Uma Implementação do Framework PSkel com Suporte a Aplicações Estêncil Iterativas para o Processador MPPA-256

Emmanuel Podestá Jr.[1], **Alyson D. Pereira**[1], **Rodrigo C. O. Rocha**[2], **Márcio Castro**[1], **Luís F. W. Góes**[2]

[1] Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD)
Universidade Federal de Santa Catarina (UFSC) – SC, Brasil

[2] Grupo de Computação Criativa e Paralela (CreaPar)
Pontifícia Universidade Católica de Minas Gerais (PUC Minas) – MG, Brasil

emmanuel.podesta@grad.ufsc.br, alyson.pereira@posgrad.ufsc.br,
rcor@pucminas.br, marcio.castro@ufsc.br, lfwgoes@pucminas.br

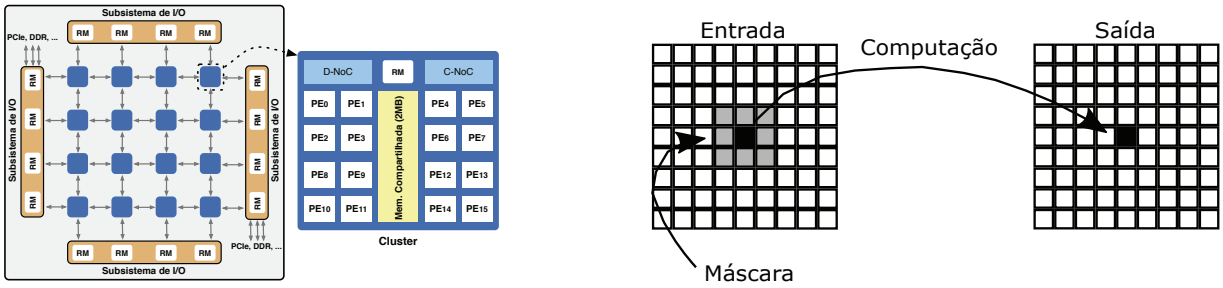
Resumo. *Neste artigo é proposta uma adaptação do framework PSkel para o processador manycore MPPA-256. O framework permite simplificar o desenvolvimento de aplicações estêncil iterativas para o MPPA-256, escondendo do desenvolvedor detalhes de implementação tais como a comunicação e a distribuição de computações entre os núcleos de processamento. Os resultados mostraram um peso significativo da comunicação no desempenho da solução.*

1. Introdução

Diversos padrões de computação paralela são conhecidos na literatura, tais como *map*, *reduce*, *pipeline*, *scan* e *estêncil*. Dentre eles, o padrão estêncil é um dos padrões mais utilizados em aplicações como simulação de física de partículas, previsão meteorológica, termodinâmica, resolução de funções diferenciais, manipulação de imagens, entre outras [Rahman et al. 2011]. O PSkel é um *framework* de programação paralela desenvolvido para simplificar o desenvolvimento de aplicações estêncil [Pereira et al. 2015]. Utilizando uma abstração de alto nível, o programador define o *kernel* da computação estêncil, enquanto o *framework* se encarrega de executar a computação paralela em *multicores* e *Graphics Processing Units* (GPUs) de maneira eficiente.

Recentemente, uma adaptação preliminar do PSkel foi proposta para o processador MPPA-256, um processador *manycore* de baixo consumo de energia [Podestá Jr. et al. 2016]. Devido às suas diversas características peculiares e ao baixo nível de abstração requerido, desenvolver aplicações paralelas para esse tipo de processador é uma tarefa desafiadora. Neste sentido, a adaptação do PSkel para o MPPA-256 permite que detalhes de baixo nível dessa arquitetura possam ser abstraídos, além de permitir que aplicações já existentes em PSkel possam ser portadas para essa nova plataforma sem a necessidade de modificações de código.

O presente trabalho apresenta uma adaptação completa do PSkel para o processador MPPA-256, suprindo assim as limitações da solução proposta em [Podestá Jr. et al. 2016]. Essa nova versão permite: (i) execução de aplicações estêncil iterativas; (ii) flexibilidade no particionamento dos dados; e (iii) otimizações na computação do *kernel*. Além da proposta, são discutidos os resultados de desempenho e consumo de energia obtidos da execução de três aplicações estêncil implementadas no PSkel. Os resultados mostram que a forma de particionamento dos dados afeta o desempenho e que a comunicação ainda é um fator limitante da solução proposta.



(a) MPPA-256. (b) Diagrama do padrão estêncil.

Figura 1. Visão geral do MPPA-256 e ilustração do padrão estêncil.

O restante desse trabalho está organizado da seguinte forma. A Seção 2 apresenta os principais conceitos do processador *manycore* MPPA-256 e do *framework* PSkel. A Seção 3 discute a adaptação do PSkel para oferecer suporte ao MPPA-256. Os resultados são apresentados na Seção 4 e as conclusões são apresentadas na Seção 5.

2. Fundamentação Teórica

2.1. MPPA-256

O processador MPPA-256 é composto por 256 núcleos de processamento de 400 MHz denominados *Processing Element*s (PEs). Como mostrado na Figura 1a, esses núcleos são organizados em 16 *clusters* contendo 16 PEs cada um. Cada *cluster* possui uma memória local de 2 MB (compartilhada entre todos os PEs do *cluster*) e um núcleo de sistema denominado *Resource Manager* (RM). RMs são responsáveis por tarefas de gerência do sistema operacional e comunicação. Além dos *clusters*, o processador apresenta 4 subsistemas de Entrada e Saída (E/S), sendo um deles conectado a uma memória externa *Low Power Double Data Rate 3* (LPDDR3) de 2 GB. *Clusters* e subsistemas de E/S se comunicam por uma *Network-on-Chip* (NoC) *torus* 2D.

Estudos anteriores mostraram que desenvolver aplicações paralelas otimizadas para o MPPA-256 é um grande desafio [Francesquini et al. 2014] devido a alguns fatores importantes tais como: **(i) modelo de programação híbrido**: *threads* em um mesmo *cluster* se comunicam através de uma memória compartilhada local, porém a comunicação entre *clusters* é feita explicitamente via NoC, em um modelo de memória distribuída; **(ii) comunicação**: é necessário a utilização de uma *Application Programming Interface* (API) específica para a comunicação via NoC, similar ao modelo clássico POSIX de baixo nível para *Inter-Process Communication* (IPC); **(iii) memória**: cada *cluster* possui apenas 2 MB de memória local de baixa latência, portanto aplicações reais precisam constantemente realizar comunicações entre o subsistema de E/S (conectado à memória LPDDR3); e **(iv) coerência de *cache***: cada PE possui uma memória *cache* privada sem coerência com as *caches* dos demais PEs, sendo necessário o uso explícito de instruções do tipo *flush* para atualizar a *cache* de um PE em determinados casos.

2.2. PSkel

O PSkel é um *framework* de programação em alto nível para o padrão estêncil, baseado nos conceitos de esqueletos paralelos, que oferece suporte para execução paralela em ambientes heterogêneos incluindo CPU e GPU. Utilizando uma única interface de programação escrita em C++, o usuário é responsável apenas por definir o *kernel* principal da computação estêncil, enquanto o *framework* se encarrega de gerar código executável

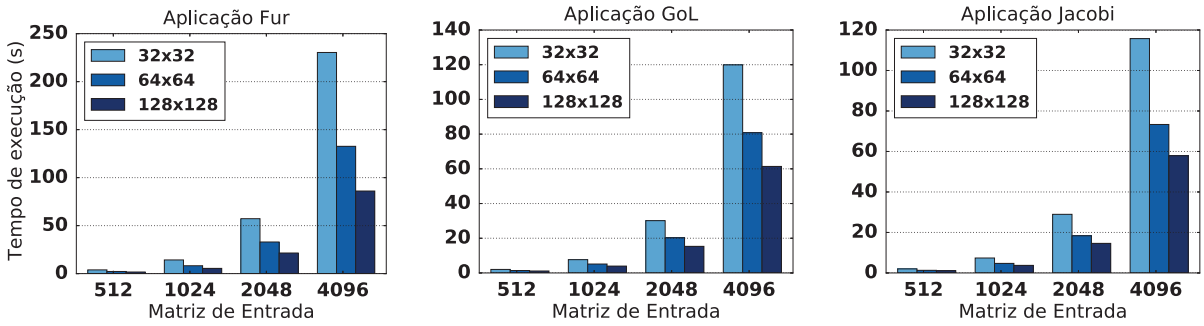


Figura 2. Tempos de execução com diferentes tamanhos de *tile* e matrizes.

para as diferentes plataformas paralelas, realizando de maneira transparente todo o gerenciamento de memória e transferência de dados entre dispositivos [Pereira et al. 2015].

A Figura 1b ilustra o funcionamento da computação estêncil em aplicações iterativas. Em cada iteração, uma máscara de vizinhança é utilizada na matriz de entrada para determinar o valor de cada célula da matriz de saída. Nesse exemplo, o valor de cada célula da matriz de saída é determinado em função dos valores das células vizinhas em todas as direções. Esse processo é realizado para todos os pontos da matriz de entrada, produzindo uma matriz saída da computação estêncil. Ao final de uma iteração, a matriz de saída será considerada como sendo a matriz de entrada da próxima iteração, gerando assim uma nova matriz de saída ao final da próxima iteração.

3. PSkel-MPPA

A implementação do PSkel para o processador MPPA-256 segue um modelo mestre/escravo. Um processo mestre é executado no subsistema de E/S conectado à memória LPDDR3 de 2 GB, sendo responsável por alocar os dados de entrada, distribuir as tarefas e controlar processos escravos. São criados 16 processos escravos, um para cada *cluster* de computação. Devido às limitações de memória dos *clusters*, o mestre subdivide a matriz de entrada em porções menores denominadas *tiles* e as envia para os processos escravos. O escalonamento dos *tiles* em cada iteração é feito sob demanda: cada processo escravo recebe um *tile*, realiza a computação do mesmo utilizando o *kernel* de computação estêncil definido pelo usuário e então devolve o resultado para o mestre. A paralelização da computação dentro do *cluster* é feita com auxílio da API OpenMP (uma *thread* é criada para cada PE). Ao ficar ocioso, um processo escravo recebe um novo *tile* a ser computado (caso ainda existam *tiles* a serem computados). Toda a comunicação entre os processos mestre e escravos é feita utilizando-se a API de comunicação do MPPA-256.

Para reduzir a quantidade de comunicações na NoC, é possível que um processo escravo compute diversas iterações do estêncil antes de enviar o resultado para o processo mestre. Para isso, devido as dependências entre as células vizinhas do padrão estêncil, os *tiles* necessitam ser enviados juntamente com uma margem extra de vizinhança aos escravos. O tamanho da vizinhança enviado é proporcional à quantidade de iterações da computação estêncil que poderão ser feitas no escravo sem a necessidade de comunicação com o processo mestre. Todavia, essa técnica exige que computações redundantes sejam feitas pelos escravos, além de aumentar a quantidade de memória ocupada nos mesmos.

4. Resultados

Foram utilizadas três aplicações estêncil para a realização dos experimentos. A aplicação **Fur** tem como objetivo modelar a formação de padrões sobre a pele de animais. A

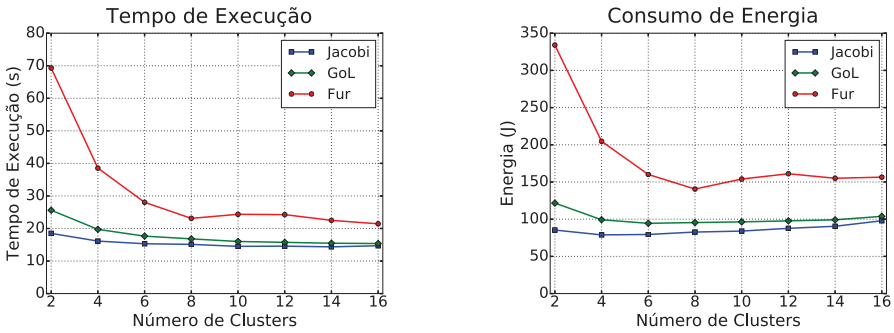


Figura 3. Desempenho e consumo de energia em três aplicações estêncil.

aplicação **Jacobi** implementa o método iterativo de Jacobi para resolução de equações matriciais. Por fim, a aplicação **GoL** é um autômato celular que implementa o Jogo da Vida de Conway. A descrição completa dessas aplicações pode ser encontrada em [Pereira et al. 2015, Podestá Jr. et al. 2016].

Os resultados de tempo e energia foram obtidos através de ferramentas disponíveis no MPPA-256. Os experimentos foram executados sobre matrizes de entrada de 512x512, 1024x1024, 2048x2048 e 4096x4096, alterando-se o tamanho dos *tiles* em 32x32, 64x64 e 128x128 (Figura 2). Além disso, foram feitos testes de escalabilidade, fixando-se o tamanho da matriz de entrada em 2048x2048 e do *tile* em 128x128 e alterando-se o número de *clusters* utilizados (Figura 3). Os valores representam médias de 5 execuções, com um coeficiente de variação máximo inferior à 0,4%. Os resultados mostram um *speedup* que varia entre 1,2x e 2,7x à medida que aumenta-se o tamanho dos *tiles*. Porém, a solução proposta apresenta problemas de escalabilidade devido ao tempo de comunicação ser muito grande em relação ao tempo total de execução da computação nos *clusters*. Esse impacto também é observado no consumo de energia obtido quando aumenta-se o número de *clusters*.

5. Conclusão

Neste trabalho foi proposta uma adaptação completa do *framework* PSkel para o processador *manycore* MPPA-256. Os resultados mostraram que mesmo obtendo resultados razoáveis ao aumentar o tamanho dos *tiles* da computação, com um ganho de até 2,7x. A comunicação é um grande problema na computação, tendo que ser otimizada para uma melhor utilização das características do MPPA-256. Como trabalhos futuros, pretende-se otimizar a comunicação e comparar os resultados de tempo e energia com outros processadores (CPU e GPU).

Referências

Francesquini, E., Castro, M., Penna, P. H., Dupros, F., de Freitas, H. C., Navaux, P. O. A., and Méhaut, J.-F. (2014). On the Energy Efficiency and Performance of Irregular Applications on Multicore, NUMA and Manycore Platforms. *JPDC*, pages 32–48.

Pereira, A. D., Ramos, L., and Góes, L. F. W. (2015). PSkel: A Stencil Programming Framework for CPU-GPU Systems. *CCPE*, 27(17):4938–4953.

Podestá Jr., E., Pereira, A. D., Penna, P. H., Rocha, R. C., Castro, M., and Góes, L. F. W. (2016). PSkel-MPPA: Uma Adaptação do Framework PSkel para o Processador Manycore MPPA-256. In *ERAD/RS*, pages 299–302, São Leopoldo, Brazil. SBC.

Rahman, S. M. F., Yi, Q., and Qasem, A. (2011). Understanding stencil code performance on multicore architectures. In *CF*, pages 30:1–30:10. ACM.