OpenMP

Exercise 1: Specify a formula for the number of iterations a thread is assigned with under a STATIC scheduling. Assume T be the number of threads, N the number of iterations.

Exercise 2: Specify a formula for the number of iterations a thread is assigned with following a (STATIC,C) scheduling. Assume T be the number of threads, N the number of iterations, and C the chunk value.

Exercise 3: Apply the outcome of the **exercise 1** and **exercise 2** to the following situation, where XXXX = STATIC or STATIC, C. Assume T be the number of threads, N the number of iterations, and C the chunk value. Which thread will execute more iterations?

```
#pragma omp parallel for schedule (XXXX)
  for (unsigned int i=12; i<234; i=i+7) {
    Do_Some_Work(i):
  }
```

Exercise 4: Given the following code, and assume the matrices `A` and `B` contain `double` values. Assume `sizeof(double)` = 8 bytes. How much data per thread is used? Assume T be the number of threads. Assume T<<N and that T divides N.

```
#pragma omp parallel for schedule(STATIC)
  for (unsigned int i=0; i<N; i++)
    for (unsigned int j=0; j<N; j++)
      for (unsigned int k=0; k<N; k++)
          A[i][j][k] = B[i][j][k] + ... ;
```

Exercise 5: The following code implements the **merge-sort** algorithm for sorting integer vectors. Propose an OpenMP task-based parallelization. Use the `final` clause.

```
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Halve the vector
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}
```

```
int main() {


    int arr[N];

    mergeSort(arr, 0, N - 1);

}
```

Exercise 6: Propose an OpenMP parallelization for the following code.

```
#define N (1024*1024)
int V[N];

int main () {
   int max;
   for (unsigned int i=0; i<N; i++) {
      if (max<V[i]) max = V[i];
   }

}
```

Exercise 7: The following code has been parallelized using loop level parallelism in OpenMP. Which scheduling should be more efficient for its execution?

```
int A[N][N];

#pragma omp parallel for scheduling(XXXX)
  for (unsigned int i=0; i<N; i++) {
    for (unsigned int j=0; j<i; j++)
      computeElement( A[i][j] );
  }
```

1- If a STATIC scheduling is applied, give a formula to model the work done by each thread. Assume T the number of threads. Assume T<<N (T is far smaller than N). Assume $T_{compute}$ the cost of one invocation to routine `computeElement`.

Tfirst(tid) = "First assigned iteration assigned to thread tid" = ...

Tlast(tid) = "Last assigned iteration assigned to thread tid" = ...

Titers(tid) = "Number of iterations assigned to thread tid" = ...

Twork(tid) = "Total work assigned to thread tid" = ...

2- Are all the threads doing the same amount of work? Applying a STATIC scheduler generates a problem of work unbalance. Propose a solution.

Exercise 8: OpenMP defines an implicit barrier synchronization at the end of each work-sharing construct. In the following code we would like to avoid the synchronization, whenever possible. Propose changes that allow to introduce the `nowait` clause to all work-sharings conetructs.

```
#pragma parallel
{

#pragma omp for scheduling(STATIC)
for (unsigned int i=0; i<N+5; i++)
   V[i] = ...

#pragma omp for scheduling(STATIC)
for (unsigned int i=0; i<N-5; i++)
   A[i] = V[i] + ....;

} // parallel
```

Exercise 9: Given the following code suppose the number of iterations in the i-loop is less than the number of threads (N<T). What kind of performance problem will this loop present? Propose a solution that solves the issue.

```
#pragma omp parallel for
  for (unsigned int i=0; i<N; i++)
    for (unsigned int j=0; j<N; j++)
       V[i][j] = ...
```

Exercise 10: Given the following code, assume Tomp the computational cost associated to the directive `omp parallel for`. Assume Titer be the cost of one i-iteration. Let T be the number of threads. In general, Tomp will increase with the number of threads, but the work done per each thread will decrease (work per thread = (N/T)*Titer). If we increment the number of threads (4, 8, 16, 32, 64 threads), at some point it can happen that the overheads associated to the parallel directive (thread creation and synchronization) exceed the work done per each thread. Propose a solution for this issue. How would you minimize the impact of the overheads related to the parallel directive?

```
while (iter<LAST_ITER) {

#pragma parallel for schedule (STATIC)
  for (unsigned int i=0; i<N; i++)
  {
      // Do computation
  }

} // while
```

Exercise 11: Given the following code, how many threads are used in each parallel region below?

```
void main() {

#pragma omp parallel
  {
    ...
    omp_set_num_threads (2);

#pragma omp parallel
    {
       ...
#pragma omp parallel num_threads(random()%4+1) if(0)
       {
        ...

       } // parallel
    } // parallel
  } // parallel

} // main
```

Exercise 12: Given the following code, what does appear on the screen if XXXXXX is `shared(x)`, `private(x)` or `firstprivate(x)`?

```
int x=1;
#pragma omp parallel XXXXXX num_threads(2)
{
  x++;
  printf("%d\n", x ) ;
}
printf("%d\n", x ) ;
```

Exercise 13: Given the following code, parallelize it with OpenMP tasks and use the OpenMP task reduction support.

```
Items Data[N][N];

int main() {

  for (unsigned int bi=0; bi<NB; bi++) {
    for (unsigned int bj=0; bj<NB; bj++) {
      float LocalMax;
      LocalMax= ComputeLocalMax (Data[bi][bj]));
      GlobalMaxs = max(GlobalMax,LocalMax);

    }
  }
}
```

Exercise 14: Make an OpenMP parallelization using loop work-sharing constructs. Notice that loops `bi` and `bj` are not parallel but some parallelism can be exploited generating a wave-front execution. Use the `flush` construct to parallelize the code.

```
for (unsigned int bi=0; bi<NB; bi++)
  for (unsigned int bj=0; bj<NB; bj++)
     AA[bi][bj] += (AA[bi-1][bj] +
                    AA[bi+1][bj] +
                    AA[bi][bj-1] +
                    AA[bi][bj+1])/4;
```

Exercise 15: Solve **exercise 14** with the usage of tasks and dependencies among them.

Exercise 16: Given the following OpenMP code sketch, determine which tasks have to be completed after the synchronization points.

```
#pragma omp task {}       // T1
#pragma omp taskgroup
{

#pragma omp task          // T2
{
    #pragma omp task {}   // T3
}

#pragma omp task {}       // T4

} // taskgroup
```

Exercise 17: Given the following OpenMP code sketch, determine which tasks have to be completed after the synchronization points.

```
#pragma omp task {}       // T1

#pragma omp task          // T2
{
    #pragma omp task {}   // T3
}

#pragma omp task {}       // T4

#pragma omp taskwait
```

Exercise 18: Parallelize the following code that computes one element in the Fibonacci sequence. Use `final` and `mergeable` clauses.

```
int fibonacci(int n) {
  int i, j;

  if (n<2) return n;
  i = fibonacci(n-1);
  j = fibonacci (n-2);
  return (i+j);
}
```

Exercise 19: Orphan directives in OpenMP are directives which appear out of the scope of a parallel region definition. Propose a OpenMP parallelization of the following code that uses orphan directives:

```
float a[N}, b[N];

float dotprod ()
{
  int i;
  for (i=0; i < N; i++)  sum = sum + (a[i]*b[i]);
}

int main (int argc, char *argv[]) {
  float sum;

  sum = dotprod();

  printf("Sum = %f\n",sum);

}
```