Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2021-22 ◇ 28/1/22 ◇ Block OpenMP ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.2 points)
Given the following function:

```
int genera_mat(double x[], double y[], double A[][N]) {
   int i, j, count=0;
   double ci=x[0]*y[0], cs=ci, c;
   for (i=0; i<N; i++) {
      for (j=0; j<N; j++){
         A[i][j]=x[i]*y[j];
         if (A[i][j]>cs) cs=A[i][j];
         if (A[i][j]<ci) ci=A[i][j];
      }
   }
   c=(ci+cs)/2.0;
   for (i=0; i<N; i++)
      for (j=0; j<N; j++)
         if (A[i][j]>c) ++count;
   return count;
}
```

0.8 p. (a) Parallelize it by means of OpenMP, using a single parallel region.

Solution:

```
double ci=x[0]*y[0], cs=ci, c;
#pragma omp parallel
{
    #pragma omp for private(j) reduction(max:cs) reduction(min:ci)
    for(i=0; i<N; i++) {
       ...
    }
    c=(ci+cs)/2.0;
    #pragma omp for private(j) reduction(+:count)
    for(i = 0; i < N; i++)
       ...
}
return count;
```

0.2 p. (b) Compute the sequential and parallel time, taking into account that the cost of each of the comparisons `A[i][j]>cs`, `A[i][j]<ci` and `A[i][j]>c` is 1 flop. Indicate all steps in the calculation of the times.

Solution:
Sequential cost:

$$t(N) = 1 + \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} 3 + 2 + \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} 1 \approx 3N^2 + N^2 = 4N^2 \texttt{ flops}.$$

Parallel cost:

$$t(N,p) = 1 + \sum_{i=0}^{N/p-1} \sum_{j=0}^{N-1} 3 + 2 + \sum_{i=0}^{N/p-1} \sum_{j=0}^{N-1} 1 \approx \sum_{i=0}^{N/p-1} 3N + \sum_{i=0}^{N/p-1} N = \frac{4N^2}{p} \texttt{ flops.}$$

0.2 p.

(c) Modify the parallel implementation of exercise (a) so that every thread prints the number of times that `A[i][j]>c` in the block of the matrix `A` processed by it; that is, the number of contributions of each thread to the final value of `count`. For instance, if the final value of `count` was equal to 10 and we have 4 threads, a possible output would be the following:

```
Thread 1:  2
Thread 0:  3
Thread 3:  3
Thread 2:  2.
```

Solution:

```
#pragma omp parallel
{
   int hilo, countp=0;
   #pragma omp for private(j) reduction(max:cs) reduction(min:ci)
   for(i=0; i<N; i++)
      for (j=1; j<N; j++){
        .....
      }
   c=(ci+cs)/2.0;
   hilo=omp_get_thread_num();
   #pragma omp for private(j) reduction(+:count)
   for(i=0; i<N; i++)
       for(j=0; j<N; j++){
          if (A[i][j]>c) {++count;++countp;}
       }
   printf("Thread %d: %d \n",hilo,countp);
}
return count;
```

Question 2 (1.1 points)

Given the following functions:

```
void prod(double A[M][N], double B[N][N]) {
  int i,j;
  for (i=0; i<N; i++) {
    A[i][i] = (A[i][i]+B[i][i])/2.0;
    for (j=0; j<i; j++) {
      A[i][j] = A[i][j]+A[i][j]*B[i][j];
    }
  }
}

void square(double A[M][N], double B[N][N]) {
  int i,j;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
```

```
      A[i][j] = 2.0*A[i][j]+B[i][j]*B[i][j];
    }
  }
}
```

we have a program that performs the following sequence of operations:

```
prod(D,C);     /* Task 1 */
square(E,C);   /* Task 2 */
square(C,E);   /* Task 3 */
prod(F,E);     /* Task 4 */
prod(F,F);     /* Task 5 */
prod(F,C);     /* Task 6 */
```

0.2 p. (a) Compute the asymptotic sequential cost in flops of each of the two functions.

Solution:

Cost of `prod`:

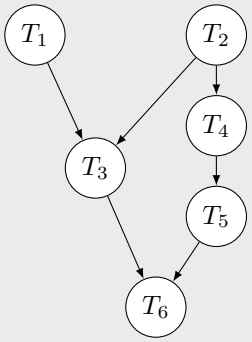
$$\sum_{i=0}^{N-1}\left(2+\sum_{j=0}^{i-1}2\right)=\sum_{i=0}^{N-1}(2+2i)\approx 2\sum_{i=0}^{N-1}i\approx N^2 \text{ flops}$$

Cost of `square`:

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}3=\sum_{i=0}^{N-1}3N=3N^2 \text{ flops}$$

0.4 p. (b) Draw the task dependency graph. Indicate which is the maximum degree of concurrency, the critical path and its length and the average degree of concurrency.

Solution:



Critical path: T2→T3→T6.

Length of the critical path: $L = 3N^2 + 3N^2 + N^2 = 7N^2$ flops

Maximum degree of concurrency: 2

$$M=\frac{N^2+3N^2+3N^2+N^2+N^2+N^2}{7N^2}=\frac{10N^2}{7N^2}=\frac{10}{7}$$

0.5 p. (c) Write a parallel version based on sections, from the dependency graph that you have drawn. The execution time must be minimized.

Solution:

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    prod(D,C);     /* Task 1 */
    #pragma omp section
    square(E,C);   /* Task 2 */
  }
  #pragma omp sections
  {
    #pragma omp section
    square(C,E);   /* Task 3 */
    #pragma omp section
    {
      prod(F,E);     /* Task 4 */
      prod(F,F);     /* Task 5 */
    }
  }
} /* End of parallel */
prod(F,C);     /* Task 6 */
```

Question 3 (1.2 points)

The following function takes a vector `V` that stores, in each of its components, the number of computes infected in the computing laboratories of UPV during one month (identified from 1 to 12) of last year by any of the `NV` currently known virus (numbered from 0 on). From these values, the function prints, for each virus, its identifier, the total number of computers infected by it along the year and the identifier of the month in which it infected the largest number of computers at UPV. Additionally, the function returns as a result the total number of virus that infected any of the computes, from the `NV` known, and fills the output vector `idVirus` with their identifiers.

```
int manage_virus(struct Tvirus V[], int n,int idVirus[]) {
  int i, virus, month, infected;
  int total_infected[NV],infected_max[NV],month_max[NV];
  int nVirus=0;

  for (i=0;i<NV;i++) {
    total_infected[i]=0;
    infected_max[i]=0;
    month_max[i]=0;
  }

  for (i=0;i<n;i++) {
    infected=V[i].infected;
    month=V[i].month;
    virus=V[i].virus;
    total_infected[virus]+=infected;
    if (infected>infected_max[virus]) {
      infected_max[virus]=infected;
      month_max[virus]=month;
    }
  }
```

```
  for (i=0;i<NV;i++) {
    if (total_infected[i]>0) {
      idVirus[nVirus]=i;
      nVirus++;
    }
  }

  for (i=0;i<NV;i++) {
    if (total_infected[i]>0)
      printf("%d %d %d\n",i,total_infected[i],month_max[i]);
  }

  return nVirus;
}
```

Parallelize the function efficiently by means of OpenMP, using a single parallel region. Do not parallelize the first and the last loops, those in charge of initializing the vectors to 0 and printing the resutsl.

Solution:

```
int manage_virus(struct Tvirus V[], int n,int idVirus[]) {
  int i, virus, month, infected;
  int total_infected[NV],infected_max[NV],month_max[NV];
  int nVirus=0;

  for (i=0;i<NV;i++) {
    total_infected[i]=0;
    infected_max[i]=0;
    month_max[i]=0;
  }
  #pragma omp parallel
  {
    #pragma omp for private(infected,month,virus)
    for (i=0;i<n;i++) {
      infected=V[i].infected;
      month=V[i].month;
      virus=V[i].virus;
      #pragma omp atomic
      total_infected[virus]+=infected;
      if (infected>infected_max[virus]) {
        #pragma omp critical
        {
          if (infected>infected_max[virus]) {
            infected_max[virus]=infected;
            month_max[virus]=month;
          }
        }
      }
    }
    #pragma omp for
    for (i=0;i<NV;i++) {
```

```
          if (total_infected[i]>0) {
            #pragma omp critical
            {
              idVirus[nVirus]=i;
              nVirus++;
            }
          }
        }
      }

      for (i=0;i<NV;i++) {
        if (total_infected[i]>0)
          printf("%d %d %d\n",i,total_infected[i],month_max[i]);
      }

      return nVirus;
    }
```