PAR – In-Term Exam – Course 2023/24-Q1

November 2^{nd}, 2023

Problem 1 (5.0 points) Given the following code:

```
#define BS 8
#define N  24
double A[N][N];
int ii, jj, i, j;
...

for (ii=0; ii<N; ii+=BS)
  for (jj=0; jj<N; jj+=BS) {
     tareador_start_task("Comp_ii_jj");
     for (i=max(1,ii); i<min(ii+BS,N); i++)
       for (j=max(1,jj); j<min(jj+BS,N-1); j++)
          A[i][j] = A[i][j-1] + A[i-1][j] + A[i][j+1]; // cost of this line: tc t.u.
     tareador_end_task("Comp_ii_jj");
  }
...
```

1. (1 point) Draw the Task Dependence Graph (TDG) based on the above Tareador task definitions and for `BS=8` and `N=24`. Each task should be clearly labeled with the values of ii, jj and its cost in time units.

2. (2.0 points) Compute the values for T_1, T_∞ and P_{min}. Draw the temporal diagram for the execution of the TDG in the previous question on P_{min} processors. As indicated, consider the cost of the innermost loop body to be t_c time units.

3. (2.0 points) Assume the same task definition, and consider a distributed memory architecture with P processors, $BS = N/P$, and N a very large value multiple of P (you can assume $BS-1$ is approximately the same as BS to simplify the model). Let's assume that matrix A is initially distributed by columns (N/P consecutive columns per processor) and tasks are scheduled so that a task is executed in the processor that stores the data the task has to update.

We ask you to:

(a) (1.0 points) Draw the time diagram for the execution of the tasks in P processors clearly identifying the computation and the data sharing time. Note: you can assume $P = 3$ for this question a).

(b) (1.0 points) Write the expression that determines the execution time, T_p as a function of N and P clearly identifying the contribution of the computation time and the data sharing overheads, assuming the data sharing model explained in class in which the overhead to perform a remote memory access is $t_s + t_w \times m$, being t_s the start-up time, t_w the time to transfer one element and m the number of elements to be transferred; at a given time, a processor can only perform one remote access to another processor and serve one remote access from another processor.

Problem 2 (5.0 points) Given the following sequential recursive algorithm:

```
#define N 1024
#define NSTATES 128
#define MINROWS 2

int histogram[NSTATES];

int base_processing (int data[N][N], int start, int nrows) {
     int outofrange=0;

     for (int i=start; i<start+nrows; i++) {
       for (int k=0; k<N; k++) {
          int value = compute (data[i][k]); /* perform computation on the parameter */
          if (value >= NSTATES)
               outofrange++;
          else if (value >= 0)
               histogram[value]++;
       }
     }
     return outofrange;
}

int rec_processing (int data[N][N], int start, int nrows) {
    int res1, res2=0;

    if (nrows < MINROWS)
            res1 = base_processing (data, start, nrows);
    else {
            res1 = rec_processing (data, start, nrows/2);
            res2 = rec_processing (data, start+nrows/2, nrows-nrows/2);
    }
    return res1 + res2;
}

int main() {
  int data[N][N];
  ...
  int res = rec_processing(data, 0, N);
  ...
}
```

We ask you to answer the following **independent** questions:

1. (2.5 points) Write an OpenMP parallel version of the `base_processing` function, following an *Iterative* Task Decomposition, making use of the OpenMP explicit tasks. Your implementation should minimize synchronization overheads and take into account **the potential imbalance** generated by the `compute` function.

Some considerations about the all the proposed solutions:

- All of them have `parallel` and `single`.
- All of them create explicit tasks.
- All of them avoid creating tasks with only one iteration of k (too much task creation overhead).
- All of them avoid creating tasks doing full loop k (too much imbalance due to compute).
- All of them avoid waiting for all tasks at each iteration of i (we are looking for parallelism). I.e. nogroup should be used in the taskloop, but them reduction is not allowed.
- All of them perform a reduction of `outofrange` variable (avoid data race condition and reduce data synchronization).
- All of them perform an of `atomic` to update histogram (reduction could be possible also but it may be costy in memory).

```
*/
```

*/

```
*/
```

2. (2.5 points) Write an OpenMP parallel version of the sequential recursive program, following a *Recursive* Task Decomposition using the *Tree* strategy. The implementation should take into account the overhead due to task creation, by limiting their creation once a certain level in the recursive tree is reached.