Introdução ao PERF
Determinação de Hotspots de execução de kernels recorrendo a PERF (linux-tools)

Filipe Oliveira
Departamento de Informática
Universidade do Minho
Email: a57816@alunos.uminho.pt

1. Introdução – Contextualização da utilização da ferramenta PERF

O presente trabalho prático surge como um "diário de bordo"relativo ao acompanhamento de um conjunto de 3 parte do tutorual: **"PERF tutorial: Finding execution hot spots"**.[1]

A parte 1 descreve a utilização da ferramenta PERF como meio de identificação e análise de hotspots de execução em kernels. Cobre maioritariamente os comandos PERF mais básico, e respectivas opções e eventos de medição de performance relativa.

A parte 2 introduz eventos que permitem a medição de contadores de hardware e demonstra a utilização do PERF nesse sentido. Discute ainda a importância de certas métricas de performance e a sua influência relativa na performance geral de uma aplicação.

A parte 3 recorre aos eventos de contadores de hardware para, uma vez mais e desta vez de uma forma mais aprofundada, analisar os hotspots de um kernel/aplicação.

Todo o tutorial assenta em duas versões de uma aplicação de multiplicação de matrizes: beginitemize

Naive: Algoritmo de multiplicação de matrizes com acessos a memória "column-wise".

Interchange: Algoritmo de multiplicação de matrizes com acessos a memória "row-wise".

A primeira versão serão, com seria de esperar, apresentará problemas de performance relacionados com o acesso à memória. É importante também realçar que as aplicações são ambas compiladas com a versão de compilador GCC 4.9.0., com as seguintes flags de compilação -O2 -ggdb -g". Denote que a flag de otimização -O3, que activa um conjunto de otimizações de código[2] está desativada, muito pela necessidade de manter o código (apesar de pouco eficiente) legível e com capacidade de ser compreendido pelo programador.

1. http://sandsoftwaresound.net/perf/perf-tutorial-hot-spots/

2. -finline-functions, -funswitch-loops, -fpredictive-commoning, -fgcse-after-reload, -ftree-vectorize, -fvect-cost-model, -ftree-partial-pre and -fipa-cp-clone options

2. Caracterização do Hardware do ambiente de testes

Tão importante com especificar as versões de código e as propriedades é especificar os ambientes de teste nos quais pretendemos realizar as benchmarks.

Através da análise do hardware disponível no Search6 [3], uma das nossas plataformas de teste, foi seleccionado o nó do tipo compute-431, sendo a disponibilidade global do mesmo e o correcto funcionamento do perf os principais factores. Na tabela 1 encontram-se especificadas as principais características dos sistemas em teste.

Tabela 1: Características de Hardware do nó 431

Sistema	compute-431
# CPUs	2
CPU	Intel® Xeon® X5650
Arquitectura de Processador	Nehalem
# Cores por CPU	6
# Threads por CPU	12
Freq. Clock	2.66 GHz
Cache L1	192KB (32KB por Core)
Cache L2	1536KB (256KB por Core)
Cache L3	12288KB (partilhada)
Ext. Inst. Set	SSE4.2
#Memory Channels	3
Memória Ram Disponível	48GB
Peak Memory BW Fab. CPU	32 GB/s

3. Determinação do tamanho dos datasets

Talvez dos pontos mais importantes presente na tabela de caracterização de hardware seja o tamanho da Cache L3, dado que ambos os algoritmos dependem massivamente de leitura/escrita na memória principal. Assim sendo, se pretendemos realçar as penalizações resultantes de diferentes tipos de acesso aos dados devemos realizar dois tipos distintos de testes:

- Matrizes contidas na LLC:
  Sabemos que as 3 matrizes terão de ter uma tamanho de coluna/linha inferior a:

$$lado\ matriz \leq \frac{\sqrt{(\frac{12288KB}{4\ Bytes\ p/float}}}{3\ matrizes} \leq 1011$$

3. Services and Advanced Research Computing with HTC/HPC clusters

para que os datasets estejam completamente contidos em cache. Foi escolhido o lado da matriz de 512 elementos.

- Matrizes contidas na memória principal:
  Sabemos que as 3 matrizes terão de ter uma tamanho de coluna/linha superior a 1011 elemento por linha/coluna. Foi escolhido o lado da matriz de 2048 elementos.

4. Determinação do tempo médio necessário para criar e terminar um fio de execução

4.1. Nós compute-431

Por forma a calcular o tempo médio necessário para criar e terminar um fio de execução foi criado um kernel, que apenas realizava essas mesmas operações e registados os valores para os diferentes número de threads. A tabela **??** apresenta a relação entre média e desvio padrão de criação/terminação para um diferente número de POSIX Threads para os nós do tipo compute-431.

5. Parte 1

5.1. Passo 0 – uma análise aos software e hardware events disponíveis na máquina

Recorrendo ao seguinte commando:

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf list | wc -l
```

Podemos ter uma noção em ternos de eventos de hardware e software disponíveis na máquina de teste:

```
784
```

5.2. 1º Passo – o encontrar dos tempos base sem overhead de medição

Por forma a podermos estabelecer um limite dentro do aceitável das possíveis alterações à performance das versões do kernel necessitamos de primeiramente ter uma medição com o mínimo de overhead possível. Recorrendo ao seguinte comando:

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf stat -e cpu-clock ./naive
```

```
 Performance counter stats for './naive':

        201.696023      cpu-clock (msec)

       0.206153405 seconds time elapsed
```

Teremos uma base de referência futura, para podermos validar as nossas medições com overhead.
Para além do tempo de execução é ainda demonstrado o número de cpu-clocks necessários para a execução.
Como seria de esperar, é possível num única medição registar vários eventos. Recorrendo ao seguinte comando:

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf stat -e cpu-clock,faults ./naive
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf stat -e cpu-clock,faults ./naive

 Performance counter stats for './naive':

        196.395754      cpu-clock (msec)
               844      faults

       0.200874285 seconds time elapsed
```

6. O processo de procura de hotpsots de uma aplicação

O processo de procura de hotpsots de uma aplicação pode ser traduzido em 2 passos simples:

- Passo 1: Recolha de dados para profiling da aplicação
- Passo 2 : Tratamento e apresentação da informação recolhida.

Podemos recolher informação de profiling simplesmente correndo o seguinte comando:

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf record -e cpu-clock,faults ./naive
```

Neste caso específico o PERF corre a aplicação naive e recolhe dados relativos aos eventos: cpu-clock e page-faults.

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf record -e cpu-clock,faults ./naive
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.046 MB perf.data (830 samples) ]
```

Relativamente ao passo 2 - Tratamento e apresentação da informação recolhida - o seguinte comando permite visualizar a informação presente no ficheiro prof.data (ficheiro default no qual é gravada a informação):

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf report --stdio
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf report --stdio
# To display the perf.data header info, please use --header/--header-only options.
#
# Samples: 808  of event 'cpu-clock'
# Event count (approx.): 808
```

```
#
# Overhead  Command  Shared Object       Symbol
# ........  .......  ..................  ↩
    ..............................
#
    94.68%  naive    naive               [.] ↩
         multiply_matrices
     2.10%  naive    naive               [.] ↩
          initialize_matrices
     1.36%  naive    libc-2.12.so        [.] ↩
          __random
     0.87%  naive    libc-2.12.so        [.] ↩
          __random_r
     0.25%  naive    [kernel.kallsyms]   [k] ↩
          __mem_cgroup_commit_charge
     0.25%  naive    naive               [.] ↩
          rand@plt
     0.12%  naive    [kernel.kallsyms]   [k] ↩
          __call_rcu
     0.12%  naive    [kernel.kallsyms]   [k] ↩
          __mem_cgroup_uncharge_common
     0.12%  naive    libc-2.12.so        [.] ↩
          malloc
     0.12%  naive    libc-2.12.so        [.] rand


# Samples: 22  of event 'faults'
# Event count (approx.): 1157
#
# Overhead  Command  Shared Object       Symbol
# ........  .......  ..................  ↩
    ......................
#
    63.61%  naive    naive               [.] ↩
         initialize_matrices
    29.56%  naive    libc-2.12.so        [.] ↩
          __init_cpu_features
     5.62%  naive    ld-2.12.so          [.] ↩
          dl_main
     0.52%  naive    ld-2.12.so          [.] ↩
          _dl_setup_hash
     0.26%  naive    ld-2.12.so          [.] ↩
          _dl_start
     0.17%  naive    [kernel.kallsyms]   [k] ↩
          __clear_user
     0.17%  naive    libc-2.12.so        [.] ↩
          __strstr_sse2
     0.09%  naive    ld-2.12.so          [.] ↩
          _start
```

```
[a57816@search6 ESC_FLAME_GRAPH]$ perf report --↩
    stdio --sort comm,dso
```

```
[a57816@search6 ESC_FLAME_GRAPH]$ perf report --↩
    stdio --sort comm,dso
[kernel.kallsyms] with build id 09↩
    b53ce5dfe09c7b1ad9473e1356d7d922512e27 not ↩
    found, continuing without symbols
# To display the perf.data header info, please ↩
    use --header/--header-only options.
#
# Samples: 808  of event 'cpu-clock'
# Event count (approx.): 808
#
# Overhead  Command  Shared Object
# ........  .......  ..................
#
    97.03%  naive    naive
     2.48%  naive    libc-2.12.so
     0.50%  naive    [kernel.kallsyms]


# Samples: 22  of event 'faults'
# Event count (approx.): 1157
#
```

```
# Overhead  Command  Shared Object
# ........  .......  ..................
#
    63.61%  naive    naive
    29.73%  naive    libc-2.12.so
     6.48%  naive    ld-2.12.so
     0.17%  naive    [kernel.kallsyms]
```

```
[a57816@search6 ESC_FLAME_GRAPH]$ perf report --↩
    stdio --sort comm,dso --header
```

```
[a57816@search6 ESC_FLAME_GRAPH]$ perf report --↩
    stdio --sort comm,dso --header
[kernel.kallsyms] with build id 09↩
    b53ce5dfe09c7b1ad9473e1356d7d922512e27 not ↩
    found, continuing without symbols
# ========
# captured on: Mon May 23 23:00:56 2016
# hostname : compute-431-1.local
# os release : 2.6.32-279.14.1.el6.x86_64
# perf version : 4.0.0
# arch : x86_64
# nrcpus online : 24
# nrcpus avail : 24
# cpudesc : Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
# cpuid : GenuineIntel,6,44,2
# total memory : 12319324 kB
# cmdline : /share/jade/SOFT/perf/perf record -e↩
    cpu-clock,faults ./naive
# event : name = cpu-clock, type = 1, config = 0↩
    x0, config1 = 0x0, config2 = 0x0, excl_usr ↩
    = 0, excl_kern = 0, excl_host = 0, ↩
    excl_guest = 1, precise_ip = 0, attr_mmap2 ↩
    = 0, attr
# event : name = faults, type = 1, config = 0x2,↩
     config1 = 0x0, config2 = 0x0, excl_usr = ↩
    0, excl_kern = 0, excl_host = 0, excl_guest↩
     = 1, precise_ip = 0, attr_mmap2 = 0, ↩
    attr_mm
# HEADER_CPU_TOPOLOGY info available, use -I to ↩
    display
# HEADER_NUMA_TOPOLOGY info available, use -I to↩
     display
# pmu mappings: cpu = 4, tracepoint = 2, ↩
    software = 1
# ========
#
# Samples: 808  of event 'cpu-clock'
# Event count (approx.): 808
#
# Overhead  Command  Shared Object
# ........  .......  ..................
#
    97.03%  naive    naive
     2.48%  naive    libc-2.12.so
     0.50%  naive    [kernel.kallsyms]


# Samples: 22  of event 'faults'
# Event count (approx.): 1157
#
# Overhead  Command  Shared Object
# ........  .......  ..................
#
    63.61%  naive    naive
    29.73%  naive    libc-2.12.so
     6.48%  naive    ld-2.12.so
     0.17%  naive    [kernel.kallsyms]
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf ↩
    report --stdio --dsos=naive,libc-2.12.so
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf report --stdio --dsos=naive,libc-2.12.so
# To display the perf.data header info, please use --header/--header-only options.
#
# Samples: 808  of event 'cpu-clock'
# Event count (approx.): 808
#
# Overhead  Command  Shared Object  Symbol
# ........  .......  .............  .......................
#
    94.68%  naive    naive          [.] multiply_matrices
     2.10%  naive    naive          [.] initialize_matrices
     1.36%  naive    libc-2.12.so   [.] __random
     0.87%  naive    libc-2.12.so   [.] __random_r
     0.25%  naive    naive          [.] rand@plt
     0.12%  naive    libc-2.12.so   [.] malloc
     0.12%  naive    libc-2.12.so   [.] rand


# Samples: 22  of event 'faults'
# Event count (approx.): 1157
#
# Overhead  Command  Shared Object  Symbol
# ........  .......  .............  .......................
#
    63.61%  naive    naive          [.] initialize_matrices
    29.56%  naive    libc-2.12.so   [.] __init_cpu_features
     0.17%  naive    libc-2.12.so   [.] __strstr_sse2
```

```
Percent | Source code & Disassembly of naive for cpu-clock
```

```
Percent | Source code & Disassembly of naive for cpu-clock
------------------------------------------------------------------------------
         :
         :
         :
         : Disassembly of section .text:
         :
         : 0000000000400810 <multiply_matrices>:
         : multiply_matrices():
         :                 }
         :         }
         : }
         :
         : void multiply_matrices()
         : {
    0.00 :   400810:       pxor   %xmm2,%xmm2
    0.00 :   400814:       mov    $0x7e97c0,%edi
    0.00 :   400819:       mov    %rdi,%r8
    0.00 :   40081c:       xor    %esi,%esi
    0.00 :   40081e:       sub    $0x7e97c0,%r8
    0.00 :   400825:       nopl   (%rax)
    0.13 :   400828:       lea    0x6f5580(%rsi),%rax
    0.00 :   40082f:       lea    0x7e97c0(%rsi),%rcx
    0.00 :   400836:       mov    %rdi,%rdx
    0.00 :   400839:       movaps %xmm2,%xmm1
    0.00 :   40083c:       nopl   0x0(%rax)
         :
         :         for (i = 0 ; i < MSIZE ; i++)
                   {
         :                 for (j = 0 ; j < MSIZE ; j++) {
         :                         float sum = 0.0 ;
         :                         for (k = 0 ; k < MSIZE ; k++) {
         :                                 sum = sum + (matrix_a[i][k] * matrix_b[k][j]) ;
   24.97 :   400840:       movss  (%rdx),%xmm0
    0.00 :   400844:       add    $0x7d0,%rax
    0.00 :   40084a:       mulss  -0x7d0(%rax),%xmm0
   18.56 :   400852:       add    $0x4,%rdx
         :         int i, j, k ;
         :
         :         for (i = 0 ; i < MSIZE ; i++)
                   {
         :                 for (j = 0 ; j < MSIZE ; j++) {
         :                         float sum = 0.0 ;
         :                         for (k = 0 ; k < MSIZE ; k++) {
   22.35 :   400856:       cmp    %rcx,%rax
         :                                 sum = sum + (matrix_a[i][k] * matrix_b[k][j]) ;
    0.00 :   400859:       addss  %xmm0,%xmm1
         :         int i, j, k ;
         :
         :         for (i = 0 ; i < MSIZE ; i++)
                   {
         :                 for (j = 0 ; j < MSIZE ; j++) {
         :                         float sum = 0.0 ;
         :                         for (k = 0 ; k < MSIZE ; k++) {
   33.86 :   40085d:       jne    400840 <multiply_matrices+0x30>
         :                                 sum = sum + (matrix_a[i][k] * matrix_b[k][j]) ;
         :                         }
         :                         matrix_r[i][j] = sum ;
    0.00 :   40085f:       movss  %xmm1,0x601340(%r8,%rsi,1)
    0.13 :   400869:       add    $0x4,%rsi
         : void multiply_matrices()
         : {
         :         int i, j, k ;
         :
         :         for (i = 0 ; i < MSIZE ; i++)
                   {
         :                 for (j = 0 ; j < MSIZE ; j++) {
    0.00 :   40086d:       cmp    $0x7d0,%rsi
    0.00 :   400874:       jne    400828 <multiply_matrices+0x18>
    0.00 :   400876:       add    $0x7d0,%rdi
         :
         : void multiply_matrices()
         : {
         :         int i, j, k ;
         :
         :         for (i = 0 ; i < MSIZE ; i++)
                   {
    0.00 :   40087d:       cmp    $0x8dda00,%rdi
    0.00 :   400884:       jne    400819 <multiply_matrices+0x9>
    0.00 :   400886:       repz retq
```

```
Percent | Source code & Disassembly of naive for cpu-clock
```

```
 Percent | Source code & Disassembly of naive ↩
      for cpu-clock
-------------------------------------------------↩

           :
           :
           :
           : Disassembly of section .text:
           :
           : 0000000000400810 <multiply_matrices>:
           : multiply_matrices():
    0.00 :     400810:       pxor    %xmm2,%xmm2
    0.00 :     400814:       mov     $0x7e97c0,%edi
    0.00 :     400819:       mov     %rdi,%r8
    0.00 :     40081c:       xor     %esi,%esi
    0.00 :     40081e:       sub     $0x7e97c0,%r8
    0.00 :     400825:       nopl    (%rax)
    0.13 :     400828:       lea     0x6f5580(%rsi)↩
           ,%rax
    0.00 :     40082f:       lea     0x7e97c0(%rsi)↩
           ,%rcx
    0.00 :     400836:       mov     %rdi,%rdx
    0.00 :     400839:       movaps  %xmm2,%xmm1
    0.00 :     40083c:       nopl    0x0(%rax)
   24.97 :     400840:       movss   (%rdx),%xmm0
    0.00 :     400844:       add     $0x7d0,%rax
    0.00 :     40084a:       mulss   -0x7d0(%rax),%↩
           xmm0
   18.56 :     400852:       add     $0x4,%rdx
   22.35 :     400856:       cmp     %rcx,%rax
    0.00 :     400859:       addss   %xmm0,%xmm1
   33.86 :     40085d:       jne     400840 <↩
          multiply_matrices+0x30>
    0.00 :     40085f:       movss   %xmm1,0x601340↩
           (%r8,%rsi,1)
    0.13 :     400869:       add     $0x4,%rsi
    0.00 :     40086d:       cmp     $0x7d0,%rsi
    0.00 :     400874:       jne     400828 <↩
           multiply_matrices+0x18>
    0.00 :     400876:       add     $0x7d0,%rdi
    0.00 :     40087d:       cmp     $0x8dda00,%rdi
    0.00 :     400884:       jne     400819 <↩
           multiply_matrices+0x9>
    0.00 :     400886:       repz retq
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf ↩
     record -e cpu-clock --freq=8000 ./naive
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf ↩
     record -e cpu-clock --freq=8000 ./naive
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.070 MB perf.↩
     data (1636 samples) ]
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf ↩
     report --stdio --show-nr-samples --dsos=↩
     naive
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf ↩
     report --stdio --show-nr-samples --dsos=↩
     naive
# To display the perf.data header info, please ↩
    use --header/--header-only options.
#
# dso: naive
# Samples: 1K of event 'cpu-clock'
# Event count (approx.): 1636
#
# Overhead        Samples  Command  Symbol
# ........  ............  .......  ↩
    ......................
#
    94.74%          1550   naive     [.] ↩
        multiply_matrices
     1.47%            24   naive     [.] ↩
         initialize_matrices
     0.06%             1   naive     [.] rand@plt
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf ↩
     evlist -F
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf ↩
     evlist -F
cpu-clock: sample_freq=8000
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$      perf↩
      list | wc -l
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$      perf↩
      list | wc -l
784
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$      perf↩
      stat -e cpu-cycles,instructions ./naive
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$      perf↩
      stat -e cpu-cycles,instructions ./naive

 Performance counter stats for './naive':

         516402889      cpu-cycles
         908042685      instructions ↩
                           #    1.76  insns per ↩
              cycle

       0.200239282 seconds time elapsed
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf ↩
     stat -e cpu-cycles,instructions,cache-↩
     references,cache-misses,branch-instructions↩
     ,branch-misses,bus-cycles,L1-dcache-loads,↩
     L1-dcache-load-misses,L1-dcache-stores,L1-↩
     dcache-store-misses,LLC-loads,LLC-load-↩
     misses,LLC-stores,LLC-store-misses,dTLB-↩
     load-misses,dTLB-store-misses,iTLB-load-↩
     misses,branch-loads,branch-load-misses ./↩
     naive
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf ↩
     stat -e cpu-cycles,instructions,cache-↩
     references,cache-misses,branch-instructions↩
     ,branch-misses,bus-cycles,L1-dcache-loads,↩
     L1-dcache-load-misses,L1-dcache-stores,L1-↩
     dcache-store-misses,LLC-loads,LLC-load-↩
     misses,LLC-stores,LLC-store-misses,dTLB-↩
     load-misses,dTLB-store-misses,iTLB-load-↩
     misses,branch-loads,branch-load-misses ./↩
     naive

 Performance counter stats for './naive':

         535187277      cpu-cycles ↩
                              [20.25%]
        1044692763      instructions ↩
                         #    1.95  insns per ↩
             cycle            [25.37%]
           8196140      cache-references ↩
                                                  ↩
               [25.37%]
             36522      cache-misses ↩
                            #    0.446 % of ↩
                 all cache refs     [25.57%]
```

```
         126101720      branch-instructions #   27.67%
            258384      branch-misses #    0.20% of all branches          [28.04%]
                 0      bus-cycles                                        [11]
         246027409      L1-dcache-loads                                   [12]
                                                                          [22.50%]
          56436199      L1-dcache-load-misses #   22.94% of all L1-dcache hits   [22.39%]
           9973628      L1-dcache-stores                                  [14]
                                                                          [22.27%]
            322982      L1-dcache-store-misses                            [15]
                                                                          [22.16%]
           7391770      LLC-loads                                         [16]
                                                                          [22.05%]
              2671      LLC-load-misses #    0.04% of all LL-cache hits     [21.94%]
            218407      LLC-stores                                        [18]
                                                                          [10.92%]
             18512      LLC-store-misses                                  [19]
                                                                          [10.86%]
              2239      dTLB-load-misses                                  [20]
                                                                          [16.21%]
               446      dTLB-store-misses                                 [21]
                                                                          [21.51%]
                 0      iTLB-load-misses                                  [22]
                                                                          [21.41%]
         129163483      branch-loads                                      [23]
                                                                          [21.22%]
           5688441      branch-load-misses                                [24]
                                                                          [20.73%]
                                                                          [25]
       0.210071197 seconds time elapsed                                   [26]
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf stat -e cpu-cycles,instructions,cache-references,cache-misses,branch-instructions,branch-misses,bus-cycles,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-store-misses,LLC-loads,LLC-load-misses,LLC-stores,LLC-store-misses,dTLB-load-misses,dTLB-store-misses,iTLB-load-misses,branch-loads,branch-load-misses ./interchange
```

```
[a57816@compute-431-1 ESC_FLAME_GRAPH]$ perf stat -e cpu-cycles,instructions,cache-references,cache-misses,branch-instructions,branch-misses,bus-cycles,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-store-misses,LLC-loads,LLC-load-misses,LLC-stores,LLC-store-misses,dTLB-load-misses,dTLB-store-misses,iTLB-load-misses,branch-loads,branch-load-misses ./interchange

 Performance counter stats for './interchange':

         399561216      cpu-cycles                                        [5]
                                                                          [21.57%]
        1152237507      instructions #    2.88  insns per cycle           [6]
                                                                          [27.27%]
```

```
         429971      cache-references                                  [7]
                                                                       [27.74%]
          43034      cache-misses #   10.009 % of all cache refs      [28.20%]
      132065934      branch-instructions                               [9]
                                                                       [28.66%]
         249858      branch-misses #    0.19% of all branches          [28.22%]
              0      bus-cycles                                        [11]
      253077242      L1-dcache-loads                                   [12]
                                                                       [21.87%]
         7577858      L1-dcache-load-misses #    2.99% of all L1-dcache hits   [21.24%]
      128034804      L1-dcache-stores                                  [14]
                                                                       [20.60%]
         106020      L1-dcache-store-misses                            [15]
                                                                       [20.38%]
         262810      LLC-loads                                         [16]
                                                                       [22.23%]
           1001      LLC-load-misses #    0.38% of all LL-cache hits     [22.08%]
          69369      LLC-stores                                        [18]
                                                                       [10.96%]
              0      LLC-store-misses                                  [19]
                                                                       [10.88%]
            950      dTLB-load-misses                                  [20]
                                                                       [16.21%]
              9      dTLB-store-misses                                 [21]
                                                                       [21.47%]
              0      iTLB-load-misses                                  [22]
                                                                       [21.33%]
      129898962      branch-loads                                      [23]
                                                                       [21.19%]
        5560030      branch-load-misses                                [24]
                                                                       [21.05%]
                                                                       [25]
    0.159788849 seconds time elapsed                                   [26]
```

7. Parte 2

Tabela 2: Performance events (naive vs. interchange) para o nó compute-431

# EVENT NAME	NAIVE	INTERCHANGE
cpu-cycles	535187277	399561216
instructions	1044692763	1152237507
cache-references	8196140	429971
cache-misses	36522	43034
branch-instructions	126101720	132065934
branch-misses	258384	249858
bus-cycles	0	0
L1-dcache-loads	246027409	253077242
L1-dcache-load-misses	56436199	7577858
L1-dcache-stores	9973628	128034804
L1-dcache-store-misses	322982	106020
LLC-loads	7391770	262810
LLC-load-misses	2671	1001
LLC-stores	218407	69369
LLC-store-misses	18512	0
dTLB-load-misses	2239	950
dTLB-store-misses	446	9
iTLB-load-misses	0	0
branch-loads	129163483	129898962
branch-load-misses	5688441	5560030

Tabela 3: Performance events (naive vs. interchange) para o nó compute-431

RATIO OR RATE	NAIVE	INTERCHANGE
Elapsed time (seconds)	0.2041	0.1597
Instructions per cycle	1.95 IPC	2.88 IPC
L1 cache miss ratio	22,9389 %	2,9942 %
L1 cache miss rate PTI	54,0218	6,5766
L3 cache miss ratio	0,0361 %	0,3808 %
Data TLB miss ratio	0,00027	0,0022
Data TLB miss rate PTI	0,0021	0,0008
Branch mispredict ratio	0,002	0,0019
Branch mispredict rate PTI	0,2473	0,2168

8. Parte 3

8.1. Período de amostragem e frequência de amostragem

Dado que pretendemos encontrar os hotspots de ambas as versões, iremos usar a amostragem baseada em cpu-cycles. Desta forma, porções da aplicação que consumam mais tempo terão um maior número de amostras registadas. No entanto, este tipo de amostragem tem um preço – pode ser demasiado pesada na avaliação de desempenho da aplicação.
Analisemos o tempo que demoram as versões large_naive e large_interchange sem qualquer tipo de ferramenta de amostragem por forma a podermos validar os resultados da avaliação de desempenho futura. Denote que por forma a validar os resultados foram realizadas 50 medições, sendo o valor apresentado o K Best (sendo K = 3) :

EVENT NAME	L. NAIVE	L. INTERCHANGE
Elapsed time (seconds)	10.43	65.61

Cada máquina tem um numero de contadores maximos e fixados a um certo tipo de eventos. Multiplexagem gasta tempo. Peso muito grande na avaliação de desempenho. quais os contadores do nosso interesse e a frequencia de amostragem.
When looking for the hottest spots in an application, you may choose to profile your program using cpu-cycles because the cpu-cycles event is a measure of time just like the cpu-clock software event.

Tabela 4: Performance events (naive vs. interchange) para o nó compute-431

# EVENT NAME	NAIVE	INTERCHANGE
Elapsed time		
instructions	38376000000	41776400000
cycles	78390700000	12384800000
cache-references	4744200000	14400000
cache-misses	4008300000	11200000
LLC-loads	4815000000	14800000
LLC-load-misses	4073600000	14400000
dTLB-load-misses	1100000	100000
branches	3923900000	3847500000
branch-misses	2200000	1900000

8.2. Análise comparativa do número de amostras por evento de hardware para as versões large_naive e large_interchange

Da análise de execução para as versões large_naive e large_interchange, large$_naiveandlarge_interchange, andcollectedeventsampledat$

Tabela 5: Sampling mode: large_naive vs. large_interchange para o nó compute-431

# EVENT NAME	NAIVE	INTERCHANGE
Elapsed time		
instructions	383K samples	417K samples
cycles	783K samples	123K samples
cache-references	47K samples	144 samples
cache-misses	40K samples	112 samples
LLC-loads	48K samples	148 samples
LLC-load-misses	40K samples	144 samples
dTLB-load-misses	11 samples	1 samples
branches	39K samples	38K samples
branch-misses	22 samples	19 samples

9. Conclusão