Algoritmo *Kmeans* para Mapeamento Estático de Processos em Redes-em-Chip

Cíntia P. Avelar, Pedro H. Penna, Henrique C. Freitas

Grupo de Arquitetura de Computadores e Processamento Paralelo (CArT)
Pontifícia Universidade Católica de Minas Gerais
CEP 30.535-901 – Belo Horizonte – MG – Brasil

{cintia.avelar, pedro.penna}@sga.pucminas.br, cota@pucminas.br

Abstract. *Performance is an important issue for many-core architectures supported by networks-on-chip. One alternative for improving it is to map processes onto cores so as to mitigate the overall interprocess communication cost. In this context, this paper proposes the Kmeans algorithm as an alternative strategy to DRB and Greedy heuristics. For some communication patterns, our results pointed that Kmeans performs better than the other heuristics, thus being a good option for mapping processes on many-core architectures with networks-on-chip support.*

Resumo. *Desempenho é um ponto crucial em arquiteturas many-core com networks-on-chip. Uma das alternativas para alcançá-lo consiste em mapear processos nos núcleos de processamento de forma a minimizar o custo de comunicação global entre processos. Nesse contexto, esse trabalho propõe o algoritmo Kmeans como uma estratégia alternativa às heurísticas BRD e Guloso. Para determinados padrões de comunicação, os resultados de simulação apontaram que o Kmeans conduz a melhores mapeamentos que as outras estratégias, sendo portanto uma boa opção para o mapeamento de processos em arquiteturas many-core com networks-on-chip.*

1. Introdução

Os processadores *many-core* surgiram para suprir a crescente demanda por desempenho requisitada pelas aplicações atuais, integrando em um único *chip* múltiplos núcleos de processamento. Nessa nova arquitetura, alguns dos problemas relacionados às estruturas de interconexão (*i.e* barramentos) são recorrentes, como a resistência na propagação dos dados, a competição pelo canal de comunicação e o roteamento físico da interconexão. Com o objetivo de minimizar o impacto causado por esses problemas, foram introduzidas as *Networks-on-Chips* (NoCs), uma estrutura onde os núcleos são interconectados por uma rede *intra-chip* e comunicam-se através da troca de pacotes de dados.

A Figura 1 apresenta os três elementos básicos de uma NoC: roteador, *links* de dados e interface de rede. O roteador é responsável pela comunicação entre os núcleos, sendo encarregado de definir as rotas dos pacotes, controlar o fluxo de dados e garantir a qualidade de serviço. Os *links* de dados correspondem aos fios de interconexão por onde os pacotes irão passar, são eles quem interligam os roteadores e formam a topologia da rede. Por último, a interface de rede é responsável pela troca de dados entre o roteador e o núcleo, garantindo correta comunicação entre os diferentes protocolos [Freitas et al. 2009].

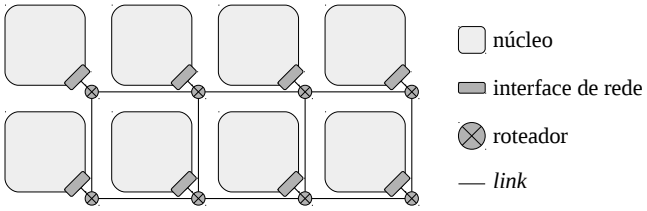

Figura 1. NoC *Mesh* 2 × 4.

NoCs conferem maior potencial de escalabilidade, flexibilidade e desempenho às arquiteturas *many-core*, quando comparadas às estruturas de interconexão tradicionais (*i.e* barramentos e chaves *crossbar*). No entanto, essas estruturas ainda possuem limitações relevantes de *hardware* e de projeto que devem ser estudadas, podendo-se citar: a latência de comunicação, o consumo de energia, o tráfego na rede e os modelos de programação.

Nesse contexto, este trabalho aborda, como tema principal, o problema de mapeamento de processos em arquiteturas *many-core* com NoCs. Esse problema, que pertence à classe NP-Difícil [Oliveira et al. 2011] [Bokhari 1981], consiste em definir uma distribuição de processos nos núcleos com o objetivo de reduzir a comunicação na rede e, consequentemente, o tempo de execução de uma aplicação paralela.

Portanto, o presente trabalho propõe uma nova estratégia que se baseia no algoritmo de clusterização *Kmeans* para efetuar o mapeamento estático de processos em arquiteturas *many-core* com NoCs. A heurística aqui proposta é avaliada quantitativamente segundo diferentes cargas de trabalho e é comparada com duas outras estratégias de mapeamento estático que vêm sendo estudadas: o Biparticionamento Recursivo Dual [Pellegrini 2008] e a Heurística Gulosa [Oliveira et al. 2011].

O restante deste trabalho está organizado da seguinte forma: na Seção 2 são apresentados alguns trabalhos correlatos; na Seção 3 são descritas as heurísticas de mapeamento estudadas; na Seção 4 é apresentada a metodologia utilizada; na Seção 5 exposta a análise de resultados; e na Seção 6 são apresentadas as conclusões e os trabalhos futuros.

2. Trabalhos Correlatos

Tendo conhecimento que o mapeamento de processos no contexto de NoCs é um problema, [Ascia et al. 2004] apresenta mapeamentos de dados para NoC baseadas em *Chip Multiprocessors*, com o objetivo de 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 que possuam diferentes fases de execução com padrões de acesso à dados potencialmente diferentes entre si.

Em [Carvalho and 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 *mesh* 8x8. A estratégia é baseada em um processador gerente, 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* internos da NoC.

No trabalho de Avelar et al [Avelar et al. 2011], é realizada uma avaliação do problema de mapeamento de processos em NoCs. Para isso, os autores limitam o escopo

do trabalho ao estudo de NoCs com topologias *mesh* 2D 4x8 e avaliam três diferentes estratégias para mapeamento estático de processos: (I) direto por linha, (II) direto por coluna e (III) *communication-aware*. Os mapeamentos direto por linha e direto por coluna são bastante similares: ambos ignoram o custo de comunicação entre os processos mapeando-os linha-a-linha e coluna-a-coluna, respectivamente. Já no mapeamento *communication-aware*, os processos são primeiramente ordenados pelo número de comunicações e, depois, mapeados em sequência na NoC. A ideia desta última estratégia é de mapear processos que realizam muitas comunicações em núcleos próximos. Os autores avaliam as três estratégias utilizando uma metodologia de simulação baseada em *traces* de execução de aplicações do *NAS Parallel Benchmarks*. Ao final, os resultados revelam que as três estratégias proporcionam o mesmo *throughput* e autores concluem que isso se deve ao padrão de comunicação coletiva das aplicações selecionadas para o estudo.

No trabalho de Oliveira et al [Oliveira et al. 2011], é proposto um algoritmo guloso para o mapeamento estático de processos em NoCs com topologia *mesh* 2D e protocolo de roteamento XY. A ideia central desse algoritmo é de identificar o processo que se comunica mais e atribuí-lo ao núcleo que possui o maior número de *links*. Em seguida, até que todos os processos estejam mapeados escolhe-se o processo vizinho que ainda não está mapeado e que possui o maior número de comunicações com o que foi mapeado e escolhe-se o núcleo disponível mais próximo do último mapeado. Esse algoritmo possui complexidade de $O(n^2)$ no pior caso mas garante solução ótima local. Para avaliar o ganho de desempenho proporcionado por essa estratégia, os autores adotaram a mesma metodologia de [Avelar et al. 2011]. Ao final constatou-se que o algoritmo possibilita um ganho em até 23.04% em *throughput* quando comparado à estratégia de mapeamento direto por linha.

Este trabalho se diferencia dos trabalhos relacionados apresentados anteriormente em dois pontos: em primeiro lugar ele propõe um novo algoritmo para o mapeamento estático de processos baseado em na heurística de clusterização *Kmeans*; e em segundo lugar as cargas de trabalho são avaliadas sob quatro redes *mesh* de diferentes tamanhos.

3. Heurísticas de Mapeamento de Processos

Nesta seção são apresentadas as três heurísticas de mapeamento de processos consideradas neste trabalho: a Heurística Gulosa, que realiza uma busca local para resolver o problema; o Biparticionamento Recursivo Dual (BRD), que se baseia na técnica de divisão-e-conquista; e a heurística de clusterização, que se fundamenta no algoritmo *Kmeans*.

3.1. BRD

O algoritmo heurístico BRD foi inicialmente proposto por [Pellegrini 2008] e baseia-se na técnica de divisão-e-conquista. Nessa heurística, a ideia consiste em particionar recursivamente os conjuntos de processos e processadores e, então, construir um mapeamento a partir de um caso base.

Para tanto, a aplicação considerada é representada por um grafo valorado e não direcionado, no qual os vértices indicam os processos da aplicação paralela, as arestas representam as comunicações entre os processos e os pesos das arestas representam o custo de comunicação entre os processos. A arquitetura também é representada por um

grafo não direcionado, no qual os vértices representam os núcleos de processamento e as arestas representam os *links* de comunicação entre os núcleos.

O algoritmo proposto em [Pellegrini 2008] é apresentado na Figura 2 e brevemente discutido a seguir. Inicialmente, o algoritmo considera todos os núcleos de processamento D e processos P. Em seguida, o algoritmo particiona o conjunto de núcleos de processamento (PROCESSOR-BIPARTITION()) e processos (PROCESS-BIPARTITION()), considerando o custo de comunicação entre processos nessa etapa. Por fim, o algoritmo entra em recursão até que o caso base para mapeamento seja alcançado: mapear um processo em um núcleo de processamento.

```
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)
```

Figura 2. Heurística BRD.

3.2. Heurística Gulosa

A heurística gulosa proposta em [Oliveira et al. 2011] baseia-se na técnica de busca local. A ideia consiste em mapear os processos com maior custo de comunicação para os núcleos que possuam o maior número de *links*, de forma que, ao final, um bom mapeamento seja encontrado. Essa solução é apresentada na Figura 3 e detalhada a seguir.

A heurística gulosa opera sobre duas estruturas: um grafo não direcionado G_p que representa a topologia da rede, e um grafo valorado e não direcionado G_n que representa as comunicações entre os processos. Inicialmente as duas estruturas são consultadas para se identificar o processo p_1 com o maior número de comunicações e o núcleo n_1 como maior número de *links*. Feito isso, atribui-se o processo p_1 ao núcleo n_1. Em seguida, as duas estruturas são consultadas novamente para que se identifique (I) o processo p_2 que ainda não foi não mapeado e que possua o maior número de comunicações com o último processo mapeado; e (II) o núcleo n_2 que ainda encontra-se disponível e que possua o maior número de *links* com o último núcleo considerado. Uma vez encontrados, atribui-se o processo p_2 ao núcleo n_2 e o procedimento é repetido até que todos os processos estejam mapeados.

```
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
```

Figura 3. Heurística gulosa.

3.3. Heurística de Clusterização

A heurística para mapeamento de processos proposta neste trabalho é apresentado na Figura 4. Essa solução baseia-se no algoritmo de clusterização *Kmeans*, que adota 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.

Esse algoritmo 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 centróide de cada *cluster* a partir da média dos custos de comunicação dos processos que o constituem (CLUSTERS-COMPUTE-MEANS()). Depois, os processos são reagrupados considerando sua distância euclidiana para os centróides de cada *cluster*, sendo atribuídos ao *cluster* mais próximo (CLUSTERS-REPOPULATE()).

Enquanto todas os processos encontrarem-se a uma distância maior que d do centróide do seu respectivo *cluster* e o conjunto de processos que constitui um *cluster* mudar, o algoritmo continua a recalcular os centróides de cada *cluster* e reagrupar todos 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())

```
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
```

Figura 4. Heurística de clusterização.

4. Metodologia

Para avaliar quantitativamente o ganho de desempenho proporcionado pelas heurísticas avaliadas, optou-se por uma metodologia de simulação que será detalhada a seguir. No estudo, considerou-se: (I) uma NoC com protocolo de roteamento XY e topologia *Mesh* 2D; (II) *clusters* de tamanho 4, para a heurística de clusterização; e (III) a latência para entrega de pacotes e distância média de comunicação como métricas.

Primeiramente, foram selecionadas como cargas de trabalho as aplicações *Conjugate Gradient* (CG), *Embarassingly Parallel* (EP), *Fast Fourier Transform* (FT), *Integer Sort* (IS) e *Multigrid* (MG); todas do *benchmark* paralelo *NAS Parallel Benchmark* (NPB).

Em seguida, essas aplicações foram compiladas com suporte à passagem de mensagens e foram executadas em um *cluster* instrumentado para coletar *traces* de execução, com informações sobre a troca de pacotes (*processo* de origem, *processo* de destino e tamanho de pacotes). Para cada aplicação foram gerados *traces* de execução para instâncias com 32, 64, 128 e 256 processos.

Para cada comunicação foi calculado um custo para que a mensagem chegue ao destino final. O cálculo foi feito considerando as seguintes variáveis: tamanho do pacote de dados a ser enviado, o número de comunicações entre os dois processos e o número de saltos entre os núcleos de origem e destino. Todos esses valores foram multiplicados para gerar o custo de cada comunicação. Para calcular o número de saltos, considerou-se núcleos dispostos em uma rede de topologia *Mesh 2D*. Estes custos entre cada comunicação são utilizados para gerar o mapeamento para cada *trace* de execução.

A simulação foi o método escolhido para a avaliação dos resultados do comportamento da rede diante dos mapeamentos de processos. O ambiente de simulação de NoCs escolhido foi o Topaz [Abad et al. 2012]. Este é um simulador de redes de interconexão de propósito geral que permite a modelagem de uma ampla variedade de roteadores. Além disso, 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, como topologias, controle de fluxo, padrões de tráfego, roteadores, dentre outros. O simulador oferece, também, diversas métricas para analisar o desempenho da rede. Dentre eles estão: mensagens geradas, recebidas e injetadas; latência da rede, dos *buffers* e latência total; *throughput*; consumo de memória; e performance dos canais virtuais.

Em uma NoC é importante que sejam considerados, também, o tamanho dos pacotes que trafegam pela rede, pois pacotes muito grandes podem gerar gargalos, perdas ou atrasos na entrega. Para evitar tráfego intenso na rede, deve-se considerar a limitação dos tamanhos dos pacotes. Para que a troca de informação não fique prejudicada, foi definido que, para pacotes muito grandes de uma aplicação em simulação no Topaz, o núcleo de origem envia ao de destino um pacote pequeno, contendo o endereço de memória de onde o conjunto de dados se encontra. Tem-se como hipótese que, limitando o tamanho do pacote, os custos de comunicação na NoC são reduzidos e o desempenho não sofre prejuízos. Portanto, nesse artigo, avalia-se a comunicação entre núcleos com pacotes pequenos e de tamanho fixo, com a hipótese de que a comunicação núcleo-memória deve ser feita por outra rede ou interconexão em *chip*.

5. Análise de Resultados

Essa seção apresenta os resultados dos testes e a análise do comportamento/desempenho da rede para cada heurística e carga de trabalho. Na realização dos testes foram utilizados pacotes de 10 *flits*. Considera-se um *flit* igual a um *byte*. Pode-se obter maiores variações de tamanho de pacotes e possíveis alterações nos resultados das simulações quando há a integração entre o Topaz e o GEM5. Como este trabalho não aborda essa integração, os testes ficaram restritos à variação citada. Para cada uma das aplicações selecionadas, foram estudadas quatro arquiteturas *many-core* com NoC (variando de 32 a 256 núcleos) e três estratégias de mapeamento, totalizando 60 configurações de execução diferentes.

5.1. Latência das Mensagens nos *Buffers*

É possível observar no gráfico da Figura 5 (b) que no caso da carga CG, não houve um algoritmo que pode ser destacado, pois as variações entre os resultados não passam de 1%. Esse fato ocorre para qualquer tamanho de rede. A aplicação CG possui um elevado número de transmissões, 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.

Diferente da aplicação CG que possui um padrão de comunicação *unicast*, a EP é caracterizada pelo alto número de comunicações *broadcast*, ou seja, todos os núcleos recebem pacotes a todo momento, o que pode aumentar o tempo de espera para serem processados. Isso justifica o aumento da latência no *buffer* com relação ao CG. Nos testes com 32 processos, o uso do *Kmeans* trouxe um ganho de 11,14% em relação ao

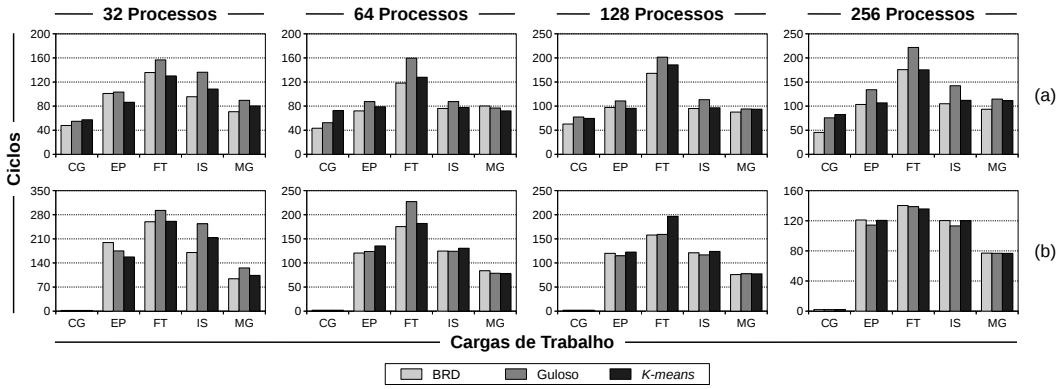

Figura 5. Resultados: (a) latência das mensagens na rede, (b) latência das mensagens nos *buffers*.

Guloso e 24,48% em relação ao BRD. Já nos testes com 64 processos, a melhor latência foi alcançada com o uso do BRD, sendo 2,63% e 12,21% mais vantajoso que o uso do Guloso e do *Kmeans*, respectivamente. Nos resultados com 128 processos, o cenário é diferente: o Guloso apresentou um resultado 4,39% e 6,64% melhor que o BRD e o *Kmeans*, nessa ordem. O uso do Guloso também foi melhor com os testes realizados com 256 processos: cerca de 5,60% melhor que com o uso do *Kmeans* e do BRD.

A aplicação FT possui, em sua maioria, um padrão de comunicação *broadcast* [de Oliveira 2012]. Os testes para latência dos *buffers* para 32 processos apresentou melhores resultados com o uso do algoritmo BRD: em comparação ao *Kmeans*, a melhora foi irrelevante, porém comparando com o Guloso, o ganho foi de 12,67%. No cenário com 64 processos, a menor latência foi alcançada utilizando o algoritmo BRD, logo após o *Kmeans*, com apenas 3,74% de diferença e o Guloso com 29,59%. Nos testes com 128 processos, o mapeamento feito pelo BRD ainda alcança os melhores resultados, porém a diferença com o uso do Guloso foi menor que 1%, já a diferença com relação ao *Kmeans* foi de 24,59%. E com 256 processos, a menor latência foi obtida com a utilização do *Kmeans*, sendo 2,21% e 3,25% menor que com o uso do Guloso e BRD, respectivamente.

As transmissões *broadcast* representam a maior parte das comunicações na aplicação IS. Mas também possui comunicações *unicast*, que a torna uma aplicação de padrão misto. Nos testes com 32 processos, a menor latência dos *buffers* foi obtida com o uso do BRD: com o uso do *Kmeans* a latência aumentou 25,39% e com o Guloso 49,13%. No cenário com 64 processos, o Guloso obteve um pequeno ganho sobre os demais: menos que 1% sobre o BRD e 4,97% sobre o *Kmeans*. 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 *Kmeans*, respectivamente. No caso dos testes com 256 processos, o uso da heurística Gulosa mostrou ser a melhor opção e em seguida o *Kmeans*, com 6,01% de diferença e 6,23% com relação ao BRD.

A aplicação MG se aproxima do padrão de comunicação da CG, por apresentar um padrão de comunicação heterogêneo, com comunicações *broadcast* e *unicast*. Nos testes com 32 processos, a menor latência foi alcançada com o uso do algoritmo BRD, cerca

de 10,50% e 33,48% menor do que com o uso do *Kmeans* e do Guloso, respectivamente. Com 64 processos é perceptível o ganho de desempenho com a utilização do *Kmeans*: 1,24% em relação ao Guloso e 7,95% comparando com o uso do BRD. Nos testes com 128 processos, há um pequeno ganho com o uso do BRD, aproximadamente, 2% em relação ao uso dos demais algoritmos. Já nos testes com 256, o mapeamento feito com o uso do *Kmeans* obteve uma latência 1% menor com relação ao uso dos demais.

5.2. Latência das Mensagens na Rede

A Figura 5 (a) apresenta os resultados dos testes com relação à latência das mensagens na rede. Nos testes realizados com a aplicação CG, de modo geral, o algoritmo que obteve melhores resultados foi o BRD, em seguida o Guloso, sendo esse cenário diferente apenas para a rede com 128 núcleos/processos, onde o *Kmeans* alcançou uma latência menor que com o uso do Guloso.

Com a aplicação EP, nos testes de 32 processos, os melhores resultados foram obtidos com o uso do algoritmo *Kmeans*. Ele foi 16,86% melhor que o BRD e 19,71% melhor que o Guloso. 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 *Kmeans* e Guloso, respectivamente. Os resultados para 128 processos se assemelham aos resultados de 32 processos: a menor latência foi obtida com o uso do *Kmeans*, cerca de 1,90% menor que o BRD e 16,03% menor que o Guloso. E por ultimo, 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 *Kmeans* e do Guloso, respectivamente.

Nos experimentos realizados coma aplicação FT, para 32 processos, os melhores resultados foram alcançados com a utilização do *Kmeans*, 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 *Kmeans* 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 *Kmeans* e do Guloso, respectivamente. Com a utilização do algoritmo *Kmeans* 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%.

Já nos testes com o programa IS, nos testes realizados com 32 processos, menores latências foram alcançadas com o uso do BRD: 13,30% e 42,49% menor em relação ao *Kmeans* e ao Guloso, respectivamente. No cenário dos testes com 64 processos, 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, o uso do BRD e do *Kmeans* apresentaram resultados muito próximos, mas com o Guloso a latência foi 19,31% maior que com o BRD (que apresentou melhor resultado). Por fim, nos experimentos com 256 processos, a menor latência foi adquirida com o uso do BRD, que obteve um ganho de 6,78% e 35,78% em relação ao *Kmeans* e ao Guloso, respectivamente.

Com a aplicação MG, nos testes realizados com 32 processos, a menor latência foi alcançada com o uso do BRD: 13,46% e 20,60% menor do que com o uso do *Kmeans* e o Guloso, respectivamente. O cenário para 64 processos é um pouco diferente: o *Kmeans* mostrou ser a melhor opção,obtendo uma diferença de 6,62% do BRD e 11,21% do

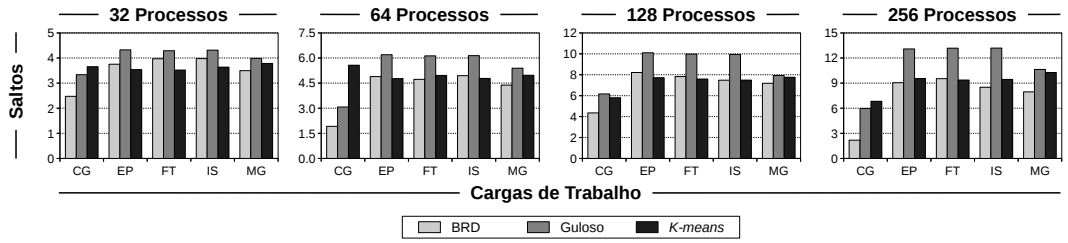

Figura 6. Resultados: distância média.

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 *Kmeans* 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 *Kmeans* e 22,56% melhor que o Guloso.

5.3. Distância Média

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 seu destino final. Os resultados apresentados nos gráficos da Figura 6 reforçam a relação entre essas duas métricas. Nos testes com a aplicação CG, com 32 processos, o Guloso e o *Kmeans* obtiveram resultados inferiores ao BRD: 34,92% e 47,83%, respectivamente. No cenário com 64 processos: o BRD atingiu uma distância menor que os demais: com o Guloso a distância foi 60,38% maior e com o *Kmeans* foi 190,92% maior. Nos resultados com 128 processos, o uso do *Kmeans* 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 *Kmeans*.

Já nos testes com a aplicação EP, para 32 processos, a menor distância foi obtida com a utilização do *Kmeans*, 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 *Kmeans* alcançou a menor distância, cerca de 2,48% menor que o BRD e 29,99% menor que o Guloso. A utilização do *Kmeans* 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 *Kmeans* e ao Guloso, nessa ordem.

Nos experimentos com o programa FT, nos testes com 32 processos, os resultados foram melhores com a utilização do *Kmeans*, 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 *Kmeans* e do Guloso, respectivamente. Nos testes com 128 processos, o cenário é semelhante ao de 32: com a utilizaação do *Kmeans* 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 *Kmeans* 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.

Nos testes realizados com a aplicação IS, com 32 processos, a menor distância foi alcançada com o uso do *Kmeans*, 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 *Kmeans*, 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 *Kmeans* não é significativa, sendo menor que 1%, já com relação ao uso do Guloso houve uma diferença considerável 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 *Kmeans* e Guloso, nessa ordem.

Nos resultados dos experimentos com o programa MG, para o 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 *Kmeans* 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 *Kmeans* 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 *Kmeans* e do Guloso, nessa ordem. Por fim, nos testes com 256 processos, com o uso do BRD obteve-se diferenças significativas: 28,98% e 33,64% em relação ao uso do *Kmeans* e do Guloso, na devida ordem.

De modo geral, pode-se observar que cargas que possuem um padrão de comunicação homogêneo apresentam melhores resultados com o mapeamento gerado pelo *Kmeans*, como podemos observar 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 *Kmeans* 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 do mapeamento a maioria das comunicações já ocorriam entre nós próximos. E após o mapeamento, considerando as outras variáveis, 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. Foi a estratégia que menos se mostrou eficiente em relação às outras.

6. Conclusão

Como foi dito, a tarefa de mapear processos pertence à classe NP-difícil. Portanto, não se trata de uma tarefa trivial e nem mesmo que possui uma solução ótima definida. Por isso, podemos reforçar a ideia de que o conhecimento das características das aplicações é de grande ajuda para analisar e definir uma estratégia de mapeamento mais adequada.

O algoritmo heurístico proposto, o *Kmeans*, apresentou resultados satisfatórios

quando trata-se de uma carga de padrões de comunicações homogêneos. Para padrões heterogêneos, o mapeamento feito pelo BRD se mostrou mais eficiente. No entanto, isso não inviabiliza o uso do *Kmeans*, pois, por diversas vezes, a diferença entre o desempenho dos dois algoritmos foi pequena. Além disso, o a heurística de mapeamento baseada no *Kmeans* pode ser ainda mais explorada, estudando-se qual a quantidade de *clusters* ideal para se usar, por exemplo.

Como trabalhos futuros, a integração do simulador Topaz com o GEM5 é cogitada para termos uma liberdade maior quando se trata do tamanho dos pacotes. Outro ponto a ser mais estudado é o aprimoramento do *Kmeans* para que alcance resultados melhores para outros padrões de comunicação e a análise de consumo de energia da rede.

7. Agradecimentos

À CAPES, FAPEMIG e CNPq pelo suporte parcial no projeto.

Referências

[Abad et al. 2012] Abad, P., P.Prieto, L.Menezo, A.Colaso, V.Puente, and Gregorio, J. (2012). Topaz: An open-source interconnection network simulator for chip multiprocessors and supercomputers. *NOCS*.

[Ascia et al. 2004] Ascia, G., Catania, V., and Palesi, M. (2004). Multi-objective mapping for mesh-based noc architectures. pages 182–187.

[Avelar et al. 2011] Avelar, C., Oliveira, P., Freitas, H., and Navaux, P. (2011). Evaluating the problem of process mapping on network-on-chip for parallel applications. In *Workshop on Architecture and Multi-Core Applications (WAMCA)*, pages 18–23, Vitória, Brazil.

[Bertsekas 1988] Bertsekas, D. (1988). The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14(1):105–123.

[Binkert 2011] Binkert, N. (2011). The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39.

[Bokhari 1981] Bokhari, P. H. (1981). On the mapping problem. *IEEE Trans. Comput.*, pages 207–214.

[Carvalho and N. Moraes 2007] Carvalho, E. C. and N. Moraes, F. (2007). Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs. *Rapid System Prototyping, 2007. RSP 2007. 18th IEEE/IFIP International Workshop on*, pages 34–40.

[de Oliveira 2012] de Oliveira, P. A. C. (2012). Avaliação de desempenho e caracterizacão de cargas de trabalho paralelas para redes-em-chip sem fio. *Dissertacão do Programa de Pos-Graduacão em Informática. Pontifícia Universidade Católica de Minas Gerais*.

[Freitas et al. 2009] Freitas, H. C., Alves, M. A. Z., and Navaux, P. O. A. (2009). Noc e nuca: Conceitos e tendências para arquiteturas de processadores many core. *Minicurso do Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD-SSC)*.

[Oliveira et al. 2011] Oliveira, P., Avelar, C., Guimaraes, S., and Freitas, H. (2011). A greedy heuristic for process mapping on networks-on-chip. In *Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD-SSC)*, pages 1–8, Vitória, Brazil.

[Pellegrini 2008] Pellegrini, F. (2008). Scotch 5.1: User's guide. *Technical report, LaBRI*.