Table of Contents

Benchmark on i7 architectures (Nehalem and Westmere)

18/01/2009

daily work Intel Core i7

compiling root

```
time make
real   19m14.377s user   17m47.768s sys   1m26.761s

time taskset -c 0-3 make -j 4
real   5m24.641s user   18m42.483s sys   1m27.961s

time make -j 4
real   5m16.044s user   19m27.527s sys   1m36.460s

time make -j 6
real   4m53.437s user   25m30.875s sys   2m0.751s

time make -j 7
real   4m44.131s user   27m48.481s sys   2m9.644s

time make -j 8
real   4m42.815s user   29m38.349s sys   2m16.792s

time make -j 12
real   4m52.696s user   30m3.299s sys   2m18.458s
```

compressing text files

```
time gzip -9 root_v5.22.00.source.tar
real   0m10.852s user   0m10.758s sys   0m0.091s

time /data/vin/pigs/pigz-2.1.4/pigz -9 -p 2 root_v5.22.00.source.tar
real   0m5.539s user   0m10.943s sys   0m0.160s

time /data/vin/pigs/pigz-2.1.4/pigz -9 -p 4 root_v5.22.00.source.tar
real   0m2.741s user   0m10.759s sys   0m0.161s

time /data/vin/pigs/pigz-2.1.4/pigz -9 -p 8 root_v5.22.00.source.tar
real   0m2.155s user   0m16.587s sys   0m0.226s
```

18/01/2009

more Evaluation of the Intel Core i7

`scimark2` can be used to evaluate cache and memory access. standard scimark2 fits in cache even in old cpu while its `large` version was supposed to stress memory access. This is less and less true. For instance on `core 2` we get

```
[pcphsft50] ~/w1/scimark2 > ./scimark2 5
Composite Score:          827.37
FFT             Mflops:   667.01    (N=1024)
SOR             Mflops:   806.26    (100 x 100)
```

```
MonteCarlo:     Mflops:   359.71
Sparse matmult  Mflops:   962.00    (N=1000, nz=5000)
LU              Mflops:  1341.85    (M=100, N=100)
[pcphsft50] ~/w1/scimark2 > ./scimark2 -large 5
Composite Score:          529.93
FFT             Mflops:    60.86    (N=1048576)
SOR             Mflops:   768.39    (1000 x 1000)
MonteCarlo:     Mflops:   359.71
Sparse matmult  Mflops:   458.17    (N=100000, nz=1000000)
LU              Mflops:  1002.51    (M=1000, N=1000)
```

with top showing some 0.1% memory activity

while on the `core i7` (very same binary)

```
./oldscimark2 5
Composite Score:         1006.79
FFT             Mflops:   735.97    (N=1024)
SOR             Mflops:   941.13    (100 x 100)
MonteCarlo:     Mflops:   416.18
Sparse matmult  Mflops:  1217.86    (N=1000, nz=5000)
LU              Mflops:  1722.82    (M=100, N=100)

./oldscimark2 -large 5
Composite Score:          835.89
FFT             Mflops:   159.34    (N=1048576)
SOR             Mflops:   858.78    (1000 x 1000)
MonteCarlo:     Mflops:   409.04
Sparse matmult  Mflops:  1090.81    (N=100000, nz=1000000)
LU              Mflops:  1661.47    (M=1000, N=1000)
```

with top showing some 0.2% memory activity

--++ 14/01/2009

Evaluation of the Intel Core i7

WP8 has acquired a desktop equipped with the *new* intel 4-core processor `Intel(R) Core(TM) i7 CPU 940 @ 2.93GHz` with 6GB memory

The *i7* introduces novel (for Intel) architecture features for multicore processors including a new three level cache hierarchy, new memory/core interconnect and hyperthreading. More info can be found for instance on wikipedia and on this very detailed article on bit-tech.net.

OS installation and os issues

Apparently the only OSes that install out of the box are Ubuntu9 and fedora10 (mainly because the desktop comes with a new ethernet card). Our machine (`pcphsft60`) has been installed with fedora10 that runs `gcc version 4.3.2 20081105 (Red Hat 4.3.2-7) (GCC)` as native compiler. Hyperthreading and clock-throttling are on by default at BIOS level. I controlled clock frequency using `cpuspeed` thats seems to do the job properly.

The scheduler does not necessarily balance the load among cores to best fit the user application: multiple jobs can be distributed manually using `taskset`. My understanding is that cpu 0,4 correspond to one core, cpu 1,5 to a second and so on. so running on cpu 0 and 1 will run on different cores while running on cpu 0,4 will run two hyperthreads (HTs) on the same core. Setting the cpu affinity for multithread application is more difficult: at the moment I prefer to watch the result using `top` with option `1` (click `1` on top of top) and eventually repeat the test until the threads are distributed uniformly among the four core.

- *edit 17/01/2009* actualy using `taskset -c n-m` one can force a multithread program to run on cpus `n` to `m`

running a simple benckmark `scimark2`

as first test I compiled and run `scimark2`

running it on two different cores I get

```
taskset -c 1 /data/vin/scimark2/scimark2 5
taskset -c 0 /data/vin/scimark2/scimark2 5
Composite Score:         1002.63
FFT             Mflops:   797.37    (N=1024)
SOR             Mflops:   935.42    (100 x 100)
MonteCarlo:     Mflops:   222.08
Sparse matmult  Mflops:  1261.82    (N=1000, nz=5000)
LU              Mflops:  1796.49    (M=100, N=100)

Composite Score:          998.36
FFT             Mflops:   795.55    (N=1024)
SOR             Mflops:   933.15    (100 x 100)
MonteCarlo:     Mflops:   221.16
Sparse matmult  Mflops:  1254.28    (N=1000, nz=5000)
LU              Mflops:  1787.67    (M=100, N=100)
```

while running on the same core one gets

```
taskset -c 4 /data/vin/scimark2/scimark2 5
taskset -c 0 /data/vin/scimark2/scimark2 5
Composite Score:          658.74
FFT             Mflops:   492.03    (N=1024)
SOR             Mflops:   877.88    (100 x 100)
MonteCarlo:     Mflops:   151.87
Sparse matmult  Mflops:   764.27    (N=1000, nz=5000)
LU              Mflops:  1007.63    (M=100, N=100)

Composite Score:          670.42
FFT             Mflops:   495.52    (N=1024)
SOR             Mflops:   878.88    (100 x 100)
MonteCarlo:     Mflops:   151.66
Sparse matmult  Mflops:   767.63    (N=1000, nz=5000)
LU              Mflops:  1058.40    (M=100, N=100)
```

so for this benchmark on average a single HT is 35% slower that single core (50% slower means no gain in using HT) which turns in a gain of 30% performance per core in running two processes on each core.

running a trivial multi-thread program

I then moved to test a trivial multi-thread program: the brute-force prime-number generator in its parallel-std implementation. It scales perfectly on a 2x4 core `Intel(R) Xeon(R) CPU E5430 @ 2.66GHz` On that machine pfmon reports

```
                          CPI: 1.1692
           load instructions %: 0.229%
          store instructions %: 0.161%
  load and store instructions %: 0.390%
  resource stalls % (of cycles): 61.144%
         branch instructions %: 22.169%
% of branch instr. mispredicted: 0.141%
          % of l2 loads missed: 41.236%
             bus utilization %: 1.902%
        data bus utilization %: 1.035%
```

```
               bus not ready %: 0.004%
  comp. SIMD inst. ('new FP') %: 0.071%
  comp. x87 instr. ('old FP') %: 21.947%
```

on the new `i7` I get (making sure that threads are properly balance among cores...)

```
export OMP_NUM_THREADS=8
real   0m24.161s user   3m10.907s sys   0m0.204s

export OMP_NUM_THREADS=4
real   0m24.428s user   1m36.672s sys   0m0.165s

export OMP_NUM_THREADS=3
real   0m32.878s user   1m37.926s sys   0m0.146s
```

i.e. HT makes no difference.

The L2 cache trashing is due to the single pass over the large vector containing the numbers to test. Anyhow load/store are not the dominant instructions...

A more realistic multi-thread example: Minuit

I tested then the new version of minuit able to compute derivatives in parallel

on the usual 8core machine pfmon reports

```
                           CPI: 0.9990
           load instructions %: 25.274%
          store instructions %: 9.860%
  load and store instructions %: 35.134%
  resource stalls % (of cycles): 61.208%
         branch instructions %: 10.526%
% of branch instr. mispredicted: 2.500%
          % of l2 loads missed: 0.101%
             bus utilization %: 0.082%
        data bus utilization %: 0.034%
               bus not ready %: 0.105%
  comp. SIMD inst. ('new FP') %: 23.684%
  comp. x87 instr. ('old FP') %: 0.000%
```

and it runs as

```
export OMP_NUM_THREADS=8
real   0m25.731s user   2m45.246s sys   0m0.032s

export OMP_NUM_THREADS=4
real   0m45.684s user   2m45.454s sys   0m0.012s
```

corresponding to a 1.8 gain doubling the processing power used (notice how the cpu-user time is the same).

on the new `i7` I get (making sure that threads are properly balance among cores...)

```
export OMP_NUM_THREADS=8
real   0m23.123s user   3m3.363s sys   0m0.077s

export OMP_NUM_THREADS=4
real   0m32.916s user   2m10.449s sys   0m0.039s

export OMP_NUM_THREADS=2
real   1m1.807s user   2m3.124s sys   0m0.019s
```

again here we observe full gain in going from 2 to 4 core and a 1.4 gain going from 4 to 8 (notice how the cpu-user time grows of 50%). In this case of course going from 4 to 8 threads does not increase the amount of power used....

results from running CMS reconstruction

Results from running 4 CMS reconstruction jobs in parallel (reading different event files!) has been reported already below: essentially running one job per core takes 4 seconds per event while running 2 jobs on each core takes 6 seconds per event i.e. a net yield gain of 50%.

12/01/2009

first results of HT on intel-i7

running cms Reco on intel-i7

setting affinity to 0,2,4,6 respectively I get

```
tRecoRunV1/report.txt:  <AvgEventTime  Value="6.13458" />
ttRecoRunV2/report.txt:  <AvgEventTime  Value="6.55657" />
ttRecoRunV3/report.txt:  <AvgEventTime  Value="6.04042" />
ttRecoRunV4/report.txt:  <AvgEventTime  Value="6.59123" />
```

while setting it to 0,1,2,3

```
ttRecoRunV1/report.txt:  <AvgEventTime  Value="4.01639" />
ttRecoRunV2/report.txt:  <AvgEventTime  Value="4.28345" />
ttRecoRunV3/report.txt:  <AvgEventTime  Value="3.8462" />
ttRecoRunV4/report.txt:  <AvgEventTime  Value="4.38434" />
```

-- VincenzoInnocente - 02-Mar-2011

This topic: LCG > VIi7BenchMarks
Topic revision: r2 - 2011-03-23 - VincenzoInnocente