The Impact of Memory Subsystem Resource Sharing on Datacenter Applications

TANG, L.; MARS, J.; VACHHARAJANI, N.; HUNDT, R.; SOFFA, M. L. The impact of memory subsystem resource sharing on datacenter applications. In: International Symposium on Computer Architecture, 38, San Jose, California, USA, 2011.

Eliana Alves Moreira – RA 120437

O artigo apresenta estudos realizados para analisar os impactos do compartilhamento de recursos do subsistema de memória entre diferentes aplicações (três sensíveis à latência e duas *batch*) do *datacenter* da Google, envolvendo vários tipos de cenário de mapeamento *thread-to-core* (TCC) em uma plataforma primária *dual-socket Intel Xeon* E5345 (Clovertown). Em plataformas *multisocket multicore*, os núcleos de processamento podem ou não compartilhar recursos de memória, incluindo o último nível de cache (LLC) e largura de banda de memória. No entanto, os autores afirmam que, conforme já citado em trabalhos anteriores, é necessário reduzir a interferência causada no desempenho por aplicações de menor prioridade sobre aplicações de alta prioridade. Segundo os autores, *datacenters* modernos tem atribuído *threads* a núcleos de uma forma *ad-hoc*. O aumento de aplicações de *datacenter* traz à tona a importância de se compreender como é realizada a interação destas aplicações com a arquitetura de compartilhamento de recursos de memória da máquina utilizada. A caracterização das interações pode fornecer informações para novos projetos de arquiteturas para este tipo de cargas de trabalho, pois, na escala do *datacenter*, uma melhoria no desempenho de 1% em aplicações-chave pode resultar em milhões de dólares economizados.

Assim, os autores realizaram experimentos nas aplicações sensíveis à latência para analisar a variabilidade do desempenho quando a aplicação é executada sozinha ou conjuntamente com uma das aplicações *batch*. Observou-se que, se, ao executar sozinha, uma aplicação tem melhor desempenho usando LLC´s separadas e FSB´s separados, depois da transição para compartilhamento LLC existe um aumento de faltas LLC. Isto indica que a disputa de LLC ocorre entre as *threads*. O consumo de largura de banda do barramento é consistente com as faltas de LLC e tendências de desempenho, pois o desempenho destas aplicações é pior no cenário de mapeamento em que o FSB é compartilhado, devido à sua disputa. Quando executada simultaneamente com outras aplicações, a aplicação tem suas preferências alteradas dependendo da aplicação co-executante. Na aplicação sendo executada sozinha em que o desempenho é melhor ao compartilhar 2 LLC´s e um FSB, depois da transição para compartilhamento LLC existe uma diminuição nas faltas LLC. Isto indica que o compartilhamento de cache é construtivo e que *threads* estão compartilhando dados que cabem na LLC. Esta aplicação, ao executar simultaneamente com outras aplicações, mantém sua preferência pelo compartilhamento, apesar de existir oscilação de desempenho. Os experimentos mostraram também que as aplicações sensíveis à latência com média/alta demanda do barramento preferem compartilhar FSB com aplicações *batch* que tem menor demanda de barramento, para não haver degradação no desempenho. Observou-se, ainda, que aplicações com maiores taxas de falta LLC sofrem mais com a disputa de cache.

Os autores propuseram um algoritmo heurístico para prever um mapeamento *thread-to-core*. A ideia básica é caracterizar as aplicações co-alocadas, priorizando-se o desempenho de aplicações sensíveis à latência, e ainda evitar o compartilhamento de seus potenciais gargalos, maximizando o benefício. Para avaliar o algoritmo comparou-se o melhor mapeamento previsto com o melhor mapeamento baseado na verdade. A abordagem prediz corretamente o melhor mapeamento em 4 dos 6 pares de co-execução, e nos outros 2 casos a diferença é inferior a 2% . Porém, a abordagem tem limitações: características devem ser coletadas para cada aplicação e um novo algoritmo precisa ser gerado, porque cada arquitetura tem diferentes topologias e pontos de compartilhamentos.

Os autores construíram um mapeador *Thread-to-core* adaptativo (AToM), que usa uma heurística de competição para adaptativamente pesquisar pela ideal indicação *thread-to-core* para um dado conjunto de *threads*. Durante a fase de aprendizagem, AToM empiricamente coloca vários mapeamentos *thread-to-core* uns contra os outros para saber qual o mapeamento executa melhor. Durante a fase de execução, o vencedor do mapeamento *thread-to-core* é executado por um período fixo ou adaptado de tempo antes que outra competição seja realizada. AToM mostrou-se eficaz, alcançando um desempenho perto do ótimo. Em cada tipo de mapeamento, AToM supera o caso médio em até 22%, e tem significativamente um melhor desempenho do que as atribuições de pior caso.

Os resultados mostraram que o impacto no desempenho do compartilhamento de recursos de memória para as aplicações sensíveis à latência é significativo e que cada aplicação prefere diferentes configurações de compartilhamento. Ainda, oscilações do desempenho das aplicações entre o seu melhor e pior mapeamento TTC pode ser significante quando executam simultaneamente com outras aplicações. Estas características podem ser utilizadas para construir algoritmos heurísticos para realizar mapeamentos *thread-to-core*, bem como uma abordagem adaptativa. Os autores concluíram que esta traz melhores resultados do que a primeira em relação à melhor utilização do compartilhamento dos recursos disponíveis.