Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	2010 – 01 – 11
Sal	R34
Tid	08:00 – 12:00
Kurskod	TDDC78
Provkod	TEN1
Kursnamn/benämning	Programmering av parallelldatorer – metoder och verktyg
Institution	IDA
Antal uppgifter som ingår i tentamen	11
Antal sidor på tentamen (inkl. försättsbladet)	6
Jour/Kursansvarig	Christoph Kessler, Lars Elden
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

2010-01-11, 08:00–12:00, R34 / R44

Christoph Kessler och Lars Eldén
Institutionen för datavetenskap, Matematiska institutionen
Tekniska Högskolan i Linköping

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

Examinator: Christoph Kessler (TDDC78/TDDB78), Lars Eldén (TANA77)

Jourhavande lärare:
Christoph Kessler (IDA) 013-28-2406; 0703-666687; visiting at ca. 09:30
Lars Eldén (MAI) 013-28-2183; 0760-248842

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.

1. (4 p.) **Performance tools and analysis**

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

   (b) (3 p.) Given a message-passing system with $P = 4$ processors connected by a star network (i.e., there is a direct communication link between any two processors), and $o = 2\mu s$, $g = 3\mu s$, $L = 5\mu s$. Assume that each processor initially generates a single number (32-bit integer) at time 0. How long does it take *at least* to find a minimum of these P values among the processors and to store a copy of the minimum on *all* processors? Use the LogP communication cost model, and draw the processor network graph and a processor-time diagram to explain your answer. (2.5p)

   What is the technical term for the communication pattern(s) used in this parallel computation? (0.5p)

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 OpenMP (short justification)? (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. (1.5 p.) **Parallel algorithm design**

   Explain the algorithmic paradigm *parallel divide-and-conquer*, and give an example of a parallel algorithm (short description of the idea, no details) where this paradigm is used.

5. (5 points) Assume that we have a very large data base of face images `F(1:n)` and that we want to identify an unknown person, i.e, given a photo `Z` of the unknown person we want to find the image in the data base for which the distance `dist(Z-F(i))` is as small as possible. The computation of a distance function is assumed to be expensive.

```
mindist=dist(Z-F(1))
minpers=1
do i=2,n
   if dist(Z-F(i)) < mindist then
      mindist= dist(Z-F(i))
      minpers=i
   endif
enddo
```

   Write a pseudocode that parallelizes this computation on a message-passing system? Explain carefully all aspects of your parallelization. Sketch the code using MPI-like communication.

6. (4 points) The following Fortran code was executed on a certain computer.

```
integer, parameter     :: n=1500
real, dimension(n)     :: x,y
real, dimension(n,n)   :: a
```

```
! ... A is given values

call cpu_time(t0)
do i=1,n
  x(i)=sum(a(i,1:n))
enddo
call cpu_time(t1)
trows=t1-t0

do i=1,n
  y(i)=sum(a(1:n,i))
enddo
call cpu_time(t2)
tcols=t2-t1
write(*,*)trows,tcols
end
```

We got the following results:

```
0.418555021,  0.16701901
```

Explain carefully the difference in timing! What is the cause? Explain the principles of the computer mechanism that causes the difference (without going into hardware details). Is it likely that the same effect will be seen on computers five years from now? Explain why!

7. (6 p.) The following is an OpenMP code for matrix-vector multiplication, where the matrix is $m \times n$. It is assumed that n is a multiple of the number of threads.

```
y(1:m)=0.
!$omp parallel private(?)
   nthr=omp_get_num_threads()
   q=n/nthr                  ! Number of vectors
                             ! in each chunk
   myid=omp_get_thread_num()
   first=myid*q+1
   last=(myid+1)*q
   yp(1:m)=0.                  ! Partial sum
   do j=first,last
      yp(1:m)=yp(1:m)+a(1:m,j)*x(j)
   enddo
   y(1:m)=y(1:m)+yp(1:m)
!$omp end parallel
```

   (a) Which variable(s) must be made private? Explain carefully!

   (b) Explain in detail why the code does not give the correct answer!

   (c) Modify the code so that it gives the right answer (if you do not remember the syntax, explain in detail what the modification is supposed to do)!

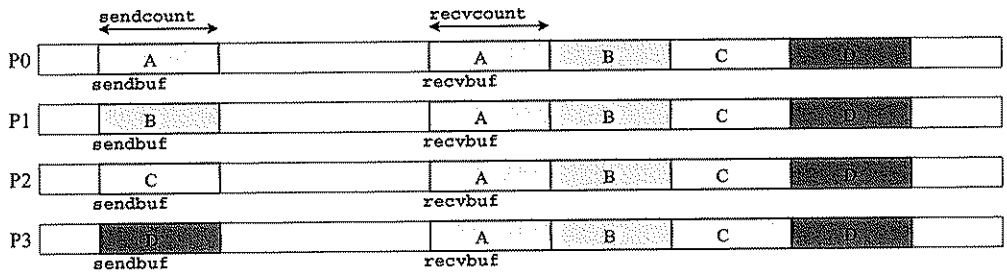

Figure 1: Effect of the `MPI_Allgather` operation on the processes' local memories, here shown for $p = 4$ processes.

8. (6 p.) **MPI**

(a) (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 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. Use C, Fortran, or equivalent pseudocode (explain your language constructs if you are not sure about the right syntax). Explain your code. (2.5p)
Show how your code behaves 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 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 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.

9. (2.5 p.) **Grid Computing**

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

   (b) (1.5 p.) What is the purpose of *grid middleware*? What kind of tasks does it perform?

10. (3 p.) **OpenMP**

   (a) What kind of parallel loops are suitable candidates for static 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.

11. (2 p.) **Parallel computer architecture**

   (a) What kind of parallelism can be exploited efficiently in graphics processing units (GPUs)? (0.5p)

   (b) (1.5 p.)
   (i) Give an example of an interconnection network topology (name and sketch) where the node degree is bounded by a constant.
   (ii) What is the advantage of a constant node degree?