



Rapid parallelization of the drift-diffusion model for semiconductor devices

G. Gazzaniga¹, P. Lanucara², P. Pietra¹, S. Roviada¹, G. Sacchi¹

¹ *IMATI - CNR - v. Ferrata,1 - Pavia*

² *CASPUR c/o Università "La Sapienza" - Roma*

lanucara@caspur.it rovida@ian.pv.cnr.it

Goals



- Investigate the various strategies to improve the performance of a sequential numerical FORTRAN code for modelling semiconductor devices:
 1. vectorization features;
 2. automatic parallelization;
 3. explicit parallelization through compiler directives.
- Avoid the expensive task of reengineering the original code maintaining portability.
- Evaluate:
 1. man-hours spent in the parallelization;
 2. the effort required to get a suitable level of knowledge of the code.

Scaled Model Equations



- Poisson equation for the electrostatic potential ψ

$$-\lambda^2 \Delta \psi = p - n + C, \quad \text{in } \Omega$$

C doping profile, λ Debye length (small parameter)

- continuity equations for the charge density p and n

$$\text{div} \underline{J}_p = 0, \quad \underline{J}_p = -(\nabla p + p \nabla \psi), \quad \text{in } \Omega$$

$$\text{div} \underline{J}_n = 0, \quad \underline{J}_n = \nabla n - n \nabla \psi, \quad \text{in } \Omega$$

$\underline{J}_p, \underline{J}_n$ density current vectors

- Dirichlet - Neumann boundary conditions.

Iterative Scheme -1



begin continuation in λ

begin Gummel iterations

$$\text{Step 1} \quad -\lambda^2 \Delta \delta\psi + (p^k + n^k) \delta\psi = \lambda^2 \Delta \psi^k + p^k - n^k + C$$

$$\psi^{k+1} = \psi^k + \delta\psi$$

$$\text{Step 2} \quad \text{div} \underline{J}_p^{k+1} = 0, \quad \underline{J}_p^{k+1} = -(\nabla p^{k+1} + p^{k+1} \nabla \psi^{k+1})$$

$$\text{Step 3} \quad \text{div} \underline{J}_n^{k+1} = 0, \quad \underline{J}_n^{k+1} = \nabla n^{k+1} - n^{k+1} \nabla \psi^{k+1}$$

end Gummel iterations

end continuation in λ

Iterative Scheme - 2



- *Step 1* is performed by means P_1 non-conforming finite element method;
- *Step 2* and *Step 3* are solved via an exponential fitting mixed finite element scheme;
- the sparse linear systems coming from the discretization of the equations for ψ , p , n are solved using *GMRES* method;
- the iterative scheme has an intrinsic mathematical 2-way parallelism due to the independency of p and n equations in *Step 2* and *Step 3*.

Iterative Scheme - 3



continuation in λ

For small Debye lengths the Gummel iterations result very sensitive to the initial guess of p , n .

In order to guarantee the stability, a continuation in the parameter λ is performed:

- start with $\lambda \simeq 10^{-1}$ and any initial guess for p , n ;
- decrease slowly λ using as initial values of p , n the solution just computed for the previous value of λ ;
- stop if the physical Debye length has been reached.

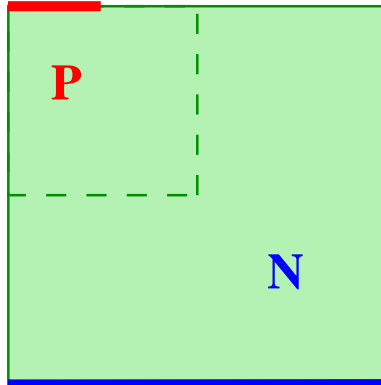
Sparse Linear Algebra Package



SLAP ver 2.0

- It contains *core* routines for the iterative solution of symmetric and non-symmetric positive definite and positive semi-definite linear systems.
- *BLAS* package is required.
- It derives from a set of routines written by **Anne Greenbaum**, *Routines for Solving Large Sparse Linear Systems*, Lawrence Livermore Nat. Laboratory, Livermore Computing Center, January 1986.
- DSLUGM implements *GMRES* method, with incomplete *LU* factorization for preconditioning, to solve possibly non-symmetric linear systems.

Test Case - pn-Diode



Physical parameters

q	elementary charge	10^{-19}	$A \text{ sec}$
ε	permittivity constant	10^{-12}	$A \text{ sec } V^{-1} \text{ cm}^{-1}$
U_T	thermal potential	0.025	V
\bar{c}	doping scale factor	10^{+16}	cm^{-3}
L	length of semiconductor device	10^{-3}	cm

$$\lambda^2 = \frac{\varepsilon U_T}{q \bar{c} L^2} = 2.5 \cdot 10^{-5} \quad \text{squared Debye length}$$



SUN Enterprise 4500

- 4 cpu UltraSparc II (sparcv9 + vis) 400MHz
- 2 GBytes RAM
- 16Kbytes primary instruction cache
- 16Kbytes primary data cache
- 4Mbytes second level cache

- SLAP, Sunperf Libraries
- OpenMP native

IMATI - CNR, Pavia <http://www.ian.pv.cnr.it>



IBM SP3 node

- 16 cpu Power3 375MHz
- 16 GBytes RAM
- 8Kbytes double words primary cache
- 2Mbytes double words second level cache

- SLAP, ESSL Libraries
- OpenMP native, Guide

CASPUR, Roma <http://www.caspur.it>



HP AlphaServer ES45

- 4 cpu Alpha 1GHz
- 12 GBytes RAM
- 64 Kbytes primary instructions cache
- 64 Kbytes primary data cache
- 8 Mbytes second level cache

- SLAP, CXML Libraries
- OpenMP native

Hewlett Packard , Milano <http://www.hp.com/it>

Profiling



- The times in *sec* of *Step 1*, *Step 2* and *Step 3* have been measured by means of an intrusive profiling, using Pmapi Library.

IBM SP3				
<i>dof</i>	<i>Step 1 time</i>	<i>Step 2 time</i>	<i>Step 3 time</i>	<i>execution time</i>
1354	2.3	13.7	15.7	33.4
5336	10.9	117.1	174.8	307.5

- The expected *speed-up* is about $1.6 \div 1.8$.

How to improve the performance



- **automatic vectorization:** enables calls to the vector math routines (supported by SUN architecture)
- **automatic parallelization:** enables automatic loop parallelization

no good results are obtained !

- **explicit parallelization:** enables parallelization of loops or regions explicitly marked with directives

OpenMP seems to be a good choice to guarantee portability.



begin continuation in λ

begin Gummel iteration

Step 1 - solve equation in ψ

Step 2 - solve equation in p

Step 3 - solve equation in n

independent steps

end Gummel iteration

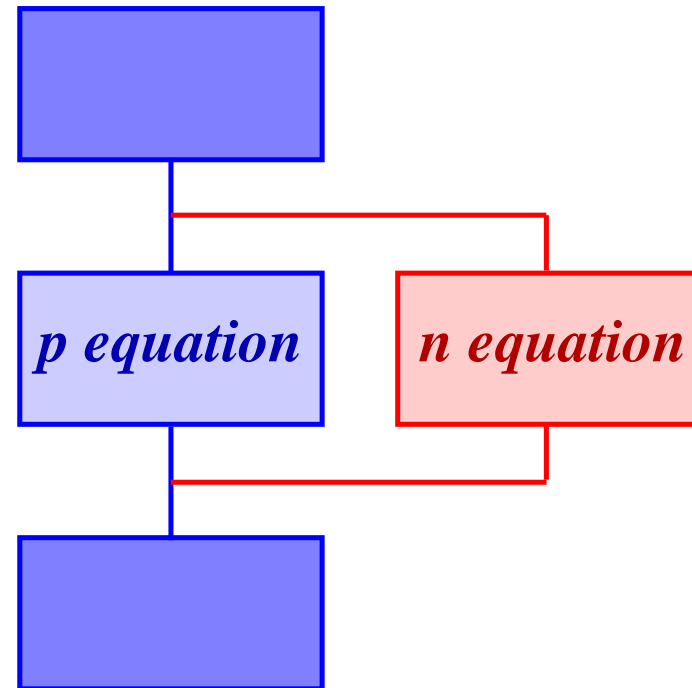
end continuation in λ

- define *parallel sections* corresponding to *Step 2* and *Step 3*
- define the *scope* of the variables
- implicit synchronization at the end of the *parallel sections*

parallel sections - 2



```
!----- begin gummel iteration -----  
    < psi equation >  
!$omp parallel sections  
!$omp& default (none)  
!$omp& shared . . .  
!$omp& private . . .  
!$omp section  
    < p equation >  
!$omp section  
    < n equation >  
!$omp end parallel sections  
    . . .  
!----- end gummel iteration -----
```



OpenMP Results



SUN Enterprise 4500						
<i>dof</i>	<i>1 thread</i>			<i>2 threads</i>		
	<i>execution time</i>	<i>Step 2-3 cumulative</i>	<i>Step 2-3 iteration</i>	<i>execution time</i>	<i>Step 2-3 cumulative</i>	<i>Step 2-3 iteration</i>
1354	100.8	89.3	0.37	59.6	48.2	0.20
5336	904.8	854.7	4.10	629.7	579.6	2.77

IBM SP3						
<i>dof</i>	<i>1 thread</i>			<i>2 threads</i>		
	<i>execution time</i>	<i>Step 2-3 cumulative</i>	<i>Step 2-3 iteration</i>	<i>execution time</i>	<i>Step 2-3 cumulative</i>	<i>Step 2-3 iteration</i>
1354	33.0	29.4	0.12	19.5	16.0	0.07
5336	302.7	287.2	1.37	187.6	172.2	0.82

HP ES45						
<i>dof</i>	<i>1 thread</i>			<i>2 threads</i>		
	<i>execution time</i>	<i>Step 2-3 cumulative</i>	<i>Step 2-3 iteration</i>	<i>execution time</i>	<i>Step 2-3 cumulative</i>	<i>Step 2-3 iteration</i>
1354	21.9	18.6	0.08	13.3	10.0	0.04
5336	207.4	193.5	0.93	129.5	115.5	0.55

OpenMP Conclusions



- Good performance at a rather low programming cost.
- Deep knowledge of the logical structure of the algorithm is not required.
- Issue of portability is guaranteed.

- Further improvement: multilevel parallelism.

A different parallelization approach



- *Step 1*, *Step 2* and *Step 3* require the solution of sparse linear systems, carried out by means of the SLAP routine DSLUGM, which implements a preconditioned *GMRES* solver.
- Most of the time is just spent in the solver.

IBM SP3						
	<i>dof</i> = 1354			<i>dof</i> = 5336		
	<i>Step 1</i>	<i>Step 2</i>	<i>Step 3</i>	<i>Step 1</i>	<i>Step 2</i>	<i>Step 3</i>
<i>total</i>	2.3	13.7	15.7	10.9	117.1	174.8
DSLUGM	1.7	12.2	14.3	8.7	111.7	169.4

- Hint: use a multithreaded version of *GMRES*.

NAG Fortran SMP Library - 1



- Why NAG Library ?
 - robustness
 - reliability
 - accuracy
- NAG SMP Library:
 - developed on OpenMP standard improving portability;
 - available on a broad range of platforms including Windows NT, SGI 6, Sun Solaris, Compaq Alpha ...

<http://www.nag.co.uk/numeric/fl/FSdescription.asp>

http://www..nag.co.uk/numeric/FL/manual19/html/genint/news_fs20.html



- From the point of view of performance and *scalability* the SMP-NAG Library consists in three sets of routines:
 1. *Tuned routines*: those that have been specially tuned and which therefore exhibit near optimal performance and scalability;
 2. *Enhanced routines*: those that call one or more of the previous tuned routines;
 3. *Additional routines*: some of these perform BLAS operations as part of their algorithm and so benefit from the performance and scalability of the particular vendor BLAS.
- The routines used
 - F11DAF implements incomplete *LU* factorization
 - F11DCF implements *RGMRES*, *CGS*, *BiCGSTAB* methods, using the preconditioner computed by F11DAFbelong just to the set 3.

NAG Results



- The experiments have been carried out for the test case with $dof = 5336$, using NAG SMP Library release 2.
- The OpenMP *parallel sections* have been disabled, parallelism is achieved only by means of the multithreaded version of the solver.

IBM SP3				
<i>number of threads</i>	<i>execution time</i>	<i>Step 1 time</i>	<i>Step 2 time</i>	<i>Step 3 time</i>
1	345	24.8	132.2	183.4
2	279	21.8	107.4	145.6

- Maximum gain obtained for the execution time about 20% with two threads.
- Increasing the number of threads a poor performance is obtained.

Further Investigations



- Perform an explicit parallelization of the solver, manually inserting directives in the SLAP source code.
- Combine OpenMP *parallel sections* with a multithreaded version of the solver.

The OpenMP grammar allows for *nested parallelism* but leaves to the runtime system any decision about how to split processors between the levels of parallelism. In theory, this should be done automatically, but the criteria by which that should happen are unclear.

- Experiment an *hybrid* MPI-OpenMP implementation of the code, using MPI to compute in parallel *Step 2* and *Step 3* and OpenMP for the parallelization of the solver within each step.

Switch to MPI on the higher level is more tedious than using OpenMP in this case and requires a reengineering of the code, but it should be feasible in a limited amount of time.

Acknowledgments



The stimulating discussions with Dieter an Mey and his useful suggestions are acknowledged.

We wish to thank A. Galli of the Hewlett-Packard Company (Milano, Italy) for his assistance and for allowing us to use the HP computing facilities.