EPFL

Parallel and High Performance Computing

Dr. Pablo Antolín

Series 0

February 20 2025

Some basic concepts

This serie can be done exclusively with pen and paper. The objective is to introduce a few basic concepts and to perform the theoretical analysis of a code and understand if it is worth or not to parallelize its sequential version and what could be its theoretical parallel performance. The answers to exercises can be found in the slides of the first lecture and the videos proposed on moodle.

Exercise 1: *Big O notation*

a) Define in one sentence time and space complexities of an algorithm.

b) Assume you have two algorithms A1 (with time complexity $\mathcal{O}_t(N \log N)$ and space complexity $\mathcal{O}_s(N)$) and A2 (with $\mathcal{O}_t(N^3)$ and $\mathcal{O}_s(N^3)$). Which one is worth to be parallelized?

Exercise 2: *Vocabulary*

a) Why a DAG (Direct Acyclic Graph) is useful for the analysis of a parallelization strategy for a particular algorithm?

b) What is the critical path in a DAG?

c) What are the three main kinds of parallelism?

d) What is one of the main solutions for solving the "irregular parallelism" problem?

Exercise 3: *Machine model*

a) What ILP, TLP, and DLP stand for?

b) What is a vector intrinsic?

c) Sort the following memories according to their speed of access: DRAM, L2, L1, L3, SSD.

d) Which are the 4 types of parallel architectures according to Flynn's taxonomy? Provide an example of each.

Exercise 4: ***Threads and processes***

a) What are the main differences between thread and process?

Exercise 5: ***Performance aspects***

a) What is the definition of speedup?

b) What is the definition of parallel efficiency?

Exercise 6: ***Performance aspects of the 2D Poisson solver***

We are going to study a Poisson equation solver in 2D. The Poisson's problem reads:

$$\Delta u = f \quad \text{in } \Omega = [0,1] \times [0,1] \tag{1}$$

$$u = 0 \quad \text{on } \partial\Omega \tag{2}$$

with

$$f(x,y) = -2\pi^2 \sin(\pi x)\cos(\pi y). \tag{3}$$

In 2D, the partial differential equation (1) becomes

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u(x,y) = f(x,y). \tag{4}$$

The equation is discretized in Ω as a rectangular grid of size $N \times N$. We use a centered finite differences discretization scheme to approximate (4), that yields a linear system $Ax = b$. Using a Jacobi method, that system can be solved iteratively by expressing u^{k+1} as a function of u^k (solution u at iteration k). Thus, introducing the discretization scheme we obtain

$$u_{i,j}^{k+1} = \frac{1}{4}\left(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k - f_{i,j}h^2\right), \tag{5}$$

where $h = 1/N$. $u_{i,j}^k$ corresponds to the solution u at iteration k, at the point i,j of the grid, with $i,j = 1,\ldots,N$. The discrete counterpart of $f(x,y)$ reads $f_{i,j} = -2\pi^2 \sin(\pi i h)\cos(\pi j h)$.

Below you can find the bulk of a sequential code to solve the 2D Poisson equation using Jacobi's method. We set `EPS = .005` the minimal error that the L^2-norm must reach.

```c
#define EPS  0.005
#define N    1024

int main() {
  const float h = 1. / (float)N;

  // Allocation of the arrays of pointers
  float **u, **uo, **f;
  // malloc . . .

  // initialization of u0 and f
  // for(int i = 0; i < N; ...

  k = 0;
  const float start = second();
  do {
    float l2 = 0.;

    for(int i = 1; i < N-1; ++i) {
      for(int j = 1; j < N-1; ++j) {
        // computation of the new step
        u[i][j] = 0.25 * (uo[i-1][j] + uo[i+1][j] + uo[i][j-1] + uo[i][j+1] -
        ↪  f[i][j] * h * h);

        // L2 norm
        l2 += (uo[i][j] - u[i][j]) * (uo[i][j] - u[i][j]);
      }
    }

    // copy new grid into old grid
    for(int i = 0; i < N; ++i) {
      for(int j = 0; j < N; ++j) {
        uo[i][j] = u[i][j];
      }
    }

    k++;
  } while(l2 > EPS);

  const float t = second() - start;
  printf("T=%.5f s for %d steps (%.5f s/step)\n", t, k, t / k) ;

  // Deallocation
  // free ...

  return 0;
}
```

a) Compute the computational complexity of the algorithm ("big O" notation) in time.

b) Compute the computational complexity of the algorithm ("big O" notation) in space.

c) Is this problem worth to be parallelized?