Examination Parallel Computing

2012-06-26

- The use of electronic or printed material is not allowed (GEEN OPEN BOEK).
- Give sufficient explanations to you answers.
- In some of the problems you may need to use a specific function (routine) from the corresponding application program interface. You are not expected to remember the exact name and syntax of that function. If you do not remember these details, you may give the function an appropriate name and syntax and explain what it is supposed to do.

1. Speed-up

The execution time $t(1)$ of a program on one processor ($p = 1$) consists of a part $t_{par}(1)$ spent on computations that can be executed in parallel and another part $t_{seq}(1)$ spent on computations that are inherently sequential and cannot be parallelized. For a given program, a speed-up $S(p)$ of 15 is achieved with $p = 16$ processors, $S(16) = 15$.

(a) Give a definition of the speed-up $S(p)$ with which a program can be executed on a parallel computer with p processors.
(b) Derive a formula for the speed-up $S(p)$ as a function of p and the ratio $x = t_{par}(1)/t_{seq}(1)$.
(c) What is the value of the ratio x for the considered program?
(d) What is the maximum value of the speed-up which can be achieved for that program?
(e) On how many processors p should one execute that program in order to achieve half of the maximum possible speed-up?
(f) A certain program takes 29.5 s to execute on one processor and 7.6 s to execute on 4 processors. After compiler optimization, the same program takes 4.8 s on one processor and 1.7 s on 4 processors. What are the values of the speed-up before and after optimization? What is your explanation of the difference between the two speed-up values?
(g) What is meant by the term 'super-linear' speed-up? Are you aware of situations in which super-linear speed-up can be observed and what is the reason for it?

2. Parallel bubble sort and Pthreads

Describe and explain the bubble sort algorithm for sorting a sequence of n numbers. Design a pipeline of functions that is a parallel implementation of this algorithm and implement this pipeline using Pthreads.

3. Cache memory

Consider the following three sections of code, to be referred to as ikj, jki and jik, respectively:

```
/* ikj */
for (i=0; i<n; i++) {
  for (k=0; k<n; k++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}

/* jki */
for (j=0; j<n; j++) {
  for (k=0; k<n; k++) {
    r = b[k][j];
    for (i=0; i<n; i++)
      c[i][j] += a[i][k] * r;
  }
}

/* jik */
for (j=0; j<n; j++) {
  for (i=0; i<n; i++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum
  }
}
```

Assume that the arrays are mapped to memory in such a way that the elements of one row follow each other and have consequtive memory addresses. For instance, $a[i][j]$ may have address $s + n * i + j$ in the memory, where s is some start offset (this is C-style mapping). Assume the above codes are executed on a computer system in which each core has a cache of m cache lines, each cache line having $l = 16$ machine words . Assume that $n^2 \gg l * m$. Assume that 1 clock cycle per machine word is needed to access data that is found in the cache and that $64 + l$ clock cycles are needed to transfer a cache line from the main memory to the cache.

a) For each of the three codes and each of the involved arrays a, b and c, estimate the frequency of cache misses.

b) How many clock cycles does it take to execute each of the above codes? (You may neglect the clock cycles needed to execute the atithmetic operations by the ALU of a processor and count only the clock cycles needed to fetch operands and store results.) Which of the above three codes will run fastest? For the other two codes, how much slower are they than the fastest one?

4. OpenMP

a) Consider the following program:

```
void main()
{
    int ID = 0;
    printf("Hello world from thread %d !\n", ID);
}
```

Modify this program, adding to it OpenMP statements and constructs, such that (a default number of) multiple threads will be created and each thread will print "Hello world from thread i !" where i is the number of that thread.

b) Consider the following section of code:

```
for (k=0; k<n-1; ++k)
{
  for (i=k+1; i<n; ++i)
  {
    A[i][k] /= A[k][k];
    for (j=k+1; j<n; ++j)
    {
      A[i][j] -= A[i][k]*A[k][j];
    }
  }
}
```

Add to this code appropriate OpenMP work sharing directives or constructs, such that the work is shared by multiple threads. Mind the data dependencies!

c) Consider the following section of code:

```
for(i=0;i<N;i++) {d[i]=a[i]+b[i];}
for(i=0;i<N;i++) {z[i]=x[i]*y[i];}
for(i=0;i<N;i++) {c[i]=d[i]+z[i];}
```

Add to this code appropriate OpenMP work sharing directives or constructs, such that each thread executes a different loop. Make sure that there are no race conditions.

5. MPI

Consider the following section of code: