Aluno: Fernando André Kronbauer **RA:** 057619

Artigo: Guilherme Ottoni, Ram Rangan, Adam Stoler, David I. August. *Automatic Thread Extraction with Decoupled Software Pipelining*. pp. 105 - 118; MICRO 2005

O artigo apresenta a técnica *Decoupled Software Pipelining* – DSWP. Trata-se de uma técnica de compilação que extrai paralelismo não especulativo dos laços de um programa, de forma que diferentes partes do laço possam ser executados paralelamente em diferentes núcleos de processamento, acelerando desta forma a execução do programa como um todo. É proposto no artigo um método completamente automático de extração de paralelismo em nível de linha de execução, no qual o paralelismo altamente granular inerente à maioria dos programas é extraído e transformado em linhas de execução paralelas de longa duração.

O artigo lembra outra técnica de extração de paralelismo, chamada DOACROSS, e a compara a DSWP. O paralelismo em DOACROSS é obtido pela execução concorrente de partes de cada iteração do laço, distribuídos em vários núcleos de processamento. As dependências são respeitadas pelo repasse de valores de um núcleo para outro, geralmente através de memória com sincronização. É apresentado o exemplo no qual uma lista ligada é percorrida, realizando-se algum processamento em cada nó. O carregamento do ponteiro para o próximo nó da lista precisa ser repassado de um núcleo de processamento para outro a cada iteração do laço. Enquanto o método DOACROSS sobrepõe a execução do corpo da iteração corrente com o carregamento do ponteiro do nó da próxima iteração, os custos de comunicação neste caso podem simplesmente anular quaisquer ganhos de paralelismo.

Em DSWP, em vez de colocar cada iteração do laço em um núcleo, quebra-se o corpo do laço em um número de partes igual ao número de núcleos de processamento disponíveis, de forma que os núcleos formem um *pipeline*, ou seja, valores produzidos por um núcleo de processamento servem como entrada para o núcleo seguinte no *pipeline*. Tomando novamente o exemplo da lista ligada, a parte de carregamento do ponteiro do próximo nó é realizada em um núcleo enquanto que o processamento do nó é realizado em outro. Desta forma a dependência crítica do laço não precisa mais ser roteada de um núcleo para outro. A transmissão dos valores de um núcleo para outro é feita em uma fila de mensagens implementada em *hardware*, com baixo custo de sincronização.

Ao contrário das técnicas DOACROS e outras técnicas de extração de paralelismo desenvolvidas anteriormente, DSWP requer que o fluxo de dados entre um núcleo e outro seja acíclico. Isto cria a oportunidade para que o processamento de cada núcleo seja o mais independente possível dos demais, aumentando a tolerância à latência de comunicação. Técnicas DOACROS acabam sendo mais restritivas do que DSWP, pois geralmente requerem que as iterações do laço sejam contadas, que operem somente em vetores, que tenham padrões simples de acesso à memória ou que tenham fluxo de controle simples ou inexistente.

O artigo segue apresentando o algoritmo do DSWP, aplicado no código assembler gerado pelo compilador, mas antes do processo de alocação de registradores. Este algoritmo constrói o grafo de dependência das instruções que compõe o laço, e identifica os seus componentes fortemente conectados. Estes componentes são utilizados pelo algoritmo para construir as partições que serão designadas a cada núcleo, de forma a garantir que o fluxo de dados seja acíclico entre as partições, e tentando balancear as cargas de cada núcleo através de uma heurística (o problema de balanceamento de cargas dos núcleos é NP-completo). Duas novas instruções são acrescentadas ao ISA: **produce** e **consume**. **produce** recebe como parâmetro o número da fila que receberá a mensagem, mais a mensagem em si, do tamanho de uma palavra do computador. **consume** por sua vez recebe como parâmetro o número da fila de onde retirará a mensagem ali depositada por uma instrução **produce**. Após a partição do corpo do laço, o compilador insere instruções **produce** e **consume** nas posições adequadas de forma a garantir que as dependências de dados, de controle e de sincronização de memória sejam satisfeitas.

Cada partição do laço que irá executar em paralelo é transformada pelo compilador em uma função auxiliar. O compilador então injeta código no início programa para que uma linha de execução auxiliar seja instanciada e execute estas funções no momento apropriado. Esta linha de execução executa um laço principal, onde fica aguardando em uma fila mestre de mensagens o valor do ponteiro da função que irá executar em seguida. A linha de execução principal, ao chegar ao laço otimizado por DSWP, executa uma operação **produce** passando como parâmetro a fila mestre e o endereço da função auxiliar a ser executada pela linha de execução auxiliar.

Os resultados apresentados pelo artigo foram produzidos através da adaptação do compilador IMPACT e simulação do código em um simulador validado de um processador Itanium 2. Devido a natureza altamente detalhada do simulador, os autores do artigo não conseguiram simular o código completo dos programas gerados. Em vez disso, as simulações detalhadas foram restritas aos laços otimizados pelo DSWP. Os resultados experimentais mostraram que DSWP pode ser aplicado a quase todos os laços de um programa. Utilizando um simulador de dois núcleos de processamento baseado no modelo validado de um núcleo do Itanium 2, e utilizando um compilador de otimização de código de alta qualidade, DSWP alcançou *speedup* médio de aproximadamente 19,4% em loops importantes dos *benchmarks*, traduzido em *speedup* médio de 9,2% sobre *benchmarks* completos. Os autores concluem o artigo reconhecendo áreas nas quais o método pode ser melhorado, como análise de memória mais adequada, otimizações adicionais para quebrar ciclos de dependência, heurísticas de particionamento mais elaboradas, e novas técnicas de otimização para reduzir o número de fluxos entre núcleos.