Cálculo do número de instruções dos *benchmarks* e implementação de novas *pin tools*

Gustavo Ciotto Pinton

Universidade Estadual de Campinas - UNICAMP
MO601B - Arquitetura de Computadores

19 de Setembro de 2016

Contagem de instruções dos *benchmarks*

Escolha da *pin tool*

- *Tools* disponíveis para a contagem de instruções:
  - `inscount0`: Nenhuma otimização. Funções de rotina inseridas a cada instrução.
  - `inscount1`: medida de granularidade é o BBL (*basic block*), economizando diversas chamadas à função de análise.
  - `inscount2`: BBL, inserção `IPOINT_ANYWHERE` e `PIN_FAST_ANALYSIS_CALL`;
  - `inscount_tls`: BBL, inserção `IPOINT_ANYWHERE`, `PIN_FAST_ANALYSIS_CALL` e unidade de armazenamento rápido, `TLS`, indexado pelos *ids* das threads.

- Melhor performance para o nosso caso: `inscount2`!
  - Não são necessárias informações sobre o número de *threads*;
  - Aproximadamente 20 horas para um Intel i3.

Contagem de instruções dos *benchmarks*

Resultados

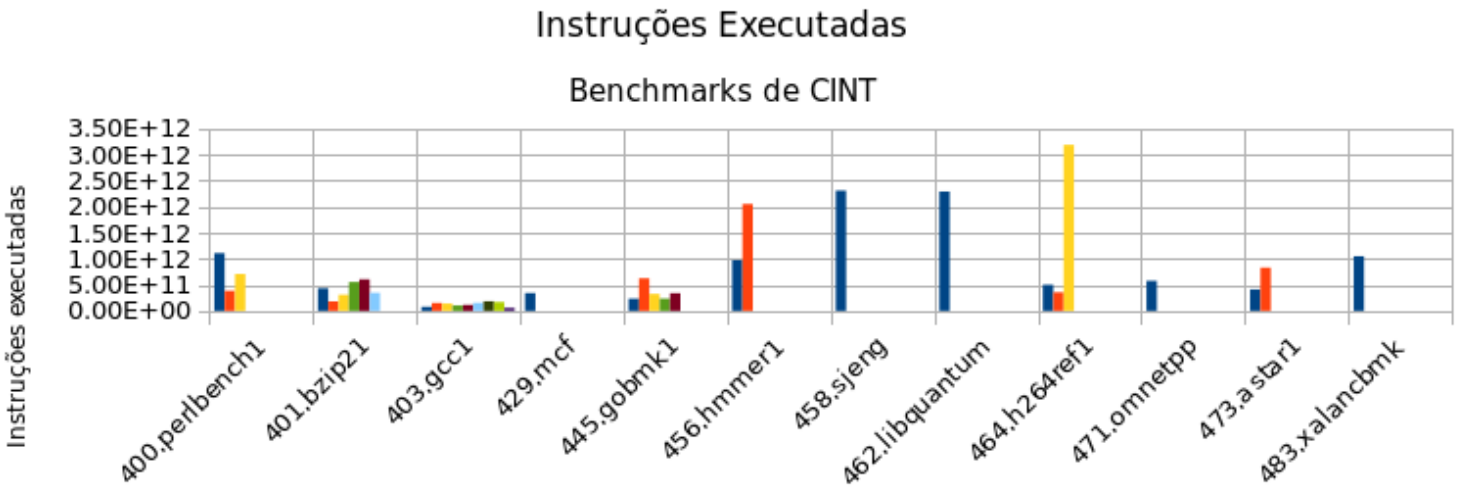

Figura: Número de instruções dos *benchmarks* pertencentes ao conjunto `CINT`.

Contagem de instruções dos *benchmarks*

Resultados

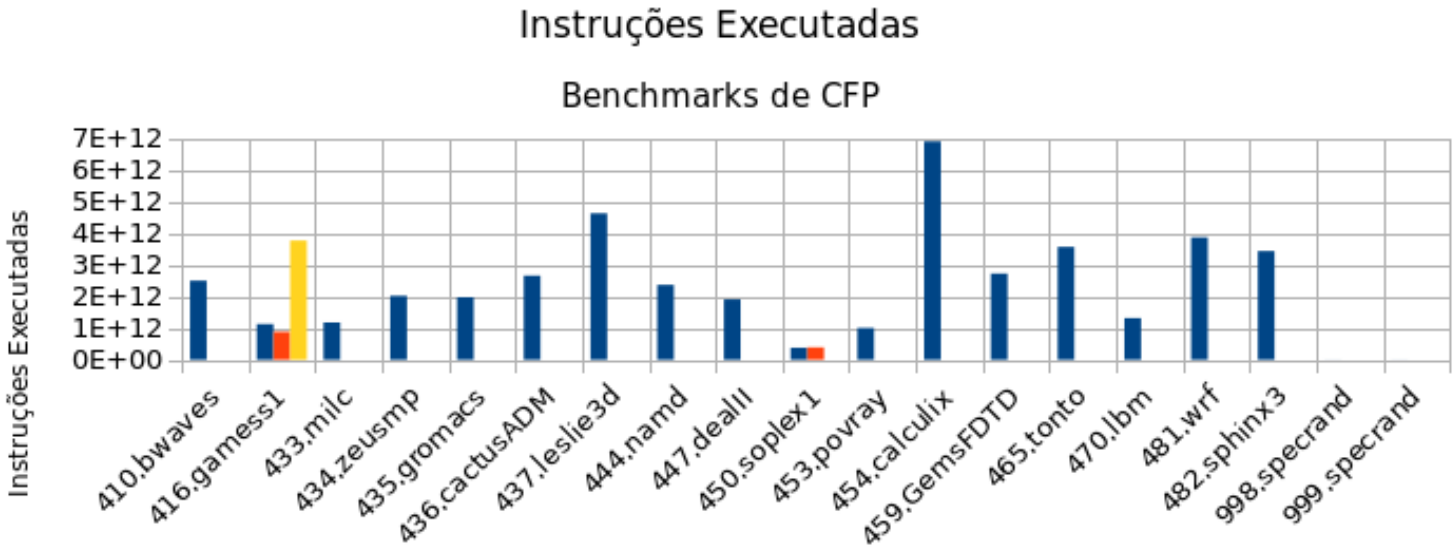

Figura: Número de instruções dos *benchmarks* pertencentes ao conjunto `CFP`.

Contagem de instruções dos *benchmarks*

Resultados

- `403.gcc` utiliza mais entradas em seus testes, com 9 entradas, totalizando 1.152.387.847.873 instruções executadas;
- `454.calculix` : maior número de instruções com 6.894.341.859.256;
- `998.specrand` e `999.specrand`: mesmo número de instruções (536.611.748 instruções cada);
- Em geral, os *benchmarks* do conjunto inteiro executam menos instruções que os do conjunto de ponto flutuante.

Implementação de novas *pin tools*

- Contagem do número de instruções por rotina e por *thread*;
  - Função de instrumentação: a cada **rotina**;
  - Função de análise: a cada **instrução**.
  - 2 listas ligadas: 1 de rotinas e outra de *threads*:
  - Cada nó de rotina contém *nome* (`RTN_Name`), *id* (`RTN_Id`), o primeiro nó de uma lista de *threads* e um ponteiro para o próximo nó;
  - Cada nó de *thread* contém a sua *id* (`THREAD_ID`), o número de instruções executadas da respectiva rotina e um ponteiro para o próximo nó;
  - Programas com muitas *threads*: forte impacto negativo à performance, uma vez que um nó de *thread* é procurado a cada instrução (rotina de análise).

Implementação de novas *pin tools*

- Simulação de dois modelos de *branch prediction* discutidos em aula (1 e 2 *bits*).
  - Função de instrumentação: a cada **instrução**;
  - Função de análise: a cada **instrução** de *branch* ou chamada de rotina.
  - Função *hash*: 12 *bits* menos significativos;
  - Modelo de *1 bit*: apenas o último salto é considerado;
  - Modelo de *2 bits*: máquina de estados com 4 estados (`STRONG_TAKEN`, `WEAK_TAKEN`, `STRONG_NOT_TAKEN` e `WEAK_NOT_TAKEN`).
- Testes no conjunto `ref` dos *benchmarks* `400.perlbench`, `401.bzip2`, `403.gcc`, `445.gobmk` e `999.specrand`

Implementação de novas *pin tools*

Resultados

- Contagem do número de instruções por rotina e por *thread*:
  - *Benchmarks* **testados** apresentam apenas uma *thread*: performance não é *extremamente* degradada;

Tabela: Resultados obtidos para algumas rotinas importantes.

Benchmark	*E*	*Rotina*	%
`400.perlbench`	3	`S_regmatch`	50
`401.bzip2`	6	`BZ2_compressBlock`	22.2
`403.gcc`	9	`bitmap_operation`	9.2
`445.gobmk`	5	`do_play_move`	8.4
`999.specrand`	1	`GI___printf_fp`	32.6

Implementação de novas *pin tools*

Resultados

- Simulação de dois modelos de *branch prediction* discutidos em aula (1 e 2 *bits*).

Tabela: Resultados obtidos para os modelos *1-bit* e *2-bit*.

		Modelo 1			**Modelo 2**		
Benchmark	*E*	*Misses*	*Hits*	%	*Misses*	*Hits*	%
`400.perlbench`	3	4.0×10^{10}	9.4×10^{10}	0.70	2.7×10^{10}	1.1×10^{11}	0.80
`401.bzip2`	6	9.1×10^{9}	5.4×10^{10}	0.86	1.3×10^{10}	5.0×10^{10}	0. 79
`403.gcc`	9	4.7×10^{9}	2.3×10^{10}	0.83	4.9×10^{9}	2.3×10^{10}	0. 82
`445.gobmk`	5	2.0×10^{10}	4.3×10^{10}	0.68	1.8×10^{10}	4.5×10^{10}	0.72
`999.specrand`	1	3.9×10^{7}	6.4×10^{7}	0.62	4.7×10^{7}	5.9×10^{7}	0.58