Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	2012 – 06 – 02
Sal	TER3
Tid	14:00 – 18:00
Kurskod	TDDC78
Provkod	TEN1
Kursnamn/benämning	Programmering av parallelldatorer – metoder och verktyg
Institution	IDA
Antal uppgifter som ingår i tentamen	7
Antal sidor på tentamen (inkl. försättsbladet)	5
Jour/Kursansvarig	Christoph Kessler, Marco Kupiainen
Telefon under skrivtid	Se täckbladet (sida 1) av tentan
Besöker salen ca kl.	Se täckbladet (sida 1) av tentan
Kursadministratör (namn + tfnnr + mailadress)	Gunilla Mellheden, IDA, 2297, gunme @ ida.liu.se
Tillåtna hjälpmedel	Engelsk ordbok
Övrigt (exempel när resultat kan ses på webben, betygsgränser, visning, övriga salar tentan går i m.m.)	Se täckbladet (sida 1) av tentan

TENTAMEN / EXAM

TDDC78/TANA77

Programmering av parallelldatorer / Programming of parallel computers

2012-06-02, 14:00–18:00, TER3

Christoph Kessler and Henrik Brandén
Dept. of Computer Science (IDA), Dept. of Mathematics (MAI)
Linköping University

Hjälpmedel / Allowed aids: Engelsk ordbok / dictionary from English to your native language

Examinator: Christoph Kessler (TDDC78), Henrik Brandén (TANA77)

Jourhavande lärare:
Christoph Kessler (IDA) 013-28-2406; 0703-666687; visiting at ca. 16:00
Marco Kupiainen (MAI) 011-4958703. **In case of questions call before 16:30.**

Maximalt antal poäng / Max. #points: 40

Betyg / Grading (prel.): The preliminary threshold for passing (grade 3) is at 20p, for grade 4 at 28p, for grade 5 at 34p.
Because of new regulations by Linköping University, we can no longer give ECTS grades.

General instructions

- Please use a new sheet of paper for each assignment. Order your sheets by assignments, number them, and mark them on top with your exam ID number and the course code.
- You may answer in either English or Swedish. **English is preferred** because not all correcting assistants understand Swedish.
- Write clearly. Unreadable text will be ignored.
- Be precise in your statements. Unprecise formulations may lead to a reduction of points.
- Motivate clearly all statements and reasoning.
- Explain calculations and solution procedures.
- The assignments are *not* ordered according to difficulty.

Note: In the case of questions regarding the assignments 3 to 4, please contact Marco Kupiainen in the first hand; for questions regarding the other assignments, contact Christoph Kessler in the first hand.

1. (6.5 p.) **Performance tools and analysis**

   (a) (2 p.) Describe performance analysis through tracing. (1p)
   What are the advantages and disadvantages of the approach, in comparison to alternative techniques for performance data collection? (1p)

   (b) (1p) Which aspect of parallel computation is modeled well by the PRAM (Parallel Random Access Machine) model, and which important property / properties of real parallel machines does it abstract from?

   (c) (2 p.) (*i*) Derive Amdahl's law and (*ii*) give its interpretation.
   (Note: a picture is nice but is not a proof; a calculation is expected for the derivation of Amdahl's Law.)

   (d) (1.5p) Consider the following (contrived) sequential code:

```
#define N 1000000
float a[N];
...
for (iter=0; iter<1000; iter++)
 for (i=0; i<N; i++) {
   if (i%2) // even:
     a[i] = a[i] * 1.14;
   else     // odd:
     a[i] = a[i] * 0.89 + 0.5;
 }
...
```

   i. Run on a single core of a modern processor, this code takes about 10 seconds. After a simple loop restructuring of the `i` loop, we can perform the same calculation in less than half the time (still single core, single threaded execution). How and why? (1p)
   ii. Can the `i` loop be parallelized? Why or why not? (0.5p)

2. (4 p.) **Parallel program design methodology**

   Foster's design methodology consists of four stages. Name and explain them. Give details about their goals. What are the important properties of the result of each stage? Be thorough!

(In case of questions about the following two assignments please ask Marco Kupiainen in the first hand.)

3. (6 p.) The problem

$$\frac{du_j}{dt} + D_0 u_j(t) = 0 \quad j = 2, 3, 4, \ldots, N-1$$

with boundary data $u_1(t) = 1$, $u_N(t) = u_{N-1}(t)$, is an approximation of the partial differential equation $u_t + u_x = 0$, with boundary data $u(0,t) = 1$. The centered difference operator is given by

$$D_0 u_j = \frac{u_{j+1} - u_{j-1}}{2\Delta x}$$

Use message-passing for your solution.

   (a) Draw a picture and explain how the array `u[N]`, representing the solution u_j can be distributed onto p processors. (2 p.)

   (b) Write down the algorithm to evaluate $D_0 u_j$ at all points, $j = 2, 3, \ldots, N-1$, for the array distributed on the parallel computer. (2 p.)

   (c) What changes would be necessary in (a) and (b) if we replace the boundary conditions by the periodic $u_1 = u_N$? Think especially about the type of send-receive needed in (b). (2 p.)

4. (9 p.) Assume that we are given two grids covering a domain of computation, and that we have a computer with two processors. Grid 1 has $n_i^1 \times n_j^1 \times n_k^1$ points, and grid 2 has $n_i^2 \times n_j^2 \times n_k^2$ points. *(Hint: these 1, 2 are upper indexes, no exponents).*

We have the array $u[g]^n_{i,j,k} \approx u(t_0 + (n-1)\Delta t, x_0 + (i-1)\Delta x, y_0 + (j-1)\Delta y, z_0 + (k-1)\Delta x)$ defined on the grid $g = 1, 2$, which is the solution of a partial differential equation. Taking one time step involves only direct neighbors in all three directions,

$$u[g]^{n+1}_{i,j,k} = S(u[g]^n_{i+1,j,k}, u[g]^n_{i,j,k}, u[g]^n_{i-1,j,k}, u[g]^n_{i,j+1,k}, \cdots)$$

We consider two different schemes to distribute the domain, i.e. the two grids and thus the two arrays $u[g]$, $g = 1, 2$, across the two processors. Assume that the time to evaluate the difference scheme S in one grid point is given by Ct_a, where C is a constant and t_a the time to do one arithmetic operation on the computer. Write down a formula for the time it takes to do one time step for the following cases (a) and (b):

   (a) The two grids (arrays) are each split into two equal-sized blocks that are distributed on the machine, i.e., each processor holds $n_i^1/2 \times n_j^1 \times n_k^1$ points (elements) of Grid 1 (array $u[1]$) and $n_i^2/2 \times n_j^2 \times n_k^2$ points (elements) of Grid 2 (array $u[2]$) (3 p.)
   [Hint: here you should include the time it takes to communicate the overlap boundary between processors. We do not consider the possible communication between the two grids (arrays) and assume for simplicity that they are processed independently of each other.]

   (b) Grid 1 (entire array $u[1]$) is assigned to processor 1 unsplit, and Grid 2 (entire array $u[2]$) is assigned to processor 2 unsplit. (3 p.)

   (c) Which approach is best if grid 1 holds $100 \times 100 \times 100$ points and grid 2 holds $150 \times 100 \times 100$ points? Assume $C = 200$, $t_a = 10^{-9}$, $t_s = 10^{-6}$, $t_w = 10^{-8}$. (3 p.)

5. (5.5 p.) **MPI**

   (a) (2 p.) MPI supports *nonblocking* (also called *incomplete*) point-to-point communication. What does that mean?

   Give an example scenario demonstrating how using a nonblocking send (`MPI_Isend`) or receive (`MPI_Irecv`) routine instead of their blocking counterpart could speed up program execution. What kind of routine is necessary with nonblocking communication if we need to make sure that data dependences are preserved?

   (b) (1 p.) Give two good reasons for using collective communication operations instead of equivalent combinations of point-to-point communication (MPI_send and MPI_receive) operations.

   (c) (2.5 p.) *One-sided communication in MPI*

      i. Explain the principle of one-sided communication in MPI-2. (1p)
      Hint: You might want to illustrate your answer with a pseudocode example and/or an annotated picture.

      ii. Why is one-sided communication considered being "closer" to the shared-memory programming model than ordinary two-sided message passing? (0.5p)

      iii. How does one-sided communication still differ from the shared-memory programming model? (0.5p)

      iv. And which nice feature of shared memory programming is, even with one-sided message passing, still missing? (0.5p)

6. (4 p.) **OpenMP**

   (a) What is *guided self-scheduling*? How does it differ from the other scheduling schemes for parallel loops defined in OpenMP? What kind of parallel loops are suitable candidates for guided self-scheduling? Why? (2p)

   (b) What is the memory consistency model guaranteed by OpenMP implementations? (short answer) (0.5p)

   (c) OpenMP allows so-called *orphaning* of some directives. What does that mean, and why is it useful in programming? (1.5p)

7. (5 p.) **Parallel computer architecture**

   (a) What kind of parallelism can be exploited efficiently in computational grids? (0.5p)

   (b) Identify two important trends in supercomputer architectures that can be derived from the (recent) TOP-500 lists. (1p)

   (c) Given an application with a fixed performance requirement (in GFlops). Why can, provided that the application can be parallelized (work-)efficiently, the transition from a single-core to a multicore execution platform be advantageous from a power efficiency point of view? (1p)

   (d) Explain the *write-invalidate protocol* used to achieve sequential consistency in a cache-based shared-memory system with a bus as interconnection network. (1.5p)

   (e) Give an example of an interconnection network topology (name and sketch) where the node degree grows logarithmically in the number of nodes. (1p)