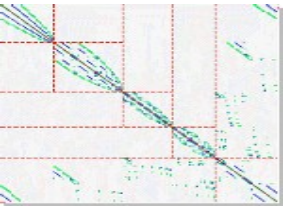
Preconditioned IDR solution methods in scientific and industrial applications

Alexander Fedoseyev,
Ultra Quantum Inc, Huntsville, Alabama, USA



The work sponsored in part by **NASA** program

Precond24, Atlanta, Georgia, USA, June 10, 2024

Outline

1. Intro: Need for linear solver in scientific and industrial applications
2. CNSPACK linear solver history
3. Current Status and Goals
4. Parallelization approach for multi-core processor architecture
5. OpenMP implementation – algorithm and results
6. Conclusions

Goal – Robust Parallel Linear Solver for Scientific and Industrial Applications

Applications of CNSPACK solver:

3D Semiconductor transport

Semiconductor device/circuits simulations

Complex Cooling Systems

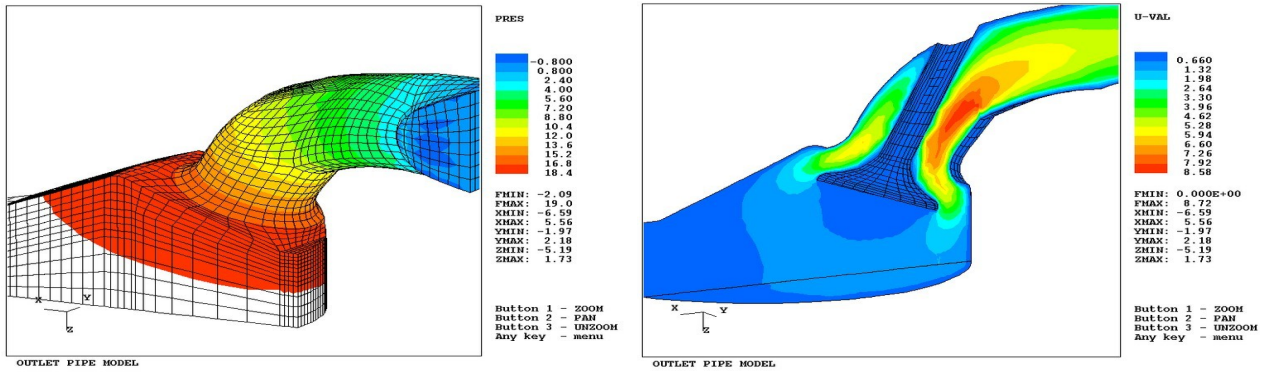
Turbulent Flows

Hypersonic Flows

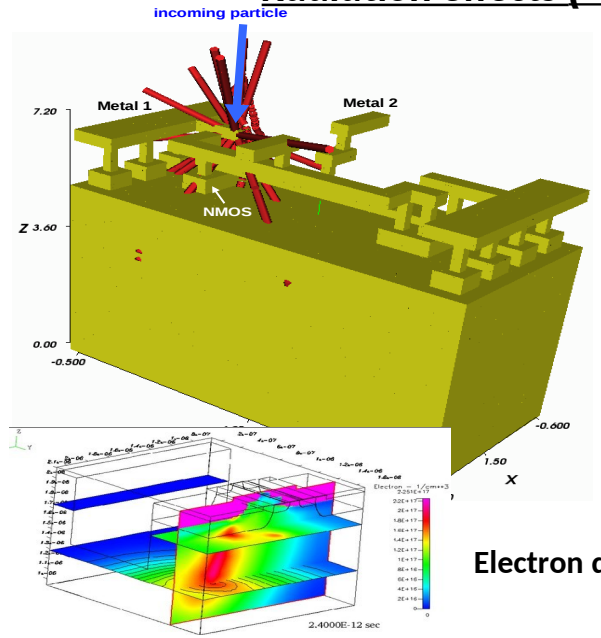
High Altitude Flows

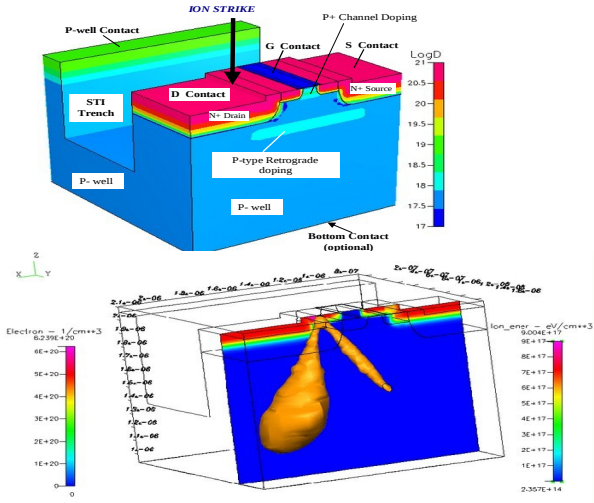
Industrial Applications with CNSPACK Linear Solver

3D Internal Flows in Cylinder-Piston-Valve System



Radiation effects (ion strike) in NMOSFET



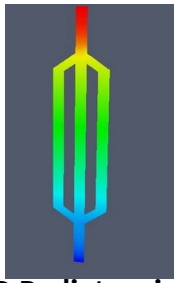


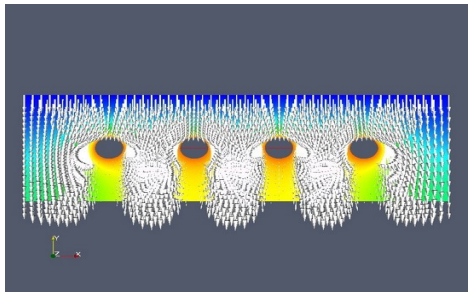
3D simulations performed using Femina-3D multiphysics solver with CNSPACK

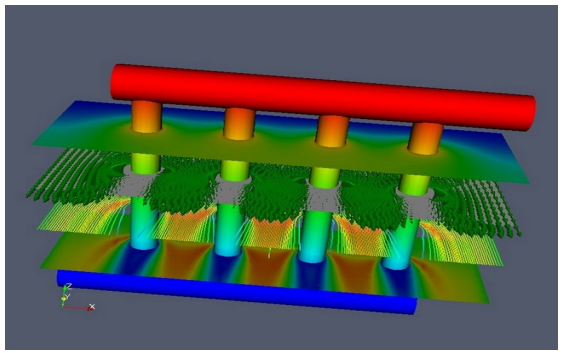
Electron density

Isosurface of ion energy deposition

1D-2D-3D models for radiator



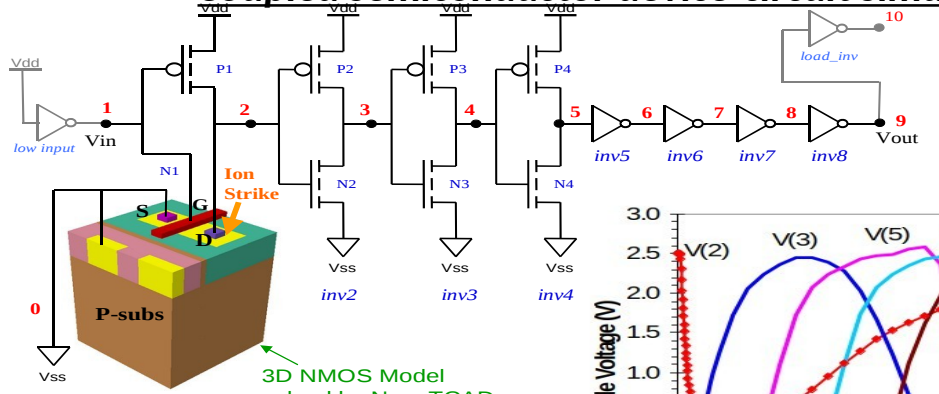


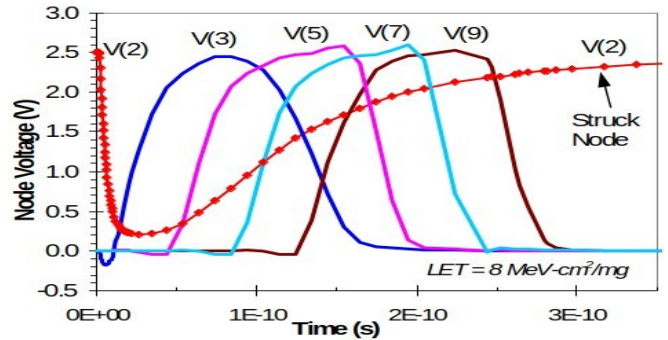


3D Radiator simulation:

- Inlet tube with hot coolant at temperature Tin;
- Temperature distribution in the air flow ;
- Averaged velocity field (by arrows) in the air flow;
- Stream tracers in the air flow, colored by flow velocity magnitude;

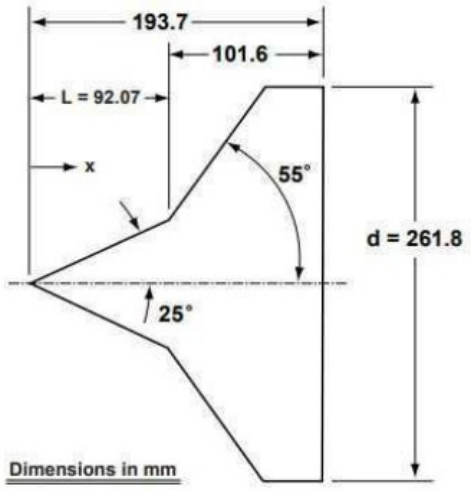
Coupled semiconductor device-circuit simulations

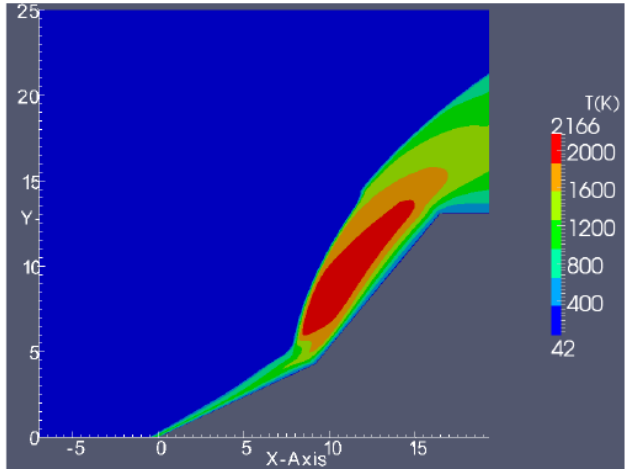




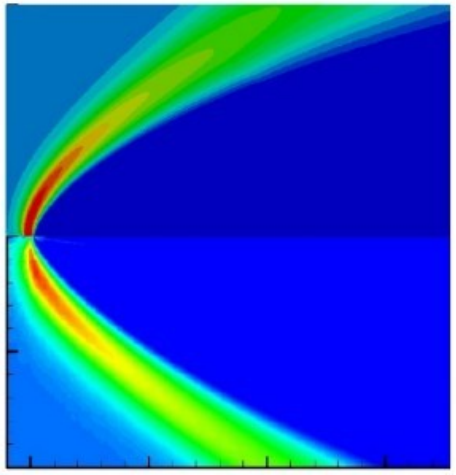
Industrial Applications with CNSPACK Linear Solver for Hypersonic Flows

FEMINA/3D flow solver with CNSPACK (hypersonic flows)





Bi-conic 22/55° flow Ma=15.6: Temperature



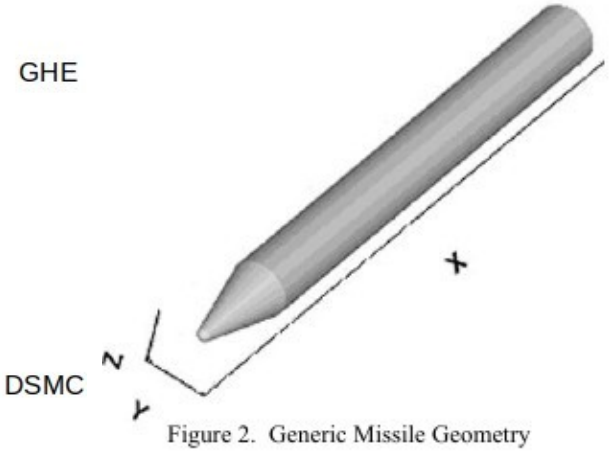
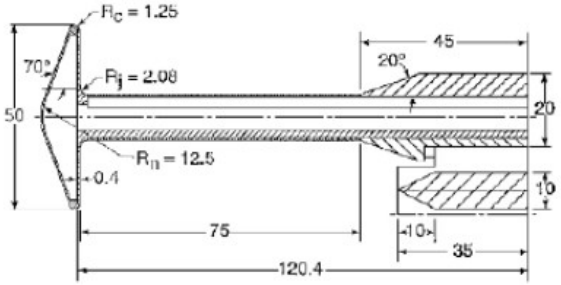


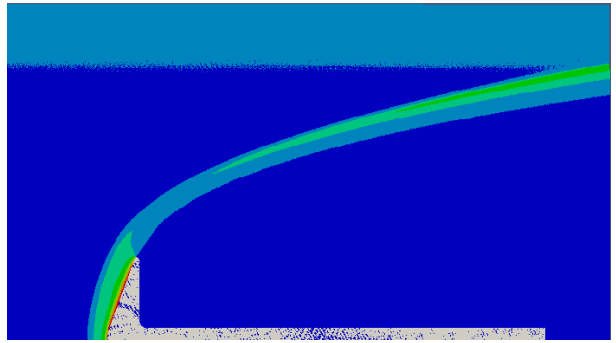
Figure 2. Generic Missile Geometry

GHE/ DSMC flight temperature at 160km comparison with Papp(2004)
Mesh 50,000 nodes

FEMINA/3D flow solver with CNSPACK (hypersonic flows)

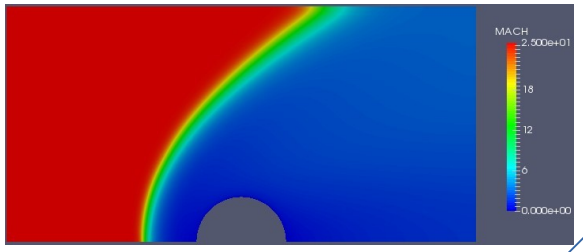
70° blunt cone flow Ma=20.2: ρ density



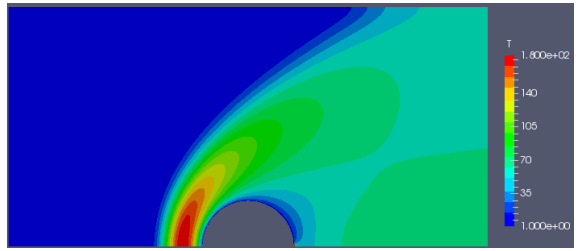


Geometry **Density**

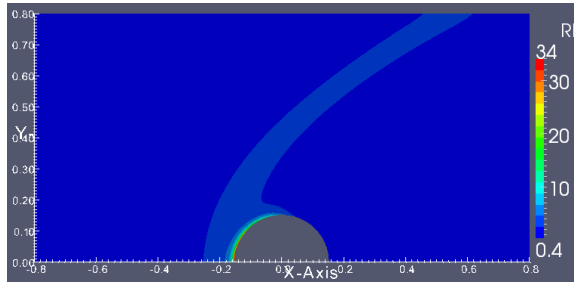
2D-cylinder hypersonic flow Ma=25, Kn=0.05, Re=850



Mach number



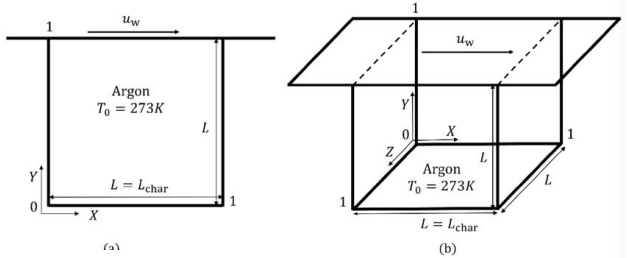
Temperature

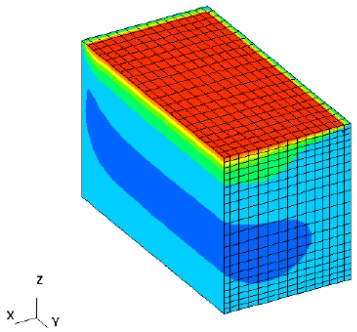


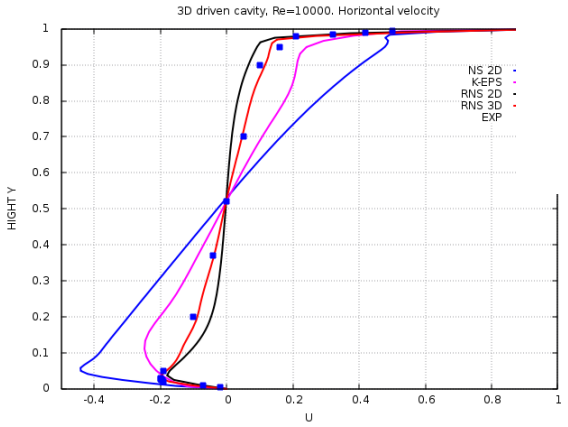
Density

Scientific Applications with CNSPACK Linear Solver: Turbulent Flows

FEMINA/3D flow solver with CNSPACK (turbulent flows)

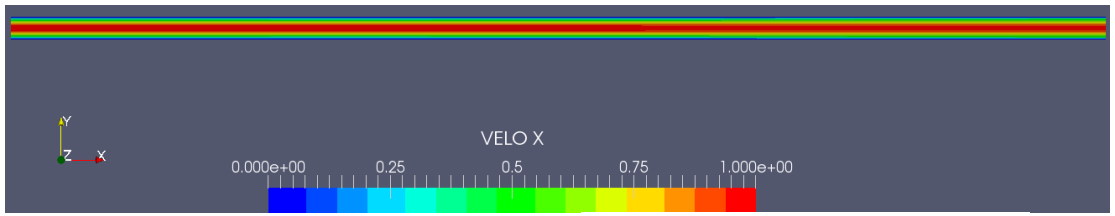


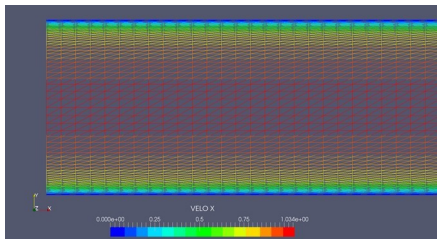


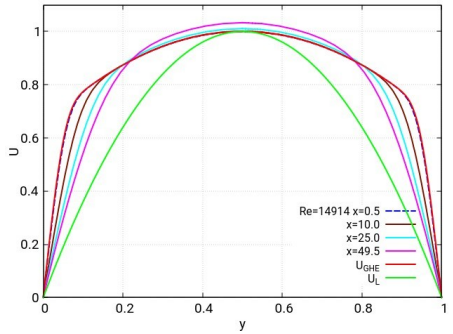


Mesh 50,000 nodes

3D lid-driven cavity flow Re=10,000



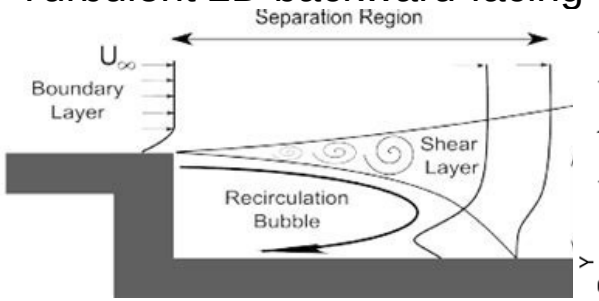


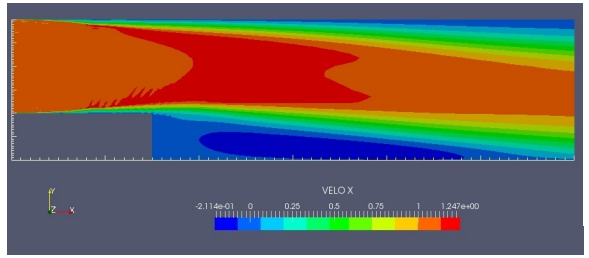


Turbulent flow in channel Re=14,914
Mesh 20,000 nodes

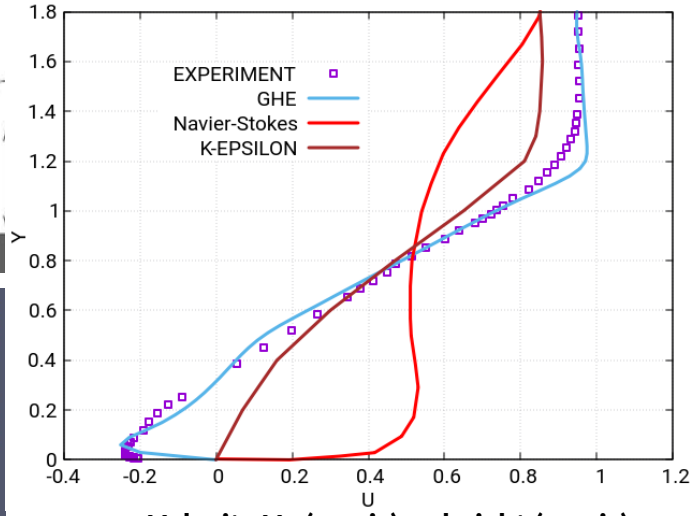
CNSPACK convergence requirement 1E-16

Turbulent 2D backward-facing step flow at Re=132,000





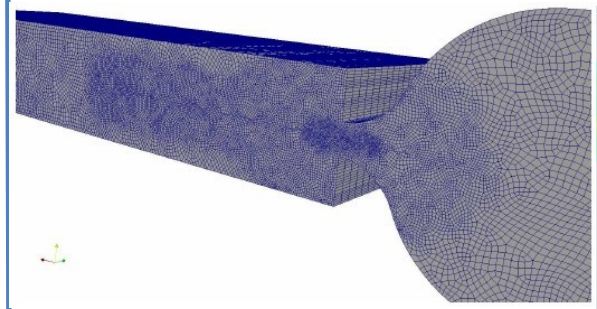
Velocity U



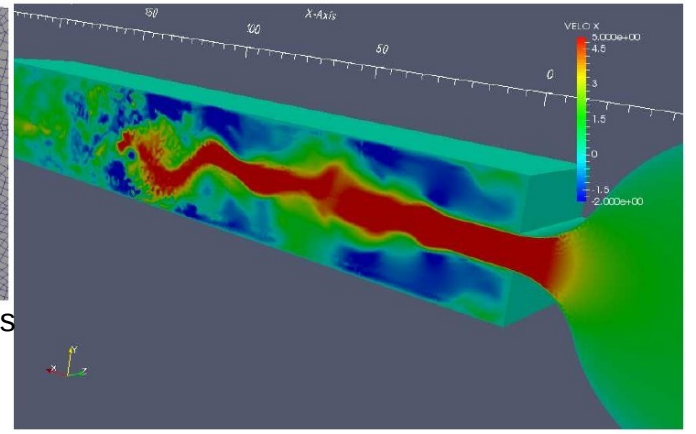
Velocity Vx (x-axis) vs height (y-axis)

Mesh 30,000 nodes

3D backward-facing step flow at Re=55,000 experiment



Mesh 800,000 nodes refined near walls



Velocity $|\boldsymbol{V}|$

History

- CNSPACK started as a linear solver for CFD in 1990s.
- Heavy numerical experiments have been performed to select the most efficient solver (IDR, CGS, GMRES, etc) and preconditioners with minimal memory requirements.
- IDR and ILU(m) have been an optimal choice at that time.
- No ILU(0) convergence for Finite Element analysis of incompressible flow (P2-P1 approximation for velocity-pressure) : ILU(m>0) is a must.
- No pivoting for ILU construction to make it fast.
- Kershaw modification was used to deal with zero diagonals (due to equation, *div* ***V*** =0).
- Typical convergence for CFD: 20 to 200 iterations for N=2,500,000, eps=1E-9.
- More application in different areas: **CFD and semiconductor modeling are the most challenging cases**.
- Recent attempts to make CNSPACK a parallel solver were partly successful.

Current Focus

Linear Preconditioned IDR Solver for

- Semiconductor Device Simulator:

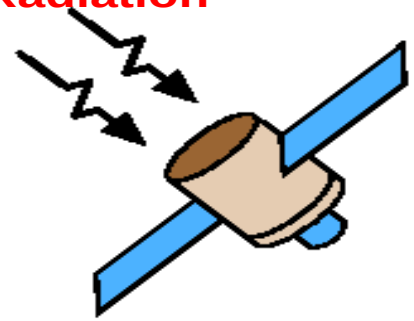
  Radiation Effects in Space Environment, and

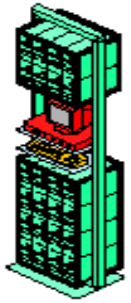
- Computational Fluid Dynamics:

  Hypersonic and turbulent flows

Goal: **Fast Analysis and Prediction of Radiation Effects in Nanoscale Electronics (Space, Extreme Environments)**

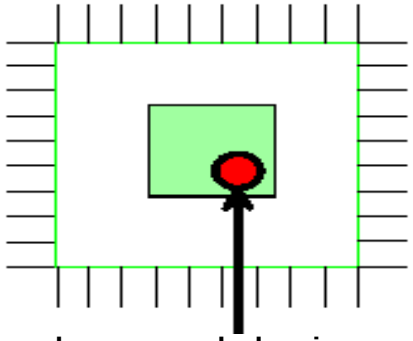
Space / Nuclear Radiation





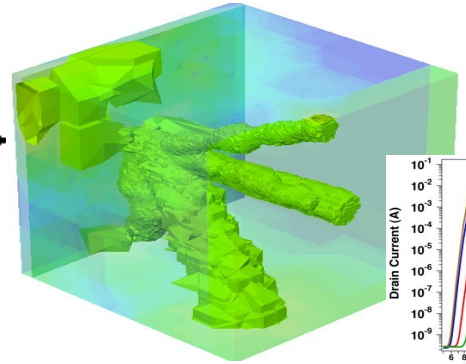
damaged logic board

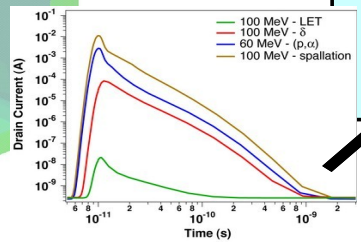
damaged chip



damaged devices

Radiation-Response Mechanisms & Models

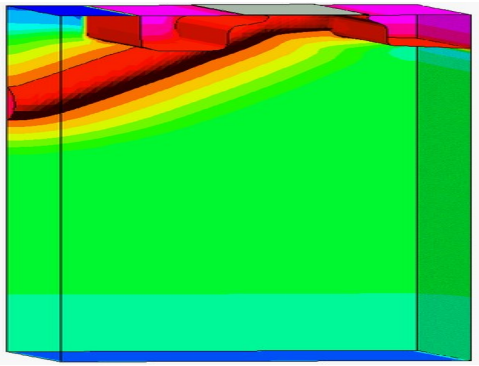




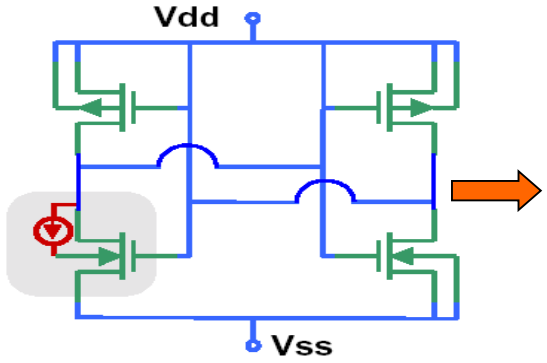
Automated Analysis & Design Tools

Rad-hard devices & circuits ⇨ minimum damage!

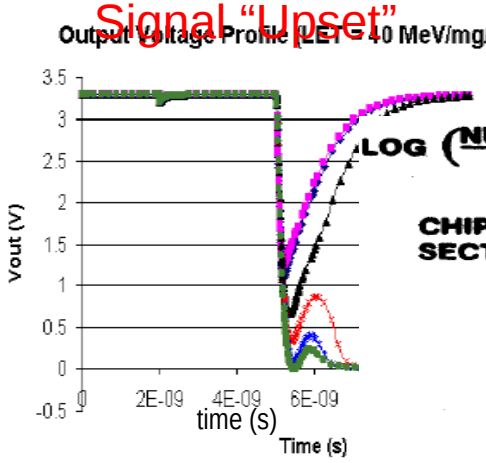
3D Device Simulation



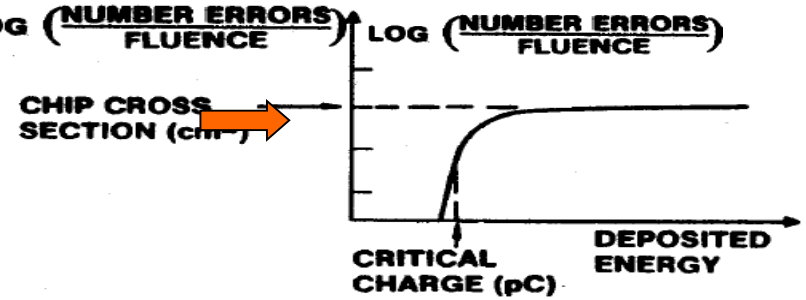
Mixed-Mode or Circuit Model



Signal "Upset"

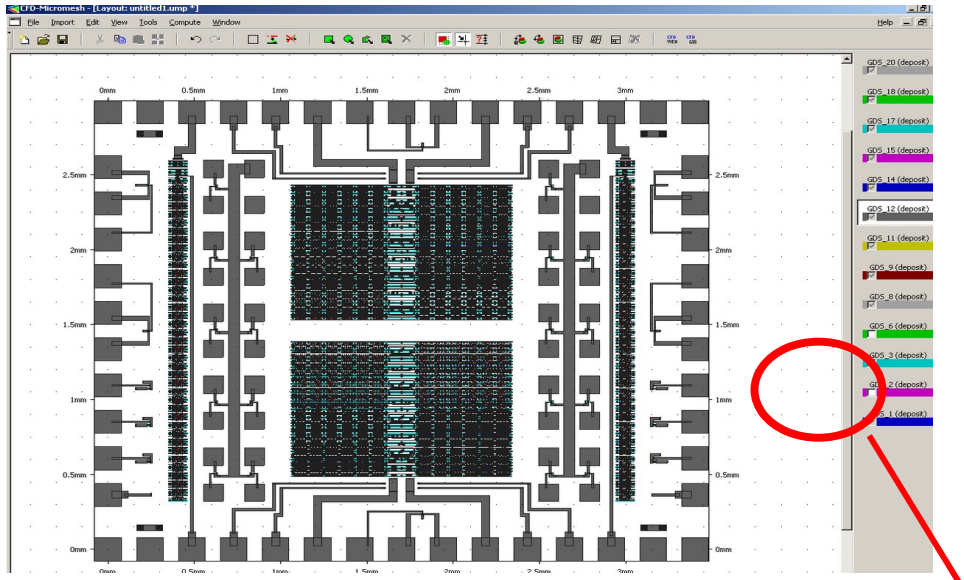


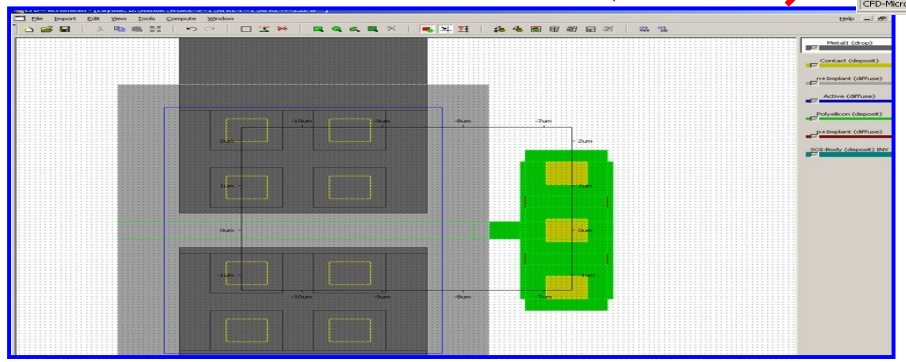
CHIP CROSS SECTION (cm^2) (Sensitivity to Radiation)



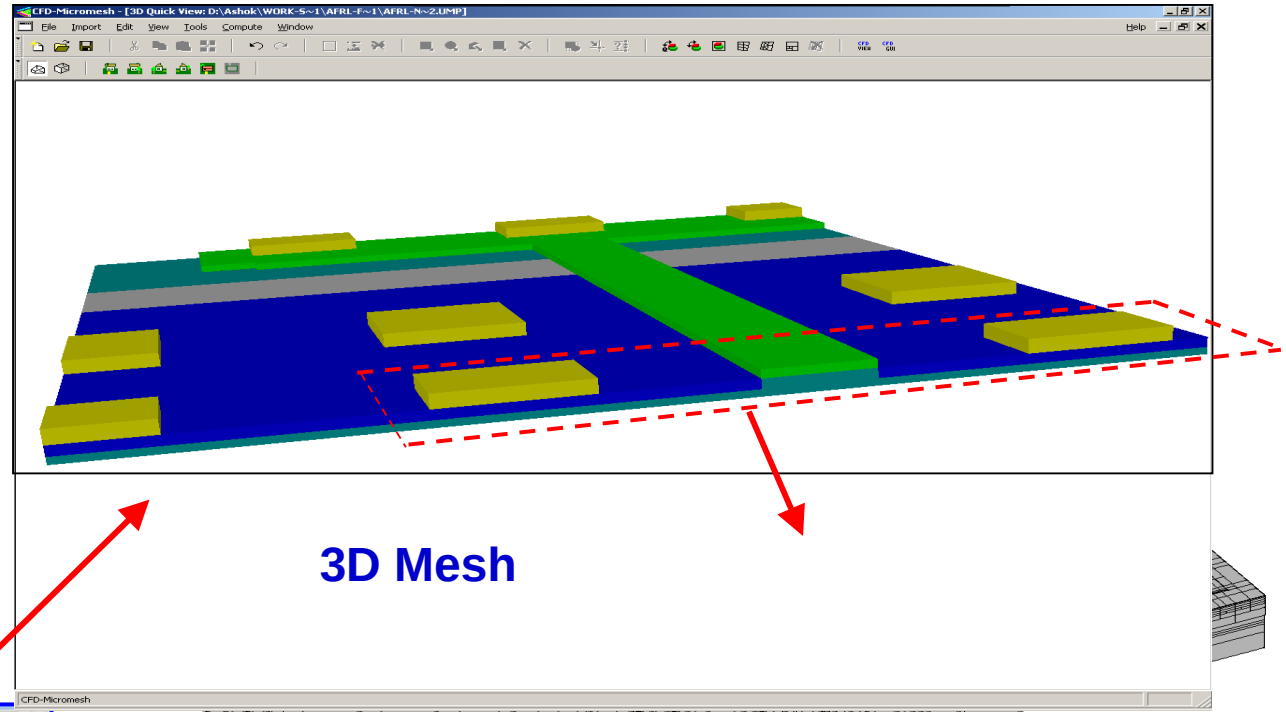
IC Layout → 3D Model → 3D Mesh → Simulation

Imported Layout
(GDSII, CIF, DXF, GIF)

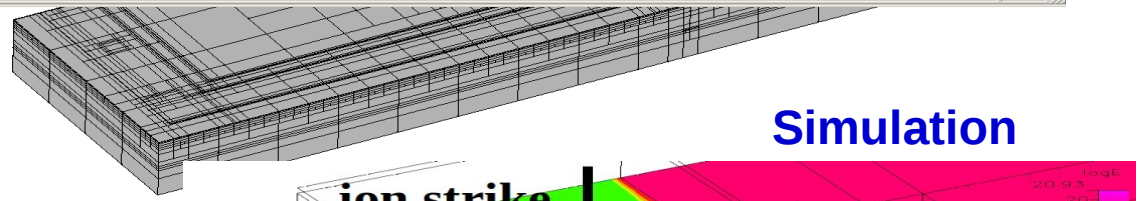




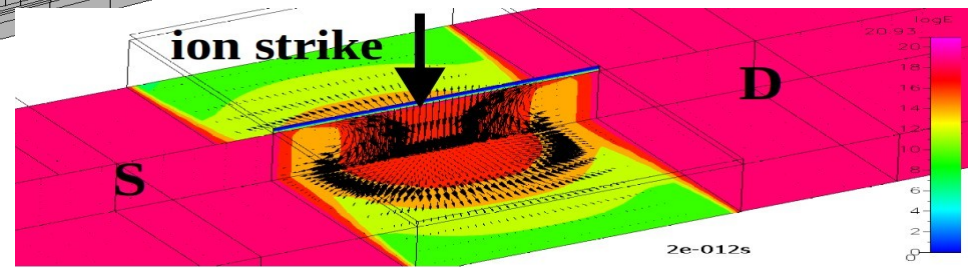
Automatic Generation of 3D Model



3D Mesh

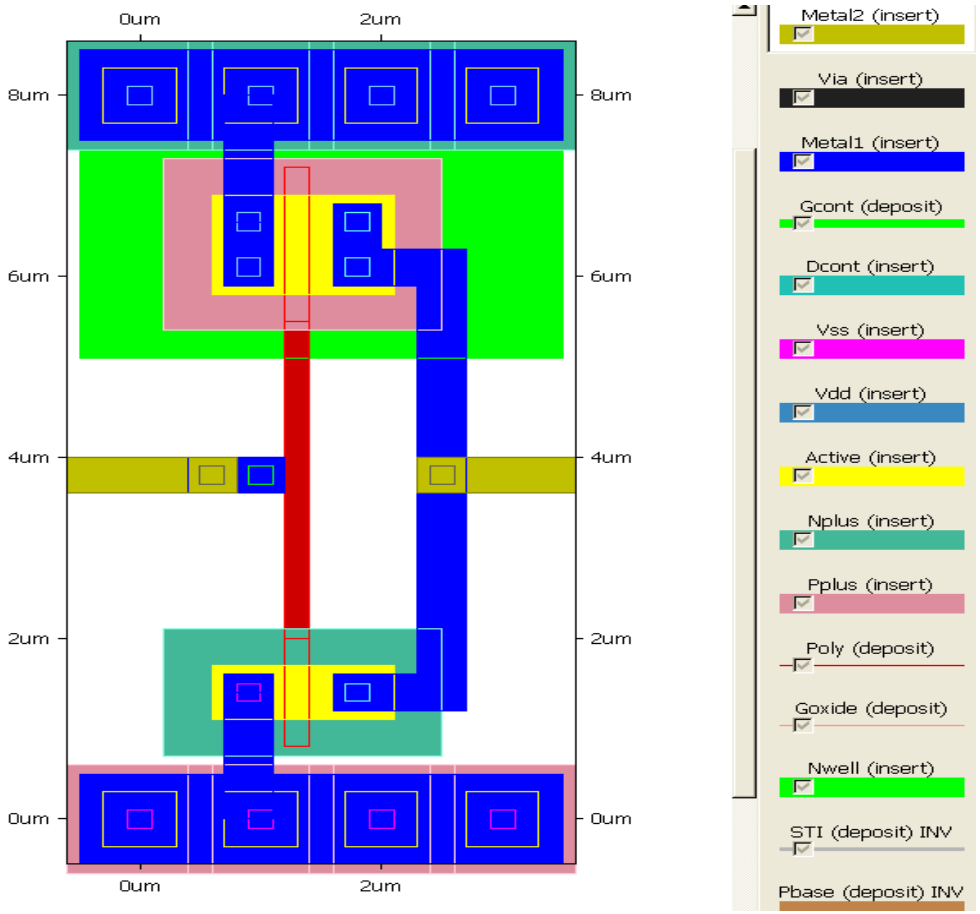


Simulation

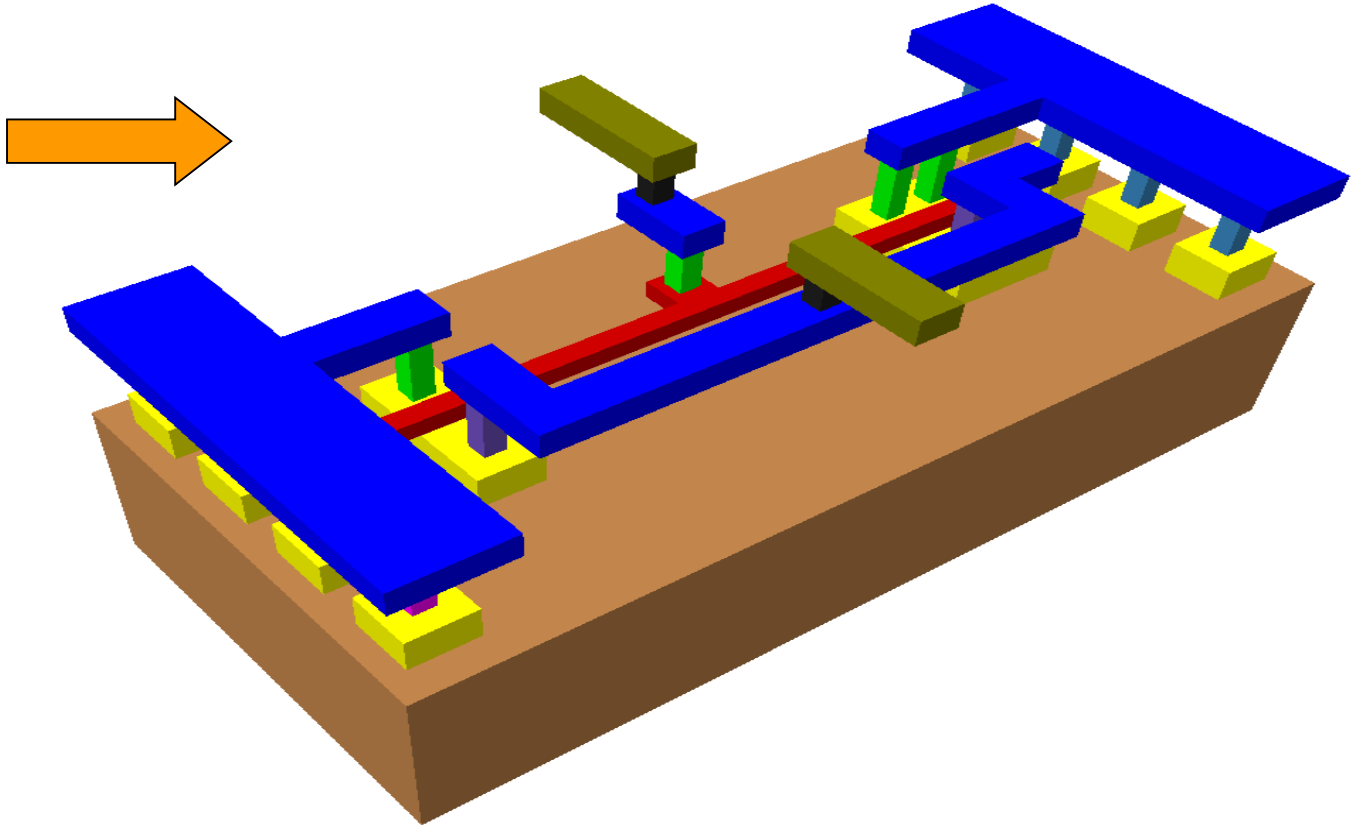


IC Layout : 3D Model with Metallization

Imported IC Layout
(GDSII)

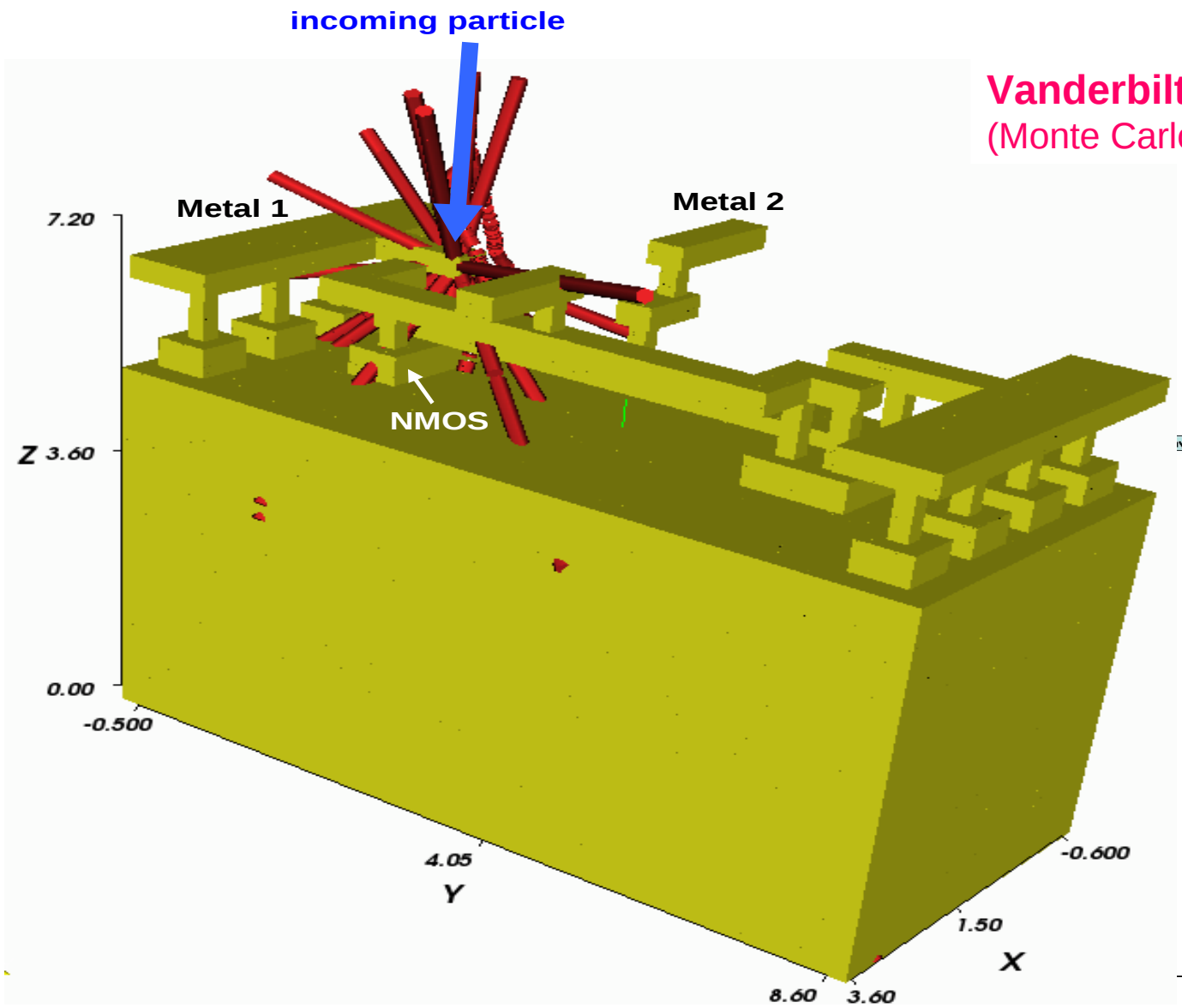


Automatically Generated 3D Model



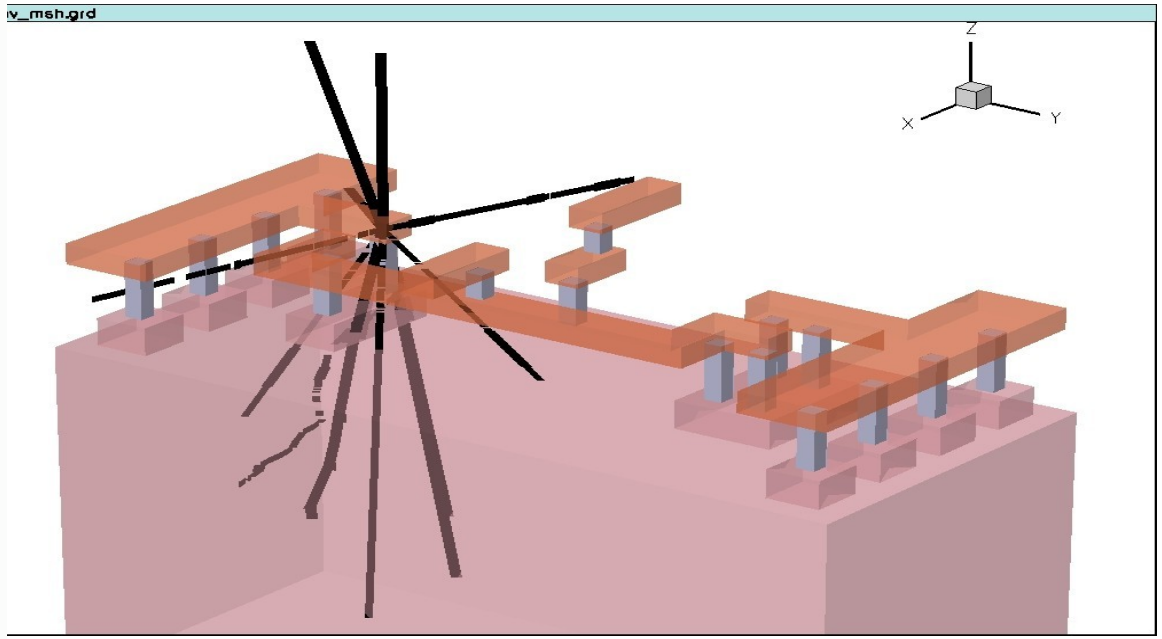
MRED/Geant4 Radiation Tracks in 3D Model

Vanderbilt University



Vanderbilt MRED
(Monte Carlo Radiative Energy Deposition)

MRED/Geant4 simulated ion beam, focused on the tungsten via



TCAD Solver

Full 3-D Semiconductor Device Simulator, based on
Drift-Diffusion (DD) or Hydrodynamic (HD) Semiconductor Models:

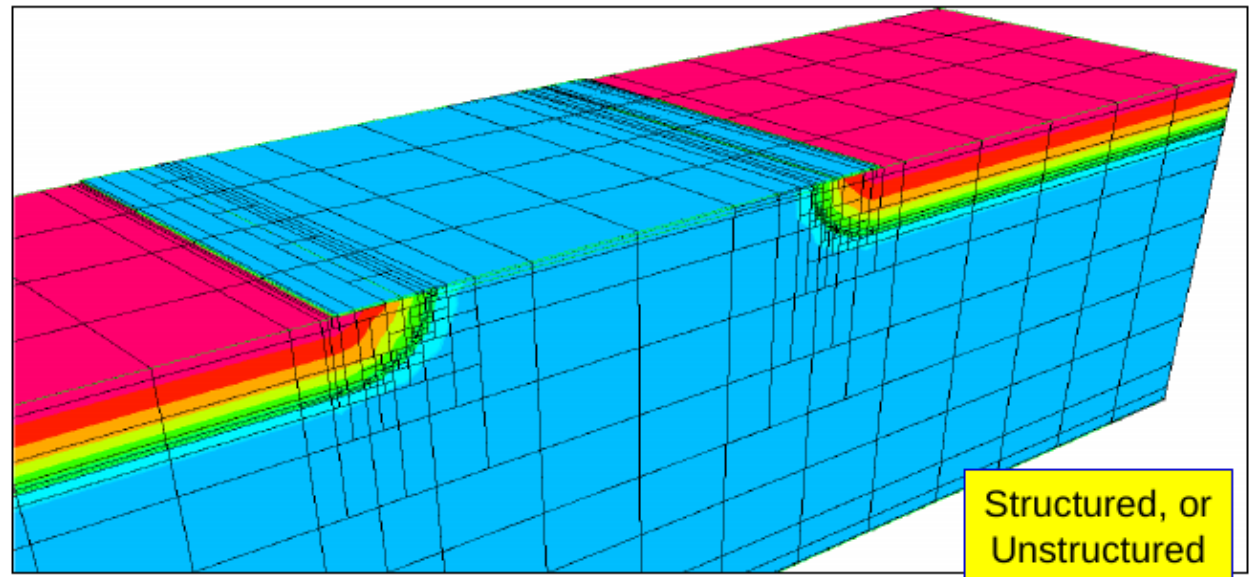
Electric Potential Equation

$$\nabla \cdot (-\varepsilon \nabla \phi) = q(p - n + N_D^+ - N_A^-)$$

Carrier Continuity Equations

$$q\frac{\partial n}{\partial t} - \nabla \cdot \vec{J}_n = q(G - R)$$

$$q\frac{\partial p}{\partial t} + \nabla \cdot \vec{J}_p = q(G - R)$$



Structured, or Unstructured Meshes

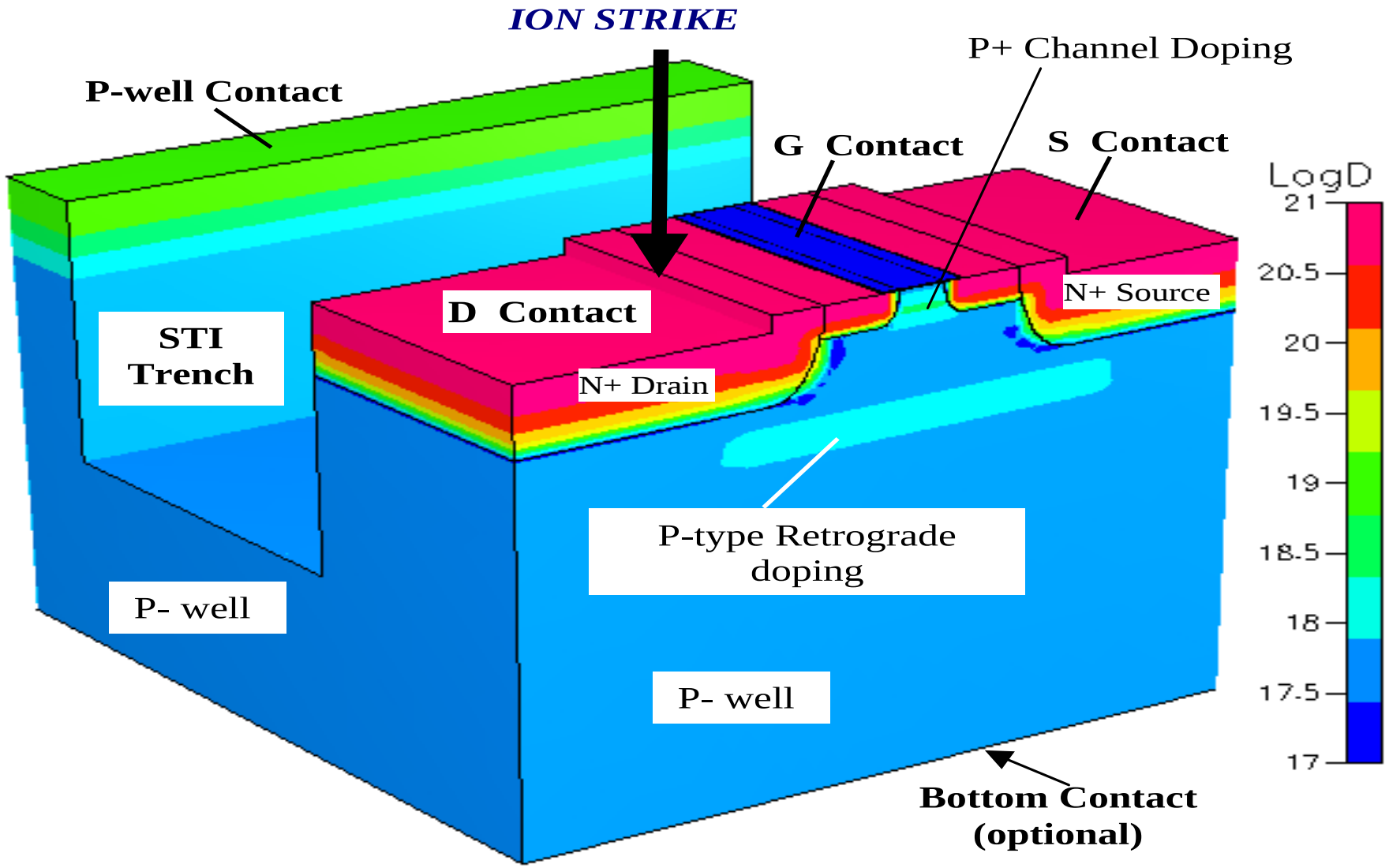
where current densities are:

$$\vec{J}_n = qD_n \nabla n + qn\left\{\mu_n \nabla\left(-\phi + \frac{E_c}{q}\right) + D_n \nabla T_n - \frac{3}{2} D_n \nabla \ln m_n\right\}$$

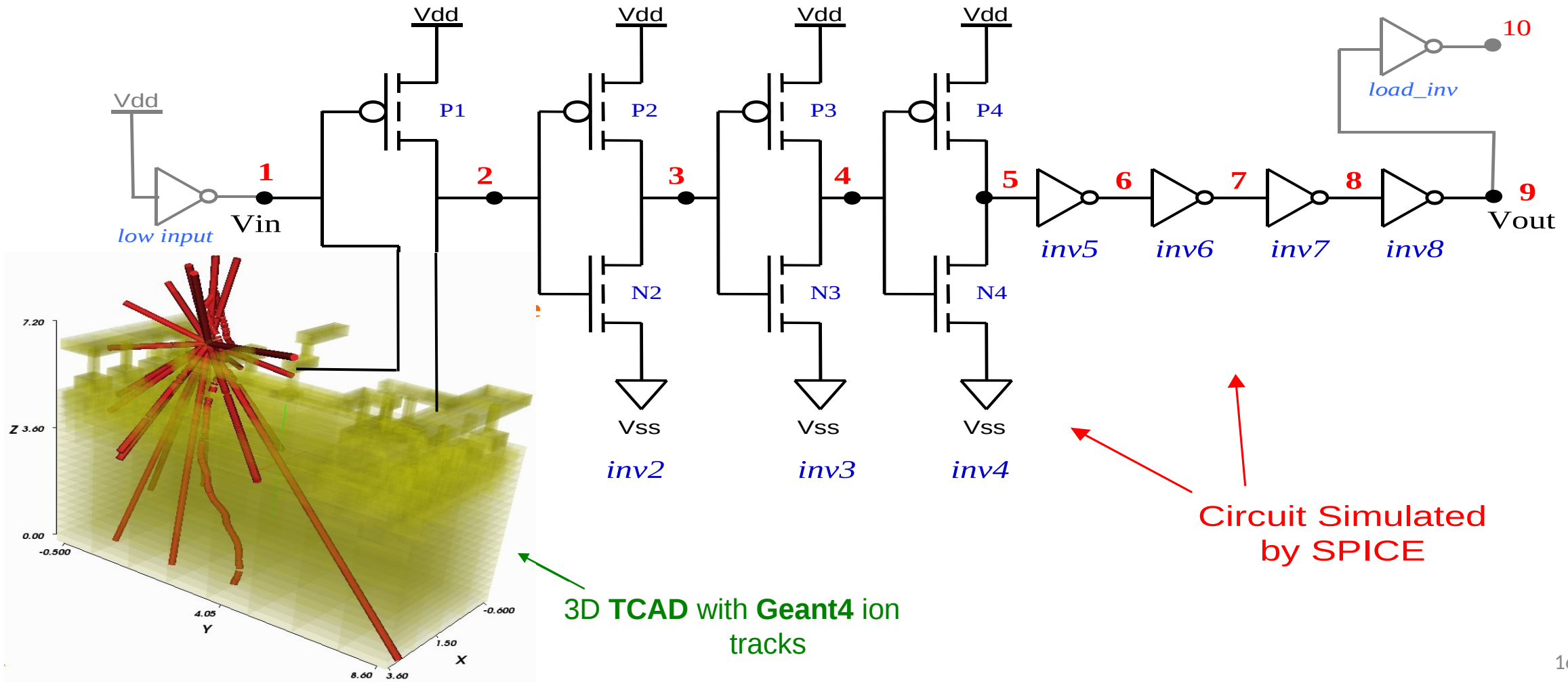
$$\vec{J}_p = -qD_p \nabla p + qp\left\{\mu_p \nabla\left(-\phi + \frac{E_v}{q}\right) + D_p \nabla T_p - \frac{3}{2} D_p \nabla \ln m_p\right\}$$

3D Model Setup

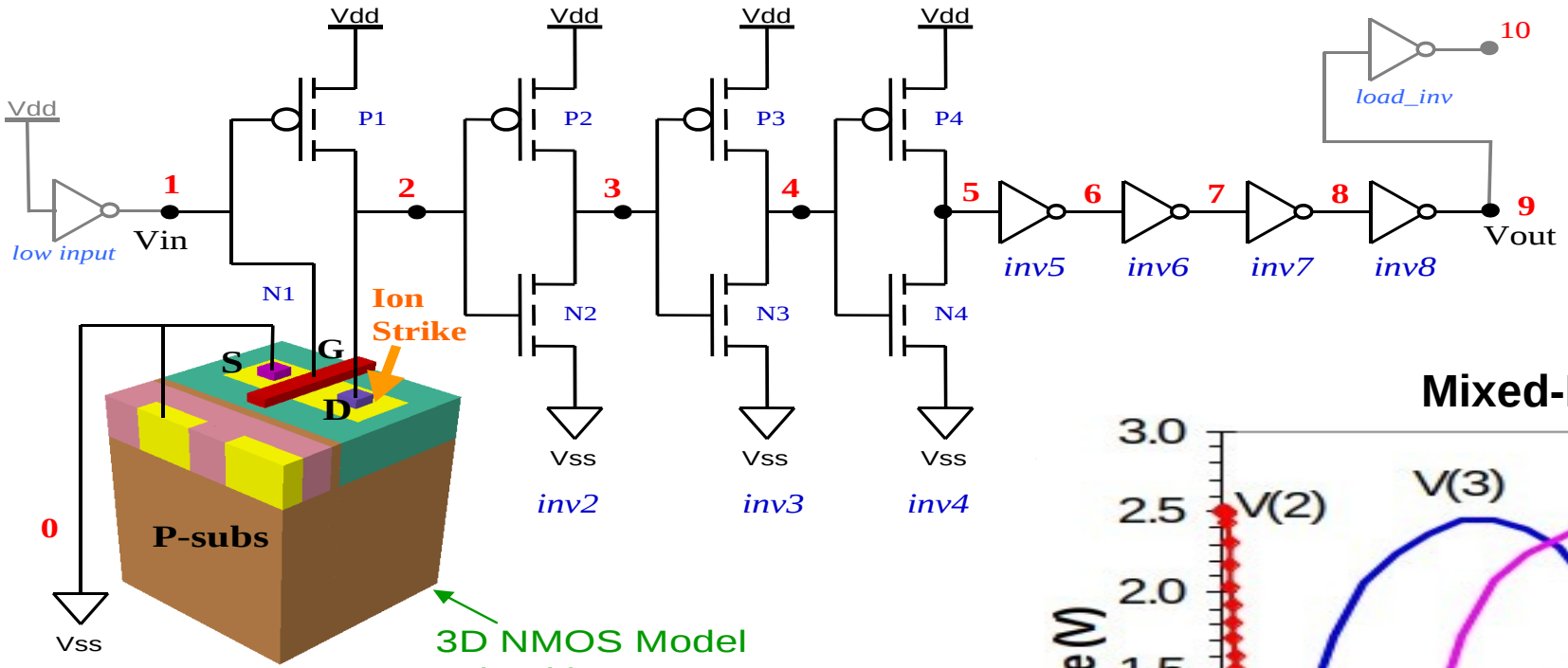
Our 3D Model of NMOSFET, for Mixed-Mode Simulations:

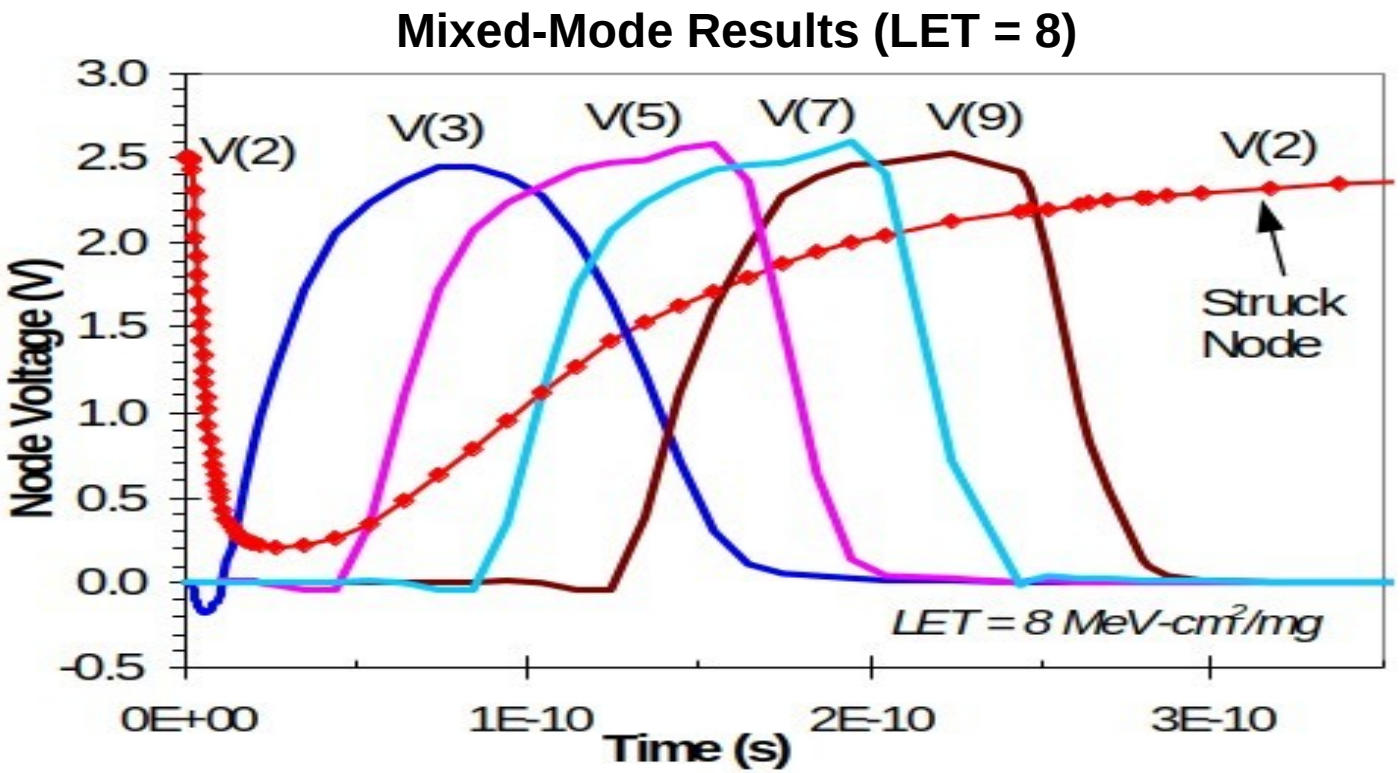


Coupled **Mixed-Mode** *TCAD* **3D Simulations with** *MRED/Geant4* **Nuclear Reactions** in Nanoscale Electronics



DSET Pulse Propagation

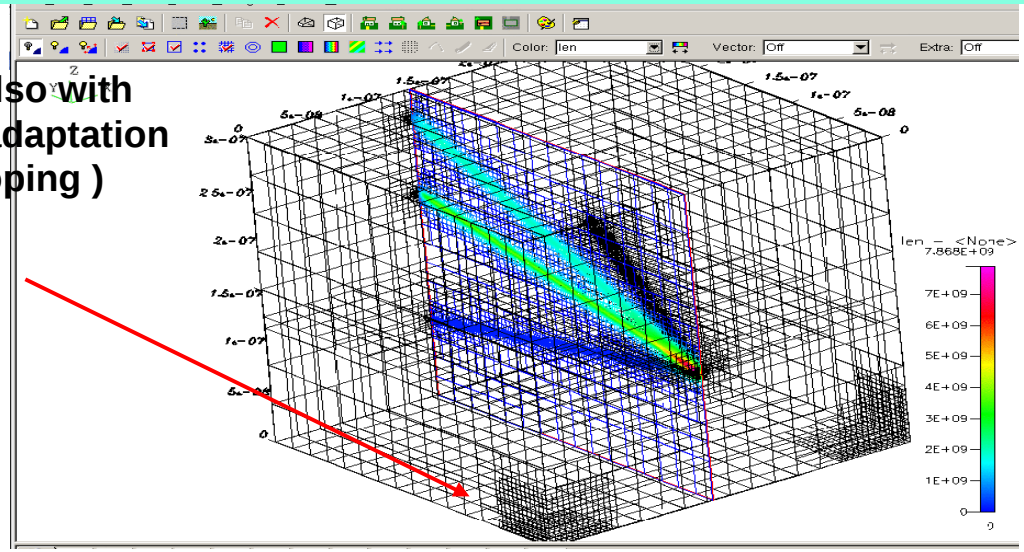


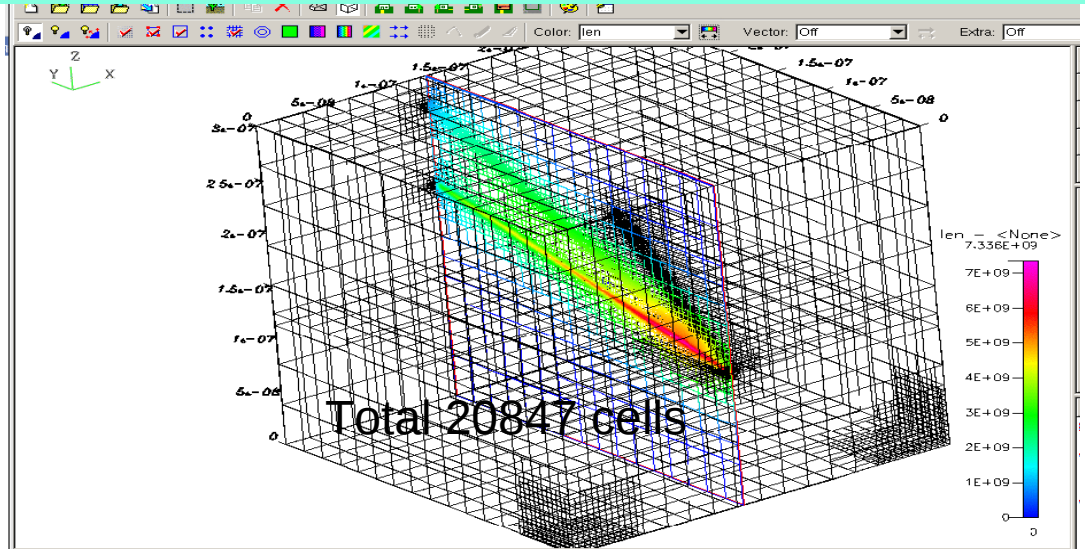


Radiation Effects in Semiconductor Devices -Mesh Adaptation to Ion Tracks from Geant4

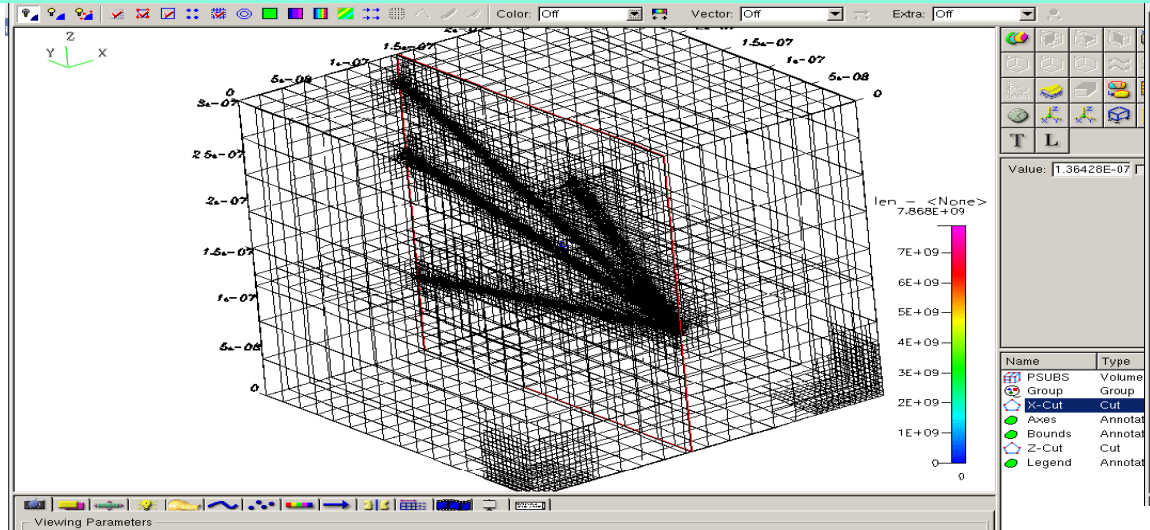
Example of using FilterThreshold to remove track segments with low deposited energy

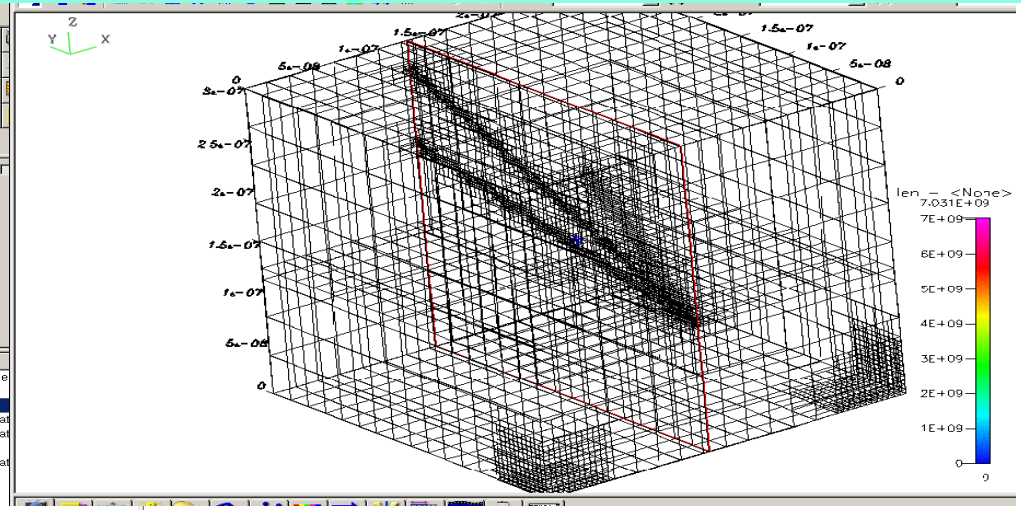
- **works also with additional adaptation (e.g., doping)**





Example of fine mesh generation on 4 segments and a coarser mesh on 3 segments





Physics Based Model for Fluid Flow

Generalized Hydrodynamic Equations (GHE) [Fedoseyev 2012]

Momentum transport equation for the incompressible fluid :

$$\frac{\partial \mathbf{V}}{\partial t}+(\mathbf{V}\nabla)\mathbf{V}-Re^{-1}\nabla^2\mathbf{V}+\nabla p-\mathbf{F}=\tau^*\left\{2\frac{\partial}{\partial t}(\nabla p)+\nabla^2(p\mathbf{V})+\nabla(\nabla\cdot(p\mathbf{V}))\right\}$$

GHE continuity equation for the incompressible fluid is:

$$\nabla\cdot\mathbf{V}=\tau^*\left\{2\frac{\partial}{\partial t}(\nabla\cdot\mathbf{V})+\nabla\cdot(\mathbf{V}\nabla)\mathbf{V}+\nabla^2 p-\nabla\cdot\mathbf{F}\right\}$$

τ^* (GHE timescale), $\tau = l^2/L^2\,\mathrm{Re} = K\,\mathrm{Re}$, $K = l^2/L^2$

ASSUMPTIONS:

- τ is constant.
- neglect the nonlinear terms of the third order in the fluctuations, and terms of the order τ /Re and smaller
- assume the *slow* flow variation => neglect second derivatives in time

ADVANTAGES :

- Finite Element Method with equal order approximation for *velocity and pressure*
- Allows to solve using efficient iterative methods (CG-like)

CHALLENGES:

- The model to be carefully tested, results to be compared with published results and experimental data obtained for different flow problems

Kinetic Physics Based Model

Generalized Boltzmann Equation (GBE) [Alexeev 1994; 2004]

The GHE model is based on results from kinetic theory by Alexeev(1994) *

The Boltzmann equation (BE) describes the evolution of the one-particle velocity distribution function ***f*** in time and space and is the basis of the kinetic theory of gases.

$$\frac{Df}{Dt} = J$$

Here J is a collision integral, and D/Dt is a material derivative. It takes into account the changes in the f on a scale of the mean time between collisions and on hydrodynamic time scale. As shown by Alexeev a generalization of BE accounting for a third scale (related to the finite dimension of the particles) results in an additional term :

$$\frac{Df}{Dt} - \frac{Df}{Dt}\left\{\tau \frac{Df}{Dt}\right\} = J$$

where τ is a mean time between collisions. The new term is proportional to the Knudsen number, *Kn*, and, therefore, in the hydrodynamic limit, to viscosity.

The Generalized Hydrodynamic Equations (GHE) for gases are obtained by a standard procedure from the generalized BE (GBE) and contain new terms, related to an additional term in GBE, that represents spatial and temporal Kolmogorov fluctuations.

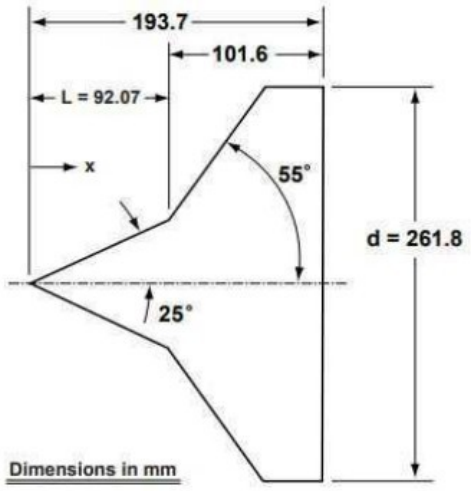
References:

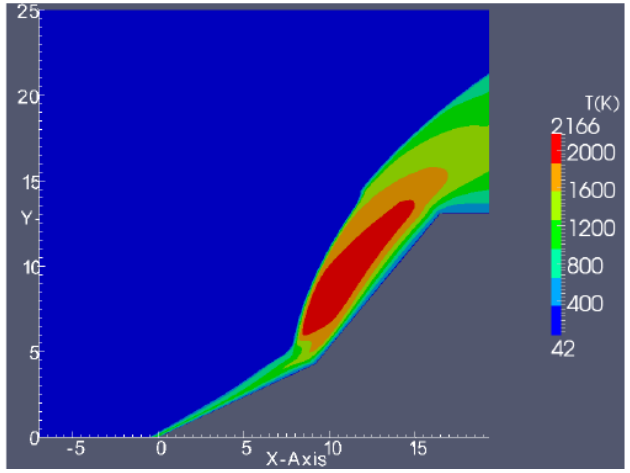
* Alexeev, The generalized Boltzmann equation, generalized hydrodynamic equations and their applications. Phil. Trans. Roy. Soc. London, A. 349 (1994), 417-443

** Alexeev, Generalized Boltzmann Kinetics, Elsevier, 2004.

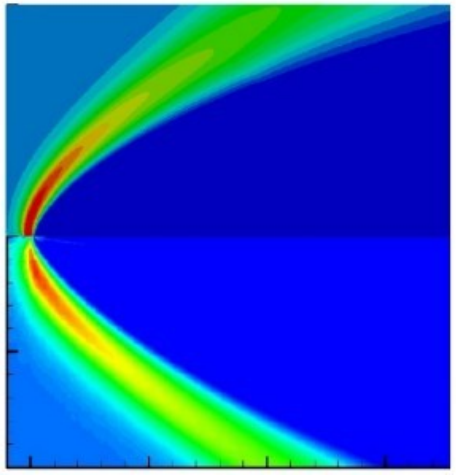
CFD Industrial Applications with CNSPACK Linear Solver - Hypersonic Flows

FEMINA/3D flow solver with CNSPACK (hypersonic flows)





Bi-conic 22/55° flow Ma=15.6: Temperature



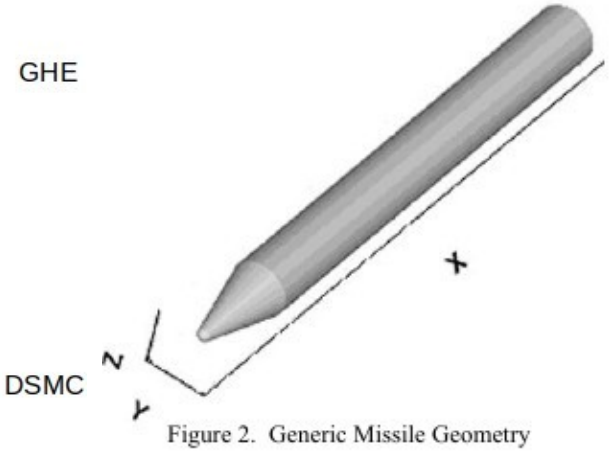
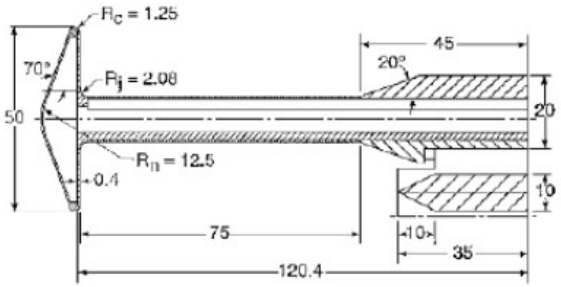


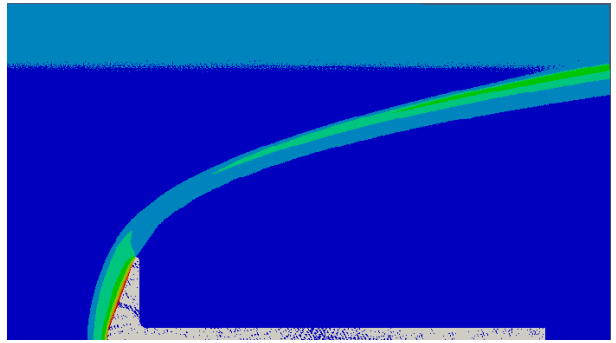
Figure 2. Generic Missile Geometry

GHE/ DSMC flight temperature at 160km comparison with Papp(2004)
Mesh 50,000 nodes

FEMINA/3D flow solver with CNSPACK (hypersonic flows)

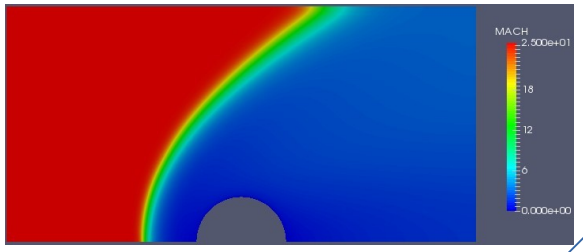
70° blunt cone flow Ma=20.2: ρ



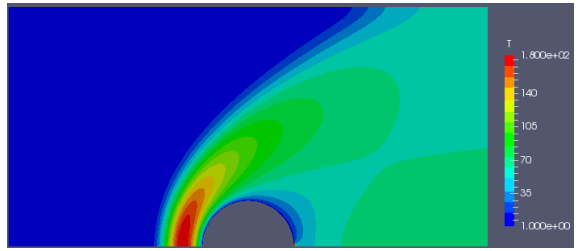


Geometry **Density**

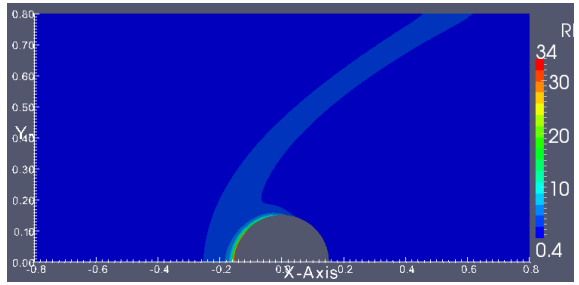
2D-cylinder hypersonic flow Ma=25, Kn=0.05, Re=850



Mach number



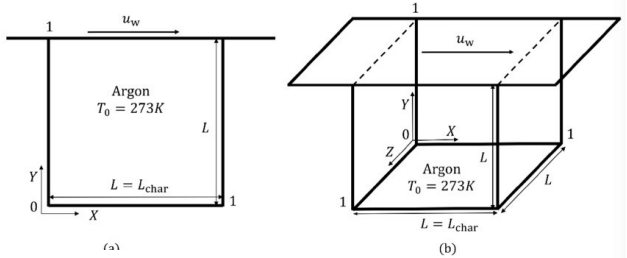
Temperature

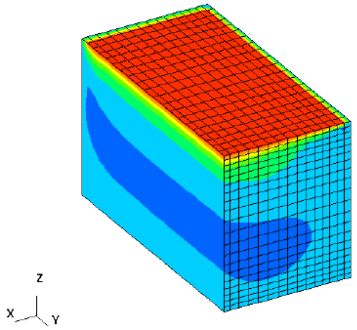


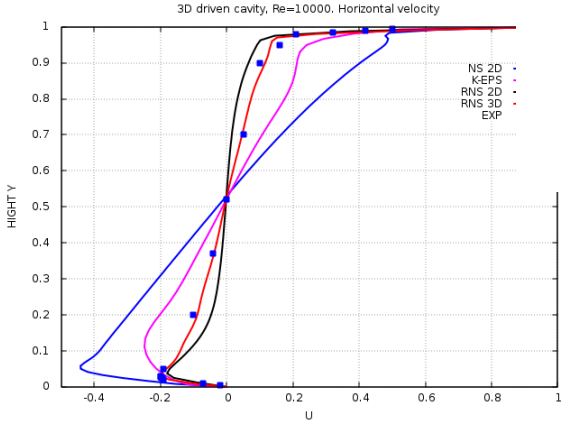
Density

CFD Scientific Applications with CNSPACK Linear Solver: Turbulent Flows

FEMINA/3D flow solver with CNSPACK (turbulent flows)

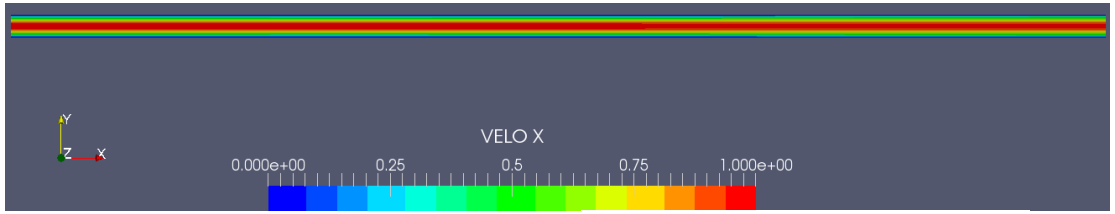


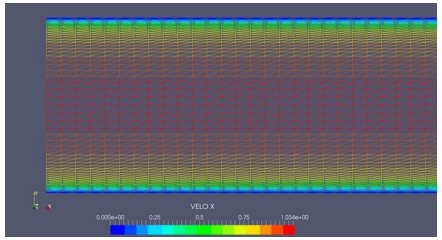


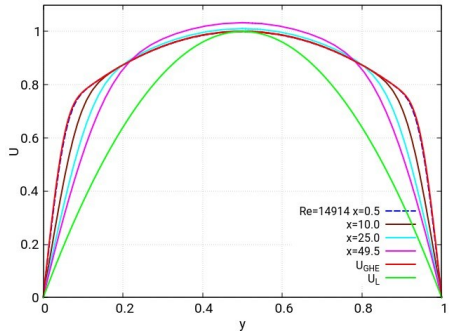


Mesh 50,000 nodes

3D lid-driven cavity flow Re=10,000



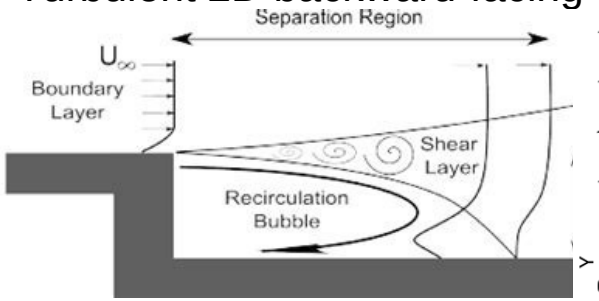


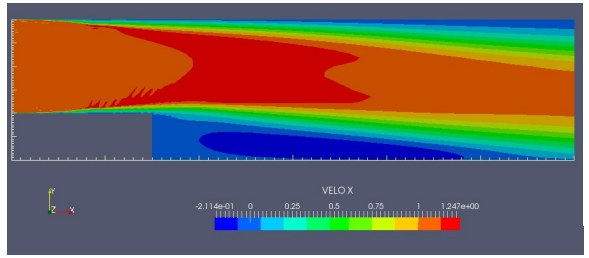


Turbulent flow in channel Re=14,914
Mesh 20,000 nodes

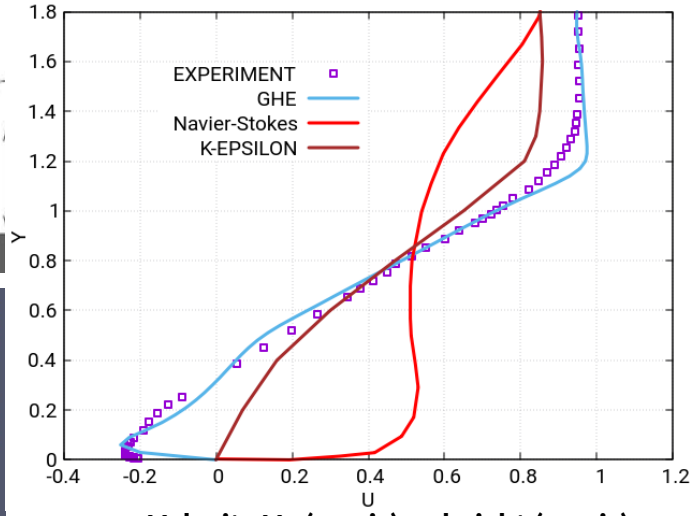
CNSPACK convergence requirement 1E-16

Turbulent 2D backward-facing step flow at Re=132,000





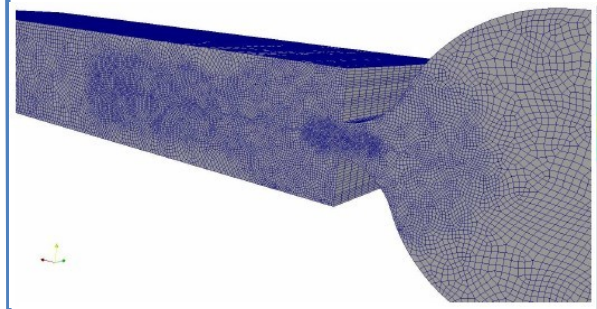
Velocity U



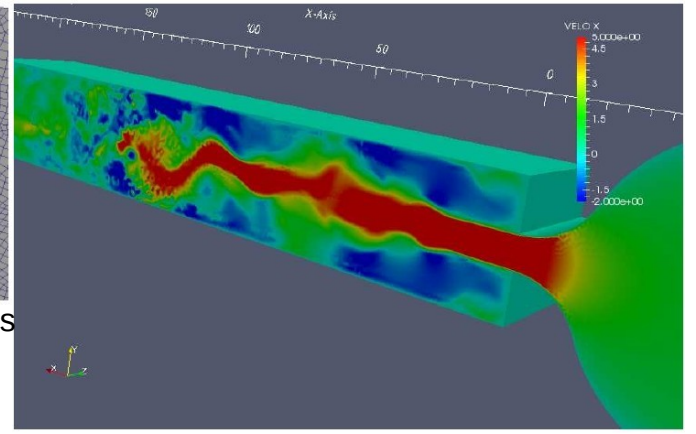
Velocity Vx (x-axis) vs height (y-axis)

Mesh 30,000 nodes

3D backward-facing step flow at Re=55,000 experiment



Mesh 800,000 nodes refined near walls



Velocity $|\boldsymbol{V}|$

Matrix properties and tricks to speed up computations

$$Ax = b$$

- A is nonsymmetric, not positively definite
- Unstructured and large
- Sparse
- Banded
- Number of nonzero elements in each row ~ 100 (varies)
- Gibbs-Pool-Stockmayer (GPS) method is used to reduce the matrix bandwidth.
- Sparse storage scheme is implemented for rows and columns with fast access.
- Storing A as real*4 to save memory, arithmetic is done in double precision

IDR & ILU algorithms (P.Wesseling & P.Sonneveld (1980)

IDR ILU

(Krylov/Lanczos type method) (m=1)

Algorithm 4.1 (IDR method)

$n:=\omega_n:=0;\ f_n:=Ax_n-b,\ dg_n:=dy_n:=0;$

for n:=n+1 do

begin $s_n:=f_{n-1}+\omega_{n-1}dg_{n-1}; t_n:=As_n;$

if n=1 V n is even then $\tilde{\alpha}_n:=(t_n,s_n)/(t_n,t_n)$

else $\tilde{\alpha}_n:=\tilde{\alpha}_{n-1};$

$dx_n:=\omega_{n-1}dy_{n-1}-\tilde{\alpha}_n s_n;$

$df_n:=\omega_{n-1}dg_{n-1}-\tilde{\alpha}_n t_n;$

$x_n:=x_{n-1}+dx_n; f_n:=f_{n-1}+df_n;$

if n is even then

begin $dg_n:=dg_{n-1}; dy_n:=dy_{n-1}$

end else

begin $dg_n:=df_n; dy_n:=dx_n$

end;

$\omega_n:=-(p,f_n)/(p,dg_n)$

end of IDR method;

Number of vectors ***p*** is user input.
Usually one vector ***p*** is used, and the memory for the method is 8*[dim ***p***]

Algorithm 5.2

$A^o:=A;$

for r:=1(1)N do

begin $a^r_{rr}:=\mathrm{sqrt}(a^{r-1}_{rr});$

for $j>r \wedge (r,j)\in P$ do $a^r_{rj}:=a^{r-1}_{rj}/a^r_{rr};$

for $i>r \wedge (i,r)\in P$ do $a^r_{ir}:=a^{r-1}_{ir}/a^r_{rr};$

for $(i,j)\in P \wedge i>r \wedge j>r \wedge (i,r)\in P \wedge (r,j)\in P$ do

$a^r_{ij}:=a^{r-1}_{ij}-a^r_{ir}a^r_{rj}$

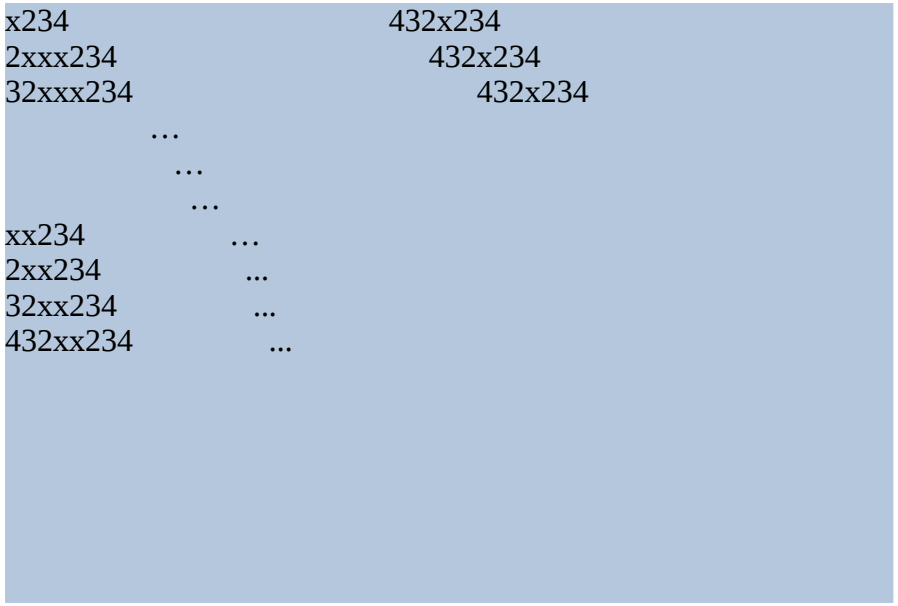
end incomplete LU decomposition;

Incomplete LU: Elements of L and U are calculated only where the matrix has non-zero elements.

P. Wesseling, P. Sonneveld (1980). Numerical experiments with a multiple grid and a preconditionned Lanczos type methods. Lecture Notes in Mathematics, 771, Berlin: Springer, 1980, 543-562. (algorithms are provided from the original paper)

ILU algorithm (for m>1)

ILU(m), *m* is user chosen



Incomplete LU (m>1): Elements of LU are calculated also near the matrix hs non-zero elements:

X -original martix nonzero elements
2 – (m=2) ILU decomposition, the elements at these positions are also computed in ILU
3 - (m=3) ILU decomposition, these elements are also computed in ILU
4 - (m=4) ILU decomposition, these elements are also computed in ILU
…

Typically used m:
m = 1 or 2 for CFD problems.
m = 3 or 4 for semiconductor problems.
m = 4 to 6 for eigenvalue problems.
Sometime m is made as adaptive: if the solution is not converged within a limit of the number of iterations, then m is increased by 1, and the solution repeated, etc.

If m=N (order of matrix), then it is a complete LU decomposition

P. Wesseling, P. Sonneveld (1980). Numerical experiments with a multiple grid and a preconditionned Lanczos type methods. Lecture Notes in Mathematics, 771, Berlin: Springer, 1980, 543-562.

IDR & ILU algorithms implementation in CNSPACK

IDR

•Compare the IDR and GMRES methods: both methods converge well, if a good preconditioner is used.

•The IDR method needs comparably less memory to store only eight work vectors.

• Both solvers remain accessible in CNSPACK, with time found out that users prefer the IDR solver.

•Number of ***p*** vectors is arbitrary, users prefer just one vector. Inceasing the number of vector decrease the number of iterations, but the time increases proportionaly.

ILU

• A simple incomplete decomposition of the firstorder, ID = 1, can result in the divergence of the iterations. That may be a reason, why this approach is not widely used yet.

We implemented the fast arbitrary order ILU preconditioning ID = m for unstructured matricies. Users typically choose 1 to 3. Sometimes ID is adjusted adaptively, if not converged. For eigenvalue solutions ID is up to 4 or 6.

•To avoid a diagonal pivot degeneration we use the Kershaw diagonal modification:
If the value of diagonal element was small, i.e.
$|a_{ii}| < \alpha = sqrt(2^{-t}\sigma\mu)$, the diagonal was replaced by α.
Here σ and μ are the maximum magnitudes of row and column elements, and 2^{-t} is a machine precision (*t* bits in mantissa), details in *Kershaw (1978)*.

P. Wesseling, P. Sonneveld (1980). Numerical experiments with a multiple grid and a preconditionned Lanczos type methods. Lecture Notes in Mathematics, 771, Berlin: Springer, 1980, 543-562.

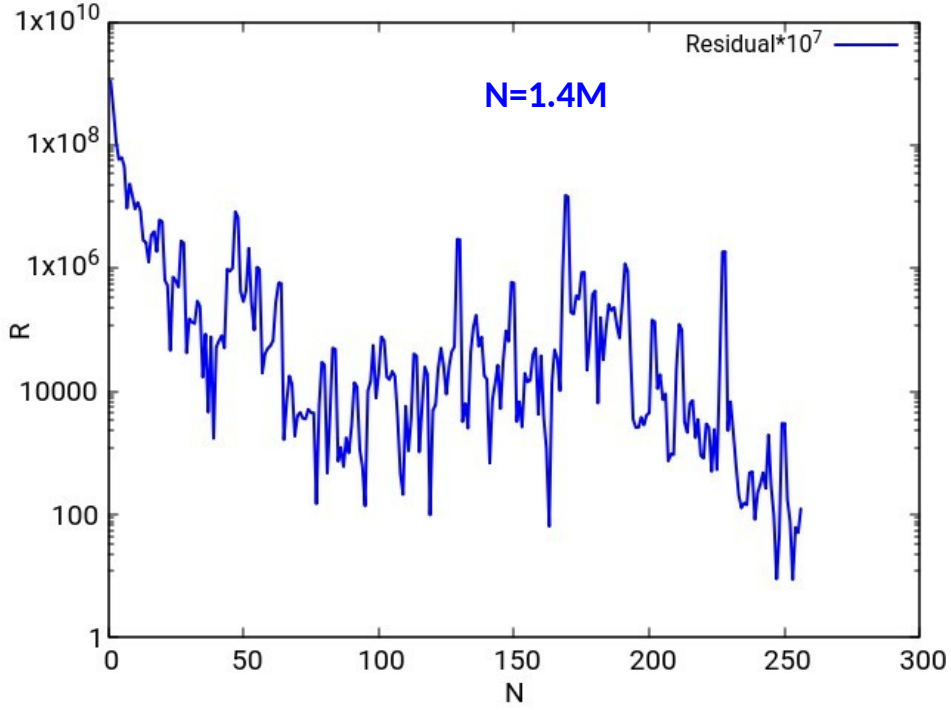
D. S. Kershaw (1978). The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. J. Comput. Phys., 2, 43-65.

Typical performance of CNSPACK for CFD

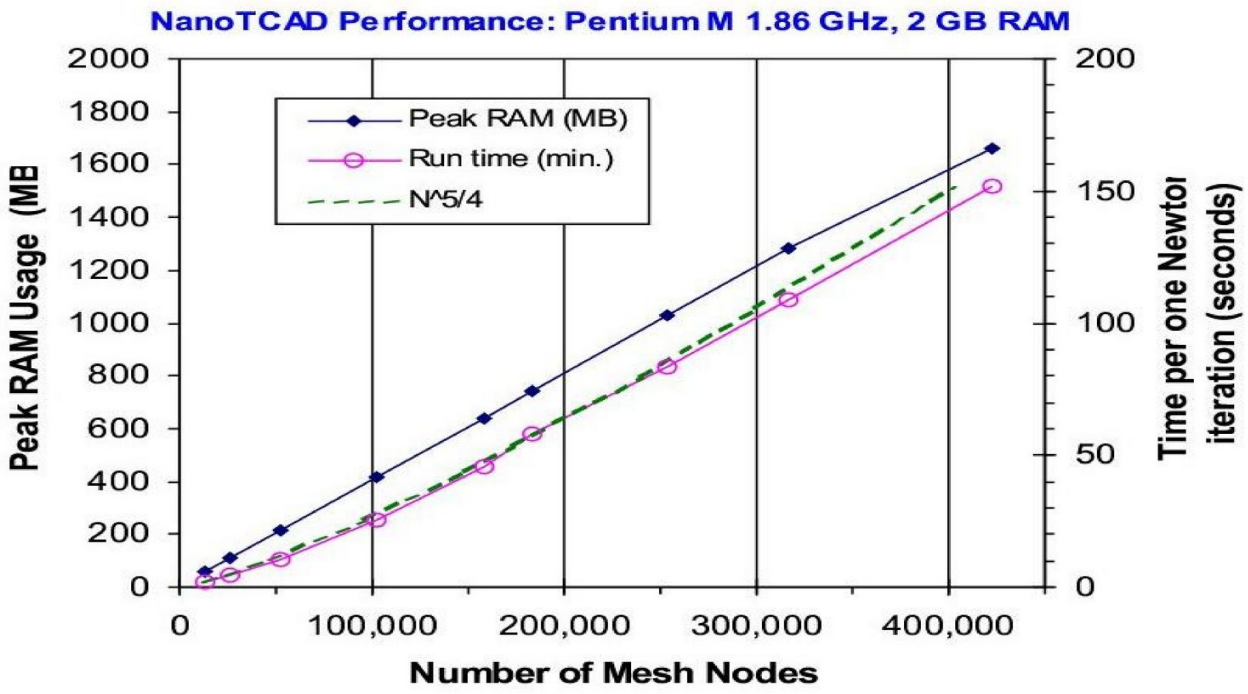
Performance versus matrix order N:
- CPU is almost linear vs N
- Memory is linear vs N

Performance of CNSPACK (Dell T5500Intel(R) Xeon(R) CPU X5650 @2.67GHz)
Matrix N =2.5M, Termination criterion: residual R < Eps = 1E-8 * R_0
Matrix assembly 22s
ILU decomp 18s
Solve (250 iter) 100s

Need to reduce one iteration cost:
- search for faster backward substitution algorithms / parallelization algorithms.

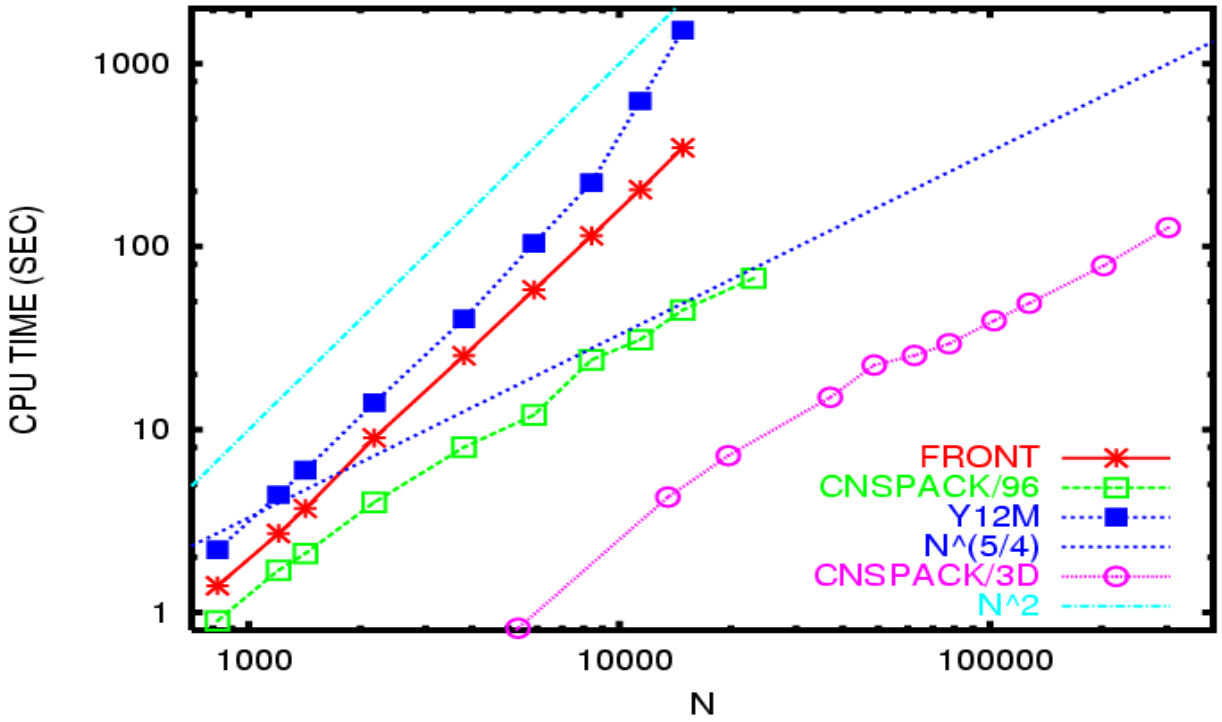


Typical performance of CNSPACK for CFD and semiconductor modeling



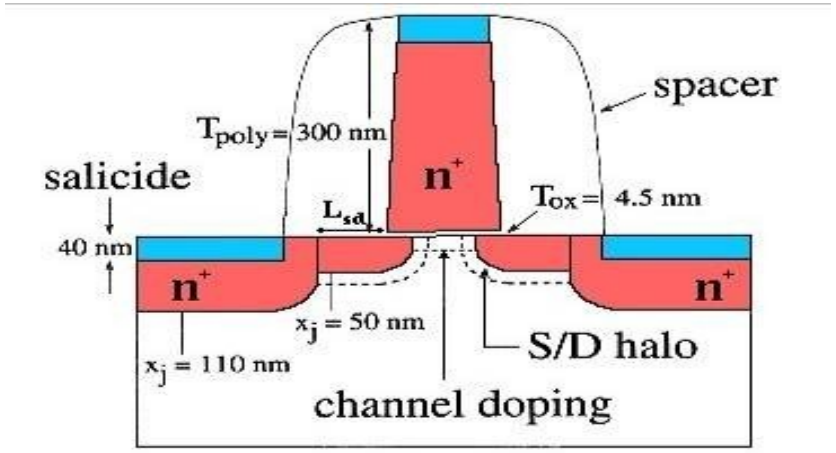
(a) Performance of CNSPACK linear solver for electronic device simulator in modeling 3D devices using unstructured meshes. Memory (in secs) is shown by solid curve with diamonds (in MB) for IBM Thinkpad T42 laptop, INTEL PENTIUM M 740 1.73GHZ .
CPU time is shown by solid curve with circles (per Newton iteration, [Fedoseyev (2009)]). One can see that the dependence on mesh nodes is almost linear for both memory and CPU time, and close to optimal theoretical estimate for CPU time, $T=O(N^{5/4})$ (dashed dark blue curve);

Each node has at least 4 variables: e,h, Ψ,T

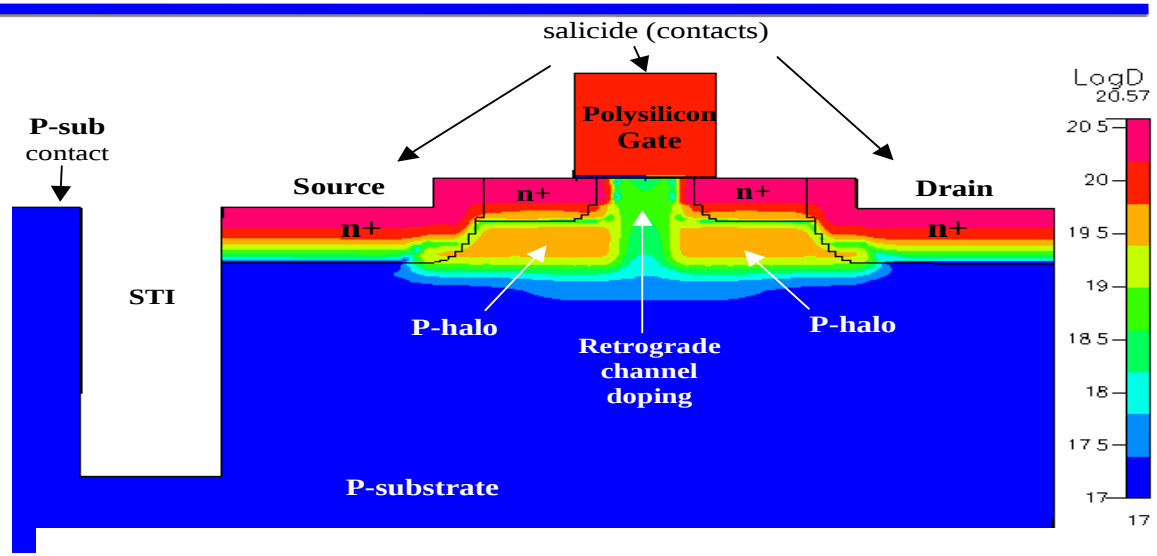


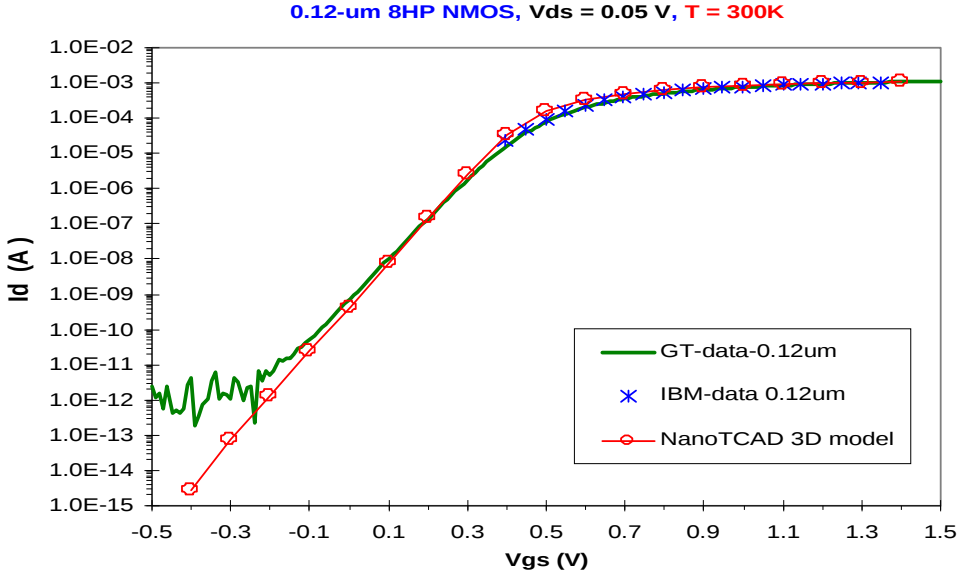
(b) Comparison of CNSPACK software (pink color curve with circles labeled CNSPACK/3D) to other linear solvers for CFD unstructured matrices shows one to two order faster performance of CNSPACK on an Intel P6 200MHz CPU, [Fedoseyev (2001)]

Typical semiconductor device and simulation requirements

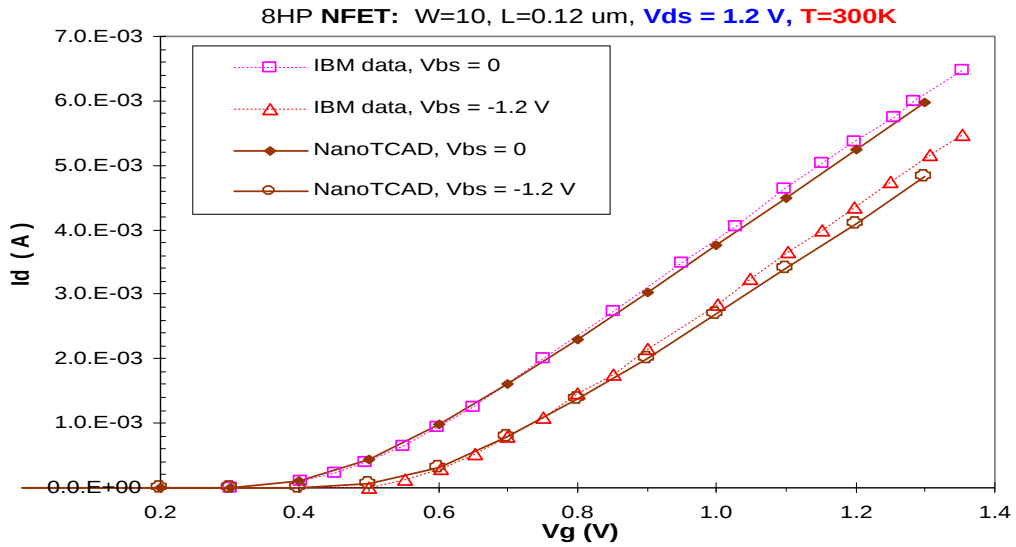


Schematic cross-section of 0.12-µm NMOS device





3D model ID-VGS subthreshold results (at VDS = 0.05V) for NMOSFET compared with Georgia Tech measured data and IBM 8HP Process Manual data



3D model results for NMOSFET (Id-Vg at Vd = 1.2V) compared with IBM 8HP Process Manual data, for two different substrate-to-source biases (Vbs)

Parallel solution methods – algebraic domain decomposition

(1) Algebraic Domain Decomposition with preconditioning:

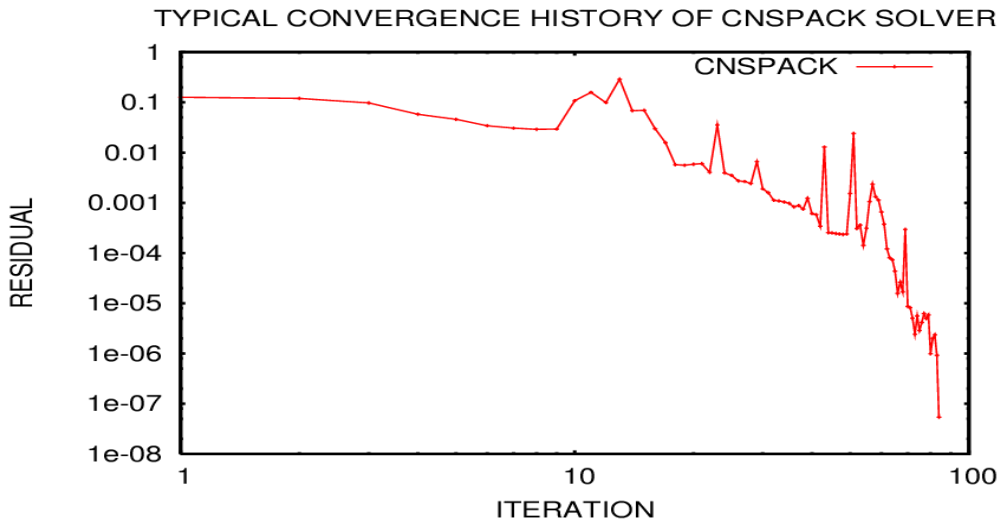
-Cai and Sarkis (1997) proposed a Restricted Additive Schwarz (RAS) preconditioner.

-Other similar approaches and proof of convergence: Frommer (2001), Nabben and Szyld (2001), and Benzi et al (2001)

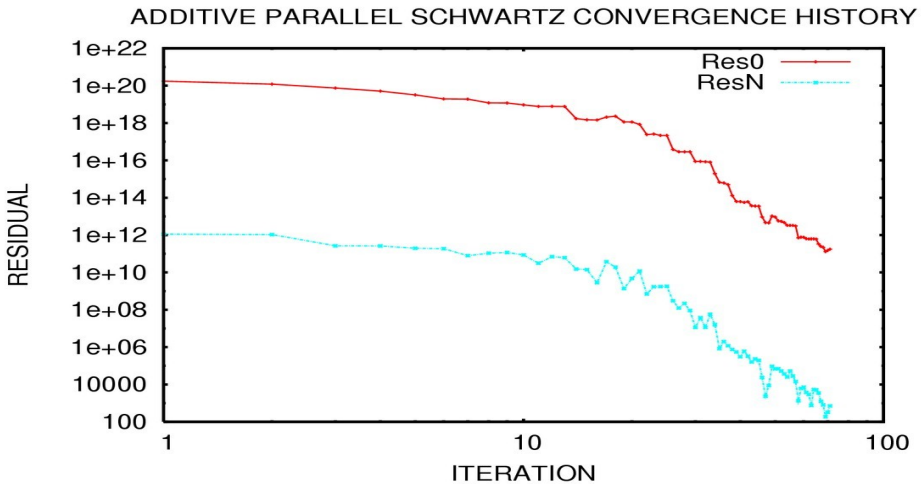
-Preliminary numerical experiments have shown convergence of the method, even for a very small overlapping

-Implementation attempt: RAS with MPI on parallel computer cluster.

*** Slow MPI communications. Long solution times. Development discontinued**



Typical convergence history of preconditioned IDR solver in CNSPACK.



Convergence history of Additive Schwartz preconditioner for 8 processors. Test case 3DMOSFET1. eps=1e-7, iq=20, ns=8.

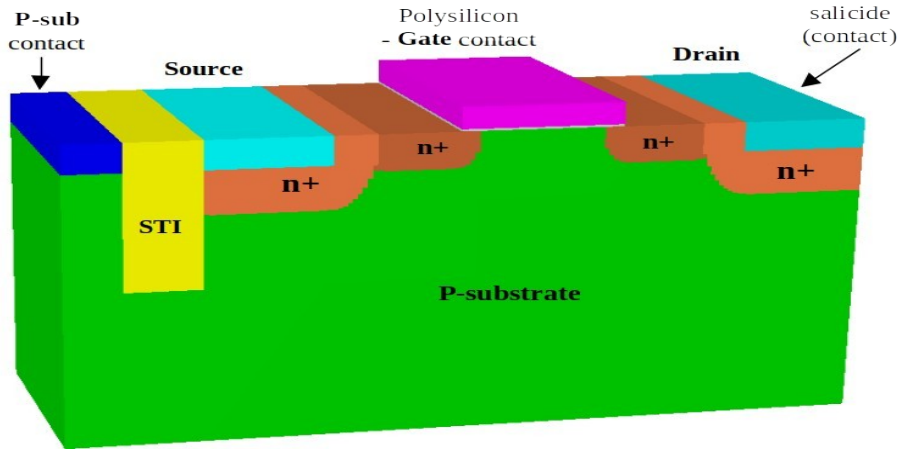
Parallel solution methods and implementation for multi-cores

(2) Multicore processor is a shared memory system : simplifies parallelization in convenient OpenMP environment, and fast communication.

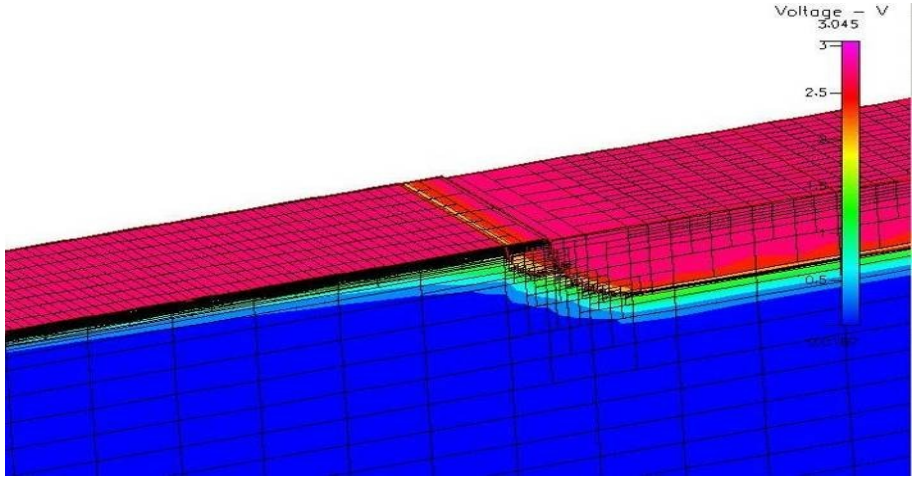
-All the original algorithm has been retained, solution in a single domain

- Implementation of preconditioning and IDR/CGS methods have been done using modified scheme to fit into OpenMP parallelization

- Development started in collaboration with the Dr. Bessonov and successfully completed within two months.



CFDRC 3D model of the 0.12-µm nMOSFET



3D MOSFET problem – details of mesh and electrostatic potential distribution. Overall this 3D bin-tree mesh contains 30K poly-cells resulting in 120K linear equations to be solved in parallel.

Implementation issues for multi-cores

- **CNSPACK algorithm requires both the speed of arithmetic calculations and the throughput of memory accesses.**

- **Because of the computation-memory disbalance of modern processors, the latter property becomes even more important (i.e. the algorithm is memory-bound rather than computational-bound).**

- **The above determines the choice of a computer system used for the implementation (more CPU channels to memory).**

- **On the other hand, there are no more memory-size limitations, so the algorithms can be re-arranged correspondingly in favor of the computational speed.**

Analysis of alternative storage schemes for parallelization

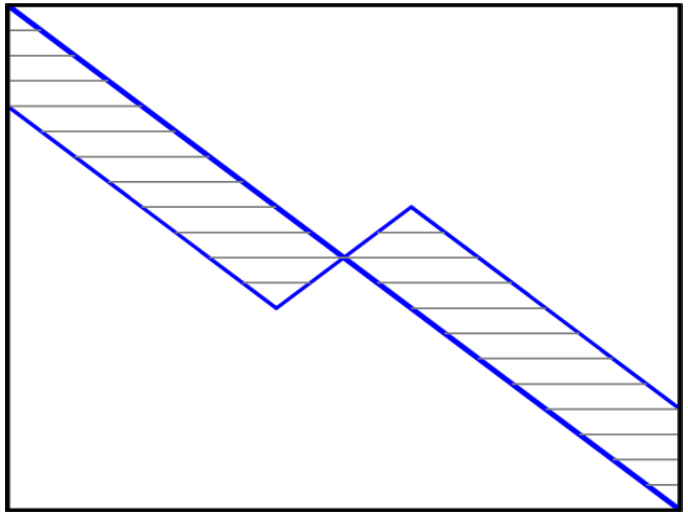
(i)Analysis of alternative storage schemes.

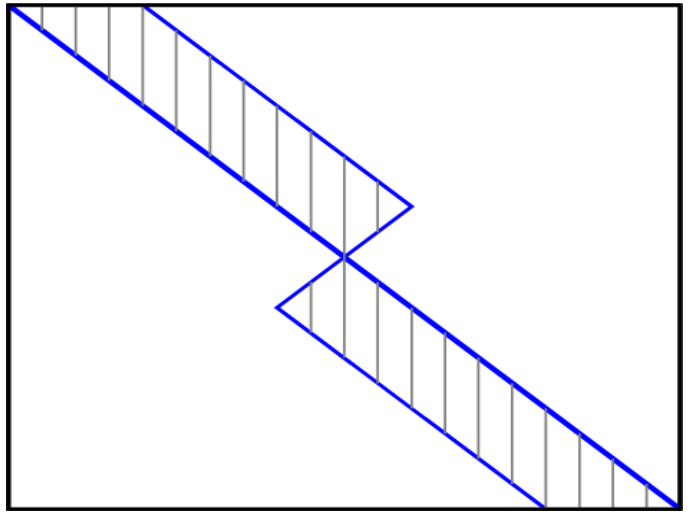
- **Application of row-wise storage scheme for preconditioning was analyzed as a preparation step to the implementation of the block-pipelined method.**

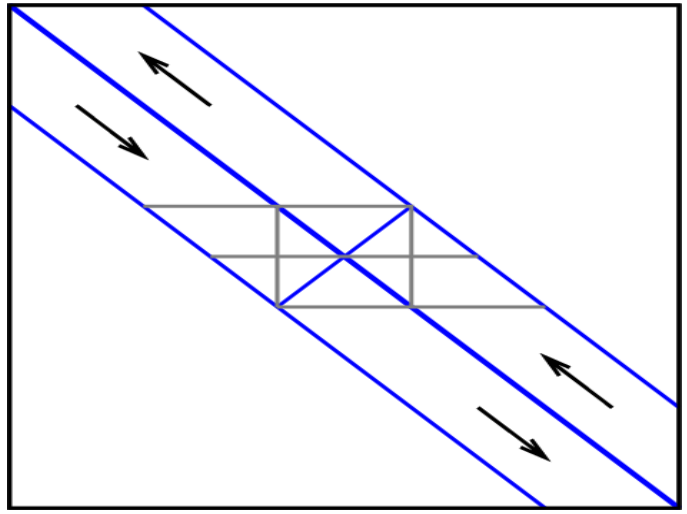
- **Next, the special storage scheme was implemented which stores elements by rows both in lower and upper parts of the original matrix A.**
- **This scheme is very convenient for parallelization of the multiplication routine (CNSMUL) because it makes splitting of the matrix and vectors in the matrix-by-vector expression Y = A X very natural: lines of both matrix parts, L and U, are used to update a single elements in vector Y (while in the standard storage scheme, a sparse group of elements in Y is updated through the processing of a column in U).**
- **As a result, parallelization of CNSMUL for any number of threads becomes possible without any additional care.**

Twisted factorization for ILU

The mentioned row-wise storage scheme was incorporated into the parallel algorithm in a twisted form, which is consistent with the twisted storage scheme of the main matrix.







- **Schematic representation of the main principle of twisted factorization is illustrated. This algorithm is just a way to perform Gauss elimination of unknowns from both sides of a banded matrix. As a result, factors of the matrix decomposition may look as presented on this picture.**

Twisted factorization for ILU (2)

The question is how the central part of the splitting should be organized in order to be able to resolve the factored system.

Analysis has shown that the only way to split the original matrix is to ensure that portrait of each factor is a transpose of its counterpart. These factors will be traditionally named as L and U, despite they are no more "Lower" and "Upper".

It is convenient to represent the decomposition as $K = (L + D)\, D^{-1}\, (D + U + R)$. Here R is the "reverse diagonal" that separates two part of each factor.

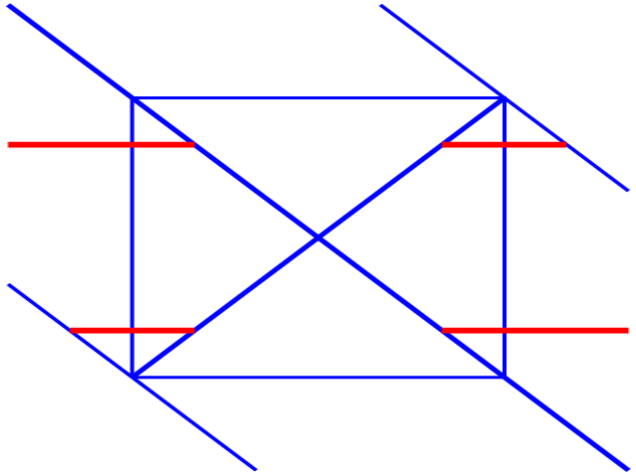
The role of this reverse diagonal is seen on the right part of the figure above, where a preconditioning matrix K is represented as a product of factors. We can distinguish three areas on the matrix portrait: the central part (a square area with adjacent areas on the left and on the right), the first part located above it, and the last part located below it.

Twisted factorization for ILU (3)

Elimination of non-zero elements in the first and in the last parts is straightforward (as in the standard Gauss process) because these parts are formed by simple multiplication of corresponding (first or last) parts of factors L and U.

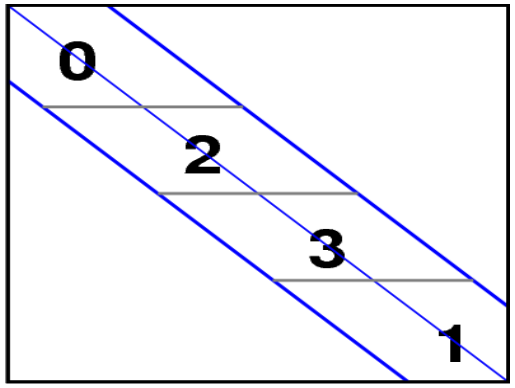
Elimination of non-zero elements in the central part is much more complicated, because it is formed by multiplication of complex central parts of L and U and contributions of two dot products (left row in L by upper column in U. and right row in L by lower column in U) are used in calculation of any elements in this part. Because of this, the central part of the matrix being decomposed can be called as overlap.

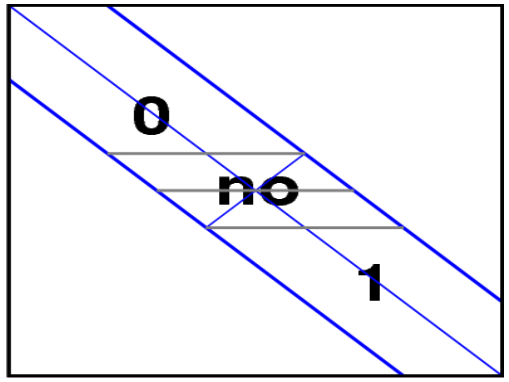
Elimination procedure for the central part: it is necessary to process simultaneously four rows of the matrix (thick red lines), using elements of both the main diagonal and the reverse diagonal for joint update of rows being processed.



Twisted factorization for ILU (4)

Parallelization of twisted factorization algorithm:





- **Parallelization of the matrix-by-vector multiplication (left) is straightforward due to the row-wise storage scheme applied to both parts, L and U: multiplication of each subset of the matrix by the vector is performed independently within corresponding thread.**
- **Parallelization of the preconditioning routine (right) is, in turn, limited by 2 threads. The central part of the matrix (within overlap) can't be parallelized and must be processed sequentially, while the first and the last parts (beyond overlap) are independent and can be executed in parallel both in the forward (elimination) and backward (back-substitution) sweeps.**
- **Size of the central part is approximately equal to half-bandwidth in that place. It means that only about 5% of the current matrix is processed sequentially.**

Advanced block-pipelined parallelization

(ii) Implementation of advanced block-pipelined parallelization method for preconditioning.

- **New advanced block-pipelined parallelization method of ILU-preconditioning was developed and implemented. The idea of this method is to split a matrix half-band into pairs of adjacent trapezoidal blocks that have no mutual data dependences and can therefore be processed in parallel.**
- **As a result, parallelization of the preconditioning routine for 4 threads was implemented. For this method, new blocked storage scheme was designed.**

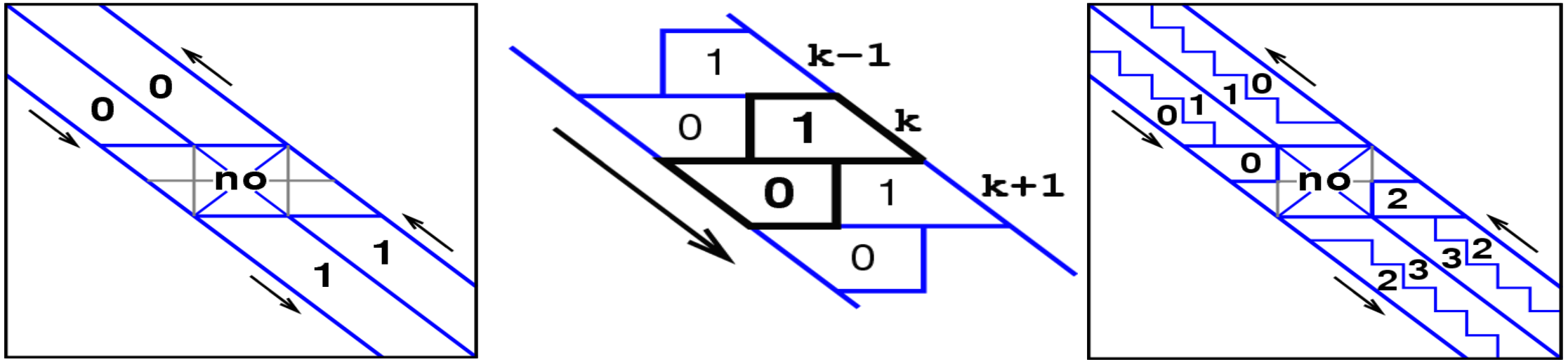


Illustration of implementation of pipelined parallelization of the lower part L of a matrix during the elimination process (forward sweep). Backsubstitution process (backward sweep) is implemented similarly.

Parallel performance of new algorithms

Test matrix 1

(MOSFET 1 & 2) -produced by the discretization of a 3D Finite element grid with irregular structure.

Main characteristics of this matrix are:

- number of equations – 77571;
- number of non-zero element positions in each part (L or U) – 636626;
- average number of non-zero elements in a row (column) – 18.2;
- maximal half-width – 4658.

Thus, this matrix is very sparse, with non-uniform sparsity pattern and variable bandwidth.

Additionally, the matrix is ill-conditioned that leads to irregular convergence behaviour: number of iterations varies from 600 to 900 (to 10^{-25}) and depends on many random factors (such as order of arithmetic operations etc.).

Parallel performance of new algorithms

Test matrix 2

CFD matrix - produced by the discretization of a regular 3D grid with the following characteristics:

- number of equations – 302500;
- number of non-zero elements in each part – 13385919;
- average number of non-zero elements in a row (column) – 44;
- maximal half-width – 2608;
- 25% of diagonal are zeros.

This matrix is much dense and much more regular, with almost uniform distribution of elements throughout its band. The latter property will be very important for efficient block-pipelined parallelization (considered in the next section).
Additionally, this matrix is better conditioned, that allows using single-precision (32 bit) values for storing both the original matrix A and the preconditioning matrix K.

As a result, the convergence behaviour of this matrix is more regular: it takes 150 to 170 iterations to converge (to 10^{-15}), with less dependence on secondary factors.

Parallel performance of new algorithms- MOSFET

Parallelization performance results of the new algorithm (CNSPACKZ) are presented in the table, in comparison with the previous parallel code (CNSPACKX).
These results represent performance of the target computer system Intel Core i7-920 (2.66 GHz, 3 memory channels DDR3-1333).
Only the iterative part of the algorithm was measured (algorithm iterates until the residual reduces by 10^{25} times for the first matrix (MOSFET) and by 10^{15} times for the second one (Generalized Navier-Stokes).

1-st matrix (MOSFET) **real*8**	original CNSPACK	new CNSPACKX		new CNSPACKZ
	1 thread	2 threads	4 threads	4 threads
time of one iteration	5.73 ms	3.60 ms	3.15 ms	2.69 ms
acceleration of one iteration	1.00	1.59	1.82	2.13

Parallel performance of new algorithms- CFD

2-nd matrix (CFD) **real*8**	original CNSPACK	new CNSPACKX		new CNSPACKZ
	1 thread	2 threads	4 threads	4 threads
time of one iteration	89.1 ms	55.6 ms	48.1 ms	41.0 ms
acceleration of one iteration	1.00	1.60	1.85	2.17

Results for two matrices look similar.

- **In both cases, additional performance gain of 17% was achieved with the new block-pipelined method (CNSPACKZ).**
- **However, with the first matrix (MOSFET) the new method additionally benefits from removal of non-zero elements (they comprise about 5% of the preconditioning matrix).**
- **Lower performance results for the first matrix follow from its properties – low density and irregular structure, resulting in bad load balance between blocks "0" and "1" in pipelined parallelization.**

Parallel performance of new algorithms- CFD (32bit)

2-nd matrix (CFD) **real*4**	original CNSPACK	new CNSPACKX		new CNSPACKZ
	1 thread	2 threads	4 threads	4 threads
time of one iteration	89.1 ms	51.3 ms	41.6 ms	32.05 ms
acceleration of one iteration	1.00	1.74	2.14	2.78
real*4 speed ratio to **real*8**	1.00	1.08	1.16	1.28

The last table presents results for the second matrix (CFD) using single precision (32-bit) values for storing both the matrix A and the preconditioning matrix K.

- Single precision storage can reduce total size of arrays and, consequently, memory access traffic thus increasing performance of a memory-bound algorithm.
- However, using mixed precision arithmetic incurs additional overhead of data conversion.
- As a result, effect of saving memory traffic becomes visible only in parallel runs (curiously, single-precision results for one thread absolutely coincide with double-precision results in our case).

Conclusions

The universal fast linear solver was developed for scientific and industrial applications.
Performance of CNSPACK solver is quite satisfactory, with minimal storage and CPU time for a large variety of problems.

TCAD using CNSPACK is the only industrial parallel solver for semiconductor applications.

Performance of the parallel algorithm was evaluated on the target computer system (Intel Core i7-920) using two test unstructured matricies for MOSFET and CFD problems.

On a dual-processor shared memory system with individual memory controllers in each processor, the new method will gain much more owing to the doubled memory throughput limitation – expected acceleration is 3.5 or more.

Contact information:
Alex Fedoseyev,
Email: af@ultraquantum.com

References

[1] A. Fedoseyev, O. Bessonov. Iterative solution of large linear systems for unstructured meshes with preconditioning by high order incomplete decomposition. Computational Fluid Dynamics Journal, V.10, No.3, 2001, 299-303. http://www.ipmnet.ru/~bess/bess-cfdj.pdf

[2] G. Accary, O. Bessonov, D. Fougere, S. Meradji, D. Morvan. Optimized parallel approach for 3D modelling of forest fire behaviour. Lecture Notes in Computer Science, 4671, 2007, pp.96-102. http://www.ipmnet.ru/~bess/bess-pact2007.pdf

[3] G. Accary, O. Bessonov, D. Fougere, K. Gavrilov, S. Meradji, D. Morvan. Efficient parallelization of the preconditioned Conjugate gradient method. Lecture Notes in Computer Science, 5698, 2009, pp.60-72. http://www.ipmnet.ru/~bess/bess-pact2009.pdf

[4] A. I. Fedoseyev, M. Turowski, L. Alles, and R. A. Weller, Accurate Numerical Models for Simulation of Radiation Events in Nano-Scale Semiconductor Devices, Math. and Computers in Simulation, 2007, doi:10.1016/j.matcom.2007.09.013.

[5] P. Wesseling, P. Sonneveld (1980). Numerical experiments with a multiple grid and a preconditionned Lanczos type methods. Lecture Notes in Mathematics, 771, Berlin: Springer, 1980, 543-562.

[6] D. S. Kershaw (1978). The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. J. Comput. Phys., 2, 43-65.

[7] N.E. Gibbs, W.G. Pool, Jr. and P.K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix, SIAM J. Numer. Anal., 13 (1976) 236-250.

[8] O. A. Bessonov and A. I. Fedoseyev, Parallelization of the preconditioned IDR solver for modern multicore computer systems, AIP Conf. Proc. 1487, 314 (2012); doi: 10.1063/1.4758973

[9] A. Fedoseyev and O. Bessonov (2001) Iterative Solution Of Large Linear Systems For Unstructured Meshes With Preconditioning By High Order Incomplete Decomposition, Japan Society of CFD/ CFD Journal 10(1), 299-303, 2001.

[10] O. A. Bessonov and A. I. Fedoseyev (2011), Efficiency analysis and parallelization of the CNSPACK implementation of a preconditioned CGS solver for modern multicore computer systems, Preconditioning 2011, May 16-18, 2011, Bordeaux, France.

[11] Fedoseyev A., Alexeev B.V., Generalized hydrodynamic equations for viscous flows-simulation versus experimental data,in AMiTaNS-12, American Institute of Physics AIP CP 1487, 2012, pp.241-247, Ed. M.Todorov..

[12] Fedoseyev A., Griaznov V., Ouazzani J., Simulation of rarefied hypersonic gas flow and comparison with experimental data II, Proc. AMITANS-2022 Conf., AIP CP 2953, 2023, Ed. M.Todorov.