Linköping University
Department of Computer and Information Science (IDA)
Prof. Dr. Christoph Kessler

EXAM
TDDC78
Programmering av parallelldatorer – Metoder och verktyg
Programming of parallel computers – Methods and tools

2023-08-16, 14:00–18:00, KY24, KY35, E236(F)

Hjälpmedel / Allowed aids: English dictionary.

Examinator: Christoph Kessler

Jourhavande lärare:
Christoph Kessler (IDA), 013-28-2406; 0703-666687;

Maximalt antal poäng / Max. #points: 40p

Betyg / Grading (prel.): For passing (3), at least 20p (50%) in total. Grade 4 from 28p, 5 from 34p.

Tentavisning / Exam review session: There is no exam review session for re-exams.

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.
- 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.
- Plan 6 minutes per point on average, if you intend to answer all questions.
- The assignments are *not* ordered according to difficulty.

1. (7 p.) **Performance models, tools and analysis**

   (a) (1p.) Which performance data collection method is required in order to be able to draw a processor-time diagram (also known as a *Gantt chart*)? 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) (1.5p) Which aspect of parallel computation is modeled well by the PRAM (Parallel Random Access Machine) model, and which important property / properties of real parallel machines does it abstract from?

   Why should we nevertheless use the PRAM model early in the design of parallel algorithms?

   (d) (2.5p) I need to use some parallel (PRAM) algorithm A with a fixed large problem size, for which I know the parallel execution time as a monotonically decreasing function $t_A(p)$ for p processors, $p \geq 1$. The algorithm A is work-optimal, but unfortunately far from cost-optimal (also in the *asymptotic* sense).

   (i) Knowing this, what can I expect for the behavior of the (relative) *parallel efficiency* of A in dependence of p? (Example: is it likely monotonically increasing with p?) Motivate your answer. (1p)

   (ii) My concrete computation using A needs to be done within a given time budget T_{max} (it must not run longer, but finishing earlier than T_{max} is not important). I also know that $t_A(1) > T_{max}$, hence, using multiple processors is unavoidable here. Describe a simple method or formula (using the above symbols) to determine how many processors I should use *in order to achieve best parallel efficiency*. Explain your solution. (If you need to make additional assumptions, state them carefully.) (1.5p)

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

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.) **Parallel computer architecture**

   (a) (1p) What does *Moore's Law* really say? Be precise!

   (b) (0.5p) What can the TOP-500 list statistics tell us about supercomputer performance in relation to Moore's Law in recent years?

   (c) (2p) Cache-based CPU architectures usually have cache line sizes that contain more than one memory word (currently, 64 bytes are a typical size).

(i) What kind of memory access patterns in programs can make best use of caches with such relatively large cache line sizes, and why? Give a short code example to explain your answer. (1p)

(ii) How does the *hardware* benefit from having such relatively large cache line sizes instead of small ones (like 4 bytes), assuming that the overall cache capacity is not changed? (1p)

(d) (2.5p) What is *false sharing*?

Write a short example OpenMP program (pseudocode; explain your code) that exhibits a *false sharing* problem on a typical shared-memory multiprocessor. Explain why. If you need to make some assumptions about the hardware architecture, state them carefully. Finally, suggest how to fix the problem for your example.

4. (6 p.) **OpenMP**

(a) (1p) Consider the following parallel loop:

```
#pragma omp parallel shared(N)
{
#pragma omp for
        for (i=0; i<N; i++)
            calculate_strange_function( i );
}
```

We know that the execution time of `calculate_strange_function` varies considerably depending on its argument, and that it cannot be predicted at program design time; some calls may take tens of microseconds, while most others return already after less than a microsecond. We also know that the value of `N` is about 1000000 times larger than the available number of cores. Which OpenMP scheduling clause do you recommend for this loop? Motivate your decision.

(b) (3p) The `omp for` construct in OpenMP has a certain restriction that makes it inapplicable to loops such as the following:

```
struct element { double data; struct element *next; } *listhead;
...
// listhead points to the first of many elements in a linked list
...
#pragma omp parallel
{
 ...
 // How to parallelize this loop?
 for (struct element *p = listhead; p!=NULL; p = p->next )
    process_element_data( p->data ); // much computational work...
}
```

even if the processing of the different list elements in the different loop iterations are independent of each other.

(i) What is the restriction? Explain why the OpenMP compiler would complain if we would use the `omp for` directive here. (0.5p)

(ii) How could you suitably parallelize this loop in OpenMP 3 and later? Explain which construct(s) you use (OpenMP pseudocode). (1.5p)

(iii) Explain for the OpenMP (3+) language construct used in (ii) how OpenMP technically implements the parallel execution. (1p)

(c) (2p) What does the `flush` directive in OpenMP do, and where and why do we need to put it? Be thorough!

5. (5 p.) **Parallel Basic Linear Algebra**

(a) (0.5p) Give an example of a Level-2 BLAS function (short answer: name and what it does)

(b) (1.5p) Name and shortly describe one sequential algorithm for matrix-matrix multiplication that fundamentally *differs* in its structure from the well-known loop-based textbook algorithm with the three nested loops. (1p)

Which fundamental parallel algorithmic design principle is exploited to identify parallel computations in this algorithm? (Short answer, 0.5p)

(c) (3p) How does the SUMMA algorithm for distributed-memory parallel matrix-matrix multiplication work? (Principle and pseudocode) (2p)

How does its inter-process communication structure differ from that of the systolic matrix-multiplication algorithms (Kung/Leiserson and Cannon) that we discussed in the lecture? (1p)

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

(a) (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?

(b) (4 p.) Given a cluster with p nodes connected by some network. Assume we have p processes, each process running alone on a processor on its own node in the cluster, and each processor holds a local array of N integer elements.

Write a MPI program for p processes that computes the *global maximum* of all pN array elements in parallel and that uses only point-to-point communication (i.e., send and receive operations, not collective communication operations). To obtain full points, your program should be *time-optimal* and scale well for large p.

Draw a processor-time diagram to show the communication over time.

Then, use the LogP model to analyze the execution time, work and cost of your program (as formulas of N, p and possibly some other model parameters as appropriate). (Hint: If you do not recall the LogP model, use the Delay model instead.)

(c) (2p) Explain the principle of *one-sided communication* in MPI. How does it work, and what are the fundamental operations for one-sided communication between MPI processes? Illustrate your answer with an annotated drawing to show what happens.

7. (4 p.) **Transformation and Parallelization of Sequential Loops**

(a) Why is it, in general, so hard for (C) compilers to *automatically* parallelize loops, by just analyzing the source code? (1p)

(b) What is *tiling* of a (sequential) loop nest, in general? Give also a short C code snippet to illustrate your answer. (Hint: Make sure to name the two sub-transformations that tiling is based on.) (1p)

(c) Given the following C loop:

```
a[0] = b[0] + c[0];
for (i=1; i<N; i++) {
   a[i] = a[i-1] + b[i] + c[i];
}
```

(Assume for simplicity that the arrays `a`, `b` and `c` do not overlap in memory.)

i. Can the loop iterations be executed in parallel in this form? Explain why or why not (dependence-based argument). (0.5p)

ii. What kind of computation does this loop actually do? (0.5p)

iii. Suggest a *time-optimal parallel algorithm* for the same problem that could utilize up to N processors in parallel in the EREW PRAM model (give the basic idea and asymptotic parallel time complexity, but no details). (1p)