Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	2010 – 05 – 27
Sal	KÅRA
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	9
Antal sidor på tentamen (inkl. försättsbladet)	5
Jour/Kursansvarig	Christoph Kessler, 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

2010-05-27, 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:
Christoph Kessler (IDA) 013-28-2406; 0703-666687; visiting at ca. 16:00
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.
- 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. (5 p.) **Performance tools and analysis**

   (a) (1p) Give two typical examples for kinds of performance-related data about message passing programs that can be collected using *software counters.*

   (b) (1p) Modern tool-suites for performance analysis of parallel programs consist of a collection of several kinds of tools. Give four different kinds of such tools. *(I.e., no concrete tool names, but a short term saying what each kind of tool does.)*

   (c) (1p) There exist several possible causes for *speedup anomalies* in the performance behavior of parallel programs. Name and explain one of them.

   (d) (2 p.) Derive Amdahl's law and give its interpretation.
   (NB — a picture is nice but not a proof; give a calculation.)

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 (at top level) in OpenMP (short explanation)? (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!

4. (6 p.) In FORTRAN 90, a SAXPY operation can be implemented as

```
y(1:n) = a*x(1:n) + y(1:n)
```

   In FORTRAN 77, the same operation looks like

```
do i=1,n
  y(i) = a*x(i) + y(i)
end do
```

   (a) What is the benefit of the FORTRAN 90 notation? (1p)

   Consider data parallel programming on a distributed memory machine with p processors organized as a ring. Assume a vector length $n = rp$ for some integer $r > 0$.

   (b) How is the given FORTRAN 90 code executed in terms of communication, synchronization, and arithmetics? (2p)

   (c) What is the efficiency E_p? (2p)

   (d) Is the operation scalable? Motivate your answer! (1p)

5. (9 p.) Consider matrix-vector multiplication $y = Ax$ on a

   (a) distributed memory machine using data parallel programming, (3p)

   (b) distributed memory machine using message passing, and on a (3p)

   (c) shared memory machine using shared memory programming. (3p)

   For each case, explain one algorithm of your own choice, including the partitioning and distribution of data.

6. (7 p.) **MPI**

   (a) (5.5 p.) The `MPI_Allreduce` operation is a collective communication operation with the following parameters:

```
int MPI_Allreduce ( void *sendbuf, void *recvbuf,
                    int count, MPI_Datatype elemtype,
                    MPI_Op op, MPI_Comm comm );
```

   where op is a pointer to a function combining two elements of type `elemtype` into a single element of the same type. Examples of predefined combine functions in MPI are `MPI_MAX` (maximum), `MPI_SUM` (sum) or `MPI_LAND` (logical AND).

   `MPI_Allreduce` works as follows: If executed by a group of p MPI processes, with each process i contributing a local array $s_i[0 : m-1]$ in `sendbuf` with `count`$= m$ elements of type `elemtype`, for $i = 0, ..., p-1$, each of the p processes will afterwards hold, in its array `recvbuf` with m elements of same type `elemtype`, an array $r_i[0 : m-1]$ that is the elementwise op-combination of the operand arrays $s_0, ..., s_{p-1}$, i.e.,

   $r_i[j] = s_0[j]$ op $s_1[j]$ op ... op $s_{p-1}[j]$ for $j = 0, ...,$ count $- 1$ and $i = 0, ..., p-1$.

   All processes of the group executing a call to `MPI_Allreduce` must pass in equal argument values for `count`, `elemtype` and `op`, respectively. A typical call could look as follows:

```
MPI_Allreduce ( S, R, m, MPI_FLOAT, MPI_SUM, MPI_COMM_WORLD );
```

   i. What is the difference in behavior to `MPI_Reduce`? (0.5p)

   ii. Write an implementation of `MPI_Allreduce` 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. (2.5p)
   (Note: There exist several possibilities. An algorithm that is time-optimal for large p gives a 1p bonus, provided that the analysis is correct and the optimality is properly motivated.)
   Show how your code behaves over time by drawing a processor-time diagram for $p = 4$ showing when messages are sent from which source to which destination, and what they contain. (0.5p)
   Let m denote the value of `count`, and assume that the `elemtype` is `MPI_FLOAT` (normal floatingpoint numbers) and that any call to function op executes in time t_{op}, where $t_{op} \in O(1)$. Analyze asymptotically the (worst-case) *parallel execution time* and the *parallel work* of your implementation (for arbitrary values of p and m) as a function in p and m. You may use the delay model, the BSP model or the LogP / LogGP model for this purpose (state which model you use). If you need to make further assumptions, state them clearly. (2p)
   Hint: Communication contributes to the work done by a parallel algorithm.

   (b) (1.5 p.) Explain the principle of one-sided communication in MPI-2.

7. (1.5 p.) **Grid Computing**

   (a) (1 p.) What sort of applications (with what kind of parallelism) can use computational grids effectively?

   (b) (0.5 p.) Name one task performed by *grid middleware*.

8. (3 p.) **OpenMP**

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

   (b) (2 p.) What is the purpose of the `flush` directive in OpenMP? Give a short example to illustrate how it is used. Name at least one technical cause that makes the explicit use of `flush` in the program necessary to guarantee a correct program execution.

9. (2.5 p.) **Parallel computer architecture**

   (a) 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)

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