A pre-design for the Jacobi method implementation. A.Y. 2016/2017

Matteo Busi
matteo.busi42@gmail.com

February 5, 2017

1 Introduction

A Jacobi iterative method implementation to solve linear systems is required. Given a linear system in the form:

$$Ax = b$$

where the searched solution is $x \in \mathbb{R}^N$, and the known data are $A \in \mathbb{R}^{NxN}$, $b \in \mathbb{R}^N$.
The Jacobi method works by decomposing A as $D + R$ s.t.

$$d_{i,j} = \begin{cases} a_{i,i} & \text{if } i = j \\ 0 & \text{o.w.} \end{cases} \qquad r_{i,j} = \begin{cases} a_{i,j} & \text{if } i \neq j \\ 0 & \text{o.w.} \end{cases}$$

and solving for x. At the $k+1$-th iteration the approximate solution is:

$$x^{(k+1)} = x^{(k)} + D^{-1}(b - Ax^{(k)})$$

given an initial guess $x^{(0)}$. The method proceeds as a convergence criterion is met, e.g. $||Ax - b||_2^2 < \varepsilon$, with ε the wanted tolerance.

2 Discussion

The idea is to given three different implementations: a sequential one, a `pthread` one, and a `FastFlow` one using the *parallel for*.

2.1 Outline of the ideas

A possible schema for the sequential implementation is the following:

```
Initialize A, b, x^(0).
Compute D, D^-1.
k = 0
Δx = D^-1(b − Ax^(k))

while ||Δx||_2^2 ≥ ε:
        Δx = D^-1(b − Ax^(k))
        x^(k+1) = x^(k) + Δx
        k++
```

As is evident from the pseudo-code above the critical points are in the evaluation of the computation of Δx and in the computation of the norm.
In the `pthread`-based implementation the following considerations will be used:

Computation of Δx It can be both vectorized and parallelized. In this case each worker will compute the product of a matrix row and the vector.

The norm evaluation can be parallelized and vectorized too, but some profiling is needed to determine whether is necessary or not to parallelize the computation, since it may be a too fine-grained computation.

Moreover some sort of synchronization at the end of the body of the loop is needed to correctly compute the norm, this may affect the performance in case of unbalanced load between workers (e.g. with some particular matrices).
For `FastFlow`-based implementation most of the ideas stated above hold. Most important differences lies in the parallelization of the computation of Δx that will be achieved using `FastFlow`'s *parallel for* and in the absence of the need of *explicit* synchronization mechanisms at the end of body loop.