Título do Artigo: Compiler Optimizations for Transaction Processing Workloads on Itanium® Linux Systems

Citação Bibliográfica do Artigo: Gerolf Hoflehner, Knud Kirkegaard, Rod Skinner,Daniel Lavery, Yong-fong Lee, Wei Li. Compiler Optimizations for Transaction Processing Workloads on Itanium® Linux Systems; pp. 294 - 303; 37th Annual International Symposium on Microarchitecture, 2004.

Aluno: Mário Luiz Rodrigues Oliveira

RA: 066254

O artigo analisado apresenta e discute algumas otimizações que contribuem para melhorar o desempenho do código gerado pelo compilador C/C++ da *Intel* para o processador *Itanium 2*. Especificamente, o artigo apresenta otimizações para aproveitar a pilha de registradores do processador *Itanium* e propõe alterações no modelo de preempção do Linux.

O uso, em conjunto, de todas as otimizações apresentadas permitiram uma melhoria de mais de 40 % em aplicações envolvendo processamento de transações *on-line* (OLTP),utilizando-se como *workloads* a base de dados do *Oracle* e um sistema composto por 4 processadores *Itanium 2* executando o sistema operacional Linux.

O desempenho das aplicações OLTP no processador *Itanium 2* é prejudicado por *misses* na cache de dados, na cache de instruções e na ITLB e pelo tráfego de dados entre a memória e o *register stack engine (RSE)*. Assim, o artigo apresenta otimizações que tentam melhorar tais aspectos.

As otimizações descritas são:

- Redução do Tráfego RSE:o processador *Itanium* possui 128 registradores e desses 96 são usados para a chamada e retorno de procedimentos. A arquitetura do processador *Itanium* permite encolher a pilha de registadores antes de uma chamada de procedimento e restaurar o seu tamanho original mais tarde. Essa característica pode reduzir o número total de registadores consumidos pelo procedimento chamador e pelo procedimento chamado. Assim pode-se diminuir o tráfego de RSE.

- Escalonamento de código e especulação de controle: para tentar reduzir *misses* na cache de instruções pode-se usar as técnicas de especulação de controle (mover as instruções de *load* para antes das instruções de *branches*) e escalonamento de código (reordenar instruções para tentar minimizar os efeitos de *misses* na cache).

- Prefetching de instrução: também para tentar minimizar *misses* na cache de instruções pode-se buscar antecipadamente instruções que são alvo de uma instrução de desvio.

- Otimizações da disposição das funções: nessa categoria pode-se incluir 3 otimizações, a saber: funções *inline*, agrupamento de funções e separação de funções. Funções *inline* removem o *overhead*de chamada de funções, agrupamento de funções permitem reduzir *misses* na cache de instruções e separação de funções diminuem os *misses* em ITLB.

- Otimizações da disposição dos dados: duas otimizações são aplicadas pelo compilador nesse item. Primeiro, constantes e *strings* devem ser mantidas

numa área de somente leitura ao invés de ficar na seção de dados. E segundo, o compilador implementa heurísticas que tentam ordenar os dados locais na pilha baseado na freqüência e no tamanho.

- Otimizações Setjmp()/longjmp(): essas otimizações visam reduzir o *overhead* na chamada das funções setjmp()/longjmp().
- Modelo de preempção do Linux: essa otimização tem por objetivo diminuir o *overhead* em tempo de exeução causado pelo modelo genérico de preempção do Linux que requer do compilador a geração de código realocável.