Parallel Programs and their Performance

Date: Friday 24th January 2014
Time: 09:45 - 12:45

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.
Question 1.

Indirection of array accesses, via an intermediate index array, is a technique that is commonly encountered in scientific simulation codes. For example, in the following sequential loop nest (a fragment from a typical Fortran program), \( j(n) \) is the integer index array for accesses to the real array \( a(n) \):

```fortran
C pertinent variable declarations
C INTEGER k, n, ix
PARAMETER (n=1000000)
INTEGER j(n)
REAL a(n), b(n), c(n)
:
:
:
C loop nest code fragment starts here
C DO k = 1, n
   DO ix = 1, k
      a(j(ix)) = a(j(ix)) + b(k) + c(ix)
   END DO
END DO
:
:
END
```

The integer index array \( j(n) \) effectively holds pointers to the elements of the real array \( a(n) \), the pointers themselves being accessed in ascending order by the innermost loop index \( ix \). Note that the value of \( j(ix) \) need not be unique. That is, \( j(ix) \) may take the same integer value, say \( v \), for several distinct values of \( ix \). However, \( v \) always satisfies \( 1 \leq v \leq n \). By this means, selected elements of \( a(n) \) are updated according to the values held in \( j(n) \).

a) Using diagrams where appropriate, describe the patterns of access to the arrays \( a, b \) and \( c \) as the loop indices \( k \) and \( ix \) vary, and comment on the nature of any computational work in the loop nest that might be performed in parallel. Remember that \( n \) is large (a million).

   (4 marks)

(Question 1 continues on the following page)
b) The following is one possible parallel implementation of the above loop nest, using a data-sharing programming model (OpenMP directives – the ATOMIC directive ensures that each element \( a(j(ix)) \) is updated by only one thread at a time):

```c
DO k = 1, n
   !$OMP PARALLEL DO
      DO ix = 1, k
         !$OMP ATOMIC
         a(j(ix)) = a(j(ix)) + b(k) + c(ix)
      END DO
   !$OMP END DO
END DO
```

Describe, for a NUMA architecture like that of chronos, the behaviour of this implementation, and discuss its potential performance in terms of overheads incurred due to synchronisation, load imbalance and remote memory accesses. State clearly any assumptions you make about the contents of the array \( j(n) \).

(8 marks)

c) Suggest a more efficient parallel implementation for the given loop nest. Explain how your revised implementation reduces the overheads identified in your answer to part b).

(8 marks)
Question 2.

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 the parallel program.

(6 marks)

In the following OpenMP/Fortran program, 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.

```c
C PROGRAM examQ2
REAL a(100000), b(100000), c(100000)
INTEGER i

DO I = 1, 100000
   CALL INITIALISE(a, b)
END DO
!$OMP PARALLEL DO
DO I = 1, 100000
   CALL DO_WORK(a, b, c)
END DO
!$OMP 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, like chronos, one thread per core, with various numbers of active threads/cores. This yielded the following execution times (in milliseconds):

<table>
<thead>
<tr>
<th>Number of active threads/cores</th>
<th>Execution time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1005.28</td>
</tr>
<tr>
<td>2</td>
<td>506.31</td>
</tr>
<tr>
<td>5</td>
<td>206.91</td>
</tr>
<tr>
<td>10</td>
<td>107.11</td>
</tr>
<tr>
<td>20</td>
<td>57.21</td>
</tr>
<tr>
<td>50</td>
<td>27.27</td>
</tr>
<tr>
<td>100</td>
<td>17.29</td>
</tr>
<tr>
<td>200</td>
<td>12.30</td>
</tr>
</tbody>
</table>

The scheduling and synchronisation overheads were measured by experiment and were found to be constant at 0.01 ms per thread. Another experiment established that the sequential calls to INITIALISE and FINALISE altogether took a total of 5.27 ms to execute, thus defining the non-parallel code overhead for the program.

(Question 2 continues on the following page)
b) There are clearly other sources of overhead contributing to less-than-ideal parallel performance for this program. Quantify the amount of each kind of overhead observed as the number of threads varies, 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.

(9 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 two subroutines, suggest ways in which you might change the program so that it executes more quickly for larger numbers of active threads/cores.

(5 marks)
Question 3.

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 4, or more, cores, the data associated with the code fits into cache memory, as opposed to runs on fewer than 4 cores, when data must be repeatedly fetched from memory.
- At the highest numbers of cores shown on the graph, communication dominates computation and the code actually takes longer to run than on smaller numbers of cores.

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.

(9 marks)

c) State Amdahl’s Law, clearly defining all the quantities used. Explain why Amdahl’s Law cannot adequately account for all of the behaviour illustrated in your speedup graph. You will need to identify clearly those code features that are not explained by Amdahl’s Law.

(9 marks)
Question 4.

a) Consider Strassen’s Algorithm for the calculation of the matrix product \( C = A \times B \) where \( A, B \) and \( C \) are \( N \times N \) matrices. Assuming that \( N \) is even, divide each of \( A, B \) and \( C \) into the \( 2 \times 2 \) block matrices

\[
A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix},
\]

where the blocks \( A_{ij}, B_{ij}, C_{ij} \) are \( N/2 \times N/2 \) matrices. Given the intermediate matrices

\[
P_1 = (A_{11} + A_{22})(B_{11} + B_{22}), \\
P_2 = (A_{11} + A_{22})B_{11}, \\
P_3 = A_{11}(B_{12} - B_{22}), \\
P_4 = A_{22}(B_{21} - B_{11}), \\
P_5 = (A_{11} + A_{22})B_{22}, \\
P_6 = (A_{21} - A_{11})(B_{11} + B_{12}), \\
P_7 = (A_{12} - A_{22})(B_{21} + B_{22}),
\]

show that the blocks of \( C \) are given by

\[
C_{11} = P_1 + P_4 - P_5 + P_7, \\
C_{12} = P_3 + P_5, \\
C_{21} = P_2 + P_4, \\
C_{22} = P_1 + P_3 - P_2 + P_6.
\]

(4 marks)

b) Considering floating point operations (flops) only, show that conventional matrix multiplication of two \( N \times N \) matrices requires \( 2N^3 - N^2 \) flops, while Strassen’s Algorithm, applied without recursion, requires approximately

\[
\frac{7}{4} N^3 + \frac{11}{4} N^2 \text{ flops.}
\]

(6 marks)

c) Given that \( N = 2^n \), where \( n \) is a positive integer, describe a recursive version of Strassen’s Algorithm. Show that this algorithm is more efficient than the conventional algorithm for \( 16 \times 16 \) matrices, but not for \( 8 \times 8 \) matrices. Hence or otherwise show that it is optimal to recur Strassen’s algorithm until the submatrices are \( 8 \times 8 \).

(6 marks)

d) Describe briefly how you would approach implementation of Strassen’s Algorithm on a parallel computer such as chronos, and identify any major obstacles to good parallel performance that you see.

(4 marks)

END OF EXAMINATION