Introduction to Supercomputing

TMA4280 · Problem set 2

Exercise 1. The previous supercomputer at NTNU, *Njord*, was based on the POWER5 dual-core chip, which had cache sizes:

- L1: 32 kB,
- L2: 1.875 MB,
- L3: 36 MB.

Assuming double precision, how many floating point numbers can fit in each cache? What is the dimension of the largest square matrix that can fit in each cache?

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 (in a processor), and still allow a memory access time of 100 ps? (1 ps = 10^{-12} s.)

Exercise 3. Assume that we have a scalar c and two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of length n. We consider three types of linear algebra operations:

- Add c to all elements in $\boldsymbol{a}$.
- Add $\boldsymbol{a}$ and $\boldsymbol{b}$ and store the result in $\boldsymbol{a}$.
- Multiply $\boldsymbol{b}$ with c and add $\boldsymbol{a}$ to the resulting vector. Store the result in $\boldsymbol{a}$.

Below we show three Fortran subroutines implementing these operations, however, the particular choice of programming language doesn't matter.

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

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

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

These three routines were tested on an older supercomputer at NTNU around the year 2000. In order to check the single-processor performance, a test program was run which called each of these routines many times. The mean number of floating-point operations for different vector lengths n was then computed. The results are summarized in the following figure.

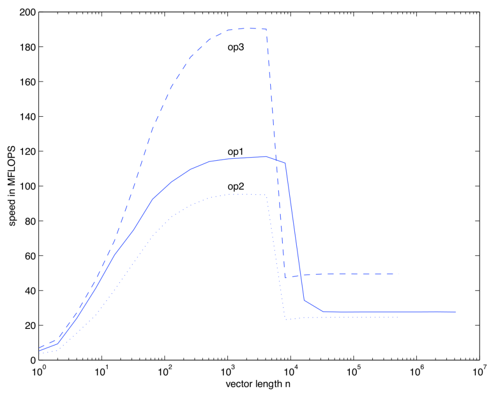

Figure 0.1: Performance on a single alpha chip on a Cray T3E supercomputer.

Explain these results. In particular,

- Why does the speed increase with n for small n, followed by being largely independent of n?
- Why is there a sudden drop at a particular vector length?
- Why does this drop happen sooner for operations 2 and 3 than for operation 1? (At $n = 4096$ and $n = 8192$ respectively. This is difficult to see in the figure.)

- Why does operation 1 run faster than operation 3, and why does operation 2 run slower still?

Exercise 4. Consider the vector operation $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$, where all vectors are of length n. One way to implement this operation is:

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

To maximize perfomance, we want to ensure that the vector elements are stored interleaved in memory:

$$\ldots, a_i, b_i, c_i, a_{i+1}, b_{i+1}, c_{i+1}, \ldots$$

- Why could this storage scheme be advantageous?
- How would you realize this in C and/or in Fortran?
- Do you think this scheme would pay off compared to the extra implementation effort?

Exercise 5. Implement a program in C or Fortran which performs the following three different operations:

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{a} + \gamma \boldsymbol{b} \\ \boldsymbol{y} &= \boldsymbol{a} + \boldsymbol{A}\boldsymbol{b} \\ \alpha &= \boldsymbol{x}^\mathsf{T} \boldsymbol{y} \end{aligned}$$

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