GENE. Installation guide and performance analysis

Xavier Sáez
Alejandro Soba

Barcelona Supercomputing Center

Technical Report TR/CASE-08-3

Jun 2008

GENE. Installation guide and performance analysis

Xavier Sáez[1] **and Alejandro Soba**[2]

[1] *Computer Application and Science Engineering, Barcelona Supercomputing Center, Edifici Nexus, Campus Nord UPC, c/ Gran Capità, 2-4, 08034 Barcelona, Spain.*
E-mail: xavier.saez@bsc.es, *Web page:* http://www.bsc.es

[2] *Computer Application and Science Engineering, Barcelona Supercomputing Center, Edifici Nexus, Campus Nord UPC, c/ Gran Capità, 2-4, 08034 Barcelona, Spain.*
E-mail: alejandro.soba@bsc.es, *Web page:* http://www.bsc.es

Abstract. In this report we present the necessary steps to install and run the code GENE 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. GENE, gyrokinetic, installation guide, execution guide, profiling

1. Introduction

The gene code (this acronym stands for gyrokinetic electromagnetic numerical experiment) is a tool for studying problem of plasma micro turbulence from a fully kinetic point of view. To this aim, the nonlinear gyrokinetic equations for a magnetized electron-ion plasma are solved on a fixed grid in five-dimensional phase space, employing finite-difference and pseudo-spectral techniques. All relevant electromagnetic effects are taken into account. Each simulation requires of the order of 10^5 grid points and up to 10^9 time steps, challenging even the most powerful present-day computers. GENE runs efficiently on multiple massively parallel platforms, achieving, e.g., 280 MFlops on the Hitachi SR-8000 at Leibniz Computing Center at Munich. To reach this level of performance, it was necessary to adapt the implementation strategy to each individual machine architecture.

GENE, is a so-called continuum (or Vlasov) code. All differential operators in phase space are discretized via a combination of spectral and higher-order finite difference methods.

For maximum efficiency, GENE uses a coordinate system which is aligned to the equilibrium magnetic field and a reduced (flux-tube) simulation domain. This reduces the computational effort by 2-3 orders of magnitude. Moreover, it can deal with arbitrary toroidal geometry (tokamaks or stellarators) and retains full ion/electron dynamics as well as magnetic field fluctuations. At present, GENE is the only plasma turbulence code in Europe with such capabilities.

For tests on higher processor numbers, the BlueGene/L system at IBMWatson Research Center was used. Here a functionally and algorithmically improved code version, GENE v11+, was used, with the same parallelization scheme as in GENE v11. Strong scaling measurements showed close to linear speedup up to 4k processors; on 8k processors, a parallel efficiency of 73% was achieved, revealing some degradation. For further scalability improvements well beyond 8k processors, the second velocity dimension was parallelized in addition [1, 2].

2. Source Code

Gene is written in FORTRAN 90/95 and it consists of approximately 31 files.
The application also includes an *IDL based tool* for data visualization and analysis.

The current used release is 1.2.

3. Libraries

Gene requires the following software in order to compile:

- a FORTRAN 90/95 compiler
- the MPI message passing interface
- an FFT routine (ESSL, MKL or FFTW)
- the BLAS/LAPACK library (also contained in ESSL and MKL)

Additional software packages that can extend its functionality are:

- the PETSC/SLEPC package for eigenvalue computations
- the OpenMP for shared memory multiprocessing

4. Installation

Set the `GENE_HOME` environment variable to the working directory and unzip the zipped source file.

Modify the `makefile` in the main directory to allow the automatic selection of your machine. The following lines are added for the selection of Marenostrum:

```
ifeq ($(MACHTYPE),ppc64-suse-linux)
 MACHINE = Marenostrum
endif
```

Furthermore, there is a makefile for each machine-specific architecture with all the information needed to compile GENE in the `src/svprob` subdirectory. Make sure there is one for your system, otherwise create it from the `new_machine` makefile.

This new makefile will configure the following variables about your system:

- the FFT routine (`FFTLIB`) and its library: *mkl, fftw or essl*
- the precision (`PRECISION`): *double or single*
- the Open MP support (`OPENMP`) and its flags
- the use of the PETSC and the SLEPC packages (`SLEPC`)
- the use of the NAG compilers (`WITHNAG`)
- the FORTRAN compiler (`FC`) and its flags (`FFLAGS`)
- the C compiler (`CC`) and its flags (`CFLAGS`)
- the linker (`LD`) and its flags (`LDFLAGS`)
- the use of the Simple Performance Toolkit (`PERFLIB`)

So, for Marenostrum, we have created a specific `Marenostrum.mk` makefile into a new subdirectory, `src/svprob/Marenostrum`:


```
cd $GENE_HOME/src/svprob
mkdir Marenostrum
cp new_machine/new_machine.mk Marenostrum/Marenostrum.mk
```

Next, the main modifications performed in the `Marenostrum.mk` file:

```
########################################################################
### SWITCHES                                                         ###
########################################################################

FFTLIB = essl
PRECISION = double
OPENMP = yes

MPI_IO= no
SLEPC= no
WITHNAG = no
PERFLIB = none

########################################################################
#   COMPULSORY LIBRARIES                                               #
########################################################################

ifeq ($(FFTLIB),essl)
 INCPATHS +=
 LIBS += -L/usr/lib64 -lessl
 PREPROC += -WF,-DWITHESSL
endif

########################################################################
### COMPILER & COMPILER FLAGS                                        ###
########################################################################

CC = xlc
FC = mpif90
FFLAGS = -q64 -O3 -qstrict -qtune=ppc970 -qarch=ppc970 \
-qcache=auto -qmoddir=$(OBJDIR) -qflag=I:I
PREPROC += -WF,-DAIX

LD =$(FC)
LDFLAGS = -q64

########################################################################
# ADDITIONAL COMPILER FLAGS (set via SWITCH in header)                 #
########################################################################

ifeq ($(OPENMP),yes)
 FFLAGS  += -qsmp=omp
 LDFLAGS += -qsmp=omp
 PREPROC += -WF,-DWITHOMP
endif

ifeq ($(PRECISION),double)
 FFLAGS +=-qrealsize=8
 PREPROC +=-WF,-DDOUBLE_PREC
endif
```


Finally, a last modification has performed in the specific makefile. The **run** target establishes the lines for running GENE in your system. In our case, the execution is submitted to the Marenostrum batch queue (File 3) and the `mpi` call is replaced with a `srun` call:

```
#######################################################################
### Linking                                                         ###
#######################################################################

run:    $(RUNDIR)/$(EXEC)
    ulimit -s unlimited;\
    cd $(RUNDIR);\
    OMP_NUM_THREADS=$(OMP_NUM_THREADS)\
    srun ./$(EXEC)
```

Once the makefile is ready, go to the `testsuite` directory and type the following command to build GENE's executable:

```
cd $GENE_HOME/testsuite
gmake -f ../makefile
```

After, there are some test problems to allow you to check the installation, you can either execute interactively using `testsuite` mpirun or submit it to a batch queue (File 1).

```
#!/bin/bash
# @ job_name         = gene.x
# @ class            = bsc_cs
# @ initialdir       = .
# @ output           = Gene%j.out
# @ error            = Gene%j.err
# @ total_tasks      = 9
# @ cpus_per_task    = 4
# @ wall_clock_limit = 01:00:00

date
./testsuite -w -outdir $GENE_HOME/testsuite
date
```

File 1. `test.cmd`: script to submit testsuite to Marenostrum batch queue

Note: If your system has installed MPI libraries which are non-thread safe, some tests can fail. In this situation, it is enough to define the precompiler switch `ia64` in the machine-specific makefile to solve the problem and to pass the tests. This flag prevents a call to OpenMP in the `comm.F90` file. For example:

```
ifeq ($(OPENMP),yes)
 FFLAGS += -qsmp=omp
 LDFLAGS += -qsmp=omp
 PREPROC += -WF,-DWITHOMP -WF,-Dia64
endif
```


5. Execution

Firstly, you have to create a folder in which you define the parameters for the problem. The `newprob` script will generate a new problem folder:

```
cd $GENE_HOME
./newprob
```

The created folder may be renamed, but the directory structure must not be changed.

The next step is to recompile the GENE executable, which will be installed into a subdirectory in the problem folder:

```
cd prob01
gmake -f ../makefile
```

Now, adapt the `parameters` file with the physical and numerical parameters of the problem. Also, specify the input and output data. This file does not have to be know until the run time, so it can be changed between executions. You can find an in-depth explanation of all the fields in the GENE user manual.

For instance, in the file 2, there is an example of `parameters` file.

```
&parallelization
n_procs_s =      2  !should be a divisor of n_spec (maximum: nspec)
n_procs_v =      1  !should be a divisor of nv0 (maximum: 0.5*nv0)
n_procs_w =      8  !should be a divisor of nw0 (maximum: nw0)
n_procs_y =      1  !should be a divisor of nky0 (keep low)
n_procs_z =      8  !should be a divisor of nz0 (maximum: 0.5*nz0)
/
&box
n_spec  = 2
nx0 =   64
nky0 =  32
nz0 =   128
nv0 =   64
nw0 =   32
lx =      125.628
kymin =       0.05
lv =          3.0
lw =          9.0
/
&in_out
diagdir = '/home/bsc21/bsc21331/EUFORIA/GENE/release-1.2/prob01'
chptdir = '/home/bsc21/bsc21331/EUFORIA/GENE/release-1.2/prob01'
read_checkpoint = .f.
write_checkpoint = .f.
istep_field =        0
istep_mom =          0
istep_nrg =          0
istep_vsp =          0
istep_schpt =        0
/
&global
nonlinear = .t.
calc_dt     = .f.
ntimesteps = 1 ! take more if time per time step is fluctuating
```



```
timelim =     64500
dt_max = 0.0385
courant = 0.4
beta =      0.001
hyp_z =     0.25
hyp_v =     0.2
magn_geometry = 's_alpha'
shat =      0.796
q0 =        1.4
curv =      2.0
trpeps = 0.18
/
&species
name =    'ions'
omn =     2.22
omt =     6.92
mass =      1.0
charge =      1
temp =      1.0
dens =      1.0
/
&species
name =    'electrons'
omn =     2.22
omt =     6.92
mass =    0.0025
charge =     -1
temp =      1.0
dens =      1.0
/
```

File 2. `parameters`: a parameters file

Once `parameters` file is edited, make sure that input files and output directories exist, and start GENE interactively or submit it to a batch queue. In our case, it has to be submitted to a Marenostrum batch queue (File 3).

```
#!/bin/bash
# @ job_name         = gene.x
# @ class            = bsc_cs
# @ initialdir       = .
# @ output           = Gene%j.out
# @ error            = Gene%j.err
# @ total_tasks      = 128
# @ cpus_per_task    = 1
# @ wall_clock_limit = 00:02:00

export OMP_NUM_THREADS=4
date
gmake -f ../makefile run
date
```

File 3. `run.cmd`: script to Marenostrum batch queue

Note that the number of MPI processes is given by $nprocs_s \cdot nprocs_w \cdot nprocs_v \cdot nprocs_y \cdot nprocs_z$. In addition, for shared memory architectures, the number of OMP processes can be set independently by setting the `OMP_NUM_THREADS` environment variable and the total number of processors needed is then $N_{MPI} \cdot OMP\_NUM\_THREADS$.

6. Profiling

The experiments have been run on IBM Power PC 970MP processors at 2.3 GHz. There are 2 processors per chip and two chips per node. Each of these nodes have 8 Gb of shared memory, so each processor has 2Gb of memory.

All the experiments are based on the physical and numerical parameters showed in File 2 with the following adaptations to run in a different number of processors:

Tests			Parameters				
MPI tasks	**thrds/task**	**procs**	**n_procs_s**	**n_procs_v**	**n_procs_w**	**n_procs_y**	**n_procs_z**
64	1	64	2	1	8	1	4
128	1	128	2	1	8	1	8
256	1	256	2	2	8	1	8
512	1	512	2	1	16	1	16
1024	1	1024	2	2	16	1	16
2048	1	2048	2	4	16	1	16
16	4	64	2	2	2	1	2
32	4	128	2	1	4	1	4
64	4	256	2	1	8	1	4
128	4	512	2	1	8	1	8
256	4	1024	2	2	8	1	8
512	4	2048	2	1	16	1	16

Figure 1 compares the two code versions: the **pure MPI** and the **hybrid** (MPI + OpenMP). The left graph shows *the performance* of both cases, we can see that the pure MPI case is faster than the hybrid case. In the right graph, *the speedup* is showed and we can see that the hybrid code has a better scalability than the pure MPI code.

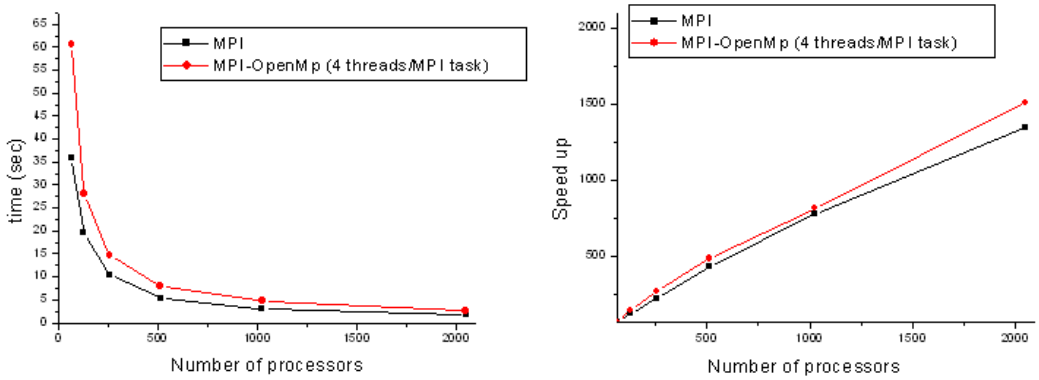

Figure 1. Performance and speedup

As this behaviour looks unusual because the hybrid version should adapt better to multiple SMP nodes architecture (like Marenostrum), we decided to analyzed an execution in detail, the case of 128 MPI tasks vs 128 threads (32 MPI task and 4 threads per task).

The following table shows the distribution of the execution time:

State	128 MPI tsks Time(s)	128 MPI tsks Time(%)	32 MPI tsks * 4 thrs Time(s)	32 MPI tsks * 4 thrs Time(%)
Running	3183.24	93.29	3143.43	82.00
Group Communication	75.17	2.20	38.88	1.01
Send/Receive	153.89	4.51	75.43	1.97
Sched & Fork/Join	0.00	0.00	5.92	0.15
Idle	0.00	0.00	569.99	14.87
Total	3412.30	100.00	3833.65	100.00

Although the hybrid version is 10% slower than MPI pure version, it spends less time in the running part. Moreover, the hybrid version is more efficient in the communications because the threads use shared memory. Then, it is possible to recognize that the performance loss is focused in the idle time (569.99 seconds).

Due to this, the hybrid version does not reach the best performance because some processors have no work during periods. To study this possibility, we tracedand visualized with PARAVER tool [3] some executions with OpenMP, in order to show graphically the thread scheduling (Figure 2).

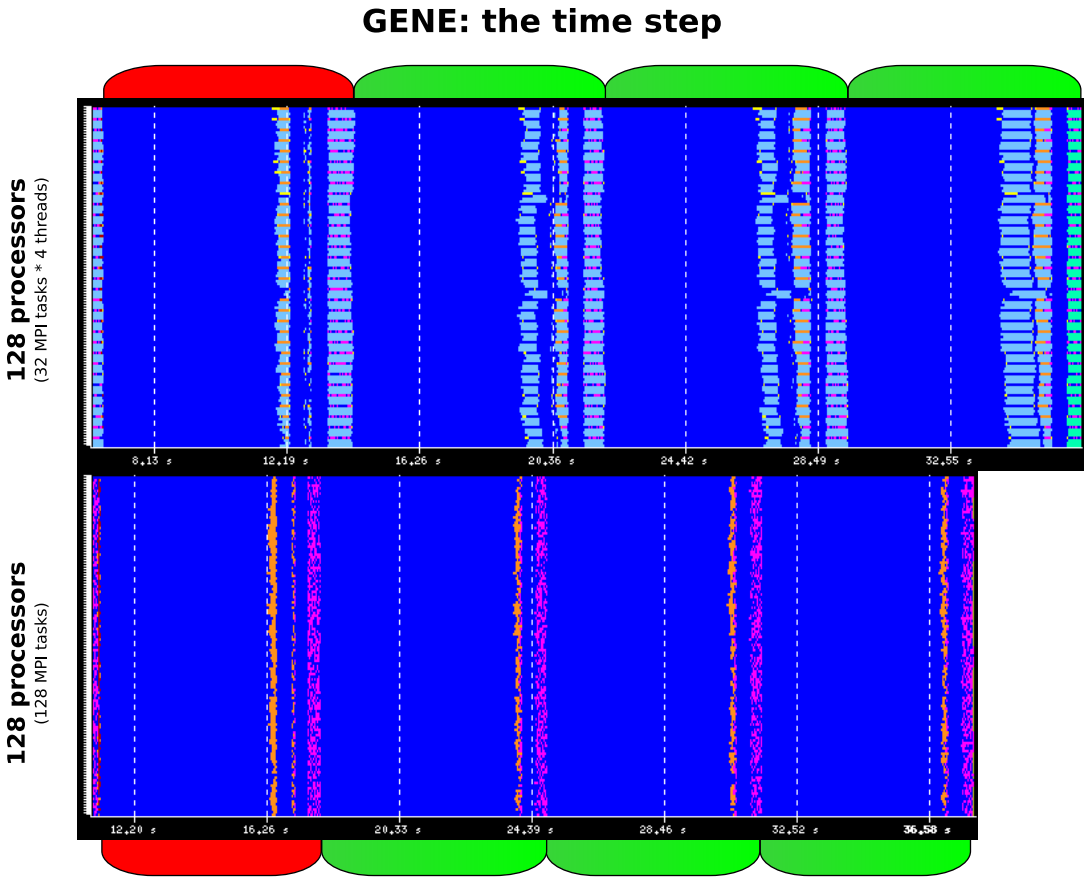

Figure 2. Trace of one time step: the upper part is the hybrid version and the lower part is the MPI pure version.

To read the figure 2, the meaning of the colours is:

- Dark blue means CPU active
- Light blue means CPU idle
- Orange means MPI collective communications
- Pink means send/receive functions.

Also, the components of the time step have been identified in the figure:

- The red box is the first Runge-Kutta step where (in nonlinear operation) the maximum ExB velocities are calculated and communicated in order to adapt the time step "dt" (second collective group),
- The green boxes are the remaining three Runge-Kutta steps which do not need to calculate the ExB velocities again.

The analysis of the figure 2 shows that the hybrid version has less pink and orange regions, so the communication phase has decreased. In contrast, an important light blue region has appeared in 3 of each 4 threads. So there are regions that are not completely parallelized. As a result, the execution of one time step takes more time in hybrid version.

7. Conclusion

The realized analysis shows that GENE haves good scalability up to 512 procesors, going down to range of processors form 512 to 2024. The called hybrid version shows better scalability besides its lower performance. The PARAVER picture shows that the iddle time in the working scheduling of the threads is responsable of this unusual behavior in the hybrid version.

It would be interesting to study the OpenMP implementation to improve the thread scheduling with OpenMP and decrease the idle time of the threads.

References

1. Gene Development Team. *The Gyrokinetic Plasma Turbulence Code. Gene: User Manual*, January 2008. `http://www.ipp.mpg.de/~fsj/`.
2. Hermann Lederer, Reinhard Tisma, Roman Hatzky, Alberto Bottino, and Frank Jenko. Application enabling in deisa: Petascaling of plasma turbulence codes. In C. Bischof, M. Bucker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, and F. Peters, editors, *Parallel Computing: Architectures, Algorithms and Applications*, volume 38, pages 713–720, Julich, September 2007. John von Neumann Institute for Computing. NIC-Series.
3. 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`.

