TMA4280 Introduction to Supercomputing

Problem Set 2

Einar M. Rønquist

Spring 2012

Exercise 1

The previous supercomputer at NTNU, `njord`, was based on the POWER5 dual-core chip. The two processors on a single chip each had a private L1 data cache of size 32 kB (kilobytes), a shared L2 cache of size 1.875 MB (megabytes), and an off-chip L3 cache of size 36 MB. Assume that we need to store floating point numbers in double precision.

- How many floating point numbers can fit in each of the caches?
- What is the dimension of the largest square matrix that we can fit in each cache? Compare this with Exercise 6 in Problem Set 1.

Exercise 2

- What limits are there to the speed of electronic circuits?
- What is the maximum distance a memory unit could be from an arithmetic unit and still make a 100-picosecond memory access time conceivable?

Exercise 3

Assume that we have a scalar c and two vectors $\underline{a}$ and $\underline{b}$ of length n. We consider three types of linear algebra operations:

1. Add the constant c to all the elements in $\underline{a}$;
2. Add the vectors $\underline{a}$ and $\underline{b}$ and store the result in $\underline{a}$;
3. Muliply all the elements in the vector $\underline{b}$ with the constant c and add $\underline{a}$ to the resulting vector. Store the final result in $\underline{a}$.

Below we show three FORTRAN subroutines which implement these operations (the particular choice of high level programming language does not matter):

```
subroutine op1 (a,c,n)
real a(n),c
do i=1,n
   a(i)=a(i) + c
enddo
return
end

subroutine op2 (a,b,n)
real a(n),b(n)
do i=1,n
   a(i)=a(i) + b(i)
enddo
return
end

subroutine op3 (a,b,c,n)
real a(n),b(n),c
do i=1,n
   a(i)=a(i) + c*b(i)
enddo
return
end
```

The above three routines were tested on an older supercomputer, a Cray T3E used at NTNU around year 2000, which comprised a number of DEC alpha chips (i.e., each processor represents a RISC architecture). In order to check the single-processor performance, we ran a test program which called each of these three routines many times. We then computed the average number of floating-point operations for different vector lengths n. Figure 1 summarizes the performance results obtained using a high level of compiler optimization.

(a) Explain these results. In particular, why does the speed of computation increase with n for smaller vector lengths, followed by a performance which is fairly independent of n? Why is there a sudden drop in performance for a certain vector length? This drop was observed to happen for $n > 4096$ for operation 2 and 3, while the drop was observed for $n > 8192$ for operation 1. Why does this drop happen earlier for operation 2 and 3 compared to for operation 1? Why is the speed for operation 3 higher than for operation 1? Why is the speed for operation 2 lower than for operation 1?

(b) Why is it important to be aware of the memory hierarchy on modern computers when implementing numerical algorithms for scientific and technical computing?

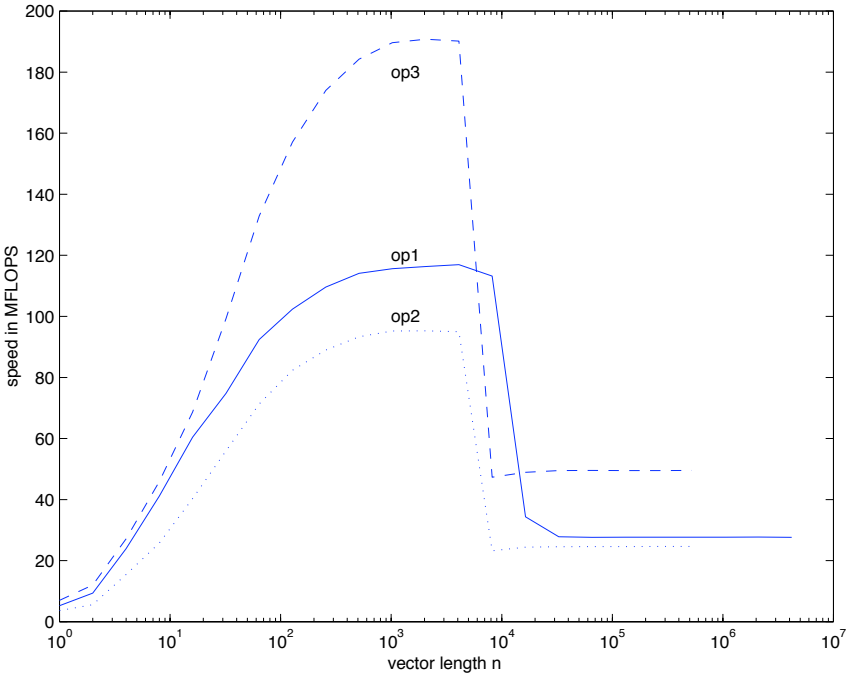

Figure 1: Performance on a single alpha chip on the supercomputer `huge`. This Cray T3E supercomputer was an earlier supercomputer at NTNU used around year 2000.

Exercise 4

Consider the vector operation

$$\underline{c} = \underline{a} + \underline{b},$$

where $\underline{a}$, $\underline{b}$, and $\underline{c}$ are vectors of length n. One way to implement this operation is:

```
for i=1,n
     c(i) = a(i) + b(i)
end
```

In order to optimize the expected performance, we want to make sure that the vector elements are stored next to each other in the main memory in the following sequence: $\ldots, a(i), b(i), c(i), a(i+1), b(i+1), c(i+1), \ldots$.

- Why could this way of storing the data be advantageous?
- How would you realize this in C and/or in FORTRAN?
- Do you think it matters much whether we store the vector elements in the above sequence, in particular, compared to just allocating the vectors $\underline{a}$, $\underline{b}$, $\underline{c}$ separately, i.e., each vector is stored contiguously in main memory?

Coding task

Implement a program in C/Fortran which performs the following operations:

$$\begin{aligned}\underline{x} &= \underline{a} + \gamma\underline{b} \\ \underline{y} &= \underline{a} + \underline{Ab}. \\ \alpha &= \underline{x}^T\underline{y}\end{aligned}$$

You can use any compatible vectors and matrices you see fit (e.g. random), and γ should be taken from the command line.