PSkel-MPPA: Uma Adaptação do Framework PSkel para o Processador Manycore MPPA-256

Emmanuel Podestá Jr.[1], **Alyson D. Pereira**[1], **Pedro H. Penna**[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,
pedro.penna@posgrad.ufsc.br, rcor@pucminas.br,
marcio.castro@ufsc.br, lfwgoes@pucminas.br

Resumo. *Neste artigo é proposta uma adaptação de um framework baseado em esqueletos paralelos que oferece suporte ao padrão estêncil (PSkel) para um processador manycore emergente denominado MPPA-256. Os resultados mostraram que a solução adotada apresenta boa escalabilidade, oferecendo reduções de tempo de execução e consumo de energia de até* $6\times$.

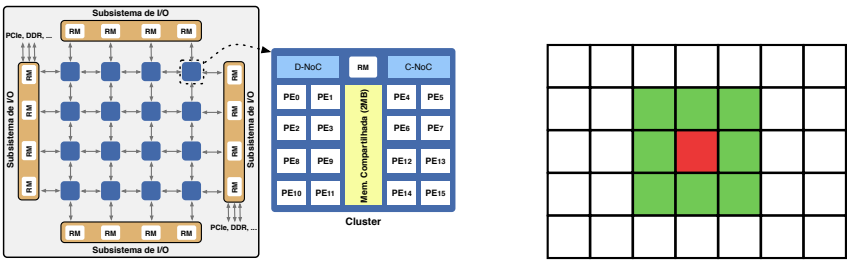
1. Introdução

Esqueletos paralelos modelam padrões de computação e coordenação que ocorrem frequentemente em aplicações paralelas. A principal vantagem dos esqueletos é permitir uma abstração de alto nível através do uso de interfaces genéricas, permitindo esconder do programador detalhes sobre paralelismo, sincronização e possíveis otimizações de código. Consequentemente, o programador pode concentrar-se apenas nos detalhes relacionados ao contexto específico da sua aplicação.

Diversos padrões de esqueletos paralelos são conhecidos na literatura, tais como *map*, *reduce*, *pipeline*, *scan* e *estêncil*. O padrão estêncil é um dos padrões de programação paralela mais utilizados na indústria e academia, em aplicações como simulação de física de partículas, previsão meteorológica, termodinâmica, resolução de funções diferenciais, manipulação de imagens, entre outras [Rahman et al. 2011].

Nesse contexto, o PSkel é um *framework* de programação do padrão estêncil, escrito em C++ com suporte para geração de código executável para processadores *multicore* e *Graphics Processing Units* (GPUs). Utilizando uma abstração de alto nível, o usuário define o *kernel* da computação estêncil, enquanto o *framework* se encarrega de executar a computação em CPU e/ou GPU, auxiliando no gerenciamento de memória e transferência de dados entre dispositivos [Pereira et al. 2015].

Neste artigo é proposta uma adaptação do PSkel para um processador *manycore* emergente fabricado pela empresa francesa Kalray denominado MPPA-256. Esse processador possui 256 núcleos de processamento em um único *chip* de alto desempenho e baixo consumo de energia. Porém, devido às suas diversas características peculiares e ao baixo nível de abstração requerido, desenvolver aplicações paralelas para esse processador é



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

Figura 1. Visão geral do MPPA-256 (esquerda) e uma ilustração do padrão estêncil oferecido pelo PSkel (direita).

uma tarefa desafiadora. Nesse contexto, a adaptação do PSkel para o MPPA-256 permitirá abstrair detalhes de baixo nível dessa arquitetura, além de permitir que aplicações já existentes em PSkel possam ser portadas para essa nova plataforma sem a necessidade de modificações de código.

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 preliminares são apresentados na Seção 4 e por fim a conclusão é apresentada na Seção 5.

2. Conceitos Básicos

2.1. MPPA-256

O processador MPPA-256 é composto por 256 núcleos de processamento de usuário cadenciados à 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 o consumo de energia de aplicações paralelas no MPPA-256 pode ser bem inferior que em arquiteturas *multicore* clássicas [Francesquini et al. 2014]. Porém, desenvolver aplicações paralelas otimizadas para o MPPA-256 é um grande desafio 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 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 uma matriz de entrada com uma máscara de vizinhança (em verde) para um determinado ponto central (em vermelho). Dados um ponto central da matriz de entrada e seus vizinhos, o *kernel* estêncil é responsável por realizar a computação que produz o ponto equivalente da matriz de saída. Esse mesmo processo é realizado para todos os pontos da matriz de entrada, produzindo a saída da computação estêncil.

3. PSkel-MPPA

Para a implementação do *framework* PSkel para o processador MPPA-256, devido a características e restrições da própria arquitetura, adotou-se um modelo mestre/escravo. O 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. Cada *cluster* executa um único processo escravo que será responsável por gerenciar a computação no *cluster*.

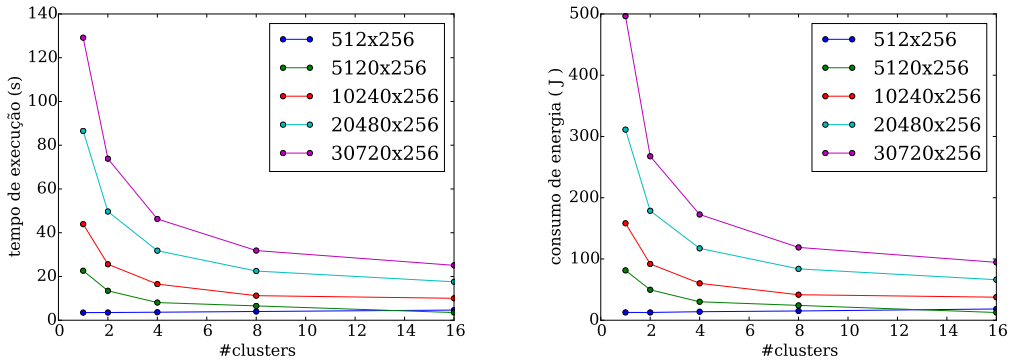
Devido às limitações de memória dos *clusters* (apenas 2 MB de memória em cada *cluster*), o mestre subdivide a matriz de entrada em porções menores denominadas *tiles* e as envia para os processos escravos. O escalonamento dos *tiles* é feito por etapas de maneira iterativa, de acordo com o número de *clusters* disponíveis, seguindo uma estratégia *round-robin*. Toda a comunicação entre os processos mestre e escravos é feita utilizando-se a API de comunicação assíncrona disponível para o MPPA-256.

Cada processo escravo realiza a computação do *tile* recebido no *cluster* utilizando o *kernel* de computação estêncil definido pelo usuário. A paralelização da computação dentro do *cluster* é feita com auxílio da API OpenMP. Em cada *cluster* podem ser criadas até 16 *threads* (uma para cada PE). Após a computação do *kernel* estêncil, os *tiles* resultantes são enviados ao processo mestre e então agrupados em uma matriz, produzindo o resultado final da computação estêncil, após todas as etapas da distribuição dos *tiles*.

4. Resultados Preliminares

Para a realização dos experimentos foi utilizada uma aplicação que tem como objetivo modelar a formação de padrões sobre a pele de animais[1]. Nessa aplicação, a pele do animal é modelada por uma matriz bidimensional de células de pigmento que podem estar em um dos dois estados: colorida ou não-colorida. As células coloridas secretam ativadores e inibidores. Ativadores fazem uma célula central se tornar colorida; inibidores, por outro lado, fazem uma célula central se tornar não colorida. A diferença entre as potências dos ativadores e inibidores é responsável por decidir a coloração da célula central, onde mais ativadores resulta em uma célula colorida e mais inibidores resulta em uma célula

[1] http://ccl.northwestern.edu/netlogo/models/Fur



(a) Tempo de execução. (b) Consumo de energia.

Figura 2. Experimento de escalabilidade variando o número de *clusters*.

não colorida. Nos casos em que as potências dos ativadores e inibidores forem iguais, a cor da célula permanece inalterada.

Com o objetivo de verificar o desempenho e o consumo total de energia do MPPA-256, foram realizados experimentos com o PSkel-MPPA variando-se o número de *clusters* para diferentes tamanhos de entrada (Figura 2). No geral, o PSkel-MPPA apresentou boa escalabilidade, com uma redução do tempo de execução e do consumo de energia com o aumento do número de *clusters* utilizados. Porém, não observamos ganhos para matrizes de tamanho 512×256, devido ao compromisso entre sincronização, comunicação e o tamanho do grão de computação envolvido. Para matrizes de tamanho 5120×256 observamos reduções no tempo de execução e no consumo de energia de até $6\times$ em relação às execuções com apenas um *cluster*. Para todos os casos de entradas maiores, os ganhos obtidos foram superiores à $4\times$ em relação às execuções com apenas um *cluster*.

5. Conclusão

Neste trabalho foi proposta uma adaptação do *framework* PSkel para o processador *many-core* MPPA-256. Os resultados preliminares mostraram que a solução adotada apresenta um bom potencial, se mostrando bastante escalável, proporcionando reduções no tempo de execução e no consumo de energia de até $6\times$. Como trabalhos futuros, pretendemos realizar otimizações no código, oferecer suporte para aplicações estêncil iterativas e comparar o desempenho da solução para o MPPA-256 com as soluções para 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. *J. Parallel Distrib. Comput.*, 76:32–48.

Pereira, A. D., Ramos, L., and Góes, L. F. W. (2015). PSkel: A stencil programming framework for cpu-gpu systems. *Concurrency and Computation: Practice and Experience*, 27(17):4938–4953.

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