Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

M.Sc. in Advanced Computer Science
MSc in Mathematics & Computational Science

High Performance Computing in Science & Engineering

Date: Friday 22\textsuperscript{nd} May 2009
Time: 14:00 – 16:00

Please answer any TWO Questions from the Four questions provided

This is an OPEN book examination

The use of electronic calculators is permitted.
1. 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: $a_i = F_i / m_i$.

The overall nature of the simulation is described in the following **pseudo-code** in which the type **TRIPLE REAL ARRAY** is an array of records of three **REAL** values. In array **positions**, the three values represent the $x, y, z$ coordinates of the corresponding star during the current time-step. Similarly, array **velocities** represents the current $u, v, w$ velocities of the corresponding star, in the $x, y, z$ directions, respectively, and array **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 code below; otherwise subroutine parameters are read-only.

```plaintext
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
        !
        CALL move_stars (masses, forces, positions, velocities)
       !
    ENDFOR

(Question 1 continues on the following page)
(Question 1 continues from the previous page)

Pseudo-code for a straightforward implementation of the subroutine \texttt{calculate\_forces} is given below:

\begin{verbatim}
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
\end{verbatim}

a) \textbf{Give pseudo-code} for the subroutines \texttt{initialise} and \texttt{move\_stars}. (6 marks)

b) \textbf{Comment on} the nature of potential parallel executions of the three main subroutines, \texttt{initialise, calculate\_forces} and \texttt{move\_stars}, stating any assumptions you make. Hence, suggest a general strategy for parallelising the whole program, \texttt{gravitational\_N\_body\_calculation}. (5 marks)

c) The given subroutine for \texttt{calculate\_forces} is inefficient. Since gravity is a symmetrical force, the \texttt{forces(i)} components, acting on star \texttt{s_i}, that are due to star \texttt{s_j} are equal in value, but opposite in sense, to the \texttt{forces(j)} components, acting on star \texttt{s_j}, that are due to star \texttt{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.

\textbf{Give alternative pseudo-code} for the subroutine \texttt{calculate\_forces} that implements this optimisation. (5 marks)
d) **Comment on** the nature of potential parallel executions of your code for part c), and explain how you would attempt to organise the actual parallel execution so as to achieve high performance. (4 marks)

2. Indirection of array accesses, via an intermediate index array, is a technique commonly encountered in scientific simulation codes. For example, in the following sequential loop nest (a fragment from a typical FORTRAN 77 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 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 \( j(n) \) 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 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 code fragment that might be performed in parallel. Remember that \( n \) is large (hundreds of thousands). (4 marks)

b) The following is one possible parallel implementation of the above code fragment, using a data-sharing programming model (OpenMP directives which cause the compiler to plant calls to the pthreads library – the ATOMIC directive ensures that each element \( a(j(ix)) \) is updated by only one thread at a time):

```c
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
END
```

Describe, for an SGI O3400-like hardware architecture, the behaviour of this implementation, and discuss its potential performance in terms of overheads incurred due to synchronisation, load imbalance and remote accesses. Clearly state any assumptions you make about the contents of the array \( j(n) \). (8 marks)

c) Suggest a more efficient implementation for the given code fragment. Explain how your revised implementation reduces the overheads identified in your answer to part (b). (8 marks)
3. a) Consider the computation represented by the following fragment of FORTRAN code:

```fortran
integer i, j, n
real error, eps
real a(n,n), f(n,n)
C iteration loop
while (abs(error) .gt. tol) do
  do j = 2, n-1
    do i = 2, n-1
      a(i,j) = 0.25*(a(i,j-1)+a(i-1,j)+a(i+1,j)+a(i,j+1))
      & + a(i,j) + f(i,j)
    enddo
  enddo
  error = . . .
end while
```

For a $p^2$–processor message-passing implementation of this algorithm, 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 rectangular blocks of dimensions $q \times r$, where $q \times r = n^2/p^2$, and $q, r \leq n/2$.

Derive the volume of communication per processor implied by each data distribution, and show that, for $p \geq 2$, the volume of communication is minimised when we choose to partition $a$ and $f$ into square blocks of dimensions $n/p \times n/p$. (10 marks)

b) Now, consider the following fragment of FORTRAN code:

```fortran
k = 1000
do i = 2, 999
  do j = 1, 1000
    k = k+1
    if (i .ge. j) then
      c(i,j) = 0.0
      do m = j, j+1
        c(i,j) = c(i,j) + a(m)*b(k+1000*(m-j))
      enddo
    endif
  enddo
enddo
```

(Question 3 continues on the following page)
(Question 3 continues from the previous page)

i) Identify the reasons why the outer, i, loop is not parallelisable and transform the code into a semantically equivalent form in which the outer loop is parallelisable. (3 marks)

ii) How would you map the transformed parallel code onto the processors so as to reduce load imbalance? (3 marks)

iii) Apply any other transformations you consider appropriate for an efficient implementation of the code on a parallel computer such as an SGI O3400. (4 marks)
4. a) Consider the matrix product $C = A \ast B$, where $A$, $B$, and $C$ are $N \times N$ matrices. If we divide each of the matrices $A$, $B$, and $C$ into $2 \times 2$ block matrices as follows:

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

where the blocks are $N/2 \times N/2$ matrices, and define

$$
P_1 = (A_{11} + A_{22})(B_{11} + B_{22}),
$$

$$
P_2 = (A_{21} + A_{22})B_{11},
$$

$$
P_3 = A_{11}(B_{12} - B_{22}),
$$

$$
P_4 = A_{22}(B_{21} - B_{11}),
$$

$$
P_5 = (A_{11} + A_{12})B_{22},
$$

$$
P_6 = (A_{21} - A_{11})(B_{11} + B_{12}),
$$

$$
P_7 = (A_{12} - A_{22})(B_{21} + B_{22})
$$

verify that Strassen’s algorithm

$$
C_{11} = P_1 + P_4 - P_5 + P_7,
$$

$$
C_{12} = P_3 + P_5,
$$

$$
C_{21} = P_2 + P_4,
$$

$$
C_{22} = P_1 + P_3 - P_2 + P_6
$$

gives the same results as conventional matrix multiplication.

[Hint: verify that $P_1 + P_4 - P_5 + P_7 = A_{11}B_{11} + A_{12}B_{21}$, etc.] (4 marks)

b) Given that conventional matrix multiplication of two $N \times N$ matrices requires $2N^3 - N^2$ flops (floating point operations) and that the addition or subtraction of two $N \times N$ matrices requires $N^2$ flops, show that the number of flops required to multiply two $N \times N$ matrices by Strassen’s algorithm is

$$
\frac{7}{4}N^3 + \frac{11}{4}N^2.
$$

(5 marks)
(Question 4 continues from the previous page)

c) Assuming that the matrix dimension satisfies $N = 2^n$, for some integer $n$, describe a recursive version of Strassen’s algorithm and, given the complexity result of part (b), show that it is optimal to recur Strassen’s algorithm until the submatrices (the blocks) have dimensions $8 \times 8$.

[Hint: show that Strassen’s algorithm outperforms conventional matrix multiplication for matrices of dimensions $16 \times 16$, but not for matrices of dimensions $8 \times 8$.]  

(7 marks)

d) Describe briefly how you would implement the recursive version of Strassen’s algorithm on a machine such as an SGI O3400, and indicate any potential difficulties with such an implementation.

(4 marks)