# Programming of Supercomputers
# Assignment 1

Marco Seravalli - 03626387

November 1, 2012

## Execution environment

All the test have been performed on the Linux cluster hosted by LRZ in the MPP partition. The technical specifications of the execution environment can be found in Table 1.

| Parameter | Value |
|---|---|
| Number of Sockets per Node | 2 |
| Number of Cores per Socket | 8 |
| Hyperthreading | No |
| Threads per Core | 1 |
| Number of Threads per Socket | 8 |
| Peak Frequency per Core | 2000MHz |
| Peak Performance per Core | ~8GFlops |
| Peak Performance per Node | ~128GFlops |
| Frequency Scaling | 800, 1100, 1400, 1700, 2000 MHz |
| L3 Cache per Socket (shared) | 12MB (2 x 6MB) |
| L2 Cache per core | 512KB |
| L1 Cache per core | 64KB instruction + 64 KB data |
| Memory per node | 16GB |

Table 1: Execution Environment

## Metrics

For the assignment Cache Miss Rate for L1 and L2 cache, $MFlops$ and CPU utilization needed to be measured.

The Cache Miss Rates are given by

$$Cache\,Miss\,Rate = \frac{Total\,Cache\,Misses}{Total\,Cache\,Accesses}$$

the Total Cache Misses are given by

$$Total\,Cache\,Misses = Data\,Cache\,Misses + Instruction\,Cache\,Misses$$

similarly the Total Cache Accesses can be derived by

$$Total\,Cache\,Accesses = Data\,Cache\,Accesses + Instruction\,Cache\,Accesses$$

The *Flops* (**Fl**oating-point **o**perations **p**er **s**econd) measure the number of floating point operations per second, the result can be achieved by taking the ratio between the execution time and the floating point operations performed in that period:

$$MFlops = \frac{Floating - point\,Operarations}{Wallclock\,Time\,(\mu sec)}$$

Lastly, the CPU utilization is given by

$$CPU\,Util\ =\ \frac{Total\,Cycles\ \times\ Clock(sec)}{Wallclock\,Time(sec)}$$

# Results

## Peak Performance Comparison

The system has a theoretical maximum performance of almost $8GFlops$ per core, while the application's peak is of $0.586GFlops$ i.e. $7.35\%$ of the theoretical peak.

## Compiler Optimization

Different compiling optimizations have been applied to the provided code. In Figure 1 it is possible to see the effect of every flag.

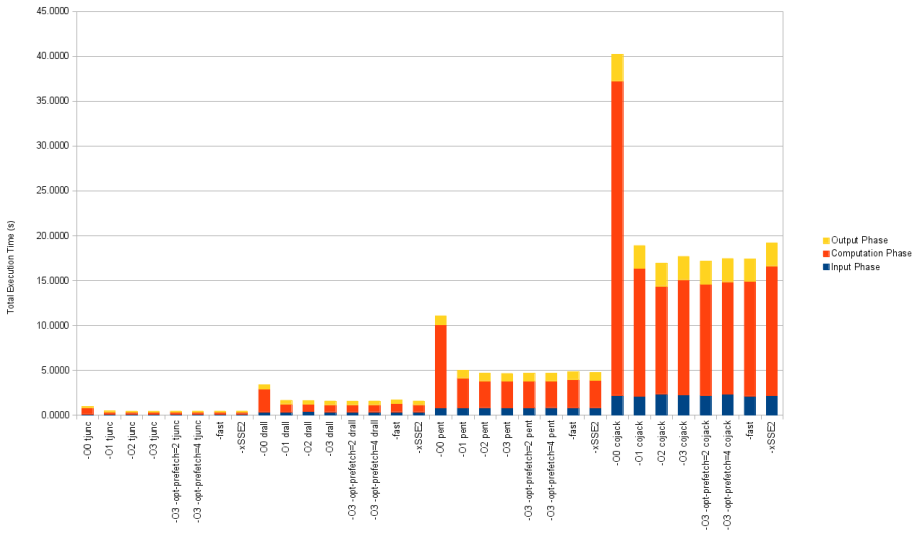


Figure 1: Compiler Optimization

In addition to the proposed optimization flags also $-fast$ was adopted, this adds to $-O3$ other improvements like $-ipo$, $-static$, $-no-prec-div$ and $-xHOST$. Unfortunately, $-fast$ does not introduce a significant speedup, in some cases there is a regression if compared to $-O3$.

Lastly, also the automatic vectorisation was tested by adopting the $-xSSE2$ flag. With this option the compiler adopts whenever possible vector instructions.

### Cache Optimization

With respect to the L2 cache the optimization flags increase the Cache Miss Rate in all given scenarios as it can be seen in Figure 2.

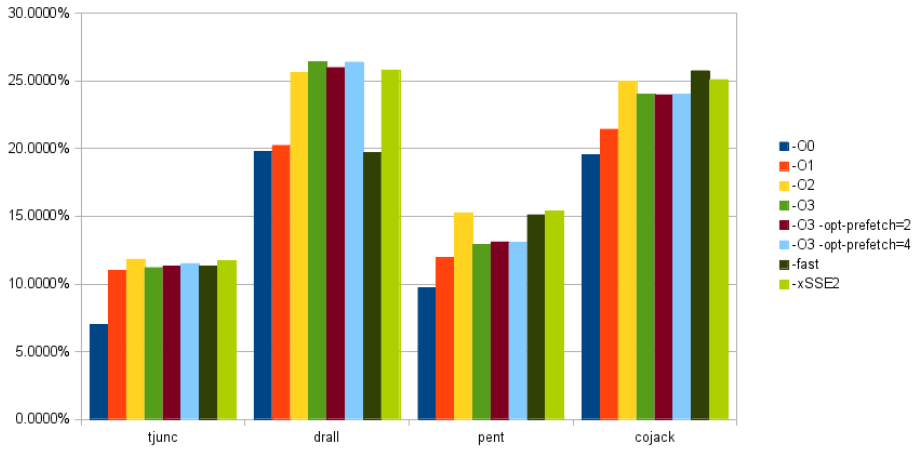


Figure 2: Cache Optimization

Also the prefetching solution does not yield to any visible improvements. On the contrary, in particular for $-opt - prefetch = 4$ the Miss Rate actually increases, this could be due to the fact that when prefectching new data, some cache lines that could have been useful are actually invalidated, causing additional overhead.

## Binary Input

One of the aims of the assignment was to test different behaviors when using a binary input instead of a text input.

The difference between these two representations consists in the fact that the first one stores only the binary representation of the numbers, while the latter stores the ASCII representation of the numbers plus additional spaces for better legibility.

The advantages of the binary representation are a smaller size of the file and a better performance since no translation from text to binary needs to be performed, the major drawback is the complete loss of legibility.

The correlation between file size and reading time can be seen in Fig.3.

By using binary representation the size of the files is reduced by a factor of 3 in all scenarios, moreover also the reading time drops dramatically: the average speedup is 10×. The low performance of text files is due to bigger file size and to the additional overhead that text involves, i.e. seeking for the next number and translating from text to binary representation.

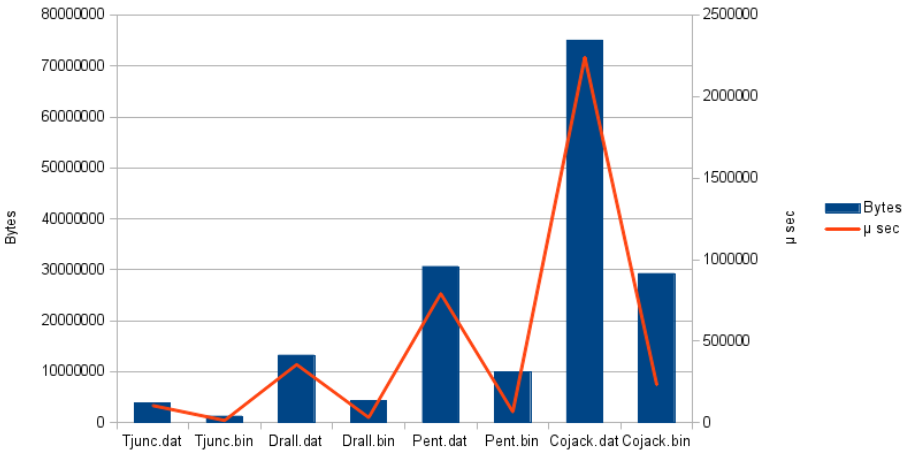


Figure 3: Text-Binary Reading