CPA Lab 2

Carlos Santiago Galindo Jiménez
Jesús Vélez Palacios

November 3, 2016

1 Objective

Parallelize the execution of a program that reconstructs an image whose rows have been shuffled. Study different approaches and some improvements to the algorithm.

2 Exercices

`encaja-e1.c` Time the method encaja. Completed with two calls to `omp_get_wtime()`.

`encaja-e2-p?.c` Parallelize the different loops (if at all possible).

- `i` This loop will never be parallelizable, because each line depends on the previous one.
- `j` This loop can be parallalelized protecting the variables with `private(x, distancia)`. The last `if` must be protected as `critical section` and the condition has to be rechecked after entering it. Source code
- `x` This loop can also be parallelized, and needs a `reduction(+:distancia)` to compute the distance correctly. Source code

`encaja-e3.c` Improve the program by exiting loop `x` if the partial sum surpasses the current minimum. Solved adding an `if` that breaks from the loop in that case.

`encaja-e4-p?.c` Parallelize `encaja-e3.c` in the loops that are viable.

- `j` This loop is parallelized in the same way as in exercise 2. Source code
- `x` This loop needs the most work. It must be parallelized as a `parallel` block. The first iteration is assigned to `omp_get_thread_num()` and each thread increments `x` by `omp_get_num_threads()`. It still needs a `reduction` for `distancia`. Source code

3 Performance

3.1 Timing

The sequential programs (`encaja-e1.c` and `encaja-e3.c`) have only been tested once. The parallel programs have been run with 2, 4, 8, 16 and 32 threads available to test their performance. As shown in the graph from Figure 3, loop `j` achieves much better results that loop `x`. On top of that, the improvement implemented in the exercises 3 and 4 pays off in the sequential version, but less and less as the number of threads grows bigger.

Program	Execution time
`encaja-e1`	15.257217
`encaja-e3`	2.725587

Figure 1: Sequential execution times

[2]

[2]The data in the graph for *nodes* = 1 is the timing from the sequential versions of the program (`encaja-e1.c` and `encaja-e3.c`)

Program	2t	4t	8t	16t	32t
`encaja-e2-pJ`	6.938746	3.548883	1.970450	1.183996	0.726187
`encaja-e2-pX`	8.337415	5.464151	5.557857	8.197272	12.789012
`encaja-e4-pJ`	1.439842	0.795035	0.496000	0.330912	0.256960
`encaja-e4-pX`	4.065065	4.217589	5.164125	8.656438	13.590685

Figure 2: Parallel execution times

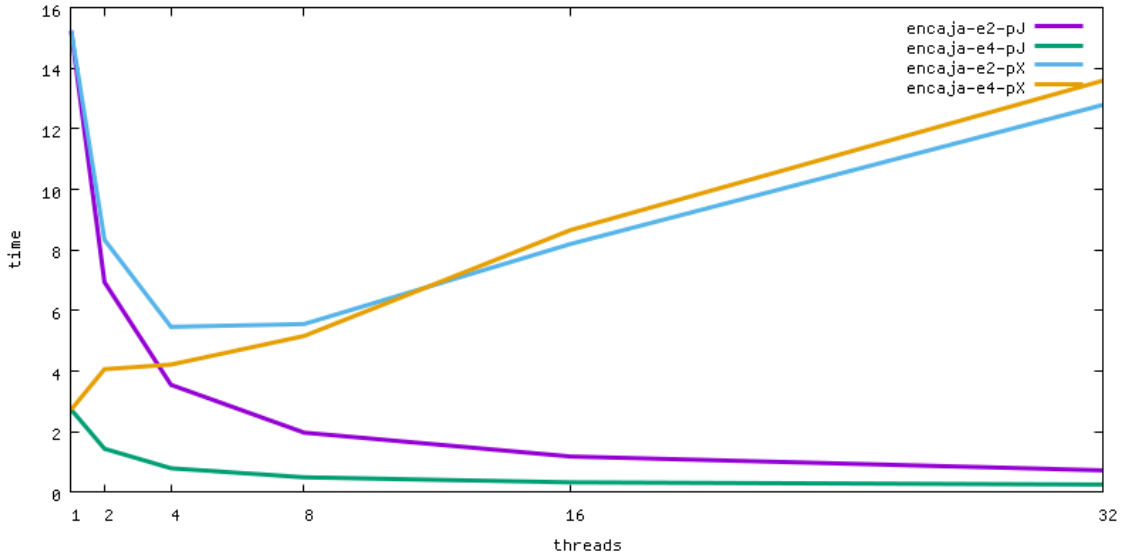

Figure 3: Threads — time graph [1]

3.2 Speedup

The speedup of a parallel program respect to the sequential version is the improvement (or not) of the parallel version out of one, and it is computed as:

$$S(n,p) = \frac{t(n)}{t(n,p)}$$

Applying the previous formula to the tables we obtain these.

Analyse data!

insert tables

3.3 Efficiency

introduce

$$E(n,p) = \frac{S(n,p)}{p}$$

introduce tables/graph and analyse data

insert tables

4 Conclussions

write conclussions

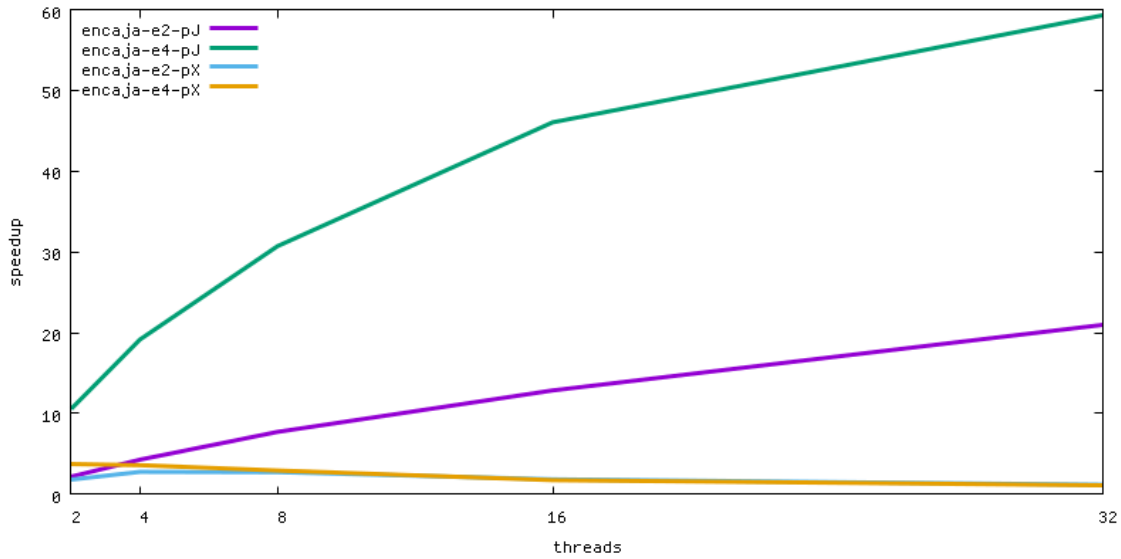

Figure 4: Threads — speedup graph

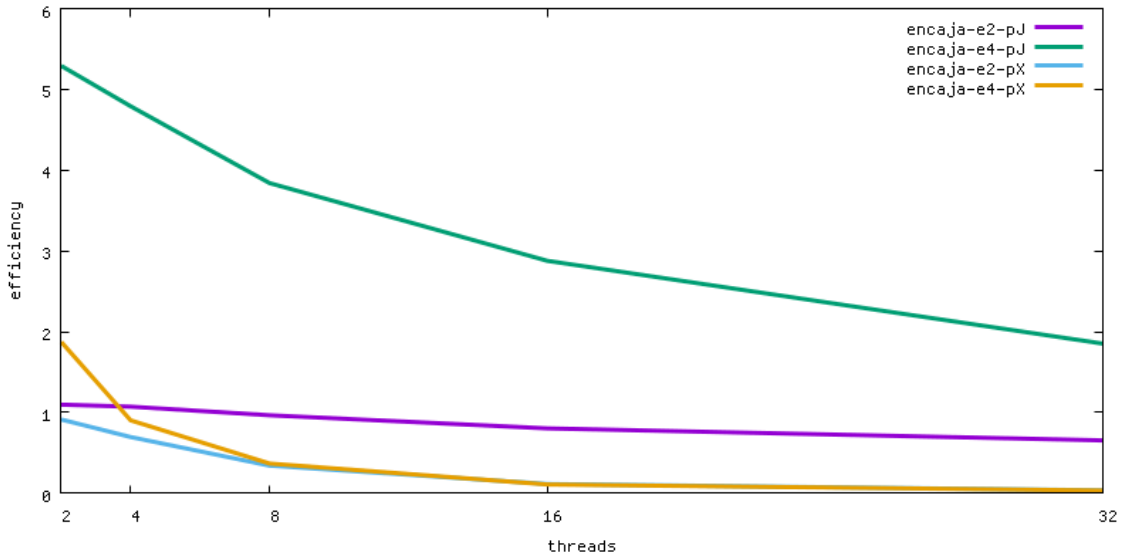

Figure 5: Threads — efficiency graph