Università di Pisa

Department of Computer Science

SPM Final Project Report

The Jacobi Iterative Method

Matteo Busi

Student ID. 494087

June 9, 2017

1 Introduction

The aim of this project was to produce a program to solve linear systems using the Jacobi method.

Three different implementations are proposed here:

Sequential implementation the sequential implementation provides a vanilla implementation of the Jacobi method,

Thread implementation is a nïve implementation of the algorithm using `threads` from `C++11` standard,

FastFlow implementation is an implementation using the `parallelFor` from FastFlow library.

Tests were conducted on a machine using a *Intel Xeon E2650* CPU (8 cores clocked at 2 GHz each with 2 contexts) and a *Intel Xeon Phi* co-processor (60 cores clocked at 1 GHz each with 4 contexts).

Summary. The next section discusses the details of program design, including theoretical analysis of the expected performance of the parallel implementation. In Section 3 reports details about the implementation, discussing main aspects of the code and its optimization. Section 4 is divided in two sub-sections. The first sub-section discusses the methodology for the experiments and chosen parameters, while the second sub-section reports the experimental results in the form of tables and graphs. Section 5 includes the user manual for the program, and indications on how to reproduce results reported here. Finally Section 6 compares obtained results against the expected ones.

2 Design

3 Implementation

4 Experiments

4.1 Methodology

4.2 Results

5 User guide

This section provides a short guide on how to use the program, how to conduct experiments and how to gather results.

5.1 Workspace

The workspace content is organized as follows:

- The folder `bin` contains the results of the compilation (including vectorization reports),
- the folder `graphs` contains the graphs generated by `reportgen.py`,
- the folder `results` contains the collection of `csv` files generated by `jacobirun.sh`,
- the folder `src` contains the source code of the program,
- the bash script `jacobirun.sh` contains the code to run experiments,
- the Python program `reportgen.py` that generates graphs starting from data in `results` folder,
- the make file `Makefile` compiles the project as explained in subsection 5.2.

In the following we assume that the current working directory is the root of the workspace.

5.2 Compilation

To compile the project a `Makefile` with four rules is provided:

1. Executing `make jacobix` the executable for the Xeon CPU is produced and placed in `bin/jacobix`,
2. executing `make jacobim` the executable for the Xeon Phi is produced and placed in `bin/jacobim`,
3. executing `make offload` the executable for the Xeon Phi is produced and placed in both `bin/jacobim` and in the home directory on `mic1`,
4. executing `make clean` the files produced by compilation, testing, and analysis are deleted.

5.3 Program usage

To run a single resolution of a random system one of the compiled executables located in `bin` must be run. Executable `jacobix` runs on the Xeon CPU, while executable `jacobim` must be offloaded to the Xeon Phi.

Executing one of the executable without arguments produces as output a guide that should be self-explaining:

```
Usage: ./jacobi N ITER ERR METHOD [NWORKERS] [GRAIN]
Where:
    N : is the size of the matrix A
    ITER : is the maximum number of iterations
    ERR : is the maximum norm of an acceptable error
    METHOD: is either
        s : indicating that the sequential implementation
            must be used
        f : indicating that the FastFlow implementation
           must be used
        t : indicating that the Thread implementation
           must be used
    NWORKERS : the number of workers that should be used
       (ignored if METHOD is 's')
    GRAIN : the grain of the computation (only if METHOD
       is 'f')

Produces a CSV line, in the form:
N_WORKERS, N_ITERATIONS, COMP_TIME, UPD_TIME, CONV_TIME,
   LATENCY, ERROR
```

5.4 Experiments and analysis

After compilation, to execute the experiments and analyse the results one must:

1. run `./jacobirun.sh` or `./jacobirun.sh MIC` (if Xeon Phi must be used),
2. run `reportgen.py` to produce graphs.

6 Conclusion