ELMFIRE. Installation guide and performance analysis

Miquel Català
Alejandro Soba

Barcelona Supercomputing Center

Technical Report TR/CASE-08-2

Jun 2008

ELMFIRE. Installation guide and performance analysis

Miquel Català[1] **and Alejandro Soba**[1]

[1] *Computer Application and Science Engineering, Barcelona Supercomputing Center, Edifici Nexus, Campus Nord UPC, Gran Capità 2-4, 08034 Barcelona, Spain.*
E-mail: miquel.catala@bsc.es, *Web page:* http://www.bsc.es/plantillaE.php?cat_id=473

Abstract. In this report we present the necessary steps to install and run the code ELMFIRE into MareNostrum. Several profiling analysis and performance test were realized in order to prove the code behaviour. We report some code's difficulties found in our architecture.

Key words. EUFORIA, ELMFIRE code, installation guide, profiling

1. Introduction

ELMFIRE is a gyrokinetic particle simulation (PIC) code for plasmas in toroidal geometry. The code has been built to study the development of instabilities in a quasineutral plasma, and its influence in the transport coefficients. It is single-threaded and mainly Fortran 90 code, with some auxiliar C functions.

In order to do that the code calculated the dynamics of a big number (10^8) of markers (each representing about 10^{10} actual plasma particles) in a self-consistent electromagnetic field. Plasma particles are electrically charged and both create and suffer the electromagnetic field.

The computational scheme of the code can be summarized in the following simplificated steps:

1. Particle initialization
2. Particle movement in existing E,B fields.
3. Binary collisions of nearby particles.
4. Calculation of new E field (current model assumes fixed B).
5. Implicit corrections to calculated movement from new E field.
6. Diagnostics
7. Goto 2 until t = t_end

Sections 1), 2), 3), 5) work with perfect parallelization, since they work on local variables. The only interaction between particles in different processors happens through the globally created E field.

Section 4) and 6) requires collective calls for joining the sparse matrix and right hand side of the discretized Poisson equation for calculating the electrostatic potential from current (and implicitely estimated) particle positions. The code has a restart system based on optimized MPI-IO, that allows for long runs to be splitted into shorter sub-runs adaptable to the queuetime available ([1, 2]).

The code was analyzed in MURSKA. For the scaling results in louhi a production test run has been used. However, it is impossible to find a suitable run that is appropiate for a range of processors 16-256. Processor number is selected so that the heaviest task (particle motion) is calculated at a reasonable rate. Normally that would mean keeping the number of particles per processor in the interval (0.7, 2.0)x 10^6.

2. Code compilation

2.1. Source code

Code is a set of sixty FORTRAN files and three C files. Moreover there are twelve header FORTRAN files. Although the code is written primarily in FORTRAN77, an slow but on-going process is to develop and port the code to FORTRAN90.

2.2. Libraries

The libraries which are used (or may be used) in the code are MPI, BLACS, PESSL, IMSL, and optionally NAG. PESSL and NAG are used only for solving the large linear system which arises from the discretization of the field equations for the electric field. Also the IBM MASS library is used to expedite the calculation of transcendental functions.

2.3. Makefile

The makefile are composed by three files. The main one is called `makefilefile` contains general information of the compilation, precompilation and link process. Beside that contains the calls to the other two makefile's components and the name of the user election work directory in the `WRKDIR` variable.

The second one is called `xtras/makefile.common` and is a switch for the architecture specific third file, which in our case is called `xtras/makefile.MareNostrum`. It defines the compilers for both C and FORTRAN languages and their flags. Here there are defined the mathematical libraries.

3. Input data

Input file name might always be labeled `icri.inp` . This file includes all relevant input parameters, detailed in README file (section 2.4).

An `icri.inp` example is distributed with the source code, that allows a first execution to check if the code is well compiled and installed.

For more exhaustive testing proposes we use two input file used to run the code in the MURSKA cluster in order to check performance ([3]).

4. Execution

To launch `elmfire` the only necessary command is the binary without parameters. The input file `icri.inp` need to exist in the same directory.

Here we show the above mentioned test, the small one (1) and the big one (2), and their behaviour when they are run.

4.1. Found problems

The first elmfire's version installed (version 10) in Mare Nostrum generates a segmentation fault after two or three simulation time steps.

Last version (version 11) is able to run due to a source code's modification realized by one of the authors*.

*Fco Ogando. fogando@ind.uned.es

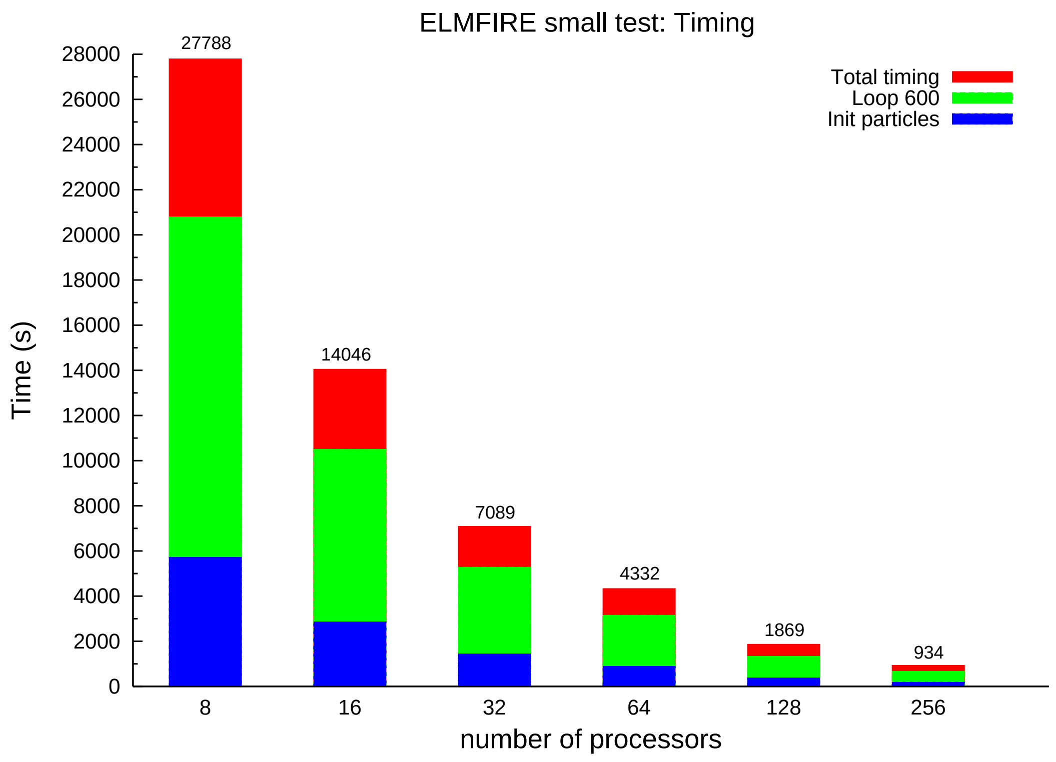

Figure 1. Elmfire: small test

5. Profiling

Last version of code was unable to be instrumented. It is possible to add the -pg flags to build the binary but the new execution flows ends up to a segmentation fault.

We believe that the code needs another source modification in order to run the instrumented version.

5.1. *Small test*

The *small test* uses a test case run by Murska using a range of processors between 8 and 64 for a measure of scalability of the code. In MareNostrum we extended this range of analysis up to 256 processors succefully. In Figure 1 we show the total timing, the timing took by the loop labeled 600 and the init particles part of the code (the last two are the most significant of the execution).

Table 1. Test: small. Number of processors: 8.

```
Estimated matrixMu =    42.00
Block Size =   36000
Tree Size  = 1512000
This run used     8925017863 random numbers.

WALL & CPU TIMES.
=================
            Total:  27719.131  27715.430
   Init Particles:   5679.830   5679.860
         Loop 600:  20749.967  20750.068
        Loop 1600:   1120.482   1120.516
```

```
  Matrix inversion:     45.852     45.834
Data transfer + AV:     16.281     13.756
Particle redistrib:      0.000      0.000
```

Table 2. Test: small. Number of processors: 16.

```
Estimated matrixMu =    42.00
Block Size =   18000
Tree Size  =  756000
This run used     8924990580 random numbers.

WALL & CPU TIMES.
=================
             Total:  14044.500  14043.730
    Init Particles:   2861.900   2861.900
          Loop 600:  10500.968  10500.953
         Loop 1600:    566.450    566.443
  Matrix inversion:     36.579     36.580
Data transfer + AV:     22.372     21.782
Particle redistrib:      0.000      0.000
```

Table 3. Test: small. Number of processors: 32.

```
Estimated matrixMu =    42.00
Block Size =    9000
Tree Size  =  378000
This run used     8924919319 random numbers.

WALL & CPU TIMES.
=================
             Total:   7104.780   7103.980
    Init Particles:   1444.850   1444.880
          Loop 600:   5294.459   5294.461
         Loop 1600:    286.251    286.209
  Matrix inversion:     20.850     20.881
Data transfer + AV:     26.679     26.089
Particle redistrib:      0.000      0.000
```

Table 4. Test: small. Number of processors: 64.

```
Estimated matrixMu =    42.00
Block Size =    4500
Tree Size  =  189000
This run used     8924765244 random numbers.

WALL & CPU TIMES.
=================
             Total:   3596.600   3595.950
    Init Particles:    708.060    708.070
          Loop 600:   2655.880   2655.819
         Loop 1600:    144.180    144.200
  Matrix inversion:     12.130     12.130
Data transfer + AV:     57.081     56.559
Particle redistrib:      0.000      0.000
```

Table 5. Test: small. Number of processors: 128.

```
Estimated matrixMu =    42.00
Block Size =    2250
Tree Size  =   94500
This run used     8924591955 random numbers.

WALL & CPU TIMES.
=================
             Total:   1844.300   1839.570
    Init Particles:    378.140    378.150
          Loop 600:   1334.940   1334.880
         Loop 1600:     73.080     73.100
  Matrix inversion:      6.590      6.570
Data transfer + AV:     36.440     33.850
Particle redistrib:      0.000      0.000
```

Table 6. Test: small. Number of processors: 256.

```
Estimated matrixMu =    42.00
Block Size =    1125
Tree Size  =   47250
This run used     8924276076 random numbers.

WALL & CPU TIMES.
=================
             Total:    934.990    931.670
    Init Particles:    188.980    188.980
          Loop 600:    663.930    663.890
         Loop 1600:     36.530     36.530
  Matrix inversion:      2.890      2.910
Data transfer + AV:     31.060     29.120
Particle redistrib:      0.000      0.000
```

5.2. Big test

The *big test* uses a test case run by Murska using a range of processors between 64 and 256 for a measure of scalability of the code. In MareNostrum we extended this range of analysis up to 1024 processors succefully. In Figure 2 we show the total timing, the timing took by the loop labeled 600 and the init particles part of the code (the last two are the most significant of the execution).

Table 7. Test: big. Number of processors: 64.

```
Estimated matrixMu =    42.00
Block Size =    8000
Tree Size  = 5376000

WALL & CPU TIMES.
=================
             Total:  20620.961  20620.820
    Init Particles:   2347.350   2347.340
          Loop 600:  17141.857  17141.664
         Loop 1600:    930.756    930.709
  Matrix inversion:     38.740     38.717
Data transfer + AV:     73.299     72.590
Particle redistrib:      0.000      0.000
```

Table 8. Test: big. Number of processors: 128.

```
Estimated matrixMu =    42.00
Block Size =    4000
```

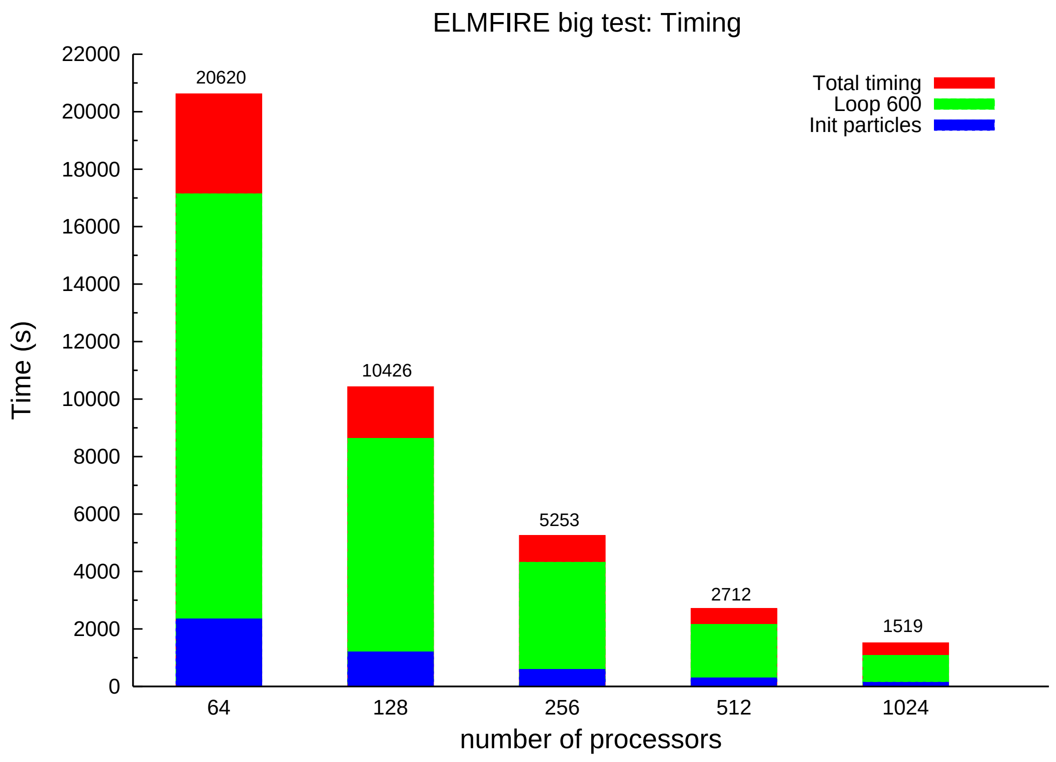

Figure 2. Elmfire: big test

```
Tree Size   = 2688000

WALL & CPU TIMES.
=================
             Total:  10426.430  10423.890
    Init Particles:   1198.040   1198.050
          Loop 600:   8634.258   8634.232
         Loop 1600:    471.479    471.509
  Matrix inversion:     15.851     15.842
Data transfer + AV:     54.577     53.896
Particle redistrib:      0.000      0.000
```

Table 9. Test: big. Number of processors: 256.

```
Estimated matrixMu =    42.00
Block Size =     2000
Tree Size   = 1344000
This run used    55434770311 random numbers.

WALL & CPU TIMES.
=================
             Total:   5253.930   5251.930
    Init Particles:    588.330    588.350
          Loop 600:   4319.360   4319.351
         Loop 1600:    233.561    233.559
  Matrix inversion:     12.150     12.110
```

```
Data transfer + AV:     67.671     67.061
Particle redistrib:      0.000      0.000
```

Table 10. Test: big. Number of processors: 512.

```
Estimated matrixMu =    42.00
Block Size =    1000
Tree Size  =  672000
This run used    55434056332 random numbers.

WALL & CPU TIMES.
=================
             Total:   2712.870   2705.610
    Init Particles:    295.000    295.000
          Loop 600:   2151.790   2151.630
         Loop 1600:    117.540    117.500
  Matrix inversion:     24.990     24.990
Data transfer + AV:     97.871     94.350
Particle redistrib:      0.000      0.000
```

Table 11. Test: big. Number of processors: 1024.

```
Estimated matrixMu =    42.00
Block Size =     500
Tree Size  =  336000
This run used    55432750995 random numbers.

WALL & CPU TIMES.
=================
             Total:   1519.050   1517.120
    Init Particles:    147.640    147.640
          Loop 600:   1074.840   1074.820
         Loop 1600:     59.460     59.460
  Matrix inversion:     67.270     67.260
Data transfer + AV:    151.640    150.440
Particle redistrib:      0.000      0.000
```

5.3. Paraver analysis

Paraver[4] is used to take a photo of the execution process. It collect all about hardware counters and communication events between threads.

We show three diferent pictures of a 8 thread execution. They show that it exists few communications between threads and all reach the group communication points together. The application is well parallelized. Figure 3 shows the amount of time where the application is running (blue color) and waiting for communication (orange). Figure 4 show the instuctions per cycle between events: deeper the color greater IPC (blue is 1.50, dark green is 0.50). Figure 5 shows MFLOPS (blue is 700 and dark green is 300)

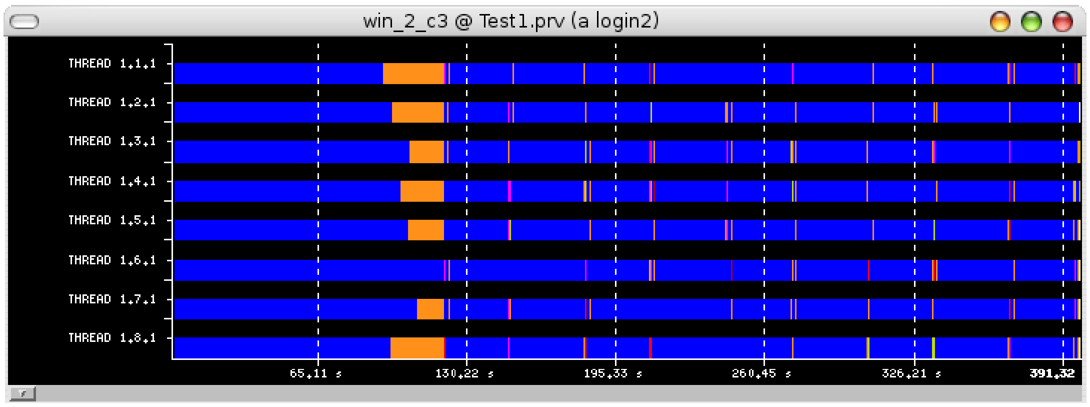

Figure 3. Elmfire: status of executions

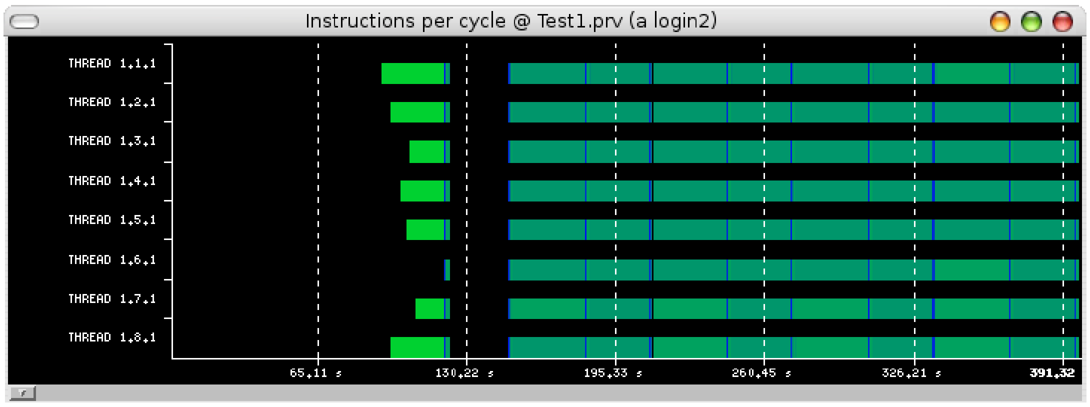

Figure 4. Elmfire: instuctions per cycle

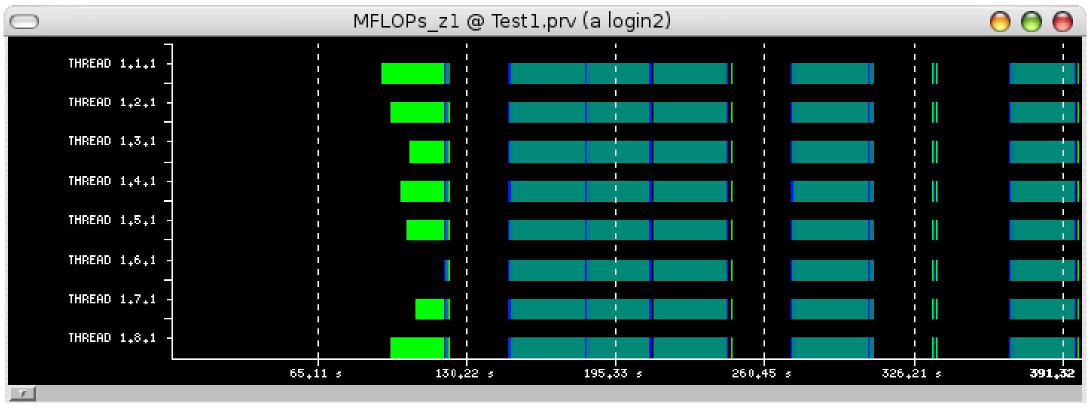

Figure 5. Elmfire: MFLOPS

5.4. Discussion

In comparison with the data obtained from other performance analysis (in a HP CP4000) the code ELMFIRE in our architecture shows a lower performance. The CPU time consumed by our executions is aproximately twice that is on the mencioned architecture. The segmentation fault errors produced by the code when we try to use profiling FLAGS make impossible to study more deeply the reasons of this lower behavior.

However the analysis with paraver [1], shows a good behavior parallelism and thread balance which produces good scalability showed in figures 1, 2. Following the internal profile provided by the code, is possible to reach the CPU time consumed by the diferents parts of the code. These results are summarized in tables 1-11. The more time consumed part correspond to the called loop 600, a general loop which run over all the particles of the simulation. Any improvement in the code needs starts with a carefully study of this part of the code.

6. Conclusions

Proves realized with ELMFIRE shows a slower performance than the data obtained from the knower analysis in MURSKA cluster. This situation cant be studied still now in MN because the mentioned problems in the compilation of the code using profiling FLAGS. Besides that, scalable results were obtained up 1024 nodes, extending the range of proves realized in the past in MURSKA cluster (up 256 processors).

In order to improve the analysis, an author's revision of the source code, similar to the realized to possibility our study is needed.

References

1. S.J. Janhunen, F. Ogando, J.A. Heikkinen, T.P. Kiviniemi, and S.Leerink. Collisional dynamics of er in turbulent plasmas in toroidal geometry. *Nuclear Fusion*, 47:875–879, 2007.
2. T. Kiviniemi. *Plasma Phys. Contr. Fusion.* Number 43. Dover Publications, 2001.
3. Tommi Kutilainen Kaisa Riikil. Turbulence in fusion plasma. *CSCnews*, pages http://www.deisa.org/applications/projects2006–2007/fullfgk.php, 2007.
4. Vincent Pillet, Jesús Labarta, Toni Cortés, and Sergi Girona. Paraver: A tool to visualize and analyze parallel code. *Transputer and occam Developments*, pages 17–32, April 1995.
`http://www.bsc.es/plantillaA.php?cat_id=485`.