TENTAMEN / EXAM

TDDC78

Programmering av parallelldatorer / Programming of parallel computers

2017-06-03, 14:00–18:00, TER2/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 (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 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, 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. **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. (5.5 p.) **Performance tools and analysis**

   (a) (0.5p) Give one typical example for performance-related data about *message passing* programs that can be collected using *software counters*.

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

   (c) (2p) Given is an idealized processor with a fully associative last-level cache of size M memory words, with perfect LRU replacement policy and a cache line size of 1 memory word. Estimate the total number of last-level cache misses (including cold misses) for the following sequential program snippet, depending on N, M and K:

```
// A is an array of at least N+K memory words,
//    initially not in cache.
// K > 1, K < N is fixed for this loop but not statically known.
// Accumulator variable s is kept in a register during the loop.

s = 0.0;
for (i=K; i<N+K; i++)
   s += A[i] * A[i-K];
```

   Hint: You need to distinguish between several cases.

   (d) (1 p.) A long-running program is known to have a perfectly parallelizable part that accounts for 90 percent of its sequential execution time. The rest is inherently sequential. If we parallelize the program, how much parallel speedup can we expect with 9 processors? Explain your calculation.

   (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. (3.5 p.) **Parallel computer architecture**

   (a) (2p) Cache-based architectures usually have cache line sizes that contain more than one memory word.

   (i) Describe, in general, the benefit(s) of having large cache line sizes instead of small ones (quantitative argument). What kind of memory access patterns can make best use of caches with large cache line sizes? (1.5p)

   (ii) For a cache-based shared memory parallel system with coherent caches, in what situation(s) would a small cache line size probably be better from a performance point of view? (0.5p, *short answer*)

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

(c) The Infiniband network used in the Triolith supercomputer is a so-called *Fat-tree* network. How does the *network diameter* in a Fat-tree network scale with the number of nodes? (0.5p)

4. (7 p.) **OpenMP**

(a) (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.

(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.

(c) 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? And how does OpenMP map the tasks to processors? (1.5p)

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

5. (8.5 p.) **Parallel Basic Linear Algebra**

(a) (2p) Name 2 different Level-1 BLAS functions and explain for each of them how it could be parallelized on a shared-memory parallel system.

(b) (4p) 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). (2p)

(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)

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

(i) What is a systolic parallel algorithm, in general? (1p)

(ii) 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)

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

6. (2 p.) **Parallel Solving of Linear Equation Systems**

   Consider the message-passing parallel implementation of Gaussian Elimination / LU decomposition as discussed in the lecture. Which data distribution(s) for the matrix A is/are most appropriate, and why? (2p) *(Hint: If you need to make certain assumptions, e.g. about the used programming language, state them carefully.)*

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)?

```
S0: t[0] = 1.0;
    for (i=1; i<N; i++) {
S1:    a[i] = sin(3.1415 * t[i-1]) + b[i];
S2:    t[i] = 1.0 / a[i] + c[i];
    }
```

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

8. (6.5 p.) **MPI**

   (a) (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.

   (b) (5.5 p.) **Simple String Matching in MPI**
   Given is a huge array A of N characters stored in the main memory of one node of a cluster computer with P (for simplicity, fully connected) nodes ($P > 1$).
   Given is also a short constant array S of M characters (with $M \ll N$, and M can be considered constant) that is available on node 0 at program start.

      i. Write a message-passing parallel program (MPI-C or message passing pseudocode is fine, explain your code) that uses all P nodes in order to count the *total number* of (contiguous) occurrences of the string S in A, i.e., the number of all positions $i < N - M$ of A where, for all $j = i, ..., i + M - 1$, $S[j] == A[j]$. Use a simple brute-force parallel algorithm for string matching like the one that we discussed in the lecture. Be thorough, and explain your code carefully. (3p)

      ii. Which type of parallelism did you use in your program in (i)? (0.5p)

      iii. Make sure to explain all communication operations in (i). Draw a figure for $P = 4$ nodes showing the distribution of A and the flow of messages between the nodes over time. Which of these communication operations are point-to-point communications and which ones are collective communication operations? (1p)

      iv. Derive the asymptotic worst-case parallel execution time for your algorithm as an expression (use big-O notation where appropriate) in N and P. (1p)