**Lab1: Experimental setup and tools**

Oscar Faixat Llanas

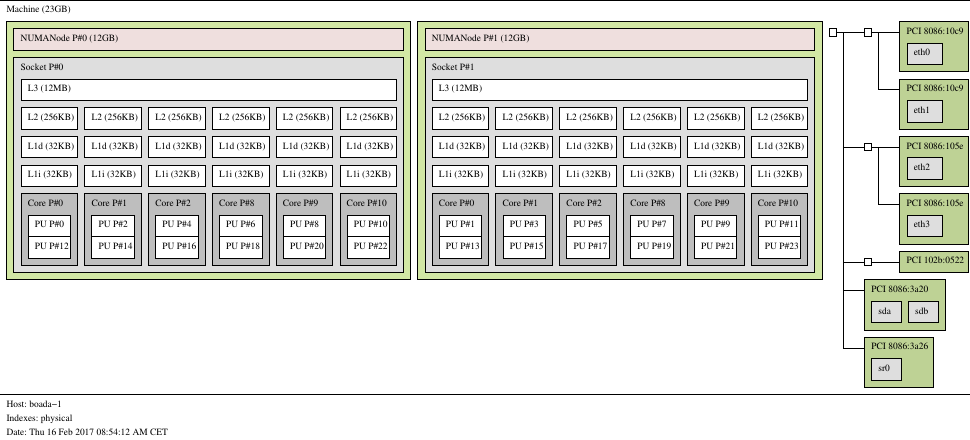
Xavier Algarra Torelló

Grup: PAR1212

PAR 2017Q2

02/16/2017

**1. Draw and briefly describe the architecture of the computer in which you are doing this lab session (number of sockets, cores per socket, threads per core, cache hierarchy size and sharing, and amount of main memory).**



**Computer arquitecture:**

* 2 Sockets
* 6 cores per socket
* 2 threads per core
* main memory of 23 GB
* 2 Numanodes of 12GB each
* There are 12MB of level 3 cache which are shared between all 6 cores. For
* eah core there are 256KB of level 2 cache. Finally there are 32KB of Level 1
* cache for instructions and 32KB of level 1 cache for addresses.

**2. Describe what do you need to add to your program to measure the elapsed execution**

**time between a pair of points in the program, clearly indicating the library header file**

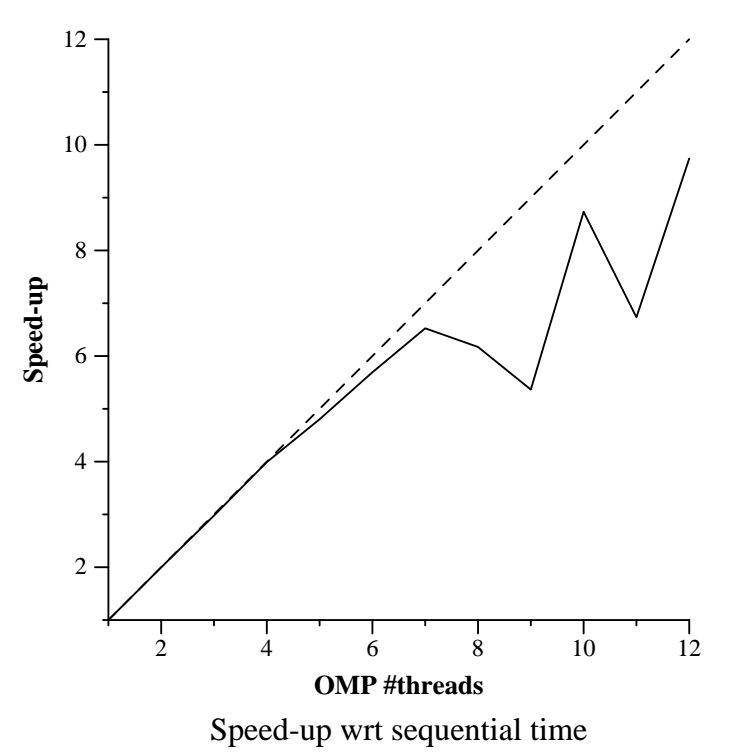
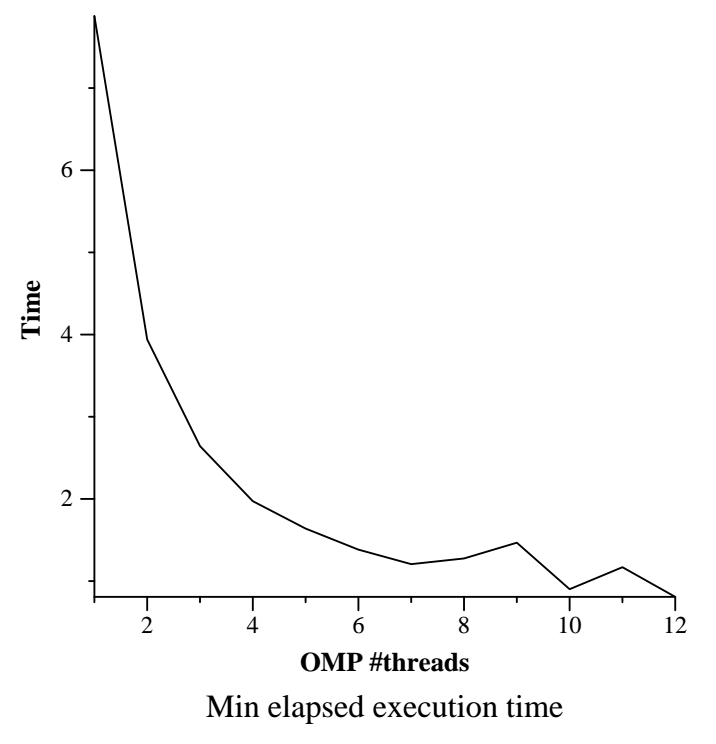
**that needs to be included, the library functions that need to be invoked, the data**

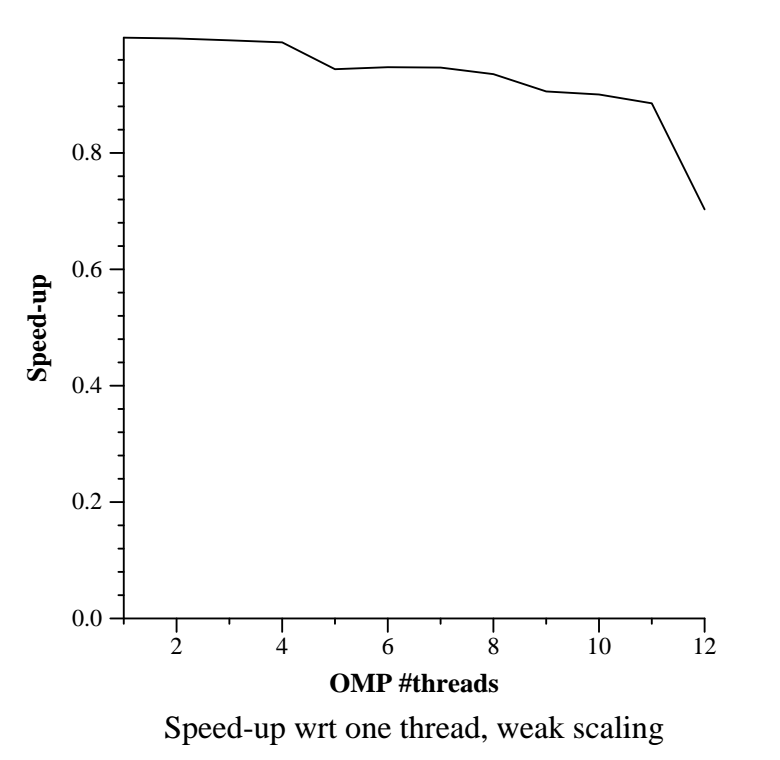
**structure and its fields.**

In order to measure the elapsed execution time we need to add <sys/time.h>.

* The library function that needs to be invoked is: gettimeofday().
* The data structure is timeval and its fields are:
  + time\_t tv\_sec
  + long int tv\_usec

**3. Plot the speed–up obtained when varying the number of threads (strong scalability) and problem size (weak scalability) for pi omp.c. Reason about how the scalability of the program.**



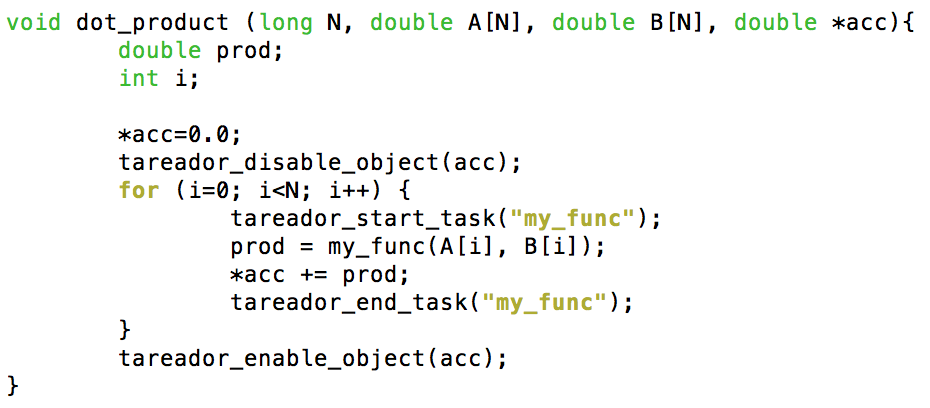


In the strong scalability plot we can observe the speedup growing linearly at the beginning of the plot and then suddenly dips because of synchronization issues. Overall we believe that the program has a somewhat good scalability.

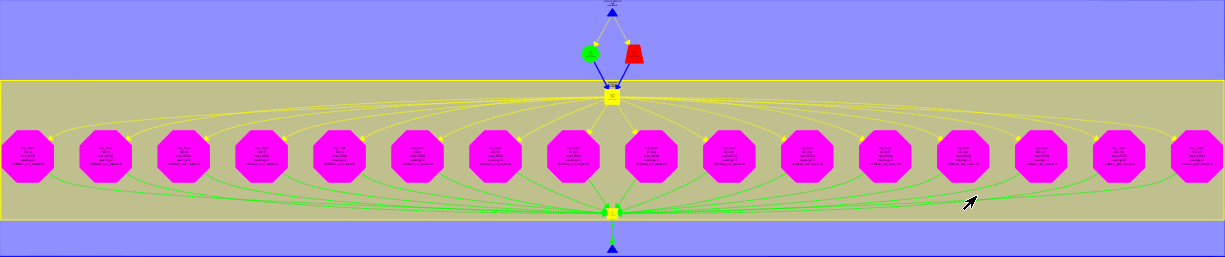
In the weak scalability plot we can observe the speedup at the beginning is similar until it reaches 5 threads. However, after dipping it is somewhat stable up to 11 threads, after which the plot destabilizes again due to synchronization issues.

It is worth mentioning, that these results can vary from execution to execution simply because of memory access failures. In order to obtain more reliable results, we would need to plot the average of various executions.

**4. Include the source code for function dot product in which you show the Tareador instrumentation that has been added to study the potential parallelism in the code. This instrumentation has to appropriately define tasks and filter the analysis of variable(s) that cause the dependence(s).**



**5. Capture the task dependence graph for that task decomposition and the execution timelines (for 8 processors) that allow you to understand the potential parallelism attainable. Briefly comment the relevant information that is reported by the tools.**

****

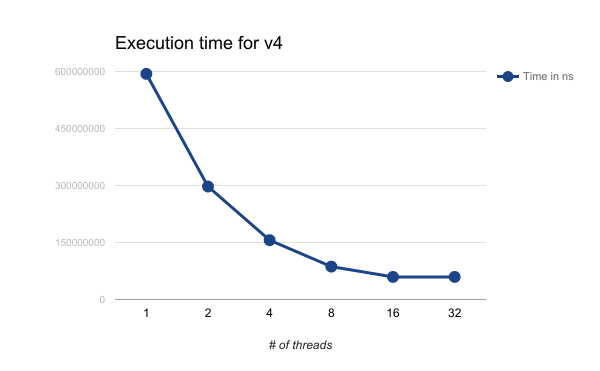


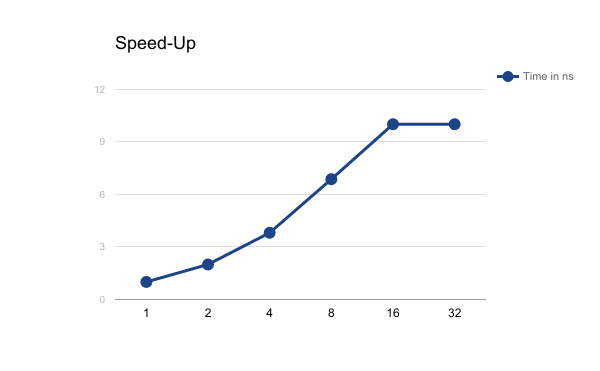
Once the matrices have been initialized, the program can start the parallelization of the dot\_product function. At the beginning however, we can see that there is a decent amount time used up synchronizing tasks. At the end, the main thread, has to wait until all of the other threads are done executing and synchronizing in order to end the program.

**6. Complete the following table for the initial and different versions generated for 3dfft seq.c, briefly commenting the evolution of the metrics with the different versions.**

|  |  |  |  |
| --- | --- | --- | --- |
| version | **T1** | **TInf** | **Paralellism** |
| **seq** | 593772001 | 593705001 | 1.000112851 |
| **v1** | 593772001 | 593705001 | 1.000112851 |
| **v2** | 593772001 | 315188001 | 1.883866134 |
| **v3** | 593772001 | 108352001 | 5.480028015 |
| **v4** | 593772001 | 59415001 | 9.9936378188 |

It seems that the smaller the granularity of the task, the better parallelization we will be able to obtain. However, this data is from a simulation and in the real world this will not be true as the smaller the granularity the more synchronization time we will need.

**7. With the results from the parallel simulation with 2, 4, 8, 16 and 32 processors, draw the execution time and speedup plots for version v4 with respect to the sequential execution (that you can estimate from the simulation of the initial task decomposition that we provided in 3dfft seq.c, using just 1 processor). Briefly comment the scalability behaviour shown on these two plots.**



﻿In the execution time for v4, we can see that there is a great benefit in using parallelization up until 16 threads. Using more than 16 threads provides a slight improvement however, it is not really noticeable. Observing the speedup plot, we can observe more or less what we deduced from the previous plot, that initially the speedup increases almost exponentially, and then after reaching 16 threads it stabilizes and there is not that much speedup.

**8. From the instrumented version of pi seq.c, and using the appropriate Paraver configuration file, obtain the value of the parallel fraction φ for this program when executed with 100.000.000 iterations, showing the steps you followed to obtain it. Clearly indicate which Paraver configuration file(s) did you use.**

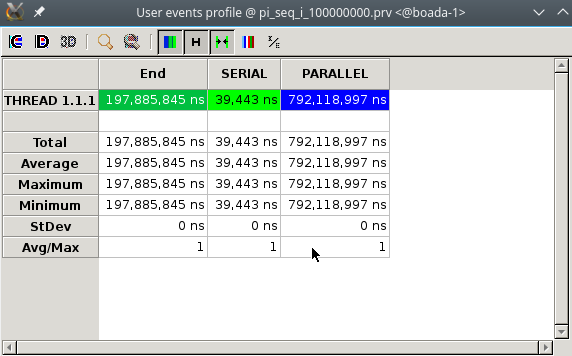
Φ = (parallel)/(parallel + sequential)

Φ = 792,118,997/(792,118,997+39,443)

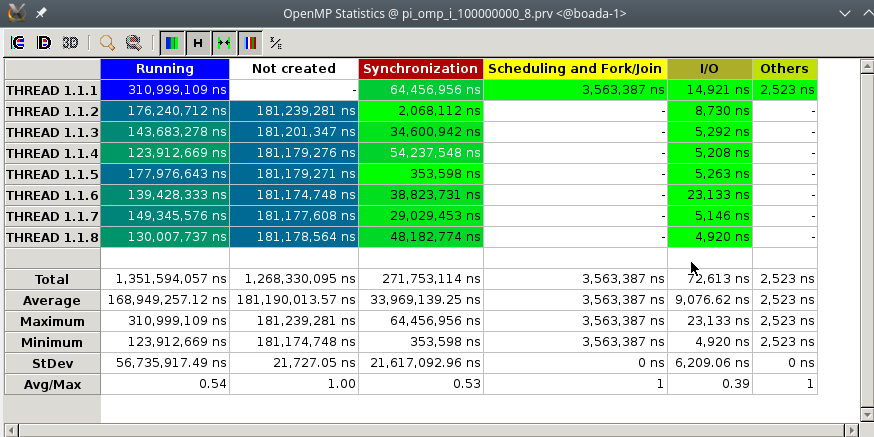
Φ = 0.9998

In order to obtain the chart to the left, we used wxparaver with the configuration:

APP\_userevents\_profile.cfg



**9. From the instrumented version of pi omp.c, and using the appropriate Paraver configuration file, show a profile of the % of time spent in the different OpenMP states when using 8 threads and for 100.000.000 iterations. Clearly indicate which Paraver configuration file(s) did you use and your own conclusions from that profile.**

****

The profile that we used in paraver is omp\_state\_profile.cfg. We can observe that there is a lot of time spent doing synchronization except in threads 1.1.2 and 1.1.5. Doing scheduling and fork/join operations barely takes up any time. Given this information it seems like increasing the number of threads again will add a lot of synchronization time and there won’t be as much speed-up.