NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Mathematical Sciences

Examination paper for
TMA4280 Introduction to Supercomputing

Academic contact during examination: Arne Morten Kvarving

Phone: 97544792

Examination date: May 21, 2014

Examination time (from–to): 09:00-13:00

Permitted examination support material: Code: C

All lecture notes, slides, codes, exercises and suggested solutions from course material.
Rottmann: Mathematical formulas.
Earlier exams+suggested solutions in TMA4280.
LINPACK specification and FAQ.
All handwritten notes, including annotations on notes/slides/codes.
Wikipedia printouts.
Introduction to parallel programming from llnl.gov
Simple, approved calculator.

Other information:

- There are 15 questions and 4 points to be earned on each one.
- Make sure to always state any assumptions you make.

Language: English

Number of pages: 6

Number pages enclosed: 0

Checked by:

Date Signature

Problem 1 We have considered two ways of parallelizing a program in the course.

a) Outline the two approaches and explain how they relate to different hardware architectures.

Fasit: We here expect the student to outline the shared memory and distributed memory approaches. As for the hardware, no details expected but the student should clearly show that she has realized the connections between hardware architecture and choice of programming model / API.

b) A scientist has a huge dataset $\{\mathbf{v}_i\}_{i=0}^{N}$ which needs to go through a filter. The filter is linear and can thus be formed as a matrix. The filter couples globally, so the resulting matrix is dense. The size of the data $\mathbf{v}_i$ is (in general) different for each i. The final program can be stated as

```
void filter_data(Matrix* out, const Matrix* in, int N)
{
  int i, j, k, l;
  Matrix filter;
  for (i=0;i<N;++i) {
    // form filter
    formFilter(filter , in[i]->rows, in[i]->cols);
    // apply filter
    for (l=0; l < filter->rows; l++) {
      for (j=0; j  < in[i]->cols; j++) {
        out[i]->data[l][j] =  0.0;
        for (k=0;  k < in[i]->rows; k++) {
          out[i]->data[l][j] += filter->data[l][k]*in[i]->data[k][j];
        }
      }
    }
  }
}
```

She finds that the code runs very slow, too slow to be useful, and therefore she comes to you, the parallel computing expert, for advice. Outline which suggestions you would give her. Remember to give the reasoning behind your suggestions!

Fasit:

- Matrix-matrix product ⇒ BLAS
- OpenMP on inner loops
- OpenMP on outer loop (requires more memory), MPI on outer loop.
- Distributed matrices (MPI)

c) What is a "race condition"? When do you expect to run into it? How can you resolve it?

Fasit: See notes on OpenMP.

d) What is referred to by the term "deadlock"? When do you run into it? Have can you resolve it? What would you do to minimize the chances of it occuring?

Fasit: See notes on MPI.

Problem 2 We here consider the solution of a 2D Poisson problem with homogenous Dirichlet boundary conditions on a unit square;

$$\begin{aligned} -\nabla^2 u &= f \text{ in } \Omega = (0,1)\times(0,1), \\ u\,|_{\partial\Omega} &= 0. \end{aligned}$$

The problem is discretized using a second order centered finite difference method, i.e., using the five point formula for the second derivatives on a structured mesh with spacing $h = 1/n$ in both spatial directions. This results in a linear system of equations

$$\mathbf{A}u = f. \tag{1}$$

We solve the problem in parallel using conjugate gradient iterations and a block domain decomposition. The computer is a distributed memory machine interconnected with a network, which we model using the standard linear model,

$$T(b) = \tau_c + \gamma b.$$

Here $T(b)$ is the time to send b bytes over the network, τ_c is a latency and γ is the inverse network bandwith.

a) Give an estimate the flop count, the memory usage and the parallel efficiency.

Fasit: Here the student need to point out whether matrix-less iterations are used or not. Assuming so in the following.

- Flop count $\sim 19n^2$ (1 MxV, 2 dotproduct, 3 axpy operations). Alternatively $\sim 15n^2$ if the multiplications by 1 is skipped.
- Memory usage $\sim 4n^2$ (4 vectors).
- Efficiency:

$$\eta_P = \frac{S_P}{P} = \frac{T_1}{pT_p} = \frac{T_1}{P\left(\frac{T_1}{P} + T_{\text{comm}}\right)} = \frac{1}{1 + \frac{PT_{\text{comm}}}{T_1}}.$$

Since $T_1 = 19n^2\tau_a$ and $T_{\text{comm}} = 2\tau_c \log_2 P + 4\left(\tau_c + \gamma\frac{8n}{\sqrt{P}}\right)$ (data exchange across boundaries, 2 dot products) this yields

$$\eta_P = \frac{1}{1 + \frac{P\left(2\tau_c \log_2 P + 4\left(\tau_c + \gamma\frac{8n}{\sqrt{P}}\right)\right)}{19n^2\tau_a}}.$$

b) It turns out this method is too inefficient due to the iteration count. We thus switch to preconditioned conjugate gradient iterations, where we use an additive Schwarz preconditioner with a single domain per process and with subdomain solvers based on diagonalization (**NOT** FST based). Give an estimate of the efficiency in this case.

Note : You can **ignore** setup costs, such as those associated with the diagonalization method.

Fasit: First, we assume that the same preconditioner is used in the serial code, i.e., the same amount of subdomains within the single process. The flop count for the preconditioner is approximately four matrix-matrix products per subdomain, i.e.

$$P\frac{8n^3}{P^{\frac{3}{2}}},$$

where we have used that a matrix-matrix product use approximately $2n^3$ flops for a $n \times n$ matrix. The extra cost of data exchange in the preconditioner is approximately 2 sends across the boundaries (one before, one after). Thus;

$$T_{\text{comm},2} = T_{\text{comm}} + 8\left(\tau_c + \gamma\frac{8n}{\sqrt{P}}\right).$$

This can be immediately plugged into the expression from earlier, yielding

$$\eta_P = \frac{1}{1 + \frac{P\left(2\tau_c \log_2 P + 12\left(\tau_c + \gamma \frac{8n}{\sqrt{P}}\right)\right)}{19n^2\tau_a + \frac{2n^3}{P^{\frac{3}{2}}}}}.$$

c) Outline how you would implement such a code using the tools offered by MPI.

Fasit: Communicator topologies, MPI_Shift.

Problem 3 For each point in this problem, please choose the correct alternative and give your reasoning. You get 0 points for a wrong answer, 1 point for a correct alternative with the wrong/lacking explanation and 4 points for the correct alternative with a correct explanation.

a) A ccNUMA machine can be programmed using either a distributed or a shared memory programming model.

Answer: true or false

Fasit: True
A ccNUMA is a cache-coherent non-uniform memory access shared memory machine. A shared memory machine can always be programmed as a distributed memory one (in fact, it helps address data locality on NUMA architectures).

b) You typically get performance closer to the theoretical peak performance of a machine when you do level 1 operations (vector operations) compared to level 3 (matrix-matrix operations).

Answer: true of false

Fasit: False Level 3 operations typically have better cache utilization (data reuse) and thus get closer to peak performance.

c) MPI-I/O always writes multi-dimensional arrays in Fortran order.

Answer: true or false

Fasit: False MPI-I/O just writes data. You can easily (in fact it's built-in through darrays) output in C order, or any other order you might want.

d) A code with a large parallel speedup has a large parallel efficiency.

Answer: true or false

Fasit: False

Large speedup does not necessarily mean high efficiency. You can throw a million processors at a problem to get a speedup of 1000. Speedup is high-ish, but the efficiency is low. If explictly stated that a fixed number of processors is assumed, true gives full score as well.

e) A modern processor typically has a cache hierarchy. These are designed as levels, where a higher level is given to faster memory.

Answer: true or false

Fasit: False

Lower levels is given to faster memory (L1 is faster than L3). If the lower/higher meaning is explictly stated and differs from this definition, this give full score as well.

f) A superscalar processor can perform two additions simultanously.

Answer: true or false

Fasit: False

That would be a vector (SIMD) capable processor. Superscalar is exploiting the nature of a pipeline to in effect obtain the result of two operations per clock tick. But no additions happens simultanously as such, each stage in the pipe only perform a single operation at any point in time.

g) Floating point numbers of a given precision can only represent a fixed range of numbers.

Answer: true or false

Fasit: True

Since the exponent has a fixed number of bits, we can only represent a fixed range of numbers.

h) OpenMP is usable from all programming languages.

Answer: true or false

Fasit: False

OpenMP is a C/Fortran language extension. While you can access it from other languages through bindings, it is (currently) either Fortran or C code that does the work.