COMP 60611

Three hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Parallel Programs and their Performance

Date: Monday 22nd January 2018

Time: 14:00 - 17:00

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 subtraction, and the calculation of the 1-norm (maximum absolute column sum) of a lower triangular matrix):

$$\underline{a} = \underline{b} - \underline{c}$$

$$p = \|L\|_1 = \max_{1 \le j \le n} \sum_{i=j}^{n} |L_{i,j}|$$

where $\underline{a}, \underline{b}, \underline{c}$ 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 C-style code initialises the vectors b, c using a random number generator function and implements the vector subtraction.

```
//
// vector initialisation
//
for (int i=0;i<n;i++){
   b[i] = rand();
   c[i] = rand();
}
//
// vector subtraction
//
for (int i=0;i<n;i++){
   a[i] = b[i] - c[i];
}
```

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

```
#pragma omp parallel for schedule(static)
```

immediately before the second "for" statement, and a second implementation (implementation B) parallelises both loops by including the **same** pragma before **each** of the "for" statements.

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. The execution time of the sequential version of the code was 0.549 seconds.

No. of Cores	**Implementation A**	**Implementation B**
1	0.552	0.550
2	0.826	0.274
3	0.547	0.181
4	0.497	0.135
6	0.361	0.0936
8	0.337	0.0738

Quantify the total overheads in each implementation and give an explanation of these results in terms of parallel overheads.

(7 marks)

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

```
//
// 1-norm calculation
//
norm = 0.0;
for (j=0;j<n;j++) {
   column_sum = 0.0;
      for (i=j;i<n;i++) {
         column_sum += abs(L[i][j]);
      }
   if (column_sum > norm) norm = column_sum;
}
```

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) In OpenMP, the "omp for" work-sharing directive allows the programmer to specify one of the following four distinct kinds of schedule:

simple static (block scheduling)
interleaved (block-cyclic scheduling)
simple dynamic (chunk self-scheduling)
guided dynamic (guided self-scheduling)

For the two C-like pseudocode fragments in parts (i) and (ii) of this question, which are taken from the same application, identify the chief source(s) of parallel overhead you would expect to see, and by analysing the behaviour of each of the available kinds of schedule suggest a suitable choice for the most appropriate kind of schedule, plus chunk size, if appropriate, for the "omp for" directive, giving reasons for your choice. Assume that the target architecture is a quad 12-core Opteron system, like mcore48, and **do not** consider any restructuring of the code.

(i)
```
#pragma omp for
for (j=0;j<n;j++) {
   for (i=0;i<n;i++) {
      if (A[i][j] < 0.0) makepositive(A[i][j]);
   }
}
```

In this code fragment, `makepos` is a function that computes a suitable non-zero value for the element of `A[i][j]`, which is of size 4000 by 4000. This computation may take a significant and variable amount of time. The computation is independent of the contents of the array `A`, however, and can be assumed to disturb the contents of caches minimally. The can be expected to be a significant number of non-zero value in the array but nothing is known about their distribution. The array `A` can be assumed to have been generated in a "pleasingly" parallel loop nest which has resulting in the array being distributed by rows over the *p* cores used in the computation.

(6 marks)

(ii)
```
#pragma omp for
for (i=0;i<n;i++) {
   for (j=i;j<n;j++) {
      sum[i] += A[i][j];
   }
}
```

This code fragment computes a sum of the elements of each row in the upper triangular part of the (full) matrix `A`. The distribution of the matrix at the start of this loop nest should be assumed to have resulted from the use of your scheduling choice for the code in part (i) of this question.

(4 marks)

b) Suggest any changes and/or restructuring or combination of the code in the two loop nests in part a) (i) and (ii) of this question that would be expected to improve performance, including reviewing the kind of schedule to use for your resulting code. Explain your expected performance improvements in terms of their expected impact on the parallel overheads involved.

(10 marks)

Question 3.

a) Consider the computation represented by the following C-like pseudocode:

```
int n;
float error, eps;
float a[n][n], f[n][n];

// Iteration loop
while ( abs(error) > tol) {
   for (int i=1; i<n-1; i++)
      for (int j=1; j<n-1; j++)
         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];

error = . . .
}
```

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 dimension $q \times r$, where $q \times r = n^2/p^2$, and $q, r \leq {}^{n}/_{2}$

For the worst case communication case, where each processor has to communicate with all four of its neighbours, derive the volume of communication per processor implied by each distribution, and show that, for $p \geq 2$, the volume of communication is minimised when we choose to partition the arrays `a` and `f` into square blocks of dimensions $n/p \times n/p$.

[9 marks]

b) Consider the following issues arising from data dependencies on shared variables when parallelising codes:

i) Explain the term *race condition* in the context of two thread incrementing a shared counter, `count`. Give a data dependence graph for this example.

[3 marks]

ii) Give C-like pseudocode, using OpenMP-like parallel features, for a solution to the race condition example in part (i).

[1 marks]

iii) By drawing data dependence graphs, or otherwise, identify whether the following two fragments of C code can be executed safely in parallel (i.e. will give the same results as a sequential execution). You may assume all arrays are large enough so that all accesses implied by the loop bounds are legal:

```
1) for (int i=1;i<=n;i++) {
     a[2*i] = i;
     b[i] = a[6*i-1];
   }
```

[2 marks]

```
2) for (int i=1;i<=n;i++) {
     a[i] = a[i+2]+3;
   }
```

[2 marks]

iv) By focusing particularly the accesses to the array `a[]` in the following loop, suggest a loop *transformation* which would enable the computation described to be executed safely in parallel:

```
for (int i=1;i<=n;i++) {
   a[i] = b[i];
   c[i] = a[i+1];
   d[i] = c[i]+1;
}
```

[3 marks]

Question 4.

Consider the following heterogeneous multicore processor, similar to an ARM big.LITTLE design:

The processor consists of a number of "big" cores which have high processing rates (and hence are "power-hungry") and a number of "little" cores which have lower processing rates but are more energy-efficient.

An application may execute on either type of core and may use both types of core in a single parallel execution, for example, P_b big cores and P_l little cores.

The big cores are β times faster than the little cores; that is, a single threaded application executing on a little core will take β times as long to execute than on a big core ($T_s^l = \beta T_s^b$**).**

a) A "pleasingly parallel" application is to be executed on this heterogeneous parallel machine using the same number of big and little cores (for example, $P_b = P_l = 4$, so 8 cores in total). For a load balanced solution, this will require a fraction, γ ($0 \leq \gamma \leq 1$) of the iteration space of the application (i.e. a fraction of the total computational work) to be executed on the big cores, and the rest to be executed on the little cores. For the fraction of work running on the big cores, the naïve ideal execution time on P_b cores is given by:

$$T_{P_b} = \frac{\gamma T_s^b}{P_b}$$

i) Give an expression for T_{P_l}, the naïve ideal execution time for the fraction of work executing on the little cores, in terms of T_s^b (*not* in terms of T_s^l), P_l, γ and β.

[3 marks]

ii) Hence, or otherwise, show that for a load balanced solution the partitioning fraction, γ, is given by:

$$\gamma = \frac{\beta}{1+\beta}$$

[4 marks]

iii) Explain why your equation for γ in part (ii) of this question leads to a load balanced solution, independent of the number of big and little cores used, illustrating your answer for a processor in which big cores are twice as fast as little cores ($\beta = 2$), and for one in which big cores are six times as fast as little cores ($\beta = 6$).

[4 marks]

b) As an alternative to using the same number of big and little cores and partitioning the iteration space to achieve load balance, consider the case where the work is proposed to be partitioned equally to big and little cores (i.e. where $(\gamma = 1/2)$). Now, for a load balanced solution, a different number of big and little cores will be required.

i) Starting with the expressions for T_{P_b}, given in part (a) of this question, and T_{P_l}, which you stated in part (a(i)), derive an expression, in terms of β, for the ratio of the number of big cores to little cores required for load balance in this case (where $(\gamma = 1/2)$).

[3 marks]

ii) Explain the implications of this ratio for the number of little cores required relative to the number of big cores used for a load balanced solution in the case where γ=1/2 as β increases. What are the implications in the case where γ=1/4 (i.e. when 1/4 of the iteration space is to be executed on the big cores)?

[4 marks]

iii) Comment on the implications of applying this ratio in practice on a real big.LITTLE processor if β is not a whole number.

[2 marks]

END OF EXAMINATION