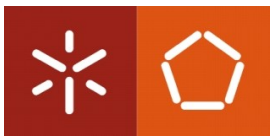
Perf

Engenharia de Sistemas da Computação

Computação Paralela Distribuída

Mestrado em Engenharia Informática

Universidade do Minho



Duarte Nuno Ferreira Duarte

pg27715

Índice

Introdução

Este trabalho demonstra a utilização do perf num programa que multiplica duas matrizes de forma muito básica e sem qualquer optimização. O perf é poderoso, consegue contadores de desempenho CPU e tracepoints entre outros. Está incluído no kernel do Linux e é regularmente atualizado.

O perf começou como uma ferramenta para usar o subsistema de contadores de desempenho no Linux, e tem tido várias melhorias.

Código de teste

O código apresentado de seguida é o código que será utilizado pelo perf.

```
void multiply_matrices() {
        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++) {
                             sum = sum + (matrix_a[i][k] *
matrix_b[k][j]);
                }
               matrix_r[i][j] = sum ;
                }
        }
}
```

Como é possível ver o código é o mais básico que existe de multiplicação de matrizes não possui qualquer tipo de otimização mas faz o esperado.

Eventos

De seguida são apresentados os eventos suportados no compute-431-10.local. Existem três tipos de eventos medidos pelo perf, os eventos de hardware, os eventos de software e ainda os eventos de cache de hardware.

cpu-cycles OR cycles	[Hardware event]
instructions	[Hardware event]
cache-references	[Hardware event]
cache-misses	[Hardware event]
branch-instructions	[Hardware event]
branch-misses	[Hardware event]
stalled-cycles-frontend	[Hardware event]
stalled-cycles-backend	[Hardware event]

Esta tabela mostra os eventos de hardware suportados neste nó do cluster e a azul escuro (bold) são os eventos que serão utilizados nas medições.

cpu-clock	[Software event]
task-clock	[Software event]
page-faults OR faults	[Software event]
context-switches OR cs	[Software event]
cpu-migrations OR migrations	[Software event]
minor-faults	[Software event]
major-faults	[Software event]
alignment-faults	[Software event]
emulation-faults	[Software event]

Esta tabela mostra os eventos de software suportados neste nó do cluster.

L1-dcache-loads	[Hardware cache event]
L1-dcache-load-misses	[Hardware cache event]
L1-dcache-stores	[Hardware cache event]
L1-dcache-store-misses	[Hardware cache event]
L1-dcache-prefetches	[Hardware cache event]
L1-icache-loads	[Hardware cache event]
L1-icache-load-misses	[Hardware cache event]
LLC-loads	[Hardware cache event]
LLC-load-misses	[Hardware cache event]
LLC-stores	[Hardware cache event]
LLC-store-misses	[Hardware cache event]
dTLB-loads	[Hardware cache event]
dTLB-load-misses	[Hardware cache event]
dTLB-stores	[Hardware cache event]
dTLB-store-misses	[Hardware cache event]
iTLB-loads	[Hardware cache event]

iTLB-load-misses	[Hardware cache event]
branch-loads	[Hardware cache event]
branch-load-misses	[Hardware cache event]

Esta tabela mostra os eventos de cache de hardware.

Eventos medidos

Com os contadores de eventos que foram usados e foram vistos acima foi possível obter os seguintes resultados.

230263545	cpu-cycles	2,735 GHz
332099971	instructions	1,46 insns per cycle
84,193928	task-clock (msec)	0,802 CPUs utilized
14	context-switches	0,166 K/sec
0	cpu-migrations	0,000 K/sec
843	page-faults	0,010 M/sec
224838277	cycles	2,670 GHz
89984496	stalled-cycles-frontend	39,54% frontend cycles idle
21576052	stalled-cycles-backend	9,48% backend cycles idle
342143914	instructions	1,50 insns per cycle
40157203	branches	476,961 M/sec
69956	branch-misses	0,17% of all branches
16598576	L1-dcache-load-misses	197,147 M/sec

Como é possível observar não houve qualquer cpu-migration pois usou sempre o mesmo CPU durante a execução. A apontar também o reduzido numero de saltos por falha (branch-misses) que ocorre apenas no fim de cada multiplicação parcial.

Perfis de execução

São agora apresentados os perfis de execução. Um obtido a partir do ambiente interativo oferecido pelo perf e outro a partir do output gerado.

```
# Overhead  Command      Shared Object
# ........  .......  ..................
#
    90.33%    naive  naive
     6.95%    naive  libc-2.12.so
     2.72%    naive  [kernel.kallsyms]


# Samples: 20  of event 'faults'
# Event count (approx.): 909
#
# Overhead  Command      Shared Object
# ........  .......  ..................
#
    84.93%    naive  naive
    14.52%    naive  ld-2.12.so
     0.33%    naive  [kernel.kallsyms]
     0.22%    naive  libc-2.12.so
```

Este foi obtido através do output do perf e mostra claramente que a multiplicação naíve usa 90.33% como seria de esperar visto que é o programa a ser testado.

Overhead	Cmd	Shared	Object
89,02%	naive	naive	[.] main
3,66%	naive	libc-2.12.so	[.] __random_r
2,20%	naive	libc-2.12.so	[.] __random
1,76%	naive	[kernel.kallsyms]	[k] __do_softirq
0,73%	naive	[kernel.kallsyms]	[k] __do_page_fault
0,44%	naive	naive	[.] rand@plt
0,29%	naive	[kernel.kallsyms]	[k] clear_page_c
0,29%	naive	libc-2.12.so	[.] rand
0,15%	naive	[kernel.kallsyms]	[k] ____pagevec_lru_add
0,15%	naive	[kernel.kallsyms]	[k] __audit_syscall_exit
0,15%	naive	[kernel.kallsyms]	[k] __bitmap_empty
0,15%	naive	[kernel.kallsyms]	[k] __lru_cache_add
0,15%	naive	[kernel.kallsyms]	[k] __mem_cgroup_uncharge_common
0,15%	naive	[kernel.kallsyms]	[k] __percpu_counter_add
0,15%	naive	[kernel.kallsyms]	[k] _atomic_dec_and_lock
0,15%	naive	[kernel.kallsyms]	[k] _spin_lock
0,15%	naive	[kernel.kallsyms]	[k] anon_vma_prepare
0,15%	naive	[kernel.kallsyms]	[k] handle_pte_fault
0,15%	naive	[kernel.kallsyms]	[k] sys_open

Agora no ambiente interativo foi possível obter esta tabela que demonstra o esperado, a multiplicação de matrizes a ocupar 89,02% do tempo e de realçar os 3,66% da inicialização com valores aleatórios da matriz.

```
 23,85 │150:   movaps %xmm2,%xmm0
       │       movlps (%rdx),%xmm0
  0,33 │       movhps 0x8(%rdx),%xmm0
  0,33 │       add    $0x4,%rdx
  4,11 │       shufps $0x0,%xmm0,%xmm0
       │       mulps  (%rcx,%rax,1),%xmm0
 47,70 │       add    $0x7d0,%rax
 10,03 │       cmp    $0xf4240,%rax
       │       addps  %xmm0,%xmm1
  9,38 │     ↑ jne    150
```

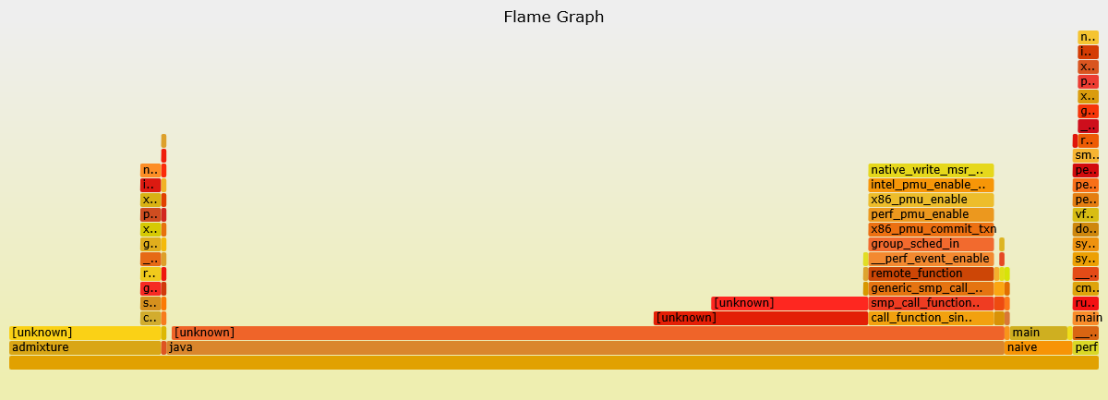
Observando mais especificamente com o perf annotate é possível ver que 47,70 é na soma dos valores da matriz que é apresentado no excerto de código seguinte.

```
sum = sum + (matrix_a[i][k] * matrix_b[k][j]) ;
```

Depois tem 10,03 na comparação para saber se termina o ciclo e 9,38 no salto caso saia do ciclo.

Flame Graph

Foi usado o software flamegraph que permite gerar gráficos a partir de medições do perf e não só. O objectivo era ver o impacto da aplicação no sistema e o impacto é agora apresentado.



Como é possível ver, com alguma dificuldade, a execução da aplicação aparece com o nome naíve na segunda linha a contar do fim do lado direito. A aplicação não tem grande impacto no sistema.

Conclusão

Foi uma boa experiencia utilizar o perf num caso prático. É uma ferramenta que tem grande potencial podendo ser uma ajuda enorme no desenvolvimento de software na medida em que contribui para ver os “hotspots” da execução. Assim é possível saber o que melhorar na aplicação de forma a obter melhores desempenhos.