Project of Distributed Systems: Paradigms and Models
Iterative Jacobi Method

Student: Diego Arcelli
Matricola: 647979
Professor: Marco Danelutto

May 26, 2022

UNIVERSITÀ DI PISA

Contents

1 Description of the problem

The Jacobi method is an iterative algorithm which can be applied to solve systems of linear equations of the form:

$$Ax = b$$

where $A \in \mathbb{R}^{n \times n}$ is the coefficient matrix, $b \in \mathbb{R}^n$ is the constant terms vector and $x \in \mathbb{R}^n$ is the unknown vector that we want to compute. The idea of Jacobi algorithm is to decompose A as a sum of three matrices:

$$A = D + U + L$$

where D is a diagonal matrix, U is an upper-triangular matrix and L is a lower-triangular matrix. Then we compute iteratively:

$$x^{(k+1)} = D^{-1}[b - (U + L)x^{(k)}]$$

where $x^{(k)}$ indicates the value of x at the k-th iteration. Actually at each iteration k we can compute independently every component i of the array using the equation:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}}\Big(b_i - \sum_{j=1}^{n} a_{i,j} x_j^{(k)}\Big) \tag{1}$$

where x_i indicates the i-th component of vector x. Discussing further details of the algorithm is beyond the scope of this project.

2 Analysis of the problem

The pseudo-code to execute the algorithm sequentially for a fixed amount of iteration l, is shown in 1. Note that at each iteration we're forced to use an auxiliary vector x' to compute the updated elements of x for the current iteration, and then, before starting the following iteration, we copy the elements of x' in x. This is necessary because suppose that at iteration $k+1$, we compute the updated value for the i-th component $x_i^{(k+1)}$ and we modify x in-place, then when we compute the update for the next component $x_{i+1}^{(k+1)}$ we won't use x_i^k, but we will use x_i^{k+1}, which is not mathematically correct. The algorithm is composed by an three nested loops, where the first iterates l times, the other two iterate n times, and all the other operations can be done in constant time, so the total time complexity is $O(ln^2)$.

Algorithm 1 Sequential code for Jacobi method

Require: A matrix, b and x vectors, l positive integer

```
for k ← 1 to l do
    for i ← 1 to n do
        val ← 0
        for j ← 1 to n do
            val ← val + A[i, j] ∗ x[j]
        end for
        x'[i] ← 1/A[i,i] (b[i] − val)
    end for
    for i ← i to n do
        x[i] ← x'[i]
    end for
end for
```

Let's now analyze how we can exploit the properties of the problems in order to parallelize the algorithm. The first important thing we notice is that in order to compute the updated values for the vector x at iteration k, we need the full updated vector x at iteration $k-1$, therefore the iterations of the algorithm needs to be strictly sequential. So let's focus on understanding what we can parallelize inside a single iteration. As we said in the description of the problem, at each iteration k every component of the new vector $x^{(k+1)}$ can be computed independently from the others components using the formula in equation 1. So this is a data parallel computation where (for the reasons described before) we cannot modify the vector x in place. So we can parallelize the code by assigning to each worker a different partition of components of x to update in the auxiliary vector x'.

Another part of the code which can be parallelized is the for loop used to copy the updated values from x' to x, since each iteration i updates only the i-th element of x. However, as we already discussed, before executing the copies, all the workers must have finished to update their chunk of vector, otherwise if a worker w_1 executes the copies of its chunk, while a worker w_2 is still computing the updates of its partition, this would compromise w_2 computations since to execute the updates w_2 needs to access to the components of x updated by w_1. Basically we simply need to put a barrier between the update phase and the copy phase.

Actually another barrier is needed to assure that all the workers ended their copy phase before starting the next iteration, otherwise it might happen that some workers start the next iteration working on not updated values of x.

The last thing we need to discuss is how to stop the the execution of the workers after l iteration. This can be easily implemented by selecting one of the worker to maintain a counter of the number of iterations. Between the update and next iteration barrier we can check whether the id number of the worker is a specific one, and if it is the case update the iteration counter. This solution

does not require mutual exclusion, since the counter is modified by just one worker, moreover it guarantees the correctness of the program since, because of the next iteration barrier, all the worker will test the condition of the while loop with the same value of the current iteration.

Actually there's still another part in the code which might been further parallelized. The for loop used to compute $A[i,j] * x[j]$ for $j = 1, \ldots, n$ is a summation of $n-1$ elements, so it's a classical case of an application of a reduce pattern. However the reduce is used inside the map, so basically we ave a pattern of the following form:

$$map(reduce(sum, n_w^2), n_w^1)$$

The routine of a single worker is described in 2, where of course *start* and *end* are two variables to store respectively the first and the last index of the chunk of the worker, assuming that each worker works on a contiguous partition of indices.

Algorithm 2 Single worker routine

Require: A matrix, b and x vectors, l positive integer
  while $k < l$ **do**
    $val \leftarrow 0$
    for $i \leftarrow start$ to end **do**
      for $j \leftarrow 1$ to n **do**
        $val \leftarrow val + A[i,j] * x[j]$
      end for
      $x'[i] \leftarrow \frac{1}{A[i,i]}(b[i] - val)$
    end for
    Updates barrier
    for $i \leftarrow start$ to end **do**
      $x[i] \leftarrow x'[i]$
    end for
    if $wid = 1$ **then** $k \leftarrow k + 1$
    end if
    Next iteration barrier
  end while