HOWTO WRITE FAST NUMERICAL CODE EXERCISE 5

Pascal Spörri
pascal@spoerri.io

April 16, 2013

1 Mini-MMM

The code of this exercise has been run on an Intel Core i5-3570K CPU and Ubuntu 12.04 with gcc 4.6.3 and the compiler flags: `-m64 -march=corei7 -fno-tree-vectorize -O3`. The CPU has a Level 1 cache size of 256 KB (4×32 KB data caches and 4×32 KB instruction caches) with 32 KB L1 data cache available per CPU. The Level 2 cache has a total size of 1 MB with 256 KB cache available per CPU. The Level 3 cache has a size of 6 MB and is shared for all CPUs.

1.1 Blocking for L1 cache

The L1 cache can hold 4096 doubles and has a cache line size of 64 bytes. Hence we have to optimize the inequality:

$$\left\lceil \frac{N_B^2}{B_1} \right\rceil + 3 \left\lceil \frac{N_B M_U}{B_1} \right\rceil + \left\lceil \frac{N_U M_U}{B_1} \right\rceil \leq \frac{C_1}{B_1}$$

1.1.1 code1

$M_U = 2$, $N_U = 2$, $K_U = 1$. The inequality:

$$\left\lceil \frac{N_B^2}{8} \right\rceil + 3 \left\lceil \frac{N_B 2}{8} \right\rceil + \left\lceil \frac{4}{8} \right\rceil \leq \frac{4096}{8}$$

holds an optimal value of $N_B = 61$. Running our code with $N_B = 60$ results in a performance of 0.68 flops/cycle.

1.1.2 code2

$M_U = 2$, $N_U = 2$, $K_U = 2$. The inequality:

$$\left\lceil \frac{N_B^2}{8} \right\rceil + 3 \left\lceil \frac{N_B 2}{8} \right\rceil + \left\lceil \frac{4}{8} \right\rceil \leq \frac{4096}{8}$$

holds an optimal value of $N_B = 61$. Running our code with $N_B = 60$ results in a performance of 1.02 flops/cycle.

1.1.3 code3

$M_U = 1$, $N_U = 8$, $K_U = 2$. The inequality:

$$\left\lceil \frac{N_B^2}{8} \right\rceil + 3\left\lceil \frac{N_B}{8} \right\rceil + \left\lceil \frac{8}{8} \right\rceil \leq \frac{4096}{8}$$

holds an optimal value of $N_B = 62$. Due to blocking we had to run the code with $N_B = 56$ and were able to achieve a performance of 1.30 flops/cycle.

Thus we are able to conclude that `code3.c` performs best for level 1 blocking. This is also supported by the performance plot in figure 1.

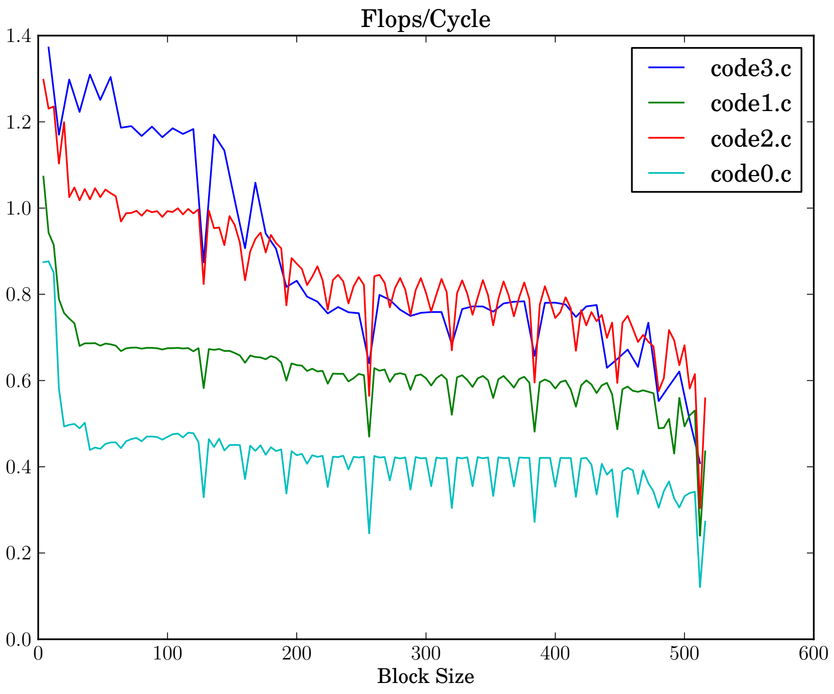

Figure 1: Performance Plot showing flops/cycle for the different codes. Compiler flags: `-m64 -march=corei7 -fno-tree-vectorize -O3` (gcc 4.6.3) on an Intel Core i5-3570K CPU.

1.2 Blocking for L2 cache

The L1 cache can hold 32768 doubles and has a cache line size of 512 bytes. Hence we have to optimize the inequality:

$$\left\lceil \frac{N_B^2}{B_1} \right\rceil + 3\left\lceil \frac{N_B M_U}{B_1} \right\rceil + \left\lceil \frac{N_U M_U}{B_1} \right\rceil \leq \frac{C_1}{B_1}$$

1.2.1 code1

$M_U = 2$, $N_U = 2$, $K_U = 1$. The inequality:

$$\left\lceil \frac{N_B^2}{64} \right\rceil + 3 \left\lceil \frac{N_B 2}{64} \right\rceil + \left\lceil \frac{4}{64} \right\rceil \le \frac{32768}{64}$$

holds an optimal value of $N_B = 177$. Running our code with $N_B = 176$ results in a performance of 0.65 flops/cycle.

1.2.2 code2

$M_U = 2$, $N_U = 2$, $K_U = 2$. The inequality:

$$\left\lceil \frac{N_B^2}{64} \right\rceil + 3 \left\lceil \frac{N_B 2}{64} \right\rceil + \left\lceil \frac{4}{64} \right\rceil \le \frac{32768}{64}$$

holds an optimal value of $N_B = 177$. Running our code with $N_B = 176$ results in a performance of 0.88 flops/cycle.

1.2.3 code3

$M_U = 1$, $N_U = 8$, $K_U = 2$. The inequality:

$$\left\lceil \frac{N_B^2}{64} \right\rceil + 3 \left\lceil \frac{N_B}{64} \right\rceil + \left\lceil \frac{8}{64} \right\rceil \le \frac{32768}{64}$$

holds an optimal value of $N_B = 179$. Due to blocking we had to run the code with $N_B = 176$ and were able to achieve a performance of 0.94 flops/cycle.

Thus we are able to conclude that `code3.c` performs best for level 2 blocking. This is also supported by the performance plot in figure 1.

2 MMM

We benchmarked our `finalcode.c` program with both block sizes $N_B = 176$ and $N_B = 56$ to show the difference in performance for L1 and L2 cache blocking.

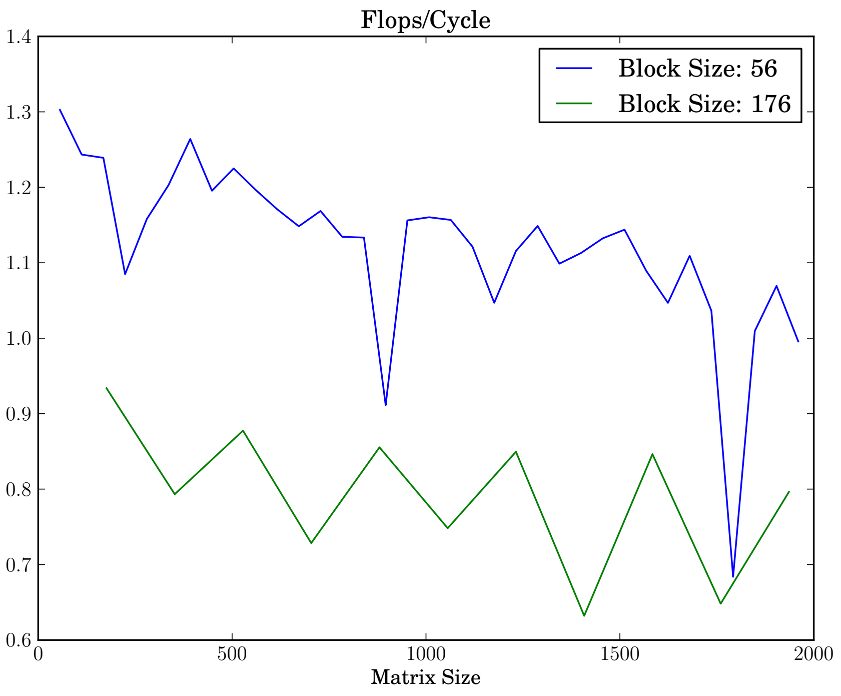

Figure 2: Performance Plot showing flops/cycle for `finalcode.c` with two different block sizes and different matrix sizes. Compiler flags: `-m64 -march=corei7 -fno-tree-vectorize -O3` (gcc 4.6.3) on an Intel Core i5-3570K CPU.