Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística (INE)

Relatório Final - PIBIC 2018/2019

Projeto: Otimização do Benchmark CAP Bench para o Processador Manycore de Baixo Consumo Energético MPPA-256

Bolsista: David Grunheidt Vilela Ordine
Orientador: Prof. Dr. Márcio Castro
Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD), INE/UFSC

Florianópolis, 5 de Agosto de 2019

Resumo

Similar ao que aconteceu com os processadores *single-core*, ao longo de sua evolução, as tecnologias voltadas para computação de alto desempenho (HPC) depararam-se com uma barreia de potencia, a qual torna desvantajoso o *trade-off* entre gasto energético e ganho em desempenho. Desta maneira, um novo buraco dentro desta área de pesquisa surgiu, o qual foi preenchido com o ramo de processadores *manycore* de baixo consumo energético, tais quais o MPPA-256 e o Adapteva Epiphany. Devido a questões arquiteturais, como a quantidade limitada de memoria em cada *cluster* de computação (CC) e o não compartilhamento de memória entre *clusters*, o desafio relacionado a estes processadores e, particularmente, ao MPPA-256, é a implementação de aplicações que beneficiam-se totalmente do seu *hardware*. Neste projeto foram propostas otimizações para as aplicações do CAP Bench, a fim de mostrar que, apesar dos desafios, são inúmeros os benefícios da utilização do MPPA-256, quando implementações são feitas de modo inteligente. Os resultados mostram que o novo *benchmark* superou, em desempenho, até ...x a implementação anterior.

Palavras-chave: *manycores*, MPPA-256, comunicação assíncrona.

Conteúdo

1 Introdução

Para que os supercomputadores atuais consigam alcançar de forma definitiva a computação em *exaescale*, é necessário que haja, de forma coesa, alto desempenho e consumo energético viável. Porém, assim como ocorreu com os avanços nas tecnologias de processadores *single-core*, os quais, nas ultimas três décadas, permitiram aumento no desempenho de um processador a uma taxa anual de 40% a 50% [Larus and Kozyrakis 2008], a dissipação de calor nos supercomputadores que utilizam processadores do tipo *multicore* chegou a um ponto que não mais permitiu a escalabilidade proporcional das variáveis citadas acima.

Seguindo os conceitos de *Green Computing*, estudos foram realizados a fim de encontrar um *trade-off* positivo entre desempenho e gasto energético, centrado na redução de gasto energético. O grande interesse da comunidade cientifica de HPC acerca deste tema foi um dos responsáveis por alavancar a produção de novos tipos de processadores, tais quais, os *manycores* de baixa potencia MPPA-256 [de Dinechin et al. 2013], SW26010, utilizado no supercomputador *Sunway TaihuLight* [Fu et al. 2016] e o Adapteva Epiphany [Olofsson et al. 2014].

Com proposito de validar as supostas qualidades do MPPA-256 e prover meios de comparação com outros processadores do estado da arte, *Souza et al*. implementaram o CAP Bench [Souza et al. 2016], *benchmark* que avalia ambos desempenho e gasto energético do processador, levando em conta diversos cenários. Em sua versão inicial, utilizava uma *Application Programming Interface* (API) de comunicaçao sincrona entre processos, denominada *Inter-Process Communication* (IPC) [de Dinechin et al. 2013]. Esta antiga API possui alguns lados negativos, como baixo nivel de abstração e realização de sincronizações implicitas, levando a queda de desempeneho.

Neste trabalho, a fim de implementar a otimização proposta, realizou-se o porte do CAP Bench com a nova API de comunicação assincrona entre processos da Kalray, a *MPPA Asynchronous Communication* (ASYNC) [Hascoët et al. 2017]. Esta API possui nivel de abstração superior a IPC, além de diferir na implementação quanto ao modelo de lógica de memória. Assim, ela simplifica a elaboração de aplicações para o MPPA-256, além de ganhar em desempenho e reduzir o custo energético, devido a sua caracteristica assincrona.

1.1 Justificativa

Os processadores *manycore* apresentados anteriormente diferem dos processadores gráficos (*Graphics Processing Units* – GPUs), como já mencionado seus *cores* são autônomos, o que possibilita a exploração de paralelismo de dados e tarefas. De modo geral, os processadores pertencentes à essa classe tem como características principais: (i) baixa potência (entre 10 W e 50 W); (ii) centenas ou até mesmo milhares de *cores* em um único *chip* operando em uma uma baixa frequência de relógio e (iii) pelo menos uma NoC para interconectar os *cores* ou grupos de *cores*. Uma outra arquitetura *manycore* bastante conhecida na atualidade é a Intel Xeon Phi. Apesar de possuir algumas das características dos *manycores* leves, a potência dissipada por esse processador é mais elevada (na ordem de 300 W).

Apesar de oferecerem potencialmente uma melhor eficiência energética quando comparados à processadores *multicore* de propósito geral, os processadores *manycore* leves apresentam diversas limitações que tornam o desenvolvimento de aplicações científicas eficientes um grande desafio [Castro et al. 2013].

Normalmente, esses processadores são construídos e otimizados para certos tipos de classes de aplicações embarcadas como por exemplo decodificação de vídeo e roteamento. Além disso, grande parte desses processadores possuem restrições de memória, como por exemplo memórias *cache* de tamanho bastante limitado. Também, esses processadores exigem que comunicações de dados estejam em conformidade com a topologia da NoC para que os custos de comunicação sejam consideravelmente reduzidos. Por fim, mostra-se necessário o uso de algoritmos de escalonamento inteligentes nesses processadores a fim de permitir uma melhor exploração dos núcleos de processamento. Para explorar de forma eficiente os recursos dos processadores *manycores*, faz-se necessário considerar três diferentes níveis:

1. **Nível aplicativo.** A grande maioria das aplicações científicas existentes são atualmente paralelizadas utilizando-se modelos de programação em memória compartilhada (por exemplo, POSIX threads e OpenMP) ou em memória distribuída (por exemplo, MPI). Porém, processadores manycore de baixo consumo como o MPPA-256 exigem a utilização de ambos os modelos de programação (compartilhado e distribuído). O desafio encontra-se em adaptar as aplicações científicas existentes para esse modelo híbrido, conforme as peculiaridades de cada arquitetura *manycore*.

2. **Nível intermediário.** A grande concentração de núcleos no mesmo *chip* exige que os programas paralelos explorem de maneira eficaz os recursos computacionais. Para que isso seja possível, mostra-se necessária a utilização de algoritmos de escalonamento e balanceamento de carga inteligentes capazes de se adaptarem a carga de trabalho das aplicações paralelas, permitindo assim uma melhor distribuição das tarefas a serem computadas nos núcleos de processamento [Penna et al. 2015]. A implementação dos algoritmos de escalonamento de tarefas e o balanceamento de carga precisa ser feita no nível intermediário, *i.e.*,, no ambiente de execução (*runtime*), para que as aplicações paralelas possam usufruir desses benefícios de maneira transparente.

3. **Nível de *hardware*.** A comunicação entre *clusters* é realizada através de NoCs, as quais se baseiam nos conceitos utilizados nas redes de interconexão para computadores paralelos [Freitas et al. 2008]. Uma NoC pode ser definida como um conjunto de roteadores e canais ponto-a-ponto que interconectam os *cores* ou grupos de *cores* de modo a suportar a comunicação entre eles. Embora as NoCs permitam uma maior escalabilidade em arquiteturas *manycore*, elas adicionam um certo custo ao sistema, pois os dados precisam trafegar através da rede. Portanto, tanto a estratégia de paralelização de uma aplicação científica quanto o ambiente de execução para *manycores* devem sempre que possível evitar que processos em execução em *cores* ou grupos de *cores* muito distantes se comuniquem frequentemente, evitando assim gargalos de comunicação devido ao aumento do tempo de comunicação ou perdas de pacotes [de Freitas et al. 2010, Avelar et al. 2011].

1.2 Objetivos

O objetivo desta pesquisa de iniciação científica é propor e implementar a otimização do *benchmark* CAP Bench para o processador *manycore* de baixo consumo energético MPPA-256. Os objetivos específicos deste projeto de pesquisa estão abaixo elencados:

1. Investigar a viabilidade do uso do MPPA-256 para a computação científica de alto desempenho;
2. Estudar as APIs de comunicação existentes para o MPPA-256.
3. Implementar um conjunto de aplicações paralelas para o MPPA-256 (*benchmark*) utilizando-se da API ASYNC;
4. Avaliar os custos e benefícios do MPPA-256 em relação ao desempenho e ao consumo de energia, assim como sua utilidade para a Computação Sustentável (*Green Computing*);
5. Difundir a pesquisa e os seus resultados através de produção científica de qualidade, em periódicos e eventos relevantes na área de Processamento Paralelo e Distribuído.

Nas seções seguintes são apresentados o desenvolvimento e os resultados produzidos, de acordo com o cronograma e as atividades propostas deste projeto de pesquisa.

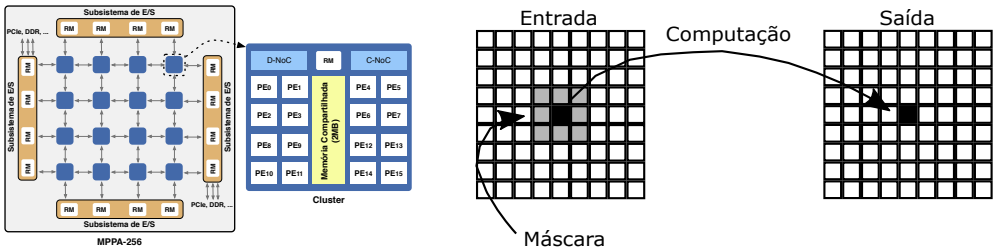
2 Revisão Bibliográfica

Esta seção apresenta a revisão bibliográfica sobre o processador *manycore* MPPA-256 e o *framework* PSkel e sua adaptação utilizada nesse trabalho. Por fim, são apresentados alguns trabalhos relacionados.

2.1 MPPA-256

O MPPA-256 é um processador *manycore* desenvolvido pela empresa francesa Kalray. Esse processador possui 256 núcleos e usuário e 32 núcleos de sistema para processamento a 400 MHz. Esses núcleos estão distribuídos entre 16 *clusters* de computação e 4 *clusters* de *Input/Output*(Entrada e Saída (E/S)), que se comunicam através de NoCs de dados e controle. O processador utilizado no desenvolver deste projeto de pesquisa possui uma memória global de baixa potência (LPDDR3) de 2GB conectada a um dos susistemas de E/S. A arquitetura do MPPA-256 é ilustrada na Figura 1(a). Cada cluster de computação tem os seguintes componentes:

- 16 núcleos chamados de *Processing Elements* (*Processing Element*s (PEs)), que são responsáveis por executar as *threads* de usuário (uma *thread* por PE), e não pode ser interrompida ou preemptada;



(a) MPPA-256. (b) O padrão estêncil.

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

- um *Resource Manager* (*Resource Manager* (RM)), responsável por executar o sistema operacional e gerenciar a comunicação;
- uma memória compartilhada de baixa latência de 2MB, que permite um grande banda e fluxo e dados e controle entre os PEs presentes no mesmo *cluster* de computação; e
- dois controladores de NoC, um para dados e outro para controle.

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 *Network-on-Chip* (NoC). Mais detalhes sobre esses desafios são apresentados em [Podestá Jr. et al. 2017a].

2.2 Padrão estêncil e PSkel

O PSkel é um *framework* de programação em alto nível para aplicações baseadas no padrão estêncil, baseado no conceito de esqueletos paralelos, oferecendo suporte para a execução dessas aplicações em ambientes heterogêneos, incluindo *Central Processing Unit* (CPU) e *Graphics Processing Unit* (GPU). PSkel oferece um interface única de programação, desacoplada do *back-end* de execução, permitindo que o usuário se preocupe apenas em implementar o *kernel* estêncil que descreve a computação, enquanto o *framework* fica responsável pela tradução das abstrações descritas para código paralelo de baixo nível em C++, gestão de memória e transferência de dados, tudo isso de forma transparente paar o usuário [Pereira et al. 2015].

2.3 PSkel-MPPA

A adaptação PSkel-MPPA, é uma adaptação do PSkel proposta por Podestá *et al*. [Podestá Jr. et al. 2017b], ela faz uso de uma API similar à POSIX IPC para comunicação, e será tratada como IPC no decorrer deste relatório. Nela, são utilizados portais de comunicação para o envio de dados e o método de *strides* para gerenciar explicitamente o envio e recebimento de *tiles*. Essa adaptação possíbilita o uso do *framework* PSkel com o processador *manycore* MPPA-256.

2.4 Trabalhos Relacionados

Devido a importância dos esqueletos paralelos, e especifcamente o padrão paralelo estêncil, muitos esforços de pesquisas recentes buscam melhorar o desempenho e o suporte desses esqueletos em processadores manycore. *Buono et al*. [Buono et al. 2013] portou um *framework* baseado em esqueletos paralelos, chamado *FastFlow*, para o processador *manycore* TilePro64, que possui 64 núcleos de processamento idênticos, interconectados por uma malha de NoC. Similarmente, *Thorarensen et al*. [Thorarensen et al. 2016] apresentou um novo *back-end* do *framework* SkePU para o processador *manycore* Myriad2. Que possui como característica uma arquitetura heterogenea, visando dispositivos com restrição de energia e principalmente aplicações de visão computacional. *Gysi et al*. [Gysi et al. 2015] propôs um *framework* para otimização automática da repartição de computações estêncil em sistemas híbridos de CPU e GPU.

Recentes trabalhos estudaram o desempenho e/ou a eficiência energética de processadores manycore de baixa potência. *Totoni et al*. [Totoni et al. 2012] comparou a potência e o desempenho do *Intel's Single-Chip*

Cloud Computer (SCC) com outros tipos de *CPUs* e *GPUs*. Porém, eles mostraram que não existe um solução única que entrega o melhor troca entre potência e performance, os resultados mostram que *manycores* são uma oportunidade para o futuro. *Souza et al*. [Souza et al. 2016] propôs um conjunto de *benchmarks* para avaliar o MPPA-256 manycore processor. O *benchmark* oferece diversas aplicações que utilizam padrões paralelos, tipos de trabalho, intensidade de comunicação e estratégias de carga de trabalho, adequado para uma ampla compreensão do desempenho e consumo de energia do MPPA-256 e novos *manycores* que estão por vir. *Francesquini et al*. [Francesquini et al. 2014] avaliou três diferentes classes de aplicação (consumo de CPU, consumo de memória e uma composição híbrida dos dois tipos anteriores) utilizando plataformas de alto paralelismo como o MPPA-256 em uma plataforma NUMA de 24 nós e 192 núcleos. Eles mostraram que as arquiteturas *manycore* podem ser competitivas, mesmo se a aplicação é irregular por natureza.

De acordo com relevante conhecimento na área, o PSkel-MPPA é a primeira implementação completa de um *framework* com uso de padrões paralelos no MPPA-256. A solução proposta livra os programadores da necessidade de lidar explicitamente com a gestão de comunicação e envio de dados pela NoC, assim como a preocupação de lidar com um ambiente híbrido de execução e a ausência de coerência de cache no MPPA-256.

3 Proposta e implementação de otimização no PSkel-MPPA

As otimizações propostas e implementação das mesmas continua aplicando o modelo mestre-trabalhador, que é um dos padrões de computação paralela que pode ser utilizado quando existem múltiplos núcleos de processamento. O processo mestre é executado no *cluster* de E/S conectado a memória LPDDR3, aonde os dados de entrada e saída (os *Array2Ds*) são alocados, enquanto os processos dos trabalhadores são executados nos *clusters* de computação (um processo trabalhador por *cluster* de computação) para realizar a computação estêncil. Devido a memória limitada nos *clusters* de computação (2MB), o tamanho do *Array2D* é particionado em *tiles* de tamanho fixo definido pelo usuário para serem enviados à eles. Quando se trata de particionamento de computação estêncil, é necessário tratar as depedências de vizinhança provenientes do padrão paralelo estêncil, antes de particionar os dados de entrada.

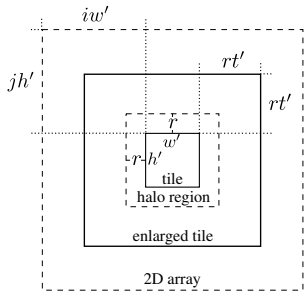


Figura 2: Técnica de *tiling* 2D [Rocha et al. 2017].

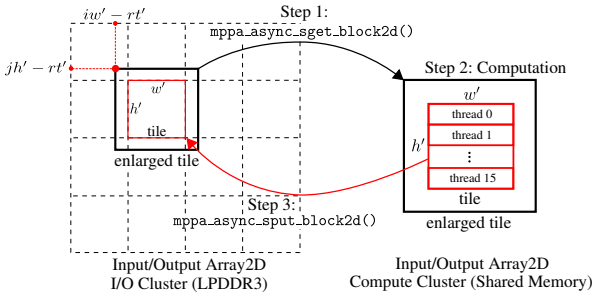
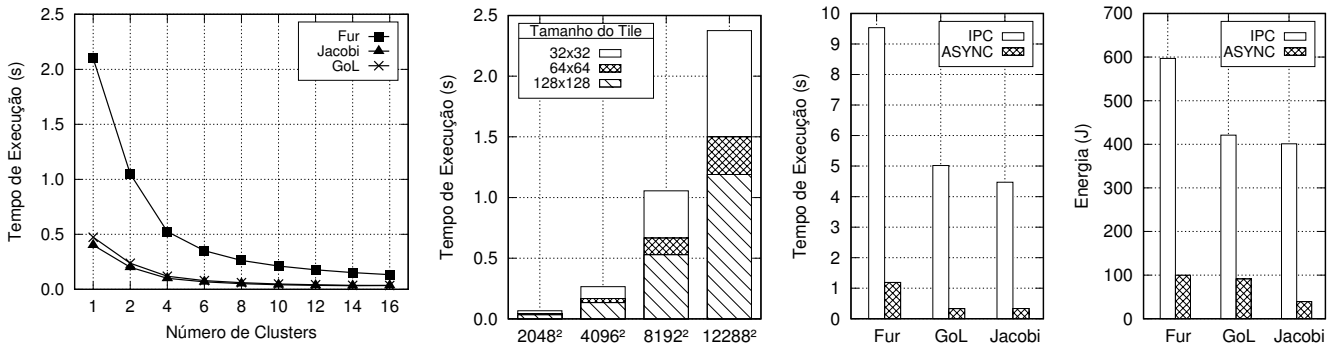


Figura 3: Comunicações com `block2d`.

O fluxo de execução do PSkel-MPPA ocorre da seguinte forma. Durante a fase de inicialização, o processo mestre que está executando no *cluster* de E/S aloca os dados de entrada e saída na *Low Power Double Data Rate 3* (LPDDR3), e cria um segmento específico para cada uma deles. Em seguida, ele calcula o número de *tiles* engordados que serão produzidos assim como suas dimensões baseado em: i) parametros definidos pelo usuário, como o tamanho da entrada de dados e as dimensões do *tile* lógico, o número de *clusters* de computação e o número de iterações internas; e ii) parâmetros do *kernel* estêncil, como o tamanho da máscara. Então, são lançados até 16 processos trabalhadores (um em cada *cluster* de computação) e é informado a cada processo trabalhador o número de *tiles* engordados gerados, suas dimensões e o subconjunto de *tiles* que cada *cluster* será responsável por processar. Por fim, o processo mestre aguarda até que todos os trabalhadores terminem de computar. Cada processo trabalhador, por outro lado, aloca dados para armazenar os *tiles* engordados de entrada e de saída na memória local do *cluster* de computação e clona ambos os segmentos de entrada e saída que foram criados pelo processo mestre para realizar futuras transferências de dados. A fase de inicialização tanto do mestre como do trabalhor está encapsulada na classe `Stencil2D`. Essa primeira etapa do fluxo de execução diferencia-se da anteriormente presente no PSkelMPPA, pois agora esta comunicação é realizada uma única vez



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

Figura 4: 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.

na inicialização, o que não ocorria anteriormente, já que exisitiam trocas de mensagens de sincronização a cada laço da computação iterativa.

As etapas acima mencionadas estão retratadas na Figura 3, para sua implementação foi utilizada uma nova API de comunicação asíncrona (ASYNC) disponível para o MPPA-256, que difere da API anteriormente utilizada, a IPC. Abaixo elas são descritas em mais detalhes:

Etapa 3. Após a computação do *kernel* estêncil, o*tile* lógico resultante é transferido de volta para a LPDDR3. A função `mppa_async_sput_block2d()` é usada para esse propósito, permitindo que o *tile* lógico seja extraído do *tile* engordado na memória local do *cluster* de computação e seja transferido para sua posicão correspondente no segmento de saída remoto.

Felizmente, todas a complexas tarefas relacionas a técnica de *tiling*, comunicação via NoC e adaptações discutidas nessa seção são transparentes para os desenvolvedores, tendo em vista que estão incluidas no *back-end* do PSkelMPPA. Isso significa que aplicações desenvolvidas com o *framework* PSkel podem executar perfeitamente no MPPA-256 sem nenhuma alteração no seu código fonte.

4 Resultados

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. A aplicação ***Fur*** realiza a simulação de padrões de pigmento sobre pelos de animais. A aplicação ***GoL*** é um autômato celular que implementa o Jogo da Vida de Conway. Por fim, a aplicação ***Jacobi*** implementa o método de Jacobi para a resolução de equações matriciais.

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.

5 Conclusão

6 Avaliação PIBIC: Benefícios e Formação Científica

Participar deste projeto de pesquisa foi enriquecedor, uma experiência única que teve como peça chave a dedicação, cobrança e incentivo de meu orientador. Assim que comecei meu projeto de pesquisa tive o seu incentivo para participar do XVIII Simpósio de Sistemas Computacionais de Alto Desempenho (WSCAD) que foi realizado em conjunto com o *International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD) na cidade de Campinas, São Paulo. Participei como um espectador e foi uma experiência nacional e internacional inesquecível, aonde conheci renomados cientistas e entusiastas do meio acadêmico, além de ser apresentado aos mais recentes projetos desenvolvidos na área.

Neste ano de pesquisa e desenvolvimento produzi um artigo científico que apresentei na 18ª Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul (ERAD/RS 2018), que ocorreu na cidade de Porto Alegre, Rio Grande do Sul. Artigo este que culminou em uma significativa contribuição para um artigo aceito em um evento internacional prestigiado na área de programação paralela e distribuída, o *24th International European Conference On Parallel and Distributed Computing* (Euro-Par 2018).

O desenvolvimento deste projeto de iniciação científica gerou um arcabouço de conhecimento na área acadêmica de maneira vertiginosa, além de ampliar meus horizontes e esclarecer o que é de fato a área acadêmica e o seu papel fundamental no desenvolvimento tecnológico e intelectual da sociedade.

Referências

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

[Buono et al. 2013] Buono, D., Danelutto, M., Lametti, S., and Torquati, M. (2013). Parallel patterns for general purpose many-core. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 131–139.

[Castro et al. 2013] Castro, M., Francesquini, E., Nguélé, T. M., and Méhaut, J.-F. (2013). Analysis of computing and energy performance of multicore, numa, and manycore platforms for an irregular application. In *Proceedings of the 3rd Workshop on Irregular Applications: Architectures and Algorithms*, IA3 '13, pages 5:1–5:8, New York, NY, USA. ACM.

[de Dinechin et al. 2013] de Dinechin et al. (2013). A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor. In *International Conference on Computational Science (ICCS)*, volume 18, pages 1654–1663, Barcelona, Spain. Elsevier.

[de Freitas et al. 2010] de Freitas, H. C., Schnorr, L. M., Alves, M. A. Z., and Navaux, P. O. A. (2010). Impact of parallel workloads on noc architecture design. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 551–555.

[Francesquini et al. 2014] 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. *Journal of Parallel and Distributed Computing (JPDC)*, 76:32–48.

[Freitas et al. 2008] Freitas, H. C., Santos, T. G. S., and Navaux, P. O. A. (2008). Design of programmable noc router architecture on fpga for multi-cluster nocs. *Electronics Letters*, 44(16):969–971.

[Fu et al. 2016] Fu, H. et al. (2016). The sunway taihulight supercomputer: System and applications. *SCIENCE CHINA Information Sciences*, 59(7):1–16.

[Gysi et al. 2015] Gysi, T., Grosser, T., and Hoefler, T. (2015). MODESTO: Data-centric analytic optimization of complex stencil programs on heterogeneous architectures. In *International Conference on Supercomputing (ICS)*, pages 177–186, Irvine, USA. ACM.

[Hascoët et al. 2017] Hascoët et al. (2017). Asynchronous one-sided communications and synchronizations for a clustered manycore processor. In *Proceedings of the 15th IEEE/ACM Symp. on Embedded Systems for Real-Time Multimedia - ESTIMedia '17*, pages 51–60, New York, New York, USA. ACM Press.

[Larus and Kozyrakis 2008] Larus, J. and Kozyrakis, C. (2008). Transactional memory. *Commun. ACM*, 51(7):80–88.

[Olofsson et al. 2014] Olofsson et al. (2014). Kickstarting high-performance energy-efficient manycore architectures with epiphany. In *Asilomar Conf. on Signals, Systems and Computers*, pages 1719–1726. IEEE.

[Penna et al. 2015] Penna, P. H., Castro, M., Freitas, H. C., Broquedis, F., and Méhaut, J.-F. (2015). Uma Metodologia Baseada em Simulação e Algoritmo Genético para Exploração de Estratégias de Escalonamento de Laços. In *WSCAD 2015 - Simpósio em Sistemas Computacionais de Alto Desempenho*, Florianópolis, Brazil. SBC.

[Pereira et al. 2015] 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. et al. 2017a] 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. et al. 2017b] 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.

[Rocha et al. 2017] Rocha, R. C. O., Pereira, A. D., Ramos, L., and Góes, L. F. W. (2017). TOAST: Automatic tiling for iterative stencil computations on GPUs. *Concurrency and Computation: Practice and Experience*, 29(8):1–13.

[Souza et al. 2016] Souza, M. A. et al. (2016). CAP bench: A benchmark suite for performance and energy evaluation of low-power many-core processors. *Concurrency and Computation: Practice and Experience*.

[Thorarensen et al. 2016] Thorarensen, S., Cuello, R., Kessler, C., Li, L., and Barry, B. (2016). Efficient execution of skepu skeleton programs on the low-power multicore processor myriad2. In *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 398–402.

[Totoni et al. 2012] Totoni, E., Behzad, B., Ghike, S., and Torrellas, J. (2012). Comparing the power and performance of intel's SCC to state-of-the-art CPUs and GPUs. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 78–87, New Brunswick, Canada. IEEE Computer Society.