Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	2011 – 08 – 16
Sal	TER1
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	8
Antal sidor på tentamen (inkl. försättsbladet)	5
Jour/Kursansvarig	Christoph Kessler (jour: Lu Li), Henrik Branden
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

2012-01-11, 14:00–18:00, KÅRA

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:
Lu Li (assistant, IDA) 013-28-6618; 0704-759692; visiting at ca. 16:00
Christoph Kessler (IDA), on travel, 0703-666687
Henrik Brandén (MAI) 013-28-5759; 0706-011969; visiting at ca. 16:00

Maximalt antal poäng / Max. #points: 40

Betyg / Grading (prel.):

MatNat		C, D, Y, DI	ECTS-graded students[a]
< 20	U	< 20	U	FX
$20 - 30$	G	$20 - 27$	3	C
$31 - 40$	VG	$28 - 33$	4	B
		$34 - 40$	5	A

[a] Swedish grades will be automatically translated to the ECTS marks for exchange and international master program students as given above, according to a decision by the LiU rector in 2008.

General instructions

- Please use a new sheet for each assignment. Number all your sheets, and mark each sheet 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 4 to 5, please contact Henrik Brandén 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) There exist several possible causes for *speedup anomalies* in the performance behavior of parallel programs. Name and explain one of them.

   (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) (1.5 p.) How are the supercomputers on the TOP500 list ranked?
   And how does the TOP500 ranking criterion differ from the so-called *peak performance* of a parallel computer system?

2. (2 p.) **Parallel programming models**

   Explain the parallel program execution styles *fork-join* and *SPMD*, and explain the main difference. (1p)

   Which one is more comfortable for the programmer, and why? (0.5p)

   Which one is used in MPI? (0.5p)

3. (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 Henrik Brandén in the first hand.)

4. (6 p.) A matrix-vector multiplication $y = Ax$, where A is $n \times n$, is to be computed using message passing on a distributed memory machine with p processors, organized as a ring. The matrix A and the vector x have been partitioned into row blocks A_i and X_i of size $n/p \times n$ and $n/p \times 1$ respectively,

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{pmatrix}, \qquad x = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{pmatrix},$$

and distributed in such a way that processor i has A_i and X_i.

   (a) Propose a suitable algorithm for computing y. (2p)

   (b) Show that the execution time of this algorithm is roughly

$$T_p = \frac{2n^2}{p}\gamma + p\left(\alpha + \frac{n}{p}\beta\right)$$

   where α is the latency, β^{-1} is the data transfer speed, and γ is the time of one arithmetic operation. (2p)

   (c) Assuming $n = p$, what is the efficiency of this algorithm? (2p)

5. (9 p.) Consider the following column version of the LU factorization algorithm

```
do k=1,n-1
   a(k+1:n,k) = a(k+1:n,k)/a(k,k)
   do j=k+1
      a(k+1:n,j) = a(k+1:n,j) - a(k,j)*a(k+1:n,k)
   end do
end do
```

   (a) Describe a parallel version of the algorithm, including the partitioning and distribution of data, for a distributed memory machine using data parallel programming. (3p)

   (b) The following is an OpenMP parallelization of the algorithm for a shared memory machine.

```
do k=1,n-1
   a(k+1:n,k) = a(k+1:n,k)/a(k,k)
   !$omp parallel do schedule (static)
   do j=k+1,n
      a(k+1:n,j) = a(k+1:n,j)-a(k,j)*a(k+1:n,k)
   end do
   !$omp end parallel do
end do
```

   Explain the algorithm, including the partitioning of data. (3p)

   (c) What kind of modification of the column LU algorithm and what kind of partitioning and distribution of data are required in order to get a scalable algorithm when using message passing on a distributed memory machine? (3p)

6. (6 p.) **MPI**

   (a) (4 p.) The `MPI_Barrier` operation is a collective communication operation with the following signature:

   ```
   int MPI_Barrier ( MPI_Comm comm );
   ```

   where the current communicator is passed as `comm`.
   Once called, the function does not return control to the caller before all p MPI processes belonging to the process group of communicator `comm` have called `MPI_Barrier` (with this communicator).

   i. Write an implementation of `MPI_Barrier` using only `MPI_Send` and `MPI_Recv` operations. Use C, Fortran, or equivalent pseudocode (explain your language constructs if you are not sure about the right syntax). *Explain your code.* (2p)
   (Note: There exist several possibilities. An algorithm that is both time-optimal and work-optimal for large p gives a 1p bonus, provided that the analysis is correct and the optimality is properly motivated.)

   ii. Show how your code behaves over time by drawing a processor-time diagram for $p = 8$ showing when messages are sent from which source to which destination, and what they contain. (0.5p)

   iii. Analyze asymptotically the (worst-case) *parallel execution time*, the *parallel work* and the *parallel cost* of your implementation (for arbitrary values of p) as a function in p. You may use the delay model or the LogP model for this purpose (state which model you use). If you need to make further assumptions, state them clearly. (1.5p)
   Hint: Communication contributes to the work done by a parallel algorithm.

   (b) (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?

7. (2 p.) **OpenMP**

   (a) What kind of parallel loops are suitable candidates for static scheduling? Why? (1p)

   (b) Why is the design of OpenMP helpful for *incremental* parallelization of sequential codes? (1p)

8. (4.5 p.) **Parallel computer architecture**

   (a) (0.5p) What does *Moore's Law* really say? Be precise!

   (b) 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)

   (c) Many cache-based shared-memory systems use *bus-snooping*. What does that mean, what is its purpose, and how does it work? (2p)

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