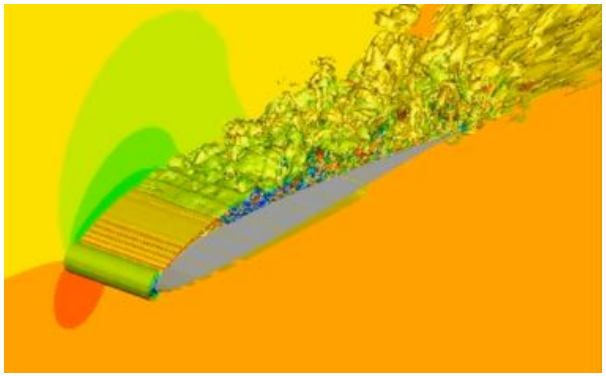
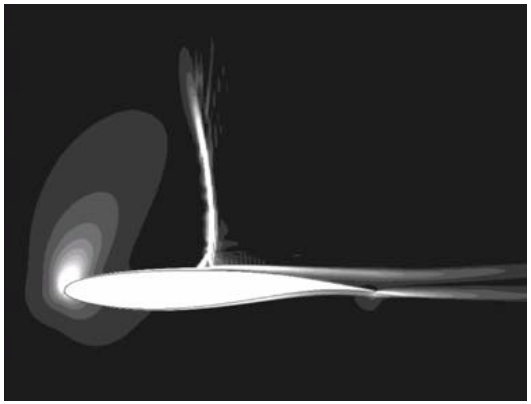
Turbulent Flow Solvers – Perspectives on HPC and Solver Implementations

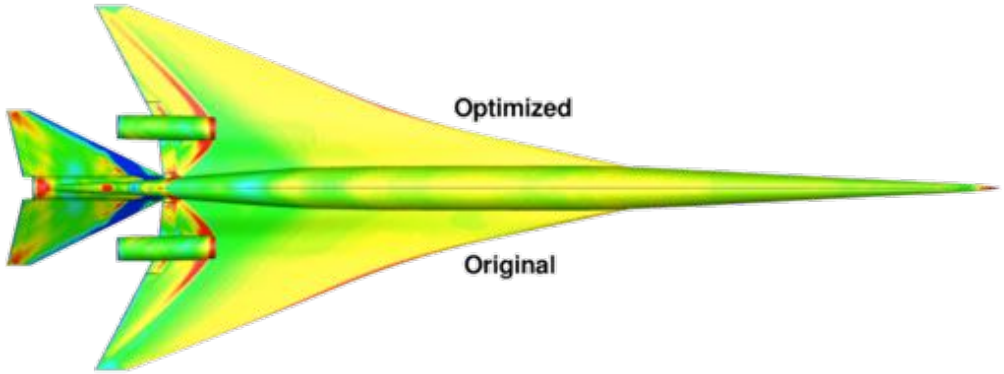
UMICHIGAN / NASA SYMPOSIUM ON ADVANCES IN TURBULENCE MODELING
ANN ARBOR, MI
JULY 12, 2017

SU2
The Open-Source CFD Code

Juan J. Alonso
aerospacedesignlab
Department of Aeronautics & Astronautics
Stanford University







Turbulent Flow CFD Applications Are Many…

- Computational methods to aid in engineering analysis and design are pervasive: higher importance in future

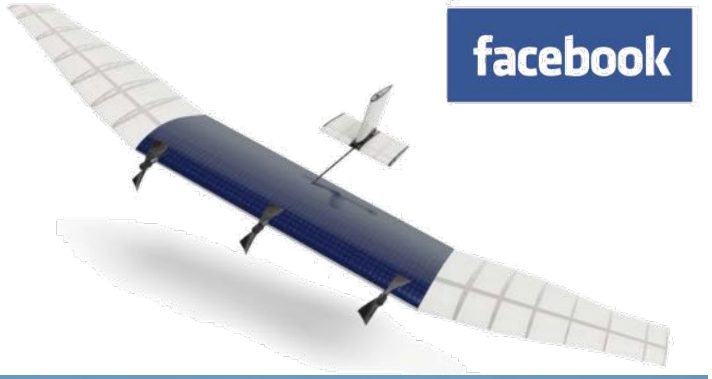
- Applications are many:
  - Gaining basic understanding of phenomena
  - Understanding multi-disciplinary interactions
  - Guiding design / optimization
  - Certification by Analysis / reducing margins for design

- Fundamental questions:
  - How do we embed high-fidelity methods in every-day design processes?
  - How do we automatically manage errors and uncertainties?
  - How do we leverage future computing power?
  - How do we show industrial relevance?
  - How good is good enough? And when?

ADL - Main Research Areas

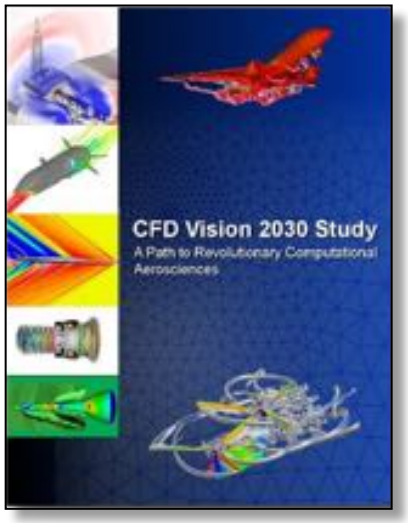
- Multidisciplinary Analysis and Optimization (MDO)
  - Managing tool fidelity / multi-fidelity approaches
  - Design under uncertainty (robust / reliability-based design)
  - Hierarchical decomposition methods
- Validation & Verification (V&V) and Uncertainty Quantification (UQ)
  - Management of numerical errors
  - Propagation of natural variability uncertainties
  - Understanding of model-form uncertainties
- System/Vehicle-Level Implications
  - Problems are not just at the component level
  - System-level interactions are fundamental
  - Stochastics, interactions, strategic players/actors

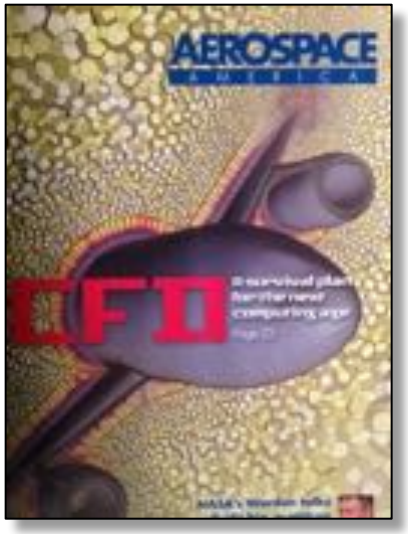
Problems of Interest … In Pictures



CFD Vision 2030 Study

- Emphasis on physics-based, predictive modeling
  - Transition, turbulence, separation, unsteady/time-accurate, chemically-reacting flows, radiation, heat transfer, acoustics and constitutive models
- Management of errors and uncertainties
  - Quantification of errors and uncertainties arising from physical models, mesh and discretization, and natural variability
- Automation in all steps of the analysis process
  - Geometry creation, meshing, large databases of simulation results, extraction and understanding of the vast amounts of information
- Harness exascale HPC architectures
  - Multiple memory hierarchies, latencies, bandwidths, programming paradigms and runtime environments, etc.
- Seamless integration with multi-disciplinary analyses and optimizations
  - High fidelity CFD tools, interfaces, coupling approaches, the science of integration, etc.





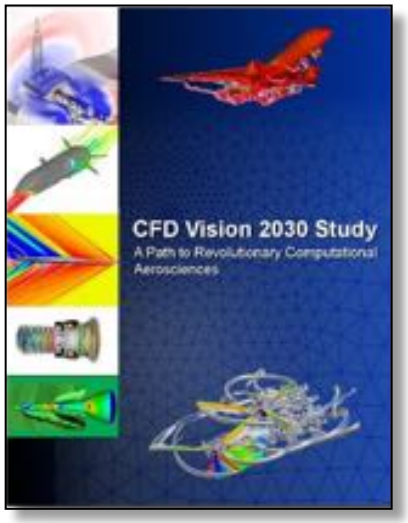
Slotnik, et al., "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences," NASA/CR-2014-218178, 2014

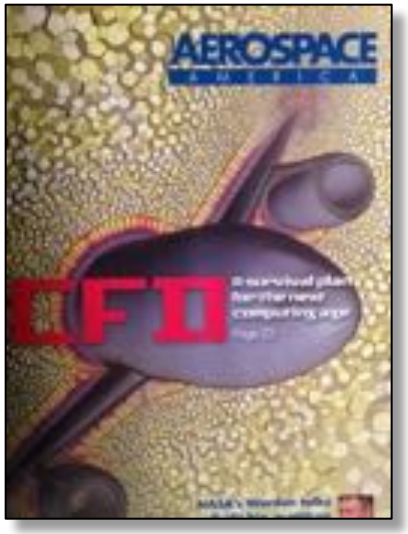
NASA BOEING

Stanford University

CFD Vision 2030 Study

- **Emphasis on physics-based, predictive modeling**
  Transition, turbulence, separation, unsteady/time-accurate, chemically-reacting flows, radiation, heat transfer, acoustics and constitutive models
- **Management of errors and uncertainties**
  Quantification of errors and uncertainties arising from physical models, mesh and discretization, and natural variability
- **Automation in all steps of the analysis process**
  Geometry creation, meshing, large databases of simulation results, extraction and understanding of the vast amounts of information
- **Harness exascale HPC architectures**
  Multiple memory hierarchies, latencies, bandwidths, programming paradigms and runtime environments, etc.
- **Seamless integration with multi-disciplinary analyses and optimizations**
  High fidelity CFD tools, interfaces, coupling approaches, the science of integration, etc.



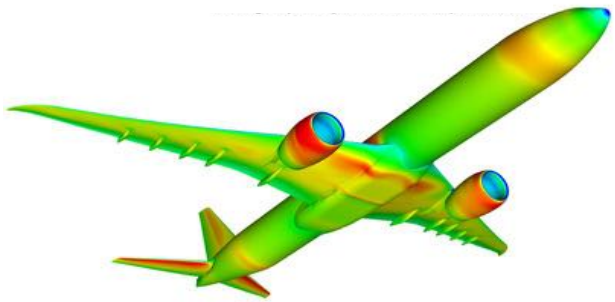


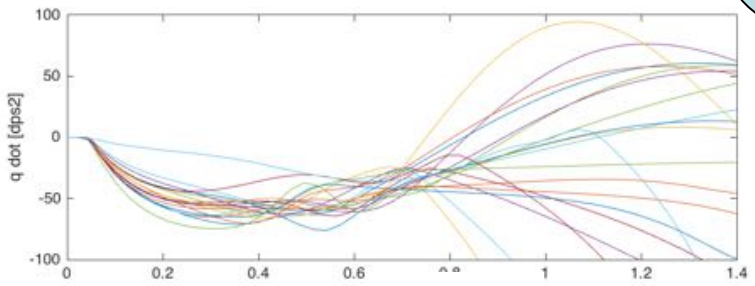
Slotnik, et al., "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences," NASA/CR-2014-218178, 2014

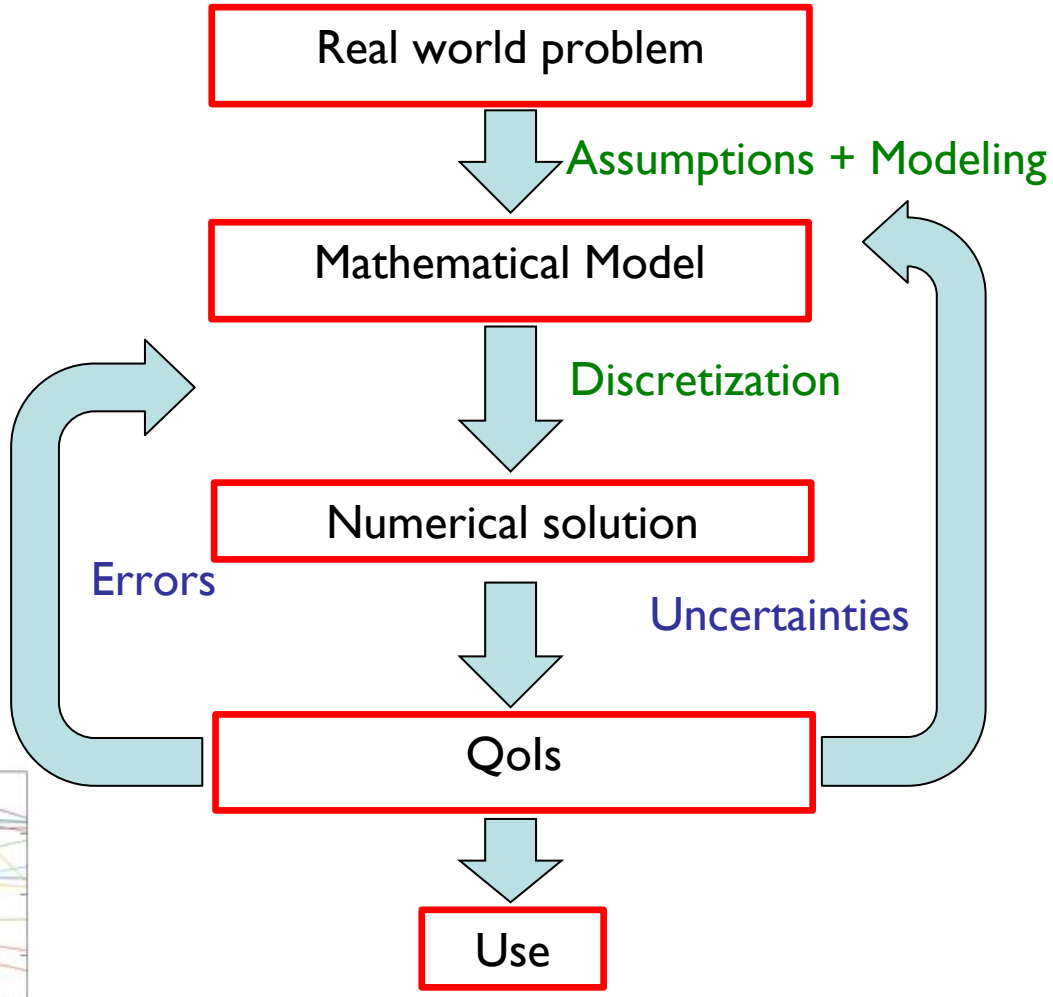
NASA BOEING Stanford University

Predictive Computational Science: V&V and UQ

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla.(\rho \mathbf{u}\mathbf{u}) = -\nabla p + \nabla.\tau$$







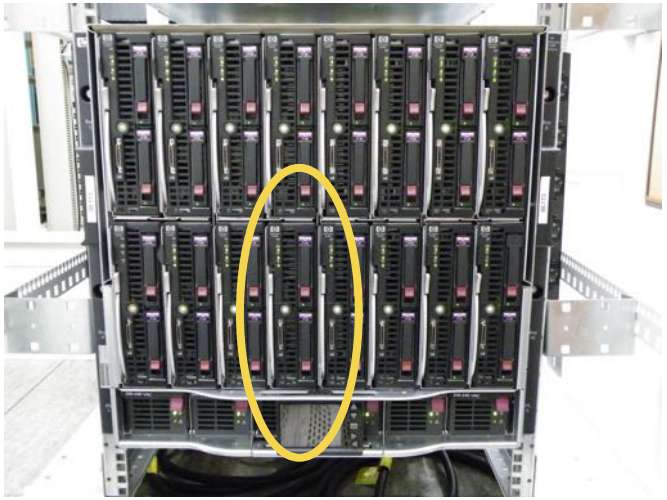
Quantifying discretization errors is a first step to quantify sources of uncertainty.

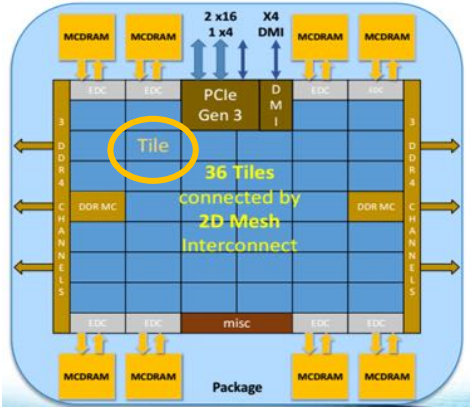
Understanding uncertainties is necessary to achieve certification.

Physical Modeling Issues in Future CFD Solvers

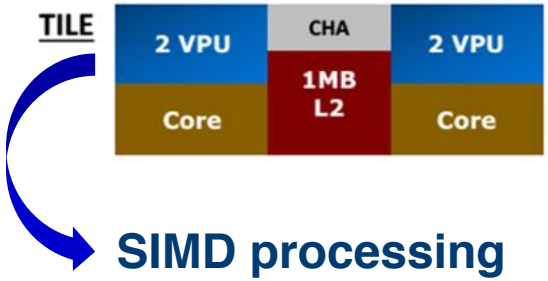
- RANS model based:
  - Robustness improvements
  - Speed of convergence (steady and unsteady RANS)
  - Overall simulation cost
  - Complex geometry representation
  - Sensitivities
- Scale-resolving methods:
  - Low-dissipation spatial discretizations
  - Accurate time-stepping methods
  - Subgrid-scale and turbulence models, wall models
  - Sensitivities of unsteady flows!
- Data-driven solvers:
  - All of the above plus...
  - Rapid embedded querying of “learned” portions of the model
  - Advanced model data decomposition techniques (for very large databases)
  - Offline: scalable machine learning tools common to other communities

Compute Hierarchy in Hardware

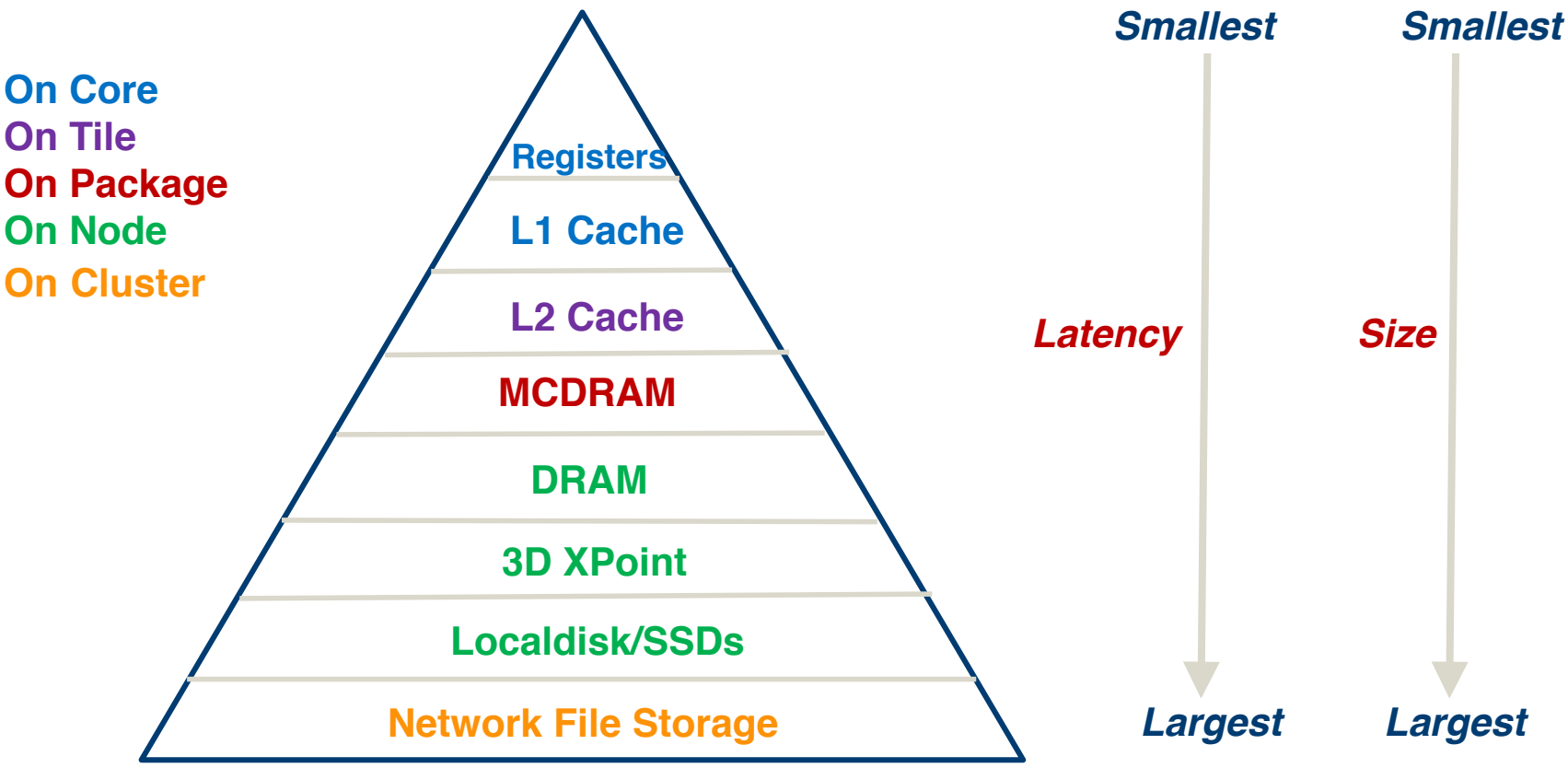




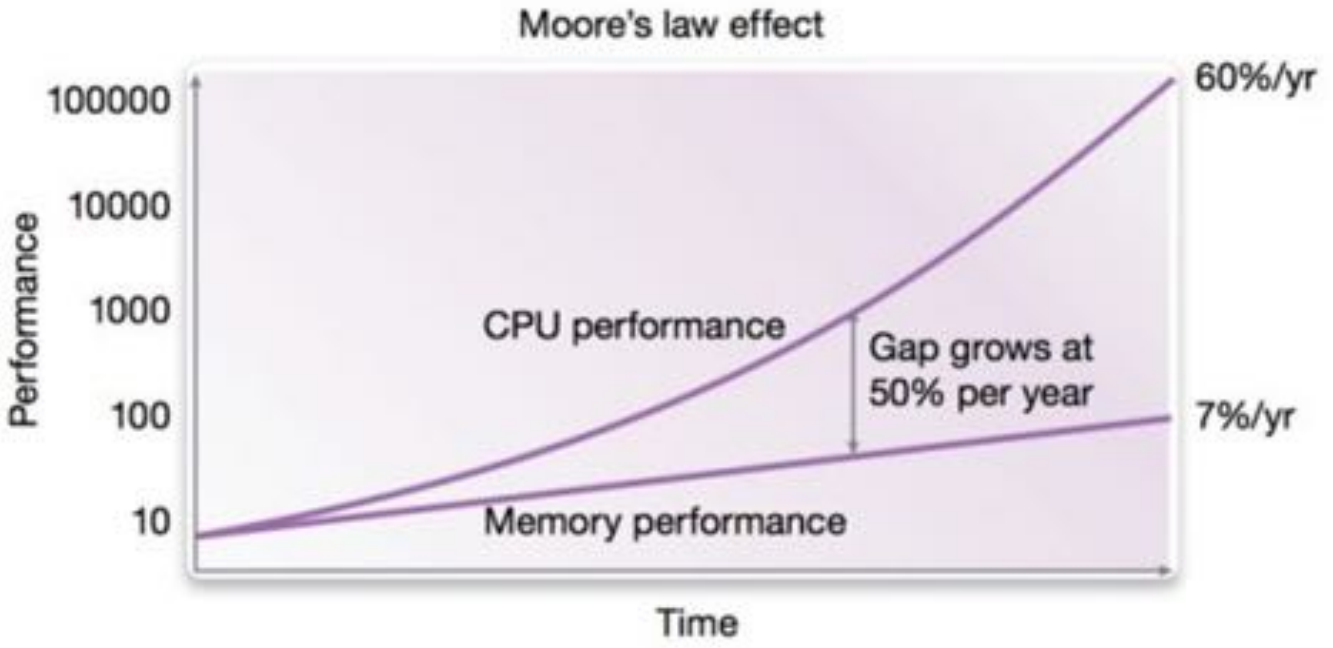
SPECIFICATIONS	
GPU Architecture	NVIDIA Pascal
NVIDIA CUDA® Cores	3584
Double-Precision Performance	4.7 TeraFLOPS
Single-Precision Performance	9.3 TeraFLOPS
Half-Precision Performance	18.7 TeraFLOPS

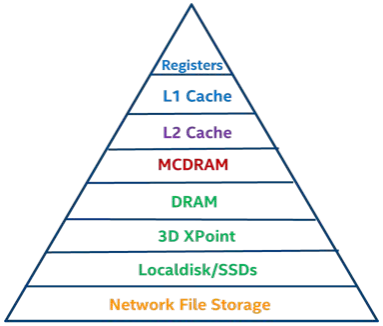


Memory Hierarchy in Hardware



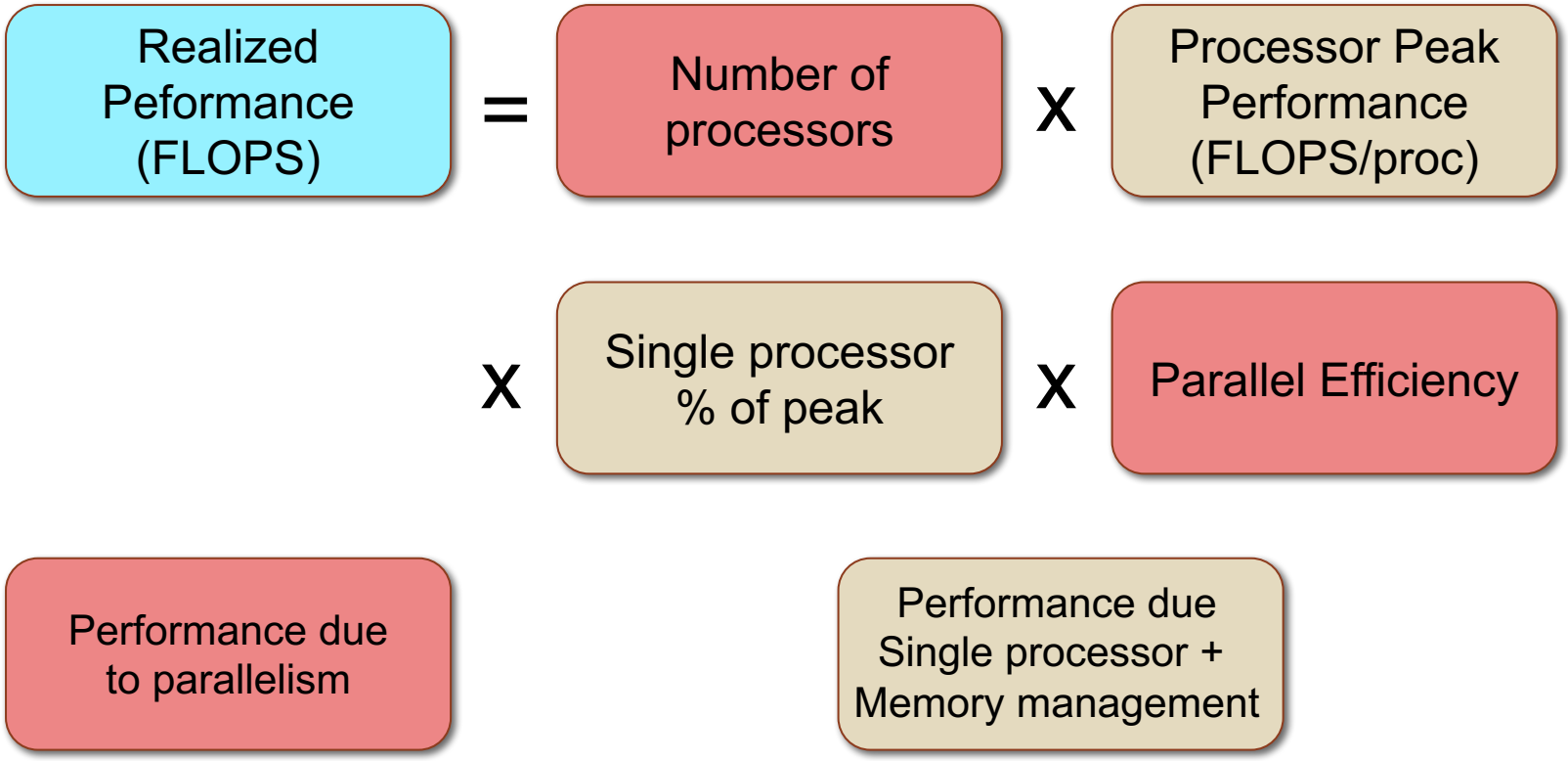
The *Compute-Memory* Gap





*Source: extremetech.com

The Performance Equation…



- Must tackle all of these elements in order to obtain scalable and performant code
- A non-trivial effort … that nobody is interested in!

But Progress IS Slow…

- These are long-term research issues…
- … but it still takes a significant amount of time before we can transition turbulence research/HPC results to industrial applications
- We have been doing an "experiment":

  SU2
  The Open-Source CFD Code

  - **SU2: Analysis and Design Optimization in Complex Configurations**
- Are community codes well placed to accelerate the transition of turbulence research and HPC implementations?

What is SU2?

The SU2 suite is an **open-source** collection of C++ based software for multi-physics simulation and design on unstructured meshes (i.e., CFD!).

First and foremost: a Community code!

SU2 is under active development at Stanford University in the Department of Aeronautics and Astronautics and **in many places around the world**.

http://su2.stanford.edu

https://github.com/su2code/SU2

2015 SU2 Team, "SU2: An Open-Source Suite for Multi-Physics Simulation and Design," AIAA Journal, 2015, doi: 10.2514/1.J053813**.**

2012 SU^2 team, "Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design", AIAA Paper 2013-0287.

2013 SU^2 team, "Stanford University Unstructured (SU2): Open-source analysis and design technology for turbulent flows", AIAA Paper 2014-0243.

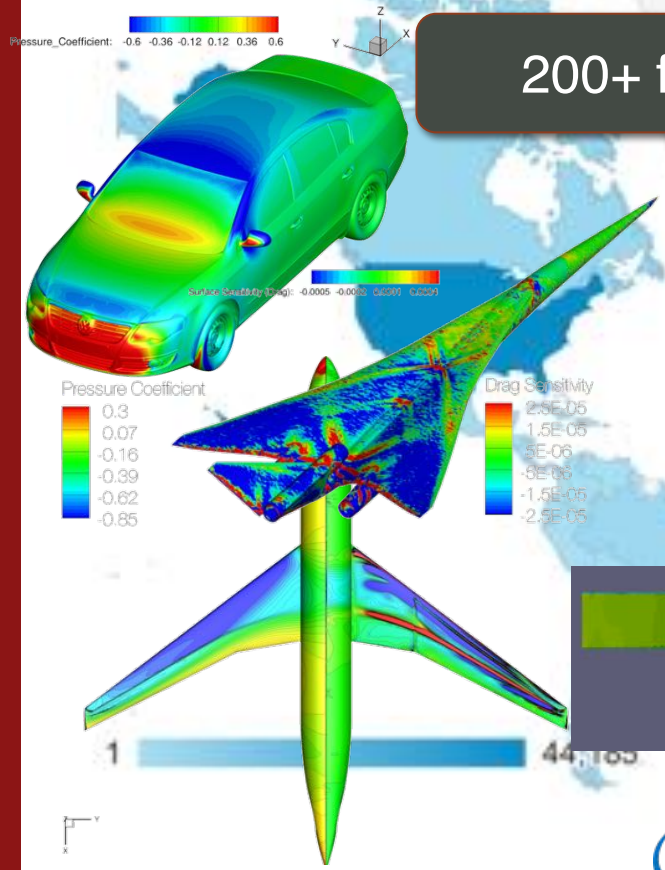
SU2 – Multi-Physics Analysis and Design

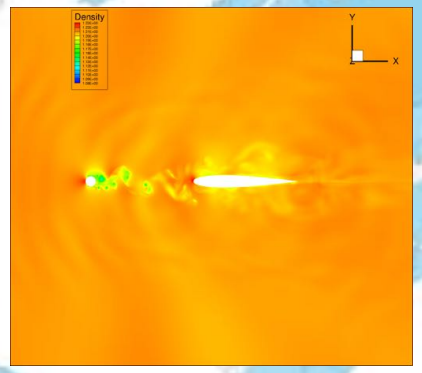
SU2
The Open-Source CFD Code

Since release in Jan. 2012: 500,000+ web visits across 178 countries

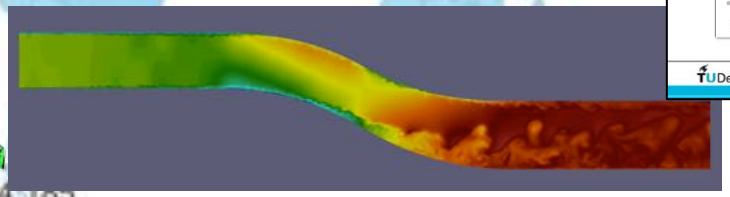
10s of thousands of downloads, 15,000+ email addresses on the user list

200+ forks, 170+ developers in list

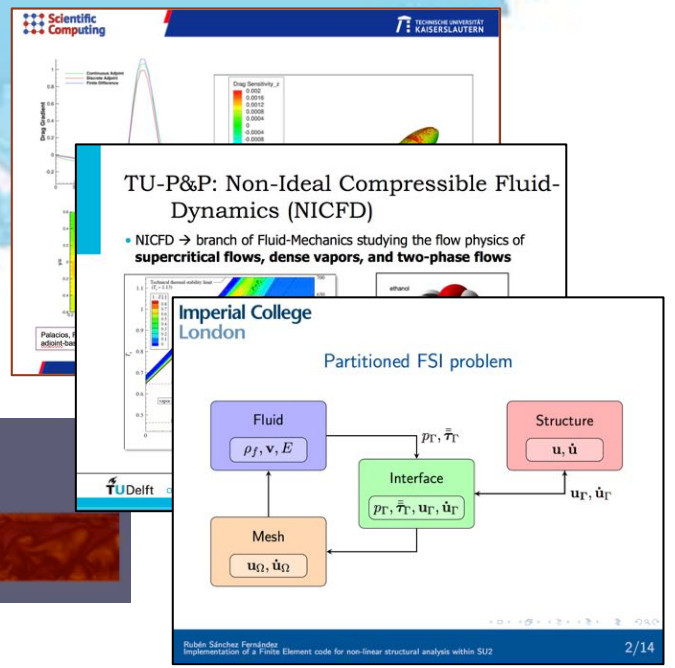




DDES + FWH

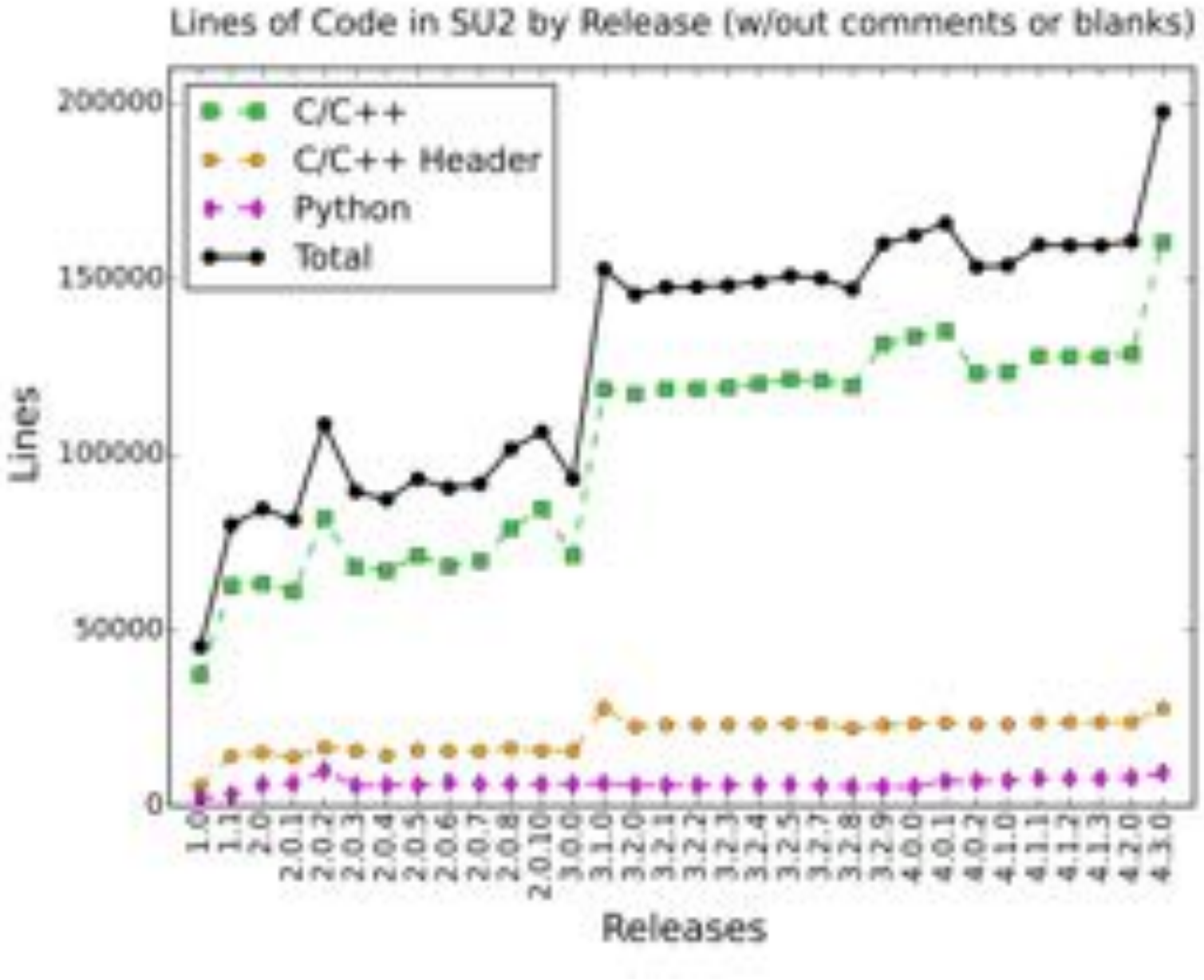


DG-FEM Higher-Order Solver



intel NASA

Stanford University



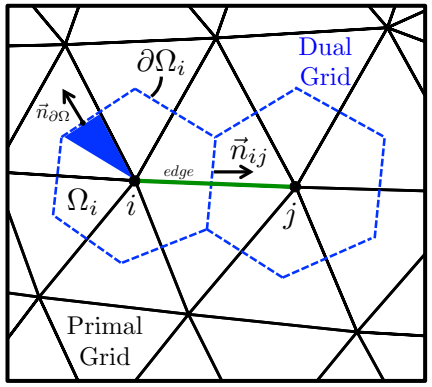
*includes code in externals/

Now, It Is Not All About Research…HPC!

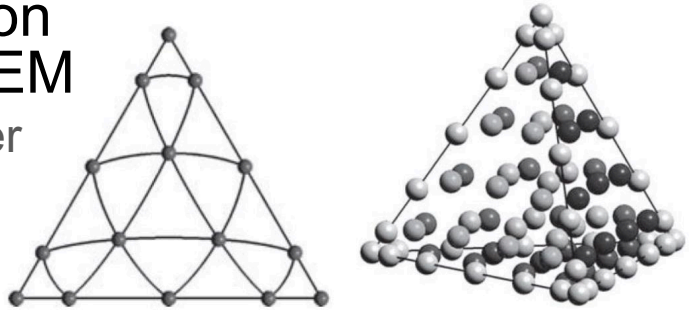
- Based on several recent experiences:
  - Intel Parallel Computing Center (IPCC) at Stanford
  - Argonne National Lab, Theta Early Science Program (ESP)
  - Argonne National Lab, Aurora Early Science Program (ESP)
- Can community codes also accelerate transition of optimized HPC implementations?
- The effort (and knowledge) required to obtain scalable and performant CFD codes has increased substantially:
  - Large teams with varied expertise now required
  - Expertise rarely resides at Universities any more

A Tale of Two Solvers In SU2…

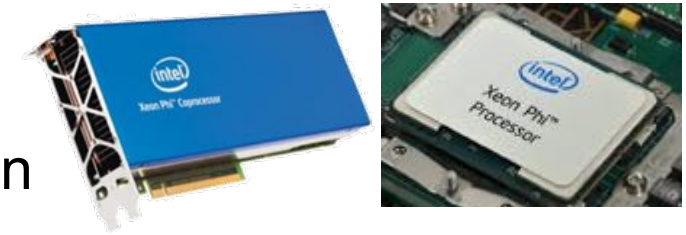
- Workhorse of SU2 is a the 2nd order FV solver: SU2 FV
  - Unstructured mesh, edge-based
  - Median-dual control volumes
  - Many flux discretizations and reconstructions
  - RANS (SA/SST, DDES variants)



- For the past 2 years, we have been developing a DG-FEM solver based on the ”same” infrastructure: SU2 DG-FEM
  - Element-wise data structure, arbitrary order
  - Flux computations at faces
  - Higher arithmetic intensity
  - *feature_hom*, to be released this summer
  - Targeted to LES and WMLES



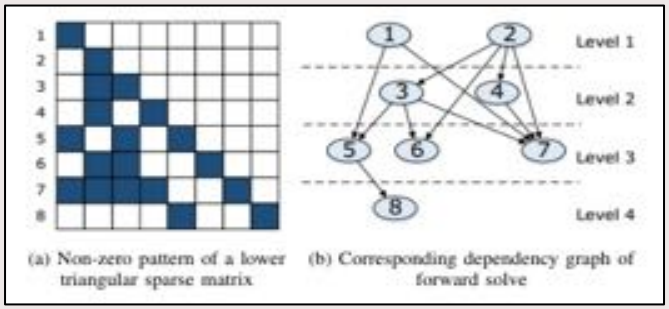
- Multi-year efforts, jointly with Intel, to improve performance and scalability on Intel Xeon Phi (KNL) platforms



Compute Patterns in SU2 FV

Compute Pattern	Challenges	Optimizations

Edge-based Loops: capture the actual physics of the problem (residuals & Jacobians computation)	Partitioning, retaining spatial locality
	Loop-carried dependencies
	Irregular - mem accesses, working-sets, write contention
		for all Edges **do**
		vertices forming the edge -$[v_1, v_2]$
		$y[v1] \; += \mathbf{F}(V[v1], V[v2])$
		$y[v2] \; += \mathbf{F}(V[v2], V[v1])$
		end for

Sparse recurrences: sparse linear algebra kernels (Solution Ax = b)	Limited parallelism, extracting concurrency Large load imbalances Low arithmetic intensity, BW-bound Irregular - mem accesses, working-sets	 (a) Non-zero pattern of a lower triangular sparse matrix (b) Corresponding dependency graph of forward solve
Collectives & Sync	Algorithm characteristics, Network bound	
Vertex-based loops	Vertex-wise concurrency, Low compute intensity, Bandwidth bound	

Performance Optimization on Modern Platforms

Hierarchical Parallelism

Coarse-Grained / multi-node
Domain decomposition

Fine-Grained Parallelism / within node
Sub-domain: 1) Multi-level domain decomposition (ex. 2-levels of METIS calls) 2) Data decomposition (coloring)

MPI-scaling

Improve load balancing

Reduce synchronization events, all-to-all comms

Utilize all the cores

OpenMP, MPI, TBB…

Reduce synchronization events, serial code

Improve load balancing

Vectorize/SIMD

Unit strided access per SIMD lane

Outer-loop vectorization

Array notation (:), Elemental functions

Efficient memory/cache use

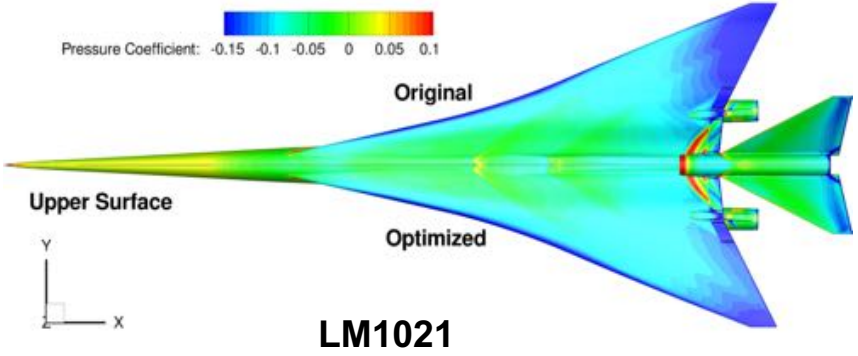
Reduce working sets when possible:

- Reorder "sparseness" – RCM etc.
- AoS to SoA

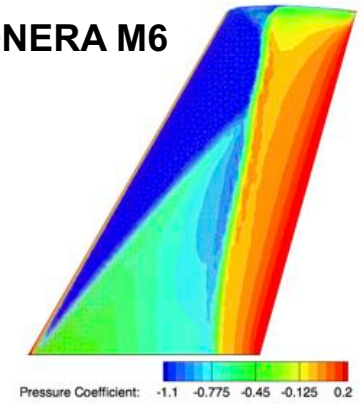
Cache Blocking

Prefetching

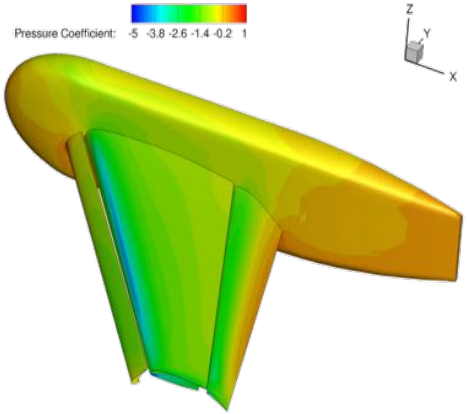
SU2 Performance Optimization Test Cases



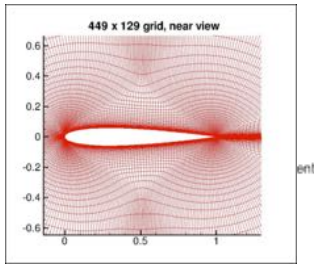
LM1021



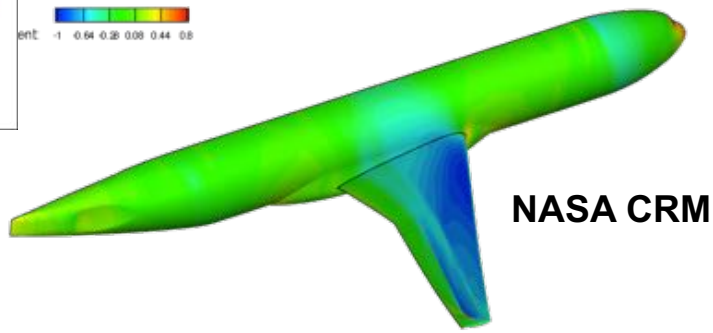
ONERA M6



NASA Trap Wing



NACA 0012



NASA CRM

We are now routinely running large-scale parallel computations with 5000+ ranks for numerical performance experiments

Key Lessons Learned – SU2 FV Solver

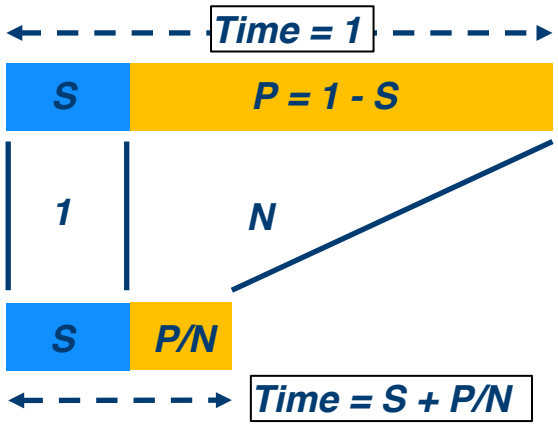
- ***Edge-loops/Face-loops and Sparse linear algebra*** – bulk of execution time
  - Primary overheads: Irregular access, limited parallelism, load-imbalance

- Imperative to exploit ***fine-scale concurrency*** among cores & ***SIMD***
  - Sub-domain decomposition vs. data decomposition
  - Avoid synchronizations (atomics, barriers…)
  - SIMD: Unit-strided memory access, outer-loop vectorization

- ***Memory optimizations***
  - Most kernels in FV CFD codes are memory BW bound
  - Memory efficient algorithms – RCM reordering
  - AoS-to-SoA transformations – Compact working sets
  - Prefetching

Performance Modelling – *How high can we go?*

Amdahls's Law for Multicores



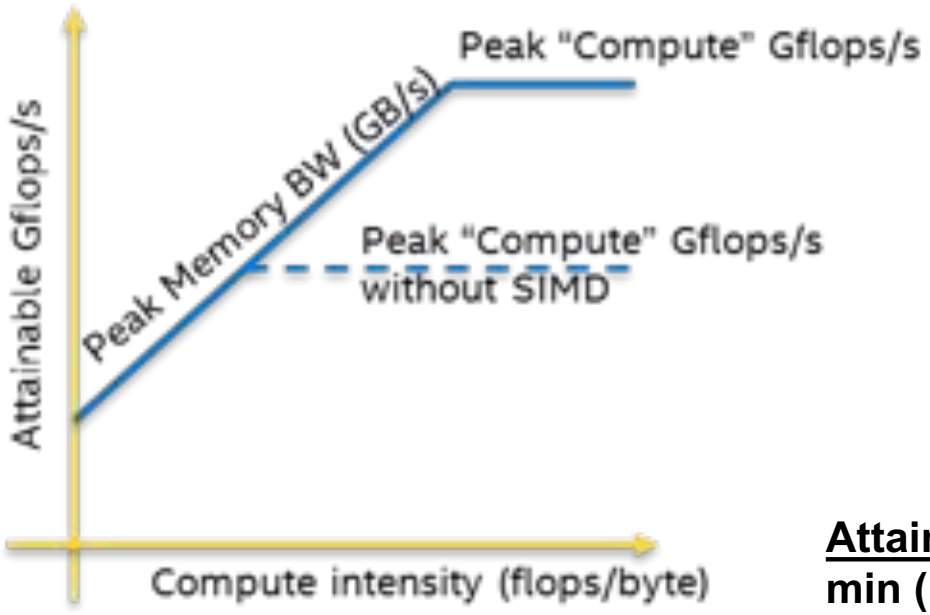
$$Speedup = \left(\frac{1}{Serial_{frac} + \frac{1 - Serial_{frac}}{\mathbf{NumCores}}} \right)$$

Amdahls's Law for Vector Multicores

$$Speedup = \left(\frac{1}{Serial_{frac} + \frac{1 - Serial_{frac}}{\mathbf{NumCores}}} \right) * \left(\frac{1}{Scalar_{frac} + \frac{1 - Scalar_{frac}}{\mathbf{VectorLength}}} \right)$$

Ideal Speedup: NumCores*VectorLength (requires zero scalar, zero serial work)

Performance Modelling – *Roofline Model*

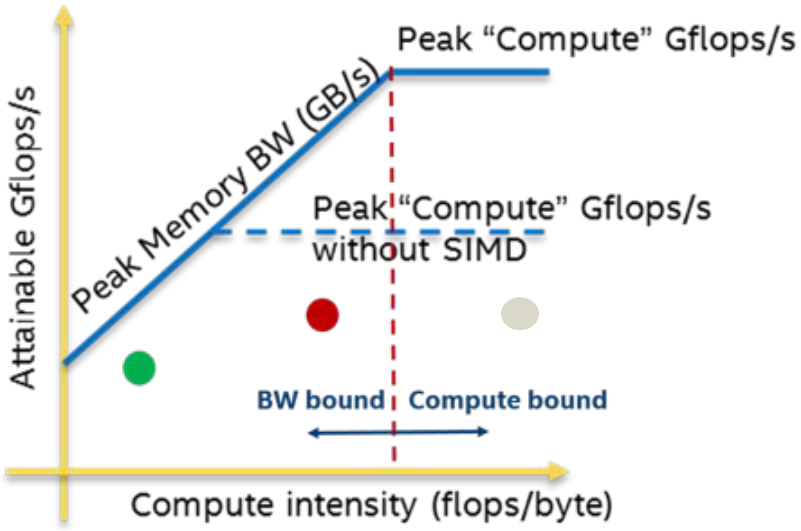


Attainable Gflops/s =
min (Peak Gflops/s, Stream BW * flops/byte)

Roofline Model – Crossover Point

Stream BW * CI = Peak Compute flops/s

Example: 400 * CI = 2000 => CI = 5



Examples:

a) Stream Triad (DAXPY)

```
for (i = 0; i < N; ++i) {
 z[i] = a*x[i] + y[i];
}
```

Bytes accessed: 3 * 8 = 24
Flops: 1 + 1 = 2
CI = 2/24 = 0.084

a) Matrix multiplication – C = A x B
(Assume square)

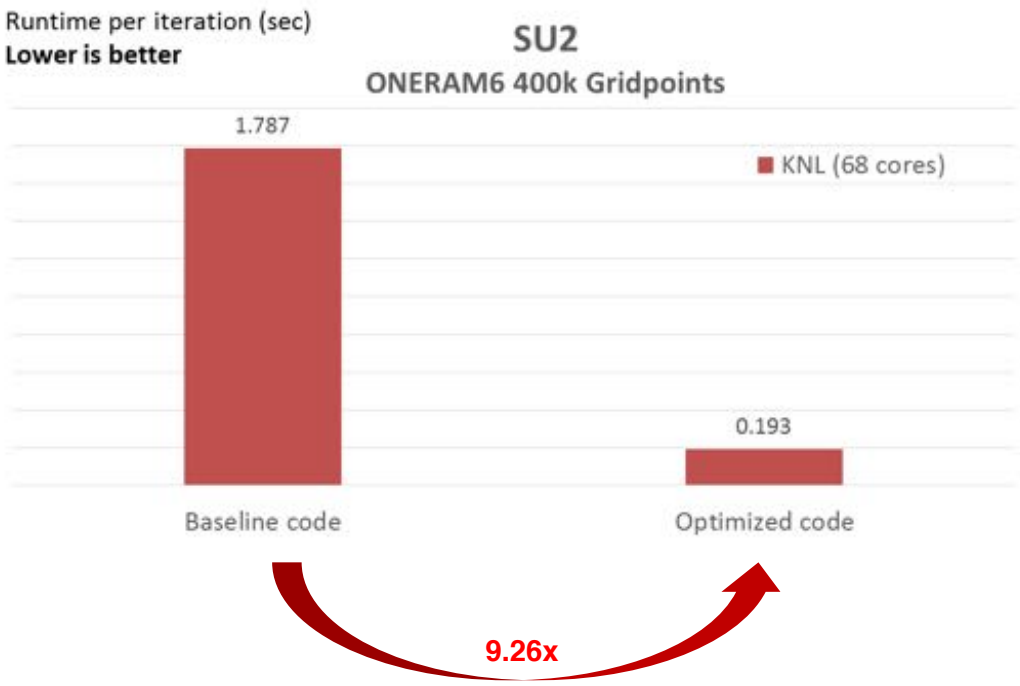
```
for (i = 0; i < N; ++i) {
  for (j = 0; j < N; ++j) {
    for (k = 0; k < N; ++k) {
      C[i][j] += A[i][k] * B[k][j];
}}}
```

Bytes accessed: 8*(N*N + N*N + N*N)
Flops: 2*N*N*N
CI = $2N^3/(24N^2)$ = N/12

SU2 Performance Improvements through:

- Using the ***cores***
- Using ***SIMD***
- Efficiently using the ***memory***

SU2 FV Performance Improvements



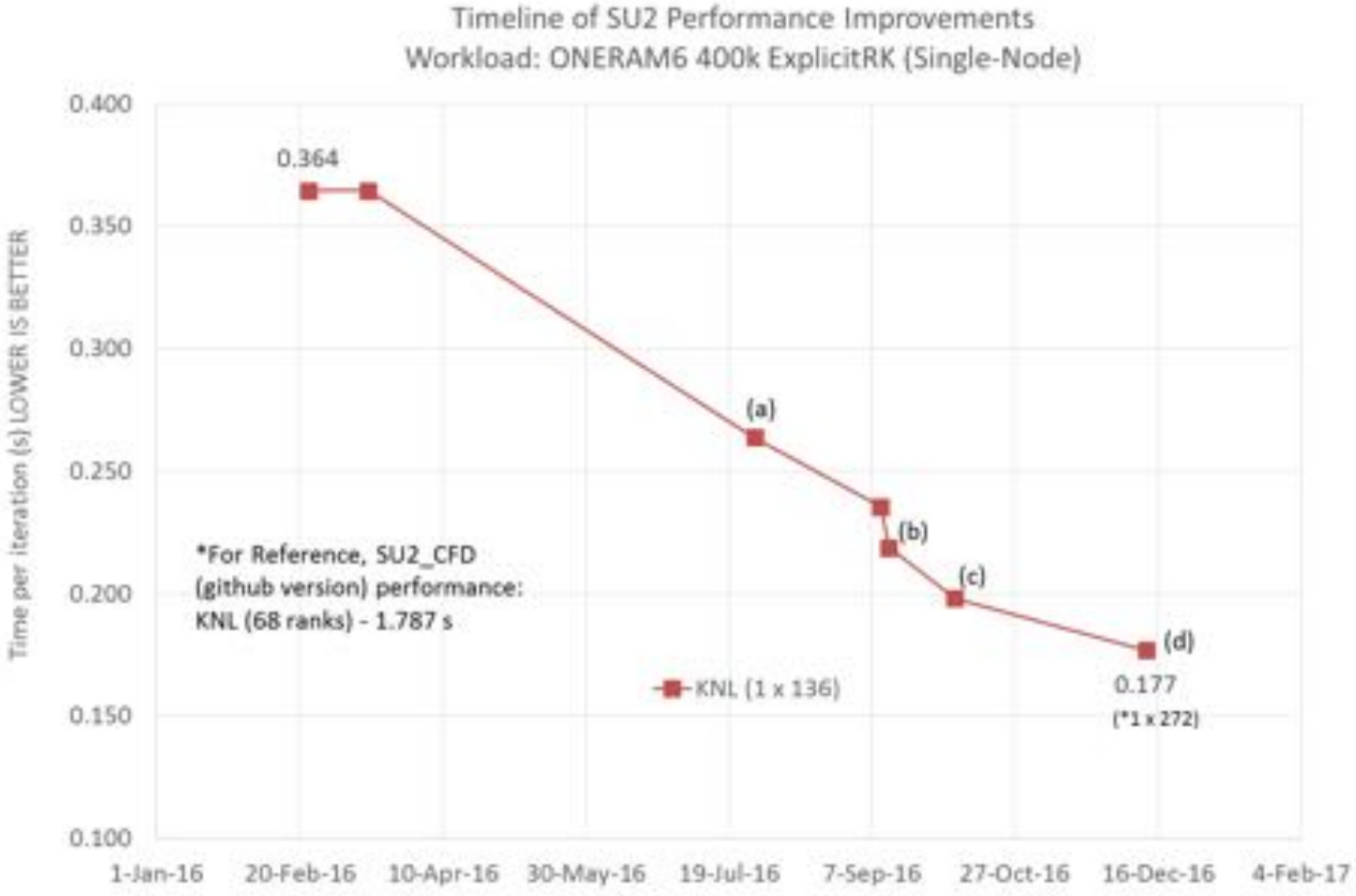
Key Optimizations performed:

1. Utilizing the ***Cores*** fully:
   - Hybridized the code with MPI+OpenMP
   - High-level OpenMP

2. Efficient ***vectorization***:
   - Reduced gathers-scatters in SIMD loops
   - Outer-loop vectorization

3. Efficiently using the ***memory***:
   - AoS-to-SoA layout change
   - RCM re-ordering

Timeline of Performance Improvements



Major optimization milestones:

(a) + Vectorized Viscous_Residual kernel, changed layout of Gradient_Primitive to be more cache/simd friendly, hard-coded some loop trip counts, other general optimizations

(b) + Added routine for edge position vectors (instead of getting Coord_i and Coord_j), several other general optimizations

(c) + Manual fusing of viscous and convective residual edge loops, data and compute layout changes in residual kernels

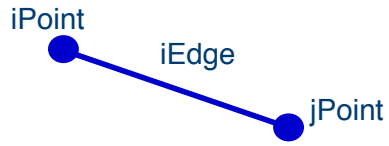
(d) + Optimized MPI (using MPI_Wait instead of MPI_Waitall), added nowait (OpenMP) and nontemporal stores (SIMD) clauses

Extracting the Parallelism from the ***Cores***

OpenMP threading – Example from SU2 CFD code

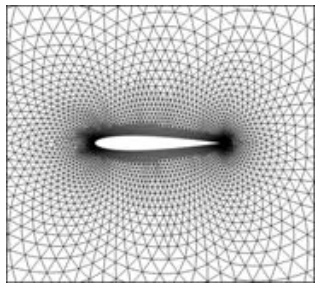
```
for (iEdge = 0; iEdge < geometry->GetnEdge(); iEdge++) {
{
   //Gather data from end-points of iEdge
    iPoint = geometry->edge[iEdge]->GetNode(0);
    jPoint = geometry->edge[iEdge]->GetNode(1);

    ProjVel_i = node[iPoint]->GetProjVel();
    ProjVel_j = node[jPoint]->GetProjVel();

    ...

    //Do some compute
    Mean_ProjVel = 0.5 * (ProjVel_i + ProjVel_j);
    Mean_SoundSpeed = 0.5 * (SoundSpeed_i + SoundSpeed_j) * Area;
    Lambda = fabs(Mean_ProjVel) + Mean_SoundSpeed;

    //Scatter the results to end-points of iEdge
    //Here we are updating Lambda at iPoint and jPoint
    //This is where there is write-contention as iPoint and/or jPoint are shared between different
        iEdges
    node[iPoint]->AddLambda(Lambda);
    node[jPoint]->AddLambda(Lambda);
}
```

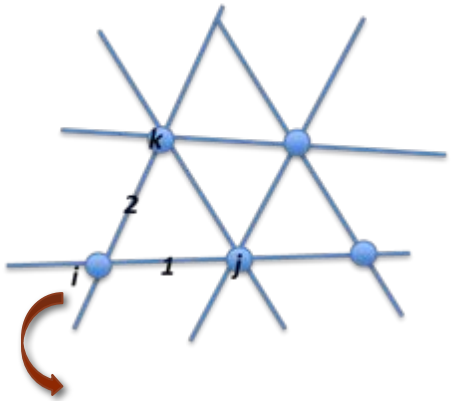
Sparse/indirect memory access



Compute (can be vectorized if large flops present)

Scatter: update results at end-points

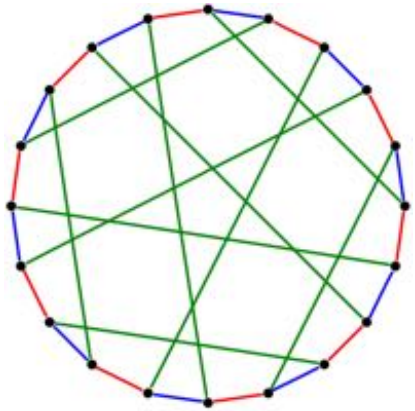




Fine-Grained ||ism: Data Vs. Domain Decomposition

- Data Decomposition

- Option 1: Basic partitioning with atomics – Just use "#pragma omp parallel for" and use "#pragma omp atomic" when writing out; Can use dynamic scheduling
- Option 2: Coloring or Level Scheduling – "Color" edges which can be operated on concurrently (no two adjacent edges have the same color). Do multiple passes. (No need for atomics)



Edge coloring, Source: Wikipedia

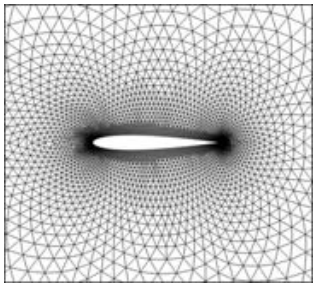
Fine-Grained ||ism: Data Vs. Domain Decomposition

Domain Decomposition – Using METIS Library

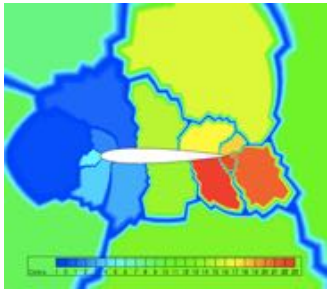
- Break graph at edges – Redundant compute on interfacial edges. *No atomics*. Owner thread of the boundary point updates the value.

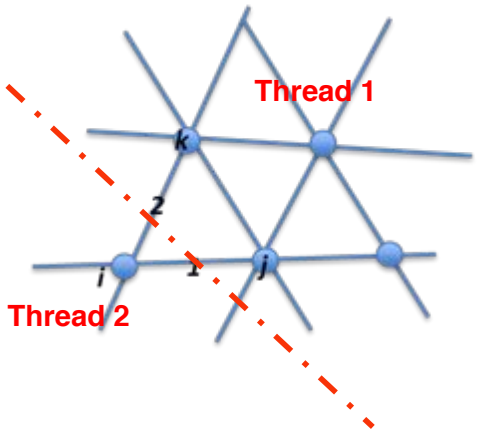
  [Cons: Cannot do dynamic thread scheduling]

MPI Domain



METIS decomposition



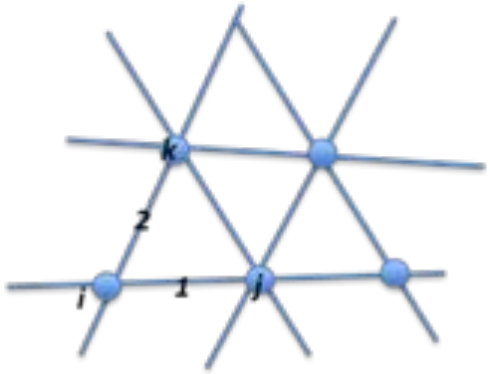


Efficiently Using the ***Memory***

AoS (array-of-structures) to SoA (structures-of-arrays) – Improves cache hits, enables contiguous memory access

```
class CVariable {
  double p;
  double u,v,w;
}
CVariable **node;
```

```
double *p;
double *u, *v, *w;
```




```
for (iEdge = 0; iEdge < Nedges; ++iEdge)
{
   iPoint = GetNode(0);

   Pres_i = node[iPoint]->p;
   Uvel_i = node[iPoint]->u;
}
```

```
for (iEdge = 0; iEdge < Nedges; ++iEdge)
{
   iPoint = GetNode(0);

   Pres_i = p[iPoint];
   Uvel_i = u[iPoint];
}
```

Efficiently using the ***Memory***

Reverse Cuthill-McKee (**RCM**) re-ordering – Improves cache hits

```
for (iEdge = 0; iEdge < geometry->GetnEdge(); iEdge++) {
{
   //Gather data from end-points of iEdge
    iPoint = geometry->edge[iEdge]->GetNode(0);
    jPoint = geometry->edge[iEdge]->GetNode(1);

    ProjVel_i = ProjVel_all[iPoint];
    ProjVel_j = ProjVel_all[jPoint];

    ...
```

Big jumps in jPoint

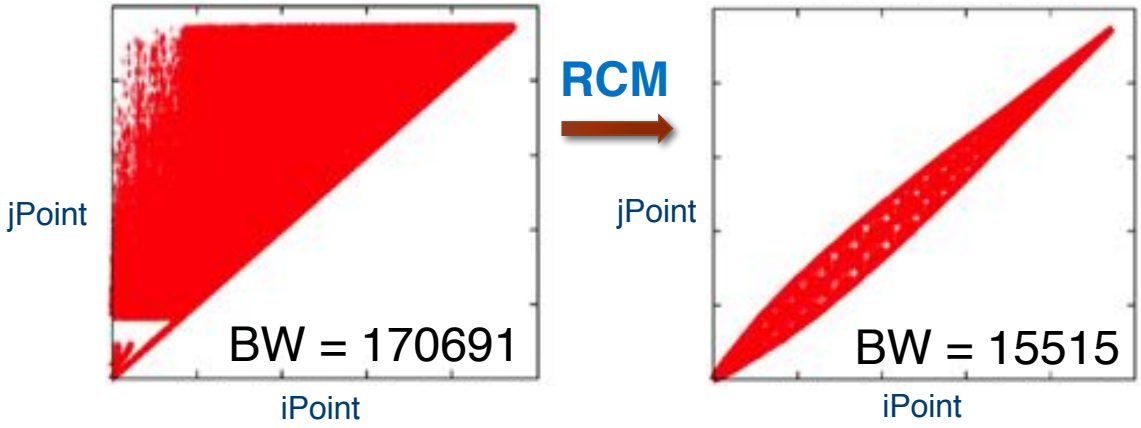
iEdge = 0, iPoint = 0, jPoint = 3731

iEdge = 1, iPoint = 0, jPoint = 83

iEdge = 2, iPoint = 0, jPoint = 1

iEdge = 3, iPoint = 0, jPoint = 84

ONERAM6 mesh

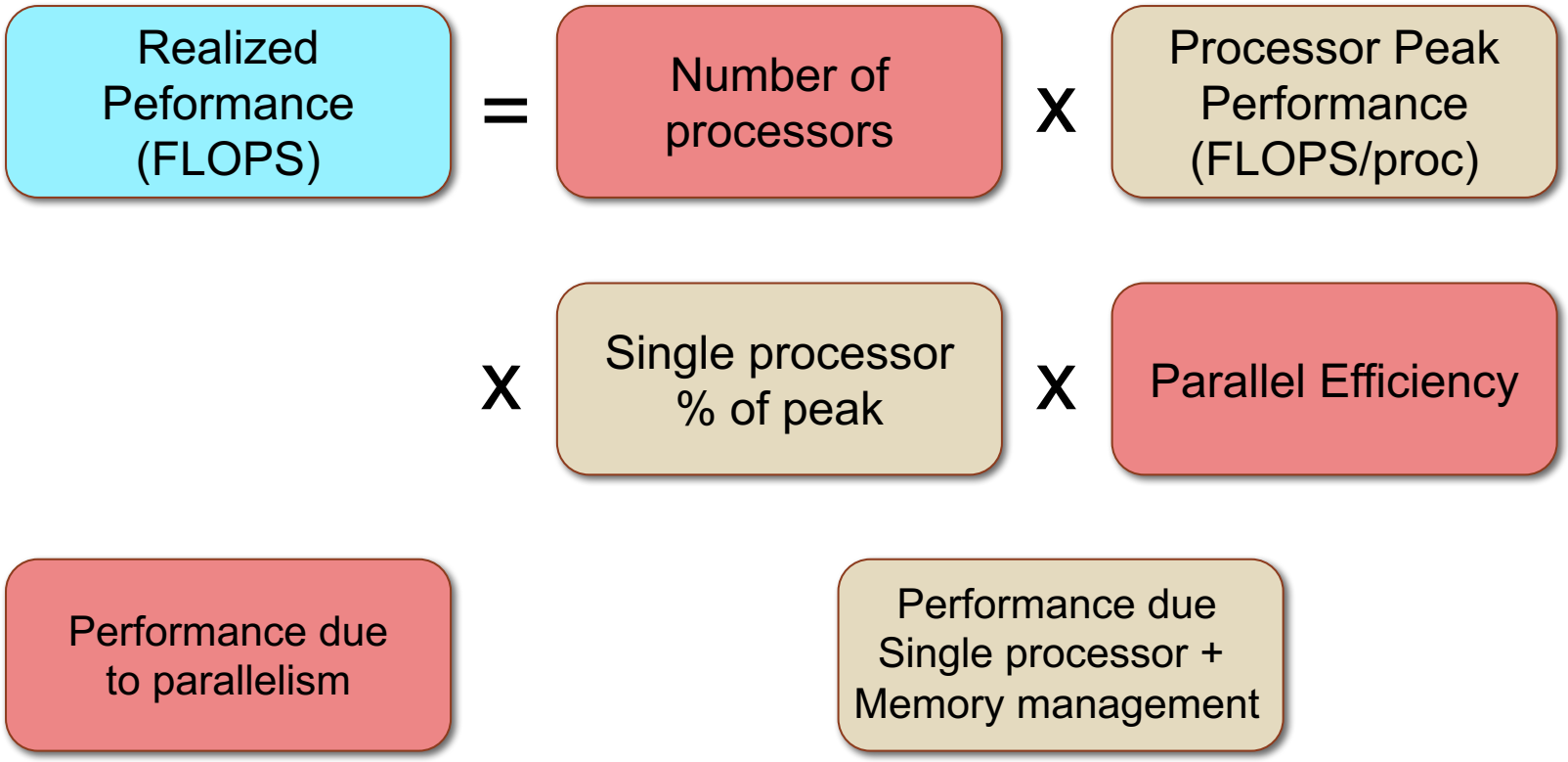


Computational Intensity of SU2 FV and DG-FEM Solvers

Computational / Arithmetic Intensity:

- SU2 FV solver (Roe+limiter+gradient reconstruction) ~ 1.25 flops / memory reference
- SU2 DG-FEM solver (p=4, hexahedra) ~ 14 flops / memory reference

Remember This Slide?

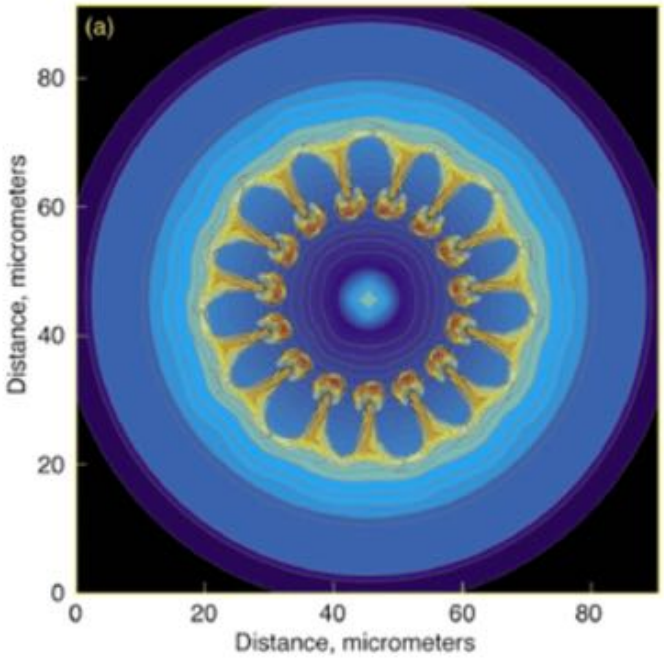


- Only way in modern hardware to get a significant % of peak is through algorithms with higher arithmetic intensity
- Difference can be 5-10% to 50-90% of peak

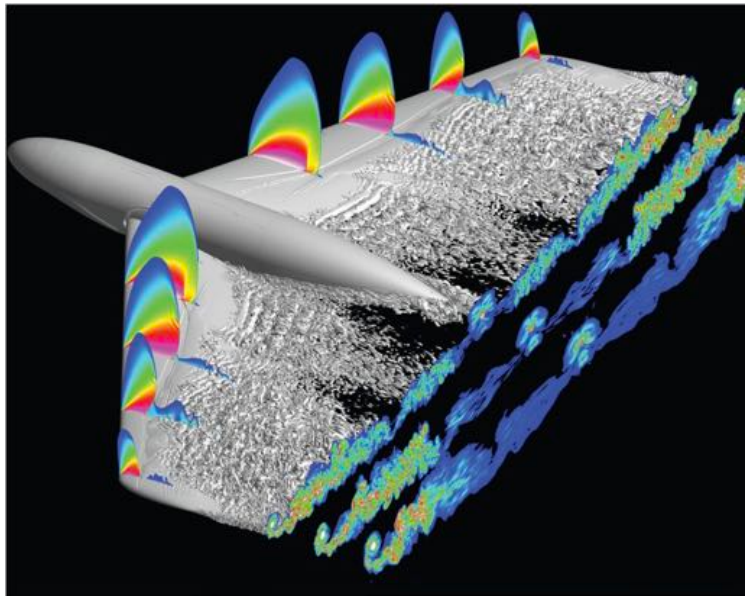
ALCF Aurora Early Science Program

Collaboration Stanford (Lele, Alonso) – Argonne

Benchmark Simulations of Shock-Variable Density Turbulence and Shock-Boundary Layer Interactions with Applications to Engineering Modeling



Hydrodynamic phenomena in ICF capsules, showing flow instabilities



Aircraft in transonic buffet conditions.
Source: Brunet & Deck, 2008.

Detailed flow physics must be resolved:
Ideally suited for higher-order schemes

Stanford University

Pros and Cons of the Methods

	Complex geometries	High-order accuracy and *hp*-adaptivity	Explicit semi-discrete form	Conservation laws	Elliptic problems
FDM	×	✓	✓	✓	✓
FVM	✓	×	✓	✓	(✓)
FEM	✓	✓	×	(✓)	✓
DG-FEM	✓	✓	✓	✓	(✓)

× : Not suited

✓ : Suited

(✓) : Suited, possibly with modifications, but not the most natural (or efficient) choice

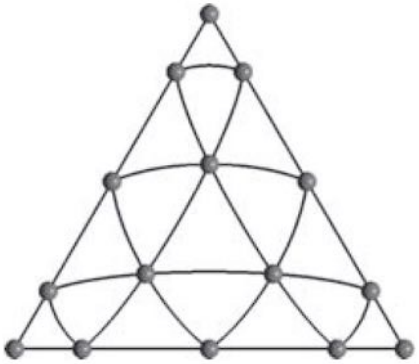
Source: Hesthaven and Warburton, 2008

2nd order: FVM is the best choice

High order: DG-FEM is the best choice

Nodal DG-FEM: Basic Principles (1)

$$\text{Element } k: \; U(x_i) = \sum_{j=1}^{N_p} U_j^k \varphi_j^k(x_i)$$



Hyperbolic system of PDE's
Weak formulation

$$\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i} = 0 \quad \Rightarrow$$

$$\iint_{V_k} \frac{\partial U}{\partial t} \varphi_m^k \, dV - \iint_{V_k} F_i \frac{\partial \varphi_m^k}{\partial x_i} \, dV + \oint_{\partial V_k} F_i n_i \varphi_m^k \, d\Omega_k = 0, \quad m = 1, \ldots, N_p$$

Nodal DG-FEM: Basic Principles (2)

Contribution from the contour integral

Solution at the interfaces is multiply defined and discontinuous

$$\oint_{\partial V_k} F_i n_i \varphi_m^k \, d\Omega_k \Rightarrow \oint_{\partial V_k} F_i\left(U_L, U_R\right) n_i \varphi_m^k \, d\Omega_k$$

Riemann problem: Any approximate Riemann solver can be used => stabilizes the discretization

1st order DG-FEM equals 1st order FVM!!!

Nodal DG-FEM: Basic Principles (3)

Nonlinear equations and/or curved elements

Integrals must be computed with high-order quadrature rules to avoid aliasing.
Expensive!!!

Diffusion problems, 2^{nd} derivatives

Discontinuous basis functions not suited => must be repaired
Even more expensive!!!

However: most operations are local to an element.
Extremely well-suited for modern computer hardware

Implementation in SU2

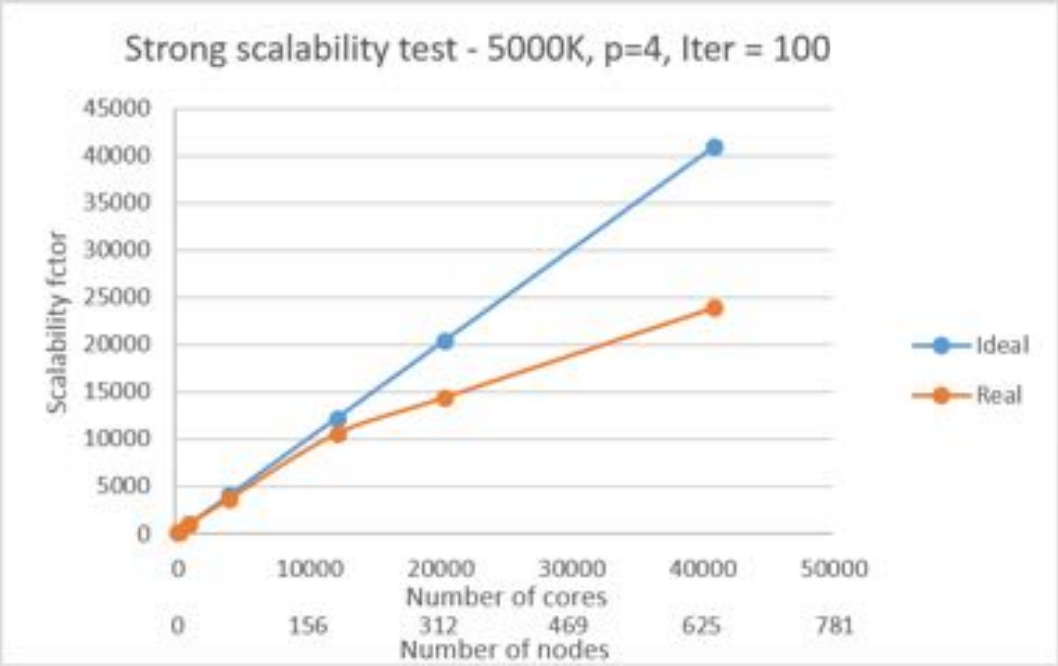
- Framework of SU2 is very flexible => high-level data structures can be used for any solver, so also DG-FEM
- Input parameter structure can be reused entirely
- FVM parallel I/O functionality could be reused (after some modifications)
- Still a lot of work was required for other low level functions
  - Partitioning of the grid (element wise)
  - Preprocessing is completely different from FVM
  - Standard elements and standard orientation of elements are introduced
  - Spatial discretization is completely new
- But can also reuse entire discrete adjoint formulation to immediately obtain sensitivities of time-accurate (and multi-physics) QoIs

Performance Optimization (1)

- Work still ongoing but nearly complete
- Target architectures: Intel Knights Landing (MIC architecture)
  Intel Xeon
- Hybrid MPI/OpenMP parallelization
- MPI
  - Domain decomposition (one complete halo layer of elements)
  - Overlap computation and communication (including ADER scheme)
  - Use of persistent communication
- OpenMP
  - Aim: Parallelization at for-loop level (sufficient for tests on Xeon)
  - Current data structures are designed for this approach
- Optimized BLAS/LAPACK/LIBXSMM functions must be used to get good performance for matrix multiplications
  - Large contiguous chunks of memory for data storage (not the case for the FVM solver)

Performance Optimization (2)

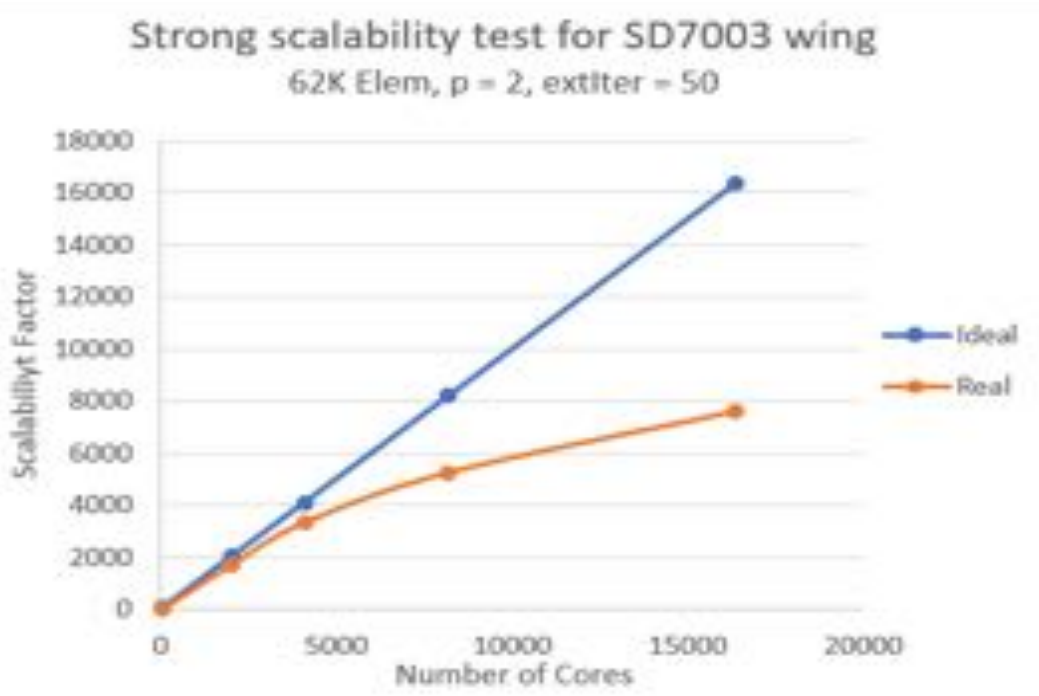
Motivation for hybrid parallelization approach
Flat MPI does not seem to work too well on KNL
Early results on Theta (Argonne)



Elem/Core	122	244	407	1.22K	4.88K	19.5K	39.1K
DOF/Core	4.27K	8.55K	14.2K	42.7K	0.17M	0.68M	1.37M
Efficiency(%)	58.4	70.6	86.6	91.0	99.2	100	N/A

Performance Optimization (2)

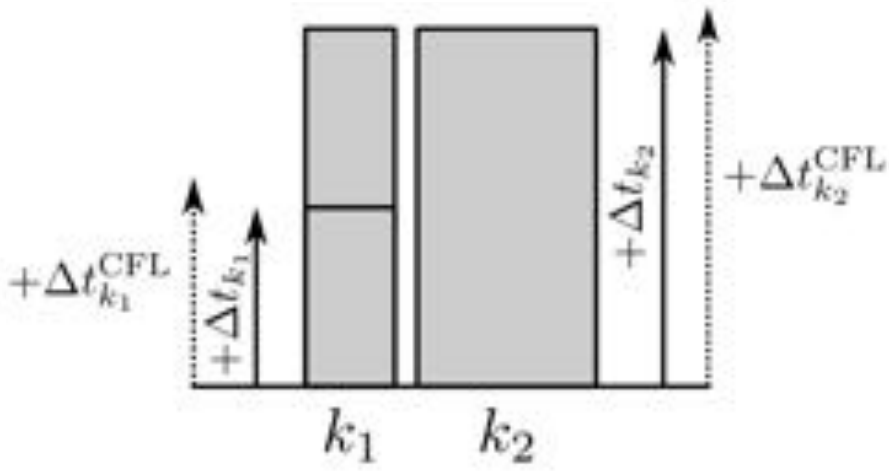
Motivation for hybrid parallelization approach
Flat MPI does not seem to work too well on KNL
Early results on Theta (Argonne)



Elem/Core	4	8	15	30	960
DOF/Core	31	62	125	250	8000
Efficiency(%)	46.3	63.9	80.6	83.1	N/A

ADER-DG Schemes Add Complexity…

- MPI Load balancing attempts to make sure every partition has "similar" amount of work
- But exact load balance cannot be achieved a priori
- Solution is to create a list of tasks
  - Volume integrals
  - Surface integrals
  - Multiple steps
  - Varying numbers of elements/faces
- And assign them to threads as work is completed.
- More advanced techniques are also possible



Performance Optimization (3)

Empirical estimation of workload across different element types for Navier-Stokes simulation

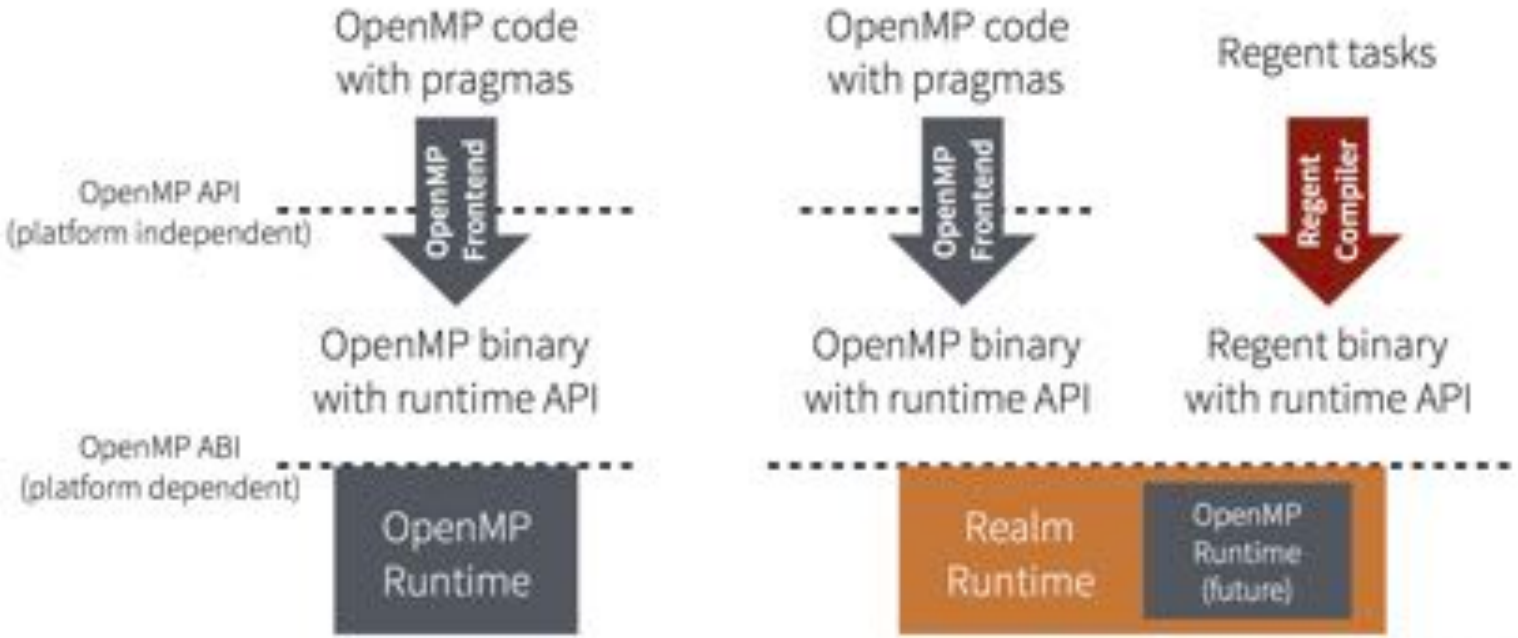
Element Type	Polynomial Order			
	p1	p2	p3	p4
Tetrahedron	1.000	3.003	6.036	10.071
Hexahedron	2.291	6.655	14.621	21.033
Prism	1.620	8.029	17.169	37.098
Pyramid	0.811	5.847	14.609	13.195

- Accurate estimate of workload for each element type is necessary for load balancing, especially in ADER implementation

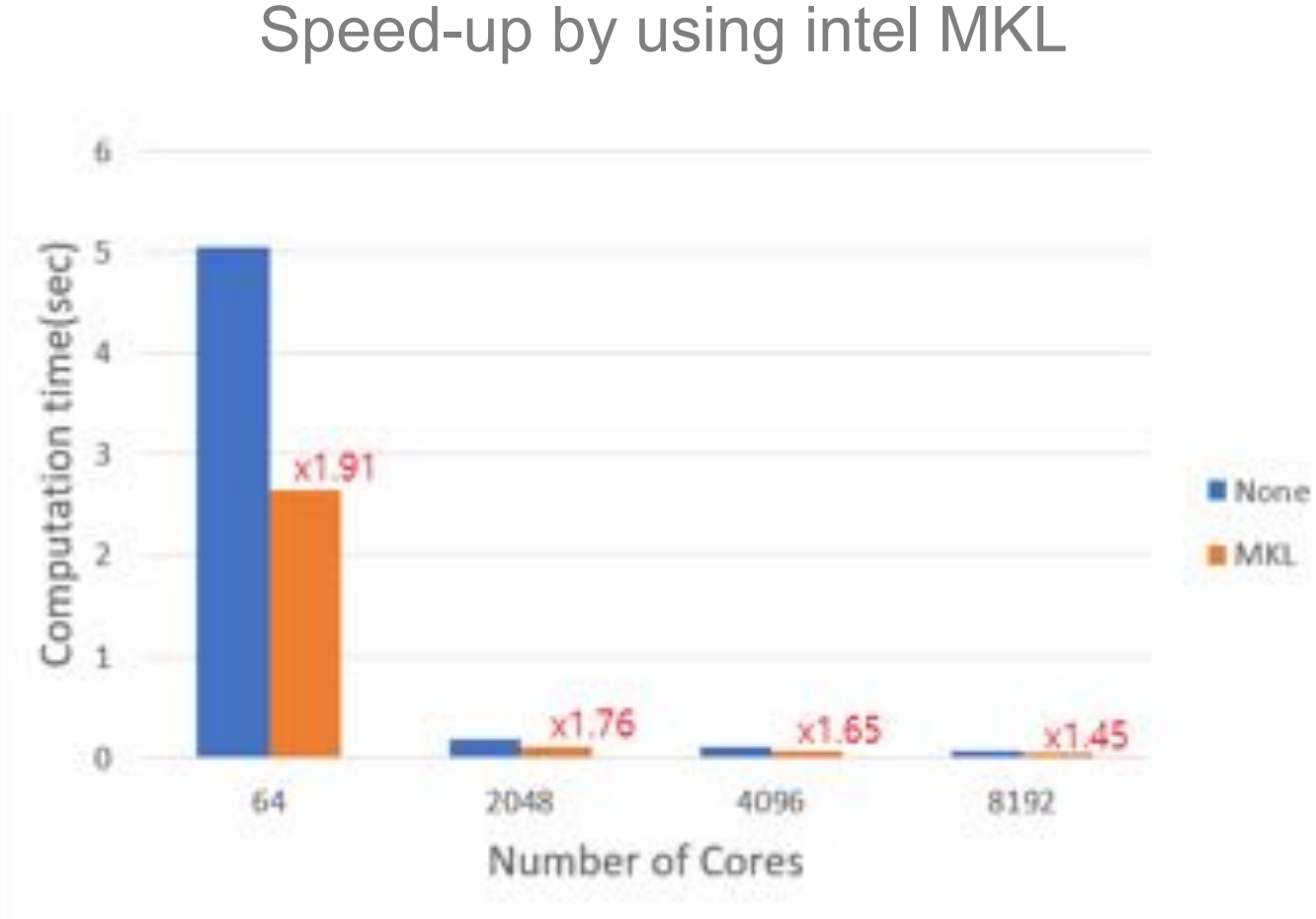
- Time accurate local time stepping (ADER) uses load balancing that is dependent on the speed of execution of volume and surface computations in each type (and order) of elements

OpenMP Tasks in Regent

- Legion now supports OpenMP tasks
  - Realm implements OpenMP ABI compatible to ICC/LLVM and GCC

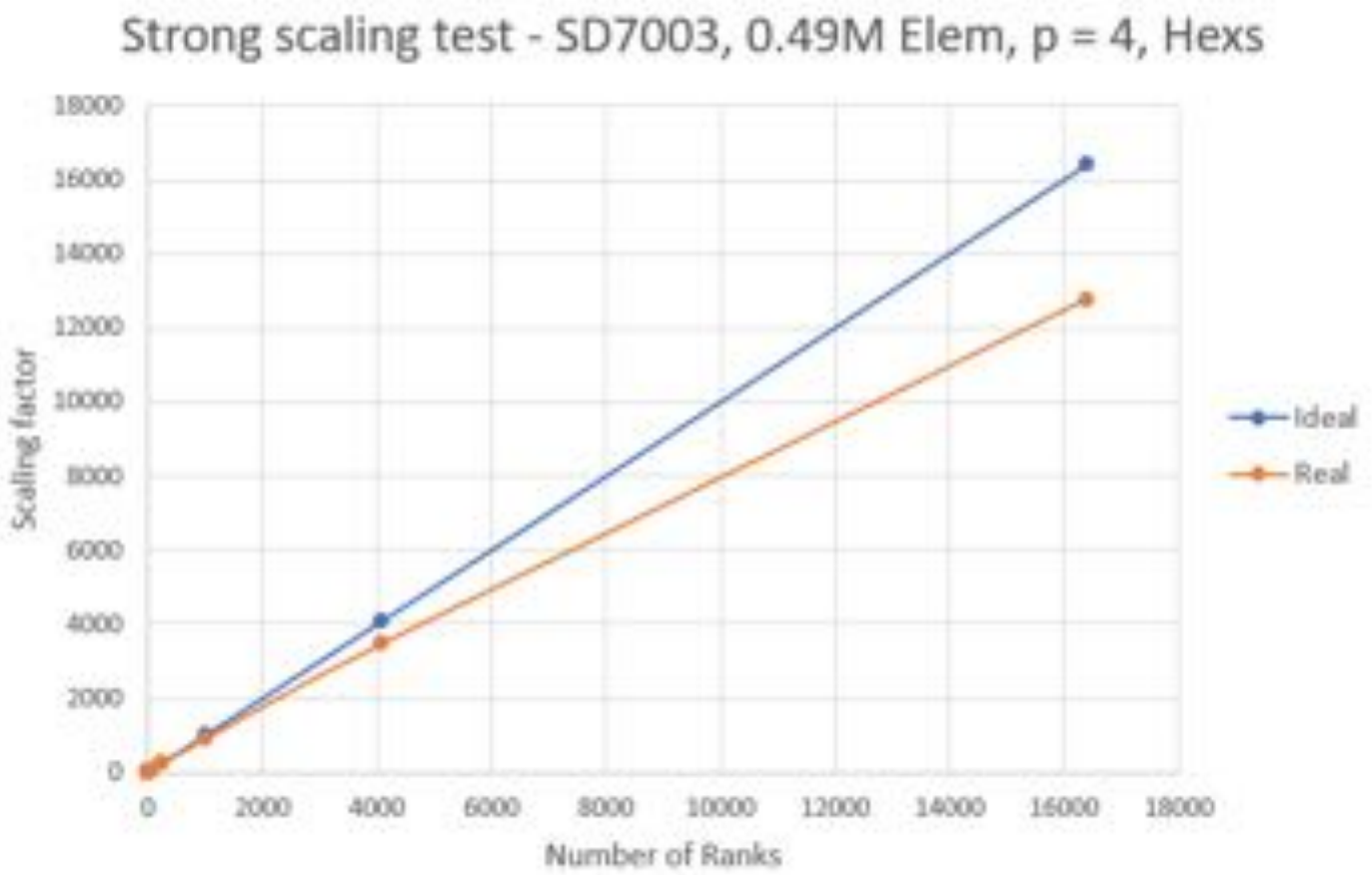


Performance Optimization (4)



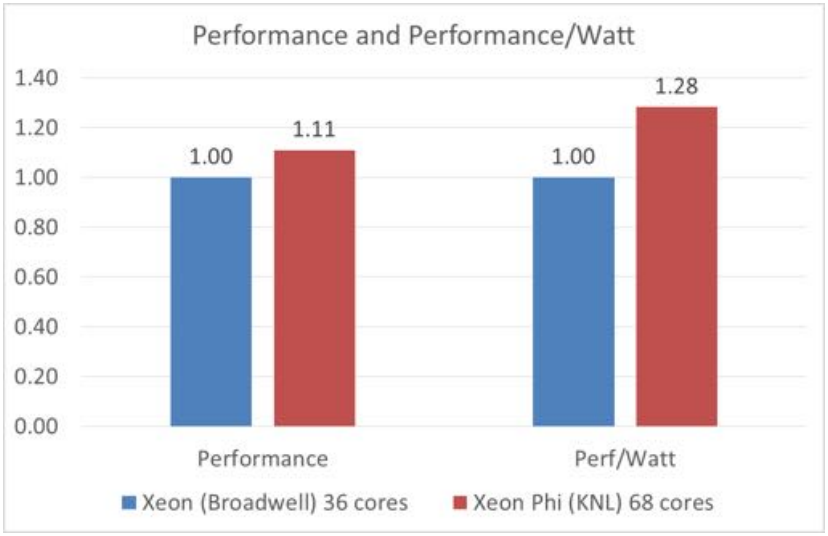
- In initial testing, use of libXSMM provides additional performance improvements
- Tests ongoing on Theta

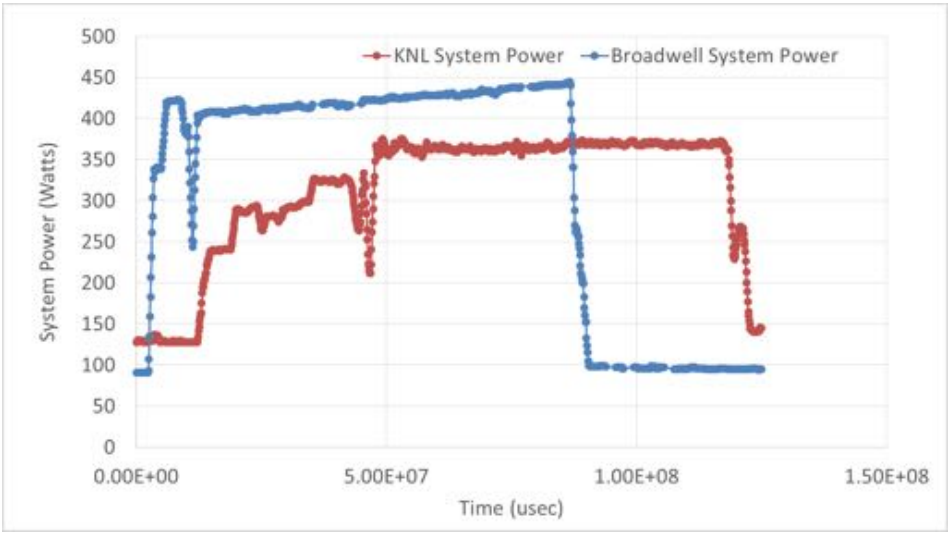
Strong Scaling Test – 3D SD7003 Airfoil



Elem/Rank	0.25M	0.12M	30.7K	7.68K	1.92K	480	120	30
DOF/Rank	30.7M	15.4M	3.84M	0.96M	0.24M	60K	15K	3.75K
Efficiency(%)	N/A	99.98	97.15	90.89	93.39	90.49	85.07	77.86

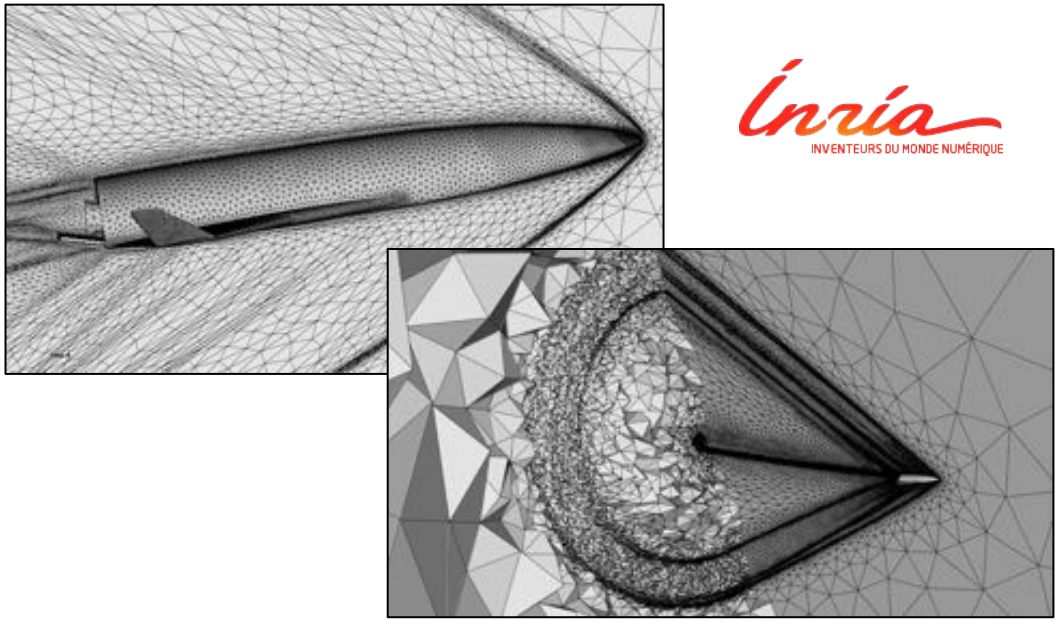
DG-FEM Solver Performance on KNL

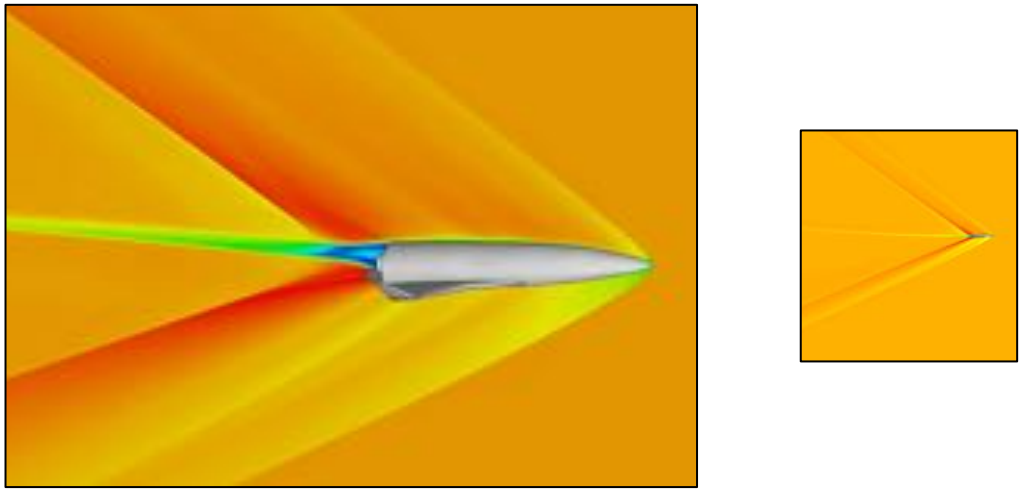




What About Managing the Numerical Error?

- Functional-based error estimation / adaptation theory has been around for a while
- In a few years this problem will be solved and commercial solvers will incorporate these capabilities
- We have teamed up with INRIA and are using their mesh adaptation library
- Will we also be able to manage the uncertainties simultaneously? This is the grand challenge





Conclusions & Ongoing Work

- Optimized CFD solvers for new computing platforms require significant effort and expertise
- Scientists and engineers should focus on doing engineering and science: turbulence modeling and others
- Performance / scalability optimization expertise is quickly disappearing in academe
- Multi-disciplinary teams are needed…and hard to come by
- Must amortize the effort over a larger base of users
- Community codes may be an answer: high-performance out of the box

UNIVERSITY OF TWENTE.

Stanford | ENGINEERING
Aeronautics & Astronautics

Imperial College
London

Stanford University


STANFORD
AERONAUTICS &
ASTRONAUTICS
SU2
The Open-Source CFD Code