263-2300-00: How To Write Fast Numerical Code

Assignment 1: 100 points

Submitted by Jinank Jain

Solution 1

Part a
Processor Manufacturer: Intel
Processor Name: i7
Processor Number: 3632QM

Part b
CPU Logical Cores: 8
CPU Physical Cores: 4

Part c
CPU Core frequency: 2.2 GHz

Part d
CPU Maximum Frequency: 3.2 GHz.
Yes it does support Intel Turbo Boost Technology (2.0)

Part e
Tick since my processor belongs to the family of Ivy Bridge Processor.

Part f-i

OpType	Latency	Throughput	Gap
Addition	3	1	1
Multiplication	5	2	0.5
rcp	7	0.5	2
FMA	NA	NA	NA

Table 1: Latency/Throughput/Gap for various operations

Part j
Peak performance: 32 flops/cycle and 102.4 Gflops/sec

Solution 2

Part a
Appropriate cost function would involve cost of multiplication, additions and divisions individually. One such example could be following:

$$C(n) = C_{add} * N_{add} + C_{mul} * N_{mul} + C_{div} * N_{div} + C_{typecast} * N_{typecast} \quad (1)$$

Part b

$$C(n) = C_{add} * (2(n-1)+1) + C_{mul} * (6(n-1)+2) + C_{div} * (2(n-1)+2) + C_{typecast} * (2(n-1)+2) \quad (2)$$

$$C(n) = C_{add} * (2n-1) + C_{mul} * (6n-4) + C_{div} * (2n) + C_{typecast} * (2n) \quad (3)$$

$$C(n) = flops(n) = 12n - 5 \quad (4)$$

Solution 3

Part a
This piece of code tries to compute matrix multiplication between two N*N matrix and store the answer in another N*N matrix

Part b:
$C(n)$ = (adds(n), mults(n)) = (n^3, n^3)

$C(n)$ = flops(n) = $2 * n^3$

Part c:
Program was compiled with the following flags:
`-O3 -mno-abm -fno-tree-vectorize -mtune=native`

Part d:
All the values that are used to plot the graph are used from RDTSC register.

- Runtime Plot See Figure 1
- Performance Plot See Figure 2
- Peak performance on a single is around 8 (flops/cycle) and this value is used to plot the percentage of the peak performance See Figure 3

Part e:
If we look at the runtime it behaves pretty much as expected with increasing size of matrix, time required to perform the computation should increase. The computation is compute bound however, due to dependency in the computation the peak performance cannot be achieved. Every iteration of i-loop loads matrix C, thus the performance drops whenever C does not fit in L1 (at n = 60), L2 (above n = 170) and L3 (above n = 836) cache.

Solution 4

Part a

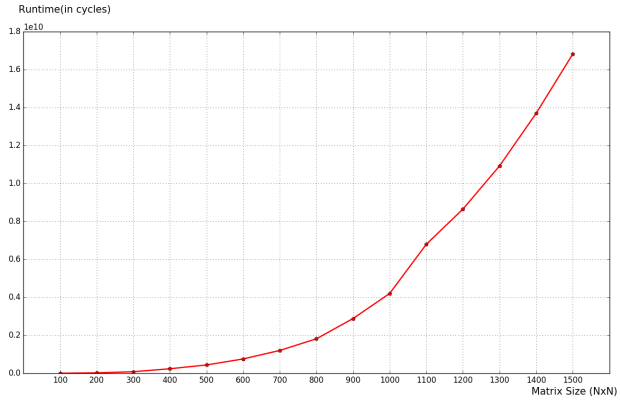

Figure 1: Plots resulting from execution of mmm.c on a IvyBridge CPU (2.2 GHz Intel Core i7-3632QM; 32 kB L1, 256 kB L2, 6 MB L3 cache; scalar peak performance: 2 f/c). The code was compiled with gcc 6.3.1 with O3 enabled and no vectorization.

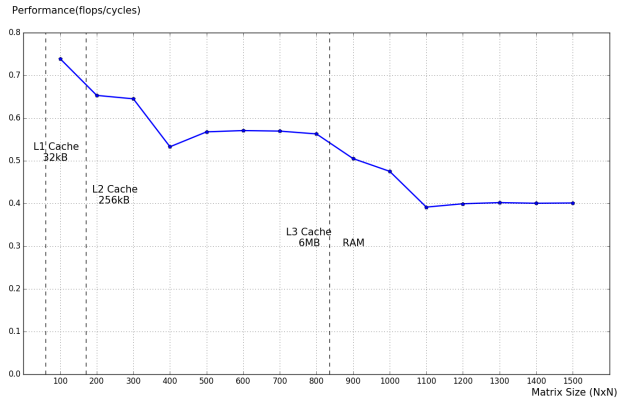

Figure 2: Plots resulting from execution of mmm.c on a IvyBridge CPU (2.2 GHz Intel Core i7-3632QM; 32 kB L1, 256 kB L2, 6 MB L3 cache; scalar peak performance: 2 f/c). The code was compiled with gcc 6.3.1 with O3 enabled and no vectorization.

```
1  inline double babs ( double x) {
2    union { uint64_t i; double d; } u = { .d = x };
3    u.i = u.i & 0x7FFFFFFFFFFFFFFF ;
4    return u.d;
5  }
6
7  void compute() {
8    uint64_t i = 0;
9    for(i=0; i<n; i++) {
10     Z[i]+=X[i]*Y[i]+U[i]*Y[i]+X[i]*Z[i]+babs(U[i]*X[i]);
11   }
12 }
```

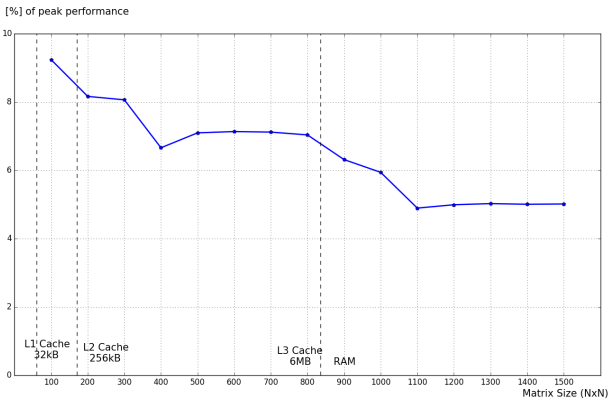

Figure 3: Plots resulting from execution of mmm.c on a IvyBridge CPU (2.2 GHz Intel Core i7-3632QM; 32 kB L1, 256 kB L2, 6 MB L3 cache; scalar peak performance: 2 f/c). The code was compiled with gcc 4.9.3 with O3 enabled and no vectorization.

Part b
Like in exercise 3 we used RDTSC instruction to measure the number of cycles, in this part I am using the same instruction to count the number of cycle.

Part c Performance was measured in four different scenario:

- Program was compiled -O3 -mno-abm -fno-tree-vectorize -mtune=native and turbo boost enabled
- Program was compiled -mno-abm -fno-tree-vectorize and turbo boost enabled
- Program was compiled -O3 -mno-abm -fno-tree-vectorize -mtune=native and turbo boost disabled
- Program was compiled -mno-abm -fno-tree-vectorize and turbo boost disabled

- In Figure 4 runtime is plotted
- In Figure 5 performance is plotted

Part d
The code is always bound by data movement

- When data fits in L1, performance is bound by 1.5 f/c. On Ivy Bridge we can either issue two loads or one load and one store at any given cycle (on port 2 and 3 of the execution core) with an average bandwidth of 1.5 doubles per cycle. The runtime for the computation is therefore approximately 4n doubles/1.5 doubles/c = 2.66n cycles
- When data no more fits in L1 it goes to L2. L2 and L3 have almost the same throughput that's why in the graph we don't see any significant fluctuation in the graph in between L2 and L3.

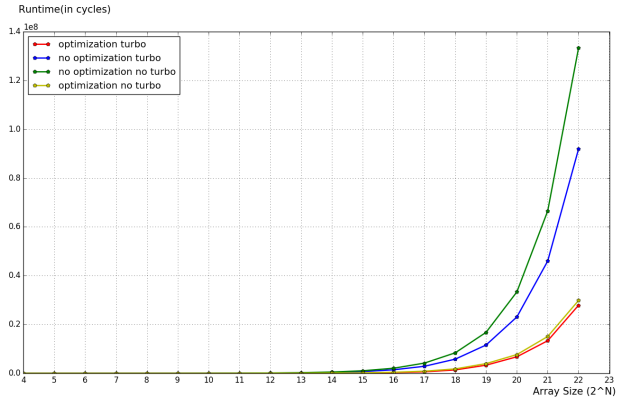

Figure 4: Plots resulting from execution of combine.c on a IvyBridge CPU (2.2 GHz Intel Core i7-3632QM; 32 kB L1, 256 kB L2, 6 MB L3 cache; scalar peak performance: 2 f/c). The code was compiled with gcc 6.3.1

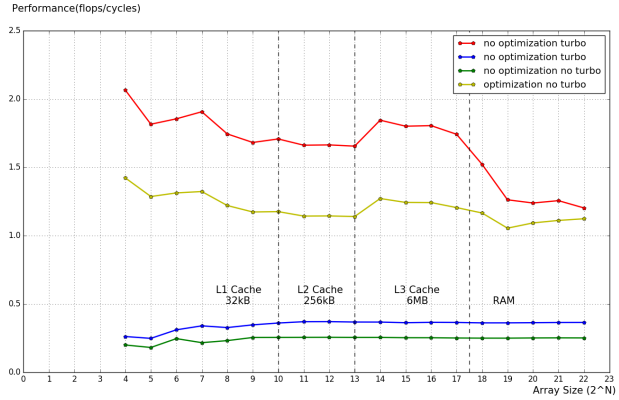

Figure 5: Plots resulting from execution of combine.c on a IvyBridge CPU (2.2 GHz Intel Core i7-3632QM; 32 kB L1, 256 kB L2, 6 MB L3 cache; scalar peak performance: 2 f/c). The code was compiled with gcc 6.3.1

- When does not fit in L3 we see a significant drop in performance when we start fetching data from RAM.

Effect of Turboboost
When turbo boost is turned on we see a significant increase in performance as it is able to operate at higher frequency.

Effect of Optimization Flags
Optimization flags have more effect than turbo boost which is pretty much evident from the Figure 5.

Solution 5

Part a

- `artcomp1`: 2N flops
- `artcomp2`: N flops
- `artcomp3`: N flops

Part b

- `artcomp1`:
  - Flops = 2N
  - Memory Transfers (floats) $\geq$ 2N
  - Read (bytes) $\geq$ 8N
  - Operational Intensity I(N) $\leq \frac{1}{4}$
- `artcomp2`:
  - Flops = N
  - Memory Transfers (floats) $\geq$ 2N
  - Read (bytes) $\geq$ 8N
  - Operational Intensity I(N) $\leq \frac{1}{8}$
- `artcomp3`:
  - Flops = N
  - Memory Transfers (floats) $\geq$ 2N + 2/3N
  - Read (bytes) $\geq$ 8N + 8/3 N
  - Operational Intensity I(N) $\leq \frac{3}{32}$

Part c
i

- `artcomp1`: N/2
- `artcomp2`: N
- `artcomp3`: 2N/3

ii

- Order L1, L2, L3, RAM
- `artcomp1`: N/4, N/4, N/2, N
- `artcomp2`: N/4, N/8, N/4, N/2
- `artcomp3`: N/4, 5N/24, 5N/12, 5N/6