Three hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Parallel Programs and their Performance

Date: Wednesday 16th January 2013
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.

[PTO]
**Question 1**

Consider the following fragments of code that perform some simple numerical linear algebra computations (vector axpy operation (linked vector addition and vector scaling), and the multiplication of two lower triangular matrices):

\[
a = \alpha x + y \\
B = L \times M,
\]

where \( \alpha \) is a scalar, \( a, x, y \) are vectors of length \( n \) (\( n \) can be assumed to be large) and \( B, L, M \) are \( n \times n \), lower triangular, matrices. (A lower triangular matrix is one in which all the elements above the diagonal are zero, \( L_{ij} = 0, \ i < j \)).

a) The following FORTRAN code initialises the vectors \( x, y \) and implements the vector axpy operation.

```fortran
C
C vector initialisation
C
DO i=1,n
   x(i) = rand()
   y(i) = rand()
END DO
C
C vector axpy
C
DO i=1,n
   a(i) = alpha*x(i) + y(i)
END DO
C
```

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 axpy operation) by including the OMP pragma

```fortran
!$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.

The execution time (in seconds) of these two implementations on 1 – 8 cores of chronos (a 16-core AMD Opteron-based server) (these timings exclude the initialisation loop in each case) 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>0.4594</td>
<td>0.4597</td>
</tr>
<tr>
<td>2</td>
<td>0.5907</td>
<td>0.2308</td>
</tr>
<tr>
<td>3</td>
<td>0.4054</td>
<td>0.1554</td>
</tr>
<tr>
<td>4</td>
<td>0.3638</td>
<td>0.1182</td>
</tr>
<tr>
<td>6</td>
<td>0.2724</td>
<td>0.08048</td>
</tr>
<tr>
<td>8</td>
<td>0.2451</td>
<td>0.06411</td>
</tr>
</tbody>
</table>

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

c) The following FORTRAN code fragment calculates the lower triangular matrix product:

```
C
C Matrix multiplication
C
DO j = 1,n
    DO i = j,n
        B(i,j) = 0.0
        DO k = j,i
            B(i,j) = B(i,j) + L(i,k)*M(k,j)
        END DO
    END DO
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 suitable for a quad quad core platform such as chronos – clearly identify all the potential overheads and include consideration of the initialisation of the arrays L and M. (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

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 ELSE
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 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 \textsc{Result} increases from being relatively short to being relatively long, compared with the rest of the necessary work.

\hspace*{2cm} (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 earlier?

\hspace*{2cm} (4 marks)

e) Discuss the difficulties that would need to be addressed if P stacks (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 earlier?

\hspace*{2cm} (4 marks)
Question 3

a) Consider the first order linear recurrence

\[ x_i = d_1, \]
\[ x_i = a_i x_{i-1} + d_i, i = 2,3,\ldots,NMAX. \]

Show that, by iterating the recurrence twice, one can obtain the recurrence

\[ x_i = d_1, x_2 = a_2 x_1 + d_2, x_3 = a_3 x_2 + d_3, x_4 = a_4 x_3 + d_4, \]
\[ x_i = \hat{a}_i x_{i-4} + \hat{d}_i, i = 5,6,\ldots,NMAX. \]

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

$$A^{(1)}x = y^{(1)}, \quad (4.2)$$

where

$$A^{(1)} = \begin{pmatrix}
  b_1^{(1)} & 0 & a_3^{(1)} \\
  0 & b_2^{(1)} & 0 & a_4^{(1)} \\
  a_3^{(1)} & 0 & b_3^{(1)} & \cdots & \cdots & a_n^{(1)} \\
  \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
  a_n^{(1)} & 0 & \cdots & b_{n-1}^{(1)} & 0 \\
  \end{pmatrix},$$

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) "This cyclic reduction algorithm is uncompetitive on serial computers, but has become popular for implementation on parallel computers." Explain this statement. (3 marks)
Question 4

A stellar system is to be modelled using a 3-dimensional, N-body, iterative time-stepping simulation of the effects of gravitational attraction (ignoring collisions). The gravitational force \( F_i \) acting on star \( s_i \) due to star \( s_j \) \((i \neq j)\) is given by:

\[
F_i = G \frac{m_i \cdot m_j}{r_{ij}^2},
\]

where \( G \) is a constant, \( m_i \) is the mass of star \( s_i \), and \( r_{ij} \) is the distance between \( s_i \) and \( s_j \). Also, the acceleration \( a_i \) of star \( s_i \), under force \( F_i \) is:

\[
a_i = \frac{F_i}{m_i}.
\]

The overall nature of the simulation is described in the following pseudo-code in which the type \texttt{TRIPLE REAL ARRAY} is an array of records of three \texttt{REAL} values. In array \texttt{positions}, the three values represent the \( x, y, z \) coordinates of the corresponding star during the current time-step. Similarly, array \texttt{velocities} represents the current \( u, v, w \) velocities of the corresponding star, in the \( x, y, z \) directions, respectively, and array \texttt{forces} represents the \( F_x, F_y, F_z \) force components currently acting on the corresponding star, in the \( x, y, z \) directions, respectively. Subroutine parameters that are updated as a result of a call are underlined in the pseudo-code below; otherwise subroutine parameters are read-only.

```
PROGRAM gravitational_N-body_calculation
REAL ARRAY masses (1:10000)
TRIPLE REAL ARRAY positions (1:10000)
TRIPLE REAL ARRAY velocities (1:10000)
TRIPLE REAL ARRAY forces (1:10000)
INTEGER step
REAL t, delta_t
!
t=0.0
READ(delta_t)
!
! delta_t, the time step size, is a program input
!
CALL initialise (masses,positions,velocities)
!
! initialise places all 10000 stars in initial positions 
! and gives them velocities and masses chosen at random 
! from appropriate distributions
!
! repeat time stepping loop 1000000 times
!
FOR step=1 TO 1000000 DO
  CALL calculate_forces (masses,positions,forces)
  !
  ! calculate_forces determines the forces on each star on
  ! the basis of the current positions of the stars

```

Page 8 of 10
CALL move_stars (masses,forces,positions,velocities)
!
! move_stars calculates new positions and velocities, at
! time t+delta_t, for each of the stars, under the
! calculated forces
!
t=t+delta_t
!
! update the time and repeat
!
END DO
END PROGRAM

Pseudo-code for a straightforward implementation of the subroutine calculate_forces is given below:

SUBROUTINE calculate_forces(m,p,f)
REAL ARRAY m(1:10000)
TRIPLE REAL ARRAY p(1:10000)
TRIPLE REAL ARRAY f(1:10000)
INTEGER i, j
FOR i=1 to 10000 DO
   Zero_The_3_Components_Of_f(i)
   FOR j=1 TO 10000 DO
      IF j.NE.i THEN
         Calculate_The_3_Force_Components_At_Star_s(i)_Due_To_Star_s(j)
         Add_The_Calculated_Components_Into_f(i)
      END IF
   END DO
END DO
END SUBROUTINE

a) Give pseudo-code for the subroutines initialise and move_stars. (6 marks)

b) Comment on the nature of potential parallel executions of the three main subroutines, initialise, calculate_forces and move_stars, stating any assumptions you make. Hence, suggest a general strategy for parallelising the whole program. (6 marks)
c) The given subroutine for `calculate_forces` is inefficient. Since gravity is a symmetrical force, the `forces(i)` components, acting on star \( s_i \), that are due to star \( s_j \) are equal in value, but opposite in sense, to the `forces(j)` components, acting on star \( s_j \), that are due to star \( s_i \). Hence, these values, which are computed twice during each cycle in the given code, could, in principle, be calculated just once per cycle.

Give alternative pseudo-code for the subroutine `calculate_forces` that implements this optimisation.

(4 marks)

d) Comment on the nature of potential parallel executions of your code for part c), and explain how you would attempt to organise any actual parallel execution so as to achieve high performance.

(4 marks)