Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	2013 – 08 – 20
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	10
Antal sidor på tentamen (inkl. försättsbladet)	6
Jour/Kursansvarig	Christoph Kessler
Telefon under skrivtid	Se täckbladet (sida 1) av tentan
Besöker salen ca kl.	Se täckbladet (sida 1) av tentan
Kursadministratör (namn + tfnnr + mailadress)	Carita Lilja, IDA, tel. 1463, carita.lilja @ 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

Programmering av parallelldatorer / Programming of parallel computers

2013-08-20, 14:00–18:00, TER1

Christoph Kessler Dept. of Computer Science (IDA)
Linköping University

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

Examinator: Christoph Kessler

Jourhavande lärare:
Christoph Kessler (IDA) 013-28-2406; 0703-666687; visiting at ca. 16:00

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.

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

   (a) (1 p.) Which performance data collection method is required in order to be able to draw a communication statistics diagram (displaying the amount of bytes communicated between every pair of processes)? Justify your answer (technical reasons).

   (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) When is a parallel algorithm for some problem (asymptotically) *work-optimal*? *(give a formal definition)*

   (d) (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.

   (e) (1p) How is the so-called *peak performance* of a parallel computer system such as Triolith calculated? (give a formula)

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. (6.5 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) What does 'hardware multithreading' mean?

   What is the main difference between a hardware-multithreaded processor and a multi-core processor?

   For what kind of code (type of instructions) can hardware multithreading help to improve performance (throughput)? (2p)

   (c) (3p) Explain the *2D-Torus* interconnection network topology. (0.5p)

   Determine its maximum node degree, the maximum distance of any two nodes, and the maximum aggregated bandwidth (given by the maximum possible number of communications proceeding simultaneously in the network). (1.5p)

   Shortly describe one kind of message-passing computation (e.g., from your labs) for which a 2D-Torus network would be more efficient than its 1D equivalent, and why. (1p)

4. (2 p.) **OpenMP**

   (a) In principle, every OpenMP program could likewise be expressed by explicit thread programming, e.g. by using libraries such as Pthreads. What is the main advantage of using OpenMP over such thread programming libraries? Explain your answer. (1p)

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

5. (3.5 p.) **OpenMP Example**

   A novice OpenMP programmer has written the following erroneous program for calculating $\sum_{i=1}^{N}(1/i)$:

```
#include <omp.h>
#include <stdio.h>

#define N 100000000

double s = 0.0;

int main()
{
 int i;
#pragma omp parallel private(i)
 {
#pragma omp for schedule(static)
   for (i=1; i<=N; i++)
      s = s + 1.0/(double)(i);
 }
 printf("Sum: %lf\n", s );
 return 0;
}
```

   (a) The programmer uses static scheduling of the parallel loop iterations. Explain how the iterations are mapped to the executing threads. (0.5p)

   Taking the program as it is (ignoring the correctness problem below), is static scheduling of the loop appropriate here? Explain why or why not. (0.5p)

   (b) The program prints an incorrect result if executed on more than one processor. Where is the bug, and what goes wrong? (1p)

   (c) There are several possible fixes to this problem, which differ considerably in their effect on performance. Suggest a correct solution (OpenMP code) that should lead to *best* achievable performance with multiple processors, and *explain* why. (1.5p; a correct but suboptimal solution with explanation gives 0.5p)

6. (8 p.) **Parallel Basic Linear Algebra**

(a) Given a $n \times n$ single-precision floatingpoint matrix A and a n-element single-precision floatingpoint vector b. For your sequential programming language of choice (C or Fortran, state what you assume), describe a sequential algorithm (pseudocode) for computing the *matrix-vector product* $y = Ab$, with

$$y_i = \sum_{j=0}^{n-1} a_{ij} b_j \text{ for } i = 0, ..., n-1$$

that has *good data locality*, i.e., is cache-efficient. Assume that the (for simplicity, single-level) cache can hold up to $2n+2$ words simultaneously and applies LRU as replacement strategy. State all assumptions carefully and calculate the number of *actual* memory references (i.e., including re-accesses due to cache misses) as a function in n. (2p)

(b) How would you optimize the code if the cache only can hold $K < n$ elements at a time? (sketch of the idea, no code required) (1p)

(c) Describe how to extend the algorithm of (a) above for a message-passing system with p processors each running one MPI process, such that the computational work load is balanced across all processors. Assume that all data initially resides on process 0 and that the result must finally be stored in process 0.

Give a suitable partitioning / distribution scheme for the main data structures and show the pseudocode. Use suitable collective communication routines (pseudocode or MPI) wherever appropriate. *Explain your code.* Draw a figure to show the resulting communication flow for $p = 4$.

Analyze the *parallel execution time* (calculation) using the delay model, assuming for simplicity that the underlying interconnection network is fully connected (i.e., each processor has a direct network link to every other processor) with message startup time α (clock cycles) and average (float) word transfer time β (clock cycles), and that a processor can only send or receive one message at a time. If you need to make further assumptions, state them carefully. (5p)

7. (4 p.) **Parallel Solving of Linear Equation Systems**

Consider the problem of solving a triangular linear equation system, e.g. a system $Ux = b$ where U is a $n \times n$ upper triangular matrix (i.e., all elements below the main diagonal are zero) by *backward substitution*, with the following sequential pseudocode:

$x_{n-1} \leftarrow b_{n-1}/u_{n-1,n-1};$
for $i = n - 2$ **downto** 0 **do**
$$x_i \leftarrow \left(b_i - \sum_{j=i+1}^{n-1} u_{i,j} x_j \right) / u_{i,i};$$

(a) Derive the asymptotic execution time of the above sequential algorithm as an expression in n. (0.5p)

(b) Suggest a possible parallelization for a shared memory system. What kind(s) of parallelism can you exploit here? Show the resulting pseudocode (using e.g. OpenMP, pthreads, or skeletons), and derive the parallel execution time as an expression in n and p. You can ignore cache issues for simplicity.

Calculate the parallel efficiency, and discuss possible performance issues where the number p of parallel threads is large. (3.5p)

8. (1.5 p.) **Transformation and Parallelization of Sequential Loops**

Given the following C loop:

```
for (i=0; i<N; i++)
   a[i] = 2.0 * a[i+1] / b[i] + c[i];
```

(a) Explain why the loop iterations cannot be executed in parallel in this form (dependence-based argument). (0.5p)

(b) Suggest a (semantics-preserving) transformation of the loop (show the new code) such that the loop iterations now could be executed in parallel. Explain why (dependence-based argument). (1p)

9. (3.5 p.) **MPI**

(a) How does the *nonblocking* (also called *incomplete*) send routine `MPI_Isend` differ from the ordinary blocking `MPI_Send`?

Under what condition is it safe to replace a `MPI_Send` call by `MPI_Isend`? (1.5p)

(b) (2 p.) Explain the Communicator concept in MPI. How does it support the construction of parallel software components?

10. (0.5 p.) **Grid Computing**

Name one task performed by *grid middleware*.