Contagem do número de instruções dos *benchmarks* do SPEC CPU2006 e implementação de uma *pintool*

Gustavo Ciotto Pinton[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Av. Albert Einstein, 1251, Cidade Universitária, Campinas/SP
Brasil, CEP 13083-852
Fone: [19] 3521-5838

ra117136@unicamp.br

Abstract. *This report describes the number of instructions executed by each benchmark in SPEC CPU2006. It was achieved thanks to the Intel's pin application. All results were obtained by running the benchmarks with the ref inputs and a single iteration. A new pin tool was equally developed, which lists the number of executed instructions of every routine of all threads that compose the program. Five benchmarks, run with inputs belonging to the test set, were used to test this new tool.*

Resumo. *Este relatório apresenta o número de instruções executadas por cada benchmark presente no SPEC CPU2006, calculado através da utilização da ferramenta pin, desenvolvida pela Intel. Os valores encontrados correspondem às entradas do tipo ref e a uma única iteração. Além disso, desenvolveu-se uma nova pin tool, cuja saída é a uma lista com o número de instruções executas pelas rotinas de um programa, separadas pela sua respectiva thread. Tal ferramenta foi testada em 5 benchmarks na configuração test.*

1. Introdução

SPEC, do inglês *Standard Performance Evaluation Corporation*, é uma organização constituída por fabricantes de *hardware*, *software* e instuições de pesquisa, cujo objetivo é definir uma série de testes relevantes e padronizados para a análise da performance de um computador. Tais testes podem ser igualmente denominados de *benchmarks* e visam avalisar um aspecto específico de processamento. Durante a aula, vimos que existe uma grande variedade de *benchmarks* disponíveis na *internet*, sendo que alguns foram implementados, por exemplo, para avaliar o desempenho de aplicações *web*. Um *benchmark* realiza um conjunto de operações definidos, chamado de **workload**, e produz um resultado, ou seja, uma **métrica** que tenta avaliar o desempenho do computador submetido ao respectivo *workload*. Tendo em vista tais conceitos, o SPEC CPU2006 é um conjunto de 31 *benchmarks* que procuram avaliar a performace de três componentes principais, sendo eles o processador, a arquitetura de memória e os compiladores. Os *benchmarks* são distribuídos em dois *suites*, denominados de CINT2006 e CFP2006, que se distinguem quanto à natureza do seu processamento intensivo: o primeiro é focado na performance das operações que utilizam números **inteiros**, sendo que o segundo, de números em **ponto flutuante**. Essencialmente, SPEC CPU2006 oferece duas métricas, **speed** e **rate** (ou **throughput**), medindo, respectivamente, o quão rápido um computador completa uma

única tarefa e quantas tarefas tal sistema pode realizar em um pequeno período de tempo. Cada métrica possui, por sua vez, quatro *variações*, que se diferenciam quanto ao *suite* (inteiro ou ponto flutuante) e ao método de compilação. Em relação a este último, duas opções são disponibilizadas: `base`, que apresenta requisições mais estritas, isto é, as *flags* de compilação devem ser usadas na mesma ordem para todos os *benchmarks* de uma dada linguagem e é exigida para um teste *reportable* (execução que pode ser publicada), e `peak`, opcional e com menos exigências.

`PIN`, por sua vez, é uma ferramenta para instrumentação e análise de programas, à medida que ela permite a inserção de código dinamicamente ao executável. Esta ferramenta é capaz de interceptar a execução da primeira instrução e gerar um novo código a partir dela. Uma `pintool` pode ser definida como uma extensão do processo de geração de código realizada pelo `pin`, já que é capaz de interagir com este último e comunicar quais funções, ou *callbacks*, o aplicativo deve inserir ao código. De maneira geral, uma `pintool` é composta por dois componentes, chamados de *instrumentation code* e *analysis code*. O primeiro deve decidir onde inserir o novo código, isto é, em que locais as rotinas de análise deverão ser lançadas. É nessa fase, portanto, que características **estáticas** do código, tais como, por exemplo nome de rotinas ou número de instruções que as compõem, devem ser exploradas. O segundo, por sua vez, é chamado à medida que o código é executado e, dessa forma, pode afetar significamente a performance de um executável se o determinado código apresentar complexidade elevada.

Neste relatório, será abordada, na primeira parte, o uso de uma `pintool` para a avaliação do número de instruções executadas por cada um dos *benchmarks* e, em seguida, a implementação de uma nova ferramenta capaz de determinar quantas instruções cada rotina de cada *thread* foram executadas.

2. Contagem das instruções

A fim de calcular as instruções de cada *benchmark* do `SPEC CPU2006`, dois *scripts bash* foram implementados. O primeiro, chamado de `run-pintool-all-benchmarks.sh` verifica e executa o *runspec*, escrito em *perl*, para cada um dos *benchmarks*, além de compilar a *pintool* que será utilizada. Tal *script* comunica alguns parâmetros importantes ao comando *runspec*, tais como o método de compilação (`base`), o conjunto de entradas que será transmitido aos programas (`ref`, no nossa caso), o número de iterações (1) e, evidentemente, o nome do *benchmark*. A execução deste comando produz um arquivo de extensão `.tmp.log` no diretório do projeto, que é copiado posteriormente a uma pasta, cujo nome é igual ao do *benchmark* que acabou de ser executado. O arquivo de configuração transmitido ao comando `runspec` faz referência ao segundo *script*, `run-spec-command.sh`, responsável por relacionar o aplicativo `pin` com o respectivo *benchmark*. Este *script* recebe como parâmetro o comando que o `SPEC` utiliza e o retransmite para o `pin`, que é responsável por executá-lo efetivamente.

No manual de referência do `pin`, são indicadas quatro maneiras distintas de se calcular o número de instruções executadas por um programa. A primeira, `inscount0`, insere a rotina de análise antes de cada instrução, produzindo, assim, uma grande perda de performance. A segunda, `inscount1`, é superior à anterior, à medida que utiliza uma outra medida de granularidade, chamado de `BBL` (do inglês, *basic block*) e, por-

tanto, economiza diversas chamadas à função de análise. A terceira rotina, chamada de `inscount2`, usa o mesmo princípio que a anterior, porém apresenta melhor desempenho, visto que faz uso de dois recursos a mais que `inscount1`. O primeiro recurso é a mudança de `IPOINT_BEFORE` para `IPOINT_ANYWHERE`, que autoriza o `pin` escolher em que ordem a função de análise é colocada, permitindo, assim, que ele escolha o ponto que requeira mínimas operações de salvamento e restaturação dos registradores. Além disso, esta ferramenta também usa a opção de *fast call linkage*, que explora o fato de que alguns compiladores podem eliminar o *overhead* que, para funções pequenas como a a função de análise, é comparável ao próprio conjunto de operações da respectiva função. Esta opção é ativada através do uso de `PIN_FAST_ANALYSIS_CALL`. Por fim, o último programa utiliza, além dos recursos comentados anteriormente, uma unidade de armazenamento rápido, chamado de `TLS`, indexado pelos *indexes* das *threads* para gerar o número de instruções por *thread*.

Tendo em mente estas características, escolhe-se o programa **`inscount2`** para o cálculo do número de instruções, visto que este último apresenta o maior número de otimizações e que, neste contexto, não visa-se encontrar uma contagem por *threads*, mas sim global. Os *benchmarks* foram rodados em apenas um *core* do processador *Intel i3* e levaram, ao todo, 20 horas aproximadamente. Os resultados encontrados estão listados abaixo. Alguns *benchmarks* possuem mais de uma entrada e, portanto, produzem mais de uma saída. Tais saídas estão separadas por *vírgulas* na lista abaixo:

Para os *benchmarks* do conjunto **inteiro**:

- `400.perlbench`: 1109198045367, 384794966182, 707786630157
- `401.bzip2`: 432459815466, 181180973754, 309587458388, 553737482083, 605707320979, 345617094417
- `403.gcc`: 77489490228, 151321617974, 139512528163, 103537789803, 113969504551, 154631785799, 183463430994, 169999746216, 58461954145
- `429.mcf`: 341546845898
- `445.gobmk`: 234525674455, 625631596347, 325147399858, 235593319559, 339210226918
- `456.hmmer`: 971544939130, 2052353062674
- `458.sjeng`: 2309967978778
- `462.libquantum`: 2291912513237
- `464.h264ref`: 500835844419, 355915755256, 3188039031373
- `471.omnetpp`: 576122145483
- `473.astar`: 411719281496, 829803428219
- `483.xalancbmk`: 1048497777434

Para os *benchmarks* do conjunto em **ponto flutuante**:

- `410.bwaves`: 2.495.514.310.671
- `416.gamess`: 1124505753634, 878108843916, 3766238100560
- `433.milc`: 1175358805504
- `434.zeusmp`: 2016616007502
- `435.gromacs`: 1971200720706
- `436.cactusADM`: 2655006820074
- `437.leslie3d`: 4626149683423
- `444.namd`: 2361163844939

- `447.dealII`: 1903231296730
- `450.soplex`: 377431323949, 389903616072
- `453.povray`: 1002529177253
- `454.calculix`: 6894341894243
- `459.GemsFDTD`: 2722715227347
- `465.tonto`: 3563812458171
- `470.lbm`: 1314569317678
- `481.wrf`: 3867428098913
- `482.sphinx3`: 3432419178361
- `998.specrand`: 536611748
- `999.specrand`: 536611748

Verifica-se, portanto, que o *benchmark* que utiliza mais entradas em seus testes é `403.gcc`, totalizando 1.152.387.847.873 instruções executadas. Os *benchmarks* que rodaram mais e menos instruções foram, respectivamente, `454.calculix` e `998.specrand`/`999.specrand`.

3. Implementação de uma nova *pin tool*

Tendo em vista que os programas apresentados na seção anterior não permitem a contagem de instruções por rotina e por *thread*, propõe-se a implementação de uma nova *pintool* contendo estas duas funcionalidades. Para isso, utiliza-se a API da ferramenta para a adição de funções de *instrumentação* para cada routina e de *análise* para suas instruções. Além disso, faz-se uso do *TLS* para a armazenagem de informações específicas a uma *thread*.

O registro da função de instrumentação é realizada através

4. Conclusões