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

EXAM

TDDE65/TDDC78

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

2024-05-30, 08:00–12:00, U6, P42, P44

Hjälpmedel / Allowed aids: English dictionary.

Examinator: Christoph Kessler

Jourhavande lärare:
Christoph Kessler (IDA), 013-28-2406; 0703-666687; visiting ca. 08:30 and possibly later

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: for the main exam, about 2.5-3 weeks after the exam, to be announced on the course webpage; afterwards, the exams will be archived in the IDA student expedition. 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. (8 p.) **Performance models, 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) (1p.) Which performance data collection method is required with MPI programs in order to be able to draw a processor-time diagram with an arrow for every message (as in VAMPIR/ITAC)? Justify your answer (technical reason).

   (c) (1p) Modern tool-suites for performance analysis of parallel programs consist of a collection of several kinds of software 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) (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?

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

   (f) (2 p.) (*i*) Derive Amdahl's Law and (*ii*) give its interpretation.

   (Note: a picture is nice but is not a proof; a calculation is expected for the derivation of Amdahl's Law.)

   (g) (1 p.) Amdahl's Law and Gustafson's Law differ only in a single assumption about the application's performance behavior. Which one, and how?

2. (4 p.) **Parallel program design methodology**

   Foster's methodology for parallel program design is separated into four stages, commonly abbreviated *PCAM*. In which of the four stages do the following actions or statements apply, if any? Answer by giving **one** of the letters P, C, A, M, or "none" for each statement.

   (a) Dependence structure between tasks is identified.

   (b) Macrotasks are assigned a core to run on.

   (c) Size of the program input ("problem size") is taken into account.

   (d) The input to this stage is a "textbook-style" parallel algorithm.

   (e) Subtrees from parallel divide-and-conquer are coalesced.

   (f) Barrier synchronization may be introduced.

   (g) Load balancing is taken into account.

   (h) Source code for each task is written in a language such as C, C++, or Fortran.

3. (5 p.) **Parallel computer architecture**

   (a) (1p) Why is it not sufficient to rank the TOP-500 list simply by the *peak performance* of parallel computer systems (as defined in the lecture)? Explain on which basis the TOP-500 list ranking is made instead. (1p)

   (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) (1p) What is the difference between *temporal locality* and *spatial locality* of memory accesses in cache-based architectures?

   (d) (2.5p) *False Sharing*

   (i) What is *false sharing*?

   (ii) 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. (7 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) What does the `reduction` clause do in a parallel `for` loop in OpenMP? In what cases should the `reduction` clause be applied? What argument(s) does `reduction` take, and what requirements does this argument / do these arguments have to fulfill? What is/are the potential advantage(s) of using `reduction` compared to solving the problem "manually" using other OpenMP constructs?

   Give also one example loop (OpenMP code without `reduction`), explain how the generated code for the loop will work instead when using `reduction` (be thorough).

   (c) (1p) What is the effect of using the `nowait` clause in an `omp for` directive, compared to the default behavior (i.e., not using it)?

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

5. (6 p.) **Parallel Basic Linear Algebra and Linear System Solving**

   (a) We learned about the sequential BLAS-1 routine DAXPY, which (re-)calculates a vector $y \leftarrow \alpha x + y$ from vectors x, y and a scalar α. Is this computation rather *memory-bound* or *CPU-bound* on modern computer systems? Justify your answer. (1p)

   (b) (0.5p) Give an example of a Level-3 BLAS function (short answer: name and what it does)

   (c) (0.5p) Why is it very important for a supercomputing center to have efficient implementations of BLAS installed? (Short answer)

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

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

   (e) (1p) In distributed-memory LU decomposition as presented in the lecture, we encountered a load-balancing problem for certain types of partitioning / distribution of the system matrix across the processes.

   (i) How did we fix this load balancing problem, to a major extent? (short answer, 0.5p)

   (ii) Would the same trick also improve the performance of the SUMMA algorithm (see (c))? (short explanation, 0.5p)

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

   (a) (1p) What does the operation `MPI_Allreduce` do? And how does it differ from the operation `MPI_Reduce`?

   Hint: The parameter types are as follows:

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

   (b) (2 p.) MPI supports *nonblocking* (also called *incomplete*) point-to-point communication. What does that mean?

   Give an example scenario (with MPI pseudocode) 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?

   (c) (2 p.) Explain the *communicator* concept in MPI, and how it supports the safe reuse of MPI software modules contributed by third-party developers.

   (d) (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. (3 p.) **Transformation and Parallelization of Sequential Loops**

   (a) (0.5p) Give an example (explained pseudocode) for a *loop-carried* data dependence in a sequential `for` loop.

   (b) (1.5p) Name and explain (including a simple example) a loop (or loop nest) transformation that can affect the *data locality* (and thus, cache performance) of the loop.

   (c) (1p) Is the following C loop parallelizable (in this form)?

```
for (i=0; i<N; i++) {
   t[i] = t[i] + sin(3.1415 * t[i-1] + a[i]);
}
```

   Explain why or why not (dependence-based argument).