COMP 60611

Three hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Parallel Programs and their Performance

Date: Tuesday 17th January 2017

Time: 14:00 - 17:00

Please answer any TWO Questions from the FOUR Questions provided.

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.

a) Explain what is meant by the execution time 'overheads' of a parallel program (you should clearly identify each different kind of overhead you might expect to occur). Describe how these overheads affect execution of a parallel program.

(6 marks)

In the following OpenMP/Fortran program, which is used in parts (b) and (c) of this question, any subroutine parameters which get written into during a `CALL` are shown underlined. The subroutines `INITIALISE` and `DO_WORK` both write to entirely independent elements of their array parameters.

```
!
program examQ
real a(100000), b(100000), c(100000)
integer i

do i = 1, 100000
    call initialise(a, b)
end do
!$omp parallel do
do i = 1, 100000
    call do_work(a, b, c)
end do

call finalise(c)
end
```

Timers were inserted around the whole program, and the code was executed on a NUMA architecture multi-core computer, similar to mcore48, one thread per core, with various numbers of active threads/cores. This yielded the following execution times (in milliseconds):

Number of active threads/cores	Execution time (ms)
1	811.60
2	413.66
5	215.44
10	115.26
20	65.46
50	28.02
100	21.87
200	18.76

The scheduling and synchronisation overheads were measured by experiment and were found to be constant (i.e. independent of the number of threads) at 0.02 ms. Another experiment established that the sequential calls to `INITIALISE` and `FINALISE` altogether took a total of 11.58 ms to execute, thus defining the non-parallel code overhead for the program.

b) There are clearly other sources of overhead contributing to less-than-ideal parallel performance for this program.

   i) Quantify the amount of each kind of overhead observed as the number of threads varies.

   (3 marks)

   ii) Explain possible sources of the additional overhead, and give details of any experiments you would perform in order to establish which of these potential sources actually gives rise to any of the observed overhead.

   (6 marks)

c) Assuming that there is only one source for the additional overhead, and making reasonable assumptions about this source and the internal nature of the subroutines `INITIALISE` and `DO_WORK`, suggest ways in which you might change the program so that it executes more quickly for larger numbers of active threads/cores.

(5 marks)

[PTO]

Question 2.

(a) List and compare the different scheduling options available for the OpenMP DO statement.

(3 marks)

(b) Identify, with reasons, the scheduling option that would be most appropriate for the DO statements in the two code fragments given below in (i) and (ii), providing sketches of the iteration space, where appropriate. You can assume that the target parallel computer architecture is similar to that of the multicore Opteron-based computer mcore48; you should not consider any restructuring of the code.

i)
```
!
! sequential loop
!
DO i=1,nstep
  ...
  ...
  ...
!$omp parallel do private (j) shared (a)
  DO j=1,100000
    IF (isSquare(j)) THEN
       CALL sub(a(j))
    END IF
  END DO
END DO
!
```

Here, `isSquare` is a logical function which returns `.TRUE.` if and only if its integer argument is a perfect square (integer j is a perfect square if there exists an integer k such that $j = k^2$), and `sub` is a subroutine which overwrites its argument and whose execution time is independent of its argument and is significantly greater than the execution time of `isSquare`. You may assume that there are a substantial number of sequential iterations (i.e. `nstep` is large).

(5 marks)

ii)

```
parameter (n = 40000)
real a(n,n), b(n)
!$omp parallel do private (i, j) shared (a, b, n)
do j = 1, n
   b(j) = 0.0
   do i = 1, n
       a(i,j) = i + j
   end do
end do

!$omp parallel do private (i, j) shared (a, b, n)
do j = 1, n
   do i = 1, j-1
       b(j) = b(j) + a(i,j)
   end do
   do i = j, n
       b(j) = b(j) + a(j,i)
   end do
end do
```

(5 marks)

c) State Amdahl's Law, clearly defining all the terms used. Explain how speedup is affected by Amdahl's Law as the number of cores increases.

(4 marks)

d) Explain the term "superlinear speedup" and sketch a performance curve to illustrate the effect. Explain why Amdahl's Law cannot adequately account for this effect and give an example of a plausible cause.

(3 marks)

[PTO]

Question 3.

a) Explain briefly, giving a simple example, and compare each of the following terms: data parallelism, reduction parallelism, divide-and-conquer parallelism, task parallelism and "embarrassingly" or "pleasingly" parallel.

(4 marks)

The following OpenMP/Fortran-like pseudocode, which is used in parts (b) and (c) of this question, implements a parallel divide-and-conquer algorithm using a shared stack to hold the outstanding jobs that need to be computed. The subroutine `pop` returns the special value `null` if it is executed when the stack is empty.

```
Do parallel
shared  stack, output, terminated
private job, job1, job2, result

do while (.not. terminated)
   critical (stack)
     pop(top of stack into job)
   end critical
   if (job .ne. null)
      if (job is large)
         create 2 subjobs, job1 & job2
         critical (stack)
            push(job1 onto stack)
            push(job2 onto stack)
         end critical
      else
         compute result (of job)
         critical (output)
            add result to output
         end critical
      end if (job is large)
   end if (job .ne. null)
   critical (terminated)
      compute terminated
   end critical
end do while
```

b) Explain what needs to be done when the shared termination condition `terminated` is computed. Briefly describe a strategy for implementing this.

(3 marks)

c) Explain clearly what you expect to be the main source(s) of parallel execution time overhead for this code. State your assumptions about the behaviour of each part of the algorithm, and make it clear what you expect to happen as the time to compute `result` increases from being relatively short to being relatively long, compared with the rest of the necessary work.

(5 marks)

d) A programmer on your team suggests the following change to the above pseudocode: instead of pushing both new subjobs onto the stack, push only one of them and then execute the other in the existing thread. Write new pseudocode (in the same style as above) that achieves this. What effect do you expect this change to have on the execution time overheads you identified in your answer to part c)?

(4 marks)

e) Discuss the difficulties that would need to be addressed if P stacks (one per thread, as opposed to a single shared stack) were used in a parallel implementation of this algorithm using P threads. What effect do you expect such a change to have on the execution time overheads you identified in your answers to parts c) and d)?

(4 marks)

[PTO]

Question 4.

(a) Consider the computation represented by the following fragment of Fortran code.

```
integer i, j, k, n
real a(n,n), b(n,n), c(n,n)
do i = 1,n
   do j = 1,n
      c(i,j) = 0.0
      do k = 1,n
         c(i,j) = c(i,j) + a(i,k) * b(k,j)
      end do
   end do
end do
```

For a parallel **message-passing** implementation of this algorithm on a $p \times p$ array of processors (p^2 processors in total), describe:

i) A block-row partitioning of the data;

ii) A block-column partitioning of the data;

iii) A block partitioning of the data, with square blocks of size $q \times q$, where $q = n/p$.

Derive the volume of data communication per processor implied by each of the above data distributions. Show that, for $p > 1$, the volume of communication is minimised when we choose to partition a, b, c into square blocks of sizes $q \times q$, i.e., partition iii) above. For simplicity you may assume that p^2 exactly divides n. (10 marks)

(b) For the code fragment in part (a), provide OpenMP-like source code implementations exploiting the following parallelism, (the OpenMP syntax need not be exact as long as the intention is clear):

(i) Parallelising the outer *i* loop only.

(ii) Parallelising the inner *k* loop only.

Identify and compare, using diagrams of the iteration space, where appropriate, the expected overheads incurred in the two implementations and describe the impact of the overheads on performance as the number of threads used increases. State any assumptions you make on how the arrays are initialised and on the problem size relative to cache size. Do not consider any code changes. (7 marks)

(c) Assuming OpenMP provided a mechanism to parallelise both the *i* and *j* loops at the same time, with which the programmer could specify the size of partition in each dimension, compare an OpenMP-like implementation using this mechanism with the square block message-passing implementation described in part (a) (iii) in terms of ease of partitioning of the data and expected performance. (3 marks)

END OF EXAMINATION