COMP 60611

Three hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Parallel Programs and their Performance

Date: Thursday 17th January 2019

Time: 09:45 - 12:45

Please answer BOTH Questions

Each question is worth 20 marks

This is an OPEN book examination.

The use of electronic calculators is permitted provided they are not programmable and do not store text

[PTO]

Question 1.

An important first step to being able to increase the performance of a parallel code is to quantify how well it performs when executed using different numbers of cores.

a) Speedup and efficiency are two measures that are used to describe the run-time performance of a parallel code. Define these quantities, clearly stating the measured quantities on which they are based.

(2 marks)

b) The performance of a given parallel code is being analysed and a speedup graph is drawn to illustrate how the code performs on different numbers of cores. The following features of the code are deduced from the speedup graph:

- The code scales well, showing a close-to-linear speedup for low-to-medium numbers of cores.
- When using 6, or more, cores, the data associated with the code fits into core-local cache memory, as opposed to runs on fewer than 6 cores, when data must be repeatedly fetched from memory.
- At 20 cores software parallelism has been exhausted, and beyond this the execution time falls slowly as cores increases.

Given these features of the code, sketch how you would expect the speedup graph to appear; you should ensure that the axes of the graph are clearly labelled. By annotating the graph, identify the parts of the graph that correspond to each of the code features listed above.

(8 marks)

c) Explain why Amdahl's Law cannot adequately account for all of the behaviour illustrated in your speedup graph for part b). You will need to identify clearly those code features that are not explained by Amdahl's Law.

(4 marks)

d) An implementation of a vector addition algorithm has been written using OpenMP on a mcore48-like machine. In the implementation, the operation loop is parallelised but the loop initialising the arrays is not parallelised. You may assume all the data fits in the private caches of a single core:

(i) Identify the main sources of overhead you would expect to see in this implementation.

(2 marks)

(ii) In the analysis of overheads, a term of the form (p-1)/p, where p is the number of cores, often appears. Explain, with the help of diagrams, how the (p-1)/p term helps to explain any of the overheads you identified in part (i).

(4 marks)

Question 2.

In OpenMP, the "omp for" work-sharing directive allows the programmer to specify one of the following four distinct kinds of schedule:

simple static (block scheduling)
interleaved (block-cyclic scheduling)
simple dynamic (chunk self-scheduling)
guided dynamic (guided self-scheduling)

a) Briefly explain the operation of each schedule.

(4 marks)

b) For the following C-like fragment, identify the chief source(s) of parallel overhead you would expect to see. Then, by analysing the behaviour of each of the available kinds of schedule when applied to the fragment, suggest a suitable choice for the most appropriate kind of schedule, plus chunk size, if appropriate, for the "omp parallel for" directive. Give reasons for your choice. Assume that the target architecture is a quad 12-core Opteron system, like mcore48, and **do not** consider any restructuring of the code.

```
#pragma omp parallel for
for (k=0;k<n;k++) {
  if (b[k]== true) process(a[k]);
}
```

In the above code fragment, *n* may be assumed to be large and `b[]` is an array of Boolean variables indicating, if set to true, that the equivalent element, *k*, of the array `a[]` should be further processed. `process()` is a function that overwrites its argument and whose execution time grows as $k^{3/2}$. You may assume that approximately 50% of the elements of `b[]` are set to true, but the precise distribution of true values is not known until run time.

(5 marks)

c) Several other parallel programming language technologies have been developed, including Chapel, Co-array Fortran, UPC, CUDA, OpenCL and X10. Select one of these programming languages, or choose another language with which you are familiar, and compare and contrast them with OpenMP and/or MPI.

(5 marks)

d) Define and give an example of what is meant by *false sharing.* Discuss whether false sharing can arise in OpenMP, MPI and the technology you discussed in part (c) and explain any trade-off involved in implementing a solution to avoid false sharing in your example.

(4 marks)

e) OpenMP, and other shared memory programming languages, provide *synchronisation mechanisms* often described by the term *locks* (for example the CRITICAL and omp_set_lock()/omp_unset_lock() constructs in OpenMP). Briefly discuss why these constructs are necessary and describe the type(s) of overhead that they can be expected to incur when used in an application.

(2 marks)

END OF EXAMINATION