

CS213 Homework 2

SPLASH Performance Profile on SGI Machine

Danhua Guo
dguo@cs.ucr.edu

1 Environment

SGI server: 64X Intel Itanium 2 Processor with 64GB RAM

2 Benchmark

SPLASH2 is used as the benchmark for this experiment. More specifically, I chose 5 applications in the benchmark. The functionality of each application is described as follows:

Contiguous partition allocation program is one of the two applications provided with the *OCEAN* package. This implementation (contained in the *contiguous_partitions* subdirectory) implements the grids to be operated on with 3-dimensional arrays. The first dimension specifies the processor which owns the partition, and the second and third dimensions specify the x and y offset within a partition. This data structure allows partitions to be allocated contiguously and entirely in the local memory of processors that "own" them, thus enhancing data locality properties.

BARNES application implements the Barnes-Hut method to simulate the interaction of a system of bodies (N-body problem). The *SPLASH-2* implementation allows for multiple particles to be stored in each leaf cell of the space partition.

FMM application implements a parallel adaptive Fast Multipole Method to simulate the interaction of a system of bodies (N-body problem).

WATER-NSQUARED is an improvement over the original Water code in *SPLASH*, but is mostly the same. The best source of descriptive information therefore is the original *SPLASH* report. The main change is that the locking strategy around the updates to the water accelerations (in *interf.C*) is improved: a process updates a local copy of the relevant particle accelerations, and then accumulates into the shared copy once at the end.

WATER-SPATIAL solves the same molecular dynamics N-body problem as the original Water code in *SPLASH* (which is called *WATER-NSQUARED* in *SPLASH-2*), but uses a different algorithm. In particular, it imposes a 3-d spatial data structure on the cubical domain, resulting in a 3-d grid of boxes. Every box contains a linked list of the molecules currently in that box (in the current time-step). The advantage of the spatial grid is that a process that owns a box in the grid has to look at only its neighboring boxes for molecules that might be within the cutoff radius from a molecule in the box it owns. This makes the algorithm $O(n)$ instead of $O(n^2)$. For small problems (upto several hundred to a couple of thousand molecules) the overhead of the spatial data structure is not justified and *WATER-NSQUARED* might solve the problem faster. But for large

systems this program is much better. That is why we provide both, since both small and large systems are interesting in this case.

All access to molecules is through the boxes in the spatial grid, and these boxes are the units of partitioning (unlike *WATER-NSQUARED*, in which molecules are the units of partitioning).

3 Performance Metrics

In this paper, performance is measured with execution time and instruction per cycle (IPC). For *OCEAN*, cache misses information is also provided. In order to compare performances under different set up, the number of processor is changed with different parameters in the benchmark.

4 Profiling Tool

The Performance API (PAPI) specifies a standard application programming interface (API) for accessing hardware performance counters available on most modern microprocessors. These counters exist as a small set of registers that count *Events*, occurrences of specific signals related to the processor's function. Monitoring these events facilitates correlation between the structure of source/object code and the efficiency of the mapping of that code to the underlying architecture. This correlation has a variety of uses in performance analysis including hand tuning, compiler optimization, debugging, benchmarking, monitoring and performance modeling. In addition, it is hoped that this information will prove useful in the development of new compilation technology as well as in steering architectural development towards alleviating commonly occurring bottlenecks in high performance computing.

5 Result

The performance results of all the applications are presented with different metrics, including program execution time, IPC and cache miss rates. Of these performance presentations, the execution time is obtained from the benchmark, while the rest are all generated with *PAPI*. Essentially with some additional *PAPI* code and libraries, it is not very difficult to get some more profiling information by further study into the *EventSet* in *PAPI*.

5.1 OCEAN-CON

5.1.1 Time and IPC

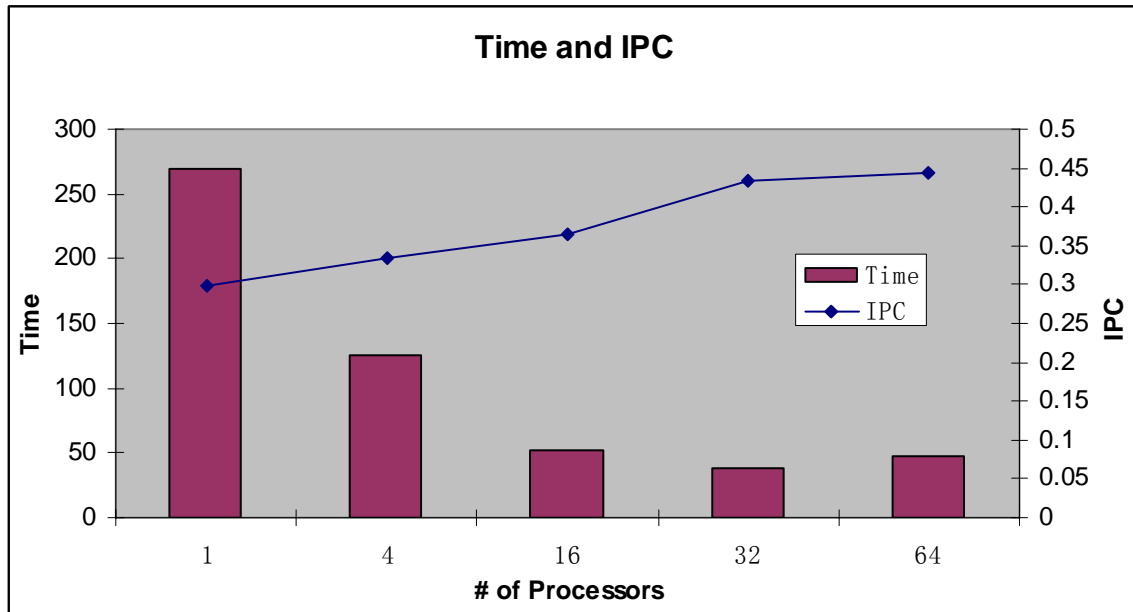


Figure 1

Figure 1 shows that the execution time decreases with the number of processors N becomes larger. This trend is not very obvious when N is greater than 16. It is possible that the processing improvement provided with more processors is offset by the time cost of context switches. Meanwhile, the IPC is proportional to the number of processors.

5.2.2 Cache Misses

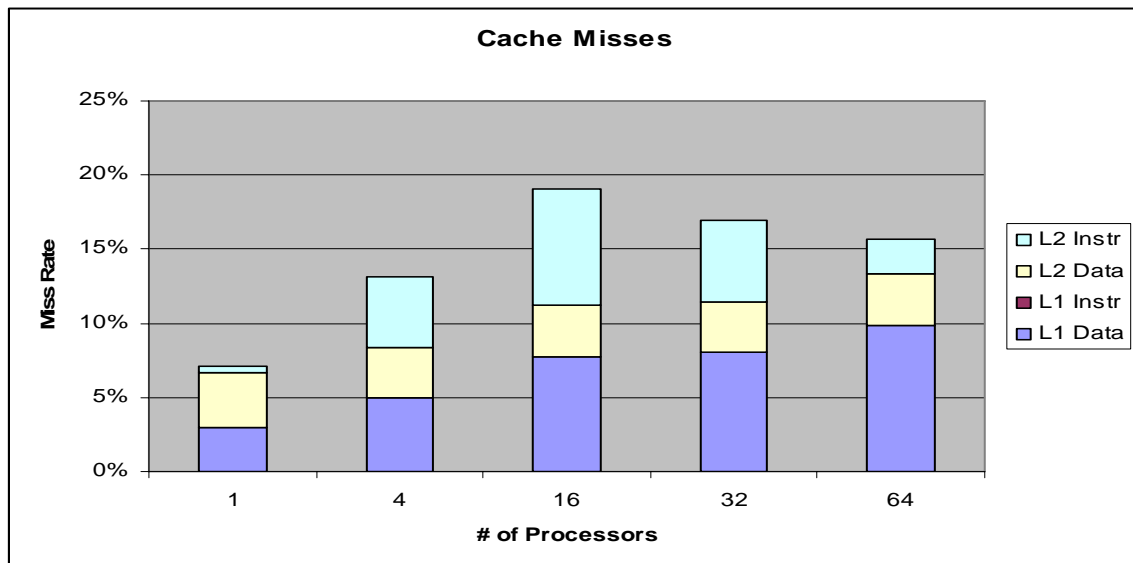


Figure 2

Figure 2 gives an overview about cache information. As shown in the diagram, cache misses happen most often when $N=16$. It is understandable that with a relative smaller number of processors working at the same time, the chances for overall cache misses also reduces. If we consider data cache independent of instruction cache, we will find L1 data cache has a higher rate of misses when N increases, whereas L1 instruction cache miss rate keeps at a very low level. As to L2 caches, the instruction cache follows the general pattern of the overall cache misses while L2 data cache remains at a stable miss rate.

5.2 BARNES

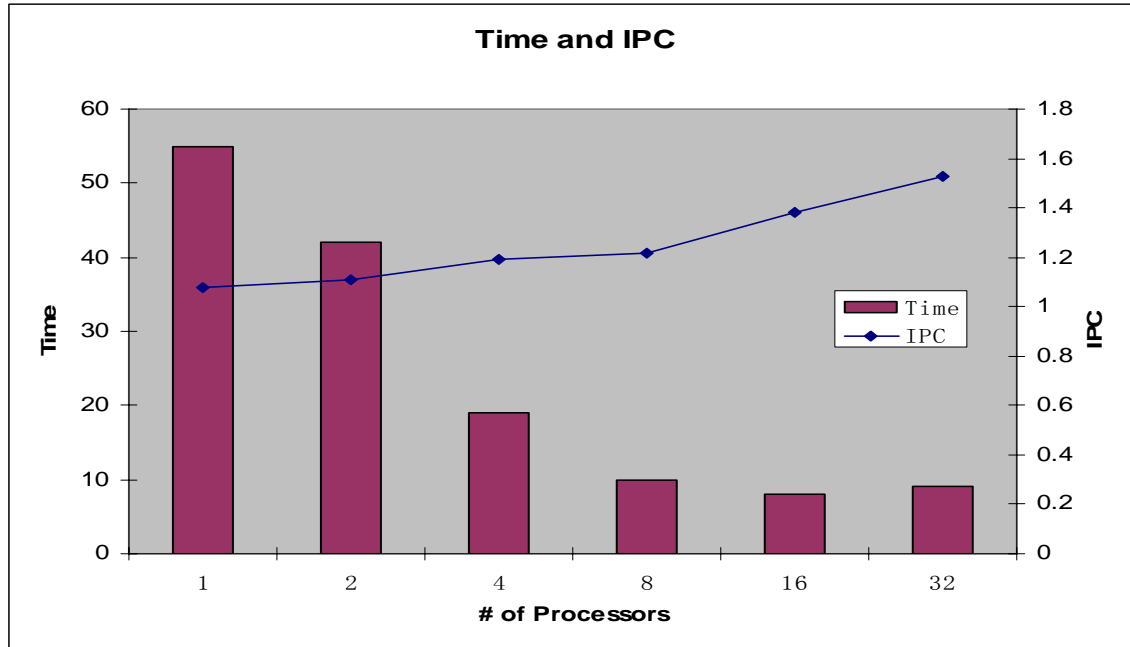


Figure 3

The *BARNES* application performance follows that of the *OCEAN*, namely a decreasing pattern of execution time together with an increasing IPC if the number of processors is increased. Same reason follows as the previously discussed.

5.3 FMM

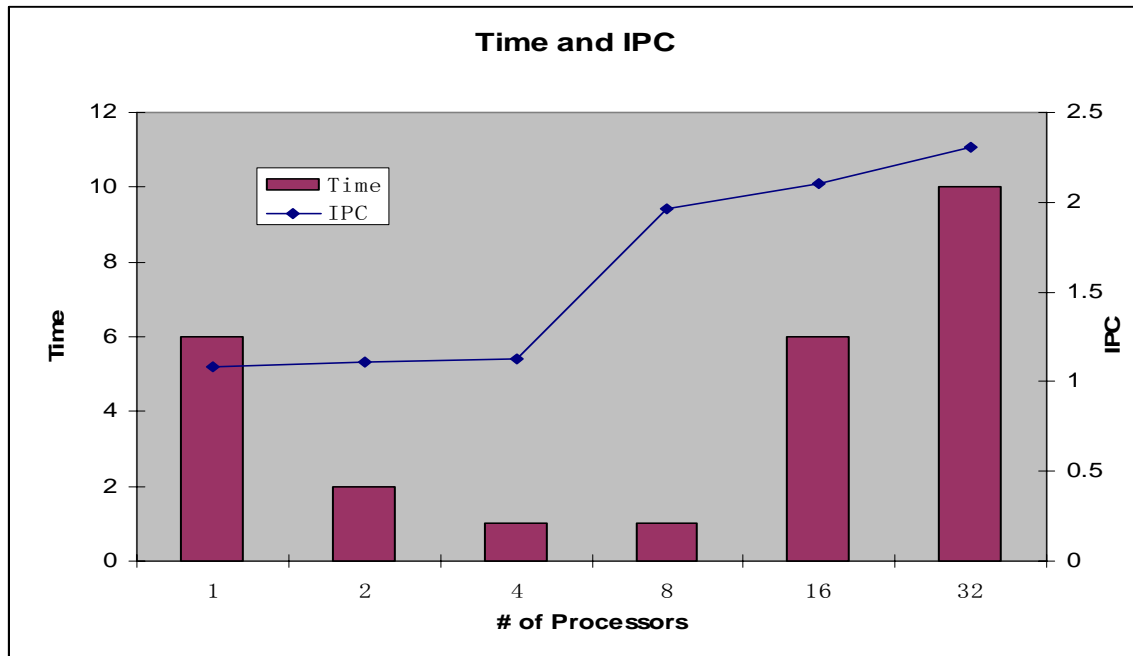


Figure 4

The performance of *FMM* is the only one in this experiment that does not follow the general pattern. While the execution time drops when the number of processors increases to 8, it takes more time to finish if this number continues to increase. When $N=32$, its execution time is even worse than that of 1 processor. This exception may be explained by its unique context switching protocol, which takes a relatively large time if more threads are involved.

5.4 WATER-NSQURED and WATER-SPATIAL

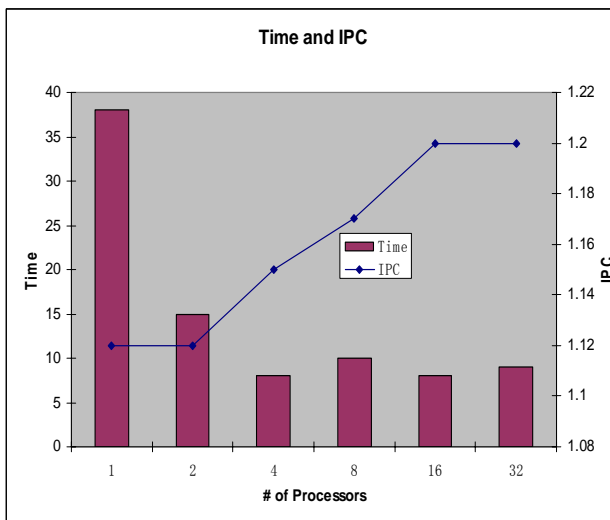


Figure 5

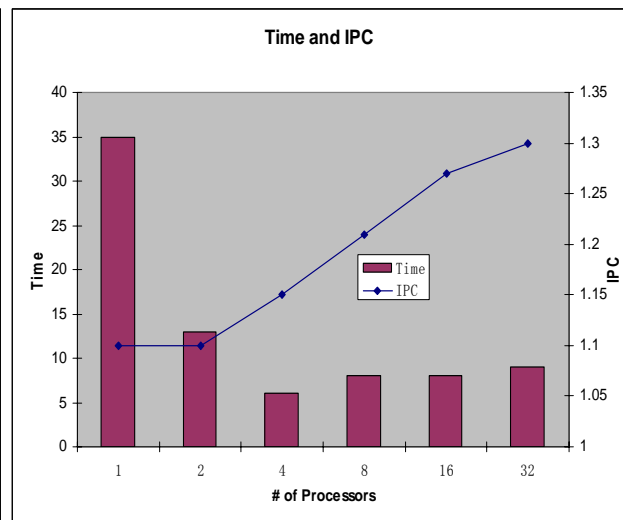


Figure 6

Figure 5 and 6 shows the performance of *WATER-NSQURED* and *WATER-SPATIAL* respectively. Although IPC follows the general rule in relation to the increasing number

of processors, execution time reaches the best performance when $N=8$. One possible explanation is that due to a large amount of data dependencies, communication costs takes a greater percentage of the overall execution time.

6 Conclusion

With the result of experiment presented in the previous section, it can be seen that SGI provides different computation performance with different set up. A general pattern of the performance is that the execution time drops quickly when the number of processors increases. But this trend slows down when the number reaches a bottle neck due to context switch costs. At the same time, IPC performance increases at a steady rate. If only 1 processor is allowed for the program, SGI receives a relatively larger delay in execution. This may be explained by the CC-NUMA architecture of SGI which introduces a non-uniformed memory access time if a certain processor is chosen for the program.

In addition, since there were a lot of server accesses in the department during the weekend, the result shown in this paper might be influenced with a temporal factor, which requires more intense data collection in the future.