Mid Semester Examination *(April 2014)*

High Performance Scientific Computing  
*(COMP3320/COMP6464 )*

*Writing period : 90 Minutes duration  
Study period : 15 Minutes duration  
Permitted materials : NONE*

The questions are followed by labelled, framed blank panels into which your answers MUST be written. Extra boxes are given at the end of the paper. No additional paper will be provided. Writing outside of the boxes may not be marked.

This exam is marked out of 60. You must answer all questions.

Your mark for this exam will contribute at most 15% of your total course mark, according to the marking scheme given on the course web page.

Student Number:

*The following are used by your friendly examiner!*

<table>
<thead>
<tr>
<th>Q1 mark</th>
<th>Q2 mark</th>
<th>Q3 mark</th>
<th>Q4 mark</th>
</tr>
</thead>
</table>

Total mark
Question 1: Timing [15 total marks]

(a) In the standard distributions of Linux/Unix the following two library calls:

```c
#include <sys/times.h>
clock_t times(struct tms *buf);

#include <sys/time.h>
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

are used to obtain the “process time” and the “wall clock time” respectively.

(i) What is meant by “process time” and “wall clock time”?

(ii) In timing your application code, when would you use process time and when would you use wall clock time?

(iii) The process time is usually broken down into “user” and “system” components. What is included in each of these components?

[7 marks]
(b) When using a timer to measure the performance of your application code, it is important to know the “resolution” and “overhead” associated with that timer.

(i) Define timer resolution and timer overhead.

(ii) Timers that report process time typically have a lower resolution and a higher overhead than timers that report wall clock time. Explain why this is and provide typical values for the resolution and overhead associated with each type of timer.

[8 marks]
Question 2: CPU Architecture [15 total marks]

(a) Modern processors make extensive use of pipelining. Explain the concept of pipelining using an example that is unrelated to computing.

[2 marks]

(b) Data dependencies impact on the ability to pipeline.

(i) Give a few lines of code containing a for loop where there is a data dependency between iterations of the loop. Clearly indicate the dependency.

[4 marks]

(ii) Explain how a loop can contain a dependency between different iterations of the loop but can still be amenable to pipelining and producing the correct answers.
(c) Pipeline throughput depends on the number of independent operations and the pipeline depth

(i) Define what pipeline depth means

(ii) Sketch (and label) in the following graph the pipeline throughput $N/T_{pipe}$ as a function of $N$ independent operations for:
   (A) a small size pipeline depth
   (B) a medium size pipeline depth
   (C) a large size pipeline depth

[4 marks]
(d) Name three concepts other than pipelining that have been developed to improve processor performance and briefly describe how they work.

[5 marks]
Question 3: Memory architecture [15 total marks]

(a) Sketch C code that illustrates i) a cache friendly operation ii) a cache unfriendly operation. In both cases provide a brief explanation as to why you consider the code to be cache friendly/unfriendly.

(b) Provide two fundamentally different reasons why a data item previously referenced and brought into cache may no longer be present in cache when referenced a second time.
(c) Explain what the following terms mean:

(i) cache line

(ii) cache thrashing

(iii) cache miss

[1.5 marks]
(d) To test the memory architecture of an unknown processor, you have written a program that allocates an array of size \( w \) (defined as working set size) and then accesses elements in this array with constant stride \( s \). The array is allocated on a 4 byte boundary and is accessed at 4 byte words. Shown below are measured read throughputs.

(i) Explain the observed read throughput in terms of the memory architecture in this processor.
Question 4: Performance tuning [15 total marks]

(a) Shown side by side below are two versions of functionally identical C code.

```c
void f1(int n, double a[], double b[],
        double c[], double d[],
        double y[], double z[])
{
    int i;
    for (i = 0; i < n; i++) {
        y[i] = a[i] / b[i];
        z[i] = c[i] / d[i];
    }
}
```

```c
void f1(int n, double a[], double b[],
        double c[], double d[],
        double y[], double z[])
{
    int i;
    double tmp;
    for (i = 0; i < n; i++) {
        tmp = 1.0 / (b[i] * d[i]);
        y[i] = a[i] * d[i] * tmp;
        z[i] = c[i] * b[i] * tmp;
    }
}
```

(i) Count data load, data store and floating point operations in both versions, and discuss expected performance differences.

[i marks]
(b) Rewrite the following C code snippets to improve serial performance on an Intel i7 processor (similar to what you use on raijin) without changing code functionality, and reason why your change improves performance. Assume that all vectors are of size \([n]\), and all matrices are of dimension \([n][n]\).

(i) vector operations

\[
\begin{align*}
    m &= \frac{n}{2} \\
    &\text{for } (i = 0; i < n; i++) \{ \\
    &\quad a[i] = b[i] \times c[i] \\
    &\} \\
    &\text{for } (j = 0; j < m; j++) \{ \\
    &\quad z[j] = b[j] \times y[j] \\
    &\}
\end{align*}
\]

(ii) matrix transpose

\[
\begin{align*}
    &\text{for } (i = 0; i < n; i++) \{ \\
    &\quad \text{for } (j = 0; j < n; j++) \{ \\
    &\quad \quad a[j][i] = b[i][j] \\
    &\quad \} \\
    &\}
\end{align*}
\]
(c) Shown below are two functionally identical C code snippets of the core loops in a Jacobi algorithm. Assume a’s dimension to be a[2][n][n] and that the compiler does not change any instructions in the code you see below. Discuss the performance of the two codes as a function of the data size and hardware.

```c
// First code snippet

float t1, t0;
for (i = 1; i < n-1; i++) {
    for (j = 1; j < n-1; j++) {
        t1 = 1;
        for (i = 1; i < n-1; i++) {
            for (j = 1; j < n-1; j++) {
                a[t1][i][j] = 0.25*a[t0][i][j+1] + 0.25*a[t0][i][j-1] +
                0.25*a[t0][i+1][j] + 0.25*a[t0][i-1][j];
            }
        }
        t0 = 0;
        t1 = 1;
    }
}
```

[6 marks]
Continuation of Question [ ] Part [ ]

Continuation of Question [ ] Part [ ]