COMP60611

Three hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Parallel Programs and their Performance

Date: Friday 29th January 2016

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):

$$a = \alpha x + y$$
$$B = L * M,$$

where α 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.

```
!
! vector initialisation
!
DO i=1,n
   x(i) = rand()
   y(i) = rand()
END DO
!
! vector axpy
!
DO i=1,n
   a(i) = alpha*x(i) + y(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 axpy operation) 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 immediately before **each** of the DO statements.

The execution time (in seconds) of these two implementations, including the initialisation loop in each case, on 1 – 8 cores of mcore48 (a 48-core AMD Opteron-based server) with 'scattered' affinity is as follows:

No of Cores	Implementation A	Implementation B
1	0.3020	0.3040
2	0.4280	0.1560
3	0.3373	0.1080
4	0.2925	0.0850
6	0.2487	0.0640
8	0.2278	0.0555

Explain these results in terms of parallel overheads. State any assumptions you make.

(7 marks)

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

```
!
! lower triangular matrix multiplication
!
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 12-core platform such as mcore48 – clearly identify all the potential overheads and include consideration of the initialisation of the arrays `L` and `M`.

(10 marks)

Question 2

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)`:

```
!
! pertinent variable declarations
!
INTEGER k, n, ix
PARAMETER (n=1000000)
INTEGER j(n)
REAL a(n), b(n), c(n)
       :
       :
       :
!
! loop nest code fragment starts here
!
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`, of size `n`, effectively holds pointers to the elements of the real array `a`, 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` are updated according to the values held in `j`.

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)

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):

```
!
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 a NUMA architecture like that of mcore48, using 'scattered' affinity and no more than 24 cores, the behaviour of this implementation, and discuss its potential performance in terms of overheads incurred due to scheduling, synchronisation, load imbalance and remote memory accesses. State clearly any assumptions you make about the contents of the array `j`.

(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 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 8, or more, cores, the data associated with the code fits into cache memory, as opposed to runs on fewer than 8 cores, when data must be repeatedly fetched from memory.
- At the highest numbers of cores shown on the graph, software parallelism has been exhausted, and the execution time remains constant as the number of cores increases.

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 how speedup is affected by Amdahl's Law as the number of cores increases.

(6 marks)

d) Explain why Amdahl's Law cannot adequately account for all of the behaviour illustrated in your speedup graph for part b). You will need to identify clearly those code features that are not explained by Amdahl's Law.

(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 $\mathbf{F_i}$ acting on star $\mathbf{s_i}$ due to star $\mathbf{s_j}$ ($\mathbf{i \neq j}$) is given by: $\mathbf{F_i} = G\ \mathbf{m_i} \times \mathbf{m_j} / \mathbf{r_{ij}}^2$, where G is a constant, $\mathbf{m_i}$ is the mass of star $\mathbf{s_i}$, and $\mathbf{r_{ij}}$ is the distance between $\mathbf{s_i}$ and $\mathbf{s_j}$. Also, the acceleration $\mathbf{a_i}$ of star $\mathbf{s_i}$, under force $\mathbf{F_i}$ is: $\mathbf{a_i} = \mathbf{F_i} / \mathbf{m_i}$.

The overall nature of the simulation is described in the following pseudo-code in which the data 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 pseudo-code below; otherwise subroutine parameters are read-only.

```
PROGRAM gravitational_N-body_calculation
PARAMETER n=1000000
REAL ARRAY masses(n)
TRIPLE REAL ARRAY positions(n)
TRIPLE REAL ARRAY velocities(n)
TRIPLE REAL ARRAY forces(n)
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 million stars in initial positions
! and gives them velocities and masses chosen at random
! from appropriate distributions
!
! repeat time stepping loop a billion times
!
FOR step=1 TO 1000000000 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
    !
```

```
    ! 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(n)
TRIPLE REAL ARRAY p(n)
TRIPLE REAL ARRAY f(n)
INTEGER i, j
FOR i=1 to n DO
    Zero_The_3_Components_Of_f(i)
    FOR j=1 TO n 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 subroutine **move_stars.**

(3 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 $\mathbf{s_i}$ that are due to star $\mathbf{s_j}$ are equal in value, but opposite in sense, to the **`forces(j)`** components acting on star $\mathbf{s_j}$ that are due to star $\mathbf{s_i}$. 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)

e) The reorganised loop is expected to be faster than the original loop, but it is still $O(n^2)$ where n is the number of stars. Outline an algorithmic strategy for reducing this complexity to $O(n \log n)$.

(3 marks)

END OF EXAMINATION