Computação Paralela

November-2020

Optimizing Performance

These exercises aim to introduce performance optimizations on shared memory systems. Compile the program with `gcc -O3 -g -std=c99 -fopenmp -lm <<file>>`. Set the number of threads, before running the program, with `export OMP_NUM_THREADS=<<NT>>`

Consider the following OpenMP program:

```c
#include <stdio.h>
#include <math.h>
#include <omp.h>

int main(){
    int result[1];

    #pragma omp parallel for schedule(dynamic,1)
    for(int i=0; i<1000000;i++) {
        result[0]+=sin(i);
    }
}
```

1) Measure the scalability (program execution time) and explain the results. Use the command `time ./a.out` to measure the execution time with 1, 2, 4 and 8 threads and/or use the `perf` tool to see the program profile with `perf record ./a.out` and `perf report` for 1, 2 and 4 threads.

2) Experiment the impact on performance of increasing the magnitude of the chunk size (task size, e.g., `schedule(dynamic,10)`) and compare with static loop scheduling (by removing the `schedule` clause).

3) Measure the scalability by comparing `critical` and `atomic` directives to avoid the data race in the shared variable (`result[0]`).

4) Change the program to use a value local to each thread to avoid the data race in the `result` variable. Suggestion: use a larger `result` array (e.g., with 8 positions) and a different position for each thread. The primitive *omp_get_thread_num()* to get the id of each running thread.