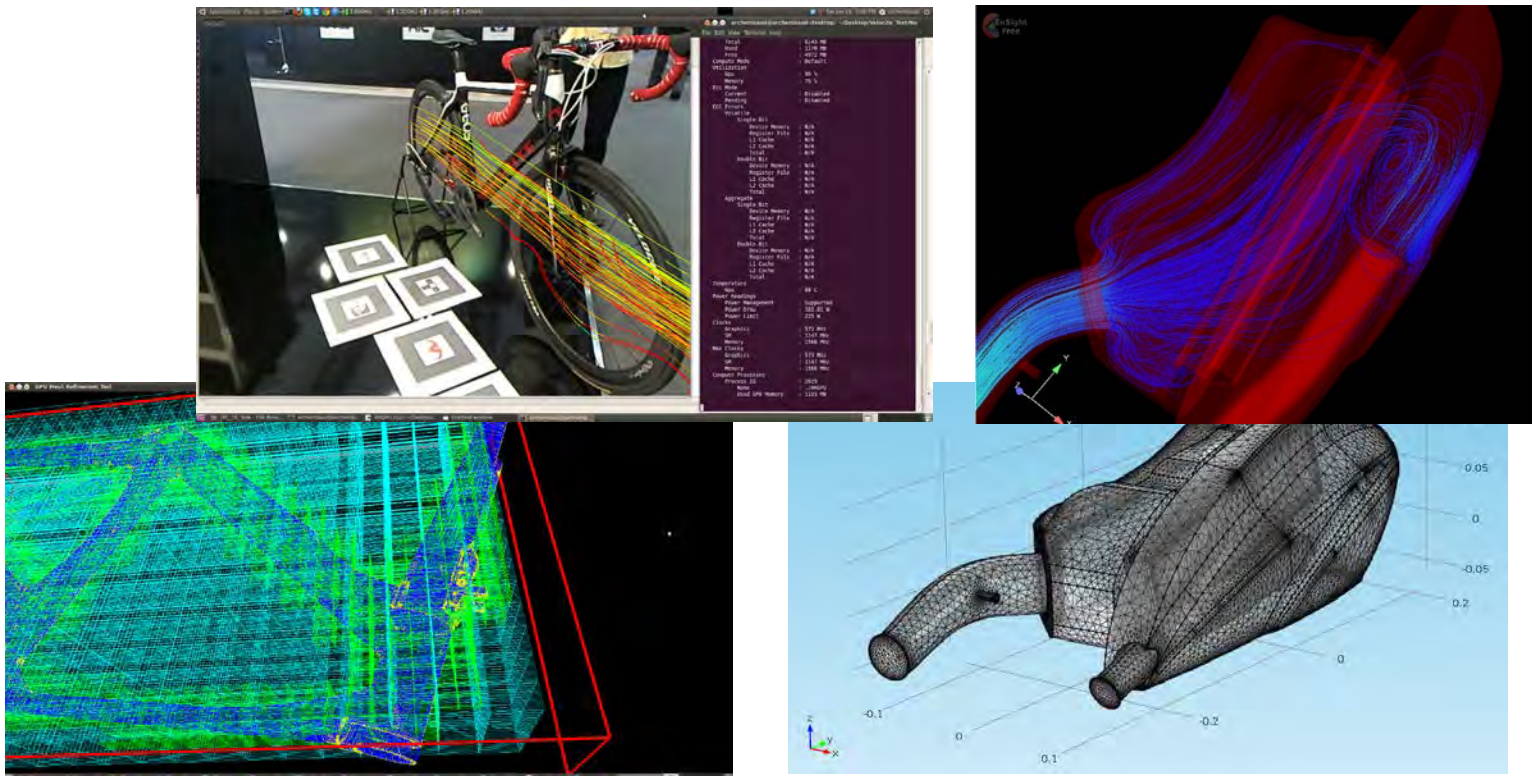
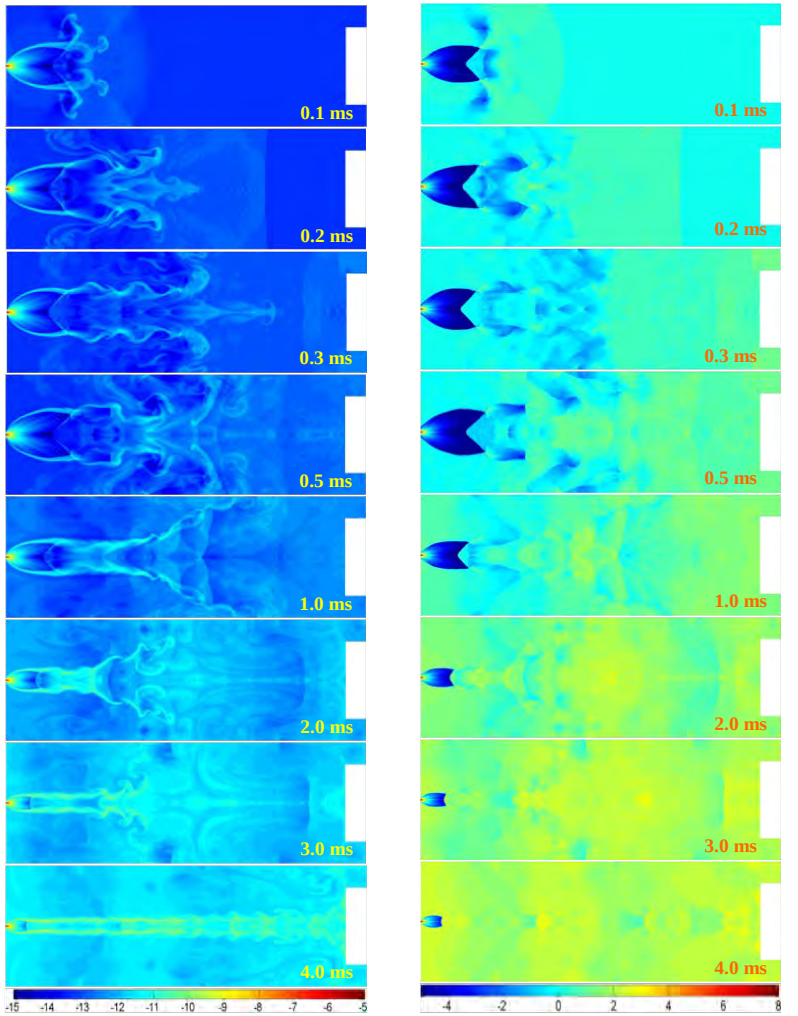
GPU PARALLELIZATION OF A THREE-DIMENSIONAL RIEMANN SOLVER USING UNSTRUCTURED TETRAHEDRAL GRIDS

Prof. Matthew Smith,
Department of Mechanical Engineering,
National Cheng-Kung University, Taiwan.

BACKGROUND

- Why use parallel computing?



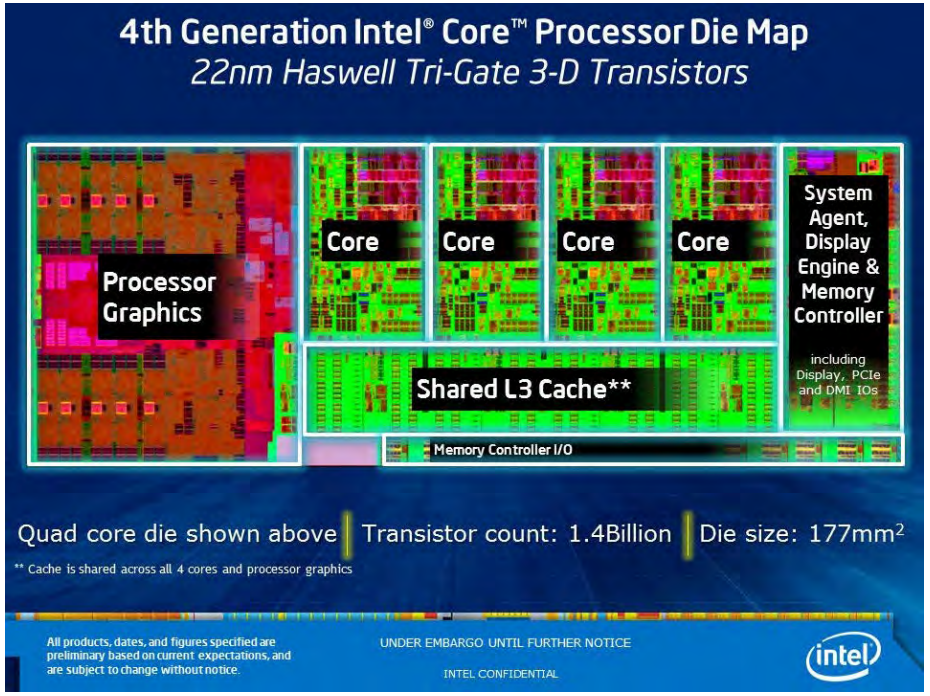


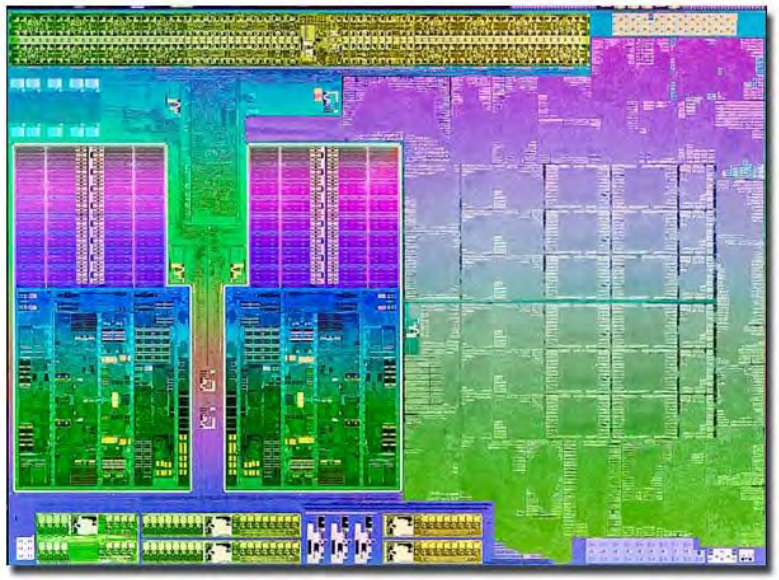
Response 1: Because we have applications which require it….

BACKGROUND

- Why use parallel computing?

Coprocessors





CPU's

Response 2: Because you have to try hard to find a system which is <u>not</u> capable of parallel computing.

BACKGROUND

- Why use parallel computing?

OpenMP

```
    while ( error > tol && iter < iter_max ) {
75.     error = 0.f;

        #pragma omp parallel for shared(m, n, Anew, A)
        for( int j = 1; j < n-1; j++) {
            for( int i = 1; i < m-1; i++ ) {
80.             Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }
85.
        #pragma omp parallel for shared(m, n, Anew, A)
        for( int j = 0; j < n-1; j++) {
            for( int i = 0; i < m-1; i++ ) {
                A[j][i] = Anew[j][i];
90.         }
        }

        if(iter % 100 == 0) printf("%d, %0.6fn", iter, error);

95.     iter++;
    }
```

OpenMP / Posix

OpenACC

Directives for Accelerators

```
        #pragma omp parallel for shared(m, n, Anew, A)
        #pragma acc kernels
75.     for( int j = 1; j < n-1; j++)
        {
          for( int i = 1; i < m-1; i++ )
          {
Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
80.    + A[j-1][i] + A[j+1][i]);
error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
          }
        }

85.  #pragma omp parallel for shared(m, n, Anew, A)
        #pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
          for( int i = 1; i < m-1; i++ )
90.       {
A[j][i] = Anew[j][i];
          }
        }

95.     if(iter % 100 == 0) printf("%5d, %0.6fn", iter, error);

        iter++;
    }
```

OpenACC

NVIDIA CUDA

C with CUDA extensions

```
__global__
void saxpy(int n, float a,
           float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int N = 1<<20;
cudaMemcpy(x, d_x, N, cudaMemcpyHostToDevice);
cudaMemcpy(y, d_y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, x, y);

cudaMemcpy(d_y, y, N, cudaMemcpyDeviceToHost);
```

CUDA

```
#include <mpi.h>;

int main(int argc, char** argv) {
  // Initialize the MPI environment
  MPI_Init(NULL, NULL);

  // Get the number of processes
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // Get the rank of the process
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  // Get the name of the processor
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int name_len;
  MPI_Get_processor_name(processor_name, &name_len);

  // Print off a hello world message
  printf("Hello world from processor %s, rank %d"
         " out of %d processors\n",
         processor_name, world_rank, world_size);

  // Finalize the MPI environment.
  MPI_Finalize();
}
```

OpenMPI / MPI

Response 3: Because it's easier than its ever been before.

Mostly.

Well....

BACKGROUND

- Why use parallel computing?

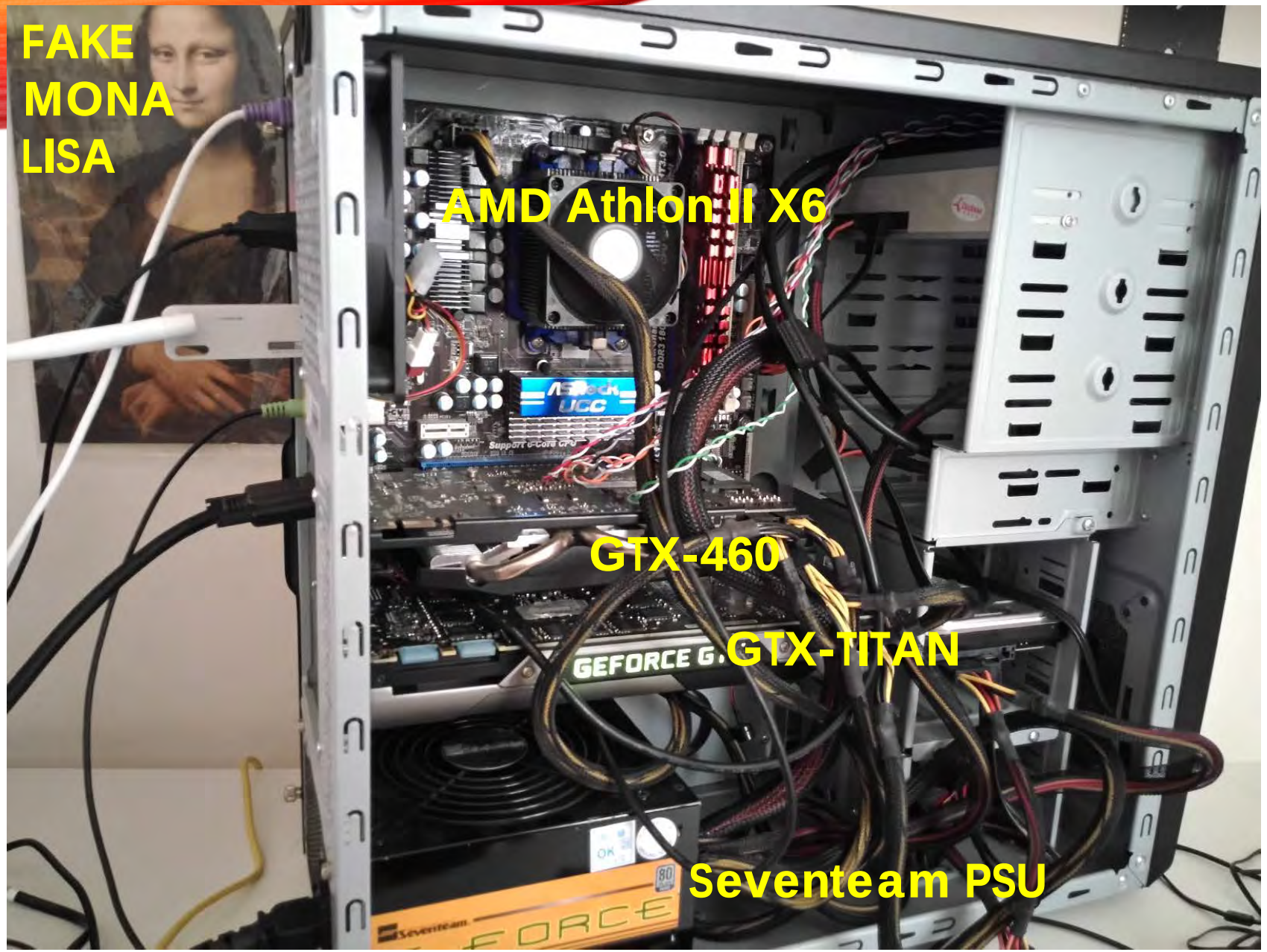
Intel i3
$4900 NTD

AMD A10-6800K
$4400 NTD

GTX-650 Ti
$4000 NTD

HD-7970
$8000 NTD

Response 4: Because it's more affordable than ever before.



BACKGROUND

- There is no longer an excuse for not having a parallel capable computer.
- It's very possible to own a computer capable of 1 TFlop for less than $800 USD.

BACKGROUND

- The pursuit of high performance computing continues in Taiwan.

- Current methodologies (in my lab) have focused on the use of:
  - Conventional (distributed) parallel computing
    - MPI-based distributed computing,
    - Hybrid shared-distributed parallel paradigms,
  - The Intel Phi device,
  - The Kepler (K20, Titan) and Fermi (M2070, C2075) Tesla Computing GPU's.

- Today's talk will focus on the GPU. Mostly.

BACKGROUND

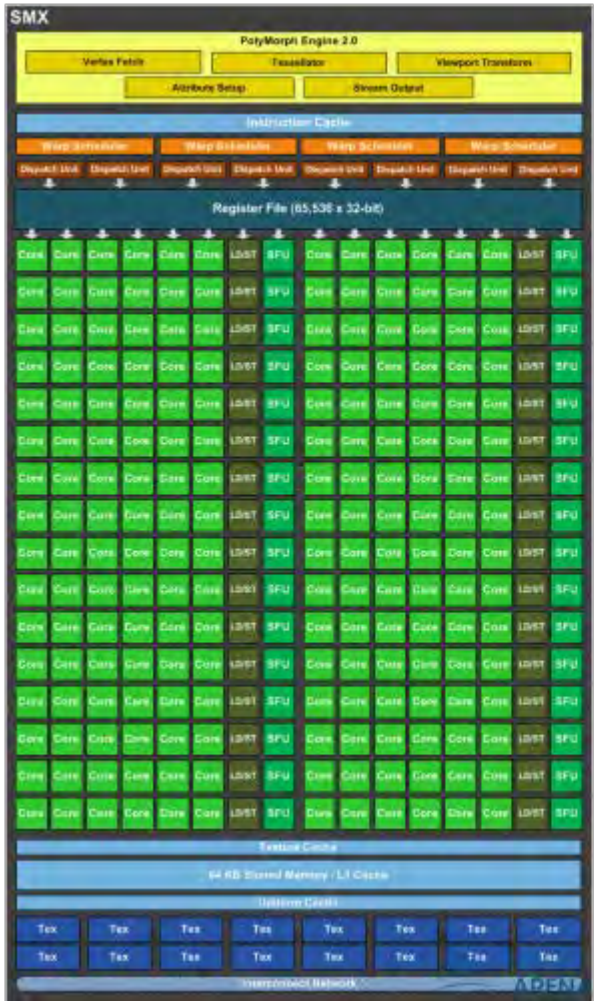
- This research is supported by the presence of two supercomputing clusters:
  - Formosa 5 @ NCHC: < 80 nodes, each with dual Xeon CPU's and 3x Tesla (Fermi) M2070. Infiniband interconnect (40GB)
  - A small phi cluster containing several Intel Phi devices over multiple computing nodes. (Also located @ NCHC)
- This is in addition to multiple test platforms located within NCKU's High Performance Heterogeneous Computing Laboratory (all of Nvidia, AMD, Intel Coprocessors and Accelerators).

Formosa 5

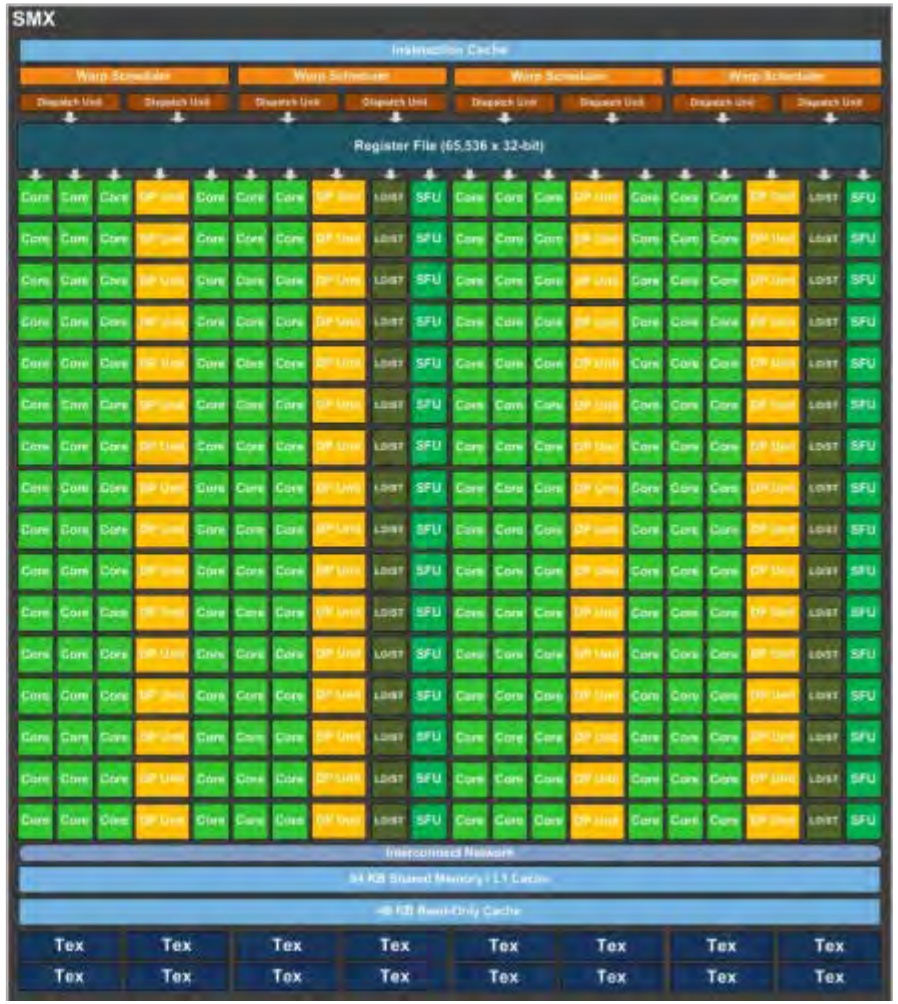
NARLabs
National Applied Research Laboratories
National Center for High-performance Computing

GPU ARCHITECTURE

Kepler for graphics



Kepler for compute



- The modern GPU contains a large number of independent streaming cores, stored separately in multiple SMX's (SMP's).
- Each core is capable of performing relatively simple operations – unlike modern CPU cores.
- Communication is possible using shared and global memory – but should be discouraged.

VECTOR SPLIT FVM COMPUTATION

- Our previous strategy for obtaining high degrees of performance on the GPU device was through the use of Vector Split Finite Volume fluxes, where the flux between cells *i* and *i+1* can be written in the general form:

$$F_{i+\frac{1}{2}} = F_i^+\left(x_i + \frac{\Delta x}{2}\right) + F_{i+1}^-\left(x_{i+1} - \frac{\Delta x}{2}\right)$$ For example….

- We've developed several low dissipation versions of such schemes:

$$F^+_{\eta,QDS} = \int_0^\infty (v'+u)\eta^+ f(v')dv' \approx \sum_{j=1}^{N} H_S(v_j) v_j w_j \eta_j^+$$ Smith et al., JCP, 2008, ParCFD 2009, 2010.

$$F^+_{UEFM} = \sum_{i=1}^{N} w_i \int_{-\bar{v}}^{a} \frac{c}{2a_i} \left\{ \rho \quad \rho(c+\bar{v}) \quad \rho\left(\left(\frac{c+\bar{v}}{2}\right)^2 + E_{in}\right) \right\} dc$$ Smith et al., CMES 2010, C&F, 2013.

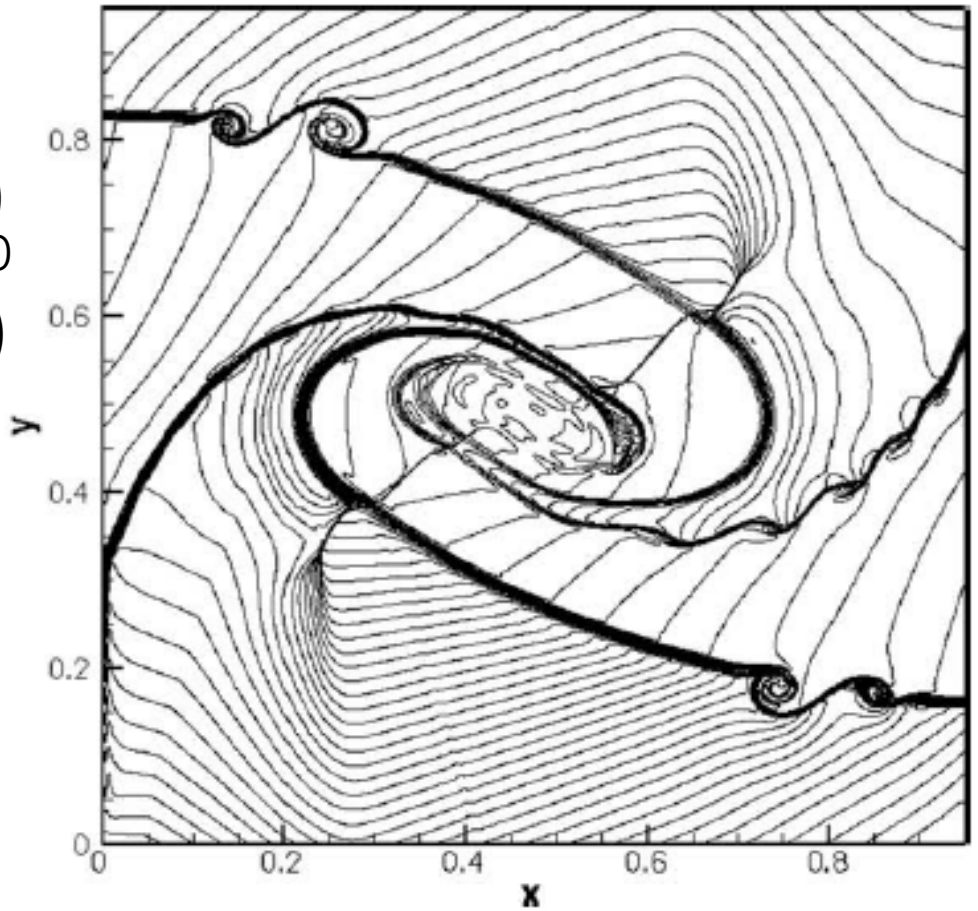
$$F^+ = F_L\left(\frac{D\chi_L + 1}{2}\right) + DU_L a_L\left(\frac{1-\chi_L^2}{2}\right)$$ Kuo et al., C&F, 2012, Smith et al. C&F 2013.

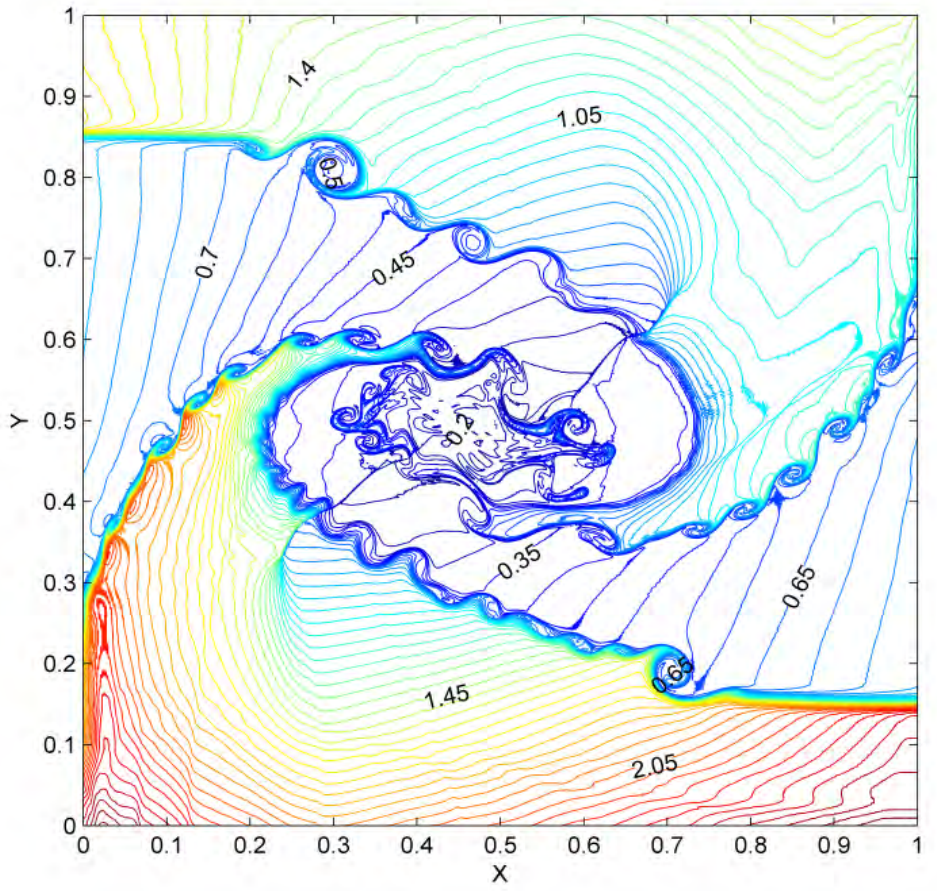
VECTOR SPLIT FVM COMPUTATION

Koren Limiter (3^{rd} order) 1000x1000

Roe (FDS) Fluxes

Double Precision





MINMOD Limiter (2^{nd} order) 960x960

SHLL (FVS) Fluxes

Single Precision

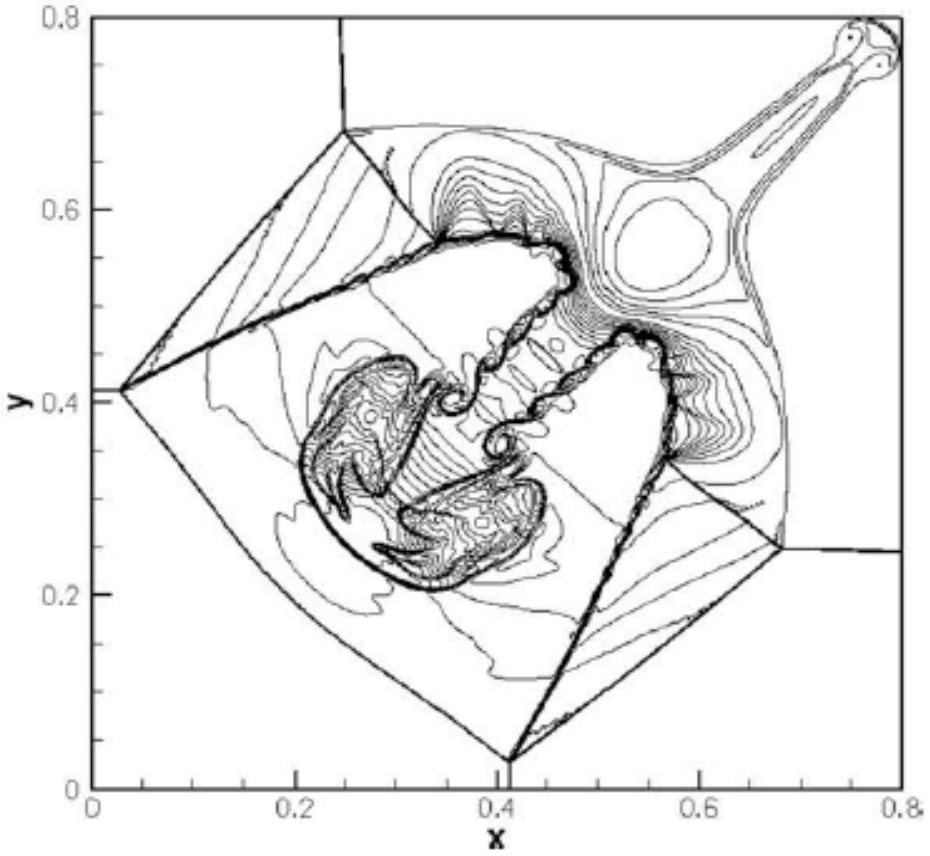
Čada. M., M. Torrilhon., "Compact third-order limiter functions for finite volume methods," *J. Compt. Phys.*, Vol.228, pp. 4118–4145, 2009.

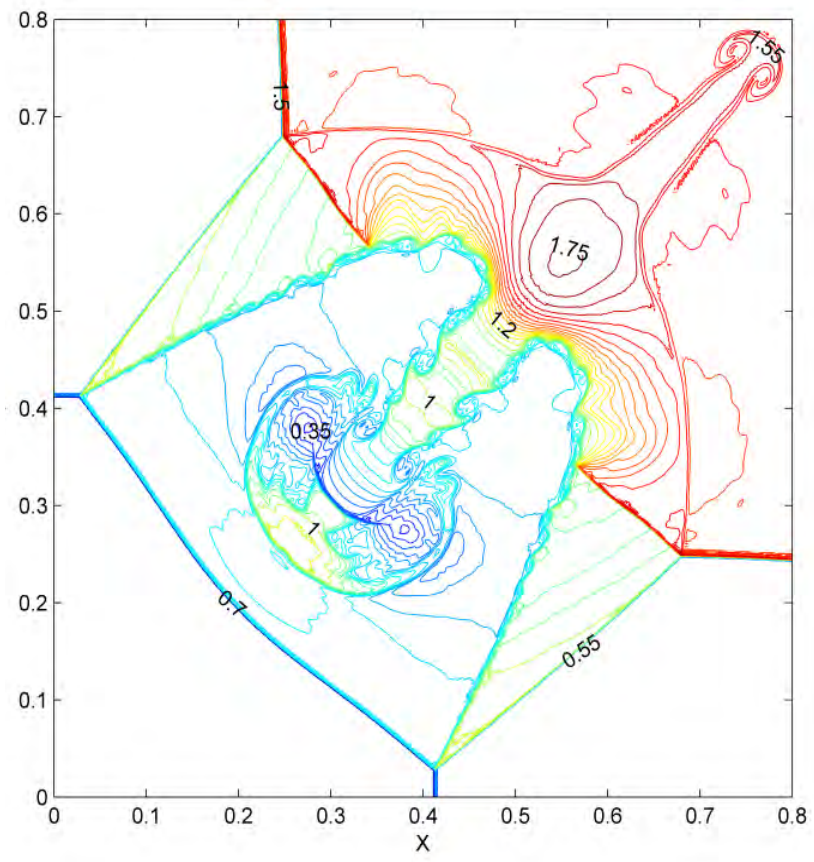
VECTOR SPLIT FVM COMPUTATION

Koren Limiter (3^{rd} order) 1000x1000

Roe (FDS) Fluxes

Double Precision





MINMOD Limiter (2^{nd} order) 960x960

SHLL (FVS) Fluxes

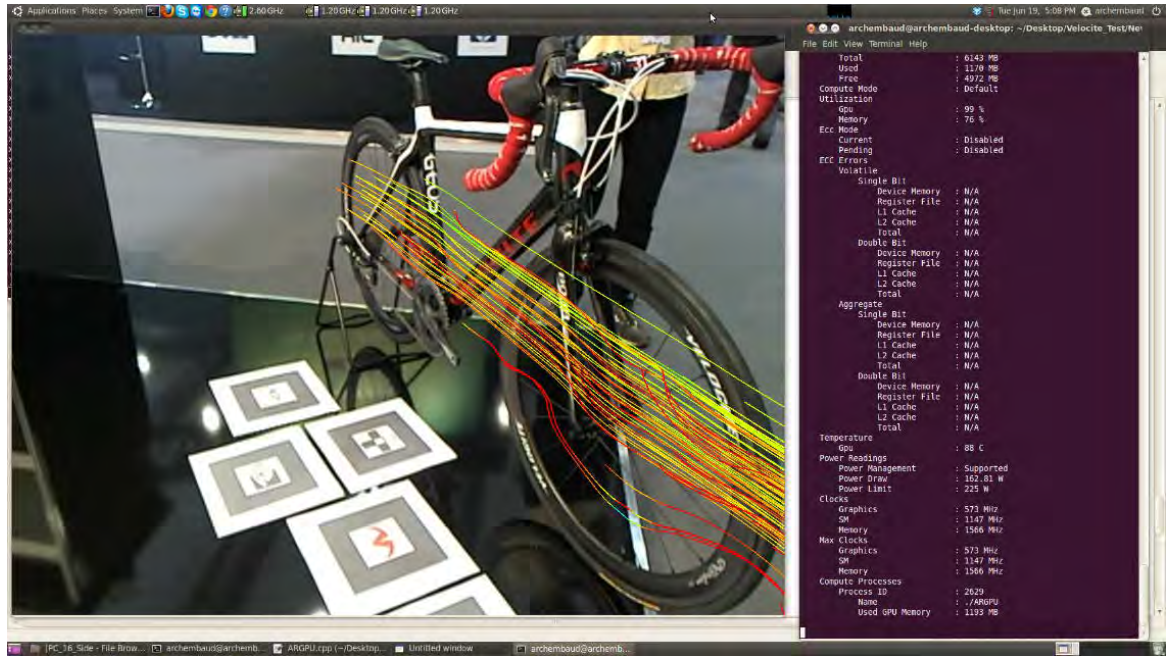
Single Precision

Čada. M., M. Torrilhon., "Compact third-order limiter functions for finite volume methods," *J. Compt. Phys.*, Vol.228, pp. 4118–4145, 2009.

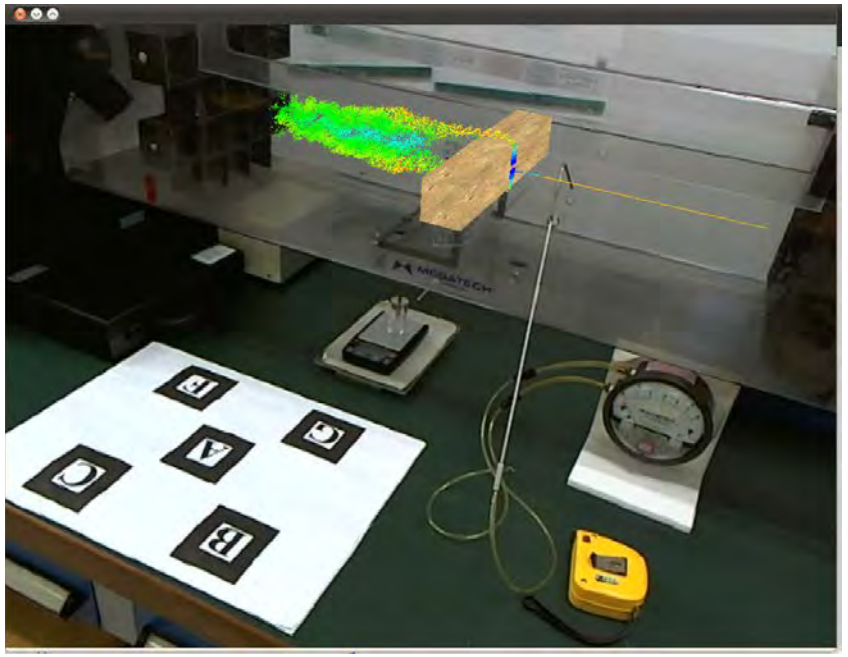
GPU IMPLEMENTATIONS

- The development of the MOCVD solver platform continued from the interactive real-time computation platform using Augmented Reality.

NARLabs
National Applied Research Laboratories
National Center for High-performance Computing



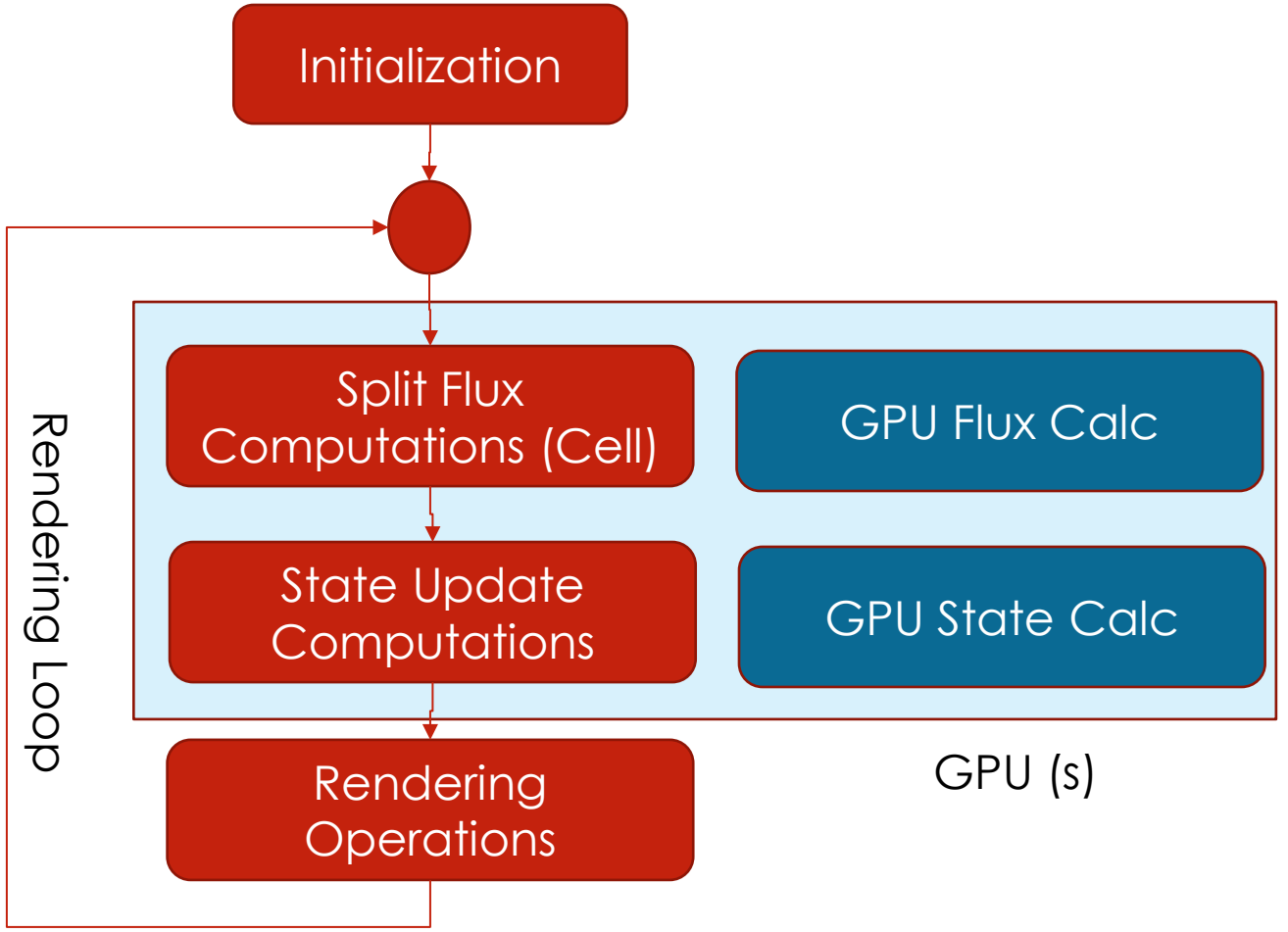
Demonstrations at SC11, ISC12, GTC'13.



Demonstrations at NTHU

GPU IMPLEMENTATIONS

- Real-time computation and visualization on the GPU is simple using OpenGL.
- This tool allows the engineers to see the flow field evolve in real-time, adjust boundary conditions in real-time or change operating conditions.
- Hence, the key to the approach is the focus on the main (OpenGL) rendering loop.



GPU IMPLEMENTATIONS

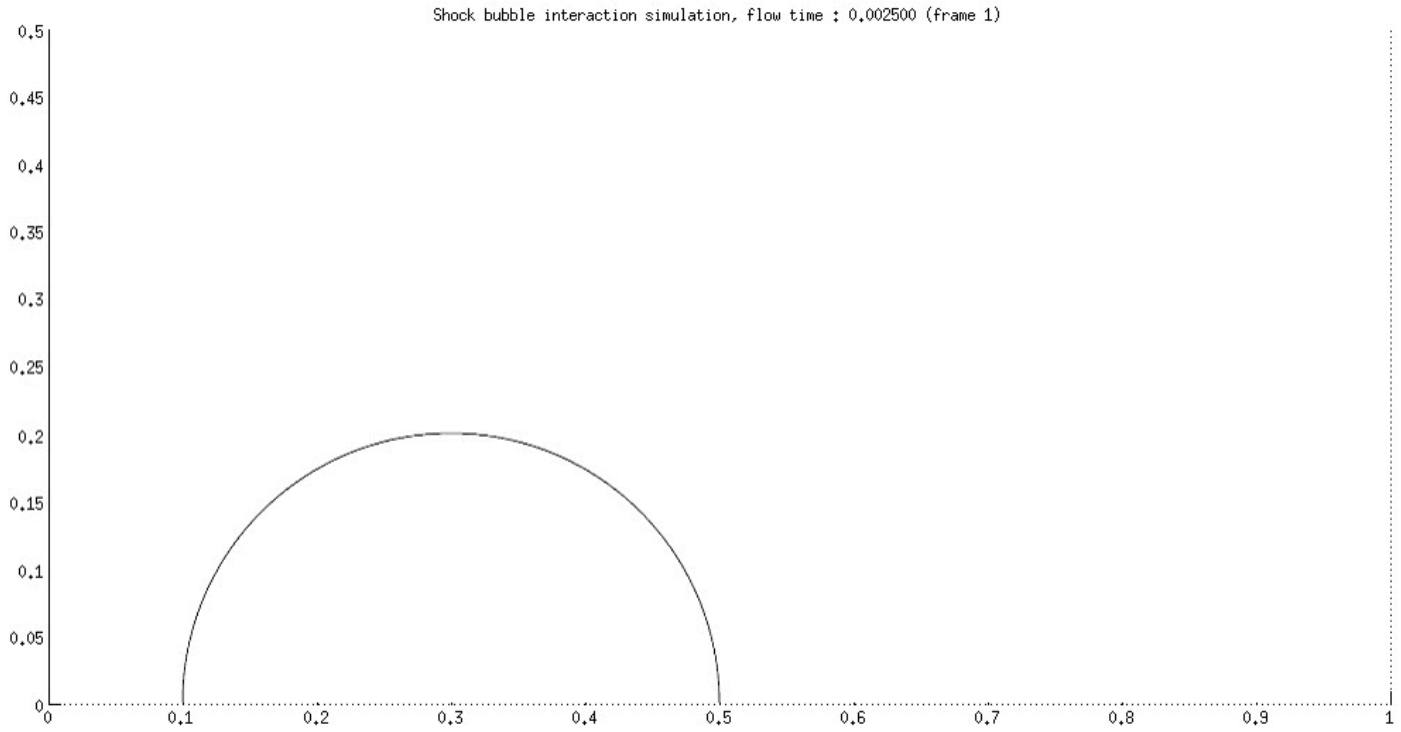
- The use of vector split fluxes for explicit Finite Volume Method (FVM) computation allows very high degrees of performance (single precision)

Device	Test 1 (sec)	Test 2 (sec)	Test 3 (sec)	Average computation time (sec)	Average speed-up (times)
CPU	75.41	75.41	75.38	75.4	1.0
GPU	0.26	0.26	0.26	0.26	290

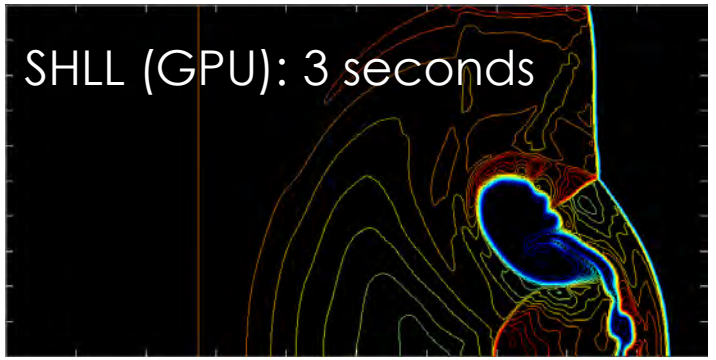
*Results shown for –O3 optimization on a regular structured grid with a single core.

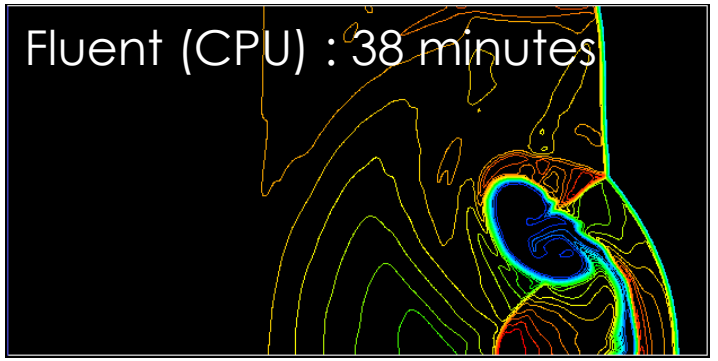
Computations performed on a GTX-Titan for a small test case.

BACKGROUND



Shock Bubble Simulation using 32 C1060 GPU Devices (ParCFD 2010, SC10)

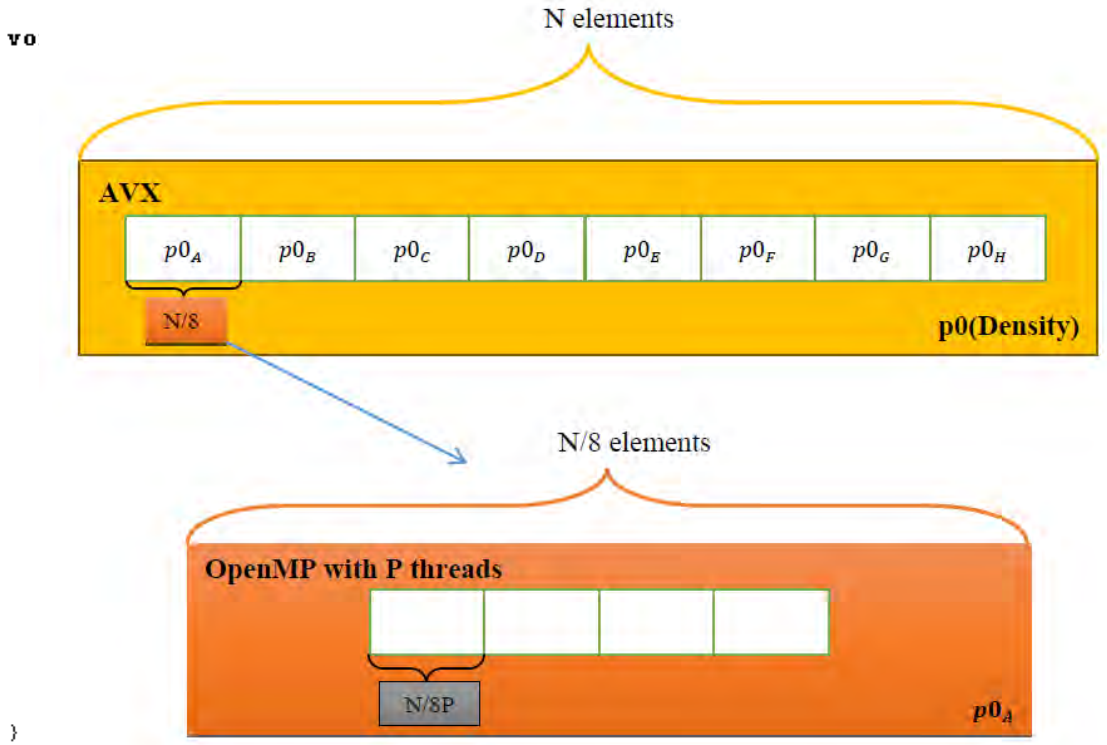




Single GPU performance (GTX-590)

(SIDE NOTE)

- In case you were wondering – yes, you can go faster on the CPU through explicit AVX-OpenMP parallelization using AVX Intrinsic functions.



Computational time (sec)	OMP + AVX (10 cores)	Single Core (SSE, -O3)	Single Core (Default Optimization)	Speedup A	Speedup B
Flux Calculations Only	0.5677	47.252	66.76	~83x	~117x
Primitive Calculations Only	0.3717	24.5832	35.99	~66x	~96x
State Update Calculations Only	1.0403	14.414	23.903	~13x	~23x

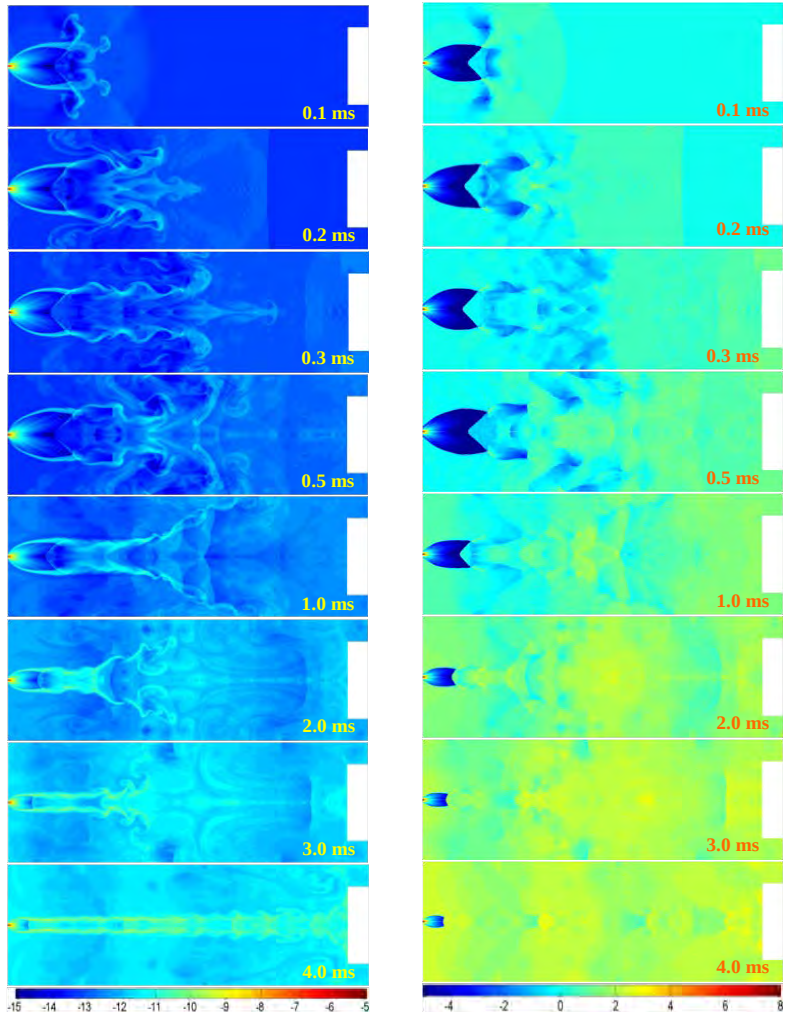
- Explicitly applying multi-level parallel computing can accelerate some workloads by hundreds of times…

CURRENT MOTIVATION

- The motivation for this research is three-fold:
  - Investigations into novel numerical schemes designed to take advantage of the unique hardware architectures currently available,
  - Applications in education – both at graduate and undergraduate levels,
  - Pressure from local industry and academia for increased capacity to predict and understand behavior of gas flows in complex industrial processes.

CURRENT MOTIVATION

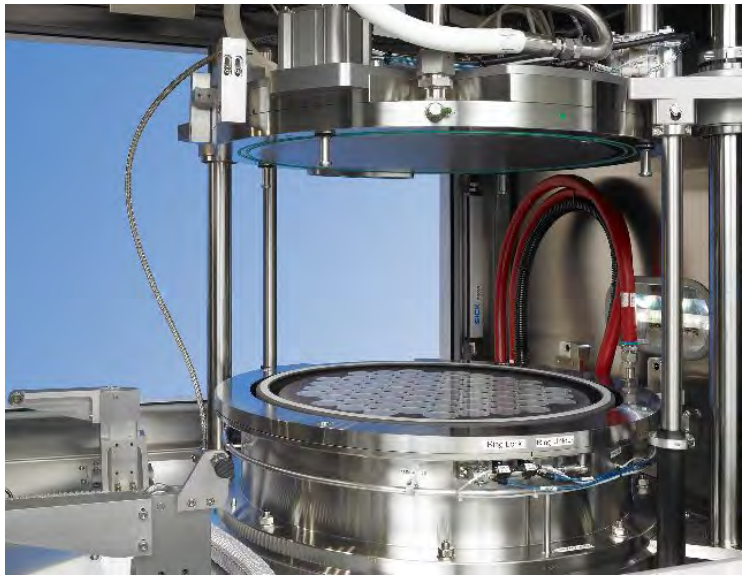
- Taiwan plays a key role in the development and application of various Chemical Vapor Deposition (CVD) technologies.
- We have several companies in Taiwan both:
  1. Making extensive use of the CVD process, and
  2. Researching and Developing new CVD process technologies.
- Currently PI of several NSC/MOST projects related to parallel simulation of CVD.

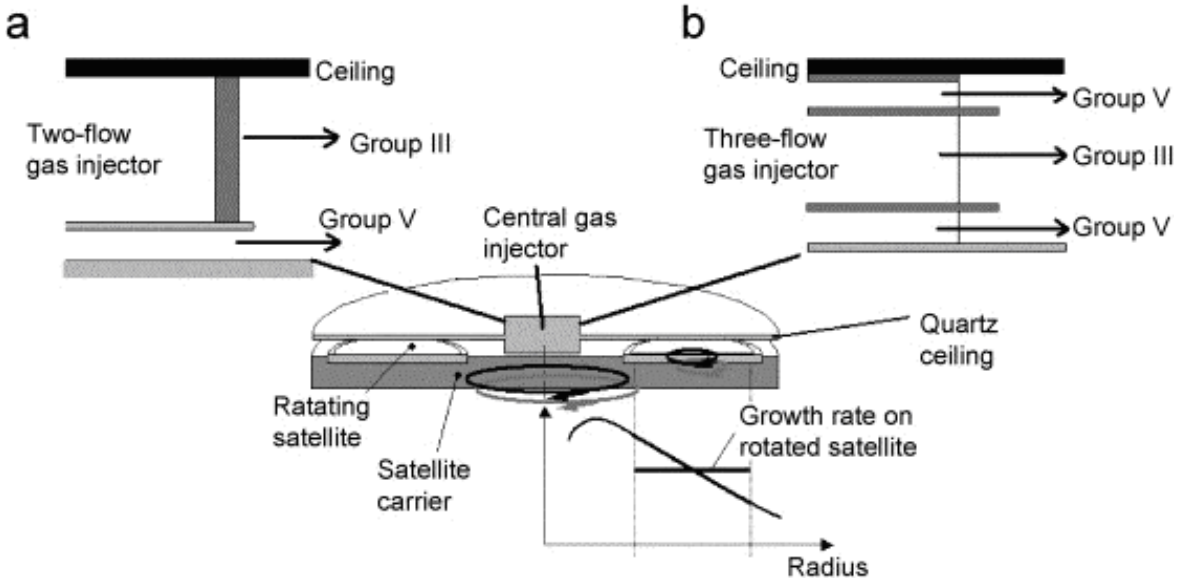


RHS: PPCVD results from: C.W. Lim, M.R. Smith, M.C. Jermy, J.-S. Wu, and S.P. Krumdieck, Computers and Fluids, 2012.

CURRENT MOTIVATION

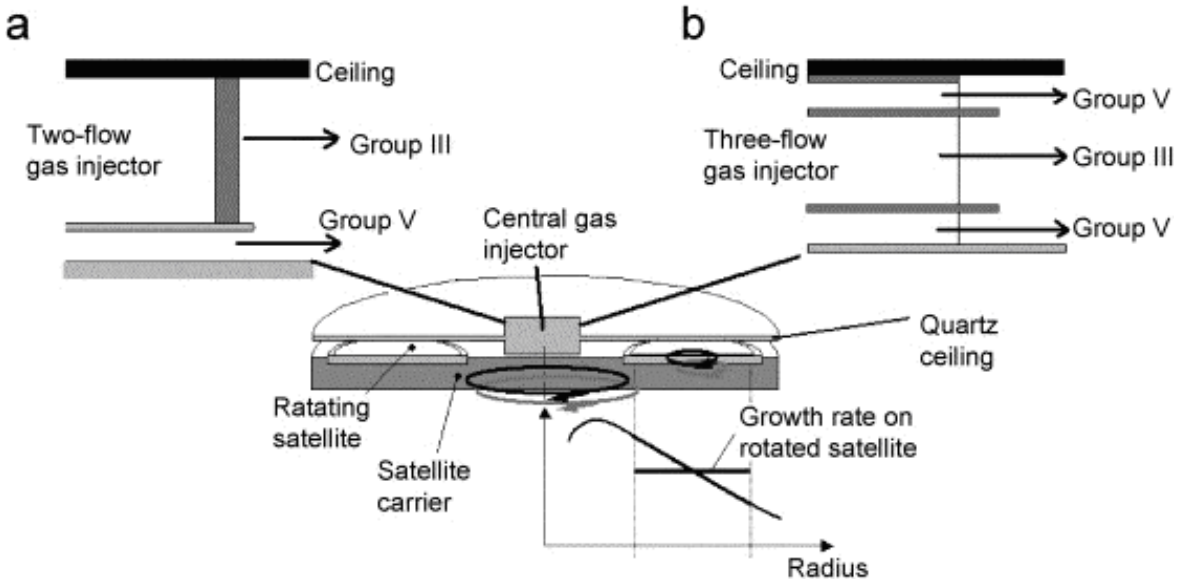
- The MOCVD process is often used in the manufacture of high power LED devices.
- There are two approaches often employed: a showerhead style reactor (left), and a planetary (satellite) reactor (center, right).

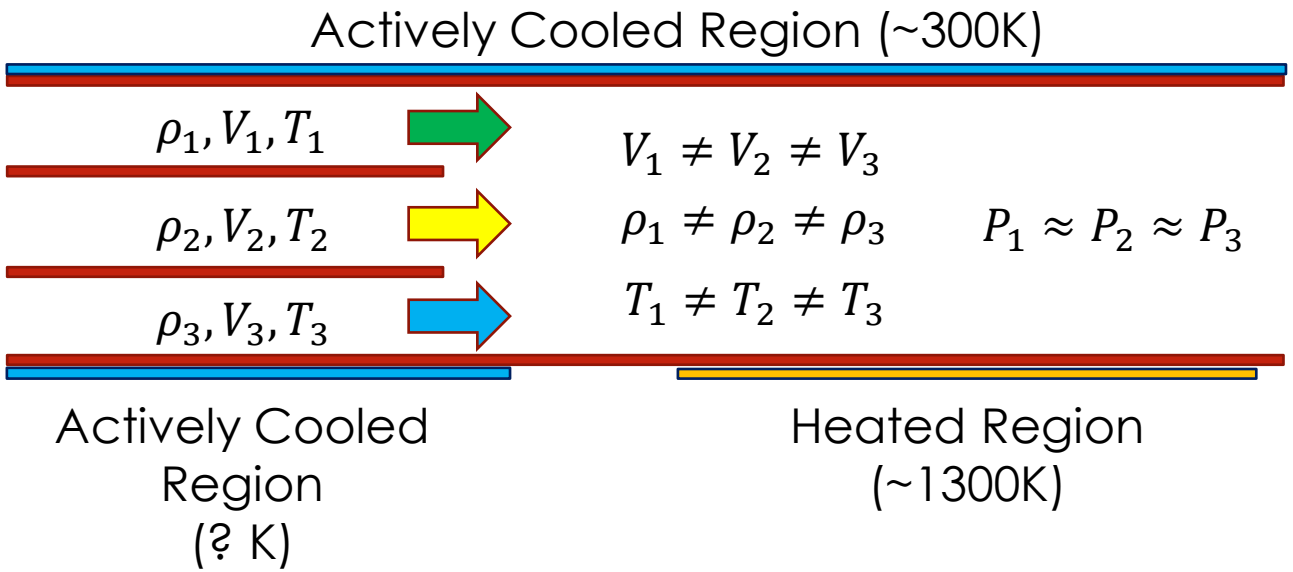




MOTIVATION

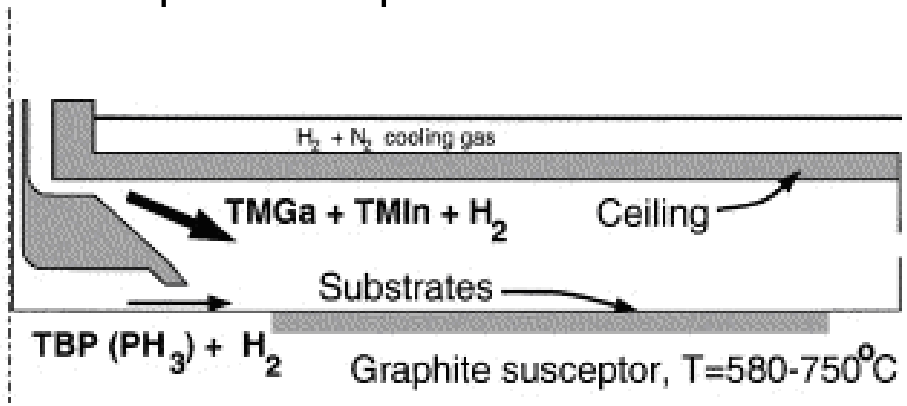
- Today's discussion will focus on the simulation of a planetary style MOCVD reactor.
- In such an approach, gases of different composition flow through 2 (or 3) separate inlets through a small, heated region.

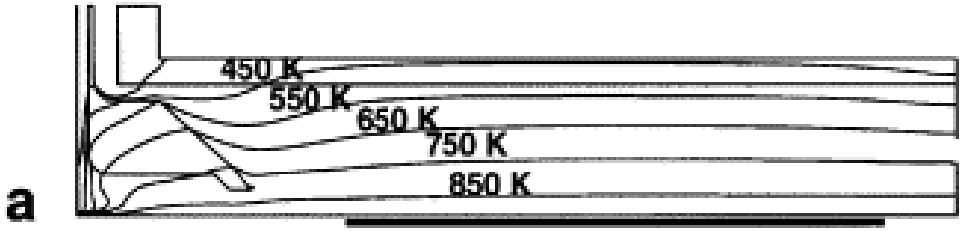




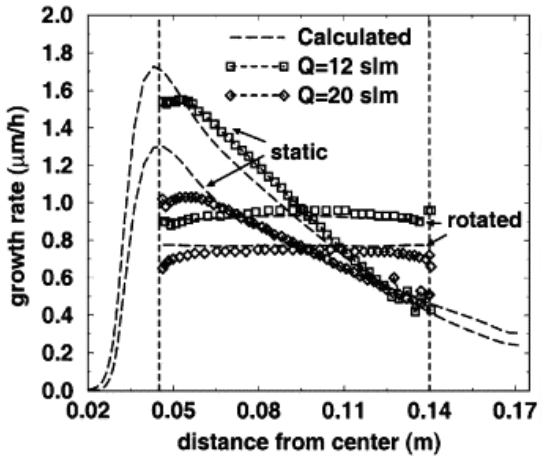
MOTIVATION

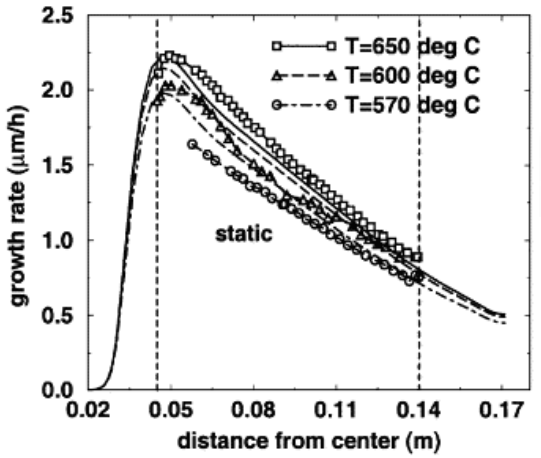
- Samples of previous work





M. Dauelsberg et al., Modeling and experimental verification of transport and deposition behavior during MOVPE of Ga1-xInxP in the Planetary Reactor, Journal of Crystal Growth, 208[1-4], pp:85-92, 2000.





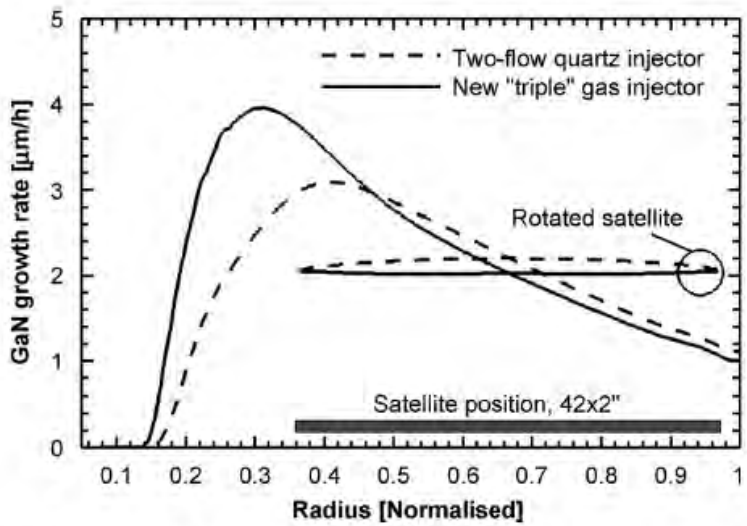


Fig. 3. Comparison of two-flow quartz injector with new injector design $42 \times 2''$ Planetary Reactor® configuration. Growth rate is shown as a function of susceptor radius. $p = 200$ mbar, $T_{growth} = 1045$°C. Gas flow rates were $Q_{H2} = 20.4$ slm, $Q_{NH3} = 10$ slm.

C. Martin et al., Modelling of group-III nitride MOVPE in the closed coupled showerhead reactor and Planetary Reactor, Journal of Crystal Growth, 303[1], pp: 318-322, 2007.

CORE FEATURES

- There are several core features associated with this process:
  - High temperatures across the relatively small cross section encourage chemical reactions in the vicinity of the susceptor.
  - The flows from each inflow consist of different densities, flow velocities and temperatures.
- These features result in some challenges, as we shall see – especially considering our previous approaches for high-speed GPU computing.

VECTOR SPLIT FVM COMPUTATION

- The challenge – all vector split flux formulations have an associated numerical dissipation.
- We can (approximately and inappropriately) estimate the numerical dissipation by rearranging the fluxes to be in Rusanov form:

$$F = (1/2)(F_L + F_R) - (\propto/2)(U_R - U_L)$$

  where alpha (in this case) is a characteristic speed associated with the system.
- Substitution of fluxes in this form into the original governing expression reveals a dissipative term:

Sub in $F = (1/2)(F_L + F_R) - (\propto/2)(U_R - U_L)$

$$\frac{\overline{U_i}^{k+1} - \overline{U_i}^{k}}{\Delta t} + \frac{F^{k}_{i+\frac{1}{2}} - F^{k}_{i-\frac{1}{2}}}{\Delta x} = 0$$

Re-arrange

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} - \frac{\alpha \Delta x}{2}\frac{\partial^2 U}{\partial x^2} = 0$$

i.e. every split scheme is a n artificial diffusion scheme.

VECTOR SPLIT FVM COMPUTATION

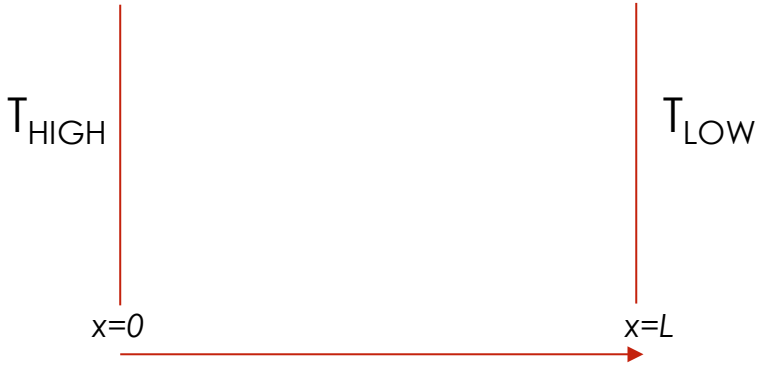
- The complication is (perhaps) not so obvious:

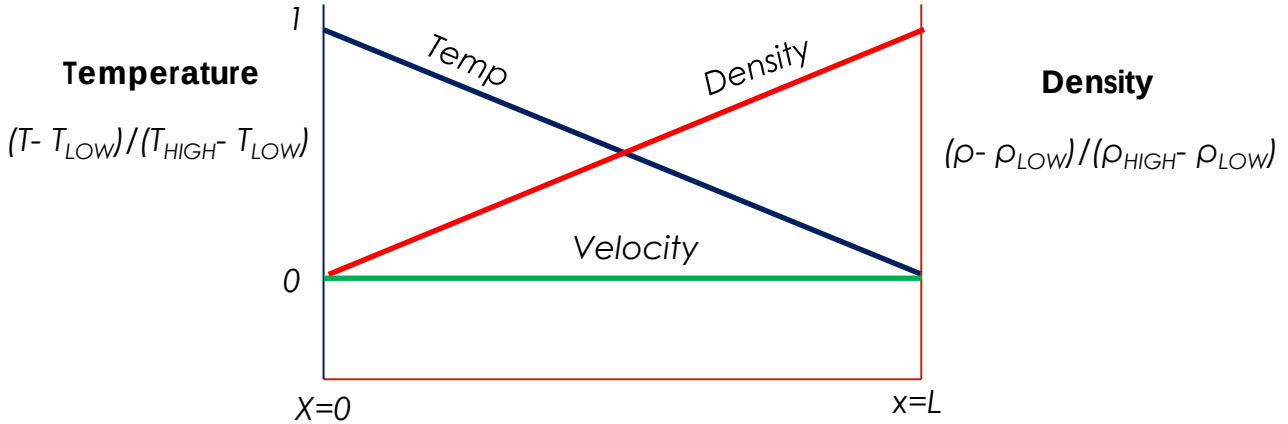
$$\frac{\partial U}{\partial t} + \left(\frac{\partial F}{\partial x}\right)_c - \frac{\alpha_{SCHEME}\Delta x}{2}\left(\frac{\partial^2 U}{\partial x^2}\right) = S$$

This term is never 0 – only approaches it as dx goes to 0.

- In the steady state, the equation can be rearranged: $\left(\frac{\partial F}{\partial x}\right)_c = S + \frac{\alpha_{SCHEME}\Delta x}{2}\left(\frac{\partial^2 U}{\partial x^2}\right)$

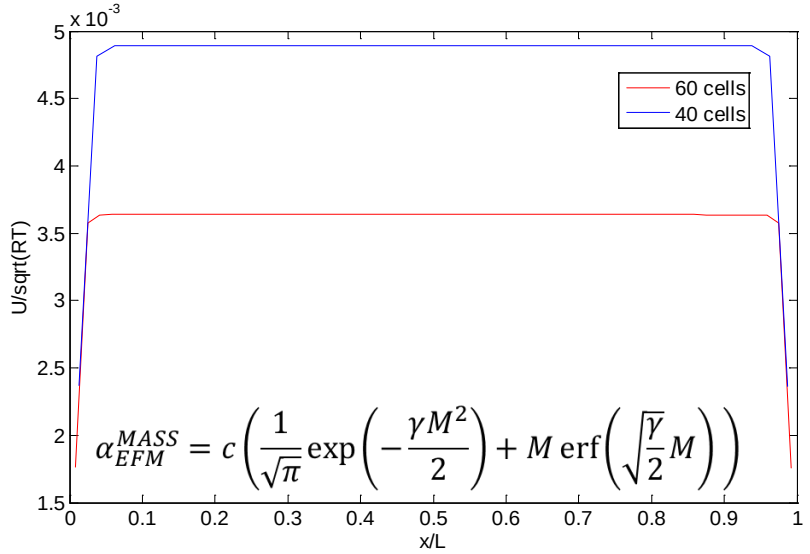
- Consider the steady 1D problem (for the steady N-S equations):

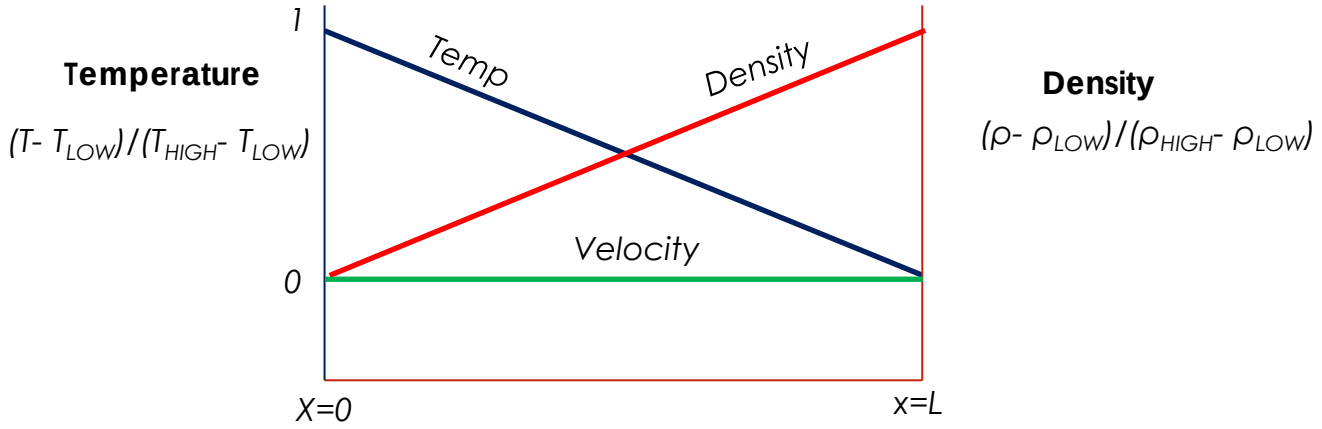




VECTOR SPLIT FVM COMPUTATION

- The presence of this additional dissipative term results in a (albeit) small, non zero and non-physical velocity field.





- Unfortunately, it cannot be neglected for low speed flows.

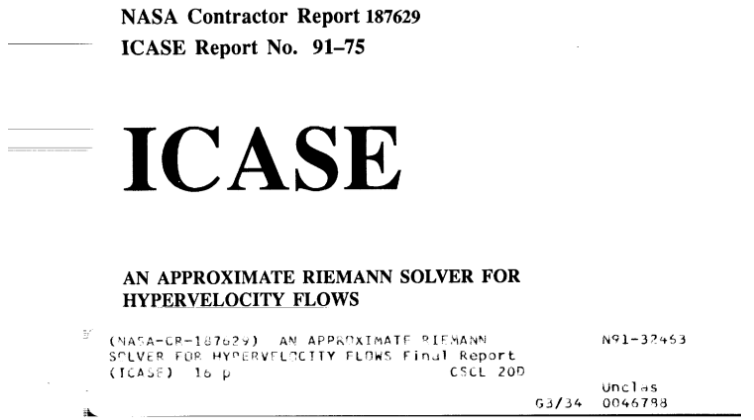
VECTOR SPLIT FVM COMPUTATION

- This means than none of our previous GPU work is capable of performing this simulation.
- We require a flux solver which can appropriately compute fluxes in the absence of a pressure field without resulting in a non-physical result.
- The obvious solution – a flux difference splitting approach might have a better chance at doing this.
- To be more specific (about my choice) – we can use the analytical solution to the Riemann problem.

FLUX SOLVER

- The solver used is a so-called "approximate" Riemann solver developed by Peter Jacobs (University of Queensland).

- The solver has similarities to Osher's approximate Riemann solver.



NASA Contractor Report 187629
ICASE Report No. 91–75

ICASE

AN APPROXIMATE RIEMANN SOLVER FOR HYPERVELOCITY FLOWS

(NASA-CR-187629) AN APPROXIMATE RIEMANN SOLVER FOR HYPERVELOCITY FLOWS Final Report (ICASE) 16 p CSCL 20D N91-32453 Unclas G3/34 0046798

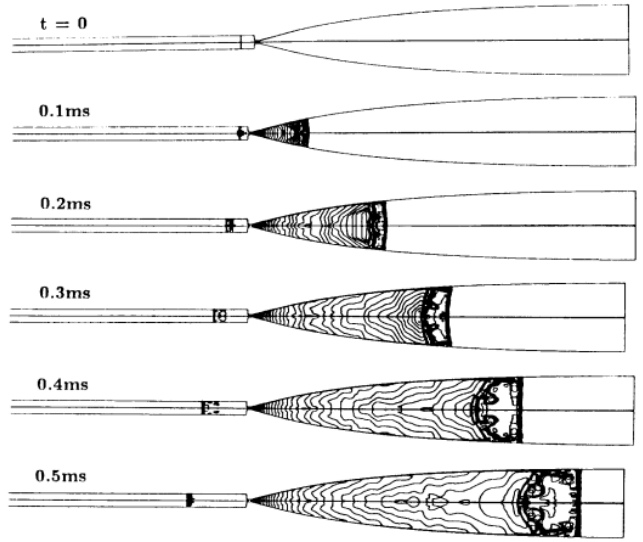
P. A. Jacobs

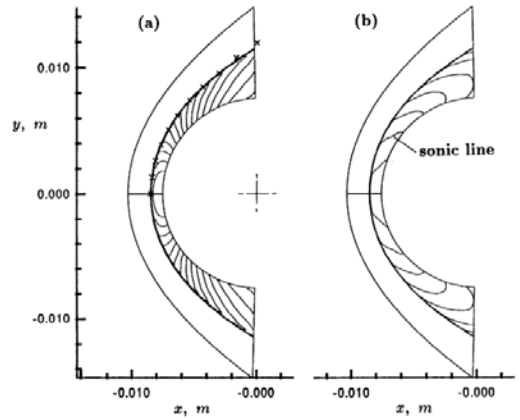
Contract No. NAS1–18605
September 1991

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665–5225

Operated by the Universities Space Research Association

NASA
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665-5225



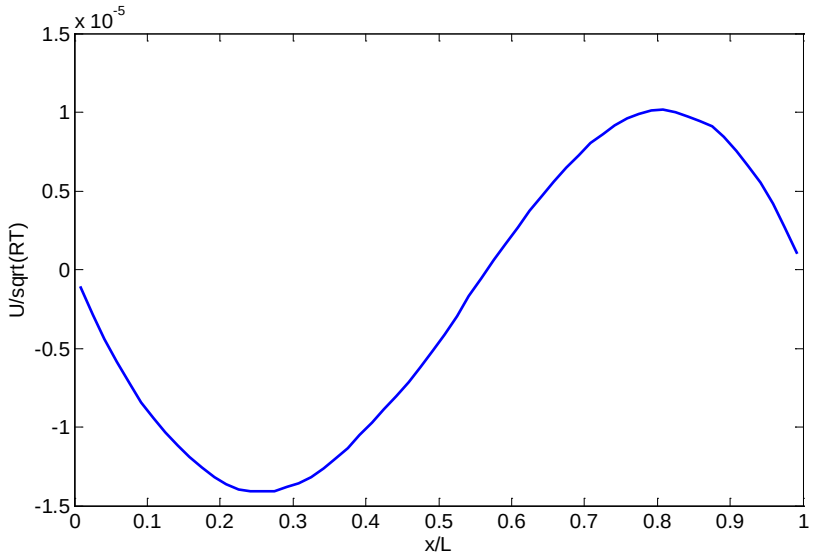


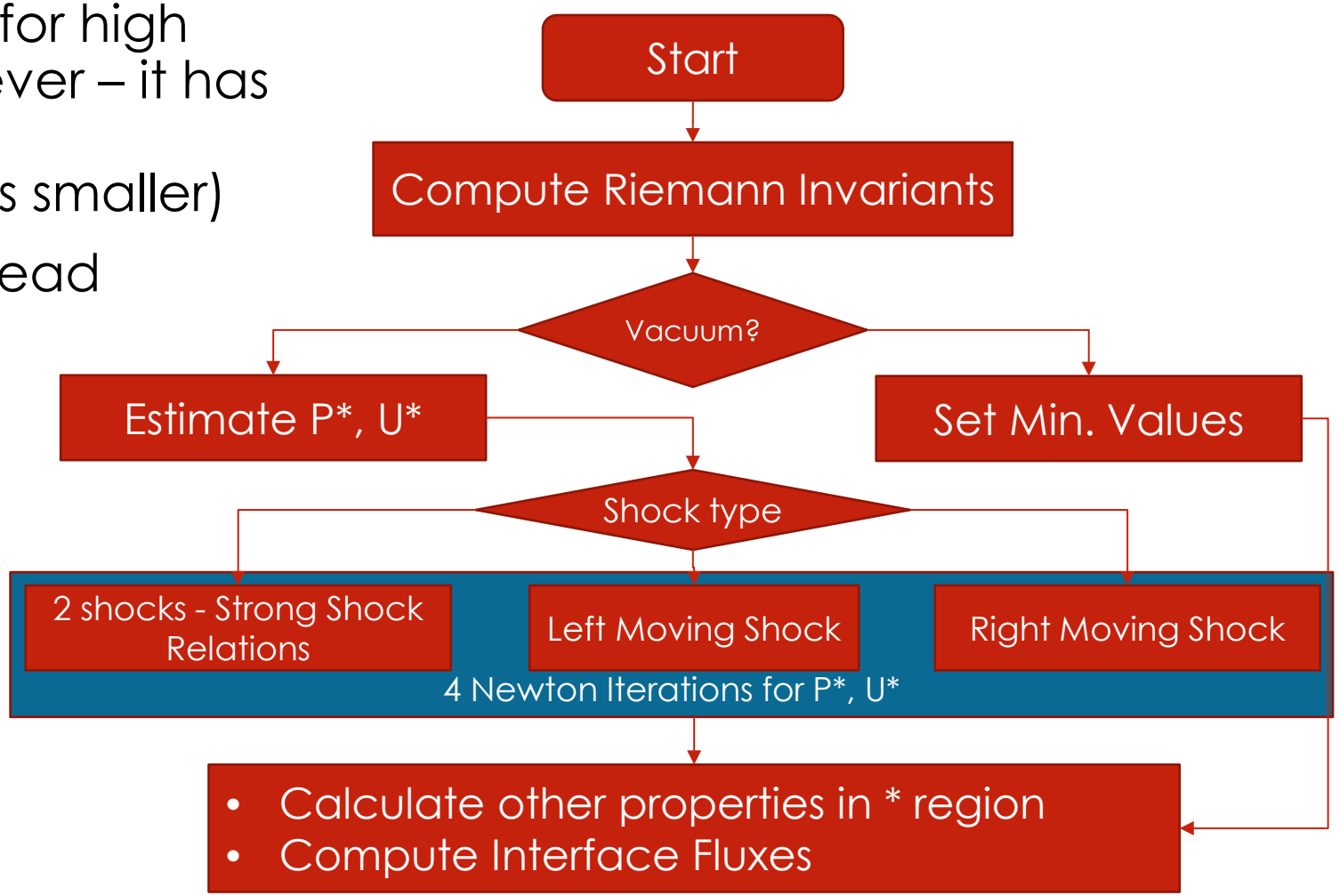
P. A. Jacobs, Approximate Riemann Solver for Hypervelocity Flows, A.I.A.A. Journal Vol. 30(10):2558—2561, 1992.

S. Osher and F. Solomon, Upwind Difference Schemes for Hyperbolic Systems of Conservation Laws, Mathematics of Computation, 38(158):339-374, 1982.

FLUX SOLVER

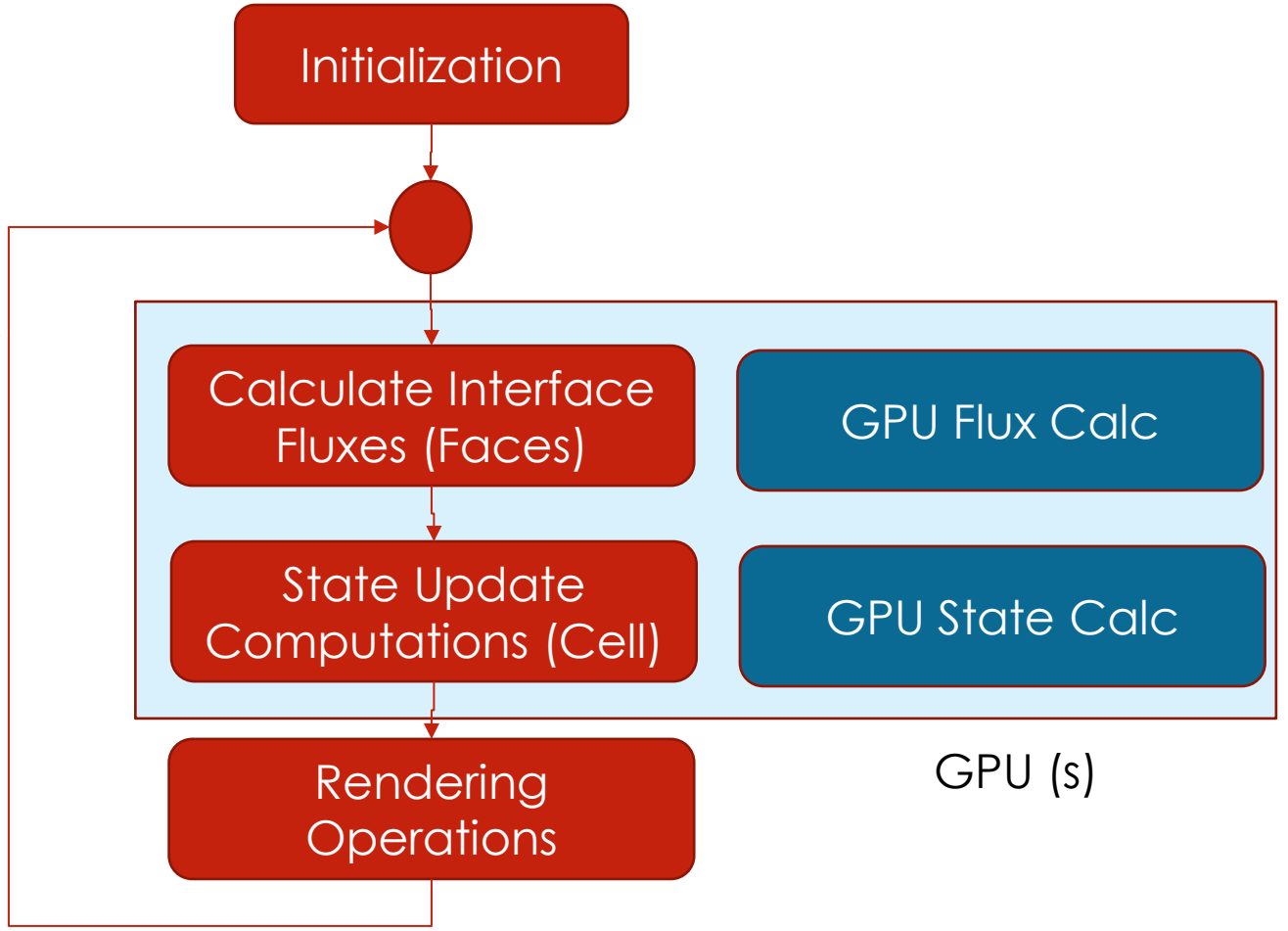
- The solver was originally design for high speed (hypersonic) flows, however – it has favorable characteristics.
  (i.e. Error 2 orders of magnitudes smaller)
- Horrible for GPU computing (thread divergence)





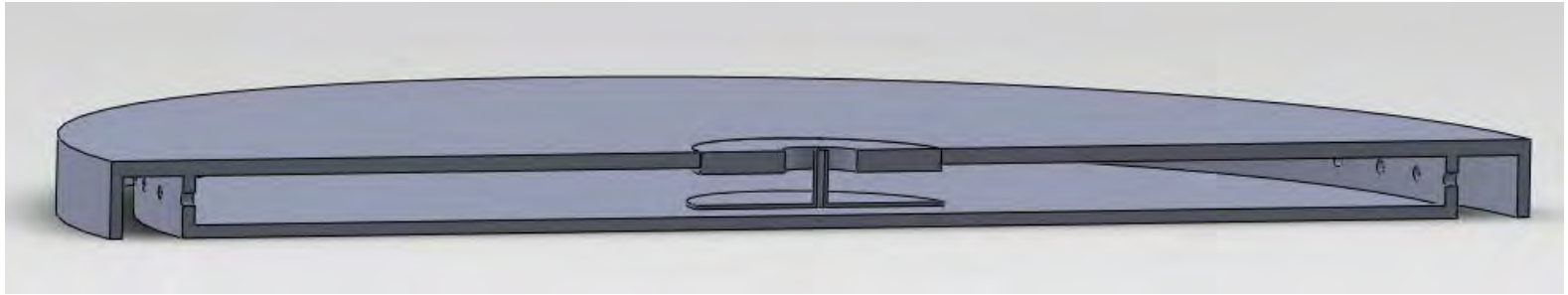
GPU IMPLEMENTATIONS

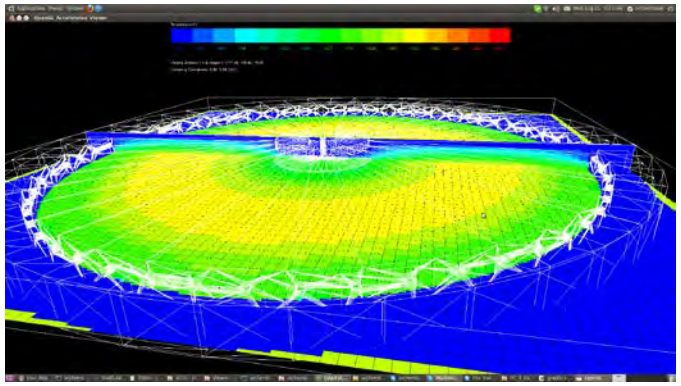
- The parallelism paradigm changes when computing fluxes across interfaces in the conventional fashion.
- Fluxes are computed in parallel over interfaces.
- Gradients of primitives (for higher order reconstruction) and state updates are parallel over cells.
- Asynchronous transfers between the host and GPU are used to hide the communication cost.

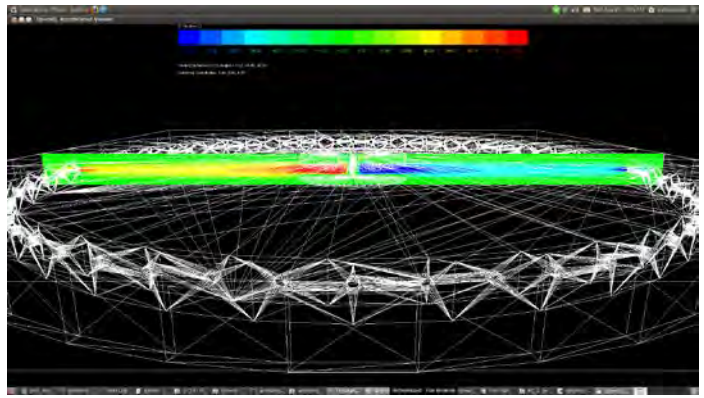


ADAPTIVE UNSTRUCTURED GRIDS

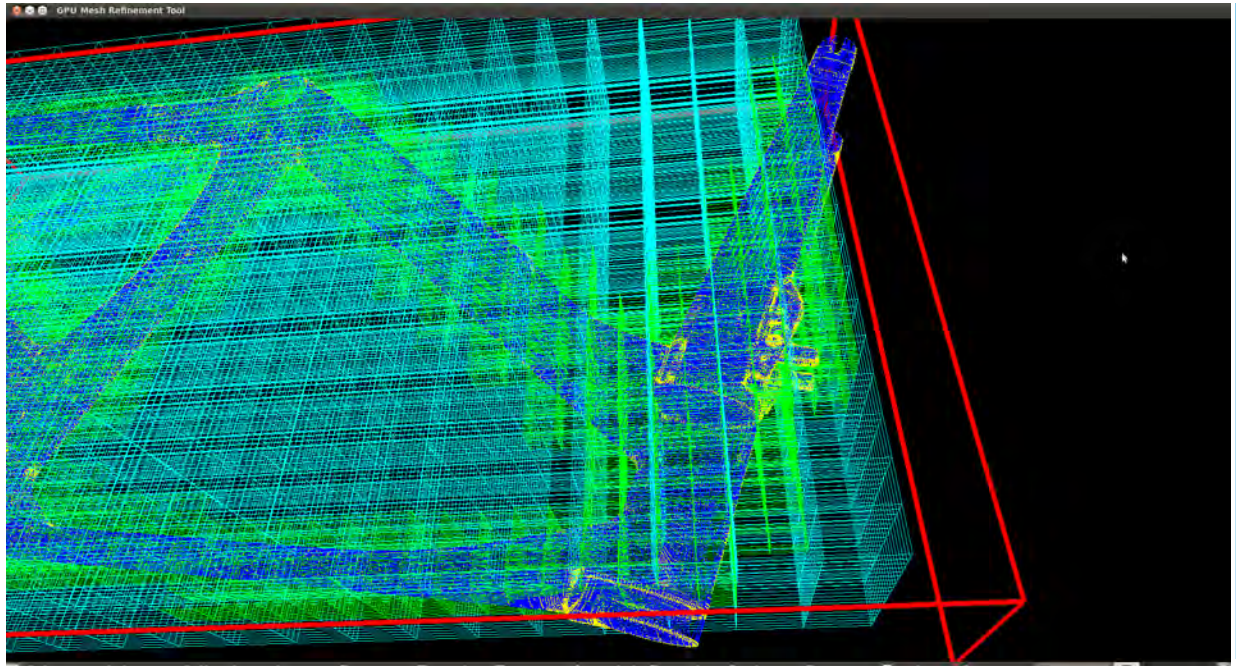
- Unfortunately, real-life industrial work is often strongly multi-scale and requires the use of *adaptive, unstructured computational grids*.
- In addition, we employ AMR to assist our computations.

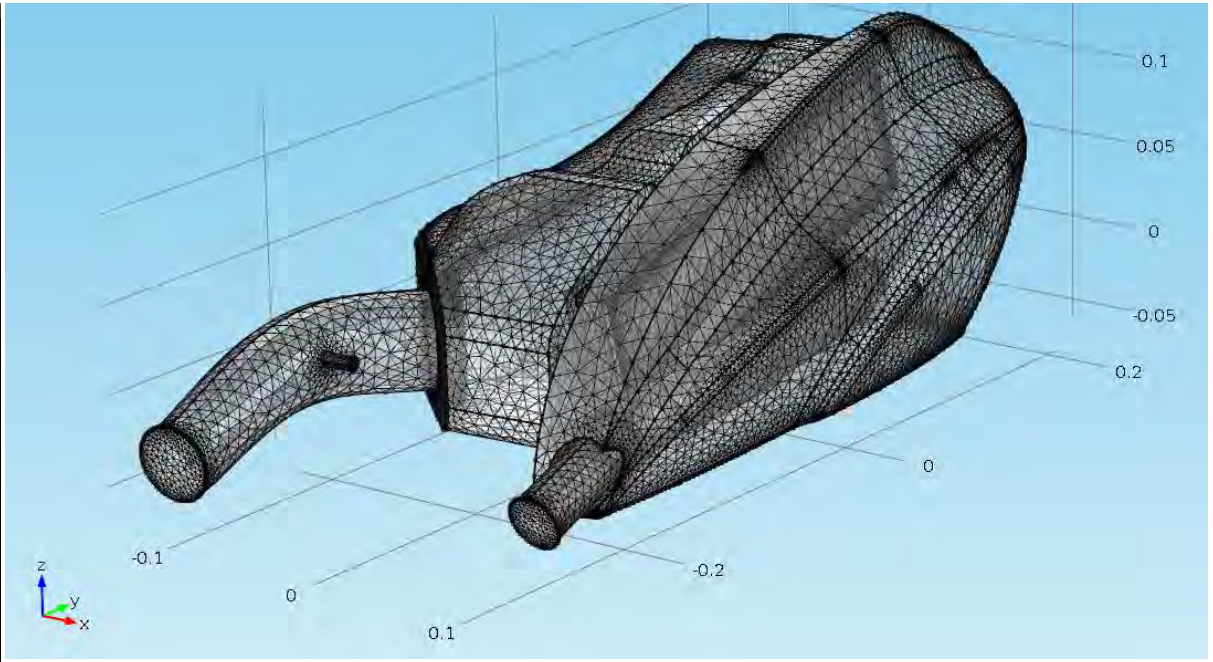




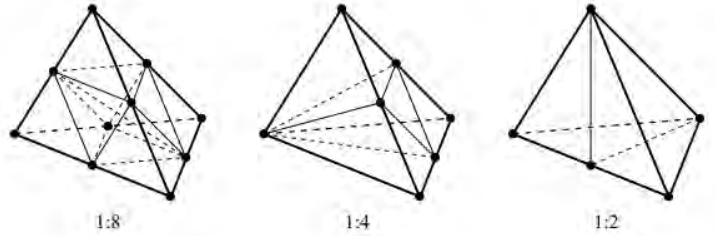


ADAPTIVE UNSTRUCTURED GRIDS





- Meshing can be performed either by (i) external software using tetrahedral (or otherwise) grids (right) or on the GPU using Cartesian cells (left).
- Cell splitting depends on the cell → rectangular cells are split 8 ways, for example.



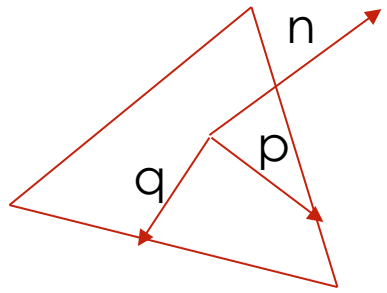
R. Biswas and R. Strawn, Applied Numerical Mathematics, 26(1-2):135-151, 1998.

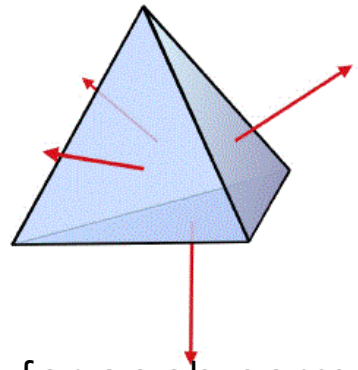
GPU IMPLEMENTATIONS

- To accelerate the code, both GPU shared memory and texture memory is used.
- Since our approach is unstructured, we absolutely need double precision.
- CUDA can be "tricked" into using double precision textures using int2.

```
// Get double from texture memory
static __inline__ __device__ double fetch1DDouble(texture<int2,1> tex, int i) {
        int2 v = tex1Dfetch(tex,i);
        return __hiloint2double(v.y, v.x);
}
```

```
// Calculate the fluxes and the interface (star) state - use texture memory for normals

GPU_Calc_Flux(flux, pstar, p[Left*NO_UNKNOWNS], p[Left*NO_UNKNOWNS+1], p[Left*NO_UNKNOWNS+2], p[Left*NO_UNKNOWNS+3], Left_cRT,
                           p[Right*NO_UNKNOWNS], p[Right*NO_UNKNOWNS+1], p[Right*NO_UNKNOWNS+2], p[Right*NO_UNKNOWNS+3], Right_cR
                           fetch1DDouble(Tex_Interface_Normals, 9*i), fetch1DDouble(Tex_Interface_Normals, 9*i+1), fetch1DDouble
                           fetch1DDouble(Tex_Interface_Normals, 9*i+3), fetch1DDouble(Tex_Interface_Normals, 9*i+4), fetch1DDoubl
                           fetch1DDouble(Tex_Interface_Normals, 9*i+6), fetch1DDouble(Tex_Interface_Normals, 9*i+7), fetch1DDoubl
```

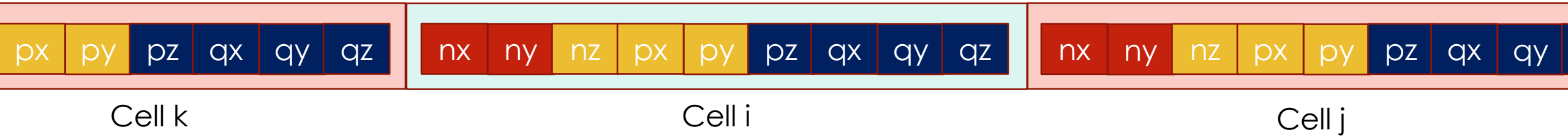




9 quantities to be stored – x,y,z for each normal.

GPU IMPLEMENTATIONS

- Coalesced memory access is guaranteed by storing desired quantities together in memory.



- All variables (primitives, conserved quantities, fluxes) are stored in this way.
- In addition, each kernel is assigned a number of threads per block based on the number of registers required for each.

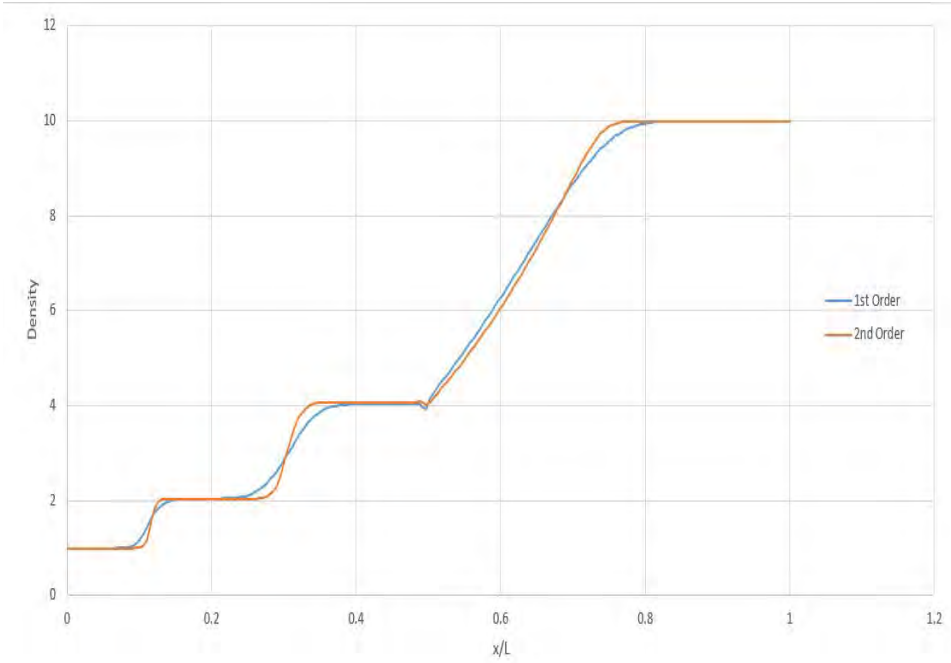
```
void GPU_Update_Interface_Fluxes() {

        int threadsperblock = 64;
        int blockspergrid = (NF + threadsperblock - 1)/threadsperblock;

        GPU_Calc_Interface_Fluxes<<<blockspergrid,threadsperblock>>>(d_p, d_Interface_flux, d_Face_Type, d_Cell_Index, d_Cell_Index_Si

}

void GPU_Update_State() {

        int threadsperblock = 128;
        int blockspergrid = (NC + threadsperblock - 1)/threadsperblock;

        GPU_Calc_State<<<blockspergrid,threadsperblock>>>(d_Cell_Index, d_Cell_Index_Sign, d_Left_Index, d_Right_Index, d_Face_Type, d_

}
```

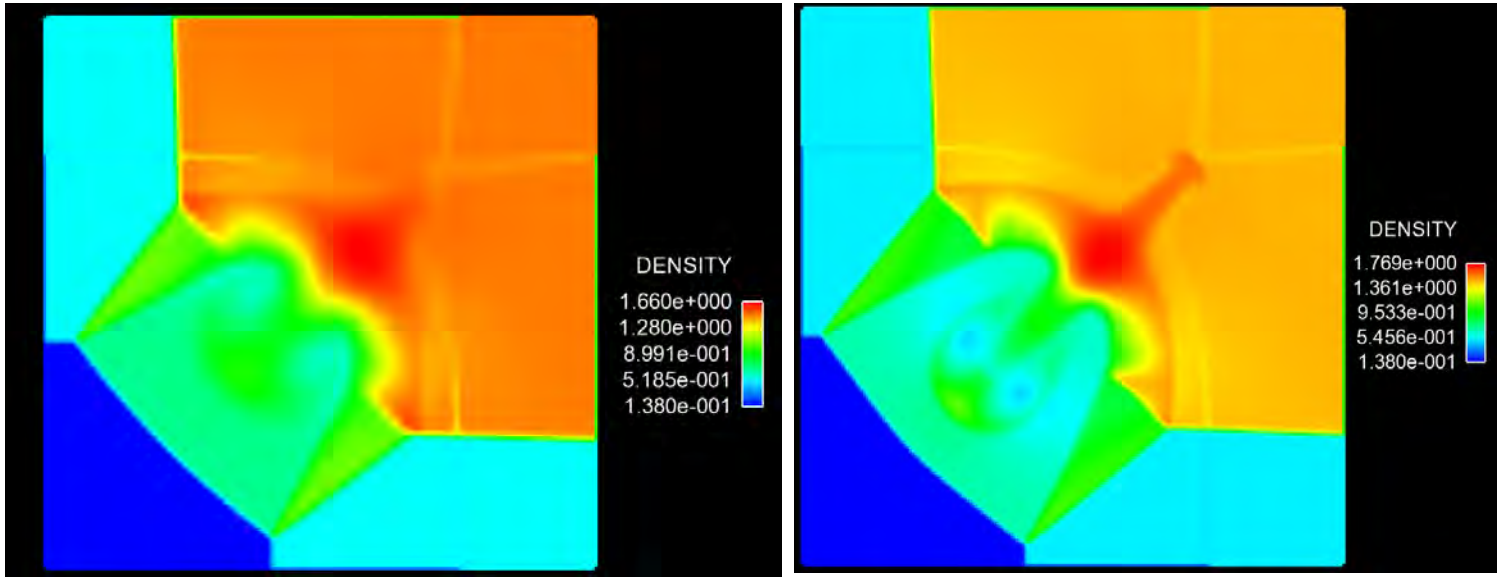
Fluxes are face parallelized.

States are cell parallelized.

HIGHER ORDER EXTENSION

- The method for higher order extension depends on the mesh type.
- TVD-MUSCL is employed for Cartesian style grids (MINMOD here)
- It's important to know that the extension hasn't backfired on us.





C.W. Schulz-Rinne, J.P. Collins and H.M. Glaz, Numerical Solution of the Riemann problem for two-dimensional Gas Dynamics, SIAM J. Sci. Comput., 14, pp. 1394-1414, 1993.
S. Serna, A Class of Extended Limiters Applied to Piecewise Hyperbolic Methods, SIAM J. Sci. Comput, 28[1], pp. 123-140, 2006.

EXAMPLE - MOCVD SIMULATION

- The unstructured, adaptive (AMR) solver is applied to MOCVD simulation.
- Governing equations: N-S equations (5) for gas flow, plus transport equations for an additional 7 gas species.
- Species transport includes:
  - Advection (with bulk gas)
  - Diffusion
    - Mass (Concentration) Diffusion (Due to presence of density gradients)
    - **Thermal Diffusion** (Due to presence of temperature gradients)
- Diffusion terms treated using higher order central differencing.
- Topic of current NSC/MOST project (GPU and AVX optimization of MOCVD)

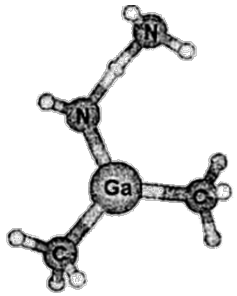
102年度 【 GPU 和 AVX 為基礎的 MOCVD 反應器高速計算最佳化 】

MOCVD SIMULATION

- We approximate the deposition properties of the reactor by computing the formation of heavy particles.
- This is a simpler (3 reaction channel) model than usually required.

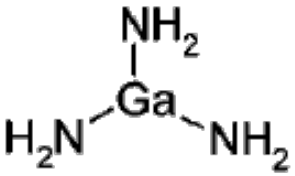
Heavy Particle Formation Chain Reaction

$$Ga(CH_3)_3 + NH_3 \rightarrow Ga(CH_3)_2NH_2 + CH_4$$



$$Ga(CH_3)_2NH_2 + NH_3 \rightarrow Ga(NH_2)_2CH_3 + CH_4$$

$$Ga(NH_2)_2CH_3 + NH_3 \rightarrow Ga(NH_2)_3 + CH_4$$

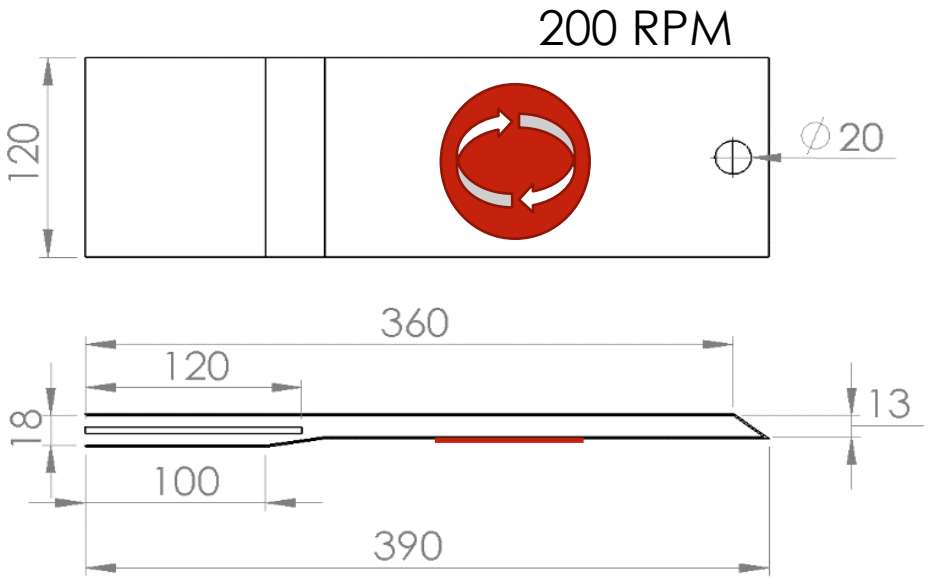


reaction	$\log_{10}A$	α	Ea	ΔH_r
$Ga(CH_3)_3 + NH_3 \rightarrow Ga(CH_3)_2NH_2 + CH_4$	7.544	1.0	17.93	−18.89
$Ga(CH_3)_2NH_2 + NH_3 \rightarrow Ga(NH_2)_2CH_3 + CH_4$	7.328	1.0	18.00	−14.11
$Ga(NH_2)_2CH_3 + NH_3 \rightarrow Ga(NH_2)_3 + CH_4$	7.211	1.0	18.22	−9.82

D. Moscatelli and C. Cavallotti, Theoretical Investigation of the Gas-Phase Kinetics Active during the GaN MOVPE, J. Phys. Chem. A, 111, pp. 4620-4631, 2007.

MOCVD SIMULATION

- Most real (modern) MOCVD reactors are protected by patents and NDA's.
- However, I can show you results from this reactor.

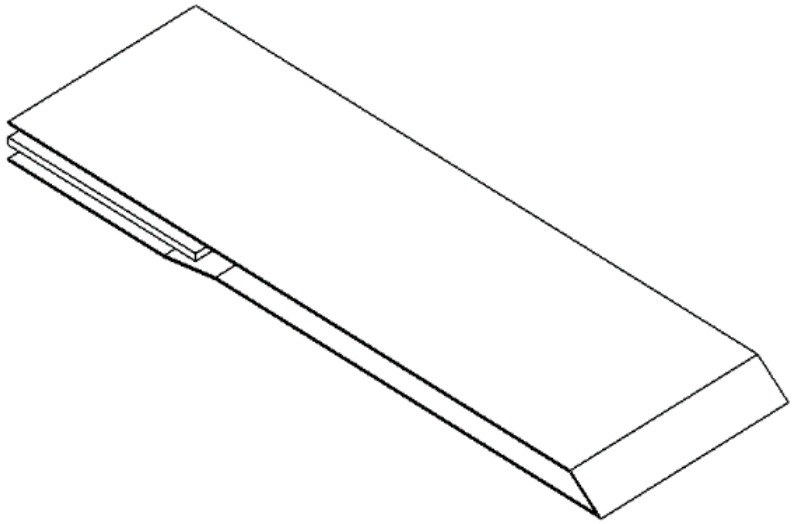


$$T(x) = \begin{cases} 300 & x < 100 \\ 300 + \dfrac{1000(x-100)}{140-100} & 100 \le x < 140 \\ 1300 & 140 \le x < 310 \\ 1300 - \dfrac{1000(x-310)}{360-310} & 310 \le x < 360 \end{cases}$$

J. Skibinski et al., Modeling of heat and mass transfer in GaN MOVPE reactor, 17th International Conference on Crystal Growth and Epitaxy ICCGE-17, 2013.

MOCVD SIMULATION

- The flow rates were adapted from a circular reactor based on matching flowrates through the cross sectional area of the injectors.
- A simple surface a**d**sorption model for Ga(NH2)3 was implemented.
- Non-slip surfaces used on all walls.

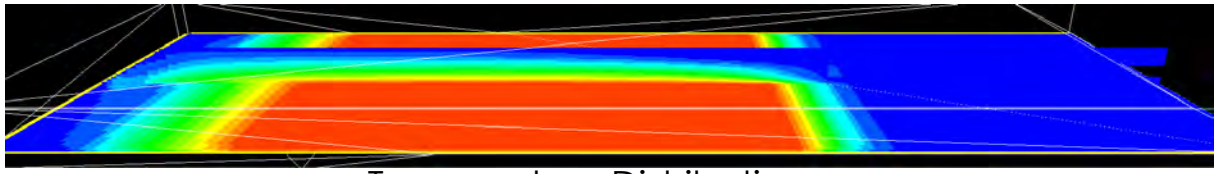


Flow Rates (SLPM) and Temperatures

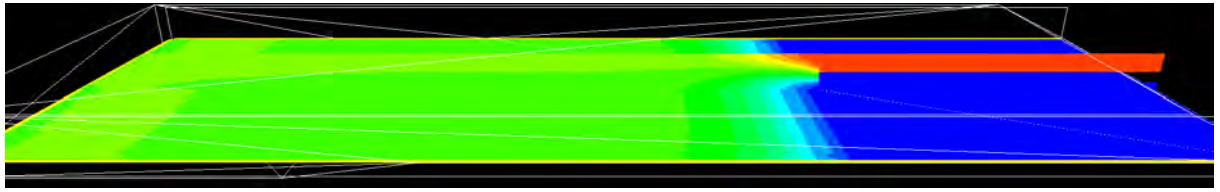
Inflow	NH_3	N_2	H_2	TMGa	Temp
Top	3	0	5	0	300K
Bottom	0	0.5	5	0.02	300K

J. Skibinski et al., Modeling of heat and mass transfer in GaN MOVPE reactor, 17th International Conference on Crystal Growth and Epitaxy ICCGE-17, 2013.

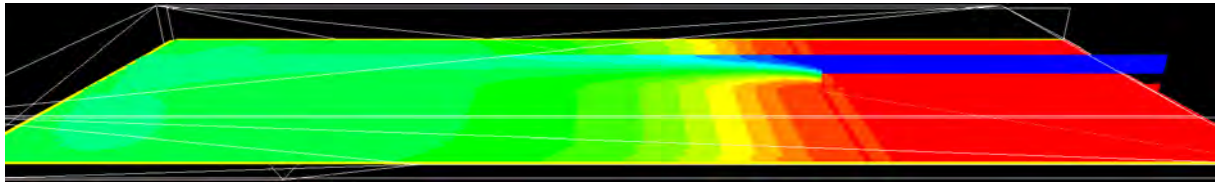
MOCVD SIMULATION



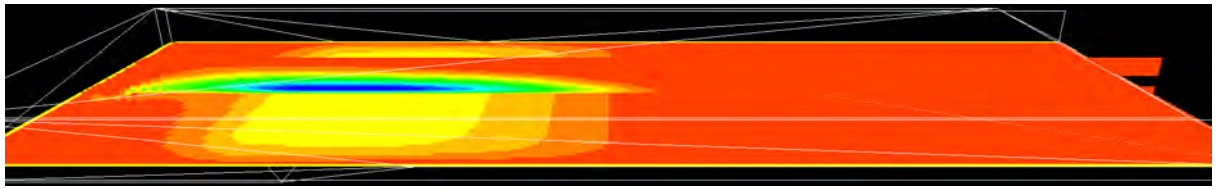
Temperature Distribution



H2 (Mass fraction) Distribution



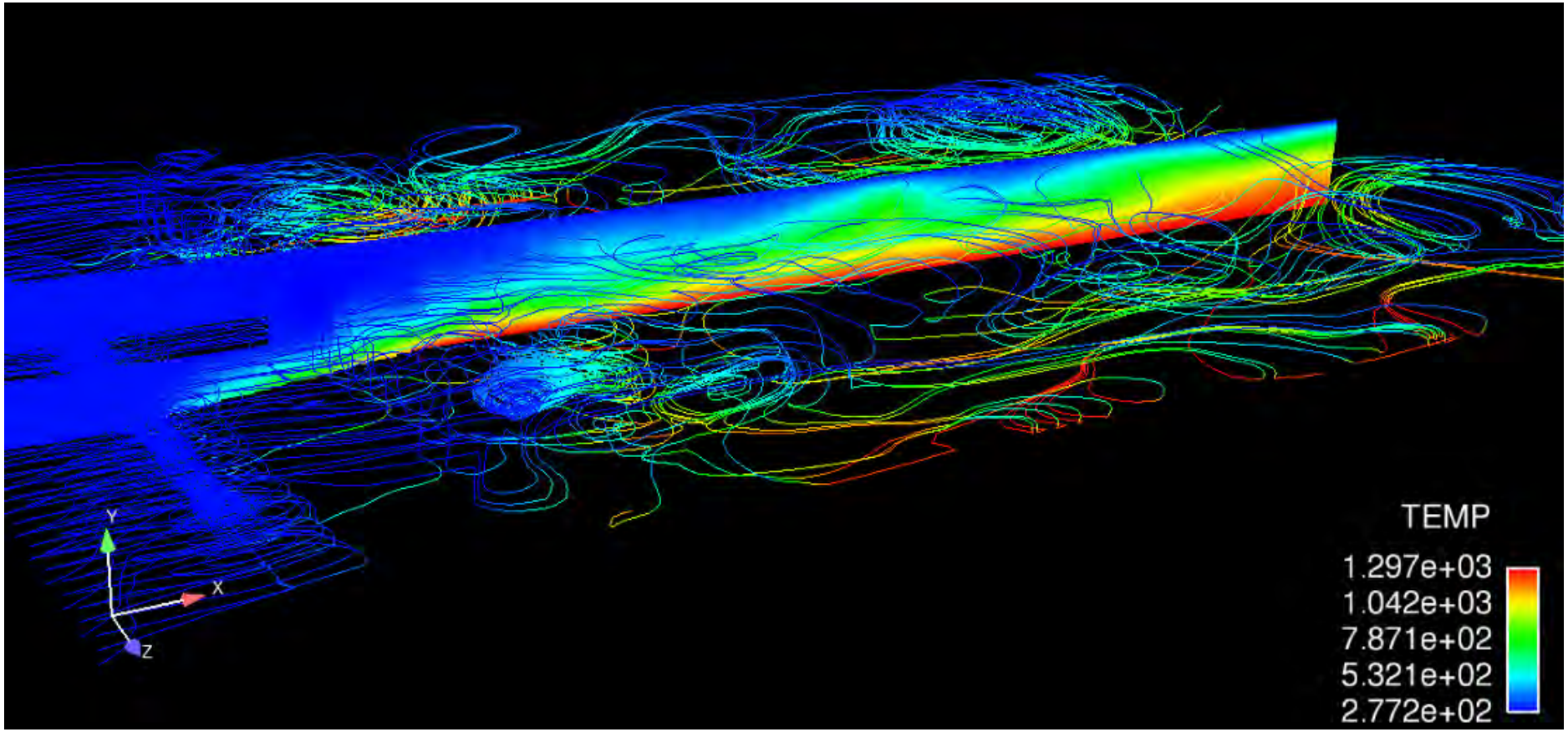
NH3 (Mass fraction) Distribution



Ga(NH2)3 (Mass fraction) Distribution

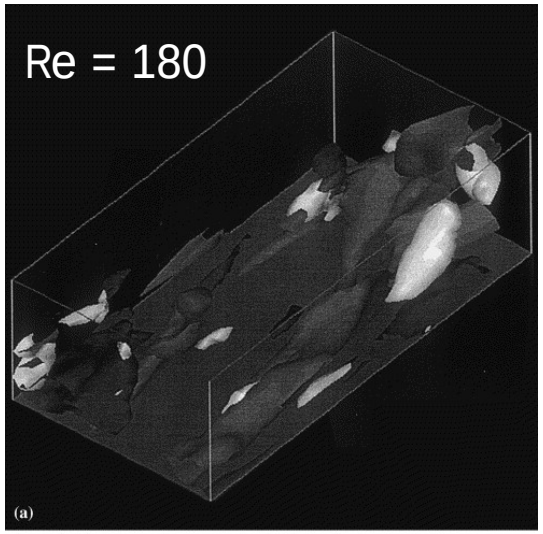
- Number of cells: ~ 2-13 million cells.
- First order results match those from commercial packages (COMSOL)
- The formation of Ga(NH2)3 is in the correct, expected location.
- There is little upstream diffusion of NH3 into the lower inflow.
- These results were far too convenient:
  - **Circulation, vortex shedding?**
  - **Temperature-diffusion induced instabilities…**

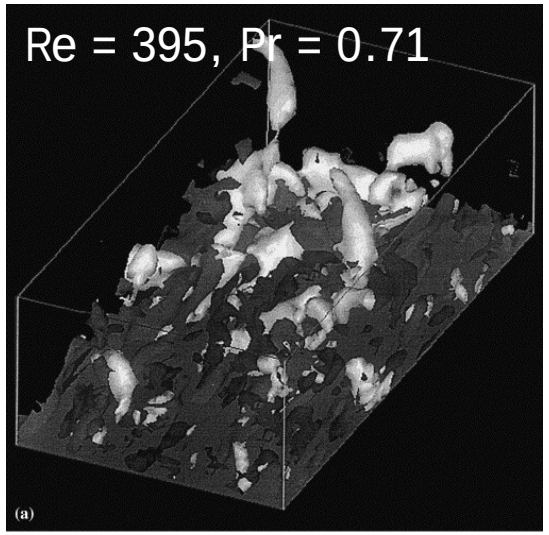
MOCVD SIMULATION

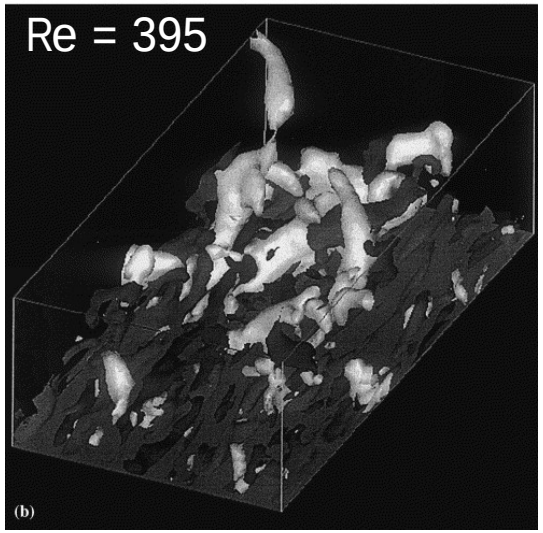


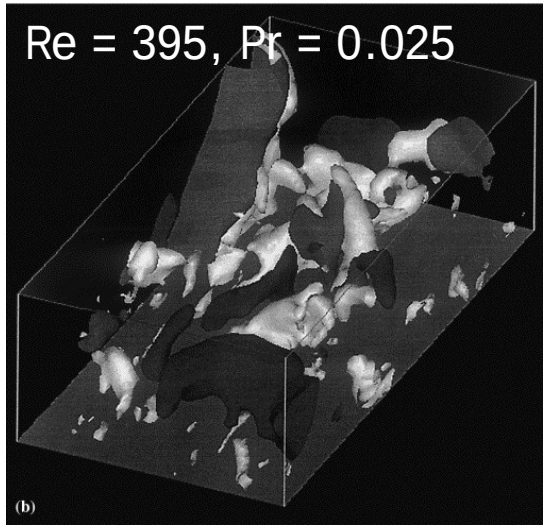
The use of higher order (2nd order, 3rd order) paints a very different picture of the flow situation. (2.3 million cells, 3 levels of AMR refinement)

MOCVD SIMULATION

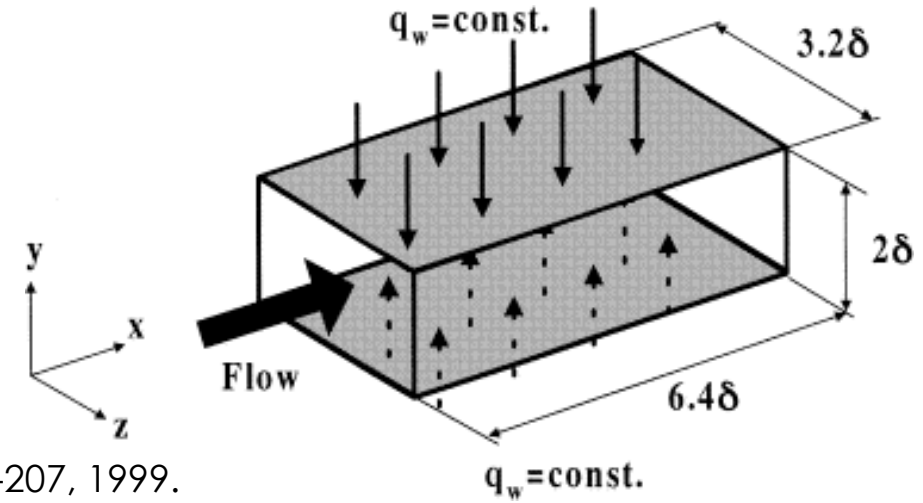








- Expected Re within the reactor were not high enough (perhaps) to explain some of the phenomena we saw.
- We found that some previous DNS simulations of similar problems provided some insight.
- Strong temperature gradients in directions normal to the flow direction result in instabilities.

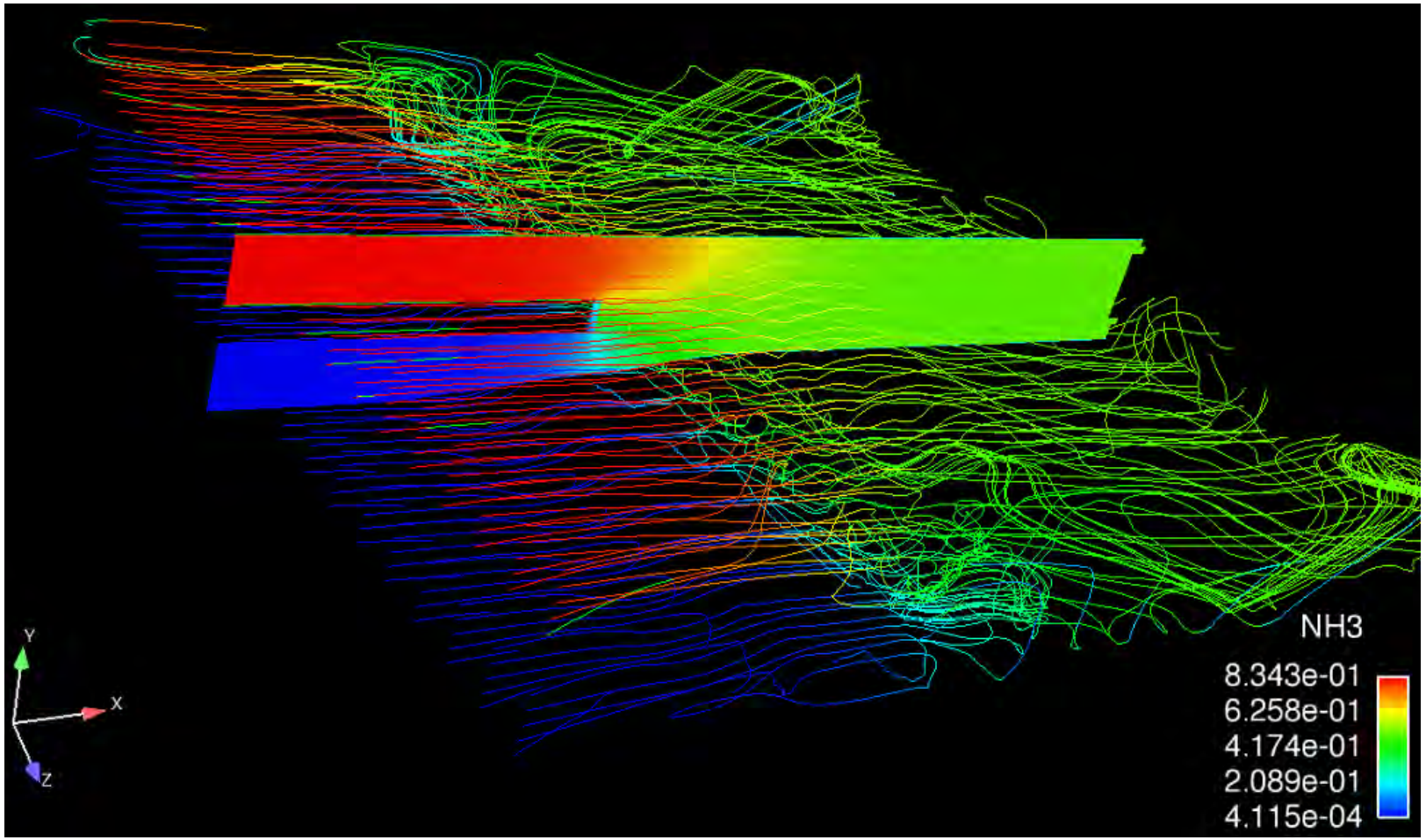


H Kawamura et al., International Journal of Heat and Fluid Flow, 20(3):196-207, 1999.

MOCVD SIMULATION

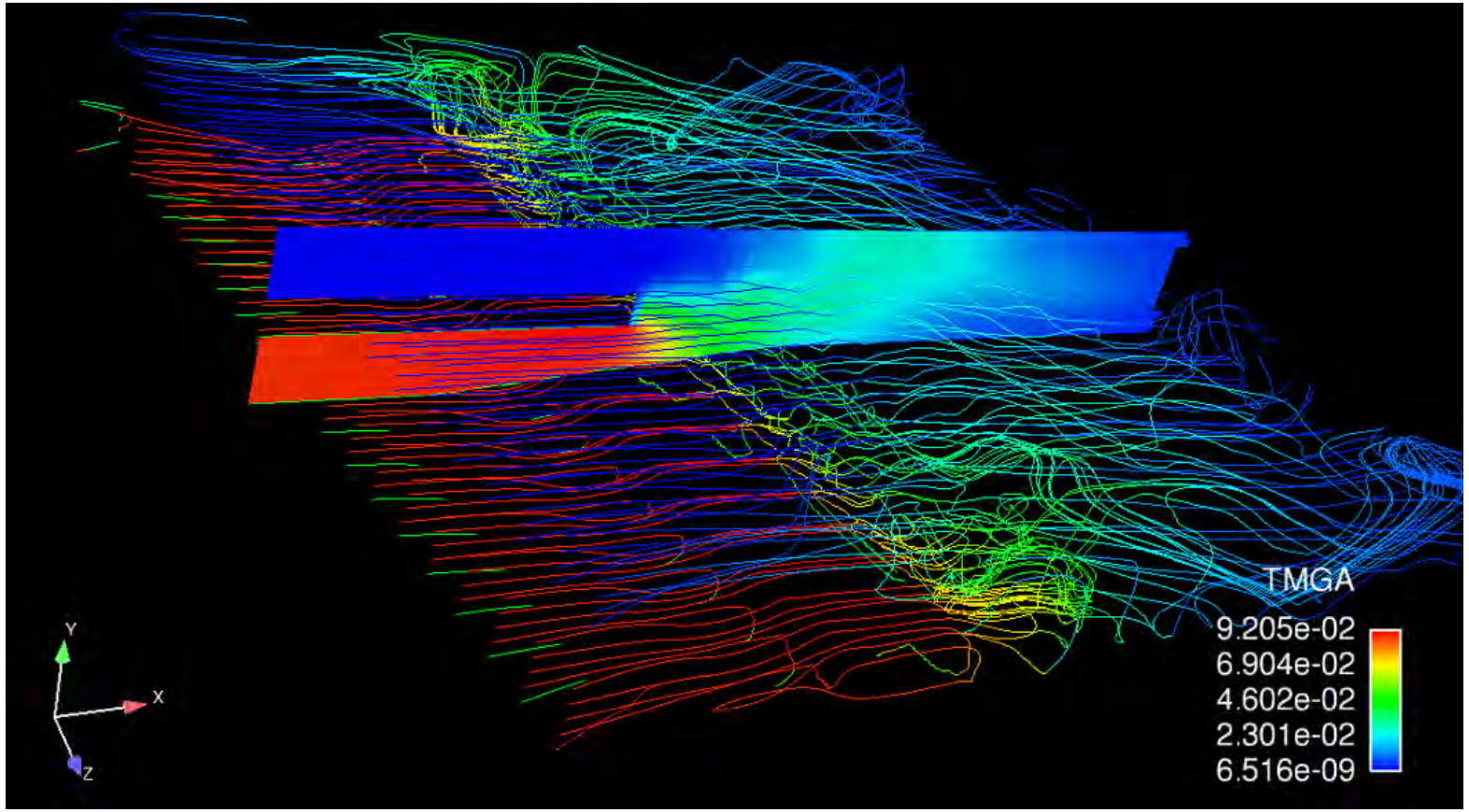
- Several flow stabilities (related to both velocity gradients and density gradients) result in the formation of several 3D vortices.
- This problem has no time-steady solution – hence, the use of a steady solution (which averages these fluctuations) is a poor idea.

MOCVD SIMULATION



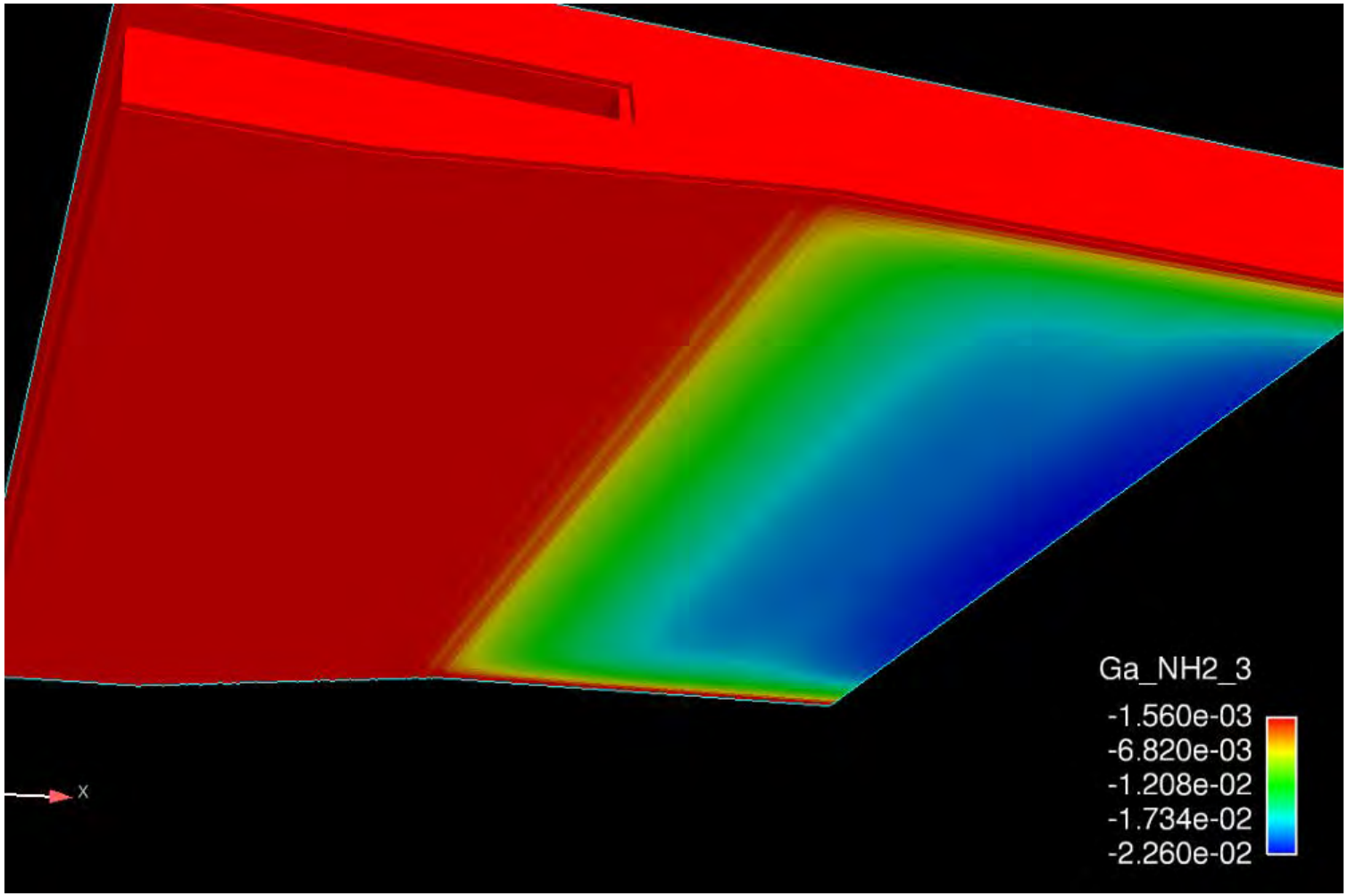
- Higher order implementation predicts several flow features:
  - **Upstream diffusion** of NH3,
  - Vortex shedding (inflow speeds are not matched!)
  - Complex flow structures resulting from strong temperature gradient in flow field.

MOCVD SIMULATION



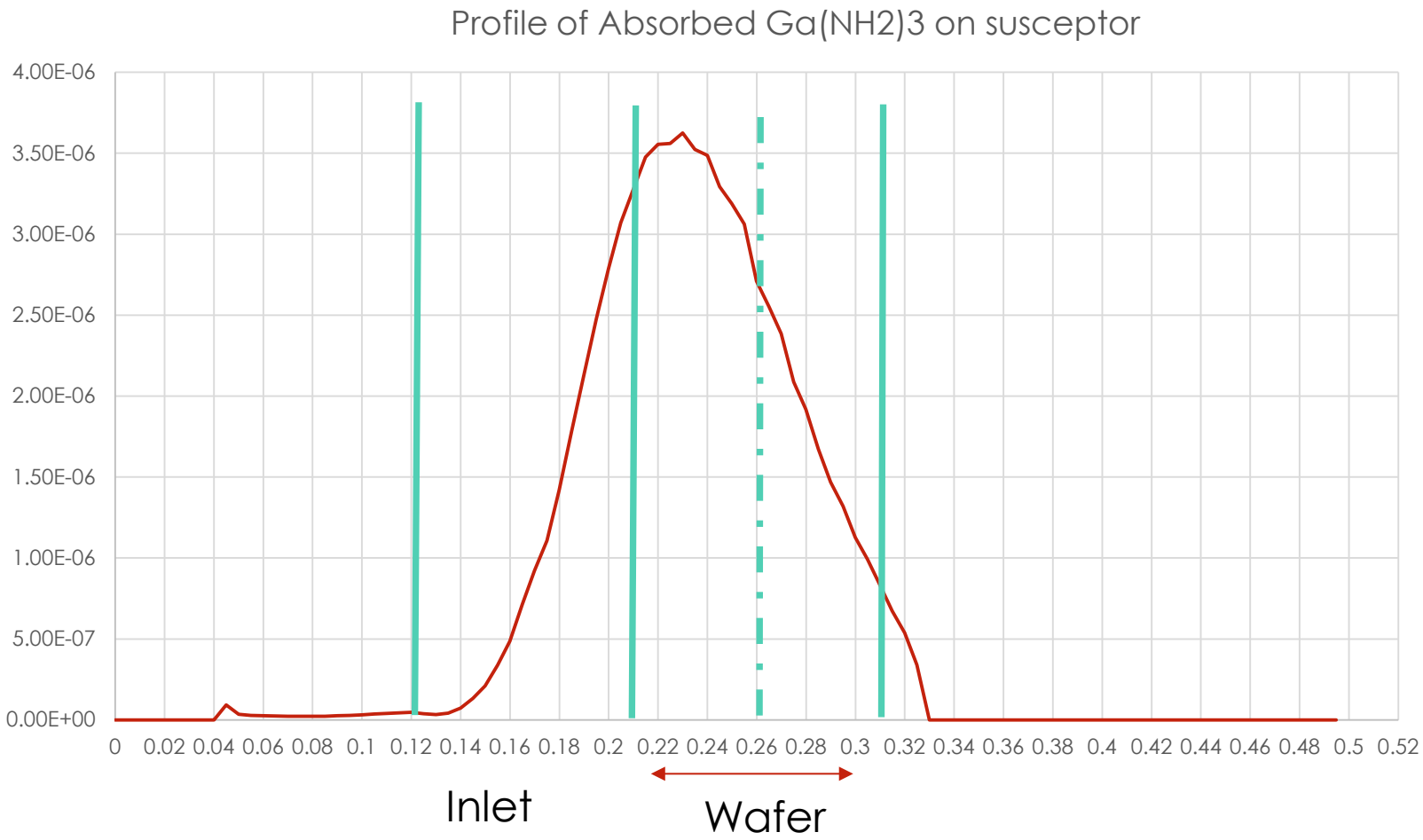
- These vortices actually encourage mixing of the unreacted TMGa.
- This actually results in a larger amount of produced $Ga(NH_2)_3$.
- This also results in incidental deposition on the upper surface of the reactor.
- These features agree with experiments performed by collaborators.

MOCVD SIMULATION



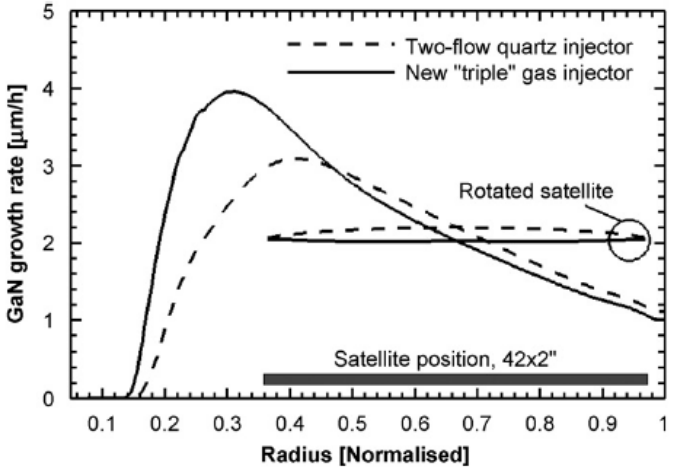
- These vortices actually encourage mixing of the unreacted TMGa.
- This actually results in a larger amount of produced $Ga(NH_2)_3$.
- This also results in incidental deposition on the upper surface of the reactor.
- These features agree with experiments.

MOCVD SIMULATION



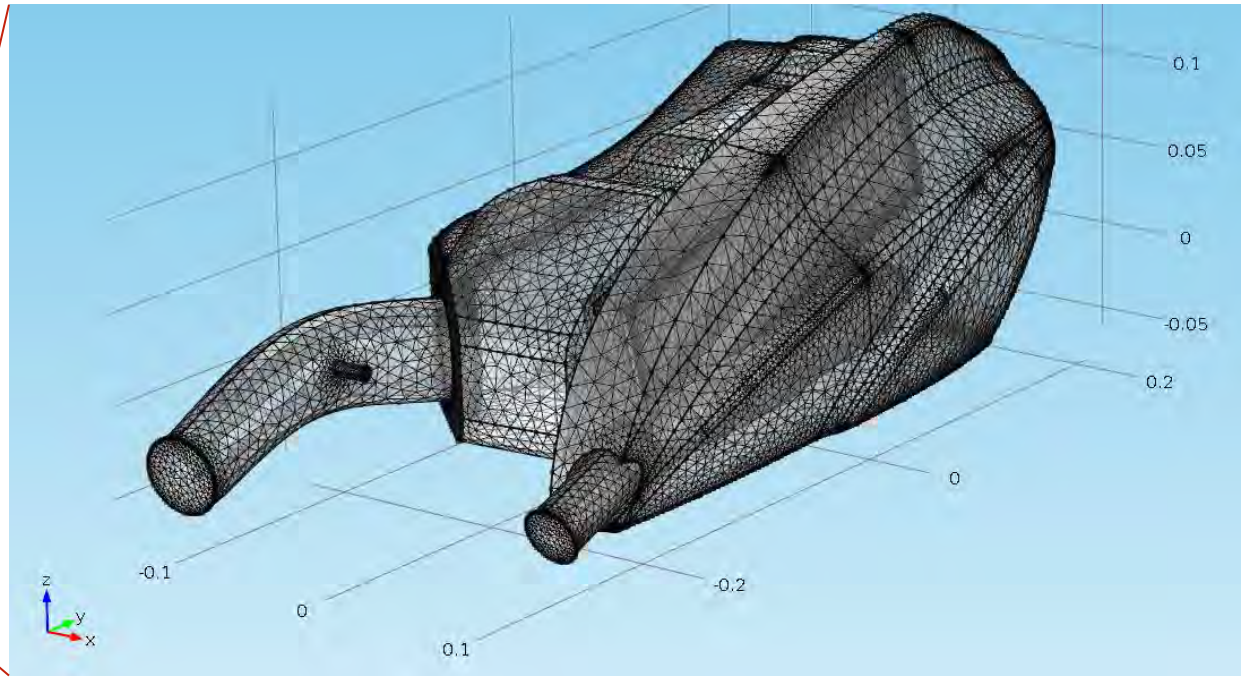
(GaNH2)3 absorbed by surface over 25k timesteps.

- The profiles produced by the simulations were (more or less) in line with what we saw in experiments.



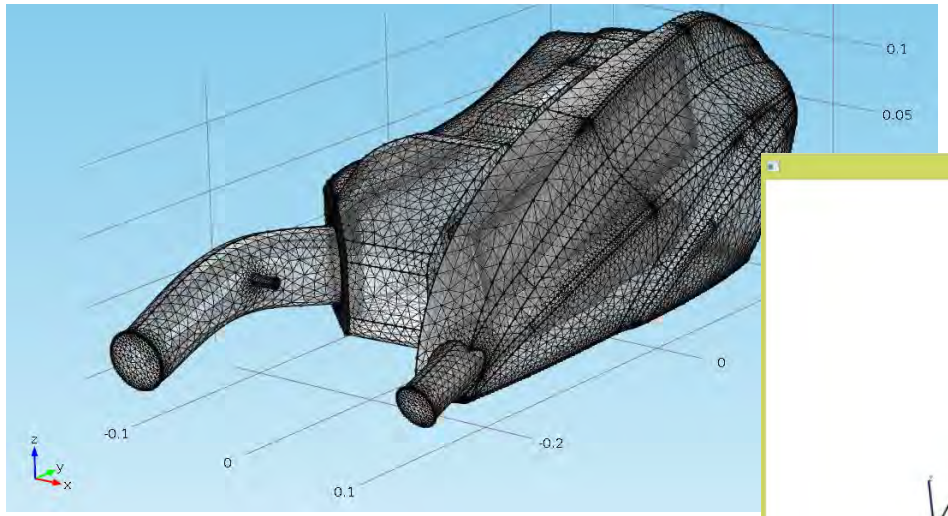
FLOW SIMULATION

- The solver is also being provided to Kymco as part of a long-term research project.

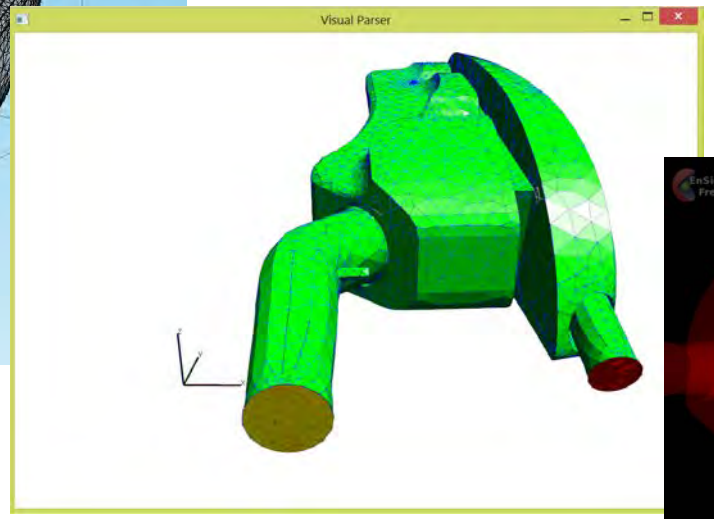


FLOW SIMULATION

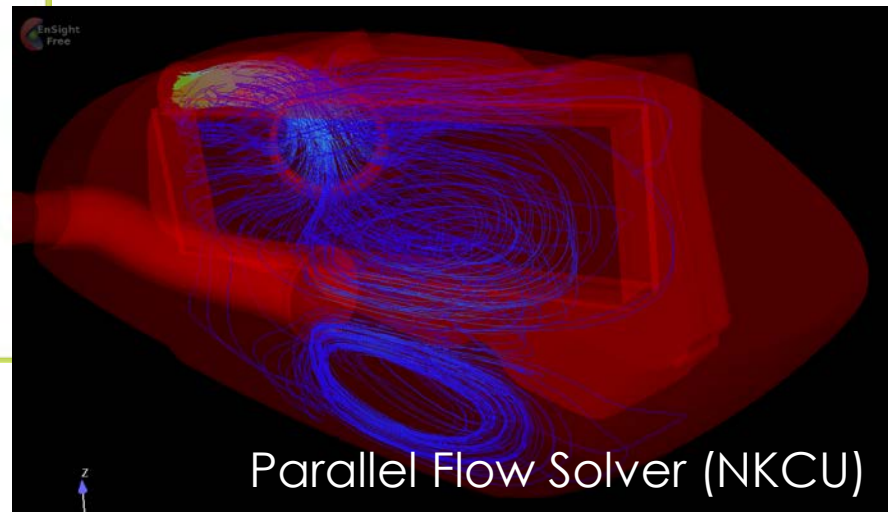
- The project is centered around a complete package for use by Kymco engineers as part of their design process.



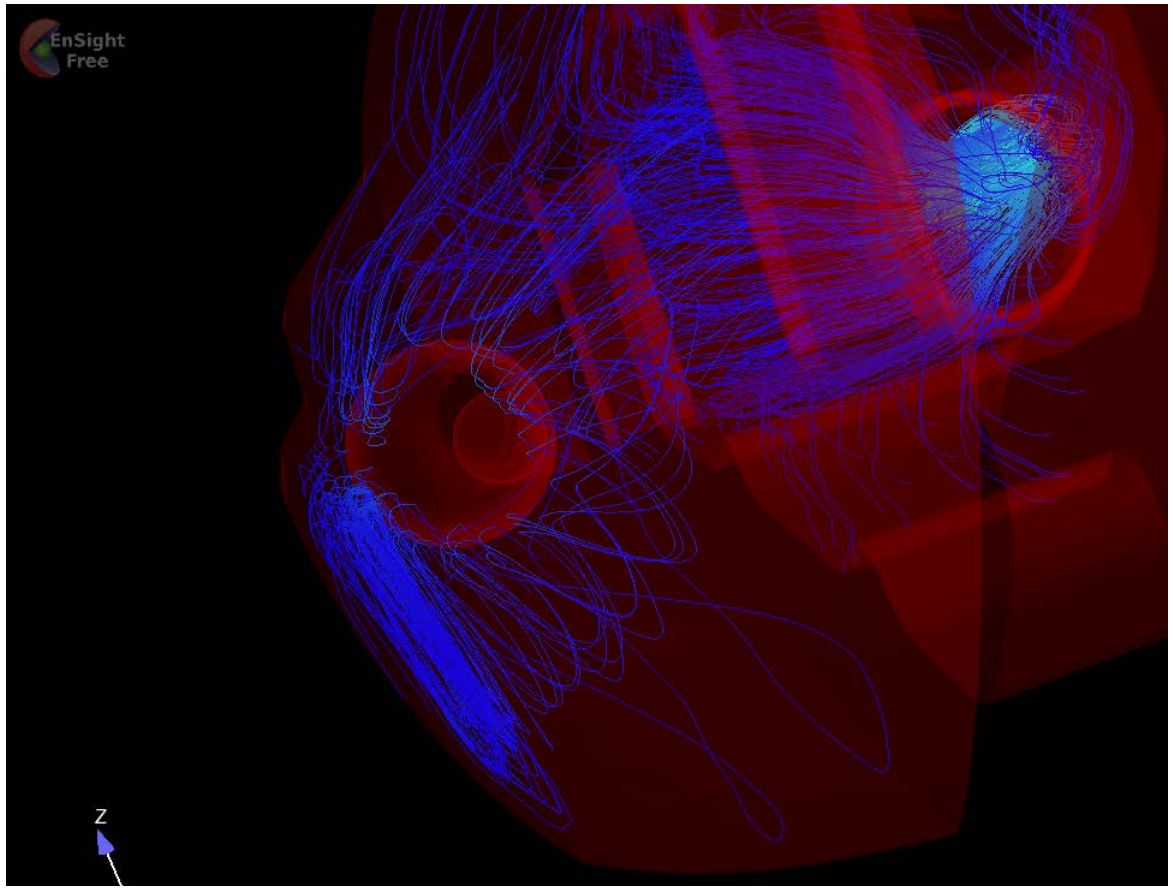
Mesh Generation (GMSH)

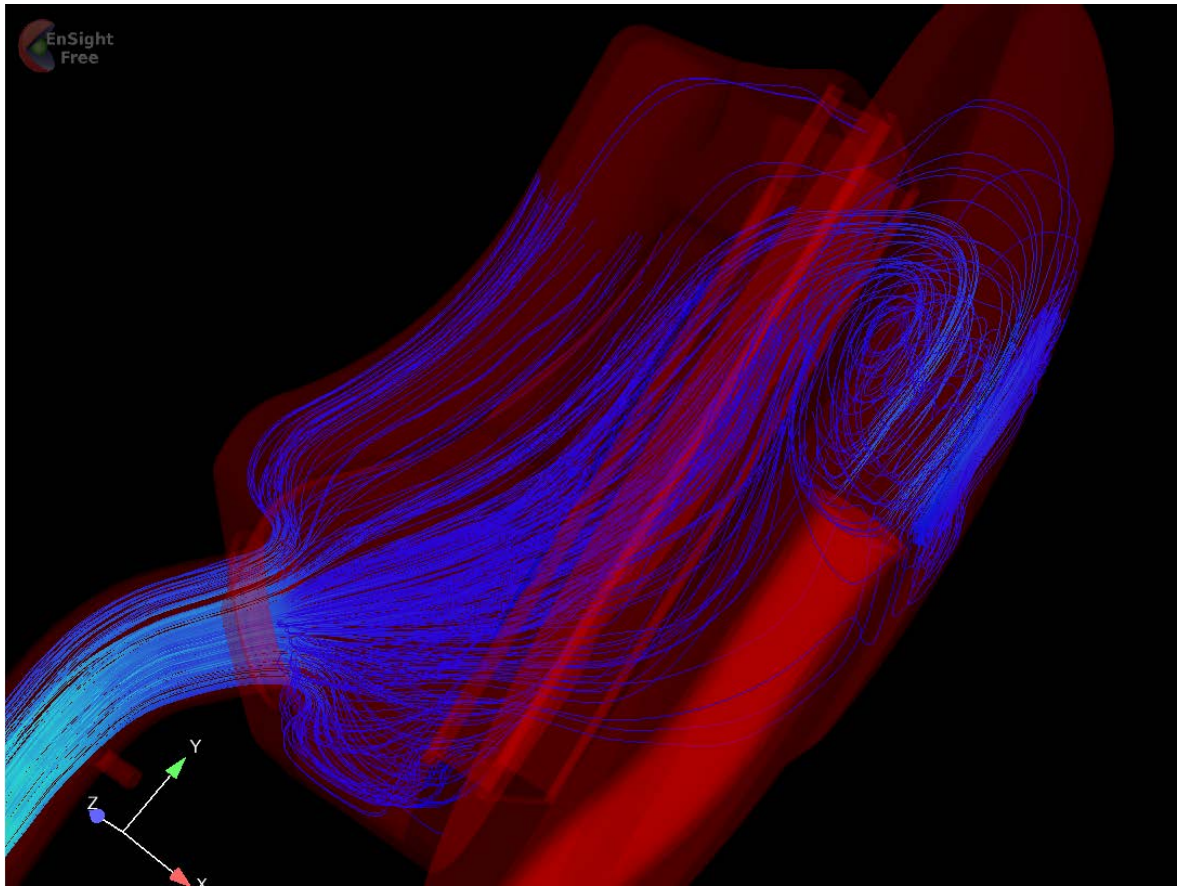


Boundary and Flow Condition (NCKU)



FLOW SIMULATION





The project (commenced March 2014) has already reproduced existing simulation and experimental results.

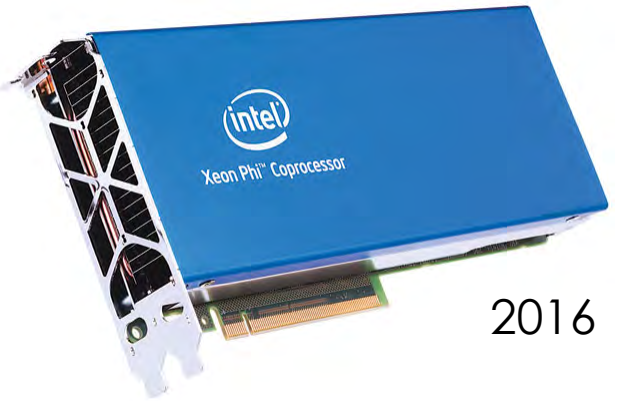
FLOW SIMULATION

- The solver is parallelized using several paradigms:
  - OpenMP / MPI hybrid (conventional model)
  - Multiple-GPU parallelization
  - Multiple Intel Phi parallelization

2015

- Two clusters are being constructed (in 2015 and 2016) at Kymco HQ and NCKU to power this solver.

- Preliminary results demonstrate approximately 60 times speed over the single CPU version of the code.



2016

PARALLEL PERFORMANCE (UNSTRUCTURED, SINGLE GPU)

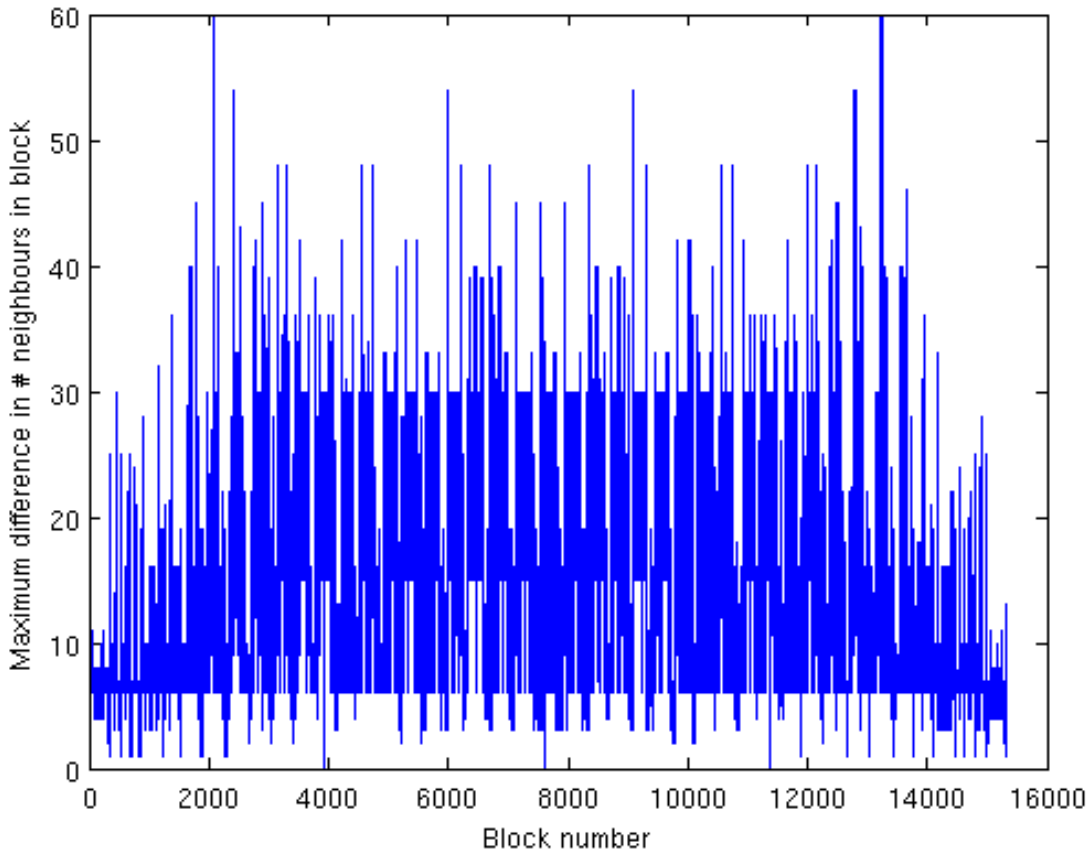
Device	Time	Speedup
Intel Xeon E5-2670 (1 core, SSE, -O3)	~ 57 hours	1x
Nvidia GTX-Titan (64/64/128) TPB	64 mins	~53x
Nvidia GTX-Titan (32/32/64) TPB	78 mins	~43x

Performance on various single GPU devices (double precision)
(Small test: 1.7 Million cells, 3 AMR levels, 25,000 steps, refresh each 1000 steps)

The primary reasons for the slow performance were (i) nesting within the OpenGL rendering loop, and (ii) rendering-related computations.

PARALLEL PERFORMANCE

- We also see a significant performance drop as a result of the change from structured to unstructured.
- Two reasons for this:
  - This is due to load unbalancing between blocks (10-20%)
  - Poorly structured memory access (80-90%)
- Re-ordering of cells plays some role in improving the performance.
- After removal of the graphical interface – utilization = 95.7%.
  - Flux computations: 69.3%, 86 registers/thread.
  - State update: 30.7%

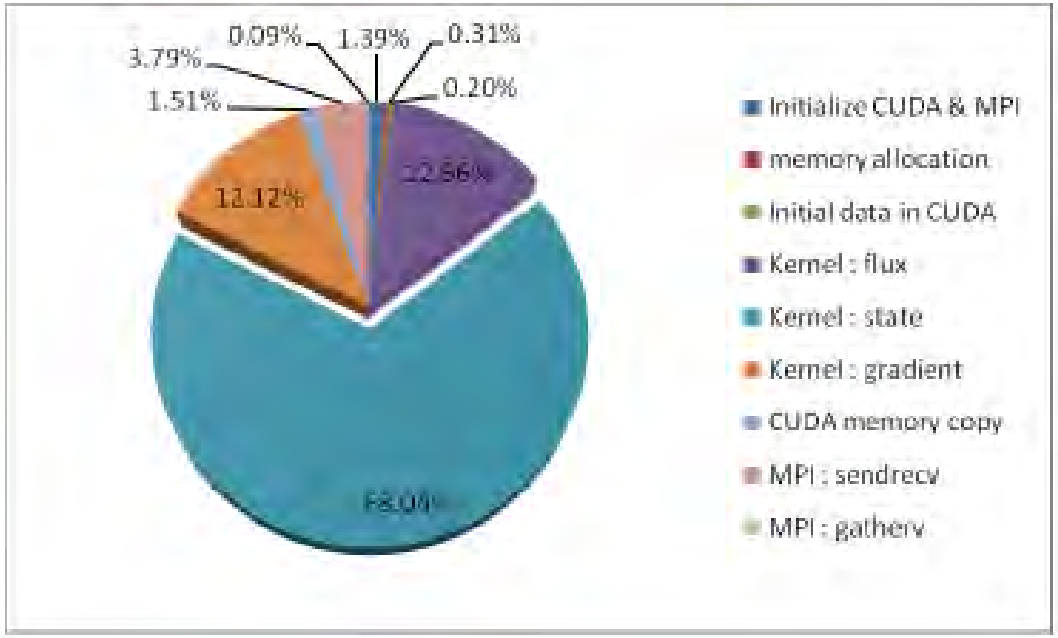


PARALLEL PERFORMANCE

- For 2nd order accuracy – comparison against previous multi-GPU code:

Solver	Fluxes (%)	State (%)	Gradient (%)
QDS (Structured)	~12%	~68%	~13%
Jacob's Riemann Solver (Unstructured)	**~57%**	~28%	~14%

- The fluxes for the approximate Riemann solver are much more complex.
- Our state function also does less work.
- Gradient computation on the unstructured grid is also more intensive.
- We are now either (i) throughput bound, or (ii) stuck due to thread divergence.



Breakdown of computational expense in previous QDS multi-GPU code using 32 GPU's on a 8 million cell problem.

CONCLUSIONS

- A conventional approximate Riemann solver has been employed in the simulation of (i) a 3D MOCVD reactor, and (ii) Kymco engine assembly.
- The large amount of thread divergence in the Riemann solver makes it non-ideal for application on the GPU → Approximately 3x-4x slower than a split flux solver. **Half of this is because of double precision**.
- Integration into an interactive real-time computation tool using OpenGL means we can see the flow field evolving – **at a cost**.
- Simulations of the simple 2-flow MOCVD injector revealed some unexpected flow features – which result in deposition profiles better explaining some of the experimental observations.
- Multiple-GPU performance scales well* (80% efficiency using 10 GPU devices) – but will not be discussed here.

QUESTIONS?

- Email me:

Matthew Smith (李汶樺) msmith@mail.ncku.edu.tw