MO401 – Trabalho 1

Maxiwell Salvador Garcia (089057)

Artigo: Francis Tseng, Yale N. Patt. *Achieving Out-of-Order Performance with Almost In-Order Complexity.* ISCA, pages 3-12, 2008. Beijing, China.

Uma das maneiras de se ganhar desempenho é escalonar instruções fora de ordem (*wider issue widths*) para aproveitar recursos ociosos do processador. Porém, isso aumenta a complexidade na construção do hardware, não é escalável e resulta em mais potência dissipada. O presente artigo introduz um mecanismo que detecta, em tempo de compilação, trechos de códigos que podem ser executados em paralelos, explorando o uso e o tempo de vida dos valores produzidos. Ao analisar o *Data Flow Graph* (DFG) do programa, é possível observar tais quesitos. A técnica consiste em dividir o DFG em várias unidades denominadas *braids,* que são facilmente processadas pelo *hardware*. Define-se *braid* como um subgrafo do DFG do programa que reside dentro de um bloco básico (*basic block*) e representa um caminho crítico. Uma operação pertencente a um caminho crítico espera por algum valor que está sendo processado por outra operação pertencente ao mesmo caminho crítico. Assim, vários *braids* podem ser gerados de um bloco básico, um para cada caminho.

Três componentes são importantes para implementar o processamento baseado em *braids*: (i) uma compilação que consiga construir *braids* a partir do DGF e gerar instruções com algumas informações importantes; (ii) um *Instruction Set Architecture* (ISA) que converta essas informações ao hardware; e (iii) uma microarquitetura preparada para explorar todas as característica dos *braids*. Para a compilação, usou-se um conjunto de ferramentas, sendo a *profiling tool* a responsável por analisar o DFG e armazena o tempo de vida de cada valor, bem como declarações e usos, permitindo que um algoritmo de coloração de grafos identifique os *braids* dentro de cada bloco básico. Todas as instruções pertencerá a algum *braid* e todas as instruções de um mesmo *braid* devem ser consecutivas. Por isso, as instruções são organizadas para atender essa restrição por um *binary translation tool*. Se a última instrução for um *branch,* o *braid* contendo-a deve ser ordenado para ser o último do bloco básico. Para a alocação de registradores, utiliza-se as informações coletadas com a primeira ferramenta, descobrindo quais valores são usados dentro e fora do bloco básico. Isso se torna útil pois a microarquitetura implementada possui um banco de registradores externo e um interno à unidade de processamento de *braids*, ambos com poucas entradas (8 bytes). Se um *braid* exceder a quantidade de valores internos (fato observado em apenas 2% dos *braids*), ele dividido em dois. Para resolver dependências verdadeiras, tenta-se colocar o *braid* em uma posição no bloco básico que satisfaça a ordem original de *store-load.* Caso não seja possível, ele é quebrado em dois e reorganizado para satisfazer a restrição (fato observado em apenas 1% dos *braids*).

A ISA sofreu poucas e simples modificações para oferecer suporte a *braids*, consistindo basicamente em inserir bits sinalizadores nas instruções. Um bit sinalizador S foi inserido no início das instruções para indicar quando um novo *braid* começa, *i.e.* a primeira instrução de um *braid* possui S verdadeiro. Um bit T foi associado a cada operando para indicar se o valor está em registrador interno ou externo. Dois bits, I e E, indicam se o destino será salvo em registrador interno, externo, ou em ambos. A microarquitetura desenvolvida difere em alguns pontos da convencional *out-of-order.* O alocador, ao analisar os *braids,* verifica quais resultados serão escritos em registradores internos, pois assim, não precisa alocá-los em registradores externos. Na renomeação, os operandos internos tem uma atenção especial, pois eles não precisam ser renomeados – na compilação isso foi resolvido pra cada *braid.* Seguindo, as instruções pertencentes a um mesmo *braid* são distribuídas para um dos *braids execution units* (BEUs). Um BEU só pode aceitar um novo *braid* caso não esteja processando nenhum outro. As principais estruturas dentro de cada BEU são: (i) uma fila (FIFO) para fazer o escalonamento das instruções em ordem e duas por vez (*two-wide in-order scheduler*); (ii) duas unidades funcionais; (iii) um banco com 8 registradores que suporta acessos simultâneos das duas unidades; e (iv) uma rede de comunicação com *buffer* (*bypass*) para salvar nos registradores externos, permitindo que um dado seja buscado no *buffer*, antes de tê-lo em um registrador externo. Como os valores dos registradores internos são de uso restrito do *braid* em execução (temporários), eles podem ser eliminados após a última instrução do *braid*. Isso simplifica, também, o processo de recuperação de um *misprediction*, não precisando armazenar os registradores internos no *checkpoint.* A configuração padrão possui 8 BEUs, ou seja, 16 unidades funcionais. Porém, devido à baixa complexidade das estruturas utilizadas, 8 BEU ocupa menos espaço que uma convencional arquitetura *8-wide out-of-order* (8 unidades funcionais).

Para os experimentos, várias configurações foram testadas, inclusive com arquiteturas convencionais. Ao variar o número de registradores de um *8-wide out-of-order* de 32 para 16, houve 21% de degradação da performance; enquanto que na microarquitetura *braid*, ao variar de 16 para 8 registradores externos, a performance não foi significativamente afetada. Ao variar o número de unidades funcionais por BEU (de 2 para 4 e 8) e o tamanho da janela de escalonamento das filas (de 2 para 4 e 8) não houve melhora relevante, confirmando que o paralelismo de um *braid* é limitado em duas instruções. O penúltimo teste avalia vários paradigmas de microarquitetura, e mostra que o uso de técnicas *wider widths* ainda consegue maior performance e a microarquitetura *braid* alcançou, na média, uma performance 9% inferior à convencional arquitetura agressiva 8-*wide out-of-order.* Por fim, utiliza-se 8 unidades funcionais para a microarquitetura *braid* – 4 BEUs com 2 unidades cada, e 8 BEUs com uma unidade cada – chegando à conclusão que, na maioria dos casos, a segundo opção é mais vantajosa, pois mais *braids* são escalonados por vez.

A execução de *braids* atingiu uma performance próxima da arquitetura *out-of-order* convencional, com uma complexidade de hardware perto da implementação *in-order*. A compilação resolve grande parte dos problemas, descomplicando na construção da microarquitetura. Poucas e simples estruturas reduz a área do circuito e diminui a potência dissipada, permitindo um *clock* muito mais alto, porém, este fator não foi avaliado neste artigo. Para melhorar os resultados, um compilador específico poderia ser criado, deixando mais eficiente a manipulação dos registradores e técnicas de Clustering poderia ser implementada com *braids*, agrupando um subconjunto de BEUs.