Proposta de conversão automática em hardware de instruções para execução em memória

Rodrigo Machniewicz Sokulski[1], **Marco Antonio Zanata Alves**[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.011 – 81.531-990 – Curitiba – PR – Brazil

{rmsokulski,mazalves}@inf.ufpr.br

Resumo. *Uma das possíveis soluções para problemas como o Memory wall e o Gargalo de von Neumann consiste na adição de unidades de processamento próximas à memória, técnica denominada Processing-In-Memory (PIM). Este documento apresenta nossa proposta de trabalho, uma extensão de hardware para a conversão dinâmica de instruções para o processamento vetorial em memória.*

1. Proposta

Grande parte das arquiteturas atuais é baseada no modelo de von Neumann. Esse modelo mantém unidades de processamento e armazenamento independentes, comunicando-se através de uma interconexão. Isso gera uma dependência constante do tráfego de dados por essa interconexão, o que é conhecido como Gargalo de von Neumann [Talati et al. 2019]. A disparidade entre as velocidades de processamento e acesso aos dados em memória também é um problema para os sistemas, sendo denominado Memory Wall [Wulf and McKee 1995]. Além disso, nos últimos anos, a tendência de melhoria no desempenho de cada *thread* de algumas das principais linhas de procesadores para desktops da Intel e AMD superou 10% ao ano [Lechner 2020], enquanto a latência de fornecimento de dados pelas memórias permaneceu praticamente constante [K. Chang 2017], agravando esses problemas ainda mais. Uma alternativa para a redução desses problemas é o uso de técnicas PIM. Elas consistem na adição de elementos secundários de processamento próximos à memória principal, reduzindo o tráfego de dados entre esse componente e a Central Processing Unit (CPU) [Santos et al. 2021].

Para seu uso, mecanismos PIM exigem a inclusão de instruções especiais, inseridas por meio de funções *intrinsic* e bibliotecas, otimizações do compilador ou camadas de software ou hardware durante a execução. Dentre essas alternativas, apesar de mais restrita, a conversão por hardware é transparente ao desenvolvedor, evitando a necessidade de novos paradigmas de programação, alterações ou recompilações de softwares antigos ou proprietários. Além disso, por não exigir camadas adicionais de software, reduz potenciais sobrecustos durante a execução. Apesar dessas vantagens, todos os dispositivos propostos com apenas alterações em hardware são somente capazes de simples conversões entre instruções escalares da CPU para instruções escalares PIM. Porém, o uso de instruções PIM escalares é pouco eficiente, o que pode ser contornado pelo uso de instruções de memória vetoriais, que além de exigirem um menor tráfego de dados pelo barramento entre a CPU e a memória principal, utilizam de forma mais eficiente a alta vazão das memórias, fornecendo maiores ganhos energéticos e de desempenho [Cordeiro 2020]. Além disso, até onde sabemos, os mecanismos propostos em

hardware, capazes de conversões mais complexas, dependem de adaptações no código da aplicação, inseridas por desenvolvedores ou compiladores adaptados, eliminando grande parte das vantagens do uso de extensões de hardware.

Nosso trabalho consiste na proposta de um mecanismo de tradução binária em hardware para a conversão em *runtime* de instruções de aplicações para instruções vetoriais PIM, independentemente de qualquer alteração ou anotação de código. Para isso, nosso mecanismo deve identificar padrões predefinidos de sequências de instruções *memory-bound*, em laços, durante a execução da aplicação, as convertendo para instruções vetoriais PIM. Esse procedimento implica na vetorização para PIM de cadeias de instruções, compostas de *loads*, operações e *stores*, mantendo a semântica da aplicação e as exceções precisas do sistema. Já as demais instruções da aplicação serão executadas na CPU, devido à sua maior velocidade para a códigos CPU-*bound*. Para as avaliações serão realizadas simulações, evitando assim os custos necessários para o desenvolvimento de um protótipo real. Com isso, esperamos reduzir os custos energéticos e aumentar o desempenho de aplicações *memory-bound* de forma automática, sem prejuízos ao desempenho das demais aplicações.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001 e com apoio do Instituto Serrapilheira (número de concessão Serra-1709-16621).

Referências

Cordeiro, A. S. (2020). Porting machine learning algorithms to vector-in-memory architecture. Master's thesis, Pós-Graduação em Informática - Universidade Federal do Paraná, Curitiba - PR.

K. Chang, K. (2017). *Understanding and Improving the Latency of DRAM-Based Memory Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh - USA.

Lechner, M. (2020). Evolution of single-threaded x86 cpu performance.

Santos, P. C., Moreira, F. B., Cordeiro, A. S., Santos, S. R., Kepe, T. R., Carro, L., and Alves, M. A. Z. (2021). Survey on near-data processing: Applications and architectures. *Journal of Integrated Circuits and Systems*, 16(2):1–17.

Talati, N., Ben-Hur, R., Wald, N., Haj-Ali, A., Reuben, J., and Kvatinsky, S. (2019). mMPU—a real processing-in-memory architecture to combat the von neumann bottleneck. In *Applications of Emerging Memory Technology*, pages 191–213. Springer Singapore.

Wulf, W. A. and McKee, S. A. (1995). Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24.