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

Tuesday December 19, 2006
Time: 15:00–19:00

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

In general:

- Short answers are fine as long as they are justified and explained.
- Final grades will be announced by January 19.

Problem 1

Let $\underline{x}$ and $\underline{y}$ be two vectors of length n. Consider the following operations (innerproducts):

$$\sigma_1 = \underline{x}^T \underline{x} = \sum_{i=1}^{n} x_i \cdot x_i \tag{1}$$

$$\sigma_2 = \underline{x}^T \underline{y} = \sum_{i=1}^{n} x_i \cdot y_i \tag{2}$$

Both operations are to be computed numerically in double precision on P processsors. Note that operation (1) is the same as operation (2) for the special case with $\underline{y} = \underline{x}$.

Assume in the following that each processor is a MIPS processor of the same type as on gridur. The clock frequency is 500 MHz and at most a single floating point number can be transferred

to or from the memory per clock cycle. Assume further that we use a distributed memory multiprocessor machine with the following network characteristics: the time it takes to send a message with k bytes, $\tau_C(k)$, can be approximated as

$$\tau_C(k) = \tau_S + \gamma \cdot k \qquad .$$

Here, τ_S is a fixed startup time, and γ is the inverse bandwidth.

a) We consider first the single process case ($P = 1$). What is the expected maximum performance of operation (1) and of operation (2)? How great is this performance compared to the maximum theoretical performance for this type of processor?

Fasit: For the innerproduct operation (2), the performance is memory bound. While we accumulate the contributions in the variable σ, this variable can be stored in a register. Hence, in order to update σ with the contribution $x_i \cdot y_i$, we need two memory references (two loads). This will take two clock cycles. During these two clock cycles, we can easily complete the necessary multiplication and addition (using pipelining and chaining). Hence, we are able to complete two floating point operations per two clock cycles. The highest performance we will observe is thus one half of the maximum theoretical performance, or 500 Mflops. This performance can be (approximately) seen for problems where the vectors $\underline{x}$ and $\underline{y}$ are fully available in L1 cache.

For the innerproduct operation (2), only a single memory reference is needed for each contribution $x_i \cdot x_i$. Hence, under optimal conditions, we should be able to complete two floating point operations per clock cycle. This performance corresponds to the maximum theoretical performance, or 1 Gflops. This performance can be (approximately) seen for problems where the vector $\underline{x}$ is available in L1 cache.

b) We now consider the multiprocessor case ($P > 1$). Assume that the vectors $\underline{x}$ and $\underline{y}$ are already distributed in an optimal way across the processors. Which operation will achieve the highest speedup, operation (1) or operation (2)? Explain your answer.

Fasit: Both innerproducts require communication. Using a binary tree for the global sum, the communication cost is $T_{comm} = (\tau_A + \tau_C(8)) \log_2(P) \approx \tau_S \log_2(P)$. The computation time on a single processor is $T_1 \approx 2n\tau_A$. The speedup is thus

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{\frac{T_1}{P} + T_{comm}} = \frac{P}{1 + P \cdot \frac{T_{comm}}{T_1}} = \frac{P}{1 + \left(\frac{P \log_2(P)}{2n} \frac{\tau_S}{\tau_A}\right)}$$

For a fixed P and n, the speedup only depends on τ_S and τ_A. The parameter τ_S is related to the network characteristics and can be assumed fixed. The parameter τ_A is the time per floating point operation. This is smallest for operation (1). Hence, operation (2) will achieve the highest speedup. This is as expected: it is harder to achieve good speedup for an optimized code.

c) Are operations (1) and (2) of interest in the conjugate gradient method?
If so, identify the operations in this method.

Fasit: Yes; both operations are of interest. Operation (1) is needed to compute the length of the residual vector, i.e., $\underline{r}^T\underline{r}$. Operation (2) is needed to compute the innerproduct between the search direction $\underline{p}$ and the vector $\underline{w}$ where $\underline{w} = \underline{A}\,\underline{p}$ (the matrix-vector product).

Problem 2

In order to check the MPI message passing on Gridur (the supercomputer at NTNU), we measure the time to complete a broadcast operation (MPI_Bcast). The performance test is as follows: we measure the time Δt to complete a broadcast operation on P processors, and where the message consists of a single floating point number. The measured times, divided by $\log_2 P$, are given in Table 1.

Table 1

P	$\Delta t / \log_2 P$ (in seconds)
2	$3.4 \cdot 10^{-6}$
4	$3.5 \cdot 10^{-6}$
8	$3.2 \cdot 10^{-6}$
16	$3.5 \cdot 10^{-6}$
32	$3.8 \cdot 10^{-6}$

a) We conclude that $\Delta t / \log_2 P \approx 3.5 \cdot 10^{-6}$ seconds for the broadcast operation. Does it seem reasonable that $\Delta t / \log_2 P$ is approximately equal to a constant?

Fasit: The broadcast operation is typically implemented using a recursive doubling algorithm (similar to the global sum algorithm). Hence, the communication pattern is a binary tree. At each stage in this communication process, each of the active processors will pass on the information to a new processor. The time this takes is approximately equal to τ_S (since $\tau_C(8) \approx \tau_C(1) \approx \tau_S$). Again, we have $\log_2 P$ stages. The total time to perform a broadcast operation can thus be expressed as

$$\Delta t \approx \log_2 P \cdot \tau_S \ .$$

Hence, we should expect that the time to perform a broadcast operation is proportional to $\log_2 P$, or that the ratio $\Delta t / \log_2 P$ stays constant as we change P.

b) What is your interpretation of the value $\Delta t / \log_2 P$? Explain your answer.

Fasit: Continuing the explanation from the previous point, we can identify the constant be the time to send a message with a single floating point number. This is approximately equal to the time to send a byte, which again is approximately equal to τ_S (or the startup cost to send a message). (We notice that the results in Table 1 suggest roughly the same value for τ_S as we obtained in an earlier exercise in the course.)

c) One way to compute the global sum such that all the processors end up with the same answer can be realized in MPI by using the library function MPI_Allreduce. Another way to obtain the same result is by first using the library function MPI_Reduce, followed by the function MPI_Bcast. Which alternative do you think is the fastest?

Fasit: The MPI_Allreduce should be the fastest. Using this alternative, only $\log_2 P$ stages are needed to find the global sum on all the processors. Using the second alternative will probably take approximately twice as long since $\log_2 P$ stages are needed both for MPI_Reduce and for MPI_Bcast.

d) The MPI communication library consists of many specific message passing operations. How would you classify all these operations into a few main groups or types of communication patterns?

Fasit: Point-to-point, one-to-all, all-to-one, and all-to-all. The last 3 groups of operations can again be classified as collective operations.

Problem 3

We consider now the Poisson problem in a rectangular domain $\Omega = (0, L_x) \times (0, L_y)$. The solution is specified to be zero along the domain boundary. The problem is discretized using a 5-point stencil on a structured finite difference grid with $(n_x + 1)$ points in the x-direction and $(n_y + 1)$ points in the y-direction. Hence, the number of unknowns is $(n_x - 1) \times (n_y - 1)$. The discrete equations are solved iteratively using the conjugate gradient method.

a) Give an expression for the solution time per conjugate gradient iteration on a single processor.

Fasit: Following the discussion from the exercises and from the lecture notes, the solution time per conjugate gradient iteration on a single processor is given as $T_1 \approx 19N\tau_A$ where $N = (n_x - 1)(n_y - 1) \approx n_x n_y$, and τ_A is the time to perform a floating point operation.

We decide to parallelize our program. In particular, we partition our problem by decomposing the $n_x \times n_y$ grid into exactly 1000 equal subgrids of size $m_x \times n_y$ with $m_x = n_x/1000$. We assume that n_x is divisible by 1000. Our intention is to run our parallel program on P processors by assigning one or more of these subgrids to each processor (i.e., $P \leq 1000$).

b) Give an expression for the communication cost per conjugate gradient iteration.

Fasit: Again, following the discussion from the lecture notes and exercises, the communication cost per conjugate gradient iteration is $T_{comm} \approx 4\tau_C(8n_y) + 2\tau_S \log_2 P$. 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.

c) Give an expression for the speedup on $P = 1000$ processors.

Fasit: The solution time per conjugate gradient iteration on $P = 1000$ processors is $T_P = T_1/P + T_{comm}$. The number of floating point operations is reduced with a factor of P since we have exactly one subgrid per processor. Hence, the speedup on $P = 1000$ processors is

$$S_{1000} = \frac{T_1}{T_P} = \frac{1000}{1 + 1000\frac{T_{comm}}{T_1}}$$

where T_{comm} and T_1 are defined earlier.

d) One of the processors fails and we decide to run our program on $P = 999$ processors. In this case, we assign two subgrids to one of the processors. How does the speedup in this case compare with the previous result, in particular, for large problem sizes where we can ignore the communication cost?

Fasit: When $P = 999$, the number of floating point operation on one of the processors will correspond to two subgrids, i.e., $T_P = 2T_1/1000 + T_{comm}$. Hence, the speedup on $P = 999$ processors will be

$$S_{999} = \frac{T_1}{T_P} = \frac{1000}{2 + 1000\frac{T_{comm}}{T_1}}$$

If the communication cost can be ignored, we see that the speedup S_{999} is half the speedup S_{1000}.

e) A former student of the course TMA4280 suggests that we run our program on $P = 500$ processors until the failed processor has been fixed. We decide to try this and assign two subgrids to all the processors. Will the solution time on $P = 500$ processors be larger or smaller than the solution time on $P = 999$ processors?

Fasit: The solution time on $P = 500$ processors will generally be smaller than the solution solution time on $P = 999$ processors: The solution time due to floating point operations will be exactly the same since we have two subgrids on one or more processors. However, the communication time will be higher when $P = 999$ because of the $\log_2 P$-dependence (although the difference is not very big).

Good luck!

Einar M. Rønquist