Norwegian University of Science and Technology
Department of Mathematical Sciences

Contact during exam:
Einar M. Rønquist (73593547 or 93404778)

FINAL EXAM FOR COURSE TMA4280
INTRODUCTION TO SUPERCOMPUTING

Friday, May 15, 2009
Time: 09:00–13:00

Permitted aids: Approved calculator.
All printed and hand written aids.

In general:

- Final grades will be announced by June 5.
- Assumptions in all the problems:

In the following you can assume that τ_A is the time to perform a single floating point operation. You can also assume that the time it takes to send a message of k bytes over the network is

$$\tau_C(k) = \tau_S + \gamma \cdot k,$$

where τ_S is the startup time, γ is the inverse bandwidth, and $\tau_S \gg \tau_A$.

Problem 1

Let $\underline{a}$, $\underline{b}$ and $\underline{c}$ be (real) vectors of length N, and let s be a scalar.
Consider the following operations:

$$\underline{a} := \underline{a} + s \cdot \underline{b} \quad (1)$$
$$\underline{a} = \underline{b} + \underline{c} \quad (2)$$
$$s = \underline{a}^T \underline{b} \quad (3)$$

where the symbol := means that each vector element on right hand side is first computed and then stored as the corresponding element in $\underline{a}$.

a) What is the total number of floating point operations for each of the operations (1), (2) and (3)?

Fasit: Operation 1: $2N$ floating-point operations.
Operation 2: N floating-point operations.
Operation 3: $2N - 1 \approx 2N$ floating-point operations.

b) Assume that we perform the operations on a single processor with superscalar capability, and that we use the highest level of optimization during compilation. Rank the operations (1)-(3) according to the expected performance, i.e., according to the expected number of floating point operations per second. Briefly justify your ranking.

Fasit: Ranking: (3), (1) and (2) with (3) being the fastest. Operation (3) has essentially two memory references per two floating point operations (the scalar s can be stored in a register); this operation can also exploit pipelining (multiple stages in the adder and multiplier), as well as chaining (superscalar processor). Operation (1) can also exploit pipelining and chaining, but has three memory references per two floating point operations (need to write $a(i)$ to memory in addition to fetching $a(i)$ and $b(i)$). Operation (1) can exploit pipelining, but not chaining; this operation has three memory references per single floating point operation.

Problem 2

We consider here the Buffon needle problem where a Monte Carlo approach is used. We perform M independent "experiments" which can be used to approximate the value π. In each experiment (experiment number i, $i = 1, \ldots, M$), we compute an approximation $\pi_{r,i}$.
The final approximation of π is then given as the average

$$\bar{\pi}_r = \frac{1}{M} \sum_{i=1}^{M} \pi_{r,i}, \tag{4}$$

while the standard deviation σ of all the experiments are given as

$$\sigma = \sqrt{\frac{1}{M} \sum_{i=1}^{M} (\pi_{r,i} - \bar{\pi}_r)^2}. \tag{5}$$

The time to perform a single experiment is completely dominated by the time to generate N random numbers. Assume that $N \gg M \gg 1$. The time to generate a random number is $10\,\tau_A$.

a) Estimate the time to perform the M experiments (including the computation of $\bar{\pi}_r$ and σ) on a single processor.

Fasit: The time to perform the M experiments on a single processor is dominated by the time to generate $M\,N$ random numbers plus the time to compute the average (4) and the standard deviation (5):

$$T_1 = 10\,M\,N\,\tau_A + M\,\tau_A + 3M\,\tau_A\,.$$

Since $N \gg M$, a good estimate is $T_1 \approx 10\,M\,N\,\tau_A$.

In the following, assume that the M independent "experiments" are distributed over $P = M$ processors (i.e., a single experiment on each processor).

b) Estimate the time to perform the M experiments (including the computation of $\bar{\pi}_r$ and σ) on M processors?

Fasit: In the case of using M processors, we only need to generate N random numbers on each processor. In addition, we need to perform a global sum in order to compute the average (4). Finally, we also need to compute the the quantity $(\pi_{r,i} - \bar{\pi}_r)^2$ for all i (a single subtraction and a single multiplication on each processor) and then perform a global sum to compute $\sum_{i=1}^{M}(\pi_{r,i} - \bar{\pi}_r)^2$ in (5). The final division by M and taking the square root of everything is done concurrently on each processor and only adds $\mathcal{O}(1)$ floating point operations. The time to perform the M experiments on M processors can therefore be estimated as

$$T_M \approx 10\,N\,\tau_A + 2\tau_S \log_2 M.$$

c) What is the speedup using M processors?

Fasit: The speedup is given as

$$S_M = \frac{T_1}{T_M} \approx \; M\left(\frac{1}{1 + \frac{\tau_S \log_2 M}{5\,N\tau_A}}\right).$$

d) Which functions from the `MPI` library are appropriate to use here?

Fasit: In addition to the usual initial and final `MPI` functions, we need to use `MPI_Allreduce` with the argument `MPI_SUM`.

e) Assume that we have access to a parallel implementation of operation (3) in Problem 1. Can we reuse this tool for the parallel implementation of our problem? If so, how?

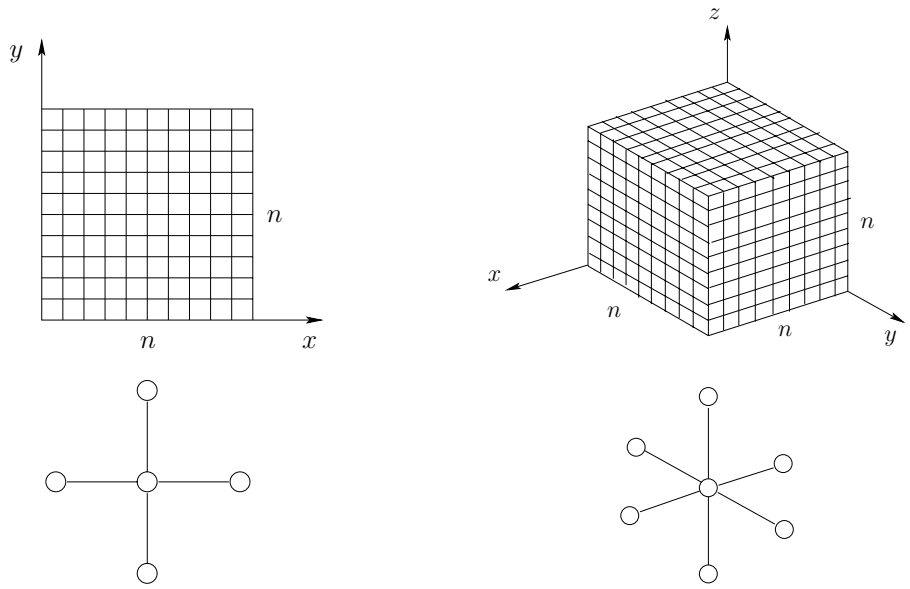

Figure 1: Computational domain and discretization using the finite difference method.

Fasit: Yes, we can! (Obama did not have anything to do with this...). If we define the vector $\underline{d}$ with elements $d_i = \pi_{r,i} - \bar{\pi}_r$, we see that (5) can be expressed as

$$\sigma = \sqrt{\frac{1}{M}\sum_{i=1}^{M} d_i^2} = \sqrt{\frac{1}{M}\,\underline{d}^T\underline{d}}.$$

Hence, we can use (3) with $\underline{a} = \underline{b} = \underline{d}$ to compute $s = \underline{d}^T\underline{d}$. The standard deviation is then given as $\sigma = \sqrt{\frac{s}{M}}$.

Problem 3

A research group wants to solve the Poisson problem on the unit square $\Omega = (0,1)^2$ in two dimensions, and on the unit cube $\Omega = (0,1)^3$ in three dimensions. The boundary condition is $u = 0$ specified on the boundary $\partial\Omega$. The researchers discretize the problem using the finite difference method, i.e., they discretize the Laplace-operator using the 5-point stencil in two dimensions, and using a corresponding 7-point stencil in three dimensions; see Figure 1. The grid spacing is $h = 1/n$ in each spatial direction, where $n \gg 1$. The number of unknowns is $N = (n-1)^d \approx n^d$ where $d = 2, 3$ is the number of spatial dimensions.

The researchers use the conjugate gradient method to solve the system of algebraic equations. The implementation is done in double precision. The method converges to a specified tolerance

in M iterations. An estimate for the total solution time on a single processor is

$$T_1 \approx 19\, n^2 \cdot M \cdot \tau_A \qquad \text{in 2D (e.g., see the course material),}$$
$$T_1 \approx 23\, n^3 \cdot M \cdot \tau_A \qquad \text{in 3D.}$$

The researchers now run their program using multiple processors. They decompose the original grid into P approximately equal subgrids, each of size $\frac{n}{P} \times n$ in two dimensions, and of size $\frac{n}{P} \times n \times n$ in three dimensions (i.e., they "slice" the grid only in the x-direction).

a) Give an estimate for the communication cost per conjugate gradient iteration using P processors. Consider both the 2D and the 3D case.

Fasit: Following the discussion from the lecture notes and exercises, the communication cost per conjugate gradient iteration is $T^1_{comm} \approx 4\tau_C(8n) + 2\tau_S \log_2 P$ in 2D and $T^1_{comm} \approx 4\tau_C(8n^2) + 2\tau_S \log_2 P$ in 3D. We have here assumed that $P > 2$ and that a red black ordering of the processors is used in order to exchange interface data. In addition, we have accounted for the cost of 2 innerproducts per iteration.

b) Give an estimate for the speedup. Consider both the 2D and the 3D case.

Fasit: The speedup is given as $S_P = P\left(\frac{1}{1+P\frac{M\,T^1_{comm}}{T_1}}\right)$ where T^1_{comm} and T_1 are given earlier.

c) Does the speedup depend on M, the number of iterations?

Fasit: No. The total communication cost is $T_{comm} = M\,T^1_{comm}$, i.e., proportional with M. The total computational time T_1 is also proportional with M. Hence, the factor M cancels out in the expression for the speedup. This is also as expected.

d) For a fixed number of processors, what happens to the speedup when n increases? Consider both the 2D and the 3D case.

Fasit: The speedup will increase. The reason for this is that T^1_{comm} scales as $\mathcal{O}(n)$ in 2D for large values of n, and scales as $\mathcal{O}(n^2)$ in 3D for large values of n. On the other hand, T_1 scales as $\mathcal{O}(n^2)$ in 2D for large values of n, and scales as $\mathcal{O}(n^3)$ in 3D for large values of n. Hence, in both 2D and 3D, T_{comm}/T_1 scales as $\mathcal{O}(1/n)$ as n increases, and thus the overhead with communication is reduced.

e) It can be shown that the number of iterations, M, scales as $\mathcal{O}(n)$.
Would you characterize the conjugate gradient method a scalable solution method?

Fasit: Inserting the estimate for M as a function of n, in the expression for the computational effort, T_1, we observe that T_1 will scale as $\mathcal{O}(n^3)$ in 2D and $\mathcal{O}(n^4)$ in 3D. Hence, the computational effort per degree-of-freedom (or per unknown) will be $\mathcal{O}(n)$. The conjugate gradient method is therefore not scalable in terms of floating point operations. However, the method is scalable in terms of memory requirement in the sense of only requiring $\mathcal{O}(n^2)$ bytes of memory in 2D and $\mathcal{O}(n^3)$ bytes of memory in 3D, i.e., a constant number of bytes of memory per unknown.

Problem 4

For each point in this problem, please choose the correct alternative.

a) The spectral bisection algorithm is useful in order to partition a computational grid into subgrids with approximately the same number of grid points in all the subgrids.
Answer: yes or no.

Fasit: Yes.

b) A double precision floating point number is often represented by 64 bits, but can sometimes be represented by only 32 bits.
Answer: true or false.

Fasit: False.

c) In the multi-processor case, a global sum can be performed using `MPI_Allreduce` and choosing the `MPI_SUM` option. The answer will be identical on all the processors, i.e., all the bits in the global sum are the same on all the processors.
Answer: yes or no.

Fasit: Yes.

Good luck!

Einar M. Rønquist