Parallel Programs and their Performance

Date: Wednesday 21st January 2015
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

Consider the following fragments of code that perform some simple numerical linear algebra computations (vector subtraction, and the calculation of the 1-norm (maximum absolute column sum) of a lower triangular matrix):

\[
a = b - c
\]

\[
p = ||L||_1 = \max_{1 \leq j \leq n} \sum_{i=j}^{n} |L_{i,j}|
\]

where \( \mathbf{a}, \mathbf{b}, \mathbf{c} \) are vectors of length \( n \) (\( n \) can be assumed to be large) and \( L \) is an \( n \times n \), lower triangular matrix (a lower triangular matrix is one in which all the elements above the diagonal are zero, \( L_{i,j} = 0, \ i < j \)).

a) The following Fortran code initialises the vectors \( \mathbf{b}, \mathbf{c} \) and implements the vector subtraction.

```fortran
!
! vector initialisation
!
DO i=1,n
   x(i) = rand()
   y(i) = rand()
END DO
!
! vector subtraction
!
DO i=1,n
   a(i) = b(i) - c(i)
END DO
!
```

Identify, without reference to any particular parallel architecture, the nature of any parallel work in the loops above.

(3 marks)

b) One implementation (implementation A) parallelises the second loop (the vector subtraction) by including the OMP pragma

```
!$omp parallel do schedule(static)
```

immediately before the second DO statement, and a second implementation (implementation B) parallelises both loops by including the same pragma before each of the DO statements. **Note that:** the specification of this pragma implies that an end parallel do pragma is automatically inserted after the following DO statement.

(Question 1 continues on the following page)
The execution time (in seconds) of these two implementations on 1 to 8 cores (one thread per core) of a 16-core AMD Opteron-based server (in each case these timings exclude the initialisation loop) is as follows.

<table>
<thead>
<tr>
<th>No. of Cores</th>
<th>Implementation A</th>
<th>Implementation B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.394</td>
<td>0.393</td>
</tr>
<tr>
<td>2</td>
<td>0.590</td>
<td>0.196</td>
</tr>
<tr>
<td>3</td>
<td>0.391</td>
<td>0.129</td>
</tr>
<tr>
<td>4</td>
<td>0.355</td>
<td>0.0966</td>
</tr>
<tr>
<td>6</td>
<td>0.258</td>
<td>0.0669</td>
</tr>
<tr>
<td>8</td>
<td>0.241</td>
<td>0.0527</td>
</tr>
</tbody>
</table>

Explain these results in terms of parallel overheads.  
(7 marks)

c) The following Fortran code fragment calculates the 1-norm of a lower triangular matrix.

```
! 1-norm calculation
!
Norm = 0.0
DO j = 1,n
  Column_sum = 0.0
  DO i = j,n
    Column_sum = Column_sum + abs(L(i,j))
  END DO
  IF (Column_sum .GT. Norm) Norm = Column_sum
END DO
!
```

Identify the nature (and limitations) of any parallel work in the above calculation as it is written. Suggest a parallel implementation of the above calculation – clearly identify all the potential overheads and include consideration of the initialisation of the array L.  
(10 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.

(5 marks)

The following OpenMP/Fortran-like pseudocode (for emphasis, the OpenMP directives are on the left and the Fortran code on the right) 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
      END IF
      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.

(2 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. Give 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 P-fold parallel implementation of this algorithm. 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)
Question 3

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) Consider the following Fortran code fragment.

```fortran
! sequential loop
! DO i=1,nstep
... ...
... !$omp parallel do
   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`.

Compare the different scheduling options available for the OpenMP DO statement and identify, with reasons, the one that would be most appropriate for the above code fragment. You can assume that the target parallel computer architecture is similar to that of the multicore Opteron-based computer mcore48, and that there are a substantial number of sequential iterations (\( n_{step} \) is large); you should not consider any restructuring of the code.

(10 marks)
Question 4

a) Consider the first order linear recurrence

\[ x_i = d_i, \]
\[ x_i = a_i x_{i-1} + d_i, \quad i = 2, 3, \ldots, n. \]

This is used to compute \( x_n \) given a starting value \( d_1 \) for \( x_1 \), but the resulting computation is entirely sequential (all other \( a_i \) and \( d_i \) values are constant). Show that, by iterating the above recurrence twice, one can obtain the new recurrence

\[ x_i = d_i, \quad x_2 = a_2 x_1 + d_2, \quad x_3 = a_3 x_2 + d_3, \quad x_4 = a_4 x_3 + d_4, \]
\[ x_i = \hat{a}_i x_{i-4} + \hat{d}_i, \quad i = 5, 6, \ldots, n \]

and thereby expose 4-fold parallelism in this computation. You should clearly derive expressions for \( \hat{a}_i \) and \( \hat{d}_i \).

(6 marks)

b) Consider now the tridiagonal system

\[ Ax = y, \quad (4.1) \]

where \( A \) is the (symmetric) tridiagonal matrix

\[
A = \begin{pmatrix}
    b_1 & a_2 \\
    a_2 & b_2 & a_3 \\
    & a_3 & b_3 & \ddots \\
    & & \ddots & \ddots & a_n \\
    & & & a_n & b_n
\end{pmatrix}.
\]

i) A cyclic reduction algorithm results from the following: using equations \( i - 1, i + 1 \) of (4.1) to eliminate \( x_{i-1}, x_{i+1} \), respectively, from the \( i \)th equation of (4.1) show that the tridiagonal system (4.1) can be replaced by

\[ A^{(i)} x = y^{(i)}, \quad (4.2) \]

where
(Question 4 continues from the previous page)

\[
A^{(1)} = \begin{pmatrix}
    b_1^{(1)} & 0 & a_3^{(1)} \\
    0 & b_2^{(1)} & 0 & a_4^{(1)} \\
    a_3^{(1)} & 0 & b_3^{(1)} & \ddots & a_n^{(1)} \\
    \vdots & \vdots & \ddots & \ddots & \vdots \\
    a_{n-1}^{(1)} & 0 & \ddots & \ddots & b_n^{(1)} \\
    a_n^{(1)} & 0 & \ddots & \ddots & 0
\end{pmatrix},
\]

and obtain expressions for the elements of \(A^{(1)}\) and \(y^{(1)}\). (4 marks)

ii) In a similar way, show that equations \(i - 2, i + 2\) of (4.2) can be used to eliminate \(x_{i-2}\), \(x_{i+2}\) respectively, from the \(i\)th equation of (4.2) to obtain

\[
A^{(2)} x = y^{(2)},
\]

where the elements of \(A^{(2)}\) and \(y^{(2)}\) are suitably defined. (3 marks)

iii) Indicate how this procedure may be continued and show that \(N = \log_2 n\) stages will be required to reduce the system of equations to diagonal form. (4 marks)

iv) The traditional way to reduce a tridiagonal system of equations to diagonal form is an \(O(n)\) algorithm which is inherently sequential. Explain why the above cyclic reduction algorithm is attractive for a parallel implementation even though it is computationally more expensive \(O(n \log_2 n)\) flops. (3 marks)

END OF EXAMINATION