TENTAMEN / EXAM

TDDC78

Programmering av parallelldatorer /
Programming of parallel computers

2018-05-30, 14:00–18:00, TER1, TERE

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

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

Examinator: Christoph Kessler

Jourhavande lärare:
Christoph Kessler (examiner) 013-28-2406; 0703-666687, visiting 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 regulations by Linköping University, we do not give ECTS grades. If you need one, please contact the course secretary after the result has been entered in LADOK.

Tentavisning / Exam review: about 2-3 weeks after the exam, to be announced on the course homepage. Afterwards, the exams will be archived in the IDA student expedition.

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.
- The assignments are *not* ordered according to difficulty.

1. (6 p.) **Performance tools and analysis**

   (a) (1p) What kind of performance data and performance data collection method could be helpful to get indications for a possible *false sharing* issue in a multi-threaded shared-memory parallel program running on a standard multicore CPU? Motivate your answer.

   (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) (2p) Which property/properties of real parallel computers is/are modeled well by the BSP (Bulk-Synchronous Parallel) model, and which property / properties of real parallel computers does it abstract from?

   And what kind of restrictions does BSP require on program structure?

   (d) (1p) When is a parallel algorithm for some problem (asymptotically) *cost-optimal*? *(give a formal definition)*

   (e) (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 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) How is the (theoretical) *peak performance* of a parallel computer system defined? (1p)

   (b) Identify two important trends in supercomputer architectures that can be derived from the (recent) TOP-500 lists. (1p)

   (c) Many bus-based shared-memory systems use *bus-snooping* for cache coherence. What does that mean, and how does it work? (1p)

   (d) Explain the difference between a *hardware-multithreaded* processor and a *multi-core* processor. (1p)

   (e) (2p) The Infiniband network used to interconnect the nodes of Triolith is based on a so-called *fat-tree* network. Describe and explain

   (a) the topology (structure) of a *fat-tree* interconnection network, and

   (b) how it improves the scalability of communication bandwidth over ordinary (leaf-oriented) tree networks;

   (c) the longest distance between any two nodes.

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 several seconds, others return already after a millisecond. We also know that the value of `N` is about 10 times larger than the available number of cores. Which OpenMP scheduling clause do you recommend for this loop? Motivate your decision.

   (b) (2.5p) What kind of loop-based computations can benefit from using the `reduction` clause in OpenMP?

   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), and give 2 main reasons why using `reduction` is likely to improve performance.

   (c) (2.5p) OpenMP 3.0 and later versions support the *task* concept.

   In what situations (e.g., for what kind of loops) can the use of tasks help to parallelize the execution, while the traditional OpenMP worksharing constructs are not applicable? Give an example (pseudocode, explain your program constructs) of such a case for illustration and explain how the tasks are defined in OpenMP 3+.

   And how does OpenMP map the tasks to processors?

   (d) What is the main difference between a *PGAS programming language* and a shared memory programming language like OpenMP? (1p)

5. (9 p.) **Parallel Basic Linear Algebra**

   (a) (3.5p) We discussed 2 variants of (sequential) matrix-vector multiplication, the ij variant and the ji variant.

   (i) Choose one of the two variants, show its sequential pseudocode (state whether you assume C or Fortran memory layout for 2D arrays), and show for this variant which level-1 BLAS function(s) it could use as subroutine(s). (1.5p)

   (ii) Describe the most appropriate way of parallelizing this variant for a message-passing parallel system. In particular, how should the main data structures be distributed? (2p)

(b) (3p) The BLAS (Basic Linear Algebra Subroutines) library API comes in three different levels (BLAS Level 1, 2, 3).

(i) Name one representative function from each level and give a short description. (1.5p)

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

(iii) Which of the levels is most likely to have efficient parallel implementations even where interprocessor communication is expensive, and why? (1p)

(c) (2.5p) We discussed two *systolic* parallel algorithms for matrix-matrix multiplication in the lecture (Kung-Leiserson and Cannon's algorithm).

(i) What kind of interconnection network topology would be suitable for these systolic algorithms, and why? (0.5p)

(ii) In the beginning of these algorithms, we need to transform the input matrices. How is this done (for one of these 2 algorithms), and why is it necessary? (1p)

(iii) How does the communication structure in the systolic algorithms for matrix-matrix multiplication (e.g., Cannon's algorithm) look like, and how does it differ from that of non-systolic algorithms such as SUMMA? (1p)

6. (1 p.) **Parallel Solving of Linear Equation Systems**

When considering the second step in Foster's design method, we discussed as an example the update schemes of iterative solver methods for linear equation systems, including *Jacobi*-style and *Gauss-Seidel*-style update schemes, over the matrix elements in each iteration step. Choose one of them (state which one), and explain its dependence structure between element updates and its maximum degree of parallelism within one iteration of the algorithm.

7. (3 p.) **Transformation and Parallelization of Sequential Loops**

(a) What is *tiling* of a loop nest, in general? (1p)

Explain why tiling can considerably improve the performance of sequential matrix-matrix multiplication on today's computer architectures. (1p)

(b) Is the following C loop parallelizable (in this form)?

```
    for (i=1; i<N; i++) {
S1:    a[i] = sin(3.1415 * t[i-1]) + b[i];
S2:    b[i] = 1.0 / a[i] + t[i];
    }
```

Explain why or why not (dependence-based argument). (1p)

8. (4 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) (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.

   (c) (1 p.) What does the `MPI_Win_get` operation in one-sided communication do?