Projeto 1 - Infraestrutura: PIN e SPEC 2006

Alceu Emanuel Bissoto[1]

[1]Instituto de Computação - Universidade Estadual de Campinas (UNICAMP)
Campinas – SP – Brasil

Abstract. *The knowledge in benchmarks and instrumentation frameworks are a "must-have"for every professional engaged in computer archtecture. In order to get a first contact with these tools, two tasks have been passed. The first, to count the amount of instructions in the different programs present on the benchmark SPEC2006, using the instrumentation framework PIN, made by Intel. The second one, is to create a new pintool and apply it to five different SPEC programs. For the second task, the new pintool that was created, and that it's going to be described in this paper, classify the different instructions in three groups in order to get more information about the softwares being analyzed.*

Resumo. *O conhecimento em* benchmarks *e ferramentas de instrumentação são essenciais para qualquer profissional que deseja trabalhar com arquitetura de computadores. De modo a obter um primeiro contato com essas ferramentas, foi designado para este trabalho que se cumprisse duas tarefas. A primeira, de contar a quantidade de instruções nos diferentes programas presentes no* benchmark SPEC 2006 *utilizando a ferramenta de instrumentação* PIN *da Intel, e a segunda, de criar uma nova* pintool *e aplicar a cinco programas do* SPEC. *Para esta segunda, foi criada uma* pintool *que será descrita neste trabalho, que classifica as instruções a serem executadas em três grupos, de modo a obter mais informações sobre os programas sendo analisados.*

1. Introdução

Neste documento, estará sendo descrito o primeiro contato com duas classes de *softwares*: *benchmarks* e programas de instrumentação.
Utilizando ferramentas de instrumentação, como *PIN* da [Intel], pode-se obetr mais informações sobre um *software* que deseja-se otimizar, possibilitando por exemplo observar as instruções que são geradas, como ele interage com a memória, observando a criação de threads, e a interação com chamadas de sistema e de sinais recebidos.
Quanto ao uso de *benchmarks* ele se mostra necessário quando se está desenvolvendo novas tecnologias relacionadas a arquitetura e hardware de forma geral, já que busca-se na maioria das vezes, uma melhoria em algum aspecto, seja esta energética, perfomática, ou de qualquer outra natureza. Com diferentes pessoas ao redor do mundo trabalhando em assuntos semelhantes, os *benchmarks* surgem como ferramentas para quantificar a performance de computadores, submetendo-os a altas cargas de trabalho e atribuindo "notas"pelo desempenho obtido. Desta forma é possível saber se uma solução é melhor que outra, e em que aspecto, ou seja, para que tipode trabalho, como por exemplo, para pontos flutuantes, inteiros, entre outros.
Desta forma, aprender a utilizar tais ferramentas se torna essencial para a formação de um pesquisador na área de arquitetura de computadores, além de dar uma percepção mais analítica para pesquisadores de outras áreas.

2. Primeira Tarefa

O primeiro trabalho realizado tem como objetivo descobrir o número de instruções executadas por cada um dos *benchmarks* presentes no *SPEC 2006*, utilizando o *PIN*, da Intel. Como este foi o primeiro contato com estes *softwares*, muita leitura da documentação foi necessária, e vários exemplos foram executados com o objetivo de ganhar familiaridade com as ferramentas em questão.
Para chegar ao resultado desejado, uma linha de código foi adicionada ao arquivo de configuração do SPEC, de modo que o PIN fosse executado para cada um dos diferentes benchmarks, e não se misturasse com as outras instruções geradas pelo comando "runspec"que inicia a execução em série dos mesmos. Foi utilizado um código exemplo disponibilizado com a instalação do PIN para a contagem de instruçes. O código adicionado foi o seguinte:

```
1 submit = /PinPath/pin -t /PinToolPath/inscount.so -- $command
2 use_submit_for_speed = yes
```

O resultado obtido, com a quantidade de instruções para cada benchmark, encontra-se na tabela 1.

3. Segunda Tarefa

A segunda parte do trabalho consiste em criar uma pintool utilizando o PIN de modo que uma nova informação fosse recebida, uma nova função fosse criada.
Olhando para os resultados obtidos na primeira parte do trabalho apenas, ou seja, a quantidade de instruções de um benchmark, não conseguimos dizer muito sobre ele, apesar de as intruções possuírem de fato o programa em si e todas suas operações e funções. Foi então que para esta segunda parte, buscou-se dividir as instruções em três diferentes grupos com a intenção de que pudéssemos dizer mais sobre os programas que estão sendo analisados. As divisões sugeridas foram:

- Operações aritméticas e lógicas - Operações como XOR, ADD, MUL.
- Operações de transferências de dados - Operações como MOV, PUSH.
- Controle do fluxo de execução - Operações como JMP, RET, CALL.

Para tal, utilizou-se de algumas funções do módulo de *Generic Inspection API - Instrumentation Object* do PIN que nos ajudam nessa divisão. Esses métodos foram chamados na seguinte sequência, em uma cadeia de if-then-else, pois alguns métodos retornam true para as mesmas instruções.

1. INS_IsBranchOrCall - Controle de fluxo de execução.
2. INS_IsRet - Controle de fluxo de execução.
3. INS_IsMov - Controle de fluxo de execução.
4. INS_IsMemoryWrite - Transferência de dados
5. INS_IsMemoryRead - Transferência de dados
6. As instruções restantes foram categorizadas como operações aritméticas e lógicas.

Durante a criação da pintool, foi possível observar a necessidade de otimizar ao máximo a ferramenta, utilizando sempre que possível funções da API oferecida. Nas primeiras tentativas, tentou-se usar vetores de string que continham as possíveis instruções

Benchmark	Quantidade de Instruções
436.cactusADM	2.598.819.640.983
435.gromacs	1.948.694.917.718
483.xalancbmk	1.057.903.253.252
471.omnetpp	571.971.917.187
458.sjeng	2.227.227.449.203
481.wrf	3.651.798.449.711
447.dealII	1.892.204.734.148
444.namd	2.288.042.088.026
403.gcc	55.429.606.648
999.specrand	570.685.954
465.tonto	3.556.084.302.173
462.libquantum	2.287.821.852.265
453.povray	996.442.214.620
470.lbm	1.259.843.806.515
482.sphinx3	3.353.483.217.474
433.milc	1.182.577.971.182
454.calculix	6.706.441.679.818
437.leslie3d	4.319.465.390.639
473.astar	865.363.805.090
400.perlbench	677.992.208.448
459.GemsFDTD	2.747.134.274.762
410.bwaves	2.372.304.023.044
445.gobmk	330.235.220.897
998.specrand	570.685.954
416.gamess	3.738.840.944.873
434.zeusmp	2.007.674.696.327
450.soplex	396.737.711.040
401.bzip2	321.075.549.753
456.hmmer	1.884.154.385.804
464.h264ref	3.035.883.185.313
429.mcf	300.181.778.624

Tabela 1. Quantidade de instruções para cada um dos diferentes programas do *SPEC 2006*

que seriam executadas, e então para cada nova instrução sendo executada, esta era confrontada com estes vetores, de modo que ela pudesse ser classificada em uma das três categorias. O problema é que procurar um elemento em uma string, acaba por ser uma operação custosa, fazendo com que o programa o qual deseja-se analisar demore até 20 vezes mais para terminar sua execução. Ao tentar utilizar esta solução em um dos programas do SPEC, foi evidente que uma otimização seria necessária. Foi necessário 1 dia inteiro para obter o resultado para um único programa. Foi então que buscou-se uma solução utilizando as funções da API disponibilizadas no PIN, de modo a ober um resultado semelhante ao esperado.

Nesta nova tentativa, foi possível obter o resultado para 5 programas diferentes do SPEC em menos de 4 horas. O trecho principal do código da pintool é evidenciado abaixo:

```
1 if (filter.SelectTrace(trace))
2     {
3         for (BBL bbl = TRACE_BblHead(trace); BBL_Valid(bbl);
4              bbl = BBL_Next(bbl))
5         {
6             for (INS ins = BBL_InsHead(bbl); INS_Valid(ins);
7                  ins = INS_Next(ins))
8             {
9                 if (INS_IsBranchOrCall(ins)) {
10                    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)
      BranchIns, IARG_END);
11                }
12                else if (INS_IsRet(ins)) {
13                    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)
      BranchIns, IARG_END);
14                }
15                else if (INS_IsMov(ins)) {
16                    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)MovIns,
       IARG_END);
17                }
18                else if (INS_IsMemoryWrite(ins) or INS_IsMemoryRead(ins
      ))
19                {
20                    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)MovIns,
       IARG_END);
21                }
22                else {
23                    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)
      ArithIns, IARG_END);
24                }
25            }
26            BBL_InsertCall(bbl, IPOINT_BEFORE, (AFUNPTR)IncrementIns,
      IARG_UINT32, BBL_NumIns(bbl), IARG_END);
27        }
28    }
29 }
```

Há a possibilidade de utilizar da função de filtro para ignorar as bibliotecas utilizadas, mas este recurso não foi utilizado na medida para os benchmarks do SPEC, já que acredita-se que elas sejam importantes para que possa-se fazer uma análise destes programas. O resultado foi reunido na tabela 2. Podemos ver que em todos os programas, as instruções de trânsferência de dados predominam, mostrando como é grande o esforço por parte do processador para ter acesso aos dados, assim como é grande a quantidade de dados que ele tem que lidar o tempo todo. As instruções de controle de execução ficaram por volta dos 23%, sem grande variação entre os programas do SPEC.

Para podermos ter uma ideia melhor sobre os diferentes programas, vamos a uma breve descrição sobre cada um dos analizados:

1. 483.xalancbmk - Transforma XML em HTML.
2. 471.omnetpp - Gerador de estatísticas de um simulador de redes.
3. 403.gcc - Compilador de 9 programas em C.
4. 999.specrand - Gerador de números pseudorandômicos.
5. 429.mcf - Otimizador que usa simulação de transporte publico.

Benchmark	Transf. Dados	Controle de Execução	Funções aritméticas e lógicas
483.xalancbmk	36,7%	27,4%	35,9%
471.omnetpp	49,1%	24,5%	26,4%
403.gcc	43,8%	23,8%	32,4%
999.specrand	45,2%	20,7%	34,1%
429.mcf	51,1%	22,9%	25,9%

Tabela 2. Porcentagem de instruções para cada categoria em diferentes programas do *SPEC 2006*

Há algumas propriedades interessantes entre eles. Os benchmarks que usam de um simulador, parecem precisar muito mais de transferências de dados, para por exemplo, carregar os dados da simulação. O 999.specrand que foca na geração de números pseudoaleatórios, e portanto, assume-se que é um benchmark rico em funções aritméticas e lógicas, tem um índice alto deste tipo de função, mas não foge muito a média e é menor que o do 483.xalancbmk, por exemplo.

4. Conclusão

Benchmarks e ferramentas de análise são fundamentais para a formação de um pesquisador em arquitetura de computadores. O primeiro contato com essas tecnologias mostrou-se desafiador, mas também mostrou a grande gama de aplicações que estas podem assumir. Com a ferramenta PIN da Intel, pode-se ter acesso às instruções e funções de um software que deseja-se analisar em tempo de execução, e conta uma documentação completa. O benchmark SPEC 2006 conta com diversos programas que buscam analisar o processador de formas diferentes, além de gerar um relatório completo para que o usuário possa comparar o resultado de seu sistema com outras pessoas.
Foram feitos estudos de modo a contar as instruções dos programas do SPEC 2006 utilizando o PIN, e também a criação de uma nova pintool que buscou obter mais informações sobre as instruções destes programas, categorizando-as. Instruções de controle de execução aparentam ter a mesma proporção independente do tipo de programa, enquanto a de transferência de dados mostram-se ser as maiores em quantidade, e em variação, podendo chegar a mais de 50% do total de instruções.

Referências

Intel. Pin: Pin 3.0 user guide. In *Data de Acesso: 15 de Setembro, 2016*. https://software.intel.com/sites/landingpage/pintool/docs/76991/Pin/html/index.html.