEIRA, 2012). Outra característica dessa aplicação é que ela possui pacotes pequenos trafegando na rede. Considerando essas características, o custo das comunicações terá mais influência do número de saltos, pois há muitas comunicações entre nós distantes.

Diferente de uma comunicação *unicast*, em um padrão *broadcast* todos os núcleos recebem pacotes a todo momento. Esse fato pode justificar o aumento da latência no *buffer* com relação ao CG. Ou seja, como a quantidade de pacotes que estão chegando é maior, logo a tendência é de que esperem mais tempo para serem processados, como pode ser visto nos gráficos da Figura 15. Outro fator importante é o aumento da latência do *buffer* com o aumento do tamanho do pacote. Isso acontece, basicamente, por três motivos: o *buffer* possui um tamanho limitado, o tempo necessário para processar os pacotes é maior e a taxa de chegada de pacotes é elevada.

Nos testes realizados com 32 processos, o algoritmo que apresentou melhores resultados foi o K-means, conseguindo ser 11,14% melhor que o Guloso e 26,48% que o BRD,

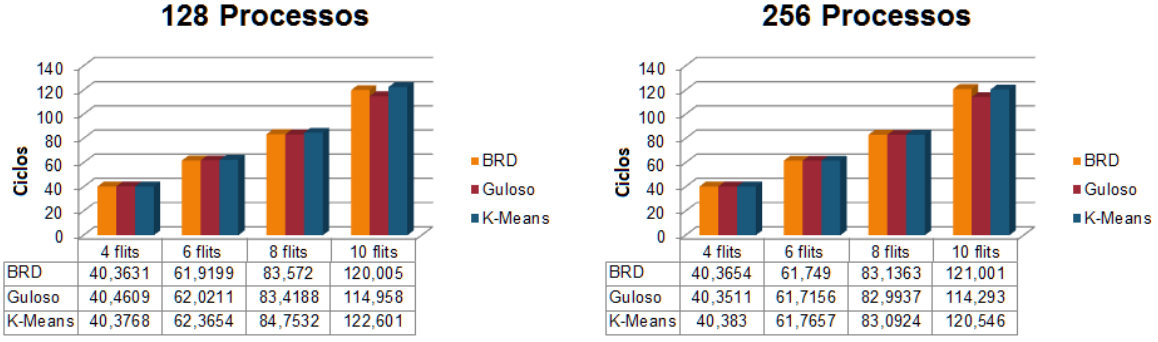
nos testes em que foram transmitidos pacotes de 10 *flits*. Já nos resultados com uma rede de 64 núcleos, o algoritmo que alcançou a menor latência de *buffer* foi o BRD, sendo apenas 2,63% e 12,21% melhor que o Guloso e o K-means, respectivamente, também os testes com pacotes de 10 *flits*. O cenário é diferente nos testes realizados com 128 processos, pois para cada variação de tamanho pacote pode-se perceber uma pequena alteração nos resultados entre os algoritmos. Porém, a diferença entre eles, utilizando pacotes de tamanhos 4, 6 e 8 *flits*, é menor que 1%, sendo considerada quase insignificante. Já para pacotes com 10 *flits*, a diferença é perceptível: o Guloso apresentou um resultado 4,39% e 6,64% melhor que o BRD e o K-means, respectivamente. Por fim, em uma rede com 256 núcleos/processos a maior diferença foi encontrada nos testes com pacotes de 10 *flits*: o K-means e o BRD ficaram apenas 5,47% e 5,86%, respectivamente, atrás do Guloso.

Figura 15 – Gráficos de latência do *buffer* da aplicação EP



	4 flits	6 flits	8 flits	10 flits
BRD	41,2633	67,7332	105,613	199,199
Guloso	41,209	65,307	98,9062	175,042
K-Means	40,8526	64,9393	93,9482	157,484

	4 flits	6 flits	8 flits	10 flits
BRD	40,4424	62,2286	83,9209	120,602
Guloso	40,5812	62,7194	85,7958	123,777
K-Means	40,7946	63,2392	87,9249	135,334



	4 flits	6 flits	8 flits	10 flits
BRD	40,3631	61,9199	83,572	120,005
Guloso	40,4609	62,0211	83,4188	114,958
K-Means	40,3768	62,3654	84,7532	122,601

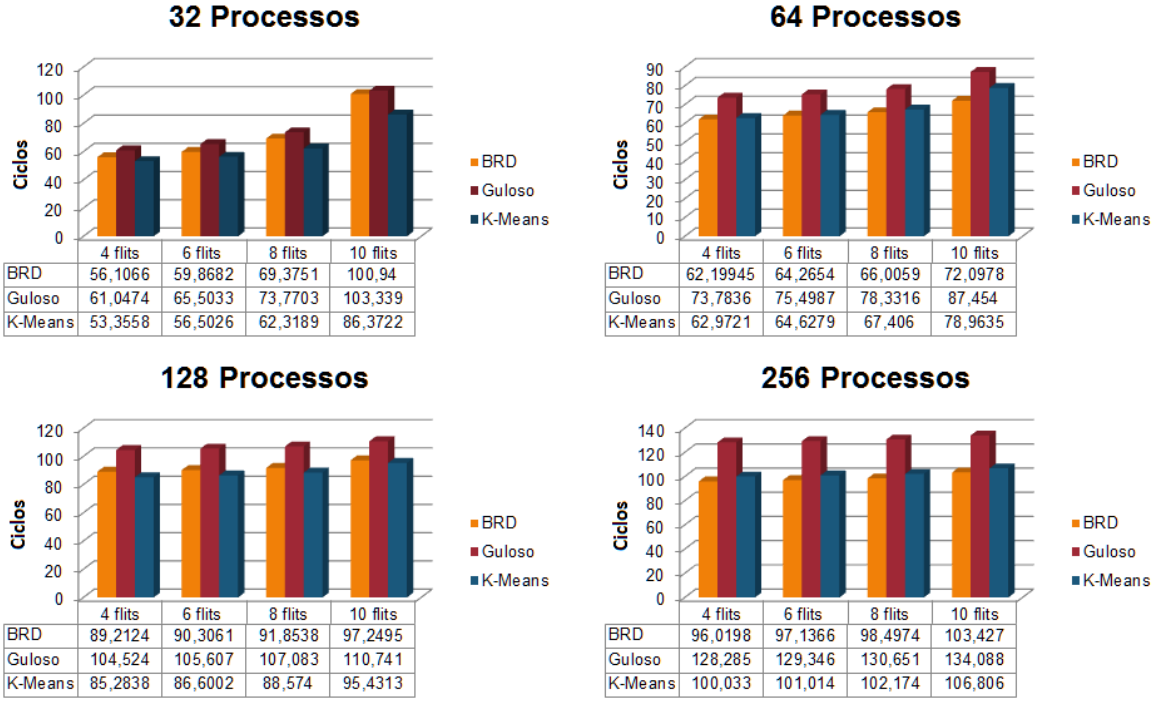
	4 flits	6 flits	8 flits	10 flits
BRD	40,3654	61,749	83,1363	121,001
Guloso	40,3511	61,7156	82,9937	114,293
K-Means	40,383	61,7657	83,0924	120,546

Fonte: Dados da pesquisa.

Na Figura 16 é apresentada a latência das mensagens na rede. Para os testes de 32 processos, os melhores resultados foram obtidos com o uso do algoritmo K-means. Para 10 *flits*, por exemplo, ele foi 16,86% melhor que o BRD, em segundo, e 19,71% melhor que o Guloso, em terceiro. Já nos testes com 64 processos, a menor latência foi obtida com o uso do BRD, sendo 9,52% e 21,29% menor que o K-means e Guloso, respectivamente. Os resultados para os testes de 128 processos se assemelham aos resultados de 32 processos: a menor latência foi obtida com o uso do K-means, cerca de 1,90% menor que o BRD e 16,03% menor que o Guloso. E por último, nos testes com 256 processos os resultados são similares aos de 64 processos: o uso do BRD gerou a menor latência, 3,26% e 26,64

menor do que o uso do K-means e do Guloso, respectivamente.

Figura 16 – Gráficos de latência das mensagens na rede da aplicação EP



	4 flits	6 flits	8 flits	10 flits
BRD	56,1066	59,8682	69,3751	100,94
Guloso	61,0474	65,5033	73,7703	103,339
K-Means	53,3558	56,5026	62,3189	86,3722

	4 flits	6 flits	8 flits	10 flits
BRD	62,19945	64,2654	66,0059	72,0978
Guloso	73,7836	75,4987	78,3316	87,454
K-Means	62,9721	64,6279	67,406	78,9635

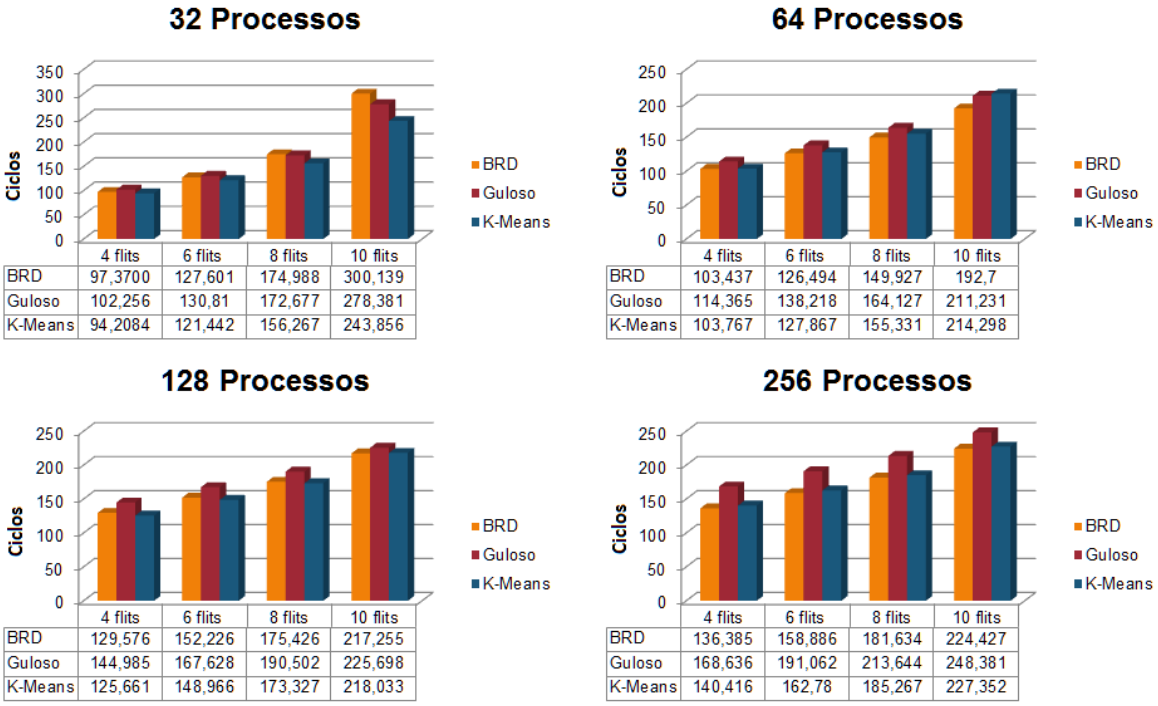
	4 flits	6 flits	8 flits	10 flits
BRD	89,2124	90,3061	91,8538	97,2495
Guloso	104,524	105,607	107,083	110,741
K-Means	85,2838	86,6002	88,574	95,4313

	4 flits	6 flits	8 flits	10 flits
BRD	96,0198	97,1366	98,4974	103,427
Guloso	128,285	129,346	130,651	134,088
K-Means	100,033	101,014	102,174	106,806

Fonte: Dados da pesquisa.

Os resultados de latência total das mensagens são representados nos gráficos da Figura 17. Como dito anteriormente, essa métrica é dada pela soma das latências de *buffer* e de rede. Nos resultados com 32 processos é possível notar que para as comunicações realizadas com pacotes de tamanho 4 e 6 *flits*, a menor latência é alcançada com o uso do K-means e em seguida com o uso do BRD. Isso ocorre devida à latência de rede que foi maior com o uso do Guloso. Já no caso dos resultados com pacotes de 8 e 10 *flits*, a latência total foi menor utilizando o K-means e em seguida com a utilização do Guloso. Isso porque com o uso do BRD a latência de *buffer* foi maior que os demais. Já nos resultados com 64 processos, a menor latência foi alcançada com o uso do BRD para todos os tamanhos de pacotes. A variação nos resultados ocorreu com a execução de pacotes de 10 *flits*: a segunda menor latência foi obtida com o uso do Guloso e não do K-means como nos demais resultados. Esse fato se deve pela alta latência de *buffer* obtida pelo uso do K-means. Nos testes com 128 processos, os melhores resultados foram obtidos com a utilização do K-means com pacotes de até 8 *flits*. Com pacotes de 10 *flits*, o uso do BRD foi mais vantajoso, porém o ganho foi praticamente insignificante, sendo menor que 1%. Por fim, nas execuções com 256 processos, os resultados de latência total sofreram maior influência dos resultados da latência de rede, mantendo o uso do BRD como melhor opção, seguido do K-means.

Figura 17 – Gráficos de latência total das mensagens da aplicação EP



	4 flits	6 flits	8 flits	10 flits
BRD	97,3700	127,601	174,988	300,139
Guloso	102,256	130,81	172,677	278,381
K-Means	94,2084	121,442	156,267	243,856

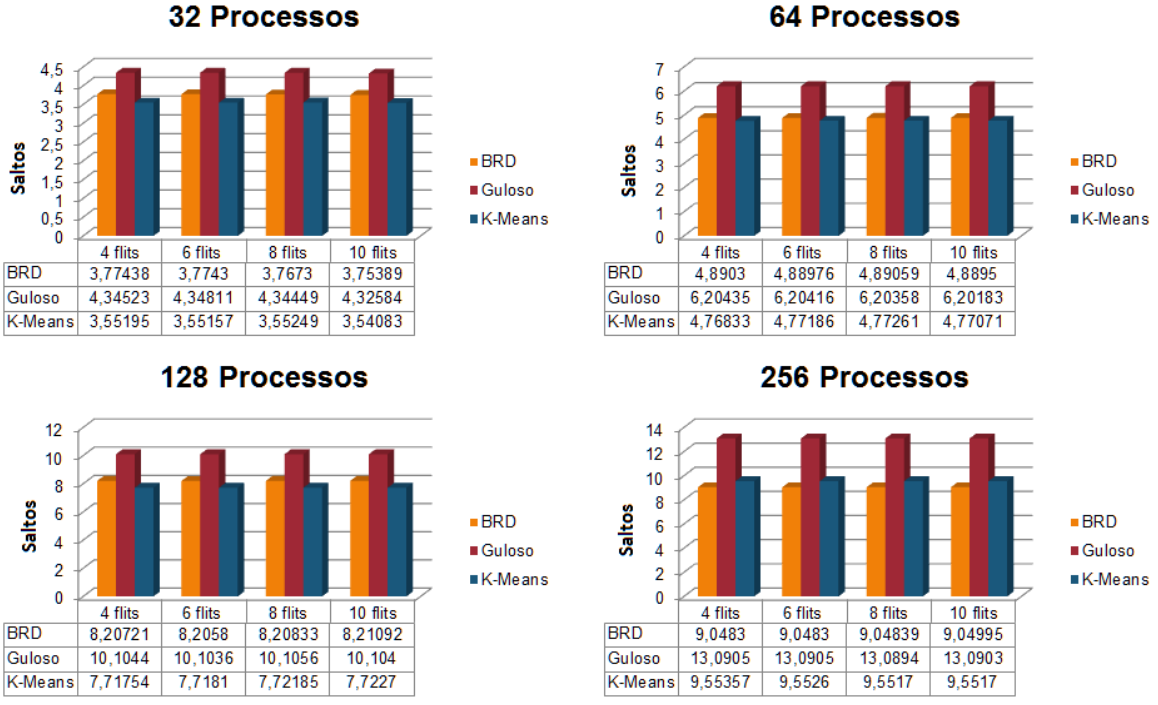
	4 flits	6 flits	8 flits	10 flits
BRD	103,437	126,494	149,927	192,7
Guloso	114,365	138,218	164,127	211,231
K-Means	103,767	127,867	155,331	214,298

	4 flits	6 flits	8 flits	10 flits
BRD	129,576	152,226	175,426	217,255
Guloso	144,985	167,628	190,502	225,698
K-Means	125,661	148,966	173,327	218,033

	4 flits	6 flits	8 flits	10 flits
BRD	136,385	158,886	181,634	224,427
Guloso	168,636	191,062	213,644	248,381
K-Means	140,416	162,78	185,267	227,352

Fonte: Dados da pesquisa.

Figura 18 – Gráficos de distância média da aplicação EP



	4 flits	6 flits	8 flits	10 flits
BRD	3,77438	3,7743	3,7673	3,75389
Guloso	4,34523	4,34811	4,34449	4,32584
K-Means	3,55195	3,55157	3,55249	3,54083

	4 flits	6 flits	8 flits	10 flits
BRD	4,8903	4,88976	4,89059	4,8895
Guloso	6,20435	6,20416	6,20358	6,20183
K-Means	4,76833	4,77186	4,77261	4,77071

	4 flits	6 flits	8 flits	10 flits
BRD	8,20721	8,2058	8,20833	8,21092
Guloso	10,1044	10,1036	10,1056	10,104
K-Means	7,71754	7,7181	7,72185	7,7227

	4 flits	6 flits	8 flits	10 flits
BRD	9,0483	9,0483	9,04839	9,04995
Guloso	13,0905	13,0905	13,0894	13,0903
K-Means	9,55357	9,5526	9,5517	9,5517

Fonte: Dados da pesquisa.

A distância média percorrida pelas mensagens é representada nos gráficos da Figura 18. Essa métrica está relacionada à latência das mensagens na rede, pois quanto maior é o

número de saltos realizados pelos pacotes de dados, maior será a latência das mensagens. Esse fato pode ser confirmado nos resultados dessa métrica. Nos testes realizados com 32 processos, a menor distância foi obtida com a utilização do K-means, sendo 6,01% e 22,17% menor que com o uso do BRD e Guloso, respectivamente. O cenário se repete nos resultados de 64 processos, onde o K-means alcançou a menor distância, cerca de 2,48% menor que o BRD e 29,99% menor que o Guloso. A utilização do K-means também foi vantajosa nos testes com 128 processos: 6,32% e 30,83% melhor que o BRD e Guloso, respectivamente. A situação é diferente para os resultados com 256 processos: o BRD obteve uma melhora de 5,54% e 44,64% com relação ao K-means e ao Guloso, nessa ordem.

É importante notar a pequena diferença entre a utilização do BRD e do K-means: esses algoritmos atingem resultados bem próximos na maioria das situações. Diante desse fato pode-se, previamente, dizer que o K-means obtém bons resultados para cargas com traços mais homogêneos, se compararmos com os resultados da aplicação CG. Outro ponto que merece destaque é que a aplicação EP possui, em sua maioria, um padrão de comunicação *broadcast*, e isso faz com que as comunicações sejam mais distantes. Esse fato pode ser comprovado se compararmos as distâncias médias dos programas EP e CG, por exemplo.

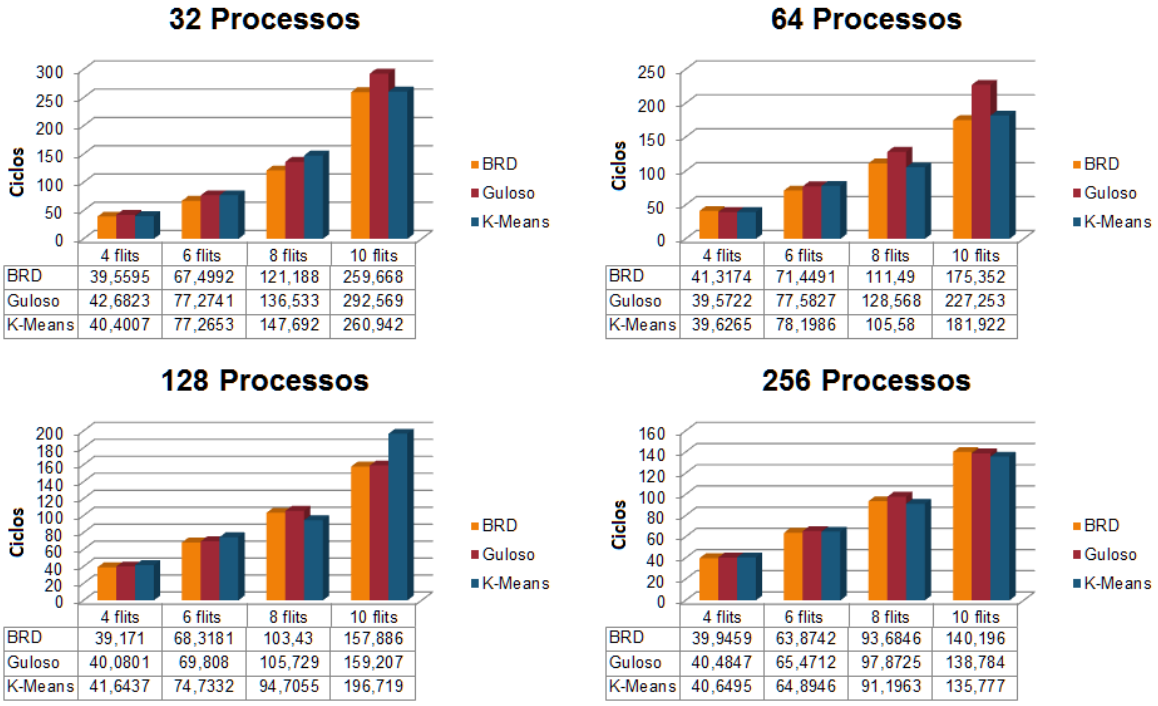
5.4 Aplicação FT

A aplicação FT possui uma pequena semelhança com a aplicação EP por apresentar, em sua maioria, um padrão de comunicação *broadcast* (OLIVEIRA, 2012). Diferente do programa CG e assim como o EP, o FT é considerado pequeno e possui um tamanho médio de mensagens relativamente elevado, alguns atingindo, aproximadamente, 65kB.

Os resultados de latência de *buffer* são apresentados nos gráficos da Figura 19. Nos testes com 32 processos, o melhores resultados foram alcançados com a utilização do algoritmo BRD, porém em comparação com o K-means a melhora não foi relevante, menos que 1%. Já em relação ao Guloso, o BRD obteve um ganho de 12,67%, comparando os resultados utilizando pacotes de 10 *flits*. O cenário é um pouco diferente nos experimentos com 64 processos, pois para cada tamanho de pacote foi obtido resultados distintos. Porém, a discrepância maior ocorreu nos testes com 10 *flits* e, novamente, o desempenho melhor foi obtido com o uso do BRD: 3,74% e 29,59% melhor que o K-means e o Guloso, respectivamente. O algoritmo BRD continua obtendo melhores resultados também nos testes realizados com 128 processos para todos os tamanhos de pacotes testados. Nas execuções utilizando 10 *flits*, por exemplo, a latência dos *buffers* com o BRD foi menos que 1% menor em relação ao Guloso e 24,59% menor comparando com o uso do K-means. A situação dos resultados para uma rede de 256 núcleos é semelhante à de 64: para cada tamanho de pacote, os resultados obtidos são diferentes. A maior diferença, neste caso,

também foi obtida com pacotes de 10 *flits*: o K-means alcançou a menor latência, sendo 2,21% e 3,25% menor que com o uso do Guloso e BRD, nessa ordem.

Figura 19 – Gráficos de latência do *buffer* da aplicação FT



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	39,5595	67,4992	121,188	259,668
Guloso	42,6823	77,2741	136,533	292,569
K-Means	40,4007	77,2653	147,692	260,942

64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	41,3174	71,4491	111,49	175,352
Guloso	39,5722	77,5827	128,568	227,253
K-Means	39,6265	78,1986	105,58	181,922

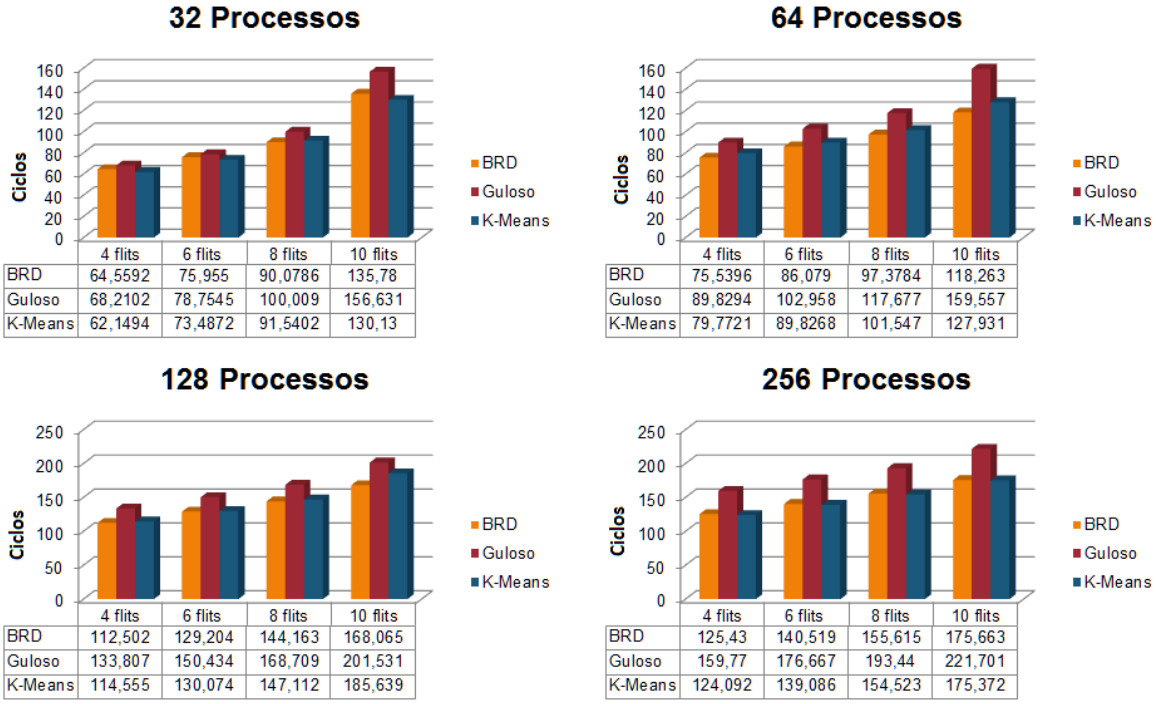
128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	39,171	68,3181	103,43	157,886
Guloso	40,0801	69,808	105,729	159,207
K-Means	41,6437	74,7332	94,7055	196,719

256 Processos	4 flits	6 flits	8 flits	10 flits
BRD	39,9459	63,8742	93,6846	140,196
Guloso	40,4847	65,4712	97,8725	138,784
K-Means	40,6495	64,8946	91,1963	135,777

Fonte: Dados da pesquisa.

As latências das mensagens na rede na execução da aplicação FT são apresentadas nos gráficos da Figura 20. Nos testes realizados com uma rede de 32 núcleos/processos os melhores resultados foram alcançados com a utilização do algoritmo K-means, com uma diferença de 4,34% 20,36% em relação ao BRD e Guloso, respectivamente. Os resultados com 64 processos apontaram para a utilização do BRD, que apresentou 8,17% e 34,91% de melhora em relação ao K-means e ao Guloso, nessa ordem. Os resultados apresentados nos testes com 128 processos, novamente, com a utilização do algoritmo BRD obteve-se latências menores: 10,45% e 19,91% menor que na utilização do K-means e do Guloso, respectivamente. Com a utilização do algoritmo K-means foi possível obter melhores resultados, mas dessa vez com uma rede de 256 núcleos. A melhoria com relação ao BRD não foi significativa, menor que 1%, mas com relação ao Guloso a melhora foi de 26,41%. Os resultados citados foram calculados com base na transmissão de pacotes de 10 *flits*, uma vez que os resultados com os demais tamanhos de pacotes são semelhantes.

Figura 20 – Gráficos de latência das mensagens na rede da aplicação FT



	4 flits	6 flits	8 flits	10 flits
BRD	64,5592	75,955	90,0786	135,78
Guloso	68,2102	78,7545	100,009	156,631
K-Means	62,1494	73,4872	91,5402	130,13

	4 flits	6 flits	8 flits	10 flits
BRD	75,5396	86,079	97,3784	118,263
Guloso	89,8294	102,958	117,677	159,557
K-Means	79,7721	89,8268	101,547	127,931

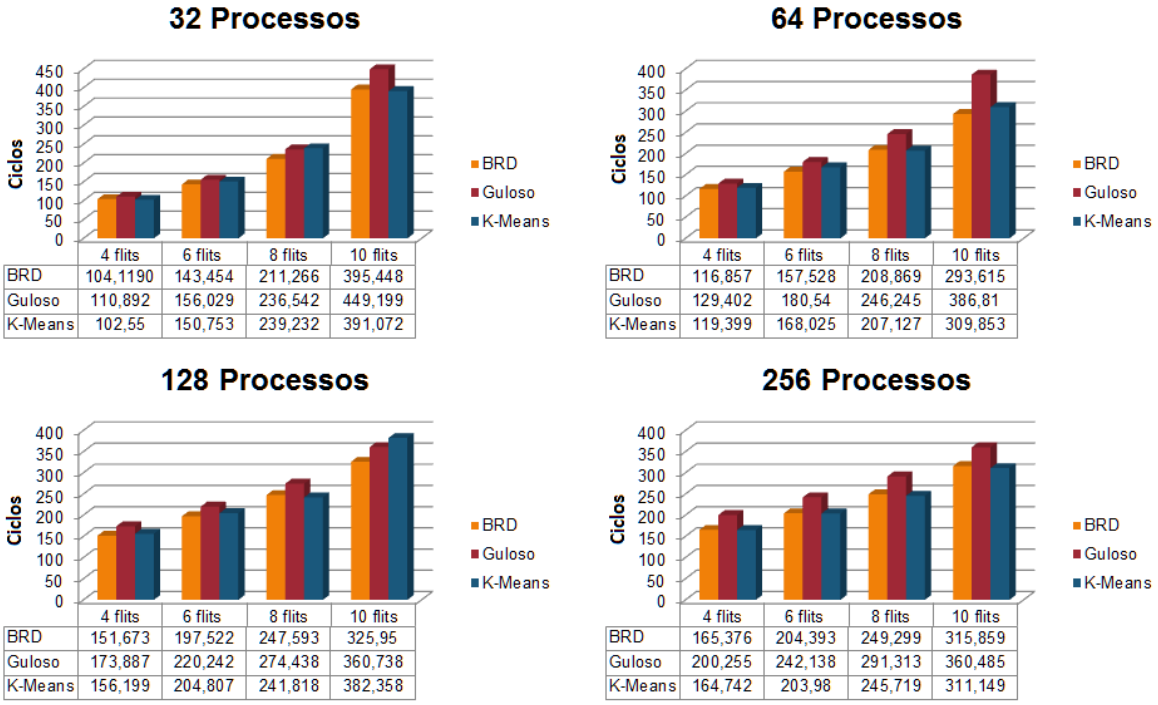
	4 flits	6 flits	8 flits	10 flits
BRD	112,502	129,204	144,163	168,065
Guloso	133,807	150,434	168,709	201,531
K-Means	114,555	130,074	147,112	185,639

	4 flits	6 flits	8 flits	10 flits
BRD	125,43	140,519	155,615	175,663
Guloso	159,77	176,667	193,44	221,701
K-Means	124,092	139,086	154,523	175,372

Fonte: Dados da pesquisa.

A latência total da rede apresentada nos gráficos da Figura 21 representa a soma das latências dos *buffers* e da rede. O aumento da latência de acordo com o aumento do tamanho dos pacotes ressalta que quanto maior o tamanho do pacote, maior será o tempo necessário para executá-lo e para transmiti-lo. Assim, pode haver mais espera no *buffer* e mais atrasos nas transmissões. Observando a latência total de maneira geral é possível perceber que os melhores resultados são alcançados com o uso dos algoritmos BRD e pelo K-means. Isso porque esses algoritmos conseguiram reduzir o número de saltos entre os processos com maior custo de comunicação, se comparados com o algoritmo Guloso. A aplicações FT possui custos de comunicação variados entre os processos.

Figura 21 – Gráficos de latência total das mensagens da aplicação FT



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	104,1190	143,454	211,266	395,448
Guloso	110,892	156,029	236,542	449,199
K-Means	102,55	150,753	239,232	391,072

64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	116,857	157,528	208,869	293,615
Guloso	129,402	180,54	246,245	386,81
K-Means	119,399	168,025	207,127	309,853

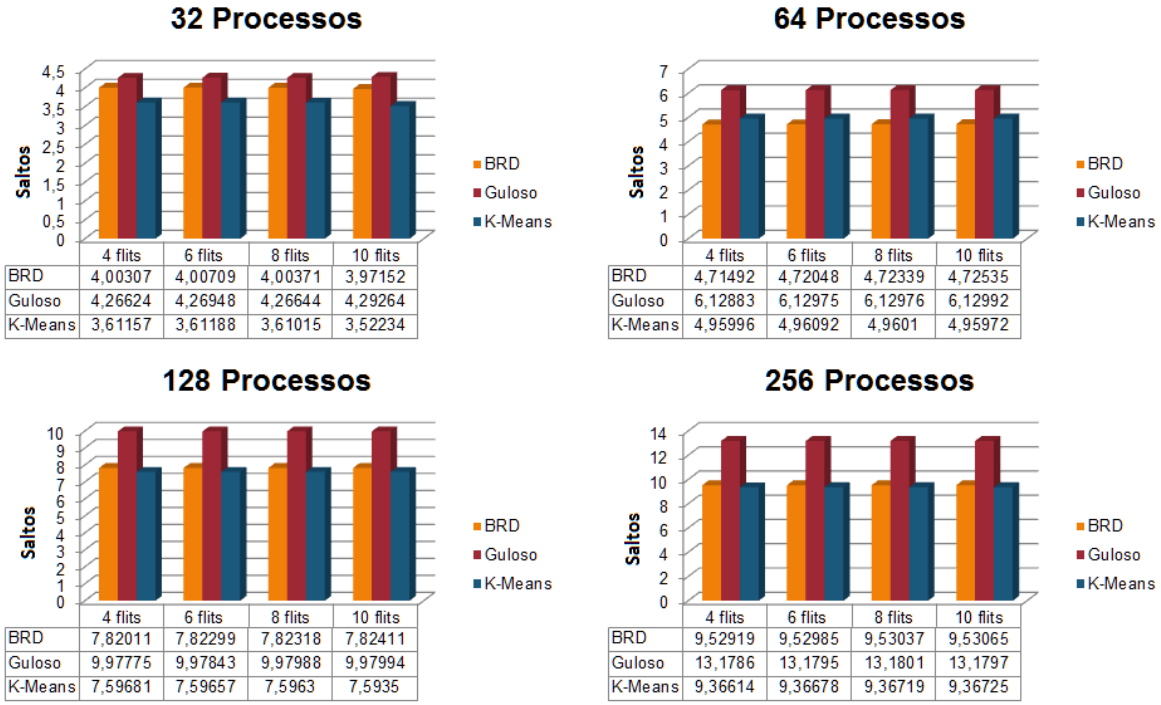
128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	151,673	197,522	247,593	325,95
Guloso	173,887	220,242	274,438	360,738
K-Means	156,199	204,807	241,818	382,358

256 Processos	4 flits	6 flits	8 flits	10 flits
BRD	165,376	204,393	249,299	315,859
Guloso	200,255	242,138	291,313	360,485
K-Means	164,742	203,98	245,719	311,149

Fonte: Dados da pesquisa.

Os gráficos da Figura 22 mostram os resultados da distância média percorrida pelos pacotes de dados. Seguindo a avaliação anterior com pacotes de 10 *flits*, nos testes com 32 processos, os resultados foram melhores com a utilização do K-means, que obteve um ganho de 12,75% comparado com o uso do BRD e 21,86% com relação ao Guloso. Já nos testes com 64 processos, o melhor desempenho foi alcançado com o uso do BRD: 4,95% e 29,72% melhor que com o uso do K-means e do Guloso, respectivamente. Nos testes com 128 processos, o cenário é semelhante ao de 32: com a utilização do K-means obteve-se melhores resultados, alcançando um ganho de 3,03% e 32,42% em comparação ao BRD e ao Guloso, nessa ordem. E por fim, nos testes com 256 processos o uso do K-means se mostrou mais vantajoso, alcançando uma distância 1,74% menor comparando com o uso do BRD e 40,69% menor em relação ao uso do Guloso.

Figura 22 – Gráficos de distância média da aplicação FT



	4 flits	6 flits	8 flits	10 flits
BRD	4,00307	4,00709	4,00371	3,97152
Guloso	4,26624	4,26948	4,26644	4,29264
K-Means	3,61157	3,61188	3,61015	3,52234

	4 flits	6 flits	8 flits	10 flits
BRD	4,71492	4,72048	4,72339	4,72535
Guloso	6,12883	6,12975	6,12976	6,12992
K-Means	4,95996	4,96092	4,9601	4,95972

	4 flits	6 flits	8 flits	10 flits
BRD	7,82011	7,82299	7,82318	7,82411
Guloso	9,97775	9,97843	9,97988	9,97994
K-Means	7,59681	7,59657	7,5963	7,5935

	4 flits	6 flits	8 flits	10 flits
BRD	9,52919	9,52985	9,53037	9,53065
Guloso	13,1786	13,1795	13,1801	13,1797
K-Means	9,36614	9,36678	9,36719	9,36725

Fonte: Dados da pesquisa.

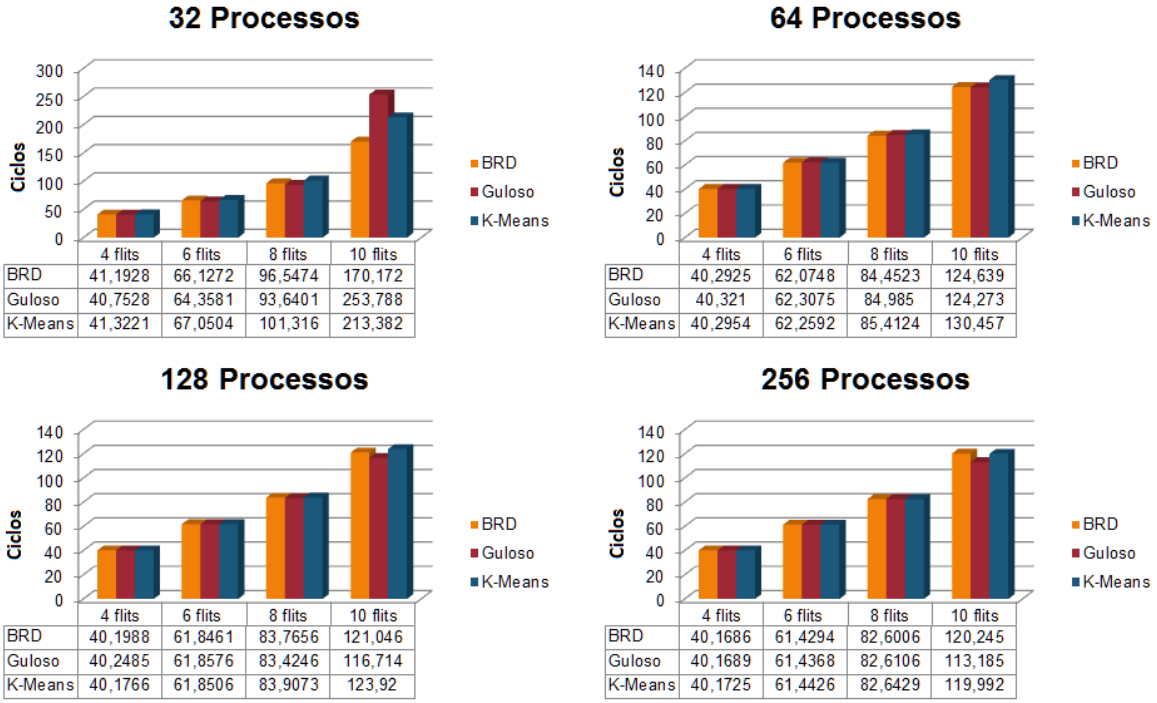
A latência total das mensagens pode ser influenciada pela distância média percorrida pelos pacotes, pois quanto maior é a distância, mais tempo os pacotes gastarão para chegar ao destino final. Considerando essa relação, os resultados da distância média percorrida pelos pacotes de dados, apresentados nos gráficos da Figura 22, pode-se ressaltar que com a utilização dos algoritmos BRD e K-means é possível obter melhores resultados para a aplicação FT. Ou seja, esses dois algoritmos conseguiram reduzir mais o número de saltos entre os processos com maior custo de comunicação. Também é possível observar a semelhança dos resultados desta aplicação com os resultados da aplicação EP. Essa proximidade já podia ser esperada, e ressaltada com os resultados, uma vez que são cargas possuem características semelhantes.

5.5 Aplicação IS

Assim como nas aplicações EP e FT, as transmissões *broadcast* representam a maior parte das comunicações na aplicação IS, em virtude das comunicações N:N. Apesar disso, é a aplicação que possui um padrão de comunicação mais misto, dentre as aplicações utilizadas, por conter comunicações N:N, 1:1, N:1 e 1:N (OLIVEIRA, 2012). É uma aplicação que possui uma diferença muito grande entre os tamanhos dos pacotes: alguns de 8 *bytes* e e outros de 65 kB. Esse fator possui grande influência no mapeamento, pois a análise de custo também considera o tamanho dos pacotes que serão transmitidos para gerar as alocações dos processos. Se tratando do tamanho da aplicação, não é considerada

pequena como a EP e nem grande como a CG.

Figura 23 – Gráficos de latência do *buffers* da aplicação IS



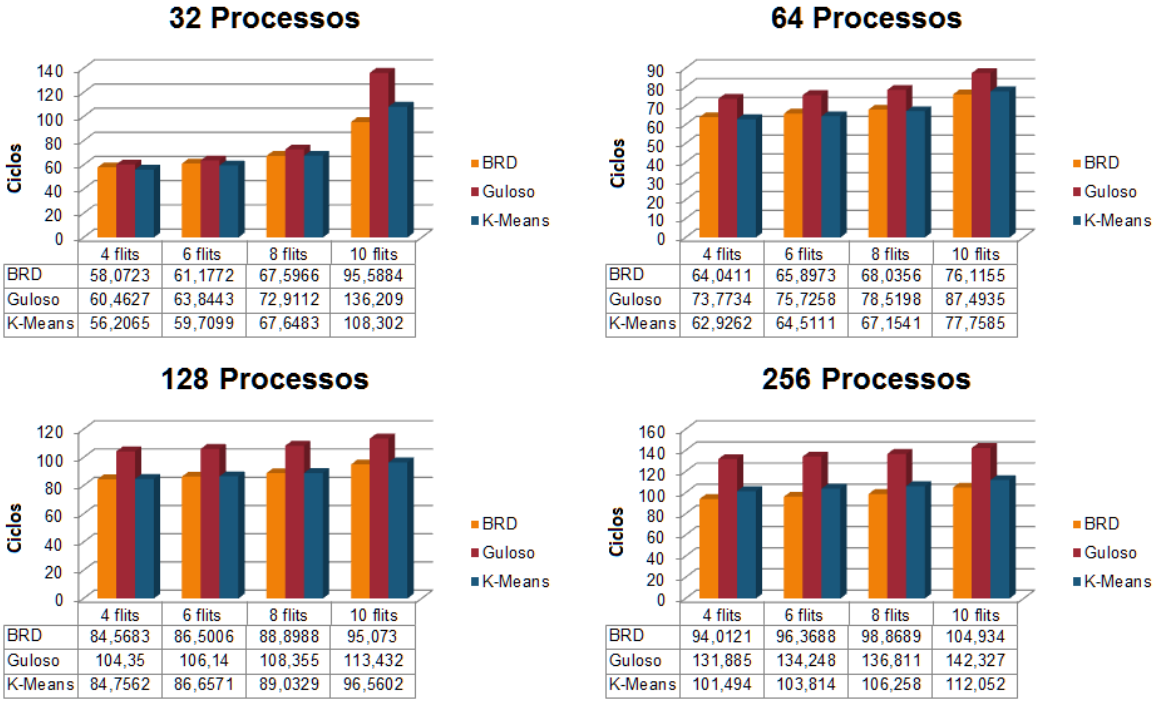
Fonte: Dados da pesquisa.

Nos gráficos da Figura 23 são apresentados os resultados de latências de *buffers*. Nos testes realizados com 32 processos é possível observar que com pacotes menores, menores que 8 *flits*, a melhor opção de mapeamento foi a heurística Gulosa, sendo 3,10% melhor que o uso do BRD e 8,19% melhor que o K-means, com pacotes de 8 *flits*, por exemplo. Já para pacotes de 10 *flits*, o melhor resultado foi alcançado com o uso do BRD: com o uso do K-means a latência aumentou 25,39% e com o Guloso 49,13%. Já nos experimentos com 64 processos, a diferença mais significativa ocorreu com transmissões de pacotes de 10 *flits*: o melhor resultado foi obtido com o uso do algoritmo Guloso, porém a diferença entre ele e o BRD é menor que 1% e, comparando com o K-means, também houve uma pequena diferença, mas mais significativa, de 4,97%. O cenário é semelhante para os testes com 128 e 256 processos, ou seja, resultados mais expressivos ocorreram com pacotes de 10 *flits*. No caso de 128 processos, o melhor resultado foi obtido com o uso do Guloso: 3,71% e 6,17% melhor que com a utilização do BRD e do K-means, respectivamente. No caso dos testes com 256 processos, o uso da heurística Gulosa mostrou ser, ainda, a melhor opção e em seguida o K-means, com 6,01% de diferença e 6,23% com relação ao BRD.

Os resultados da latência das mensagens na rede são apresentados nos gráficos da Figura 24. Nos testes realizados com 32 processos, houve uma diferença nos resultados conforme o tamanho dos pacotes. Para pacotes menores, como de 4 *flits*, o uso do algo-

ritmo K-means mostrou ser a melhor opção, obtendo 4,47% e 7,57% de ganho com relação ao BRD e ao Guloso, nesta ordem. Já com pacotes maiores, de 10 *flits*, por exemplo, as menores latências foram alcançadas com o uso do BRD: 13,30% e 42,49% menor em relação ao K-means e ao Guloso, respectivamente. O cenário nos testes de 64 processos é parecido com o anterior: para tamanhos reduzidos de pacotes o uso do K-means alcança menor latência e para tamanhos maiores de pacotes, a utilização do BRD reduziu a latência da rede. Na transmissão de pacotes de 4 *flits*, por exemplo, houve uma diferença pequena entre o segundo melhor resultado, de 1,77%, do BRD, mas comparando com o Guloso, diferença aumenta para 17,23%. E nos resultados com pacotes de 10 *flits*, por exemplo, a desigualdade entre os melhores resultados é de apenas 2,15%, já entre o BRD e o Guloso, a diferença é de 14,97%. Já nos testes com 128 processos, para todos os tamanhos de pacotes, o uso do BRD e do K-means apresentaram resultados muito próximos, mas com o Guloso a latência foi 19,31% maior que com o BRD (que apresentou melhor resultado), no caso da transmissão de pacotes de 10 *flits*, por exemplo. Por fim, nos experimentos com 256 processos, a menor latência foi adquirida com o uso do BRD para todos os tamanhos de pacotes. Para os pacotes de 10 *flits*, por exemplo, o BRD obteve um ganho de 6,78% e 35,78% em relação ao K-means e ao Guloso, respectivamente.

Figura 24 – Gráficos de latência das mensagens na rede da aplicação IS



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	58,0723	61,1772	67,5966	95,5884
Guloso	60,4627	63,8443	72,9112	136,209
K-Means	56,2065	59,7099	67,6483	108,302

64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	64,0411	65,8973	68,0356	76,1155
Guloso	73,7734	75,7258	78,5198	87,4935
K-Means	62,9262	64,5111	67,1541	77,7585

128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	84,5683	86,5006	88,8988	95,073
Guloso	104,35	106,14	108,355	113,432
K-Means	84,7562	86,6571	89,0329	96,5602

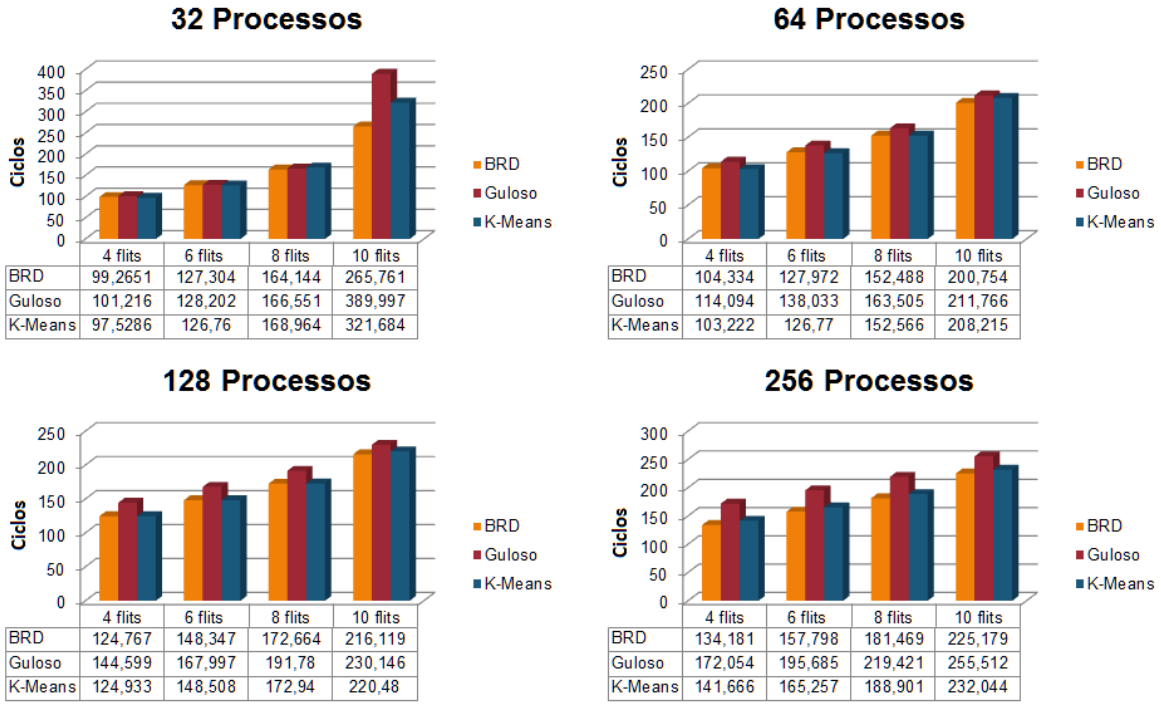
256 Processos	4 flits	6 flits	8 flits	10 flits
BRD	94,0121	96,3688	98,8689	104,934
Guloso	131,885	134,248	136,811	142,327
K-Means	101,494	103,814	106,258	112,052

Fonte: Dados da pesquisa.

A latência total, sendo a soma das latências de *buffers* e de rede, é apresentada nos gráficos da Figura 25. Nos testes de 32 processos é possível perceber que o melhor desempenho foi alcançado com o uso do algoritmo BRD. É possível verificar, também, que a diferença entre os algoritmos é pequena quando são transmitidos pacotes menores que 8

flits, já nas transmissões de pacotes de 10 *flits* ocorreu uma diferença considerável: com o uso do BRD foi possível obter um ganho de 21,04% e 46,74% em relação ao K-means e ao Guloso, respectivamente. Nos experimentos com 64 processos há uma oscilação entre os resultados no uso de dois algoritmos: o BRD e o K-means. Em transmissões de pacotes menores, de 4 ou 6 *flits*, por exemplo, o uso do K-means seria o mais indicado, porém a diferença para o segundo, o BRD é, consideravelmente, pequena. Já nas transmissões de pacotes de 10 *flits*, o melhor resultado foi alcançado com o uso do BRD: 3,71% de ganho com relação ao K-means e 5,48% comparado com a utilização do Guloso. No cenário dos testes com 128 processos, com pacotes menores que 8 *flits*, não houve uma diferença significativa entre o uso do K-means e do BRD, que foram os que obtiveram melhores resultados. Já com pacotes de 10 *flits* a diferença é perceptível: 2,01% e 6,49% de ganho com o uso do BRD em relação à utilização do K-means e do Guloso, nessa ordem. Por fim, nos testes com 256 processos, o BRD alcançou os melhores resultados para todos os tamanhos de pacotes. Com transmissões de pacotes de 10 *flits*, por exemplo, o BRD obteve uma vantagem de 3,04% para o K-menas e 13,47% em relação ao uso do Guloso.

Figura 25 – Gráficos de latência total das mensagens da aplicação IS



	4 flits	6 flits	8 flits	10 flits
BRD	99,2651	127,304	164,144	265,761
Guloso	101,216	128,202	166,551	389,997
K-Means	97,5286	126,76	168,964	321,684

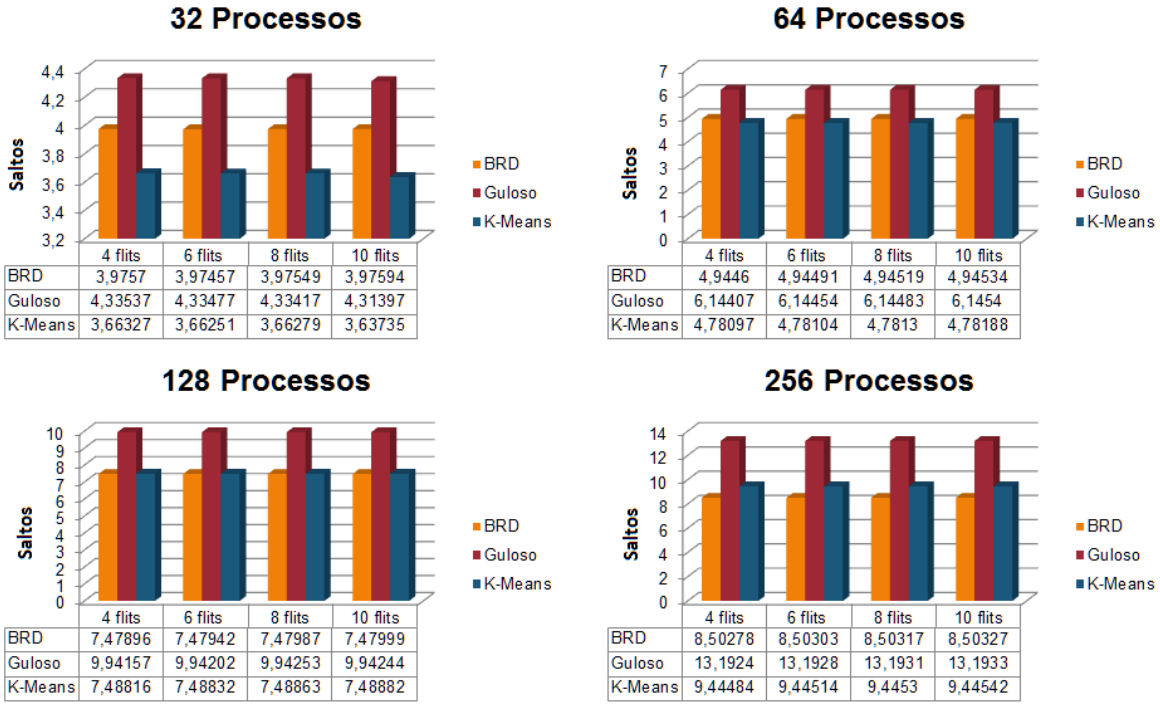
	4 flits	6 flits	8 flits	10 flits
BRD	104,334	127,972	152,488	200,754
Guloso	114,094	138,033	163,505	211,766
K-Means	103,222	126,77	152,566	208,215

	4 flits	6 flits	8 flits	10 flits
BRD	124,767	148,347	172,664	216,119
Guloso	144,599	167,997	191,78	230,146
K-Means	124,933	148,508	172,94	220,48

	4 flits	6 flits	8 flits	10 flits
BRD	134,181	157,798	181,469	225,179
Guloso	172,054	195,685	219,421	255,512
K-Means	141,666	165,257	188,901	232,044

Fonte: Dados da pesquisa.

Figura 26 – Gráficos de distância média da aplicação IS



	4 flits	6 flits	8 flits	10 flits
BRD	3,9757	3,97457	3,97549	3,97594
Guloso	4,33537	4,33477	4,33417	4,31397
K-Means	3,66327	3,66251	3,66279	3,63735

	4 flits	6 flits	8 flits	10 flits
BRD	4,9446	4,94491	4,94519	4,94534
Guloso	6,14407	6,14454	6,14483	6,1454
K-Means	4,78097	4,78104	4,7813	4,78188

	4 flits	6 flits	8 flits	10 flits
BRD	7,47896	7,47942	7,47987	7,47999
Guloso	9,94157	9,94202	9,94253	9,94244
K-Means	7,48816	7,48832	7,48863	7,48882

	4 flits	6 flits	8 flits	10 flits
BRD	8,50278	8,50303	8,50317	8,50327
Guloso	13,1924	13,1928	13,1931	13,1933
K-Means	9,44484	9,44514	9,4453	9,44542

Fonte: Dados da pesquisa.

Nos gráficos da Figura 26 são apresentados os gráficos que representam os resultados de distância média percorrida pelos pacotes de dados. Nos testes com 32 processos, com pacotes de 10 *flits*, por exemplo, a menor distância média foi alcançada com o uso do K-means, obtendo um ganho de 9,30% e 18,60% em relação ao uso do BRD e do Guloso, respectivamente. A situação é semelhante nos testes com 64 processos: os melhores resultados foram obtidos com a utilização do K-means, sendo 3,41% melhor que com o uso do BRD e 28,51% superior ou uso do Guloso. Nos resultados para 128 processos, a menor distância foi adquirida com o uso do BRD, porém a diferença entre o uso do BRD e do K-means não é significativa, sendo menor que 1%, já com relação ao uso do Guloso houve uma diferença significativa de 32,92%. Por fim, nos testes com 256 processos, novamente, os melhores resultados são alcançados com o uso do BRD: 11,07% e 55,15% melhor do que com a utilização dos algoritmos K-means e Guloso, nessa ordem.

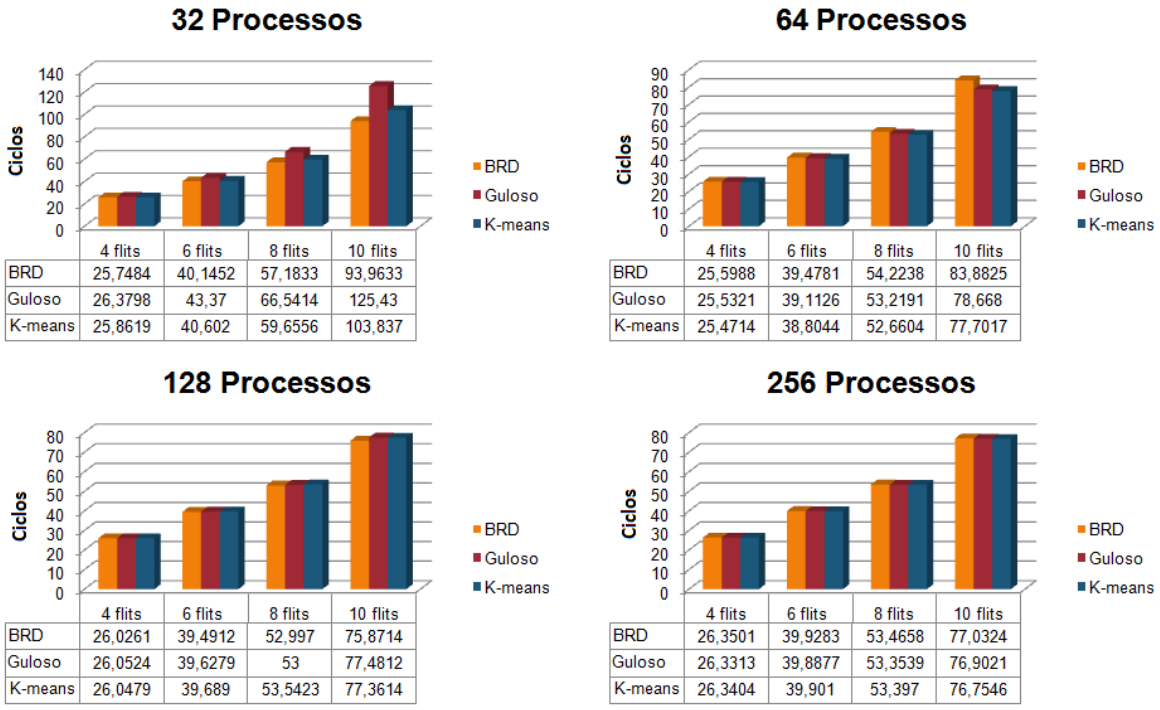
Esta aplicação possui um padrão de comunicação misto, e nesse caso, os resultados mais relevantes foram alcançados com a utilização do algoritmo BRD, assim como no programa CG, que também é uma carga heterogênea.

5.6 Aplicação MG

Assim como a aplicação CG, a MG apresenta um padrão de comunicação mais irregular, ou seja, possui pacotes *unicast* e *broadcast*. Segundo Oliveira (2012), em cenários que possuem menos de 64 núcleos prevalecem os pacotes *unicasts* e em redes maiores, a

predominância é dos pacotes *broadcast*. Outra característica importante da aplicação MG é que a maioria das comunicações ocorrem entre nós próximos. Este ponto é relevante para o mapeamento, pois o número de saltos, neste caso, realizados pelos pacotes já é reduzido, sendo assim irá prevalecer o tamanho do pacote e quantidade de comunicação entre os processos. Os tamanhos dos pacotes podem variar entre 96 *bytes* à 68 kB. O programa MG é considerado grande se comparado com os EP, FT e IS, mas é menor que o CG.

Figura 27 – Gráficos de latência do *buffer* da aplicação MG



	4 flits	6 flits	8 flits	10 flits
BRD	25,7484	40,1452	57,1833	93,9633
Guloso	26,3798	43,37	66,5414	125,43
K-means	25,8619	40,602	59,6556	103,837

	4 flits	6 flits	8 flits	10 flits
BRD	25,5988	39,4781	54,2238	83,8825
Guloso	25,5321	39,1126	53,2191	78,668
K-means	25,4714	38,8044	52,6604	77,7017

	4 flits	6 flits	8 flits	10 flits
BRD	26,0261	39,4912	52,997	75,8714
Guloso	26,0524	39,6279	53	77,4812
K-means	26,0479	39,689	53,5423	77,3614

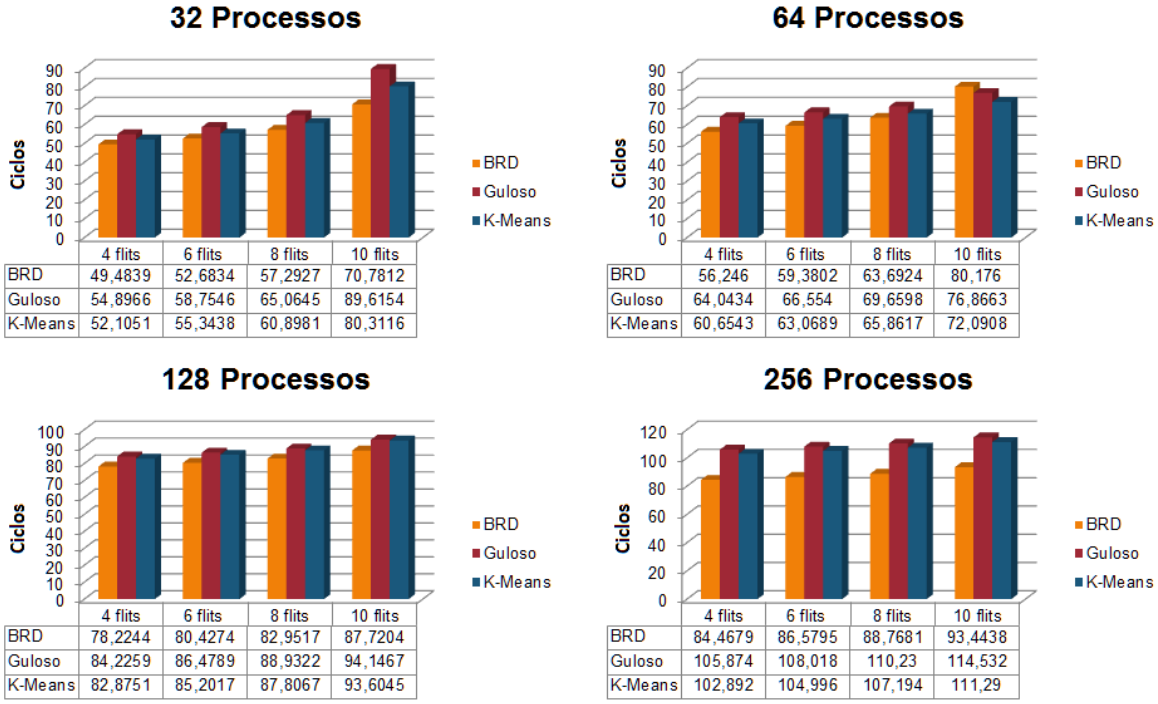
	4 flits	6 flits	8 flits	10 flits
BRD	26,3501	39,9283	53,4658	77,0324
Guloso	26,3313	39,8877	53,3539	76,9021
K-means	26,3404	39,901	53,397	76,7546

Fonte: Dados da pesquisa.

Os gráficos da Figura 27 apresentam os resultados das latências de *buffer*. Conforme mostra o gráfico dos testes de 32 processos é possível notar uma diferença relevante nos resultados quando são transmitidos pacotes de 8 ou 10 *flits*. No caso da transmissão de 10 *flits*, por exemplo, o menor latência foi alcançada com o uso do algoritmo BRD, cerca de 10,50% e 33,48% menor do que com o uso do K-means e do Guloso, respectivamente. O cenário é semelhante para os experimentos com 64 processos: com transmissões de pacotes pequenos não ocorre uma variação significativa nos resultados dos mapeamentos. Porém, com 10 *flits* é perceptível o ganho de desempenho com a utilização do algoritmo K-means: 1,24% em relação ao Guloso e 7,95% comparando com o uso do BRD. Nos testes com 128 processos, a desigualdade entre os resultados para cada algoritmo é pequena para todos os tamanhos de pacotes. Nas transmissões de 10 *flits*, por exemplo, a diferença entre o BRD e os demais é de, aproximadamente, 2%, ou seja, a menor latência é, ligeiramente, alcançada com o uso do BRD. A situação é semelhante para os testes com 256 processos, para pacotes de 10 *flits*, por exemplo, o ganho com o uso do K-means é menor que 1% com relação aos demais.

A latência das mensagens na rede é apresentada nos gráficos da Figura 28. Nos testes realizados com 32 processos, a menor latência foi alcançada com o mapeamento gerado pelo algoritmo BRD: 13,46% e 20,60% menor do que com o uso do K-means e o Guloso, respectivamente. O cenário para 64 processos é um pouco diferente: para pacotes menores que 8 *flits*, os melhores resultados foram alcançados com o uso do BRD, porém nas transmissões de pacotes de 10 *flits*, o K-means mostrou ser a melhor opção, obtendo uma diferença de 6,62% do BRD e 11,21% do Guloso. Nos experimentos com 128 processos, novamente, o uso do BRD foi a melhor opção: 6,70% e 7,32% melhor do que o uso do K-means e do Guloso, nessa ordem. A situação nos testes de 256 processos é semelhante à anterior: o mapeamento feito pelo BRD apresentou melhores resultados, sendo 19,09% melhor do que o mapeamento feito pelo K-means e 22,56% melhor que o Guloso.

Figura 28 – Gráficos de latência das mensagens na rede da aplicação MG



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	49,4839	52,6834	57,2927	70,7812
Guloso	54,8966	58,7546	65,0645	89,6154
K-Means	52,1051	55,3438	60,8981	80,3116

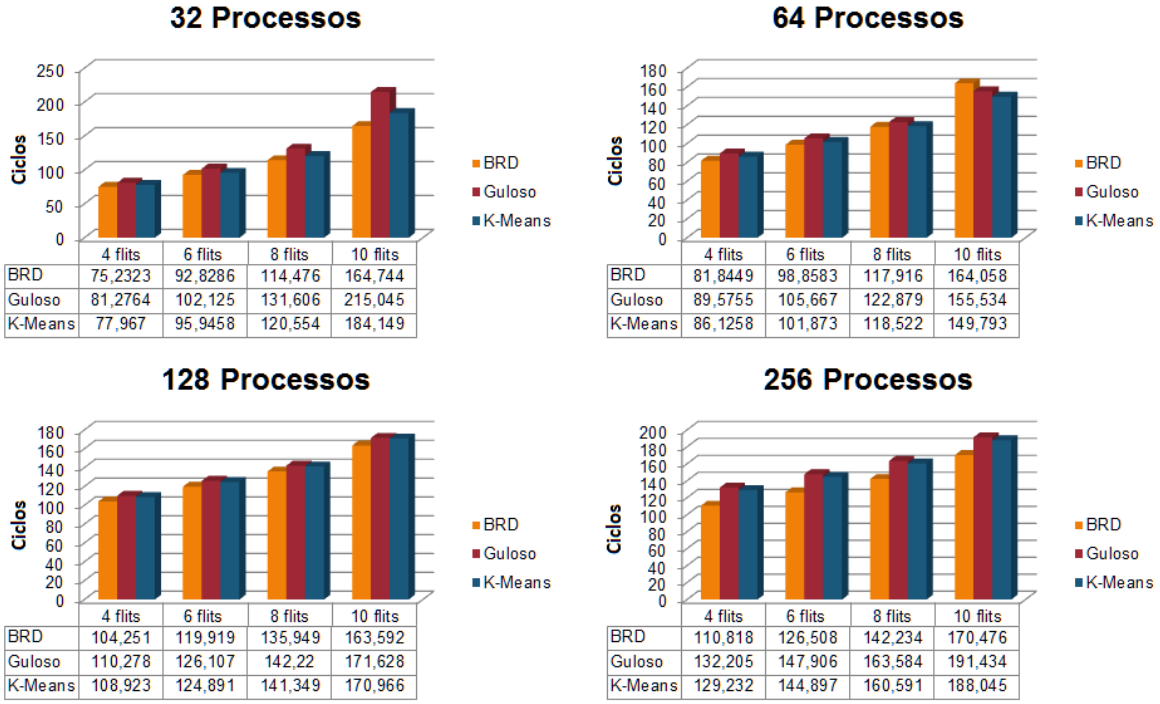
64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	56,246	59,3802	63,6924	80,176
Guloso	64,0434	66,554	69,6598	76,8663
K-Means	60,6543	63,0689	65,8617	72,0908

128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	78,2244	80,4274	82,9517	87,7204
Guloso	84,2259	86,4789	88,9322	94,1467
K-Means	82,8751	85,2017	87,8067	93,6045

256 Processos	4 flits	6 flits	8 flits	10 flits
BRD	84,4679	86,5795	88,7681	93,4438
Guloso	105,874	108,018	110,23	114,532
K-Means	102,892	104,996	107,194	111,29

Fonte: Dados da pesquisa.

Figura 29 – Gráficos de latência total das mensagens da aplicação MG



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	75,2323	92,8286	114,476	164,744
Guloso	81,2764	102,125	131,606	215,045
K-Means	77,967	95,9458	120,554	184,149

64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	81,8449	98,8583	117,916	164,058
Guloso	89,5755	105,667	122,879	155,534
K-Means	86,1258	101,873	118,522	149,793

128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	104,251	119,919	135,949	163,592
Guloso	110,278	126,107	142,22	171,628
K-Means	108,923	124,891	141,349	170,966

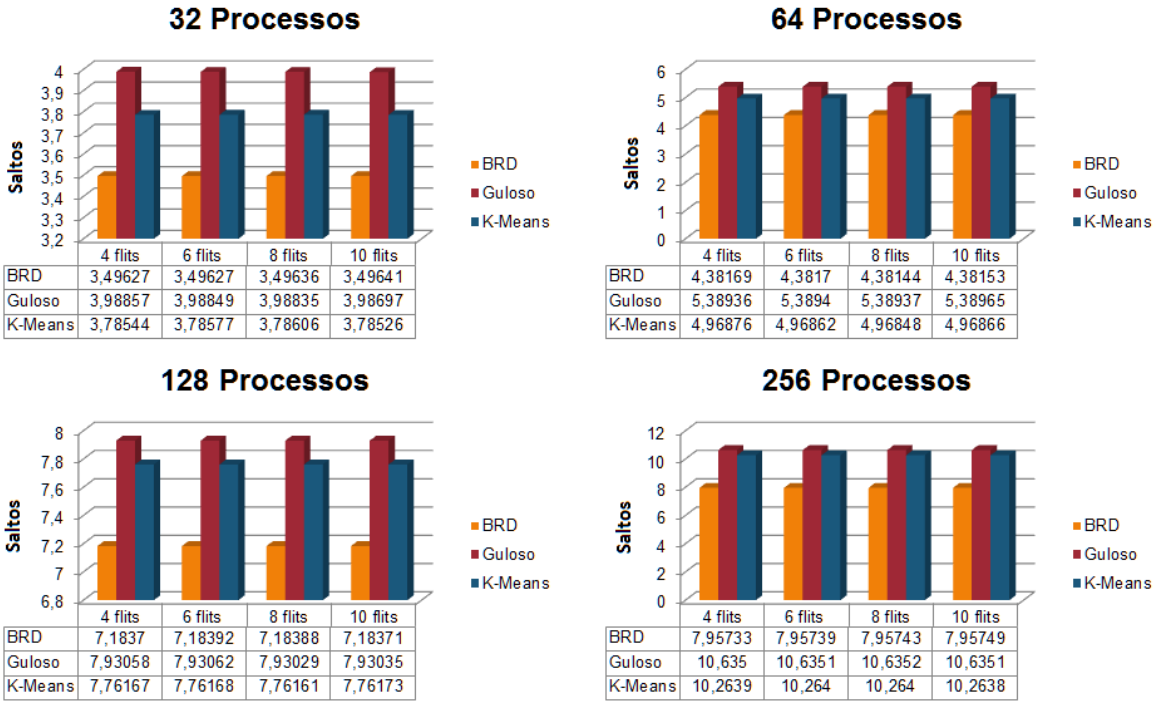
256 Processos	4 flits	6 flits	8 flits	10 flits
BRD	110,818	126,508	142,234	170,476
Guloso	132,205	147,906	163,584	191,434
K-Means	129,232	144,897	160,591	188,045

Fonte: Dados da pesquisa.

A latência total, como já dito anteriormente, representa a soma das latência de *buffer* e de mensagens na rede. Sabendo que os resultados de latência de *buffer* foram semelhantes para os três algoritmos de mapeamento, pode-se dizer que a os resultados de latência total serão mais influenciados pela latência da rede. Em outras palavras, conforme mostrado nos gráficos da Figura 29, nos testes realizados com 32, 128 e 256 processos, os melhores resultados são obtidos com o uso do BRD, seguido do K-means. Já nos resultados para 64 processos, a menor latência total é obtida com o uso do K-means, considerando pacotes de 10 *flits*, por exemplo.

A distância média percorrida pelos pacotes no uso de cada algoritmo é apresentada nos gráficos da Figura 30. No caso de 32 processos, a menor distância foi alcançada com o uso do BRD, 8,26% e 14,03% menor do que com o uso do K-means e do Guloso, respectivamente. O cenário é o mesmo para os testes com 64 processos: com o uso do BRD foi possível alcançar um ganho de 13,40% em relação ao K-means e 23% comparando com o Guloso. Nos experimentos com 128 processos, o melhor resultado foi apresentado com o uso , novamente, do BRD, sendo 8,04% e 10,39% melhor do que com o uso do K-means e do Guloso, nessa ordem. Por fim, o contexto não é diferente nos testes com 256 processos, ou seja, o BRD alcançou os melhores resultados e com diferenças significativas: 28,98% e 33,64% em relação ao uso do K-means e do Guloso, na devida ordem.

Figura 30 – Gráfico de distância média da aplicação MG



	4 flits	6 flits	8 flits	10 flits
BRD	3,49627	3,49627	3,49636	3,49641
Guloso	3,98857	3,98849	3,98835	3,98697
K-Means	3,78544	3,78577	3,78606	3,78526

	4 flits	6 flits	8 flits	10 flits
BRD	4,38169	4,3817	4,38144	4,38153
Guloso	5,38936	5,3894	5,38937	5,38965
K-Means	4,96876	4,96862	4,96848	4,96866

	4 flits	6 flits	8 flits	10 flits
BRD	7,1837	7,18392	7,18388	7,18371
Guloso	7,93058	7,93062	7,93029	7,93035
K-Means	7,76167	7,76168	7,76161	7,76173

	4 flits	6 flits	8 flits	10 flits
BRD	7,95733	7,95739	7,95743	7,95749
Guloso	10,635	10,6351	10,6352	10,6351
K-Means	10,2639	10,264	10,264	10,2638

Fonte: Dados da pesquisa.

A aplicação MG possui algumas características semelhantes à aplicação CG, por ser uma carga com traços não uniformes. Porém, em alguns momentos das transmissões, há muitas comunicações do tipo *broadcast*. Como podemos ver anteriormente, o uso do BRD obteve melhores resultados em aplicações com um padrão de comunicação heterogêneo e o K-means em aplicações com traços mais homogêneos. Isso pode justificar a oscilação dos resultados entre esses dois algoritmos, já que o programa MG possui os dois tipos de comunicações.

5.7 Considerações Finais

De modo geral, pode-se observar que cargas que possuem um padrão de comunicação homogêneo apresentam melhores resultados com o mapeamento gerado pelo K-means, no caso das aplicações EP e FT, por exemplo. A essência dessas aplicações são comunicações *broadcast*. Comunicações com esse padrão possuem custos de comunicações mais próximos, o que dificulta o mapeamento. O algoritmo K-means conseguiu trabalhar melhor essa questão por possuir uma função de balanceamento de carga ao final da alocação dos processos. Já para aplicações com um padrão de comunicação heterogêneo, que é o caso das aplicações CG e MG, por exemplo, a maioria dos melhores resultados são obtidos pelo uso do BRD. Os custos de comunicações entre os processos dessas aplicações tendem a ser mais distantes devido às inúmeras comunicações *unicast*. Outro fator importante dessas aplicações é que antes da realização do mapeamento, a maioria das comunicações

já ocorriam entre nós próximos. Após o mapeamento, considerando as outras variáveis do custo de comunicação, a distância entre as comunicações pode ter crescido. O que torna o BRD um algoritmo eficiente para esse caso. O Guloso já é um algoritmo que possui uma estratégia arriscada para os dois tipos de padrão de comunicações citados, pois a busca da solução local pode prejudicar a solução global do problema, sendo a estratégia que menos se mostrou eficiente, na maioria dos casos, em relação às outras.

A característica que algumas cargas possuem de já conterem comunicações entre processos próximos antes dos mapeamentos serem aplicados pode ajudar na análise feita no capítulo anterior sobre os processos que compartilham a mesma memória. Para essa análise, os processos deveriam permanecer próximos, após o mapeamento, se considerássemos o acesso à memória.

Outro ponto importante a ser relevado é que o simulador realiza o cálculo de distância média e das latências com base nas mensagens entregues. Esse fato explica a latência e a distância média na carga EP, que é uma carga menor, ser maior que a de CG, por exemplo. Conforme os gráficos no Apêndice A, pode-se notar a grande perda de mensagens que ocorre na execução da carga EP, já na aplicação CG, a perda de pacotes é muito pequena.

6 CONCLUSÃO

Como foi dito em alguns estudos apresentados, a tarefa de mapear processos pertence à classe NP-Difícil. Portanto, não se trata de uma tarefa trivial e nem mesmo de uma tarefa que possua uma solução ótima definida. Os estudos de mapeamento, concretizados com os resultados deste trabalho, mostram que é complexa a tarefa de encontrar um algoritmo que seja ótimo em todas as aplicações avaliadas ou que deseja avaliar. Cada aplicação possui um padrão de comunicação que interfere no desempenho dos algoritmos. Por isso, pode-se reforçar a idéia de que o conhecimento das características das aplicações é fundamental para encontrar um mapeamento mais adequado.

Existem trabalhos que tratam de mapeamento de processos considerando um mapeamento e comparando com a distribuição padrão dos processos, porém são poucos que apresentam a comparação entre abordagens diferentes de mapeamento. Essa dissertação apresentou três abordagens diferentes de mapeamento de processos: o BRD, um algoritmo guloso e o algoritmo de clusterização K-means. Os três foram avaliados com base no comportamento e desempenho de uma rede-em-chip simulada.

Com os resultados foi possível observar que o mapeamento gerado pelo algoritmo de clusterização K-means obteve melhores desempenhos quando utilizado em cargas de trabalho que possuem padrões de comunicação mais uniformes. Para aplicações cujo padrão de comunicação é mais heterogêneo, os melhores resultados foram alcançados com uso do algoritmo BRD. O algoritmo Guloso apresentou um bom desempenho na execução de algumas aplicações, porém, dentre os três avaliados, no geral, é o que obteve resultados menos satisfatórios. Uma vantagem do K-means e do Guloso em relação ao BRD é a facilidade de adaptação do algoritmo, ou seja, esses dois foram desenvolvidos durante a elaboração desta dissertação, o que pode possibilitar mais liberdade e domínio dos algoritmos. Já o BRD pertence ao *Scotch*, o que requer um estudo prévio do funcionamento da ferramenta.

Como trabalhos futuros é cogitada a integração entre o TOPAZ e o GEM5 para melhor utilização de variáveis. Isso pode trazer mais certezas do comportamento da rede com relação aos mapeamentos propostos, principalmente no que diz respeito à liberdade de manipulação dos dados trafegados na rede. Também é estudada a possibilidade de aprimoramento do K-means, uma vez que obtivemos bons resultados para um padrão de comunicação. Outro ponto importante para ser avaliado futuramente é o consumo de energia total gasto pela rede com a utilização dos algoritmos de mapeamento, que o TOPAZ não suporta, mas com a integração com o GEM5 é possível avaliar. Até então, podemos ter como hipótese que o consumo é reduzido, uma vez que as distâncias percorridas entre os pacotes são reduzidas com a aplicação de mapeamento de processos. As NoCs

possuem diversas características que podem ser exploradas futuramente para aumentar o desempenho, tais como: novas políticas de encaminhamento e roteamento de pacotes, variação dos *buffers*, uso de canais virtuais, dentre outros. Também é visto como trabalho futuro a implementação de uma arquitetura de NoC que considere o acesso à memória compartilhada para o tráfego de grandes pacotes de dados conforme avaliação feita no capítulo 4.

REFERÊNCIAS

KUMAR, R.; ZYUBAN, V.; TULLSEN, D. M. Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling. COMPUTER ARCHITECTURE, ISCA '05. 32ND INTERNATIONAL SYMPOSIUM. IEEE, USA, 2005.

FREITAS, H. C.; NAVAUX, P. O. A. Networks-on-Chip: Conceitos, Arquitetura e Tendências. WSCAD-SSC 2009: X SIMPÓSIO EM SISTEMAS COMPUTACIONAIS. 2009.

BENINI, L.; MICHELI, G. D. Networks on chips: a new soc paradigm. IEEE COMPUTER, v.1, p.70-78, 2002.

BENINI, L.; MICHELI, G. D. Network-on-chip architectures and design methods. IEEE PROCEEDINGS COMPUTERS DIGITAL TECHNIQUES, v.152, p.261-272, 2005.

AGARWAL, A.; ISKANDER, C.; SHANKAR, R. Survey of network on chip (noc) architectures and contributions. ENGINEERING, COMPUTING AND ARCHITECTURE, v.3, 2009.

CARLONI, L.; PANDE, P.; XIE, Y. Networks-on-chip in emerging interconnect paradigms: Advantages and challenges. NETWORKS-ON-CHIP, 2009. NOCS 2009. 3RD ACM/IEEE INTERNATIONAL SYMPOSIUM, IEEE, p.93-102, 2009.

BOKHARI, P. S. H. On the mapping problem. IEEE TRANS. COMPUT. p.207-214, 1981.

(NAS), N.A.S. NAS PARALLEL BENCHMARKS (NPB). 2010. National Aeronautics and Space Administration (NASA). DisponÍvel em: <http://www.nas.nasa.gov/Software/NPB/>.

MOORE, G. E. Cramming more components onto integrated circuits. ELETRONICS. v.38. 1965.

BORKAR, S. Thousand core chips - a technology perspective. DESIGN AUTOMATION CONFERENCE, 2007, p.746-749, 2007.

FREITAS, H. C.; ALVES, M. A.; NAVAUX, P. O. A. Noc e nuca: Conceitos e tendencias para arquiteturas de processadores many-core. ESCOLA REGIONAL DE ALTO DESEMPENHO (ERAD), Cap.3, 2009.

BJERREGAARD, T.; MAHADEVAN, S. A survey of research and practices of Network-on-chip. CM COMPUT. SURV., ACM, New York, NY, USA, v.38, p.1, 2006.

CASTRO, M. et. al. Analysis of Computing and Energy Performance of Multicore, NUMA, and Manycore Platforms for an Irregular Application. IN: WORKSHOP ON IRREGULAR APPLICATIONS: ARCHITECTURES ALGORITHMS (IA3) - SUPER-

COMPUTING CONFERENCE (SC). Dever, USA: ACM, 2013.

CIDON, I. Networks-on-chip in a three-dimensional environment: A performance evaluation. IEEE TRANSACTIONS ON COMPUTERS, IEEE, v.58, p.32-45, 2009.

CIDON, I. Noc: Network or chip? 2007. NOCS 2007. FIRST INTERNATIONAL SYMPOSIUM NETWORKS-ON-CHIP, IEEE, p.269-269, 2007.

GONÇALVES, N. A. Análise e Simulação de Topologias de Redes em Chip. Dissertação (Mestrado). Univerdidade Estadual de Maringá, 2010.

FREITAS, H. C.; NAVAUX, P.; SANTOS, T. Noc architecture design for multi-cluster chips. FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, 2008. FPL 2008. INTERNATIONAL CONFERENCE ON, P. 53-58, 2008.

ANGULY, A. et al. Performance evaluation of wireless networks on chip architectures. QUALITY OF ELECTRONIC DESIGN, 2009. ISQED 2009. QUALITY ELECTRONIC DESIGN, IEEE, San Jose, CA, p.350-355, 2009.

AMORIM, A. M. P.; OLIVEIRA, P. A. C.; FREITAS, H. C. Integrando traços de execução de aplicações paralelas ao Network Simulator para simulaçãoo de WiNoC. WORKSHOP DE INICIAÇÃO CIENTÍFICA (WSCAD-WIC 2012). XIII Simpósio em Sistemas Computacionais (WSCAD-SSC), 2012.

SALMINEN E.; KULMALA, A.; HAMALAINEN T. D. Survey of Network-on-chip Proposals. white-paper. OCP-IP. p.1-13. 2008.

RENTALA, V.; LEHTONEN, T.; PLOSILA, J. Network on Chip Routing Algorithms. Dissertação (Mestrado) | University of Turku, Departament of Information Technology. Finland, 2006.

TEDESCO, L. P. Monitoração e Roteamento Adaptativo para Fluxos QoS em NoC's. Dissertação (Mestrado) | Pontifícia Universidade Católica do Rio Grande do Sul, 2006.

DUMMLER, J.; RAUBER, T.; RUNGER, G. Mapping algorithms for multiprocessor tasks on multi-core clusters. 37TH INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, p. 141-148, 2008. ISSN 0190-3918.

CRUZ, E. H. M.; ALVES, M. A. Z.; NAVAUX, P. O. A. Process mapping based on memory access traces. 11TH SYMPOSIUM ON COMPUTING SYSTEMS, WSCAD - SSC, Brazil, 2010.

MURALI, S.; MICHELI, G. D. Sunmap: a tool for automatic topology selection and generation for nocs. DESIGN AUTOMATION CONFERENCE, 2004. PROCEEDINGS. 41st, p. 914-919, 2004.

ASCIA, G.; CATANIA, V.; PALESI, M. Multi-objective mapping for mesh-based NoC

architectures. IN: CODES+ISSS '04: PROCEEDINGS OF THE 2ND IEEE/ACM/IFIP INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODESIGN AND SYSTEM SYNTHESIS. New York, NY, USA: ACM, 2004. p. 182-187. ISBN 1-58113-937-3.

CALAZANS, C.; MORAES, N.; SUSIN, F.; REIS, I.; MARCON, F. Exploring noc mapping strategies: an energy and timing aware technique. ACM, New York, NY, USA, p. 182-187, 2005.

HU, J.; MARCULESCU, R. Energy-aware mapping for tile-based NoC architectures under performance constraints. ASP-DAC '03: PROCEEDINGS OF THE 2003 ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE. IEEE. p.502-507. Rio Grande do Sul, Brazil. 2003. ISBN 0-7803-7660-9

CARVALHO, E. C.; N. MORAES, F. Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs. RAPID SYSTEM PROTOTYPING, 2007. RSP 2007. 18TH IEEE/IFIP INTERNATIONAL WORKSHOP ON, IEEE, p. 34-40, 2007. ISSN 1074-6005.

CHOU, C. L.; MARCULESCU, R. Contention-aware application mapping for network-on-chip communication architectures. COMPUTER DESIGN, 2008. ICCD 2008. IEEE INTERNATIONAL CONFERENCE ON, IEEE, p. 164-169, 2008. ISSN 1063-6404.

TORNERO, R. et al. A communication-aware topological mapping technique for nocs. IN: PROCEEDINGS OF THE 14TH INTERNATIONAL EURO-PAR CONFERENCE ON PARALLEL PROCESSING. [S.l.: s.n.], 2008. p. 910-919.

CHEN, G. et al. Application mapping for chip multiprocessors. IN: DAC '08: PROCEEDINGS OF THE 45TH ANNUAL DESIGN AUTOMATION CONFERENCE. New York, NY, USA: ACM, 2008. p. 620-625. ISBN 978-1-60558-115-6.

LEE, S.-H.; C, Y.; HWANG, S.-Y. Communication-aware task assignment algorithm for mpsoc using shared memory. EUROMICRO JOURNAL OF SYSTEMS ARCHITECTURE, IEEE, v. 56, p. 233-241, 2010.

MARCON, C.A.M.; MORENO, E.I.;CALAZANS, N.L.V.; MORAES, F.G. Evaluation of Algorithms for Low Energy Mapping onto NoCs. CIRCUITS AND SYSTEMS, 2007. ISCAS 2007. IEEE INTERNATIONAL SYMPOSIUM ON, IEEE. p.389-392. New Orleans, LA. 2007.

OST, L. et al. Exploring Dynamic Mapping Impact on NoC-based MPSoCs Performance Using a Model-based Framework. PROCEEDING SBCCI '11 PROCEEDINGS OF THE 24TH SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN. ACM. p.185-190. New York, NY, USA. 2011.

AVELAR, C. P. et al. Evaluating the problem of process mapping on network-on-chip for parallel applications. SECOND WORKSHOP ON ARCHITECTURE AND MULTICORE APPLICATIONS (WAMCA), 2011.

OLIVEIRA, P. A. C. et al. A greedy heuristic for process mapping on networks-on-chip. SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHo (WSCAD-SSC), 2011.

PELLEGRINI, F. SCOTCH 5.1 User's guide. TECHNICAL REPORT, LABRI. 2008.

BERTSEKAS, D. The auction algorithm: A distributed relaxation method for the assignment problem. ANNALS OF OPERATIONS RESEARCH, 14(1):105–123. 1988.

ABAD, P. et. al. TOPAZ: An Open-Source Interconnection Network Simulator for Chip Multiprocessors and Supercomputers. NOCS. 2012.

BINKERT, N. The GEM5 simulator. ACM SIGARCH COMPUTER ARCHITECTURE NEWS. p.1. v.39, 2011.

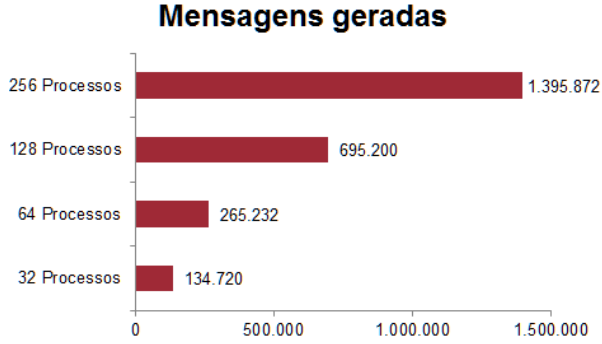
OLIVEIRA, P. A. C. Avaliação de Desempenho e Caracterização de Cargas de Trabalho Paralelas para Redes-em-Chip Sem Fio. Dissertação (Mestrado). Pontifícia Universidade Católica de Minas Gerais. 2012.

APÊNDICE A – RESULTADOS COMPLEMENTARES

As mensagens geradas e as mensagens entregues em cada aplicação podem ser vistas nos gráficos a seguir. Na análise dessa métrica é possível observar que não houve uma diferença significativa na entrega das mensagens diante das três heurísticas de mapeamento apresentadas nesse trabalho. Isso mostra que nenhum algoritmo influenciou na perda de mensagens na rede. As análises a seguir serão feitas com base nas transmissões de pacotes de 10 *flits* como exemplo da quantidade de mensagens entregues para execução de cada aplicação.

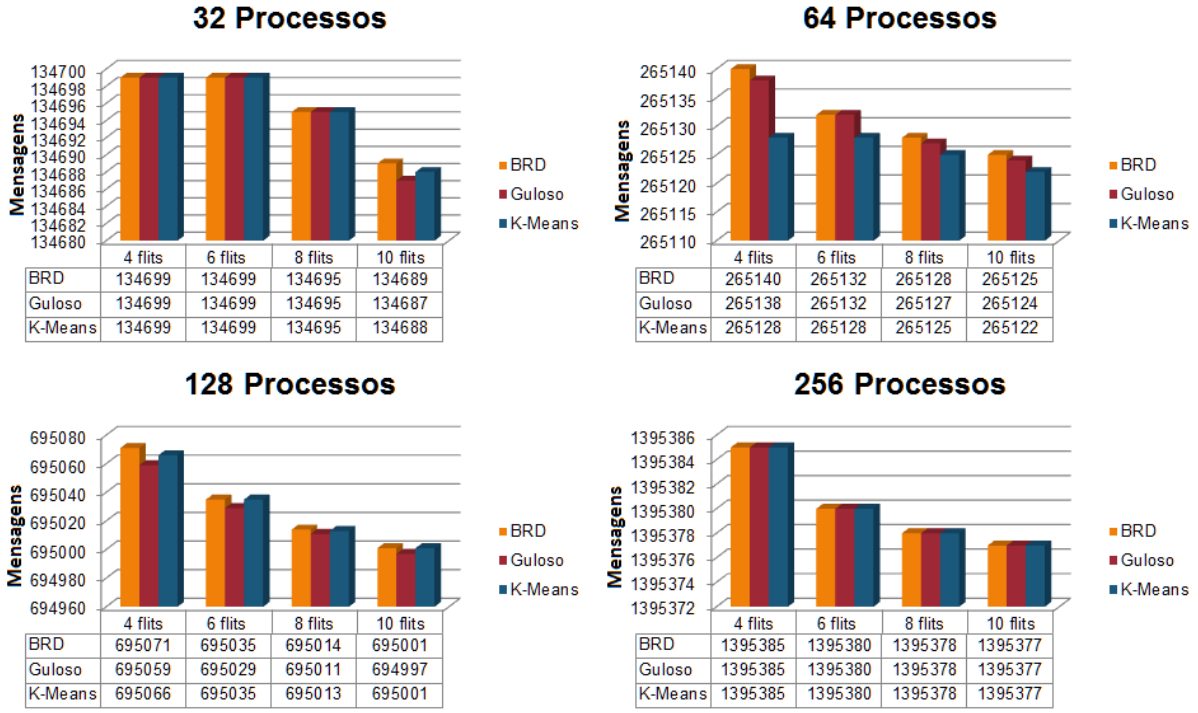
No programa CG é possível notar que não há grandes perdas de mensagens durante sua execução. A Figura 31 mostra o número de mensagens geradas para cada tamanho de rede e na Figura 32 contém a quantidade de mensagens entregues na execução de cada algoritmo de mapeamento e para todos os tamanhos de pacotes utilizados nos testes. Nos resultados com 32 processos do programa CG houve uma perda muito pequena de, aproximadamente, 0,02% das mensagens com a utilização dos três algoritmos de mapeamento. O cenário é parecido nos resultados com 64 processos: cerca de 0,04% das mensagens foram perdidas com a utilização das três heurísticas. No caso de uma rede com 128 processos, a perda foi de 0,02%, aproximadamente, diante dos três mapeamentos. E nos experimentos com 256 processos, 0,03% das mensagens foram perdidas.

Figura 31 – Gráfico de mensagens geradas na aplicação CG.



Fonte: Dados da pesquisa.

Figura 32 – Gráficos de mensagens entregues na aplicação CG.



32 Processos	4 flits	6 flits	8 flits	10 flits
BRD	134699	134699	134695	134689
Guloso	134699	134699	134695	134687
K-Means	134699	134699	134695	134688

64 Processos	4 flits	6 flits	8 flits	10 flits
BRD	265140	265132	265128	265125
Guloso	265138	265132	265127	265124
K-Means	265128	265128	265125	265122

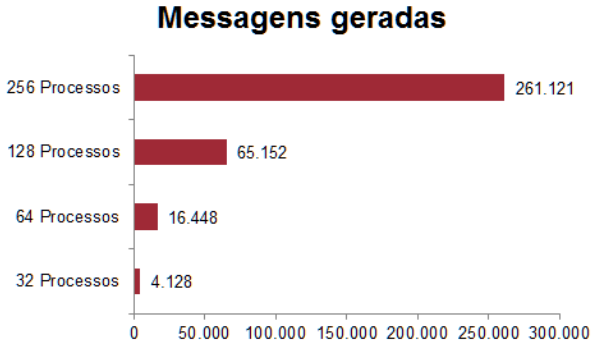
128 Processos	4 flits	6 flits	8 flits	10 flits
BRD	695071	695035	695014	695001
Guloso	695059	695029	695011	694997
K-Means	695066	695035	695013	695001

256 Processos	4 flits	6 flits	8 flits	10 flits
BRD	1395385	1395380	1395378	1395377
Guloso	1395385	1395380	1395378	1395377
K-Means	1395385	1395380	1395378	1395377

Fonte: Dados da pesquisa.

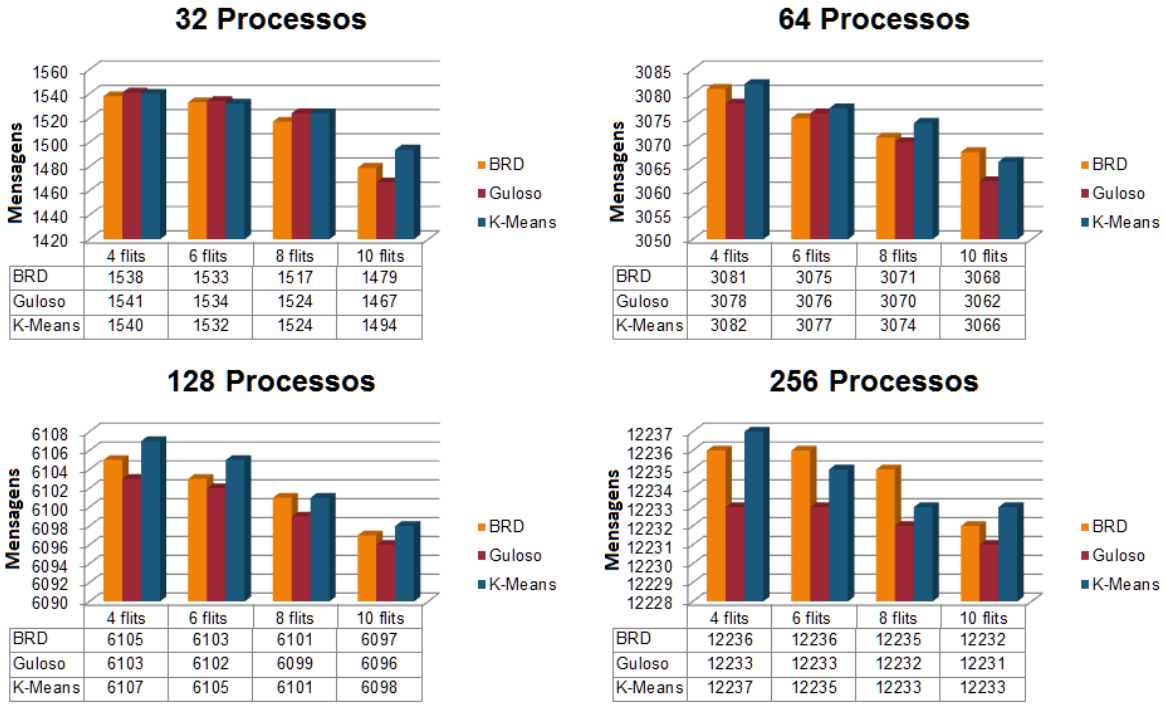
Nas Figuras 33 e 34 são apresentadas as quantidades de mensagens geradas e entregues, respectivamente, na execução da aplicação EP. Diante dos resultados das mensagens entregues é possível perceber que houve uma grande perda de pacotes na execução deste programa. Nos testes com 32 processos, aproximadamente, 64% das mensagens foram perdidas com o uso dos três algoritmos. Na execução com 64 processos, a perda foi de, aproximadamente, 81,38% na utilização dos três algoritmos de mapeamento analisados. No cenário dos testes com 128 processos a perda é ainda maior: cerca de 90,64% das mensagens não foram entregues aos núcleos de destino. A perda de mensagens na rede de 256 processos foi maior, aproximadamente, 95,31% das mensagens foram perdidas com o uso das três heurísticas de mapeamento.

Figura 33 – Gráfico de mensagens geradas na aplicação EP.



Fonte: Dados da pesquisa.

Figura 34 – Gráficos de mensagens entregues na aplicação EP.



	4 flits	6 flits	8 flits	10 flits
BRD	1538	1533	1517	1479
Guloso	1541	1534	1524	1467
K-Means	1540	1532	1524	1494

	4 flits	6 flits	8 flits	10 flits
BRD	3081	3075	3071	3068
Guloso	3078	3076	3070	3062
K-Means	3082	3077	3074	3066

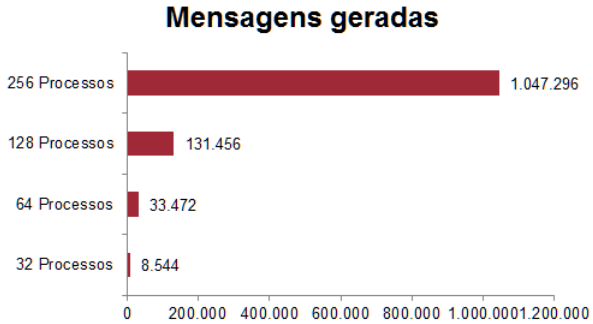
	4 flits	6 flits	8 flits	10 flits
BRD	6105	6103	6101	6097
Guloso	6103	6102	6099	6096
K-Means	6107	6105	6101	6098

	4 flits	6 flits	8 flits	10 flits
BRD	12236	12236	12235	12232
Guloso	12233	12233	12232	12231
K-Means	12237	12235	12233	12233

Fonte: Dados da pesquisa.

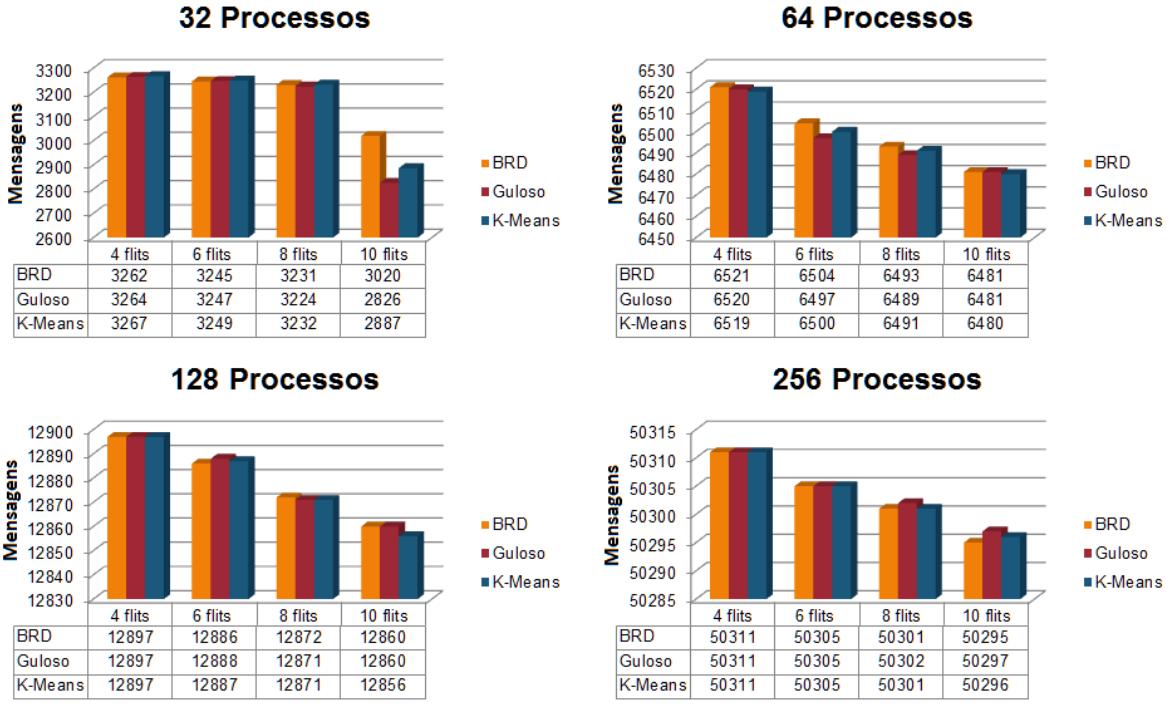
Assim como na aplicação EP, no programa FT também houve uma grande perda de mensagens na rede durante a execução dos testes, como pode ser visto nos gráficos das Figuras 35 e 36. Com 32 processos houve uma perda de 64,65% das mensagens com a utilização do BRD, 66,92% com o Guloso e 66,21% com o K-means. Já nos testes com 64 processos, aproximadamente, 19,36% das mensagens foram entregues aos seus destinos, com o uso das três heurísticas. No caso de uma rede de 128 núcleos, foram entregues apenas 9,78% das mensagens, aproximadamente, com os três algoritmos de mapeamento. No cenário dos testes com 256 processos, a perda ainda foi maior: cerca de 4,80% das mensagens foram entregues.

Figura 35 – Gráfico de mensagens geradas na aplicação FT.



Fonte: Dados da pesquisa.

Figura 36 – Gráficos de mensagens entregues na aplicação FT.



	4 flits	6 flits	8 flits	10 flits
BRD	3262	3245	3231	3020
Guloso	3264	3247	3224	2826
K-Means	3267	3249	3232	2887

	4 flits	6 flits	8 flits	10 flits
BRD	6521	6504	6493	6481
Guloso	6520	6497	6489	6481
K-Means	6519	6500	6491	6480

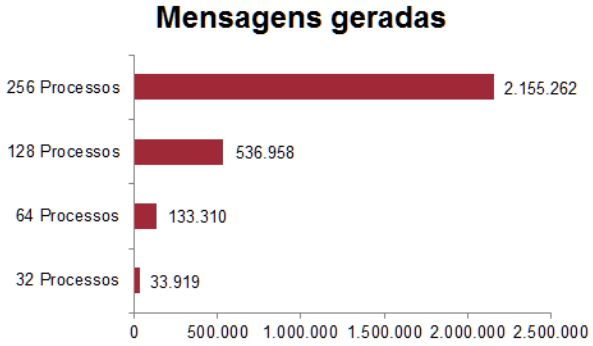
	4 flits	6 flits	8 flits	10 flits
BRD	12897	12886	12872	12860
Guloso	12897	12888	12871	12860
K-Means	12897	12887	12871	12856

	4 flits	6 flits	8 flits	10 flits
BRD	50311	50305	50301	50295
Guloso	50311	50305	50302	50297
K-Means	50311	50305	50301	50296

Fonte: Dados da pesquisa.

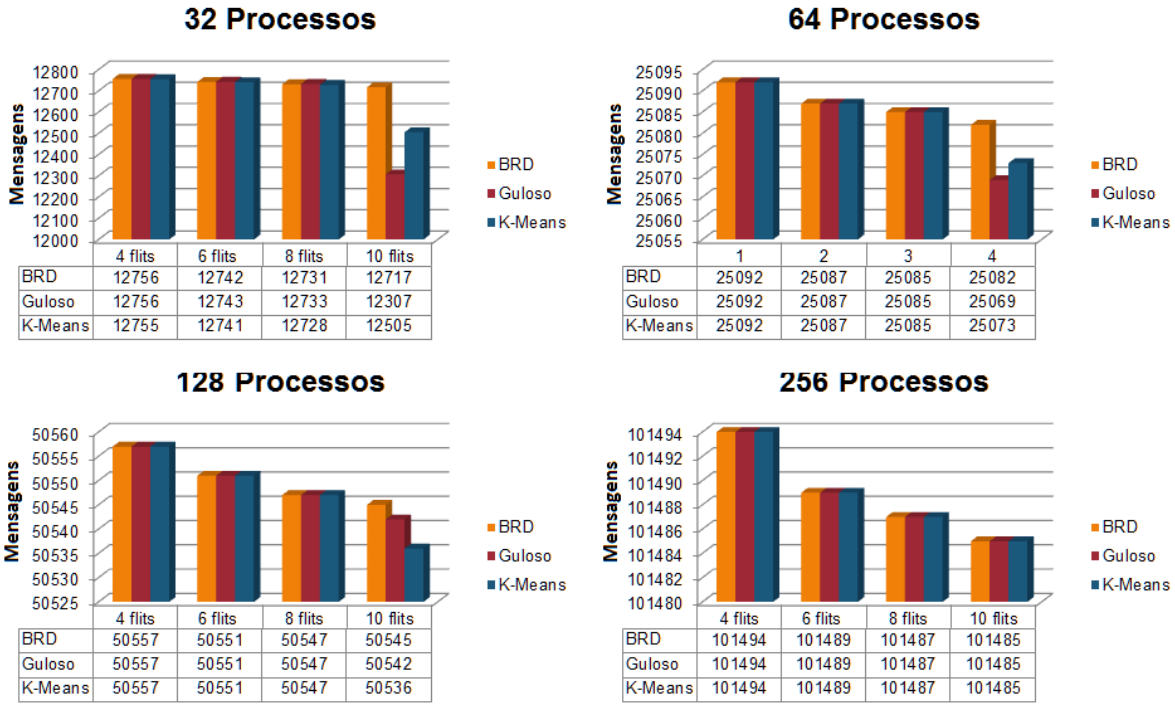
Nas Figuras 37 e 38 são apresentados os gráficos de número de mensagens geradas e entregues na execução da aplicação IS. De forma geral, pode-se perceber que assim como as aplicações EP e FT, houve muita perda de mensagens. Nos testes de 32 processos, 37,49% das mensagens foram entregues com o uso do BRD e, aproximadamente, 36% com o Guloso e o K-means. Com 64 processos, cerca de 18,8% das mensagens foram entregues aos núcleos destinos. Já com 128 processos, apenas 9,41% das mensagens, aproximadamente, foram entregues. O número de mensagens entregues nos testes com 256 processos foi ainda menor: cerca de 4,70%, na utilização das três heurísticas de mapeamento.

Figura 37 – Gráfico de mensagens geradas na aplicação IS.



Fonte: Dados da pesquisa.

Figura 38 – Gráficos de mensagens entregues na aplicação IS.



	4 flits	6 flits	8 flits	10 flits
BRD	12756	12742	12731	12717
Guloso	12756	12743	12733	12307
K-Means	12755	12741	12728	12505

	1	2	3	4
BRD	25092	25087	25085	25082
Guloso	25092	25087	25085	25069
K-Means	25092	25087	25085	25073

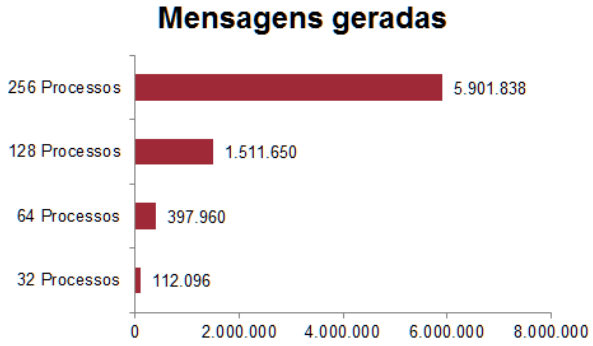
	4 flits	6 flits	8 flits	10 flits
BRD	50557	50551	50547	50545
Guloso	50557	50551	50547	50542
K-Means	50557	50551	50547	50536

	4 flits	6 flits	8 flits	10 flits
BRD	101494	101489	101487	101485
Guloso	101494	101489	101487	101485
K-Means	101494	101489	101487	101485

Fonte: Dados da pesquisa.

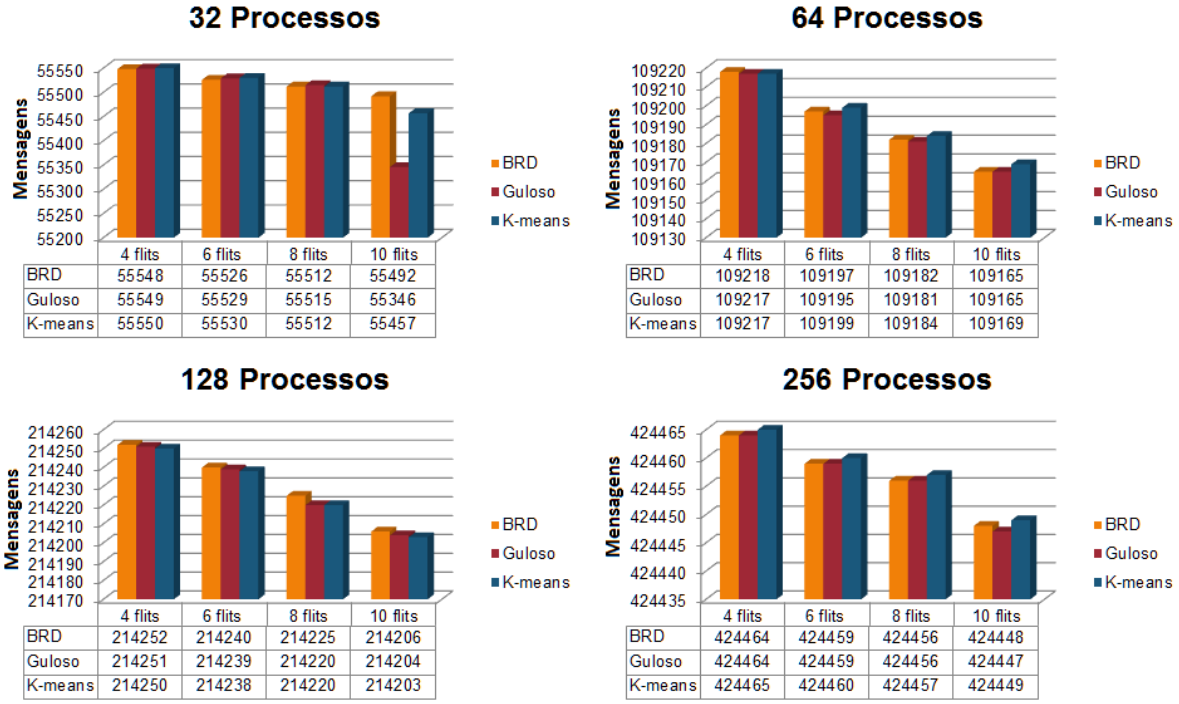
A quantidade de mensagens geradas e entregues na execução da aplicação MG pode ser visualizada nas Figuras 39 e 40. Nos testes realizados com 32 processos, 49,50% das mensagens foram entregues aos núcleos de destinos. Para uma rede com 64 núcleos, 72,57% das mensagens foram perdidas. Já nos experimentos com 128 pacotes, apenas 14,17% das mensagens foram entregues. E, por fim, nos testes com 256 processos houve uma perda de 92,80% das mensagens.

Figura 39 – Gráfico de mensagens geradas na aplicação MG.



Fonte: Dados da pesquisa.

Figura 40 – Gráficos de mensagens entregues na aplicação MG.



	4 flits	6 flits	8 flits	10 flits
BRD	55548	55526	55512	55492
Guloso	55549	55529	55515	55346
K-means	55550	55530	55512	55457

	4 flits	6 flits	8 flits	10 flits
BRD	109218	109197	109182	109165
Guloso	109217	109195	109181	109165
K-means	109217	109199	109184	109169

	4 flits	6 flits	8 flits	10 flits
BRD	214252	214240	214225	214206
Guloso	214251	214239	214220	214204
K-means	214250	214238	214220	214203

	4 flits	6 flits	8 flits	10 flits
BRD	424464	424459	424456	424448
Guloso	424464	424459	424456	424447
K-means	424465	424460	424457	424449

Fonte: Dados da pesquisa.

A porcentagem baixa de pacotes entregues justifica uma metodologia que privilegia pacotes pequenos, considerando que comunicações com grandes mensagens ocorre via uma rede secundária de memória, e não núcleo/núcleo. Além disso, os tempos de início das comunicações entre os processos, originárias do *trace* e da execução em *clusters* de computadores, sugerem que a execução e comunicação em uma rede-em-chip de um processador *many-core* devam ser diferentes, refletindo, inclusive, na programação das aplicações e na gerência da rede.