Otimização da Comunicação em Aplicações Estêncil Paralelas Implementadas com o PSkel no Processador MPPA-256

Bruno Marques do Nascimento[1], **Emmanuel Podestá Jr.**[1], **Márcio Castro**[1]

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

{bruno.mn, emmanuel.podesta}@grad.ufsc.br, marcio.castro@ufsc.br

Resumo. *Neste artigo é proposta uma versão otimizada do framework PSkel para o processador MPPA-256 com base no uso de uma nova biblioteca de comunicação desenvolvida especificamente para esse processador. Ela permite um melhor desempenho na comunicação além de oferecer uma maior abstração para o desenvolvedor. Os resultados mostraram que a implementação proposta possui um desempenho superior em até 8x em relação à implementação anterior.*

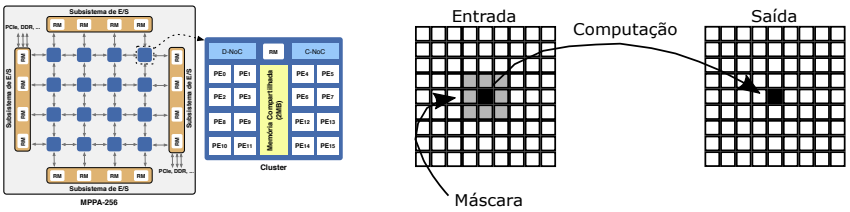
1. Introdução

Processadores *manycore* de baixa potência (*low-power manycore processors*) desenvolvidos recentemente, tais como o Sunway SW26010 [Fu et al. 2016] e Kalray MPPA-256 [Francesquini et al. 2014], apresentam melhor eficiência energética em comparação com processadores de propósito geral atuais. O supercomputador atualmente em primeiro lugar no Top500[1] (*Sunway TaihuLight*) foi o primeiro a empregar este tipo de processador, possuindo 40.960 processadores Sunway SW26010 de 260 núcleos cada. Contudo, desenvolver aplicações paralelas otimizadas para esses processadores é bastante desafiador [Francesquini et al. 2014]. Os núcleos de processamento possuem memórias *cache* não coerentes e são distribuídos em *clusters*, os quais possuem uma memória local de tamanho limitado, e se comunicam por meio de uma *Network-on-Chip* (NoC), caracterizando assim um modelo com espaço de endereçamento compartilhado e distribuído.

Nesse contexto, Podestá *et al.* [Podestá Jr. et al. 2017b] propuseram o uso de esqueletos paralelos para simplificar o desenvolvimento de aplicações nesses processadores. Mais precisamente, foi proposta uma adaptação do *framework* PSkel, o qual implementa o padrão paralelo estêncil, para o processador MPPA-256. O padrão estêncil é um dos padrões mais utilizados em áreas importantes, como física quântica, processamento de imagens e previsão do tempo. Apesar da adaptação proposta ter apresentado um bom potencial, foi identificado que a mesma apresentava gargalos na comunicação via NoC.

O presente artigo apresenta uma versão otimizada da proposta descrita anteriormente através da exploração de uma nova *Application Programming Interface* (API) de comunicação desenvolvida para o MPPA-256, permitindo assim otimizações na comunicação via NoC. As otimizações implementadas permitem ganhos significativos no desempenho e no consumo de energia. Este artigo está organizado da seguinte forma. A Seção 2 apresenta os principais conceitos do processador MPPA-256 e do *framework* PSkel. A Seção 3 discute as otimizações propostas. Os resultados obtidos são apresentados na Seção 4. Por fim, Seção 5 apresenta as conclusões deste trabalho.

[1] http://top500.org



(a) MPPA-256. (b) O padrão estêncil.

Figura 1. Visão geral do MPPA-256 e do padrão estêncil [Podestá Jr. et al. 2017a].

2. Fundamentação Teórica

2.1. MPPA-256

O MPPA-256 é um processador *manycore* desenvolvido pela empresa francesa Kalray. Esse processador possui 256 núcleos de processamento de 400 MHz denominados *Processing Element*s (PEs). Além dos PEs, o MPPA-256 apresenta 32 núcleos dedicados à gerência de recursos denominados *Resource Manager*s (RMs). Os RMs são responsáveis por gerenciar a Entrada e Saída (E/S) e controlar comunicações entre *clusters* e/ou subsistemas. Os PEs são distribuídos em 16 *clusters*, onde cada *cluster* possui 16 PEs e 1 RM. Além disso, o MPPA-256 possui 4 subsistemas de E/S, contendo 4 RMs cada um. Por fim, toda a comunicação no MPPA-256 é realizada por meio de uma NoC *torus* 2D. A arquitetura do MPPA-256 é ilustrada na Figura 1(a).

Trabalhos 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: o modelo de memória distribuída presente no MPPA-256, a capacidade de memória dentro do *chip* e a comunicação explícita através da NoC. Mais detalhes sobre esses desafios são apresentados em [Podestá Jr. et al. 2017a].

2.2. PSkel

O PSkel é um *framework* de programação em alto nível para aplicações baseadas no padrão estêncil, oferecendo suporte para a execução dessas aplicações em ambientes heterogêneos, incluindo *Central Processing Unit* (CPU) e *Graphics Processing Unit* (GPU). O usuário do *framework* será responsável por definir o *kernel* principal da computação, enquanto o *framework* irá realizar a distribuição de dados e escalonamento de tarefas de forma transparente [Pereira et al. 2015]. O padrão estêncil, ilustrado pela Figura 1(b), funciona da seguinte forma. Para cada elemento de uma estrutura n-dimensional é computado um novo valor relativo aos vizinhos do elemento atual. Os vizinhos de um elemento são determinados pela máscara da computação. Por fim, cada novo valor computado é atribuído à sua célula respectiva em uma estrutura n-dimensional de saída. Em aplicações estêncil iterativas, a estrutura de saída é utilizada como estrutura de entrada de uma nova iteração da aplicação.

3. Versão Otimizada do PSkel para o MPPA-256

A adaptação anterior do PSkel proposta por Podestá *et al.* [Podestá Jr. et al. 2017b], denominada **IPC** neste artigo, utilizava uma API similar à POSIX *Inter-Process Communication* (IPC) para comunicação. Nela, eram utilizados **portais de comunicação** para o envio de dados e o método de ***strides*** para gerenciar explicitamente o envio e recebimento de *tiles*. Porém, o desempenho desta implementação mostrou-se prejudicado pelo elevado tempo despendido nas operações de comunicação. Com o advento da nova API de

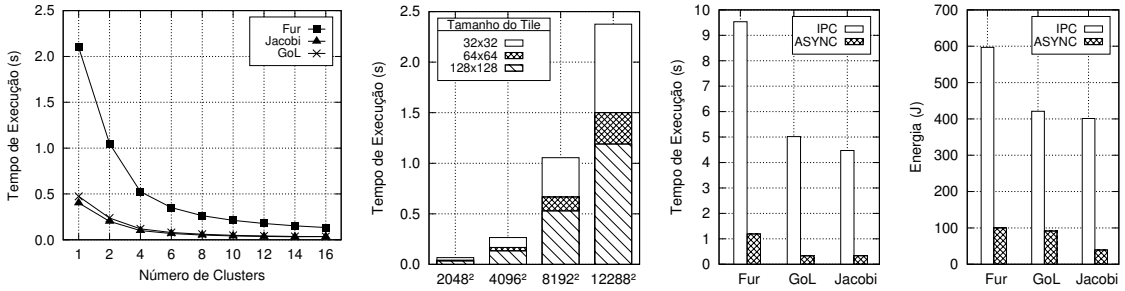
comunicação para o MPPA-256 foi possível mitigar o impacto provocado ao desempenho por estas operações. Esta API trabalha com o conceito de **segmentos**, que possibilita o endereçamento de segmentos de memória externos à memória local do *cluster*. A função `mppa_async_segment_create()` permite criar um segmento de memória no subsistema de E/S ligado à *Low Power Double Data Rate 3* (LPDDR3) ao passo que a função `mppa_async_segment_clone()` permite clonar este segmento remoto na memória local de um *cluster*, resultando em um segmento local que referencia o segmento criado remotamente na LPDDR3. Este segmento criado é posteriormente utilizado no âmbito das funções de transferência de dados disponibilizadas pela API, para endereçar o segmento de memória remoto. Estas funções são capazes de copiar dados locais para o segmento de memória criado, através dos métodos `put` e também copiar dados de um segmento remoto e armazená-los localmente através dos métodos `get`.

Na proposta apresentada neste artigo, denominada **ASYNC**, a responsabilidade pela divisão dos *tiles* foi transferida aos *clusters* e não mais realizada no subsistema de E/S. Além disso, os métodos `put` e `get` são executados apenas pelos *clusters*. Mais precisamente, os métodos utilizados foram `mppa_async_sget_block2d()` e `mppa_async_sput_block2d()`, que são métodos próprios para a manipulação de matrizes e submatrizes, presentes na operação de divisão do *grid* em *tiles* na computação estêncil. Estas facilidades agregadas ao elevado nível de abstração da API demonstram com nitidez as vantagens proporcionadas pelo uso desta nova API no desenvolvimento de aplicações estêncil. Os benefícios em termos de desempenho serão analisados na Seção 4.

4. Resultados Experimentais

Esta seção apresenta os resultados obtidos com a solução proposta, comparando-a com a versão apresentada em [Podestá Jr. et al. 2017b]. Todas as métricas foram obtidas com auxílio de ferramentas disponíveis no MPPA-256. Os dados se referem a execução de uma única iteração das aplicações Fur, GoL e Jacobi, as quais são descritas em mais detalhes em [Podestá Jr. et al. 2017b]. Foram realizadas 5 repetições de cada experimento, computando-se a média aritmética dos valores. A variabilidade dos valores obtidos foi extremamente pequena (desvio-padrão inferior à 1%), pois as *threads* da aplicação são executadas de maneira ininterrupta no MPPA-256.

Na análise da escalabilidade da versão ASYNC, apresentada pela Figura 2(a), foi utilizada uma matriz de tamanho 4.096x4.096 e *tiles* de tamanho 128x128. Foi obtido um ganho de desempenho de até $15,6$x com 16 *clusters* em relação à execução com um único *cluster* (aplicação Fur). Por outro lado, a Figura 2(b) apresenta o impacto do tamanho dos *tiles* sobre o desempenho nessa aplicação. Para obter uma maior precisão nos resultados em relação ao tempo de execução foram utilizadas matrizes de tamanho maior que 2.048x2.048. Além disso, devido à memória limitada, não foram utilizadas matrizes de entrada e *tiles* maiores que 12.288x12.288 e 128x128, respectivamente. Os resultados mostram que o tempo de execução da aplicação diminui à medida em que se aumenta o tamanho dos *tiles*, obtendo-se ganhos de até 2x. Este comportamento é decorrente do melhor aproveitamento da vazão da NoC, realizando menos transferências com maior quantidade de dados. As aplicações GoL e Jacobi também apresentaram comportamentos similares. As Figuras 2(c) e 2(d) apresentam a comparação de tempo e consumo de energia entre as versões ASYNC e IPC. Nesse experimento, foi utilizada uma matriz de entrada de tamanho 12.288x12.288, *tiles* de tamanho 128x128 e 16 *clusters*. Como pode ser observado, os resultados mostram que o tempo de execução da versão ASYNC é, aproximadamente, 8x menor em relação ao tempo da versão IPC. O resultado da ener-



(a) Escalabilidade. (b) *Tiles* vs. tempo (Fur). (c) Tempo. (d) Energia.

Figura 2. Escalabilidade (a) e impacto do tamanho dos *tiles* (b) na versão ASYNC. Comparação do tempo (c) e consumo de energia (d) do ASYNC com a versão IPC.

gia consumida segue um comportamento similar, apresentando uma eficiência energética superior em até 6x a favor da versão ASYNC. A versão IPC possui funções que manipulam dados (no mínimo, $7,1\%$ do tempo total) e distribuições de tarefas (no mínimo, 72% do tempo total), trazendo um impacto negativo sobre o tempo de execução.

5. Conclusão

O desenvolvimento de aplicações otimizadas para processadores *manycore* de baixa potência é bastante desafiador devido a fatores importantes tais como a existência de um modelo de programação híbrido, capacidade limitada de memória no *chip*, ausência de coerência de *cache*, entre outros. Neste artigo foi apresentada uma nova versão otimizada do *framework* PSkel para o processador MPPA-256. Os resultados mostraram que a nova versão obteve ganhos de desempenho de até 8x e uma redução no consumo de energia de até 6x, em comparação com a solução inicial proposta em [Podestá Jr. et al. 2017b]. Como trabalhos futuros, pretende-se implementar o suporte para a execução de aplicações estêncil iterativas, comparar o desempenho e consumo de energia com outros processadores e implementar suporte a matrizes tridimensionais.

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.

Fu, H. et al. (2016). The Sunway TaihuLight supercomputer: system and applications. *Science China Information Sciences*, 59(7):1–16.

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.

Podestá Jr., E., Pereira, A. D., Rocha, R. C., Castro, M., and Góes, L. F. W. (2017a). Uma Implementação do Framework PSkel com Suporte a Aplicações Estêncil Iterativas para o Processador MPPA-256. In *ERAD/RS*, pages 395–398, Ijuí, Brazil. SBC.

Podestá Jr., E., Pereira, A. D., Rocha, R. C. d. O., Castro, M., and Góes, L. F. W. (2017b). Execução Energeticamente Eficiente de Aplicações Estêncil com o Processador Manycore MPPA-256. In *Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)*, Campinas, SP.