Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	2010 – 08 – 17
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	7
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

2010-08-17, 14:00–18:00, TER1

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 3 to 4, please contact Henrik Brandén in the first hand; for questions regarding the other assignments, contact Christoph Kessler in the first hand.

1. (5.5 p.) **Performance tools and analysis**

   (a) (0.5p) Modern processors provide several built-in *hardware counters* that allow to collect certain kinds of performance data. Give one typical example for performance-related information (about sequential code) that can be obtained from such counters.

   (b) (0.5p) Give one typical example for performance-related data about shared-memory parallel programs that can be collected using *software counters*.

   (c) (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.)*

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

   (e) (2.5 p.) Derive Gustafsson's law and give its interpretation. Explain how it differs from Amdahl's law, and for what kind of parallel computations it is more appropriate.

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!

3. (6 p.) The sum

$$s = \sum_{k=1}^{p} a_k$$

   is to be computed using message passing on a distributed memory machine with p processors organized as a ring. A naive approach is to use the algorithm

```
do k=1,p-1
   transfer a(k) from node k to node k+1
   let node k+1 compute a(k+1) = a(k) + a(k+1)
end do
```

   (a) What is the speed-up S_p and efficiency E_p of this algorithm? (2p)

   (b) Describe the Cascade summation algorithm. (2p)

   (c) Show that the execution time of the cascade summation algorithm is

$$T_p = (p-1)(\alpha+\beta) + \gamma \log_2 p$$

   if p is a power of 2. (2p)

4. (9 p.) Consider matrix-matrix multiplication $C = AB$ 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.

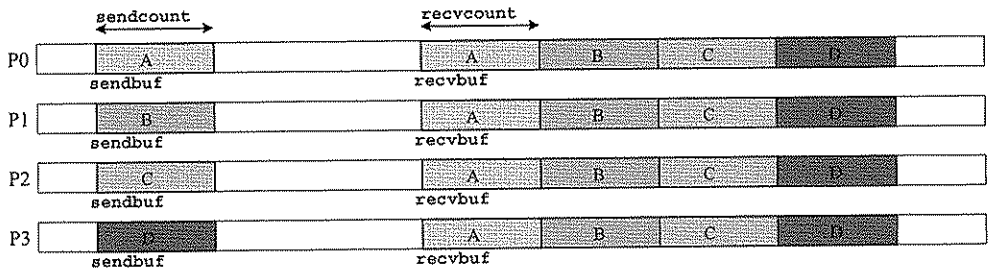

Figure 1: Effect of the `MPI_Allgather` operation on the processes' local memories, here shown for $p = 4$ processes.

5. (7 p.) **MPI**

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

```
int MPI_Allgather ( void *sendbuf, int sendcount, MPI_Datatype sendtype,
                    void *recvbuf, int recvcount, MPI_Datatype recvtype,
                    MPI_Comm comm );
```

`MPI_Allgather` is a generalization of `MPI_Gather`: If executed by a group of p MPI processes each contributing a local array in `sendbuf` with `sendcount` elements of type `sendtype`, each of the p processes will afterwards hold a copy of the entire gathered array in its `recvbuf`, which is composed of the $p \times$`recvcount` elements coming from each send buffer in the order of the processor ranks. See Figure 1.

Usually, the arguments for `sendcount` and `recvcount` and for `sendtype` and `recvtype`, respectively, are equal in calls to `MPI_Allgather`. Also, all participating processes must pass equal argument values for each of these parameters. A typical call could look as follows:

```
MPI_Allgather ( Arr1, n, MPI_FLOAT, Arr2, n, MPI_FLOAT, MPI_COMM_WORLD );
```

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

ii. Write an implementation of `MPI_Allgather` using only `MPI_Send` and `MPI_Recv` operations (i.e., no other communication 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)

iii. Show how your code behaves by drawing an annotated processor-time diagram for $p = 4$ showing *when* messages are sent from *which* source to *which* destination, and *what* they contain. (1p)

iv. Let n denote the value of `sendcount` and `recvcount`, and assume that the `send/recvtype` denotes normal floatingpoint numbers. Analyze asymptotically the (worst-case) *parallel execution time* of your implementation (for arbitrary values of p and n) as a function in p and n. 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. (1.5p)

(b) (1.5 p.) Explain the Communicator concept in MPI. How does it support the construction of parallel software components?

6. (5 p.) **OpenMP**

   (a) (3.5 p.) What scheduling methods (3) for parallel loops are defined in the OpenMP standard? Explain each of them briefly. When should they be used? How does the user setting for the chunk size affect the (expected) performance of the dynamic methods?

   (b) What is the purpose of the `reduction` clause in OpenMP parallel loops? Be thorough! (1.5p)

7. (3.5 p.) **Parallel computer architecture**

   (a) (0.5p) What does *Moore's Law* really say? Be precise!
   (Hint 1: Does Moore's Law still hold today? — Hint 2: Is Moore's Law a law *in the mathematical sense?)*

   (b) What kind of parallelism can be exploited efficiently in modern (general-purpose) graphics processing units (GPUs)? (0.5p)

   (c) 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)

   (d) (1p) Explain the *Crossbar* interconnection network. What are its main strengths and weaknesses when used as the main interconnection network of a parallel computer architecture?