A Greedy Heuristic for Process Mapping on Networks-on-Chip

Poliana A. C. Oliveira, Cíntia P. Avelar, Silvio Jamil F. Guimarães, Henrique C. Freitas
Departamento de Ciência da Computação
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte – MG – Brasil
poliana.correa@sga.pucminas.br, cintiaavelar@ieee.org, {sjamil, cota}@pucminas.br

Abstract

The state-of-the-art related to many-core processors focuses on Networks-on-Chip (NoCs) as approach to provide on-chip message-passing communication. The main problem is the number of hops from source to destination increasing the latency to exchange data based on network packets. Our goal is to propose a greedy heuristic for process mapping on NoCs clustering processes that have more communication on nearest cores. The evaluation method is based on simulation of parallel workloads on a NoC. The greedy heuristic achieves a higher throughput (up to 23.04%) and lower energy consumption (up to 9.87%) than a direct mapping approach for most workloads.

1. Introdução

Em processadores *manycore* vários núcleos são integrados dentro de um único *chip*. Neste caso, o ganho de desempenho não está somente na exploração da frequência de operação, mas na exploração do paralelismo das aplicações. O processamento simultâneo proporcionado pela divisão de uma aplicação em processos/threads e a atribuição de cada um deles a um núcleo aumenta a vazão de resultados, e consequentemente, reduz o tempo de execução da aplicação.

Como a comunicação entre os processos se dá por passagem de mensagens (pacotes de rede), o modelo de interconexão entre os núcleos pode influenciar o desempenho. Uma solução apontada pelo estado da arte e que tem se mantido por ser escalável, flexível e de alto desempenho são as Redes-em-Chip, onde pequenos roteadores são responsáveis por enviar os pacotes para os núcleos mais próximos [2, 3].

Um dos problemas desse modelo é o número de saltos entre os núcleos necessários para que um pacote alcance o núcleo destino. A quantidade de saltos afeta tanto a utilização dos recursos da rede quanto o consumo de energia. Quando o número de saltos aumenta, o tráfego entre os *links* (enlaces) também cresce. Devido ao compartilhamento de recursos, pode ocorrer perdas, retransmissões e atrasos na entrega dos pacotes o que afeta a vazão da rede como um todo.

Este artigo propõe uma estratégia gulosa de mapeamento de processos em Redes-em-Chip que se baseia no número de comunicações entre os processos a fim de que os processos que mais se comunicam sejam alocados em núcleos mais próximos em termos do número de saltos para alcançá-los. A hipótese é que essa abordagem melhore o desempenho da arquitetura além de reduzir o consumo de energia.

Portanto, a contribuição deste artigo está na avaliação do mapeamento de processos usando uma heurística gulosa para Redes-em-Chip. A avaliação é baseada em modelo de simulação e são usados traços de cargas de trabalho paralelas para comparação com uma abordagem de mapeamento direto por linha. Como resultado espera-se poder identificar características nas cargas de trabalho que indiquem pontos de melhoria nos algoritmos de mapeamento.

O artigo está organizado da seguinte forma: Seção 2 apresenta uma visão geral das Redes-em-Chip e do problema de mapeamento de processos. Seção 3 ressalta alguns trabalhos relacionados. Seção 4 destaca a heurística gulosa proposta. Seção 5 mostra a metodologia de avaliação e os resultados obtidos pela heurística gulosa. Seção 6 apresenta as conclusões e trabalhos futuros.

2. Redes-em-Chip

O modelo de interconexão em *chip* para processadores *manycore* denominado Redes-em-Chip, ou simplesmente NoC (do inglês, *Networks-on-Chip*), trata-se de uma abordagem baseada em uma rede adaptada com

pequenos roteadores que viabilizam a troca de mensagens entre os núcleos [2].

Os principais componentes de uma NoC são apresentados na Figura 1. O roteador é responsável por encaminhar um pacote para os roteadores vizinhos e, assim sucessivamente, até que ele chegue ao destino. Uma interface de rede é usada entre o núcleo e o roteador. Os enlaces/*links* são necessários para interconectar um roteador a outro.

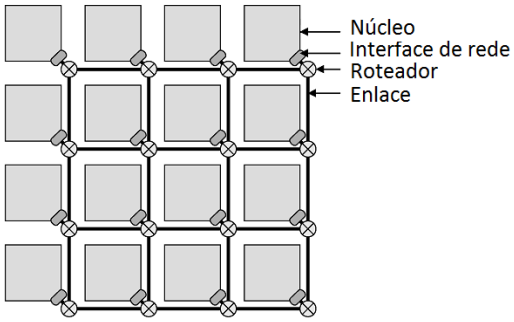

Figura 1. Componentes da NoC. Fonte: [3]

Assim como as redes de computadores, as Redes-em-Chip são caracterizadas pela topologia e pelo protocolo de roteamento [10, 16]. A topologia é definida pela forma como os roteadores são interligados na rede, tais como *Mesh*, *Torus*, *Binary tree* e *Hypercube*. Já o caminho que um pacote segue para circular pela rede é determinado pelo protocolo de roteamento (e.g. XY). Técnicas de encaminhamento de pacotes, tais como *Wormhole* e *Store-and-Forward*, podem determinar a eficiência da NoC.

2.1. Mapeamento de Processos em Redes-em-Chip

Nas Redes-em-Chip a troca de pacotes entre os núcleos é sustentada pelos roteadores que, de acordo com o protocolo de roteamento, redirecionam os pacotes para outros roteadores, e assim sucessivamente. O problema dessa abordagem são os múltiplos saltos entre roteadores até alcançar o núcleo destino que podem elevar o custo de rede para realizar a comunicação.

O mapeamento é uma técnica que tem por intuito atribuir um conjunto de processos a um conjunto de unidades de processamento [9]. O problema de mapeamento de processos consiste em realizar essa atribuição segundo algum critério (número de comunicações entre processos, disponibilidade de recursos, tempo de execução do processo, dentre outros) a fim de melhorar o desempenho de execução da aplicação.

Uma forma simples de mapeamento é a atribuição direta de processos por linha na Rede-em-Chip. Dessa forma, durante a atribuição nenhum dos critérios mencionados é levado em conta. A Figura 2 apresenta um exemplo de mapeamento direto por linha de 8 processos em uma Rede-em-Chip com topologia 2D *Mesh*.

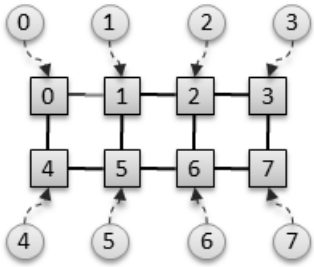

Figura 2. Mapeamento direto por linha de 8 processos em uma Mesh 2 x 4

O mapeamento de processos pode ser estático ou dinâmico. No primeiro, as características das cargas de trabalho são previamente conhecidas e, com base nesse conhecimento, os processos são atribuídos aos núcleos antes que a carga entre em operação. Já no mapeamento dinâmico, o sistema identifica que muitas comunicações estão sendo feitas entre dois processos e pode decidir remapeá-los durante a execução da carga de trabalho com a finalidade de minimizar os custos de rede.

O problema de mapeamento de processos é considerado NP-difícil, pois pode ser reduzido a um conhecido problema NP-difícil denominado problema de isomorfismo em grafos [4]. Embora exista um algoritmo com tempo polinomial para casos específicos de isomorfismo em grafos [13], o problema ainda não foi resolvido para casos genéricos. NP-difícil é uma classe de problemas que não podem ser nem resolvidos e nem verificados em tempo polinomial [12].

Como a literatura não aponta um algoritmo exato com tempo de execução viável computacionalmente para o problema de mapeamento de processos, algumas heurísticas tem sido propostas como tentativa de chegar a soluções aproximadas em tempo polinomial. A próxima seção apresenta alguns trabalhos relacionados bem como as estratégias adotadas por cada um deles.

3. Trabalhos Relacionados

Os trabalhos correlatos mostram diversas técnicas de mapeamento de processos em Redes-em-Chip e ressaltam a importância delas para melhorar o desempe-

nho de execução da aplicação. Eles apresentam diferentes formas de reduzir o tráfego de pacotes na rede e, consequentemente, aumentar o *throughput* e diminuir o consumo de energia.

O impacto da contenção de rede no mapeamento é apresentado por [7]. Uma formulação de programação linear com base no mapeamento de contenção foi usada a fim de minimizar a disputa *inter-tile*. Diferentes tamanhos de Redes-em-Chip (de 3x3 para 5x5 topologias de malha) foram verificadas. As cargas de trabalho foram baseadas em aplicações reais e sintéticas, como o decodificador MPEG-4, e os resultados mostraram uma melhoria na taxa de transferência para diversos casos.

Em [5], os autores apresentam cinco heurísticas para o mapeamento dinâmico de tarefas. Aplicações sintéticas foram modeladas pela teoria de grafos usando SystemC para realizar o experimento em uma Rede-em-Chip de topologia *Mesh* 8x8. A estratégia é baseada em um processador *Manager* e a Rede-em-Chip é dividida em: tarefas de *hardware* e tarefas de *software*. O principal resultado mostra uma redução da ocupação dos *links* internos da Rede-em-Chip.

Uma proposta de mapeamento de tarefas baseado em conhecimento de comunicação foi proposta em [18]. A abordagem é focada na prioridade de acesso à memória compartilhada quando a tarefa tem uma grande quantidade de comunicação. As cargas de trabalho são H.264, Motion-JPEG, decodificadores de MP3 e transformações 2D. O algoritmo proposto reduziu o número de ciclos de *clock*, o consumo de energia e os custos de comunicação para transferência de dados.

O trabalho de [22] apresenta um mapeamento topológico, também baseado em comunicação consciente. A estratégia é correlacionar o modelo de rede com o desempenho real da rede. Os resultados mostraram melhoria no desempenho em termos de latência e consumo de energia em comparação com outras técnicas.

Outra abordagem de mapeamento em Redes-em-Chip pode ser vista em [6], onde é proposto um compilador para mapeamento 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 em que já se espera uma frequente comunicação uns com os outros sejam atribuídos ao processador que esteja o mais próximo possível. Na terceira etapa, os elementos de dados são mapeados para memórias associadas aos nós do CMP. O principal objetivo desse mapeamento é colocar um determinado item de dados em um nó que esteja perto dos 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 e o autor lamenta pelo tema não ser muito explorado, pois os resultados são óbvios.

O mapeamento do espaço em Redes-em-Chip pode ser explorado com a aplicação de uma função multi-objetivo. Em [1], a abordagem é uma forma eficiente para obter os mapeamentos que otimizam o desempenho e o consumo de energia, onde foi utilizado um simulador orientado a eventos de rastreamento. Este torna possível analisar efeitos dinâmicos importantes que representam grande impacto sobre o mapeamento. A avaliação foi realizada com tráfego sintetizado e aplicações reais (um MPEG-2 Sistema Codificador/Decodificador) e confirma a precisão e escalabilidade da abordagem.

De acordo com [19], o problema de mapeamento de processos para arquiteturas com n núcleos consiste em encontrar uma associação de cada núcleo de uma rede de tal forma que o custo seja minimizado. O modelo CDCM (do inglês, *Communication Dependence and Computation Model*) foi usado para capturar o tempo e o volume de comunicação de uma aplicação. Os resultados foram comparados com um modelo proposto em [14] 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 da aplicação em 40% e o consumo de energia em 20%.

O diferencial deste trabalho está na aplicação de uma heurística gulosa para mapear processos em Redes-em-Chip e no uso de cargas paralelas baseadas em comunicações coletivas para avaliar o desempenho do mapeamento proposto. A estratégia consiste em aproximar os processos que mais se comunicam levando em consideração as restrições impostas pela topologia e pelo protocolo de roteamento da arquitetura a fim de reduzir o tráfego da rede e o consumo de energia.

4. Proposta de Heurística Gulosa para Mapeamento de Processos

O mapeamento de processos proposto por este artigo é voltado para uma Rede-em-Chip específica: rede

com topologia 2D *Mesh* e protocolo de roteamento XY, conforme a ilustrada pela Figura 1. A seleção foi feita com base na popularidade dessa Rede-em-Chip tanto na literatura [3] quanto em implementações pela indústria [15].

4.1. Modelagem do Problema pela Teoria de Grafos

O problema de mapeamento de processos pode ser modelado pela teoria de grafos. Assim, considerando o prévio conhecimento do comportamento da carga de trabalho através da leitura de arquivos de traços de execução das mesmas é possível expressar as comunicações entre os processos através de um grafo completo, não-direcionado e valorado $G_p = (V_p, E_p)$, onde os vértices representam os processos, as arestas a comunicação entre dois processos e cada aresta possui um valor $W_p \geq 0$ que representa o número de comunicações efetuadas entre dois processos.

A topologia da Rede-em-Chip pode ser modelada em um grafo não-direcionado e não-valorado $G_n = (V_n, E_n)$, onde os núcleos de processamento são representados pelos vértices e os *links* que interligam os roteadores de dois núcleos pelas arestas. O problema consiste em encontrar uma função de mapeamento $f(v) = \{v_i \rightarrow v_j \mid v_i \in V_p,\ v_j \in V_n\}$ tal que o custo de rede seja minimizado e satisfaça a restrição $p = n$, onde p é o número de processos e n o número de núcleos de processamento.

4.2. Modelagem do Custo de Comunicação na Rede-em-Chip

O custo de comunicação em uma Rede-em-Chip pode ser medido em termos do número de saltos necessários para que um pacote alcance o seu destino. Quanto mais saltos requer a comunicação, maior será a latência da mesma devido à necessidade de retransmissões do pacote, compartilhamento de recursos e até mesmo colisões com outros pacotes.

Em uma Rede-em-Chip o número de saltos entre dois núcleos é definido com base na topologia e no algoritmo de roteamento. Considerando a topologia 2D *Mesh* e o protocolo de roteamento XY, temos uma malha de núcleos, cada um associado a um roteador, e esses interligados por pequenos *links*. O caminho que o pacote deve percorrer é determinado por saltos no eixo X seguidos por saltos no eixo Y.

A Figura 3 utiliza uma 2D *Mesh* 2 x 4 (8 núcleos dispostos em 2 linhas e 4 colunas) para exemplificar o funcionamento do algoritmo de roteamento XY. Observe que o custo de comunicação entre o núcleo 0 e o núcleo 7 é de 4 saltos.

Assim, dado o grafo $G_n = (V_n, E_n)$ que expressa a organização dos núcleos na *Mesh* como entrada e com base no cálculo de saltos entre dois núcleos para o protocolo XY foi gerado um grafo $G_w = (V_w, E_w)$ completo, não-direcionado e valorado que expressa os custos de comunicação nessa topologia, onde os vértices representam os núcleos, as arestas a comunicação entre dois núcleos e cada aresta está associada a um valor $W_w \geq 0$ que indica o número de saltos entre dois núcleos.

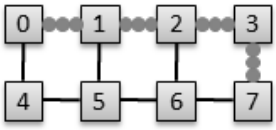

Figura 3. Exemplo de roteamento XY estabelecido entre os núcleos 0 e 7 de uma Mesh 2 x 4

4.3. Heurística Gulosa

A estratégia gulosa tem por finalidade buscar a melhor solução local para que ela possa ser usada para atingir uma boa solução global. Assim, um algoritmo é guloso se, a cada passo apenas um subconjunto de soluções viáveis é analisado e, dado algum critério de escolha, a melhor solução para esse subconjunto é selecionada sem levar em consideração escolhas anteriores evitando que o espaço de soluções seja enumerado [17].

No que se refere a otimalidade da solução global, essa dependerá da estrutura das subsoluções encontradas [8]. Embora em alguns casos a solução ótima global seja alcançada, nem sempre é possível garantir que o melhor resultado local será o melhor resultado global, como é o caso da heurística proposta por este artigo.

A heurística de mapeamento de processos proposta aposta na abordagem gulosa para solucionar de forma aproximada e em tempo polinomial esse problema. Considerando a modelagem pela teoria de grafos, o algoritmo recebe como entrada os grafos $G_p = (V_p, E_p)$, $G_n = (V_n, E_n)$ e $G_w = (V_w, E_w)$ definidos na Seção 4.1 e 4.2 respectivamente, e previamente armazenados em uma matriz de adjacência com complexidade de tempo computacional na ordem de $O(k) + O(n^2)$, onde k é o número total de comunicações efetuadas pela carga de trabalho e n é o número de núcleos da *Mesh*.

A estratégia de mapeamento foi dividida em duas fases:

1 Selecionar o vértice $v_i \mid v_i \in G_p = (V_p, E_p)$ com maior valor de $\sum_{i=1}^{p} W_i$ das arestas que incidem

em v_i. Selecionar o vértice $v_j \mid v_j \in G_n = (V_n, E_n)$ com maior valor de grau. Mapear $v_i \rightarrow v_j$.

2 Enquanto houver um vértice $v_i \mid v_i \in G_p = (V_p, E_p)$ que não esteja mapeado para um $v_j \mid v_j \in G_n = (V_n, E_n)$:

Dado o último vértice v_i mapeado, selecionar um vértice $v_k \mid v_k \in G_p = (V_p, E_p)$ que ainda não foi mapeado e que possui a aresta de maior W_p incidente em v_i. Dado último vértice v_j mapeado, selecionar um vértice $v_l \mid v_l \in G_n = (V_n, E_n)$ tal que em $G_w = (V_w, E_w)$ esse vértice possua a aresta de menor W_w incidindo em v_j. Mapear $v_k \rightarrow v_l$.

Em outras palavras, a ideia principal consiste em inicialmente atribuir o processo que realiza mais comunicações, no núcleo que possui maior número de *links*. Em seguida, enquanto houver um processo sem mapeamento: i) selecionar o processo vizinho que ainda não foi mapeado e que possui maior número de comunicações com o último mapeado, ii) selecionar o núcleo não mapeado que se encontra mais próximo em termos de número de saltos do último mapeado, iii) atribuir o processo selecionado para esse núcleo. Em caso de empate, o processo/núcleo com menor índice é selecionado. O Algoritmo 1 formaliza a heurística descrita.

Algoritmo 1: Heurística gulosa

```
1: Entrada: Gp = (Vp, Ep), Gn = (Vn, En) e
   Gw = (Vw, Ew)
2: Saída: Gp → Gn
3: INICIO
4: verticeMaiorPeso = getVerticeMaiorPeso(Gp)
5: verticeMaiorGrau = getVerticeMaiorGrau(Gn)
6: Gp(verticeMaiorPeso) → Gn(verticeMaiorGrau)
7: verticesGpVisitado[verticeMaiorPeso] = true
8: verticesGnVisitado[verticeMaiorGrau] = true
9: ultimoVerticeGpAnalisado = verticeMaiorPeso
10: ultimoVerticeGnAlocado = verticeMaiorGrau
11: while (!todosAlocados()) do
12:     verticeGp = getVizinhoArestaMaiorPeso(Gp,
        verticesGpVisitado, ultimoVerticeGpAnalisado)
13:     verticeGn = getVizinhoMaisProximo(Gw,
        verticesGnVisitado, ultimoVerticeGnAlocado)
14:     Gp(verticeGp) → Gn(verticeGn)
15:     verticesGnVisitado[verticeGn] = true
16:     verticesGpVisitado[verticeGp] = true
17:     ultimoVerticeGpAnalisado = verticeGp
18:     ultimoVerticeGnAlocado = verticeGn
19: end while
20: Retorna Gp → Gn
21: FIM
```

A abordagem gulosa está na seleção a cada passo do processo vizinho que efetua o maior número de comunicações com o último mapeado a fim de atribuí-lo ao núcleo mais próximo possível dele na *Mesh*. Tal critério considera que essa é a melhor solução local já que minimiza a latência de comunicação imposta pela topologia da rede entre os processos que mais se comunicam. Todavia, não é possível garantir que ela é a melhor solução global.

A ordem de complexidade de tempo computacional da heurística gulosa é quadrática $O(p^2)$ no pior caso, onde p é o número de processos. Pois, a cada passo são feitas p operações para selecionar o processo e n operações para selecionar o núcleo, dado que $p = n$, então temos $2 * p$ operações a cada passo. Como a atribuição processo $\rightarrow$ núcleo é realizada p vezes, então para efetuar o mapeamento são realizadas $2 * p^2$ operações.

5. Resultados

5.1. Metodologia de avaliação

Para avaliar o desempenho da Rede-em-Chip mapeada pela heurística gulosa foram selecionadas cargas de trabalho do NPB (*NAS Parallel Benchmark*) [21] devido a natureza paralela das mesmas [11]. O NPB é formado por programas paralelos representativos construídos para computação de métodos numéricos de simulações aerodinâmicas, tendo em vista, a avaliação de desempenho de supercomputadores paralelos.

Os programas selecionados foram BT (*Block Tridiagonal*), CG (*Conjugate Gradient*), EP (*Embarassingly Parallel*), FT (*Fast Fourier Transform*), IS (*Integer Sort*), LU (*Lower And Upper Triangular System*), MG (*Multigrid*) e SP (*Scalar Pentadiagonal*). Cada um deles é destinado a solução de um problema específico da aerodinâmica e por esse motivo possuem características distintas, como por exemplo, o tamanho do programa e o número de comunicações entre processos.

A implementação dos programas foi baseada em passagem de mensagens e eles foram configurados com 8 processos e executados em um *cluster* instrumentado para coletar em um arquivo os traços de cada execução, tais como dados sobre o envio de cada pacote (instante de envio, endereço de origem, endereço de destino, tamanho, dentre outros).

O modelo de avaliação escolhido foi a simulação. Para tanto, o simulador Noxim [20] foi utilizado. Trata-se de um conhecido simulador de Rede-em-Chip, simples, com interface de texto e de fácil configuração.

O tráfego de entrada para a simulação no Noxim foi um arquivo gerado com base nos traços de execução das cargas de trabalho. Este arquivo possui três informações relacionadas ao pacote: endereço de origem, endereço de destino e instante de envio. Algumas restrições foram necessárias. A primeira se deve ao instante de envio dos pacotes registrado nos arquivos de traços com valores decimais e a limitação do Noxim em

suportar somente valores inteiros, já que a sua unidade de trabalho é expressa em ciclos.

Para resolver esse problema, os arquivos de traços foram analisados considerando que o primeiro pacote é sempre enviado no instante de 1000 ciclos devido ao tempo de aquecimento (*warmup*) padrão do simulador. Para os demais pacotes, se o intervalo de tempo entre eles, em valor inteiro, for menor que 42 então o intervalo obtido é acumulado ao valor anterior para definir o ciclo que o pacote será enviado. Senão, o valor 42 é fixado para evitar que a rede fique ociosa, sendo que 10 ciclos são referentes ao atraso do primeiro flit do pacote e 32 ciclos para o término do pacote.

Estes valores foram definidos com base nas características da rede e do tamanho dos pacotes, permitindo que a simulação não ocorra sem comunicação. Ao final da análise temos o tempo de duração da simulação, dado em ciclos, que varia com a carga de trabalho.

O Noxim também não está preparado para trafegar pacotes de tamanhos diferentes. Por essa razão, o tamanho dos pacotes foi fixado em 16 flits. Além disso, como os arquivos de traços foram gerados com base na execução das cargas com 8 processos, a Rede-em-Chip foi configurada com 8 núcleos de processamento para manter a restrição de mapeamento. A Tabela 1 apresenta todos os parâmetros de configuração da simulação.

Tabela 1. Parâmetros de entrada do Noxim

Parâmetro	Valor
dimx	4
dimy	2
buffer	3 flits
size Nmin	16 flits
size Nmax	16 flits
routing	xy
traffic	table nome-arquivo
warmup	1000 ciclos (padrão)
sim	Depende da carga de trabalho

O desempenho será avaliado com base em algumas das métricas fornecidas pelo Noxim após a simulação. São elas:

- Total de pacotes recebidos
- *Throughput* global médio (Flits/Ciclo)
- *Delay* médio global (Ciclos)
- Energia total consumida (Joules)

5.2. Avaliação de desempenho

A avaliação de desempenho concentrou-se na comparação do mapeamento efetuado pela heurística gulosa em relação ao mapeamento direto por linha. A hipótese é que o primeiro apresente desempenho melhor já que usa como critério de mapeamento a proximidade na Rede-em-Chip entre os processos que realizam muitas comunicações durante a execução da aplicação.

As Figuras 4 e 5 mostram o percentual de aumento do número de pacotes recebidos e *throughput* global médio da heurística gulosa em relação ao mapeamento direto por linha para cada carga de trabalho. Na maioria delas houve melhorias nestes dois aspectos chegando a 14% de aumento no número de pacotes recebidos pela carga CG e 23% no *throughput* da carga EP.

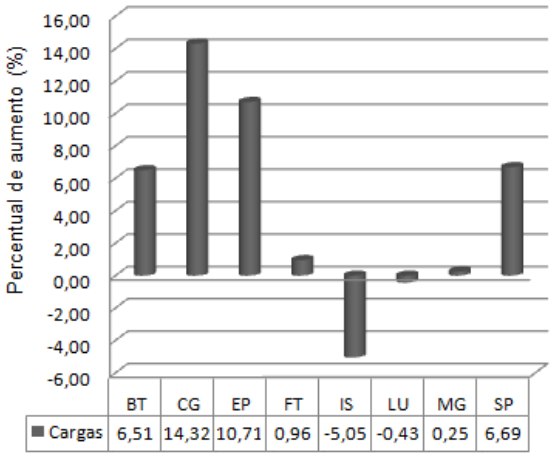

Figura 4. Total de pacotes recebidos

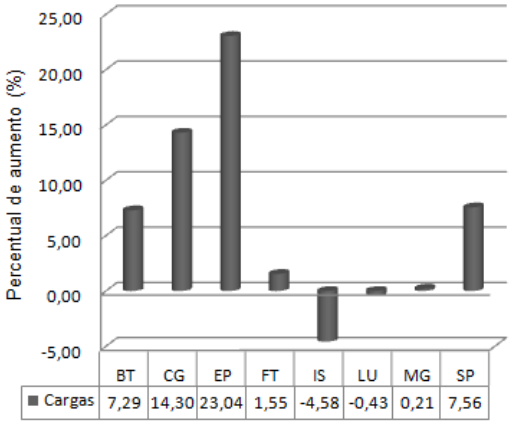

Figura 5. Throughput global médio (Flits/Ciclo)

A variação dos valores percentuais entre uma carga e outra se deve à características comportamentais diferentes entre as aplicações que favorecem ou não a técnica de mapeamento adotada. Em algumas cargas,

a maior parte das trocas de mensagens se concentra entre poucos processos, já em outras todos os processos trocam mensagens na mesma proporção.

No caso da carga IS, a heurística gulosa não se mostrou eficiente reduzindo tanto o número de pacotes recebidos quanto *throughput* da aplicação. Isto pode ter sido ocasionado pela maior uniformidade de comunicações entre os processos que essa aplicação apresenta. Como o critério em análise (número de comunicações) é praticamente o mesmo entre todos os processos, a estratégia de mapeamento gulosa não se apresentou favorável para ganhar desempenho durante a execução dessa carga.

Com relação ao *delay* médio global somente a carga BT não apresentou uma redução, ao comparar a abordagem de mapeamento gulosa com o mapeamento direto por linha, embora este aumento seja apenas de 0,1%, conforme mostra a Figura 6. Isso pode ser explicado pela intensa comunicação entre os processos dessa aplicação que elevam o tráfego na rede provocando um aumento médio de atraso na entrega dos pacotes devido à colisões, perdas e retransmissões.

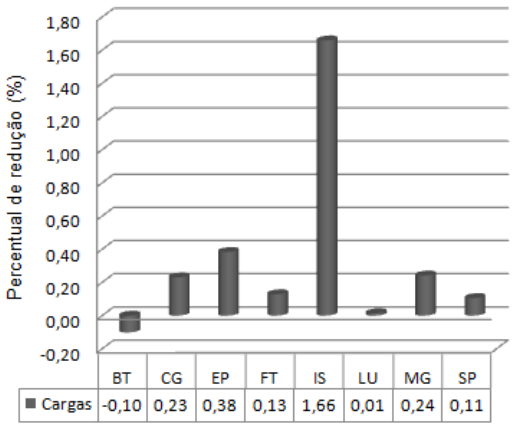

Figura 6. Delay médio global (Ciclos)

Comparando as duas estratégias de mapeamento no quesito consumo de energia, as cargas FT, MG e LU não apresentaram redução quando a heurística gulosa foi aplicada. O valor percentual de aumento de consumo nestas cargas não ultrapassa 3%, como apresentado pela Figura 7. Em contrapartida, a carga CG alcançou economia de 9% no consumo de energia. Essa diferença provavelmente está relacionada com a estratégia de atribuição de processos que para algumas cargas conseguiu reduzir mais o número de saltos entre dois processos comunicantes do que em outras.

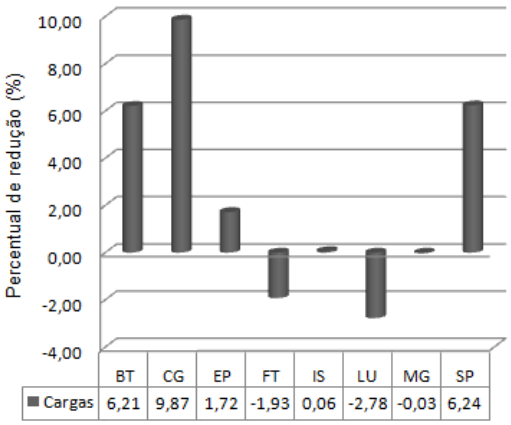

Figura 7. Energia total (J)

6. Conclusões

A heurística gulosa proposta para mapeamento de processos utiliza o número de comunicações entre processos como critério para efetuar a atribuição na Rede-em-Chip. A ideia consiste em alocar, o mais próximo possível, os processos que mais se comunicam a fim de minimizar o número de saltos dados pelo pacote até alcançar o seu destino.

Os resultados mostraram que o mapeamento proposto é melhor do que o mapeamento direto por linha para a maioria das cargas de trabalho avaliadas. As duas estratégias de mapeamento foram comparadas através de simulações de uma Rede-em-Chip com topologia 2D *Mesh* e protocolo de roteamento XY operando com tráfego de cargas de trabalho paralelas do NPB.

No que se refere ao número de pacotes recebidos e ao *throughput* houve um aumento no mapeamento por heurística gulosa com relação ao mapeamento direto por linha para a maior parte das cargas de trabalho. O maior deles foi de 14% para o número de pacotes recebidos e de 23% para o *throughput*.

Com relação ao *delay* e ao consumo de energia, a maior diversificação dos resultados indica que o mapeamento de processos em uma Rede-em-Chip está intimamente relacionado com a carga de trabalho que nela opera. As características das cargas de trabalho podem favorecer ou não a estratégia de mapeamento adotada. Nos casos em que o comportamento da carga de trabalho atuou a favor do mapeamento por heurística gulosa a redução de energia foi de até 9%.

Constatou-se que quando a carga de trabalho é caracterizada por intensa comunicação somente entre alguns processos, a heurística gulosa é beneficiada permitindo a atribuição deles em núcleos que estejam o mais próximo possível. Todavia, quando os proces-

sos da carga de trabalho apresentam uma comunicação uniforme entre eles, o critério de escolha de atribuição (número de comunicações) não é tão relevante prejudicando essa estratégia de mapeamento.

Trabalhos futuros incluem outros experimentos no sentido de verificar se a heurística gulosa mantém o ganho de desempenho quando o número de processos e de núcleos da Rede-em-Chip aumentam. Outro trabalho está na comparação da heurística proposta com alguns algoritmos apresentados no estado da arte e indicação das topologias mais adequadas para essa abordagem de mapeamento.

7. Agradecimentos

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e ao Fundo de Incentivo à Pesquisa da PUC Minas (FIP/PUC Minas).

Referências

[1] G. Ascia, V. Catania, and M. Palesi. Multi-objective mapping for mesh-based noc architectures. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '04, pages 182–187, New York, NY, USA, 2004. ACM.

[2] L. Benini and G. D. Micheli. Networks on chips: A new soc paradigm. *Computer*, 35:70–78, January 2002.

[3] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38:1–51, June 2006.

[4] S. H. Bokhari. On the mapping problem. *IEEE Trans. Comput.*, 30:207–214, March 1981.

[5] E. Carvalho, N. Calazans, and F. Moraes. Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs. In *Proceedings of the 18th IEEE/IFIP International Workshop on Rapid System Prototyping*, pages 34–40, Washington, DC, USA, 2007. IEEE Computer Society.

[6] G. Chen, F. Li, S. W. Son, and M. Kandemir. Application mapping for chip multiprocessors. In *Proceedings of the 45th annual Design Automation Conference*, DAC '08, pages 620–625, New York, NY, USA, 2008. ACM.

[7] C. Chou and R. Marculescu. Contention-aware application mapping for network-on-chip communication architectures. *IEEE International Conference on Computer Design*, 0:164–169, 2008.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, New York, 2001.

[9] E. H. Cruz, M. A. Alves, and P. O. A. Navaux. Process mapping based on memory access traces. *Symposium on Computing Systems*, 0:72–79, 2010.

[10] H. C. Freitas. Arquitetura de noc programável baseada em múltiplos clusters de cores para suporte a padrões de comunicação coletiva. Master's thesis, Universidade Federal do Rio Grande do Sul, 2009.

[11] H. C. Freitas, L. M. Schnorr, M. A. Z. Alves, and P. O. A. Navaux. Impact of parallel workloads on noc architecture design. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, PDP '10, pages 551–555, Washington, DC, USA, 2010. IEEE Computer Society.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[13] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, STOC '74, pages 172–184, New York, NY, USA, 1974. ACM.

[14] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, ASP-DAC '03, pages 233–239, New York, NY, USA, 2003. ACM.

[15] Intel. Teraflops research chip. Disponível em http://techresearch.intel.com/Project Details.aspx?Id=151, 2011.

[16] N. A. G. Júnior. Análise e simulação de topologias de redes em chip. Master's thesis, Universidade Estadual de Maringá, 2010.

[17] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[18] S.-H. Lee, Y.-C. Yoon, and S.-Y. Hwang. Communication-aware task assignment algorithm for mpsoc using shared memory. *J. Syst. Archit.*, 56:233–241, July 2010.

[19] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, and F. Hessel. Exploring noc mapping strategies: An energy and timing aware technique. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '05, pages 502–507, Washington, DC, USA, 2005. IEEE Computer Society.

[20] Noxim. Noxim the noc simulator. Disponível em http://noxim.sourceforge.net/, 2011.

[21] NPB. Nas parallel benchmarks. Disponível em http://www.nas.nasa.gov/Resources/Software/npb.html, 2011.

[22] R. Tornero, J. M. Orduña, M. Palesi, and J. Duato. A communication-aware topological mapping technique for nocs. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, Euro-Par '08, pages 910–919, Berlin, Heidelberg, 2008. Springer-Verlag.