18-640: Foundations of Computer Architecture

**Project 4**

Group NO.7 Luting Wang (lutingw) Wei Feng (wfeng1)

Part 1: Matrix Multiplication

In this part, we are going to analyze four different implementations’ performance for matrix multiplication: single thread vs. multiple thread (Thread-Level Parallelism), and scalar processing vs. vector processing (Data-Level Parallelism). Then we use Intel VTune to collect data. The runtime characteristics of these implementations are shown below.

1. Results

Table 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | mmm\_single thread\_scalar | mmm\_single thread\_simd | mmm\_multi thread\_scalar | mmm\_multi thread\_simd |
| Elapsed time\* | 31.102s | 4.838s | 9.922s | 2.270s |
| CPU time\* | 30.623s | 4.742s | 28.675s | 6.976s |
| Effective time\* | 30.623s | 4.742s | 28.675s | 6.849s |
| CPU Frequency ratio | 1.354 | 1.323 | 1.187 | 1.192 |
| CPI | 1.246 | 3.033 | 1.000 | 3.987 |
| Context Switch Time | 0.048s | 0.013s | 1525.431s | 299.888s |
| Instruction type consuming max. time | fmulp  11.015s | mulps  2.117s | faddp  2.736s | mulps  2.850s |
| Branch misprediction rate | 0.006 | 0.000 | 0.021 | 0.002 |
| Cache Hits | 0.247 | 0.161 | 0.024 | 0.133 |
| Cache Misses | 0.009 | 1.000 | 0.000 | 1.000 |

1. Questions
2. **What can we infer from the** **CPU usage histogram, for different versions of the code?**

For single thread implementation, only one CPU is used, and the one with SMID has less elapsed time which means that Data-Level Parallelism will speed up CPU computation. And for multiple thread implementations, 4 cores are used and the time for different CPU working simultaneous is different. For the scalar one, the time with only one or two CPUs working is about 40%, which means that for about 40% of total time CPUs are working idly or poorly. However, if we apply Thread-Level Parallelism, for much of the time 3 or 4 CPUs are working, which means CPU utilization is high, as you can see in following figures.

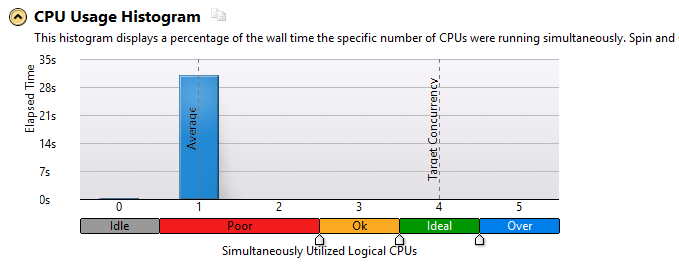


Figure 1 mmm\_single\_thread\_scalar CPU usage histagram

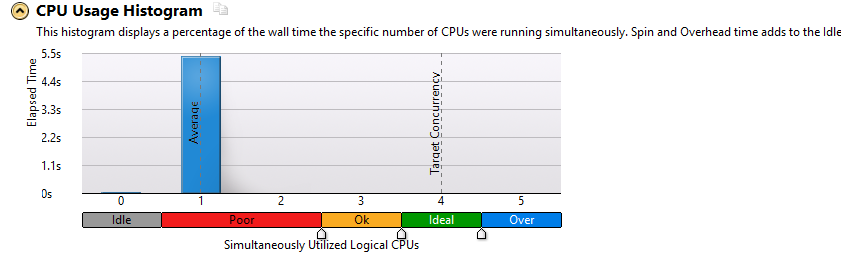


Figure 2 mmm\_single\_thread\_simd CPU usage histogram

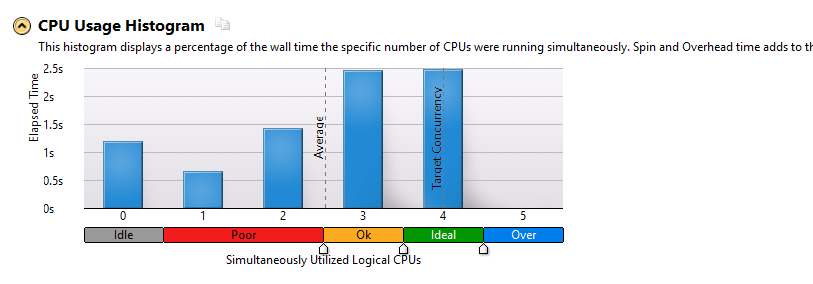


Figure 3 mmm\_multi\_thread\_scalar CPU usage histogram

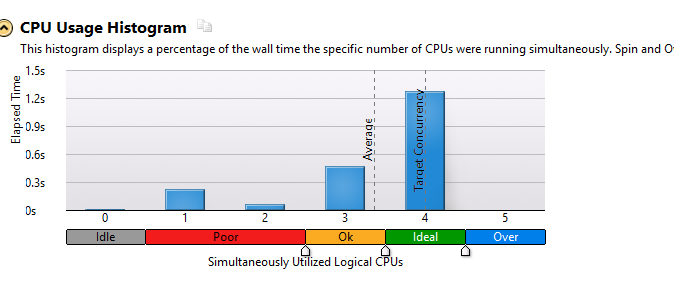


Figure 4 mmm\_multi\_thread\_simd CPU usage histogram

1. **Which instruction type is consuming the most number of cycles? What can be the possible solutions to reduce this?**

For most of implementations, multiply instruction is consuming the most number of cycles. To solve this, we can implement Instruction-Level Parallelism, for example, to overlap the execution of multiple instructions or even to change the order in which instructions are executed.

1. **What can you say about the CPI and** **CPU frequency ratio for each execution? How does it vary?**

For CPI, the implementation without Data-Level Parallelism has fewer CPI than that with Data-Level Parallelism. And the implementation only with Thread-Level Parallelism has the least CPI (1.0), and the implementation with both Thread-Level Parallelism and Data-Level Parallelism has the largest CPI (3.987). For CPU frequency ratio, the implementation with Thread-Level Parallelism has less CPU frequency ratio than that without Thread-Level Parallelism, and it seems that Data-Level Parallelism has little impact on CPU frequency ratio. So we can infer that with Data-Level Parallelism, CPI will increase, but CPU frequency ratio will not be effected significantly; while with Thread-Level Parallelism, CPI will not be effected obviously, but CPU frequency ratio will decrease.

1. **Modern processors execute many more instructions than the program flow needs. This is called "speculative execution. Are all instructions retired?** **Give appropriate reason with readings from VTune analysis.**

I think not all the instructions has retired because of speculative execution. The number of instructions that are retired is the amount of work that has been done, and only the instructions that are speculated correctly will be retired. The implementation of mmm\_single thread\_scalar has 44,528,200,000 instructions retired, while mmm\_multi thread\_scalar has 32,459,400,000 instructions retired, which shows a big difference. And the implementation of mmm\_single thread\_simd has 2,795,200,000 instructions retired while mmm\_multi thread\_simd has 2,799,600,000 instructions retired, and their difference is small. And according to Intel® VTune™ Amplifier’s “Use Instructions Retired Events to Evaluate Threading Methodology”[1], it says: instructions-retired data can be used to examine the correctness of threading methodology. When an application is threaded and run in parallel, the amount of work that gets accomplished is roughly still the same, so you should see roughly the same number of instructions retired. So we think there may be some threading problems.

Part 2: Edge Detection

In this part, we are going to analyze multiple implementations’ performance for edge detection using OpenCV libraries. The collected data are following:

1. Results

Table 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | st\_nsse | st\_sse | mt\_nsse | mt\_sse | mt\_sse\_ocl |
| Elapsed time\* | 83.563s | 54.791s | 30.843s | 25.449s | 58.222s |
| CPU time\* | 67.395s | 47.793s | 83.166s | 65.730s | 95.091s |
| Effective time\* | 67.392s | 47.792s | 83.083s | 65.650s | 87.124s |
| CPU Frequency ratio | 1.302 | 1.335 | 1.210 | 1.191 | 1.202 |
| CPI | 2.583 | 2.078 | 2.946 | 2.494 | 2.247 |
| Context Switch Time | 13.374s | 3.872s | 36.199s | 31.528s | 154.691s |
| Instruction type consuming max. time | add  32.340s | add  32.352s | add  39.173s | add  32.722s | add  30.995s |
| Branch misprediction rate | 0.011 | 0.012 | 0.010 | 0.010 | 0.018 |
| Cache Hits | 0.133 | 0.287 | 0.112 | 0.140 | 0.102 |
| Cache Misses | 0.277 | 0.287 | 0.380 | 0.512 | 0.376 |

1. Questions
2. **Describe the impact of** **context switch time consumptions and wait times?**

Context switch time has impact on performance, due to running the task scheduler, TLB flushes, and indirectly due to sharing the CPU cache between multiple tasks [2]. It includes wait time and inactive time. Wait time is used for measuring the percentage of time spent waiting for CPU resources to service a thread. If the wait time is long, it means CPUs are been overused.

1. **How much does** **branch misprediction affect the execution times of the program?**

If branch misprediction rate is high, it means program needs more cycles to execute on correct path, so the execution time will increase. Modern processors use multiple CPUs to decrease execution time and increase performance. It executes instructions in parallel and before the time when their results are actually needed, so it performs branch prediction. But if the branch is predicted wrongly, it need to clears the related instructions and re-execute on the correct path. So to decrease execution times, we need to balance parallelism and branch misprediction rate.

1. **Memory: What are the aspects with respect to memory that can be taken care of in order to improve** **performance? Report the parameters that are crucial with respect to memory.**
   1. Memory capacity
   2. Cache coherence protocols
   3. Interconnection topology
   4. Memory Hierarchy
2. **What is****Page Walk? How does it influence performance?**

Page walk is a process when the requested address is not in TLB, virtual address space to physical address space translation is proceeded by looking up the page table. It influences performance by increasing time. Because it reads the contents of multiple memory locations, requiring several loads to compute the physical address.

Part 3: Digit Detect

In this part, we are going to analyze one implementation’s performance for line detection using OpenCV libraries. The collected data are following:

1. Results

Table 3

|  |  |
| --- | --- |
|  | st\_digit |
| Elapsed time\* | 2123.143s |
| CPU time\* | 2054.880s |
| Effective time\* | 2054.865s |
| CPU Frequency ratio | 1.362 |
| CPI | 0.890 |
| Context Switch Time | 23.537s |
| Instruction type consuming max. time | faddp  383.320s |
| Branch misprediction rate |  |
| Cache Hits |  |
| Cache Misses |  |

1. Questions
2. **What are front-end and back-end executions in superscalar processors?**

In superscalar processors, front end fetches and dispatches instructions in program order, while back end completes and retires instructions in program order [3].

1. **Apart from mentioned, describe at least two other key parameters that can be used from VTune to transform our programs performance?**

We can also use Wait Rate and Call Count to evaluate our programs’ performance.

Part 4: Bonus

References:

[1] <https://software.intel.com/en-us/articles/use-instructions-retired-events-to-evaluate-threading-methodology/>

[2] <https://en.wikipedia.org/wiki/Context_switch>

[3] J.P.Shen. Modern Processor Design