Some Long Term Experiences in HPC Programming for Computational Fluid Dynamics Problems

Dimitri Mavriplis

University of Wyoming

Overview

- A bit of history
- Details of a Parallel Unstructured Mesh Flow Solver
- Programming paradigms
  - Some old and new performance results
- Why we are excited about Higher Order Methods
- Conclusions

History

- Vector Machines
  - Cyber 203 and 205 at NASA Langley (~1985)
    - Painful vectorization procedure
  - Cray 2, Cray YMP, Cray C-90, Convex C1-C2
    - Vastly better vectorization compilers
    - Good coarse grain parallelism support
- Rise of cache-based parallel architecture (aka. killer micros)
  - Early successes: 1.5 Gflops on 512 cpu of Intel Touchstone Delta Machine (with J. Saltz at ICASE) in 1992
    - See Proc. SC92: Was still slower than 8cpu Cray-C90

History

- Early difficulties of massively parallel machines
  - Cache based optimizations fundamentally at odds with vector optimizations
    - Local versus global
      - Tridiagonal solver: Inner loop must vectorize over lines
  - Unclear programming paradigms and Tools
    - SIMD, MIMD
    - HPF, Vienna Fortran, CMFortran
    - PVM, MPI, Shmem etc …?

Personal View

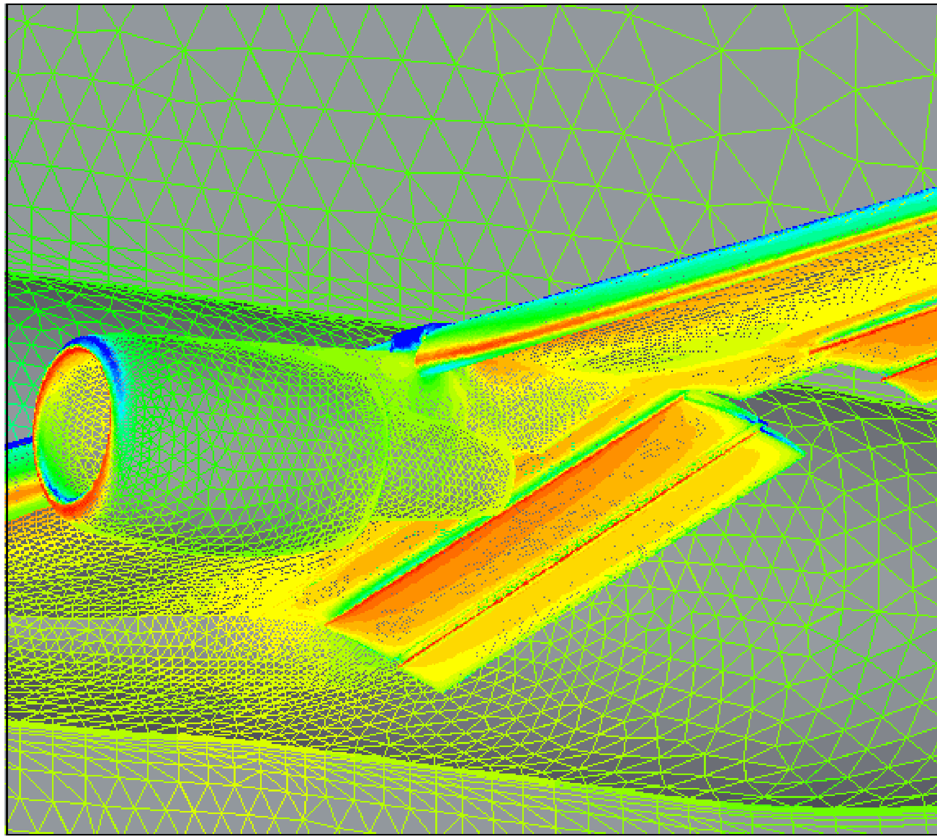
- Biggest single enabler of massively parallel computer applications has been:
  - Emergence of MPI (and OpenMP) as standards
  - Realization that low level programming will be required for good performance
    - Failure of HPF type approaches
      - Were inspired by success of auto-vectorization (Cray/Convex)
      - Parallel turns out to be more complex than vector
- Difficult issues remain but
  - Probability of automated high level software tools which do not compromise performance seems remote
    - e.g. Dynamic load balancing for mesh adaptation

Looking forward

- Can this approach (MPI/OMP) be extended up to 1M cores ?
- Challenges of strong solvers (implicit or multigrid) on many cores
- Should we embrace hybrid models ?
  - MPI-OpenMP ?
- What if long vectors make a come back ?
  - Stalling clock speeds….

NSU3D: Unstructured Navier-Stokes Solver

- High fidelity viscous analysis
  - Resolves thin boundary layer to wall
    - O(10^{-6}) normal spacing
    - Stiff discrete equations to solve
    - Suite of turbulence models available
  - High accuracy objective: 1 drag count
- Unstructured mixed element grids for complex geometries
  - VGRID: NASA Langley
  - ICEM CFD, Others
- Production use in commercial, general aviation industry
- Extension to Design Optimization and Unsteady Simulations



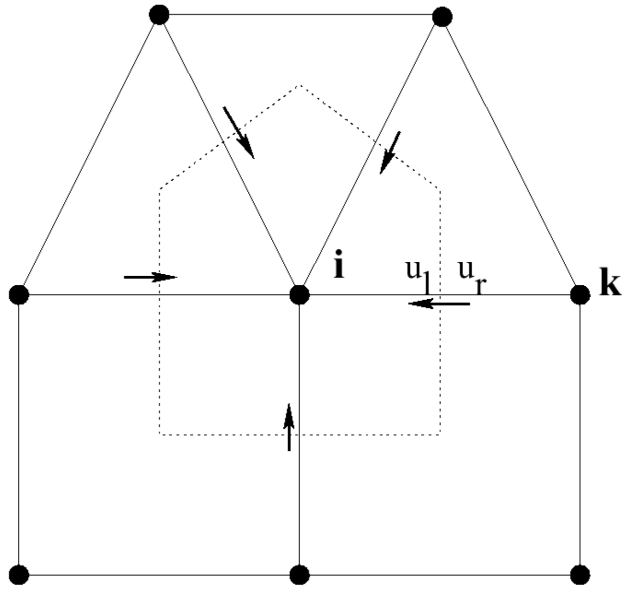
University of Wyoming

NSU3D Solver

- Governing Equations: Reynolds Averaged Navier-Stokes Equations
  - Conservation of Mass, Momentum and Energy
  - Single Equation turbulence model (Spalart-Allmaras)
  - 2 equation k-omega model
    - Convection-Diffusion – Production
- Vertex-Based Discretization
  - 2nd order upwind finite-volume scheme
  - 6 /7variables per grid point
  - Flow equations fully coupled (5x5)
  - Turbulence equation uncoupled

Spatial Discretization

- Mixed Element Meshes
  - Tetrahedra, Prisms, Pyramids, Hexahedra
- Control Volume Based on Median Duals
  - Fluxes based on edges
    - Upwind or artifical dissipation



* $\mathbf{F}_{\mathrm{ik}} = f(\mathbf{u}_{\mathrm{left}}, \mathbf{u}_{\mathrm{right}})$

* $\mathbf{u}_{\mathrm{left}} = \mathbf{u}_{\mathrm{i}}, \mathbf{u}_{\mathrm{right}} = \mathbf{u}_{\mathrm{k}}$: *1st order accurate*

* $\mathbf{u}_{\mathrm{left}} = \mathbf{u}_{\mathrm{i}} + \frac{1}{2}\nabla\mathbf{u}_{\mathrm{i}}.\mathbf{r}_{\mathrm{ik}}$

* $\mathbf{u}_{\mathrm{right}} = \mathbf{u}_{\mathrm{k}} + \frac{1}{2}\nabla\mathbf{u}_{\mathrm{k}}.\mathbf{r}_{\mathrm{ki}}$: *2nd order accurate*

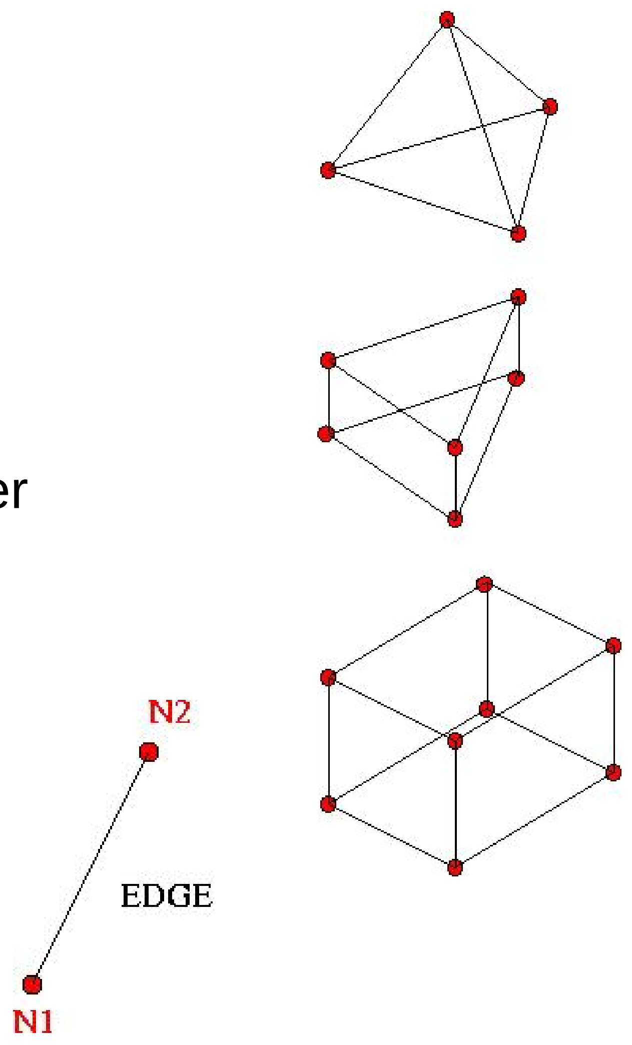
* ∇u_i *evaluated as contour integral around CV*

  - Single edge-based data-structure represents all element types

Mixed-Element Discretizations

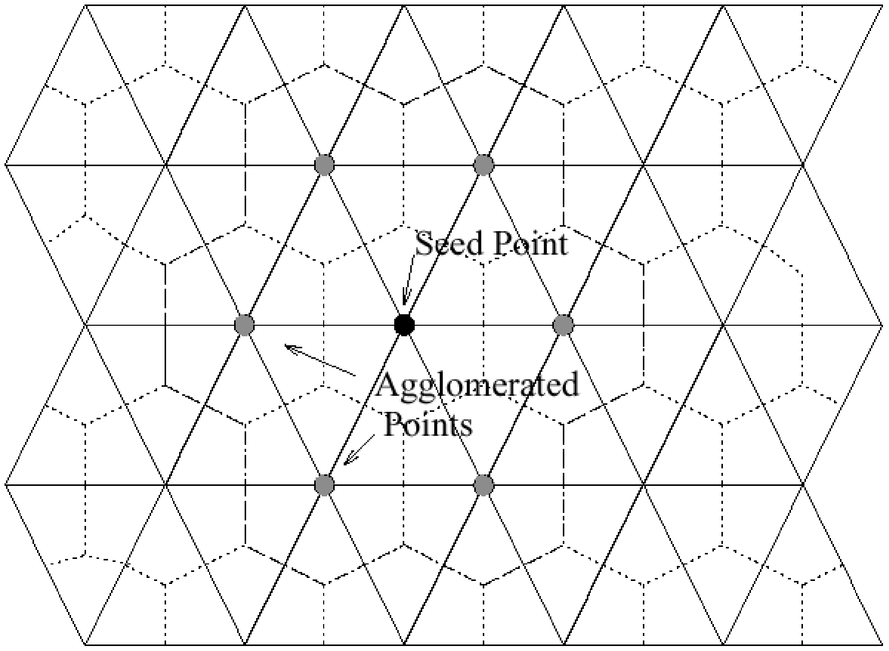
- Edge-based data structure
  - Building block for all element types
  - Reduces memory requirements
  - Minimizes indirect addressing / gather-scatter
  - Graph of grid = Discretization stencil
    - Implications for solvers, Partitioners

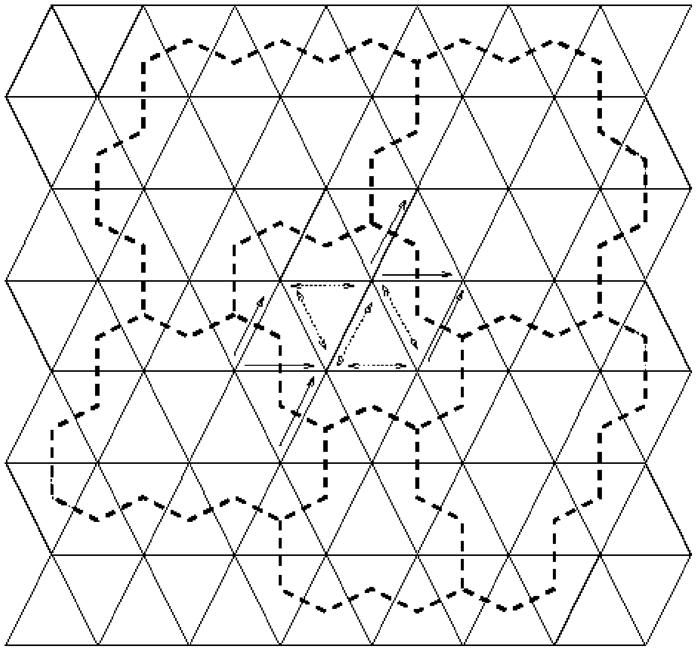
- Has had major impact on HPC performance



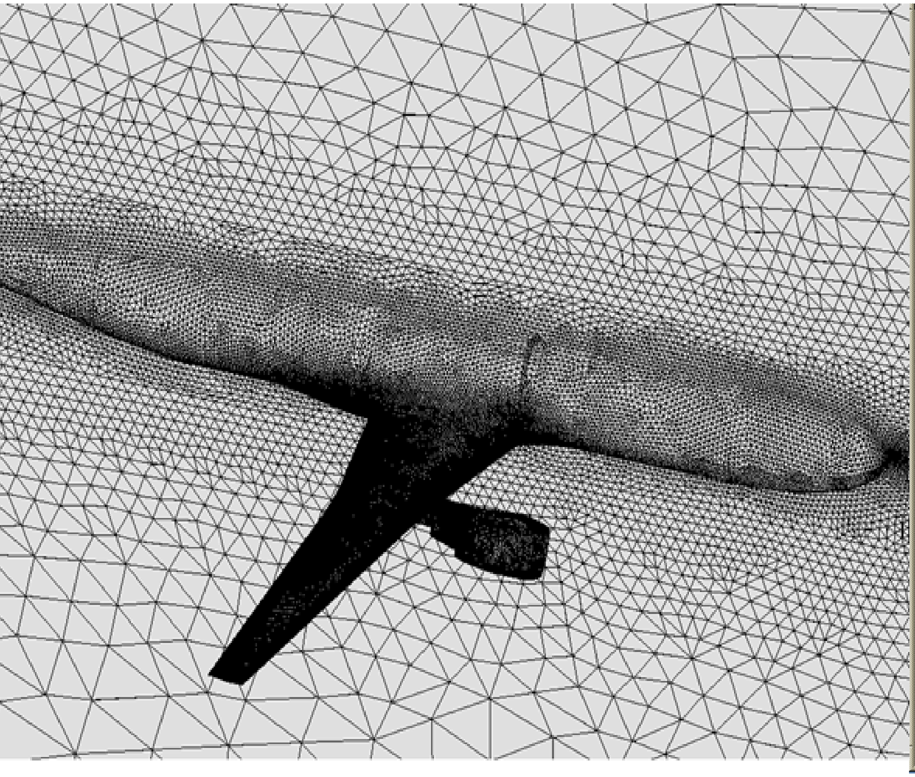
Agglomeration Multigrid

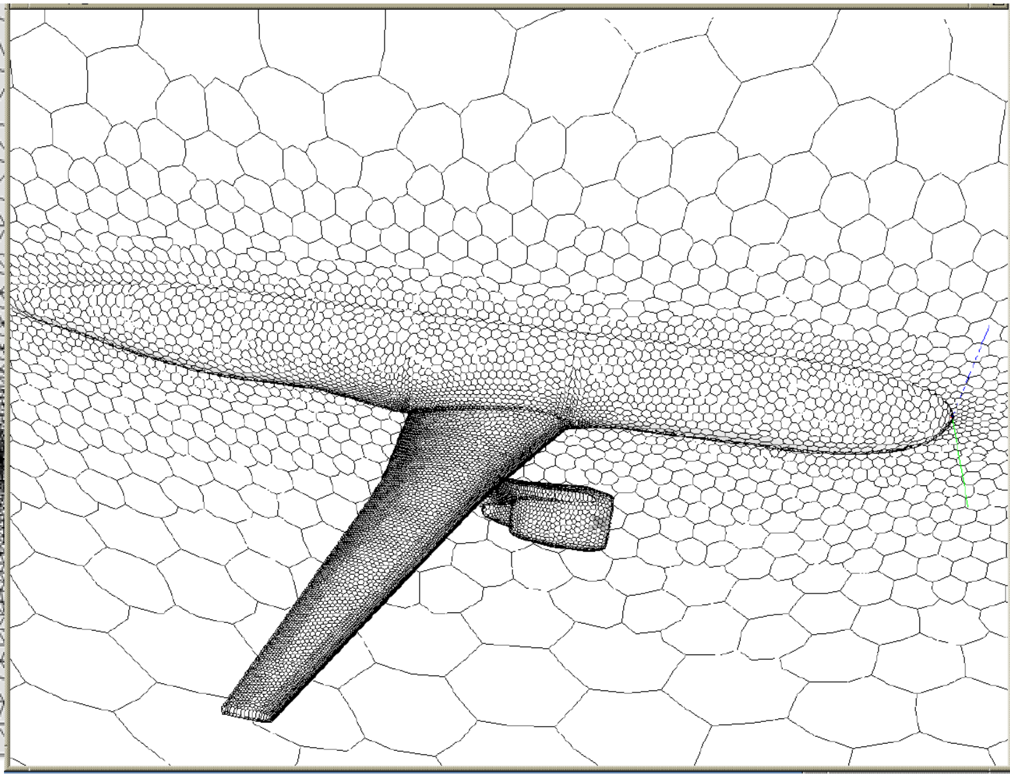
- Agglomeration Multigrid solvers for unstructured meshes
  - Coarse level meshes constructed by agglomerating fine grid cells/equations
    - Automated, invisible to user
  - Multigrid algorithm cycles back and forth between coarse and fine grid levels
  - Produces order of magnitude improvement in convergence
  - Maintains good scalability of explicit scheme





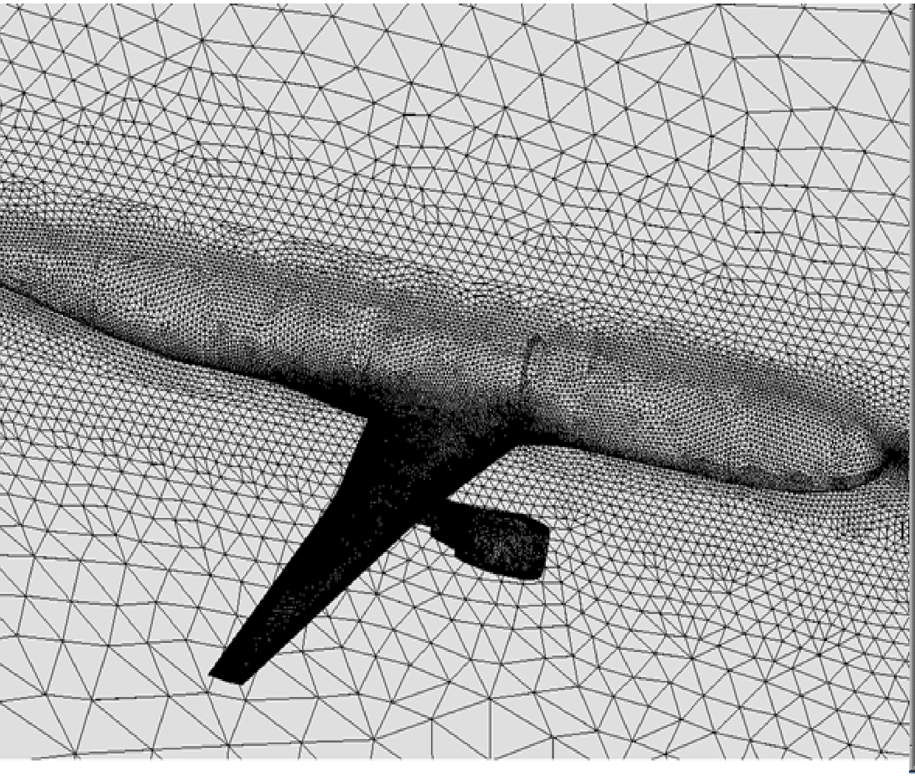
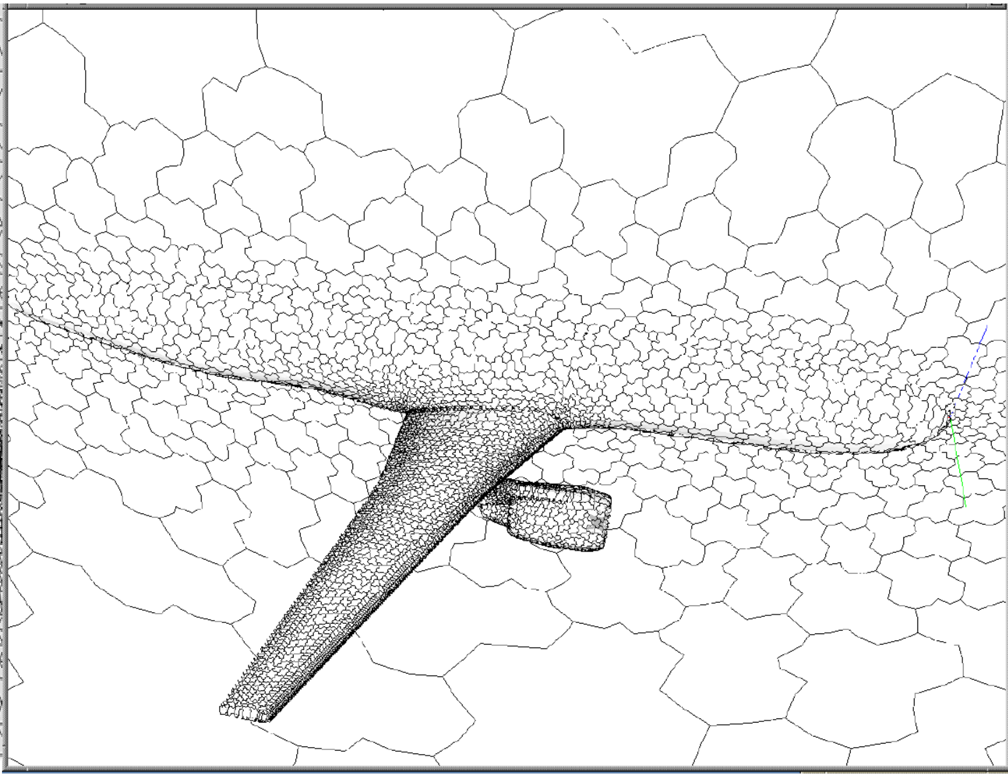
Agglomeration Multigrid





- Automated Graph-Based Coarsening Algorithm
- Coarse Levels are Graphs
- Coarse Level Operator by Galerkin Projection
- Grid independent convergence rates (order of magnitude improvement)

Agglomeration Multigrid

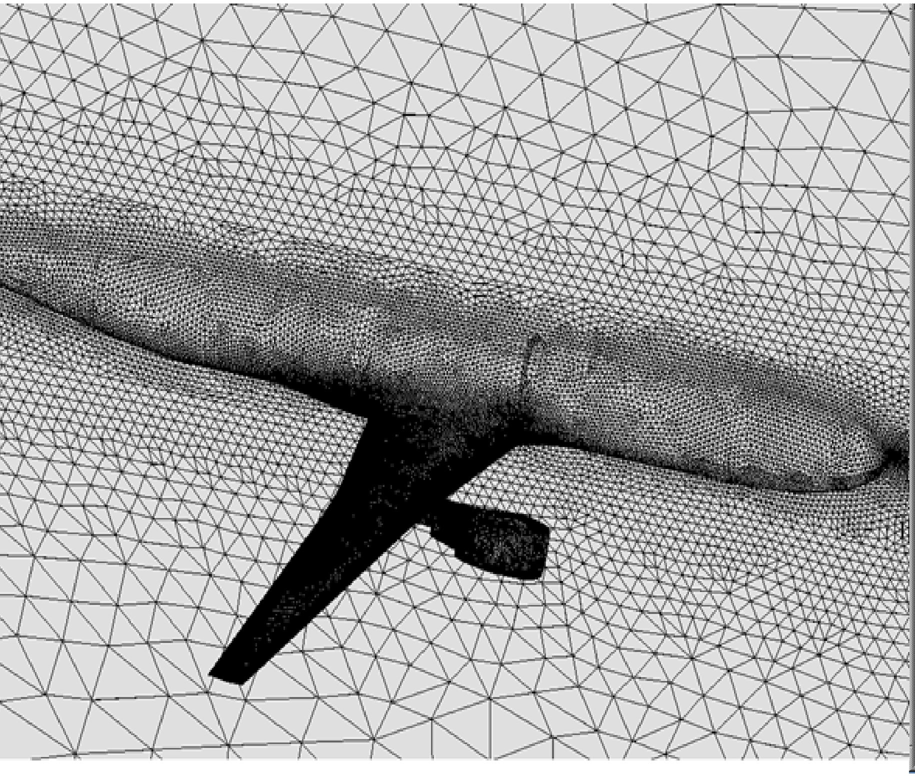
•Automated Graph-Based Coarsening Algorithm

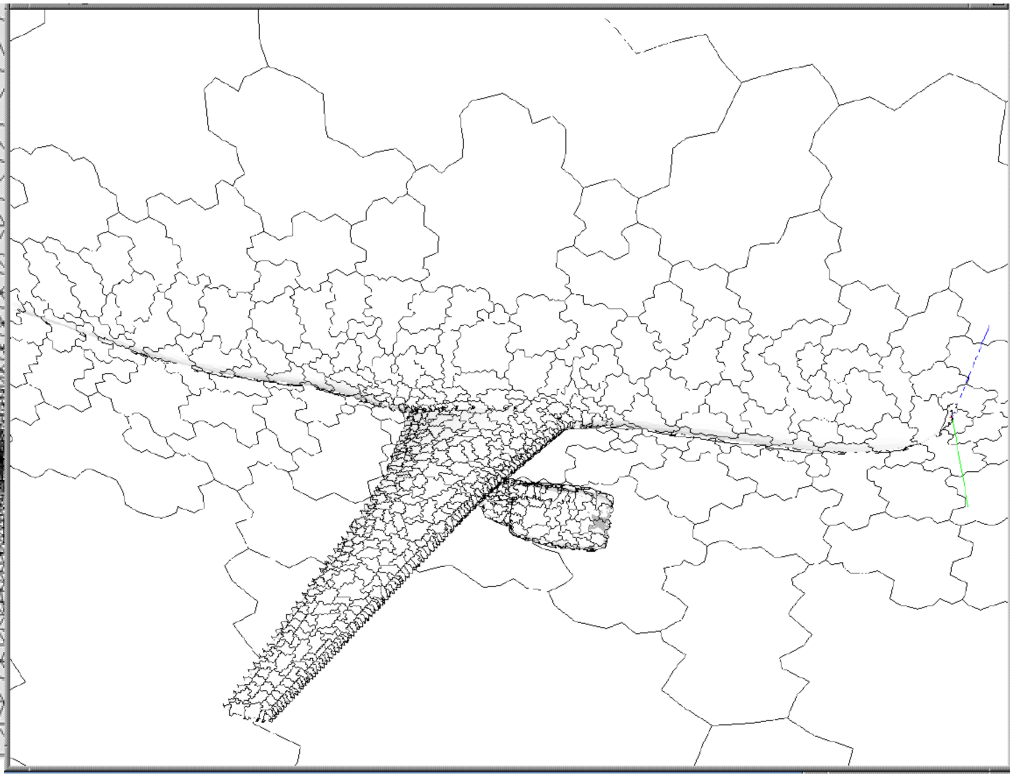
•Coarse Levels are Graphs

•Coarse Level Operator by Galerkin Projection

•Grid independent convergence rates (order of magnitude improvement)

Agglomeration Multigrid





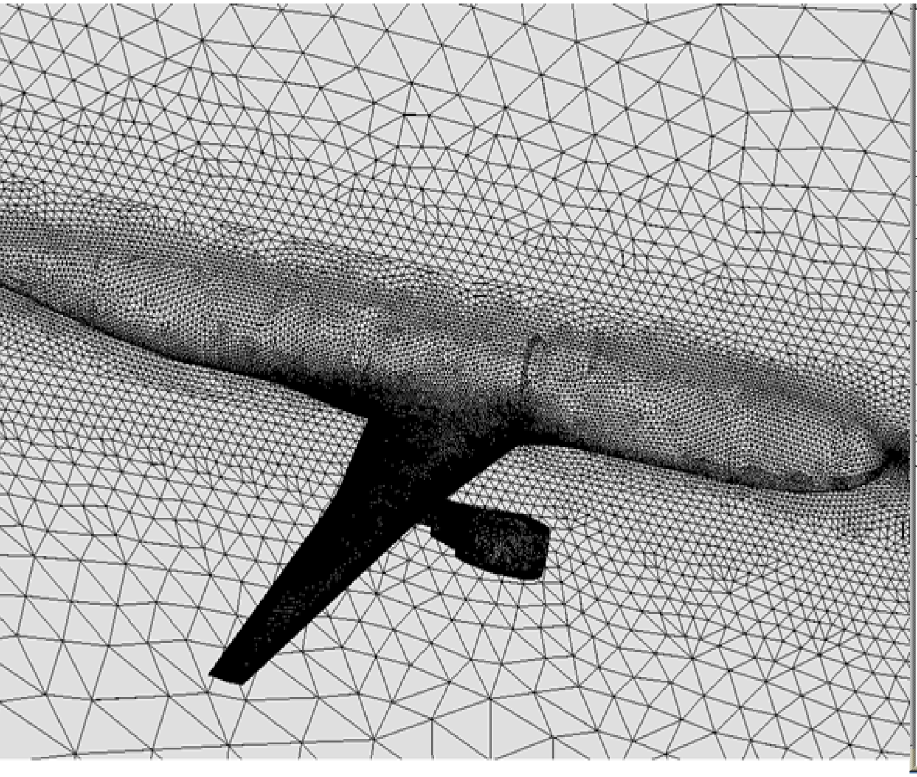
•Automated Graph-Based Coarsening Algorithm

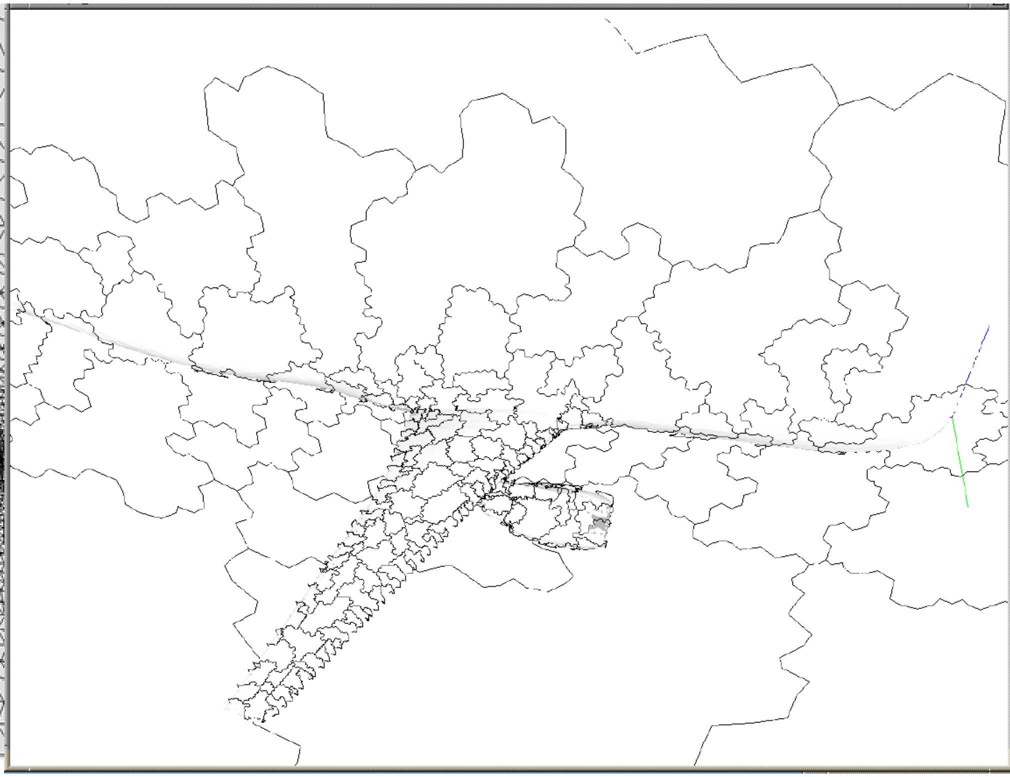
•Coarse Levels are Graphs

•Coarse Level Operator by Galerkin Projection

•Grid independent convergence rates (order of magnitude improvement)

Agglomeration Multigrid





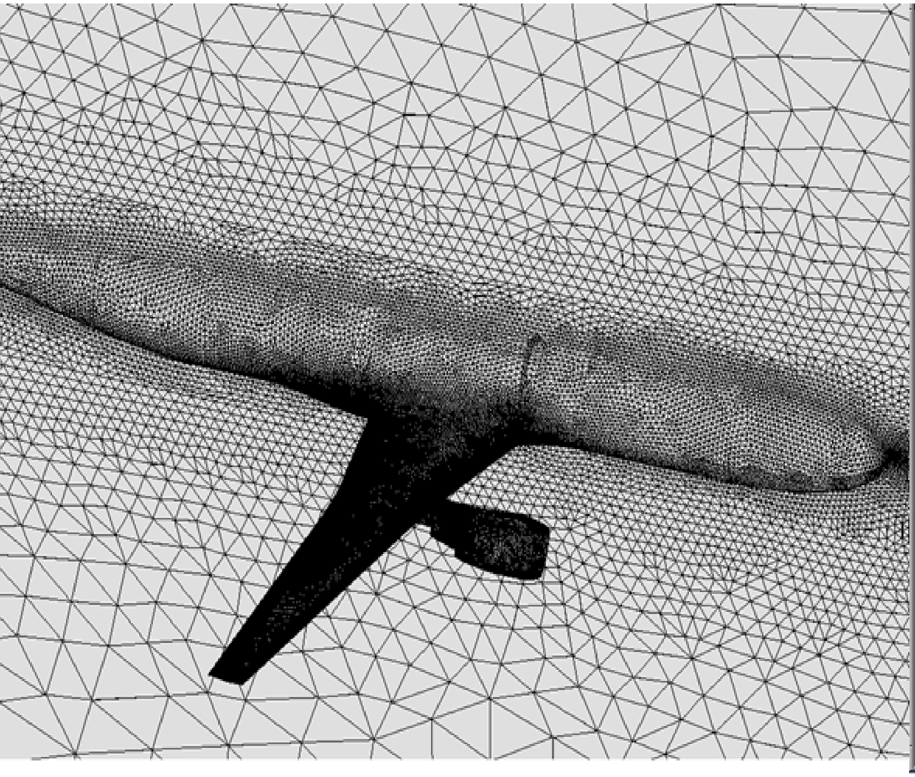
•Automated Graph-Based Coarsening Algorithm

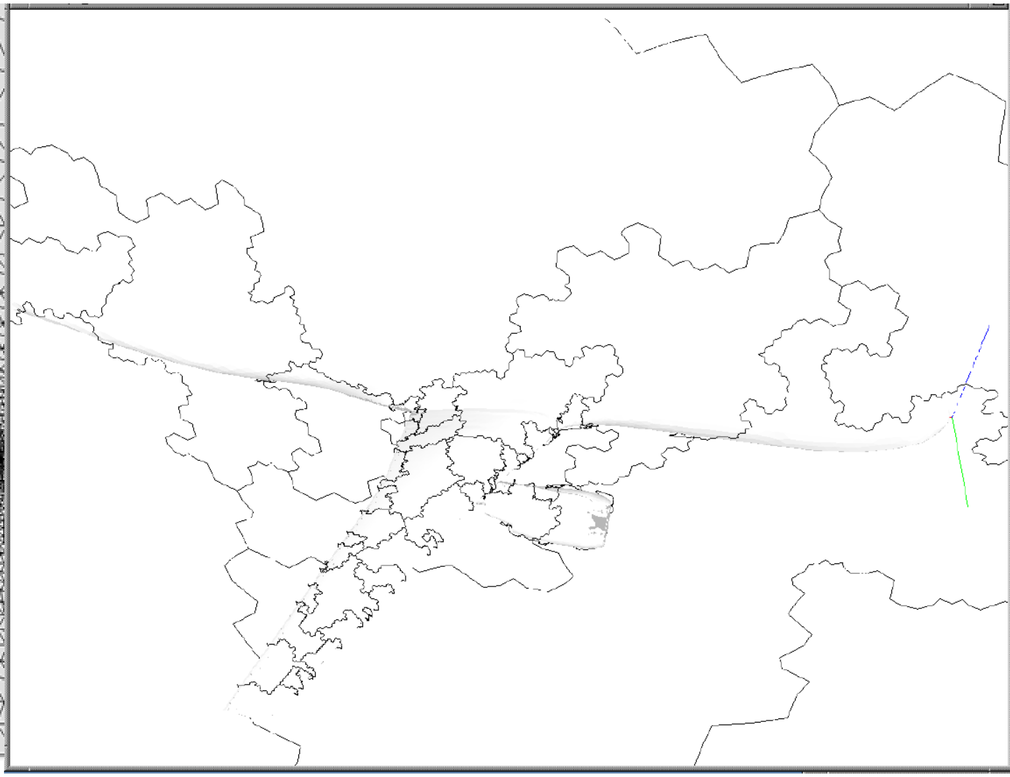
•Coarse Levels are Graphs

•Coarse Level Operator by Galerkin Projection

•Grid independent convergence rates (order of magnitude improvement)

Agglomeration Multigrid





•Automated Graph-Based Coarsening Algorithm

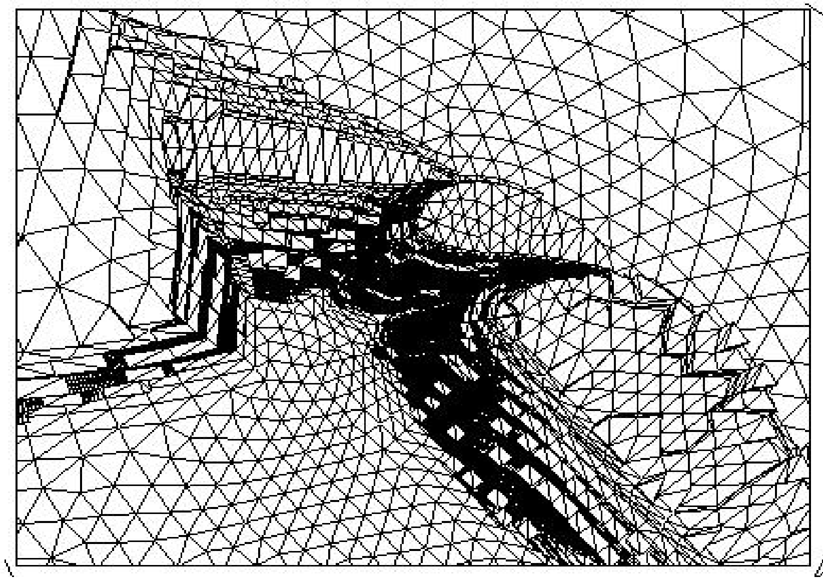
•Coarse Levels are Graphs

•Coarse Level Operator by Galerkin Projection

•Grid independent convergence rates (order of magnitude improvement)

Anisotropy Induced Stiffness

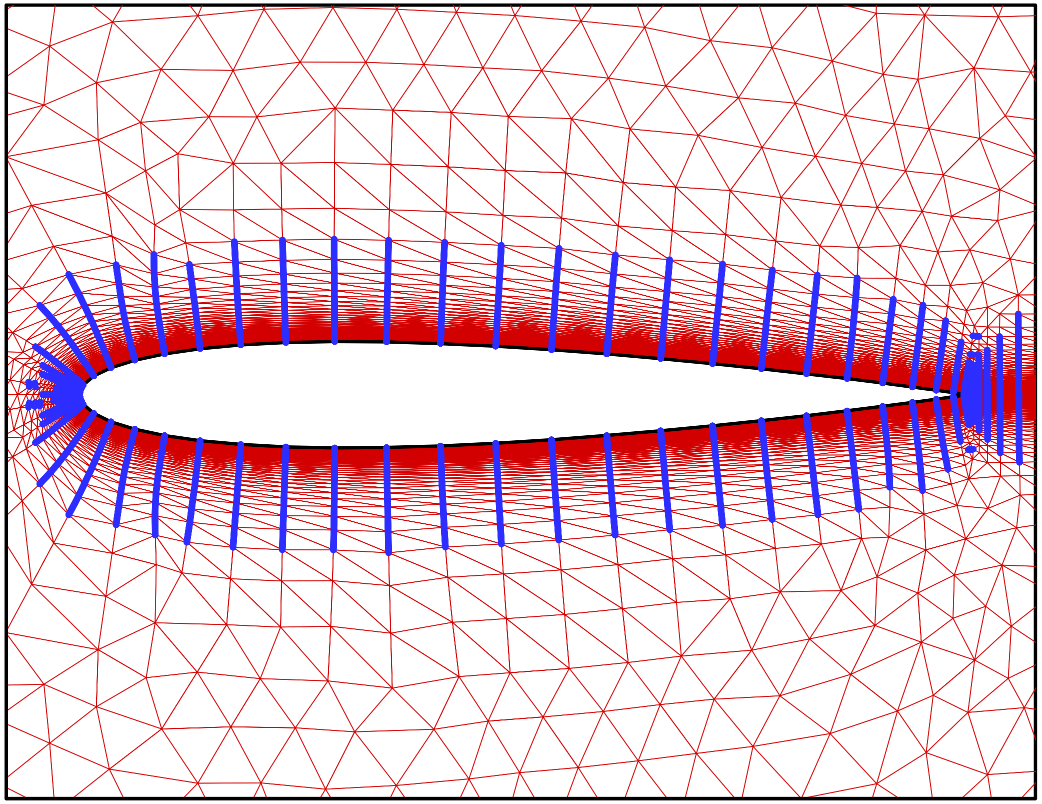
- Convergence rates for RANS (viscous) problems much slower then inviscid flows
  - Mainly due to grid stretching
  - Thin boundary and wake regions
  - Mixed element (prism-tet) grids



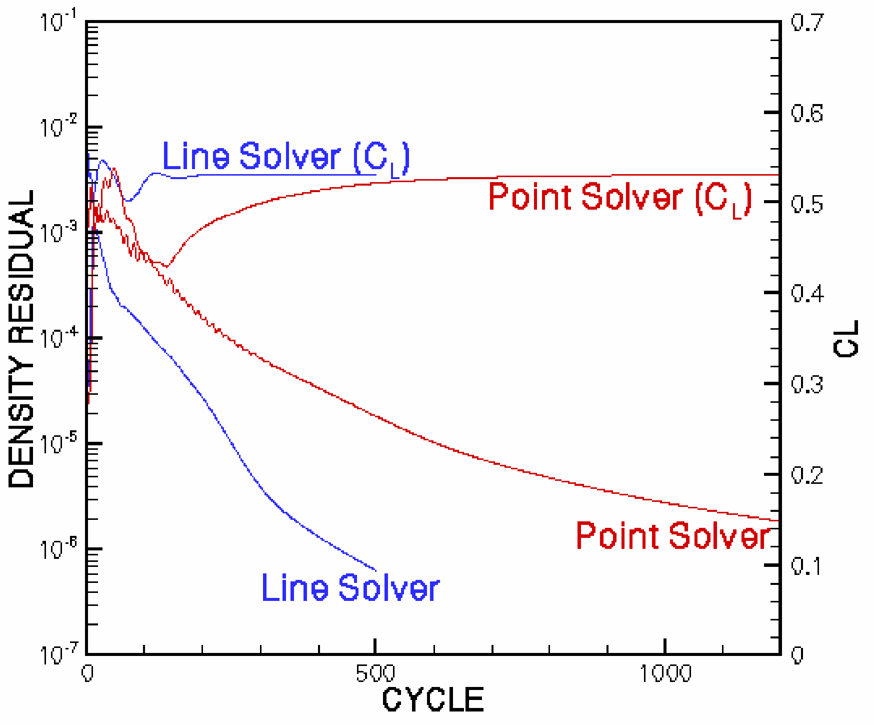
- Use directional solver to relieve stiffness
  - Line solver in anisotropic regions

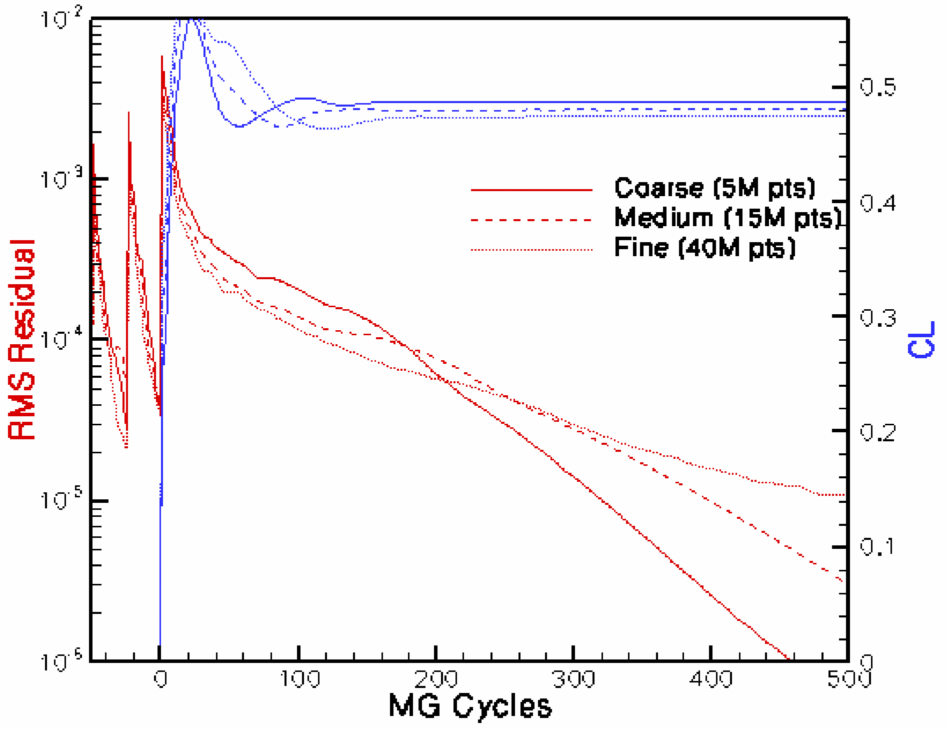
Method of Solution

- Line-implicit solver



Line Solver Multigrid Convergence

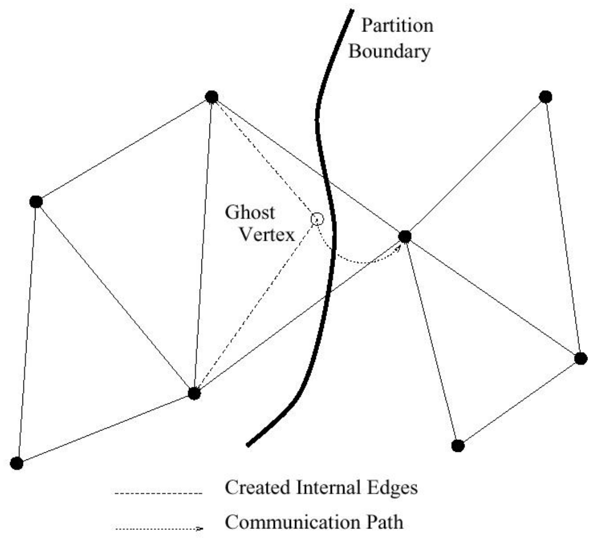




Line solver convergence insensitive to grid stretching

Multigrid convergence insensitive to grid resolution

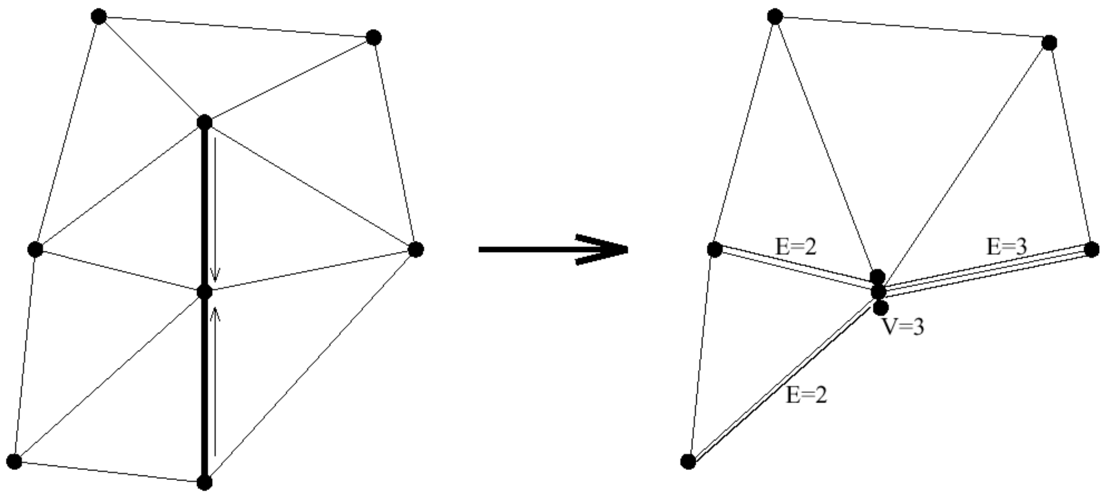
Parallelization through Domain Decomposition



- Intersected edges resolved by ghost vertices
- Generates communication between original and ghost vertex
  - Handled using MPI and/or OpenMP (Hybrid implementation)
  - Local reordering within partition for cache-locality

Partitioning

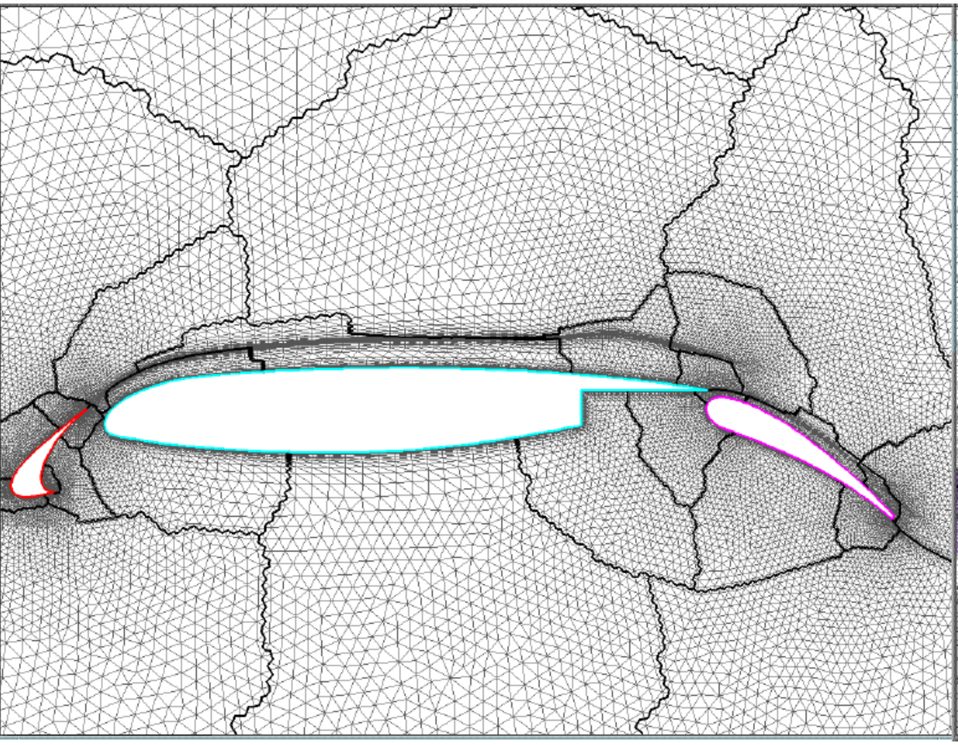
- (Block) Tridiagonal Lines solver inherently sequential
- Contract graph along implicit lines
- Weight edges and vertices

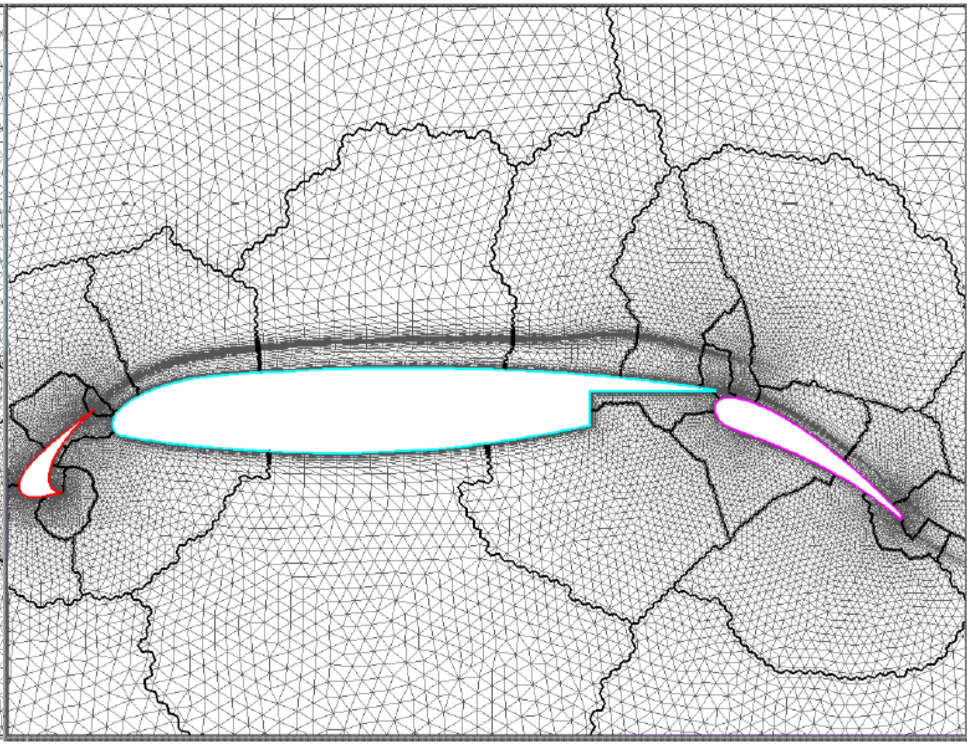


- Partition contracted graph
- Decontract graph
  - Guaranteed lines never broken
  - Possible small increase in imbalance/cut edges

Partitioning Example

- 32-way partition of 30,562 point 2D grid



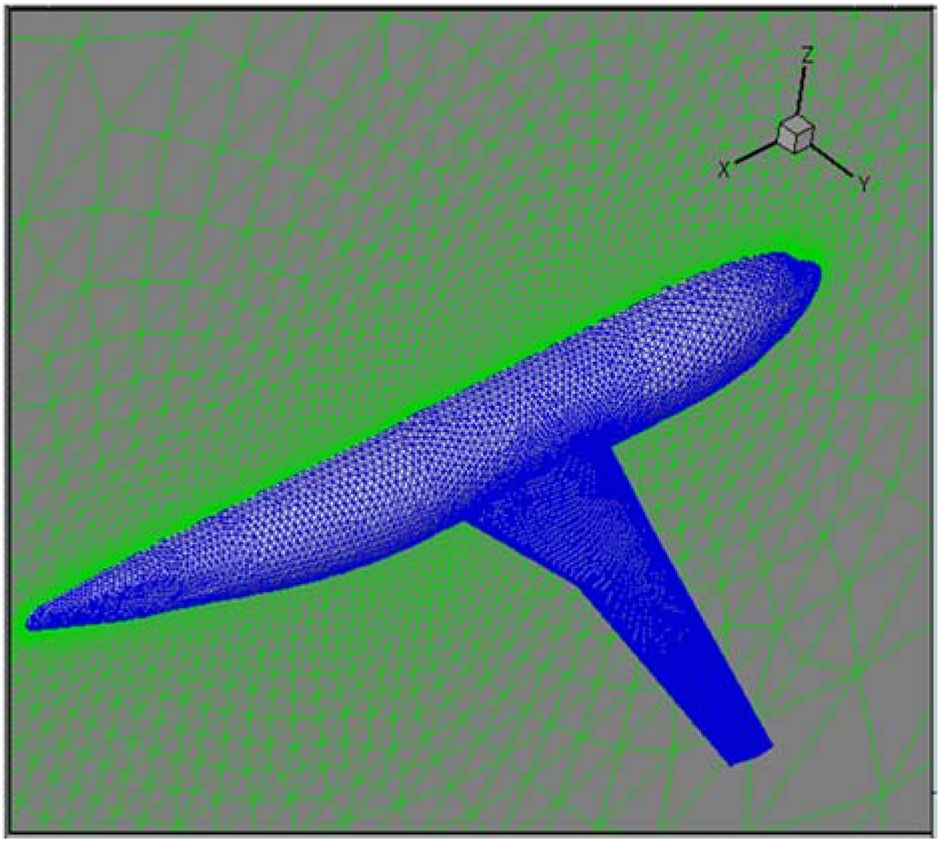


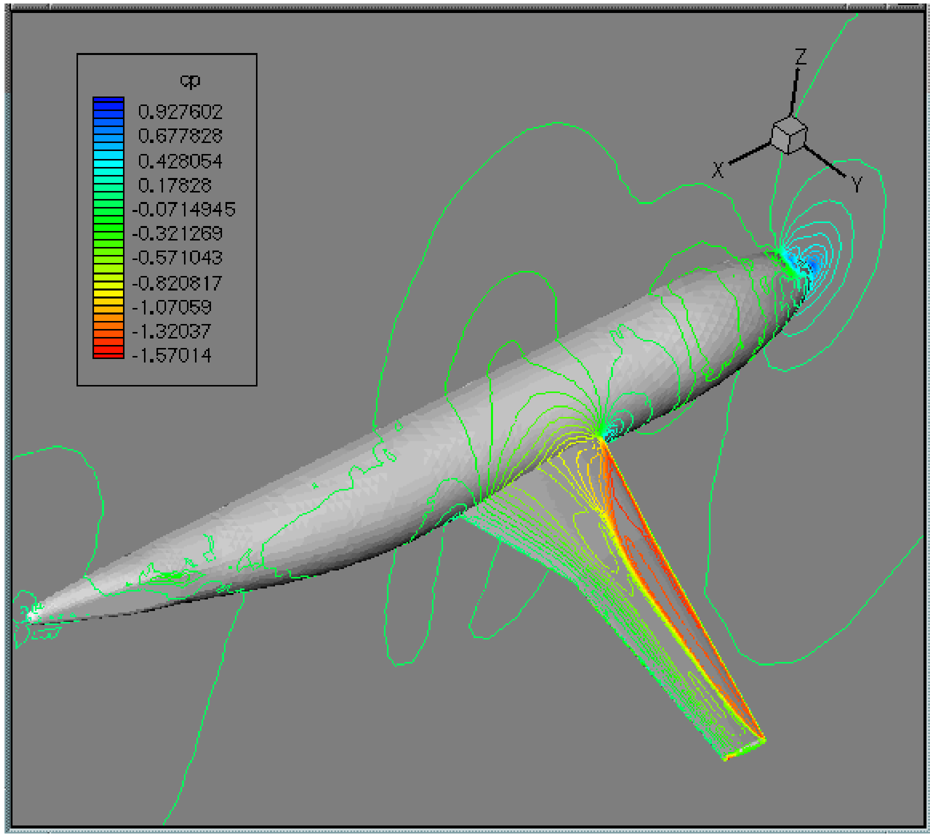
- Unweighted partition: 2.6% edges cut, 2.7% lines cut
- Weighted partition: 3.2% edges cut, 0% lines cut

Preprocessing Requirements

- Multigrid levels (graphs) are partitioned independently and then matched up through a greedy algorithm
  - Intragrid communication more important than intergrid communication
  - Became a problem at > 4000 cpus
- Preprocessing still done sequentially
  - Can we guarantee exact same solver behavior on different numbers of processors (at least as fallback)
    - Jacobi: Yes   Gauss Seidel: No
    - Agglomeration multigrid : frontal algorithm = no ?

AIAA Drag Prediction Workshop Test Case

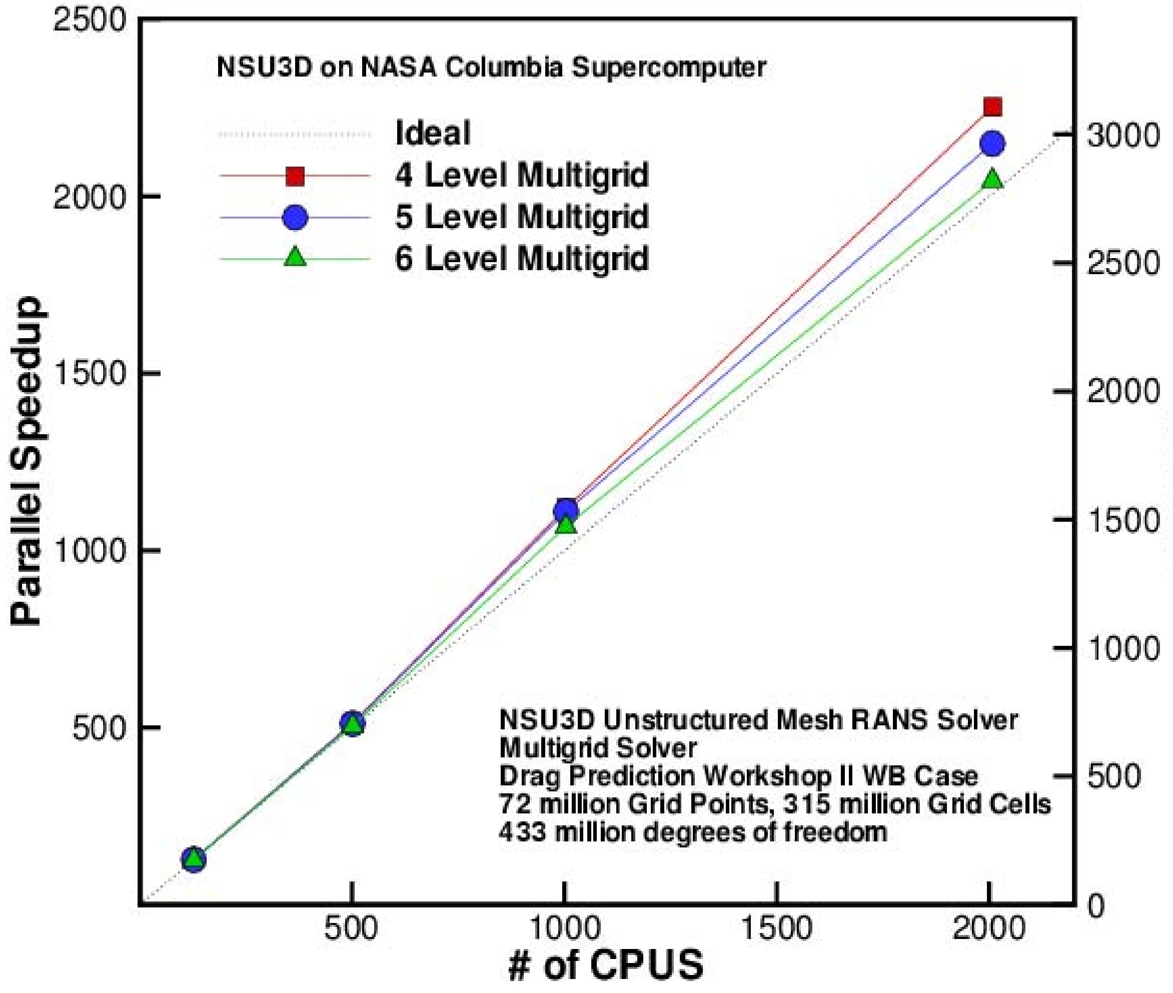




- Wing-Body Configuration (but includes separated flow)
- 72 million grid points
- Transonic Flow
- Mach=0.75, Incidence = 0 degrees, Reynolds number=3,000,000

NSU3D Scalability on NASA Columbia Machine

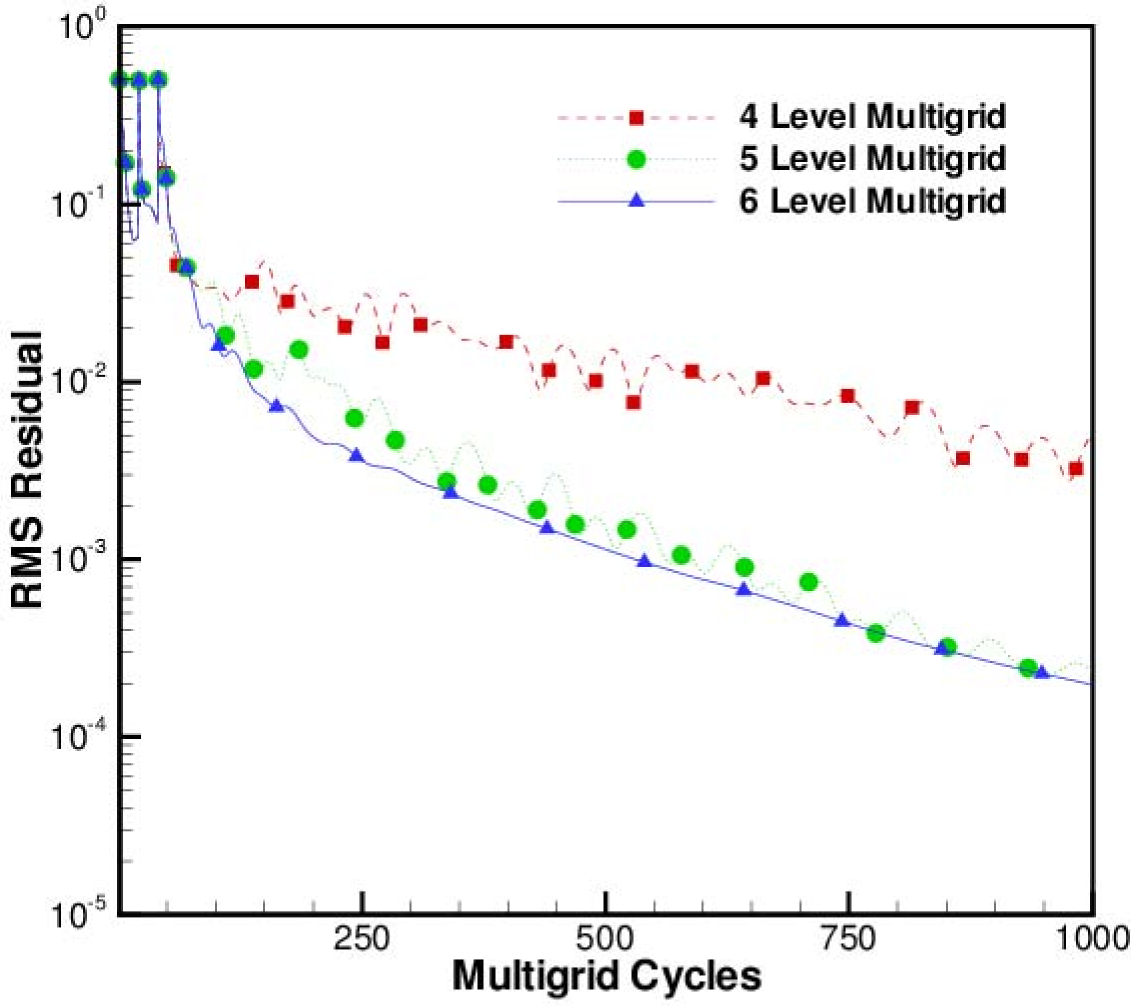
- 72M pt grid
  - Assume perfect speedup on 128 cpus
- Good scalability up to 2008 cpus
- Multigrid slowdown due to coarse grid communication
  - But yields fastest convergence



G
F
L
O
P
S

NSU3D Scalability

- Best convergence with 6 level multigrid scheme
- Importance of fastest overall solution strategy
  - 5 level Multigrid
  - 10 minutes wall clock time for steady-state solution on 72M pt grid



NSU3D Benchmark on BG/L

- Identical case as described on Columbia at SC05:
  - 72 million points, steady state MG solver
  - BG/L cpus ~ 1/ 3 of Columbia cpus: 333 Mflops/cpu
  - Solution in 20 minutes on 4016 cpus
    - Strong scalability: only 18,000 points per cpus

CPUs	Time/cycle	Scaling	Tflops (approx)
1004	9.6 secs	1.00	0.33
2008	5.06 secs	1.89/2.00	0.62
4016	2.64 secs	3.62/4.00	1.2

Note: Columbia one of a kind machine
Acess to > 2048 cpus difficult

Hybrid Parallel Programming

- With multicore architectures, we have clusters of SMPs
  - Hybrid MPI/OpenMP programming model
    - In theory:
      - Local memory access faster using OpenMP/Threads
      - MPI reserved for inter-node communication
      - Alternatively,do loop level parallelism at thread level on multicores
        - » (not recommmended so far, but may become necessary on many cores/cpus)

EXTENDING MPI CODE TO MIXED MPI-OpenMP MODEL

- MPI Process Rewritten to Handle Multiple Domains
  - Sequentially
  - In Parallel Using OpenMP
- Flexibility
  - Run MPI or OpenMP Exclusively
  - Run Two-Level MPI-OpenMP Model
  - Sequential Capability
    - Number of Domains can be Multiple of Number of Processors
- Entirely Domain-Based Parallelism

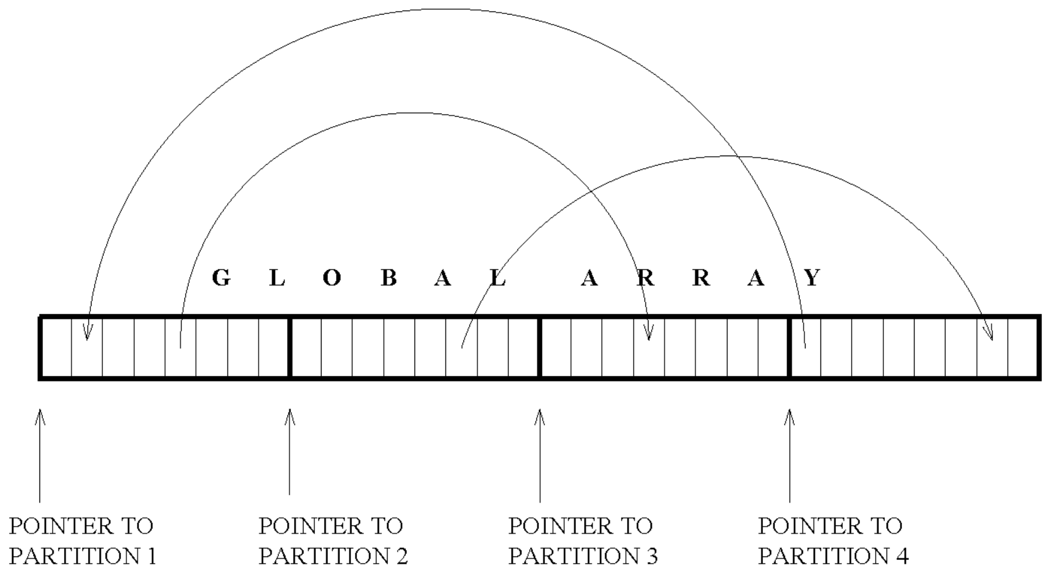
OVERALL CODE STRUCTURE

```
include OMP_DIRECTIVE
do : Loop over number of partitions
        do : Loop over number of vector groups
                do : Loop over edges in a vector group
                        n1 = edge_end(1,iedge)
                        n2 = edge_end(1,iedge)
                        flux = function of values at n1,n2
                        residual(n1) = residual(n1) + flux
                        residual(n2) = residual(n2) - flux
                enddo
        enddo
 enddo
c
include OMP_DIRECTIVE
do : Loop over number of partitions
        call OMP_communicate
enddo
c
include OMP_DIRECTIVE
do : Loop over number of partitions
        call MPI_communicate
enddo
```

- Entire Code OMP'ed with 2 or 3 Directives
- Distinct Partition Loops (instead of OMP BARRIER) enables Code to run Sequentially

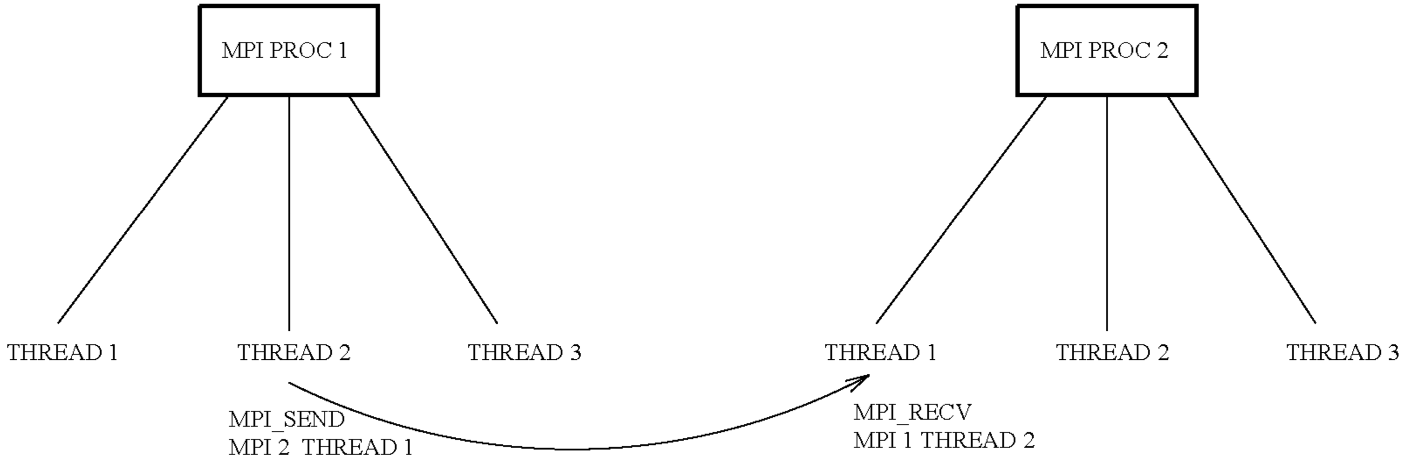
OPENMP COMMUNICATION (within an MPI Process)

- Arrays Span All Local Partitions/Threads
- Pointers used to Identify Extent of Each Partition/Thread
- Local Indices (relative to pointers) used in Computation Loop
- Global Indices Used for Communication
- Communicate by Copying Selected Values to Specific Locations in Global Array

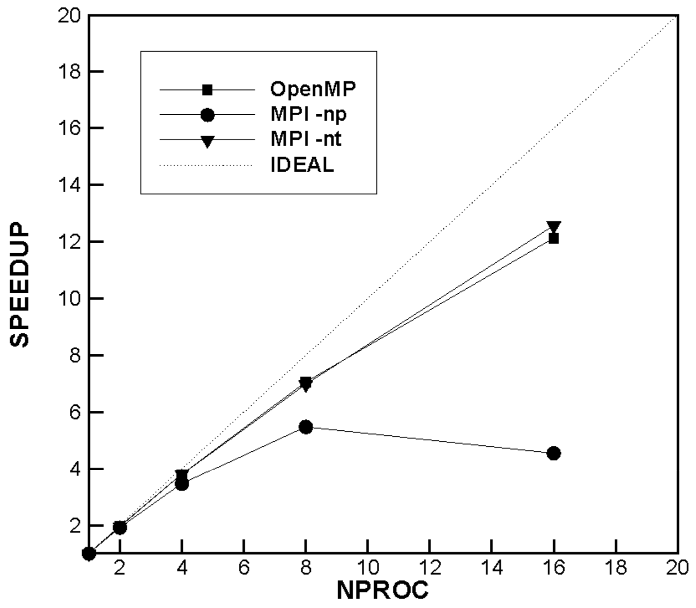


COMMUNICATION BETWEEN MPI PROCESSES

- Thread to Thread MPI Messages
  - Each Thread Sends to/ Receives from:
    - An MPI Process
    - A Thread Id (implemented as message tag)
- Entirely Parallel Provided MPI Implementation is THREAD-SAFE

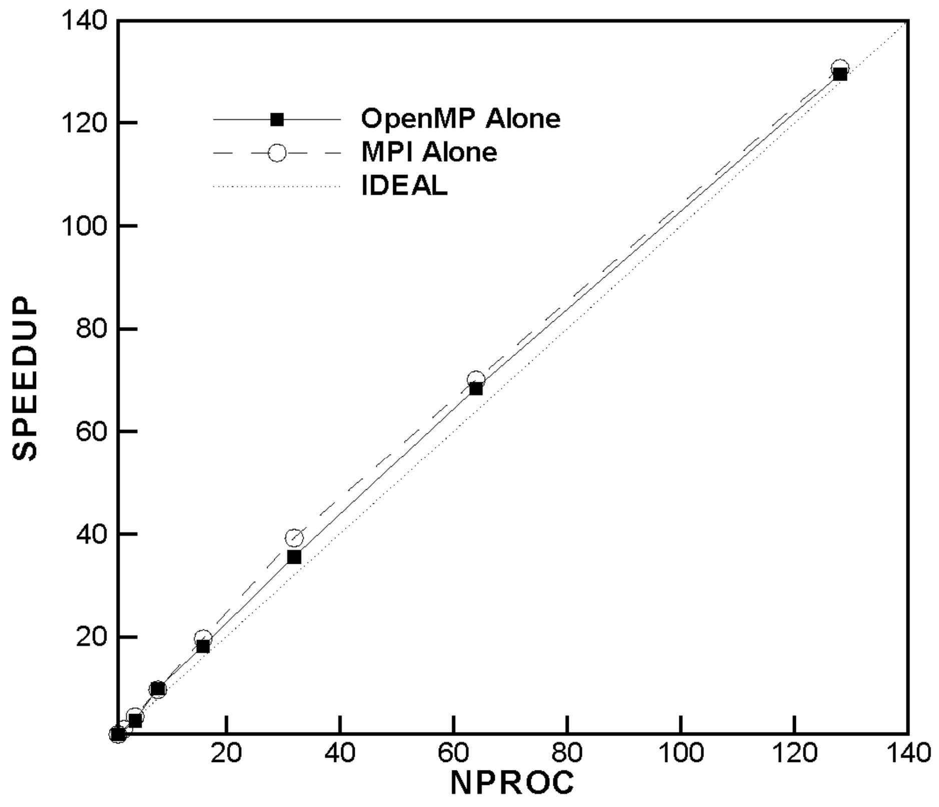


COMPARISON OF MPI and OPENMP on CRAY SV1



- Vector Machine with Uniform Access Memory
- Two Vendor MPI Implementations
  - MPI -np : Unix Sockets
  - MPI -nt : Shared Memory Communication
- 177K Point Grid, No Multigrid

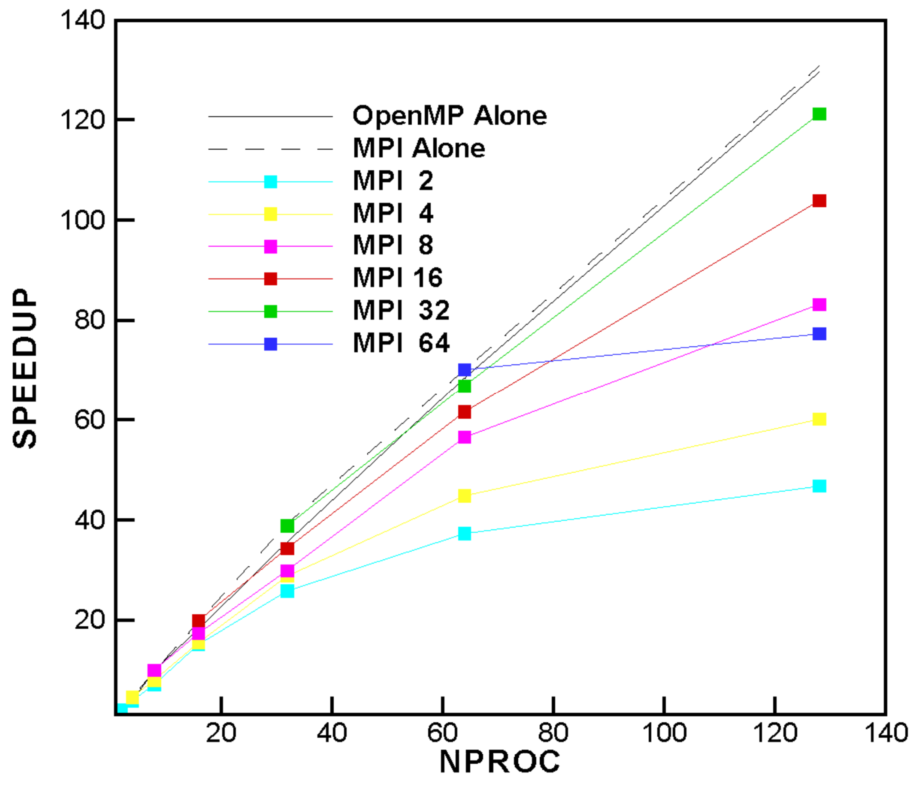
MPI vs. OPENMP ON SINGLE BOX OF ASCI BLUE MOUNTAIN



- OMP Uses Parallel Initialization (first touch memory placement)
- 3.1 million Point Grid, No Multigrid

Using domain based parallelism, OMP can perform as well as MPI

COMBINED MPI-OpenMP ON ASCI BLUE MOUNTAIN

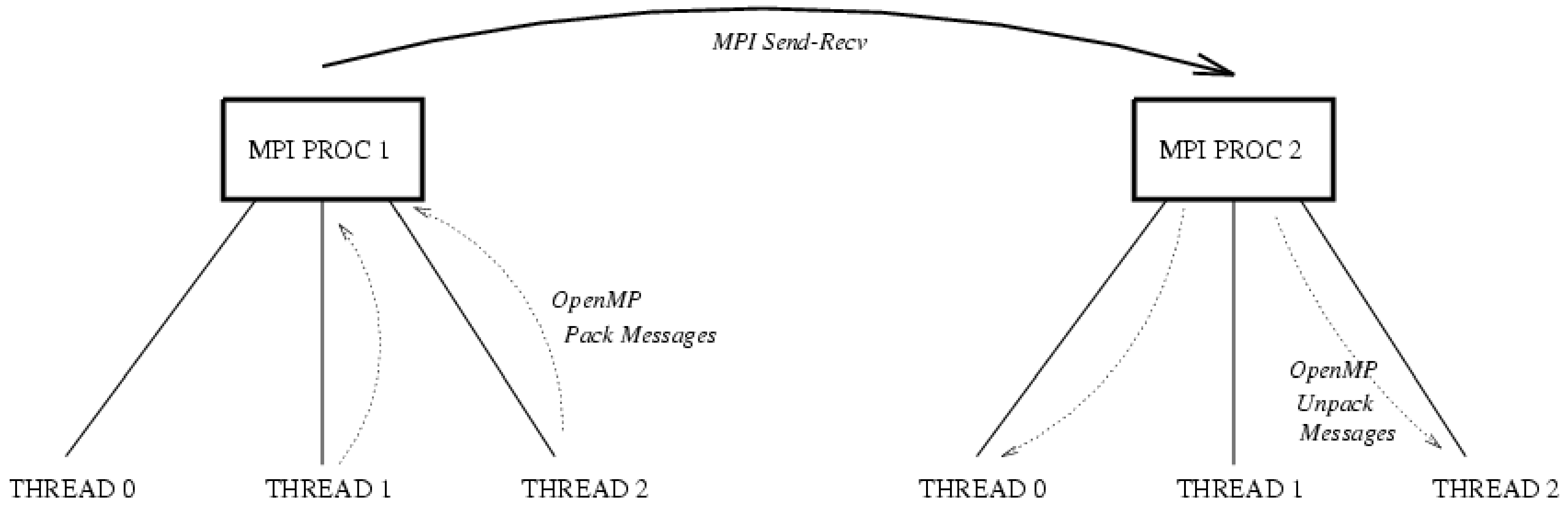


- 3.1 million Point Grid, No Multigrid

MPI/OpenMP PERFORMANCE

- OpenMP and MPI Perform Equivalently on SV1, O2000
  - Validates OMP Implementation
- Combined MPI-OMP Cases Show Degradation
  - Current Origin 2000 MPI Implementation NOT Completely THREAD-SAFE
    - Individual Thread MPI Calls are Sequentialized
    - Degradation Increases with Number of Threads
    - Acceptable for Small Numbers of Threads : Dual CPU Pentiums
- Requested Processor Map Not Always Held
  - Initialized Memory No Longer Local
  - Processes Double up On Single Processor (MPI 64, OMP 2)

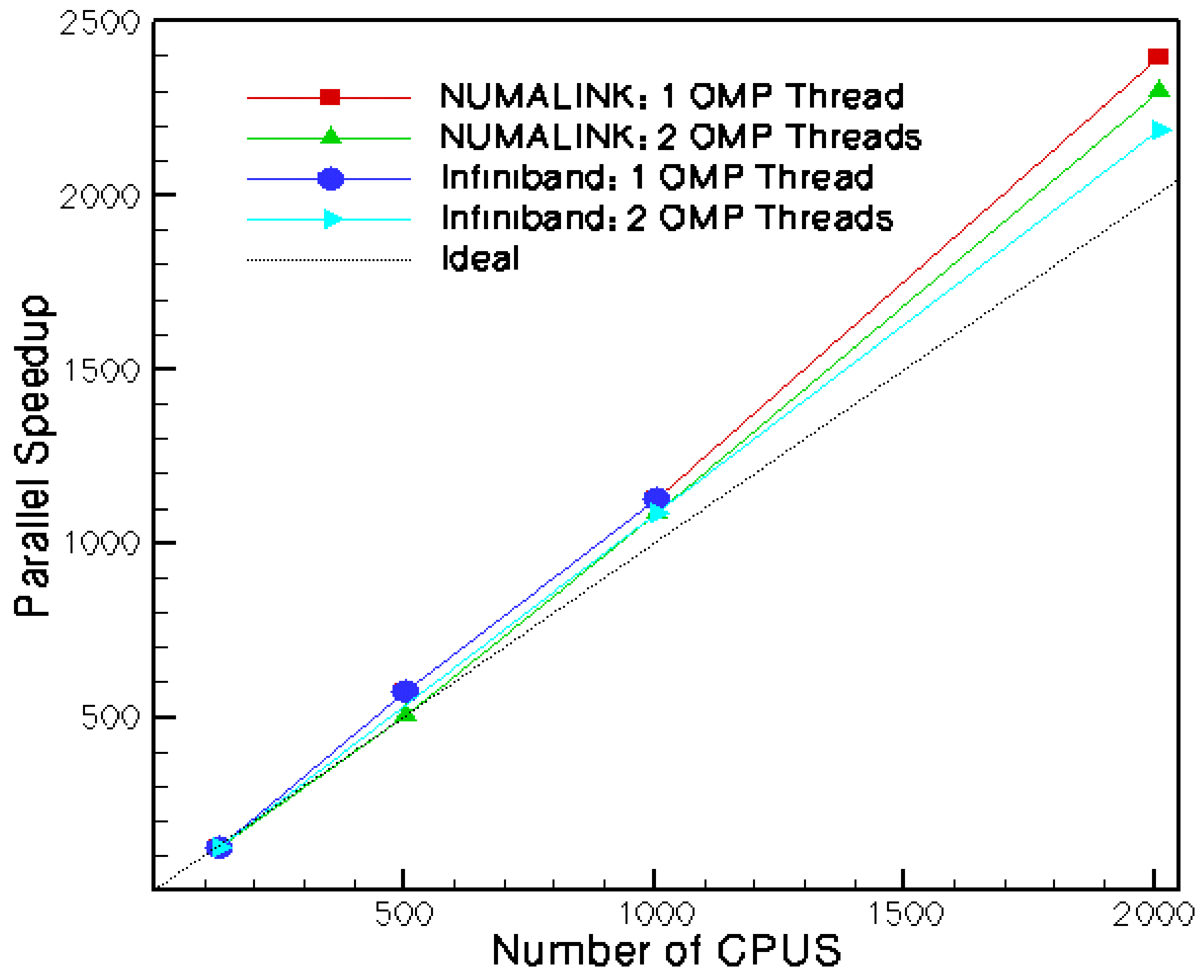
Hybrid MPI-OMP (NSU3D)



- MPI master gathers/scatters to OMP threads
- OMP local thread-to-thread communication occurs during MPI Irecv wait time (attempt to overlap)
- Unavoidable loss of parallelism due to (localy) sequential MPI Send/Recv

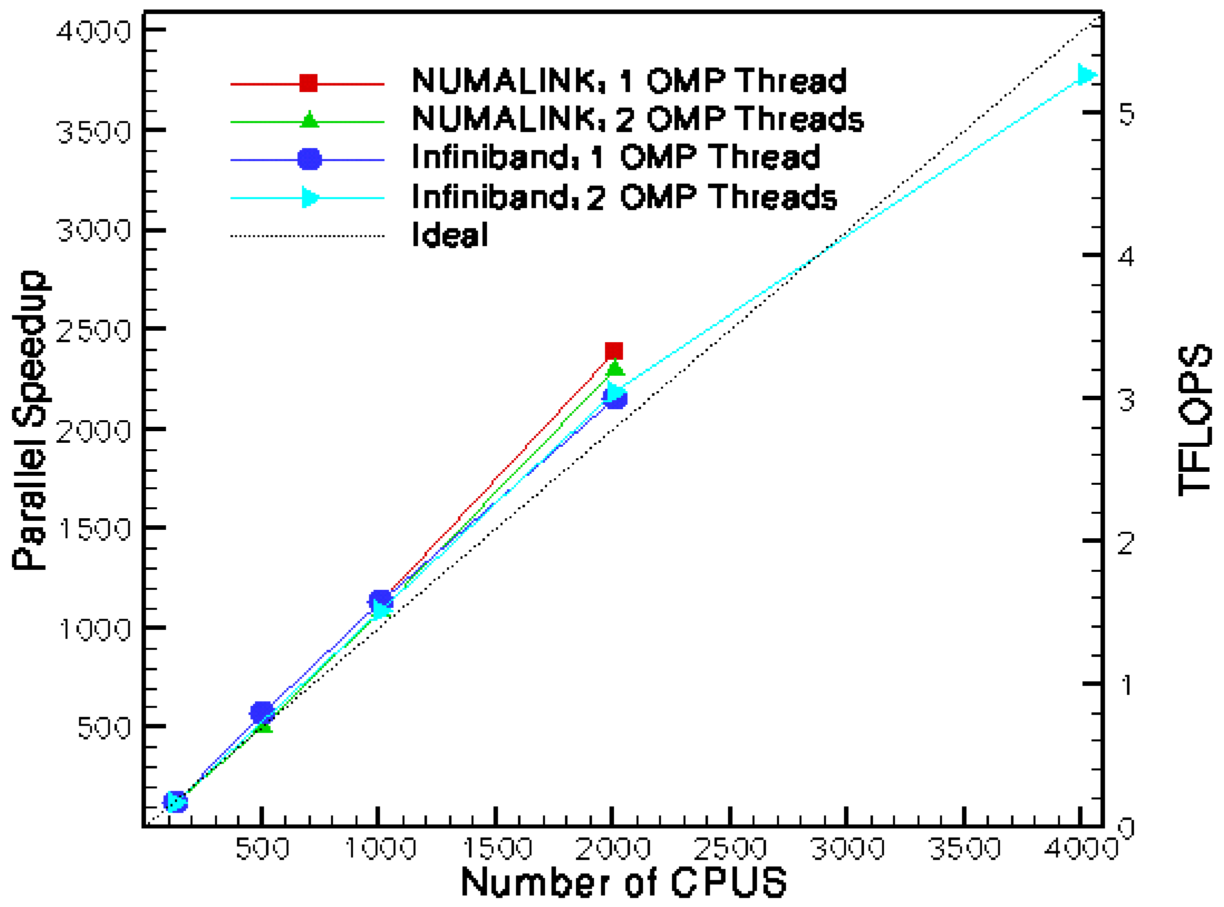
NASA Columbia Machine

72 million grid points



- 2 OMP required for IB on 2048
- Excellent scalability for single grid solver (non multigrid)

4016 cpus on Columbia (requires MPI/OMP)



- 1 OMP possible for IB on 2008 (8 hosts)
- 2 OMP required for IB on 4016 (8 hosts)
- Good scalability up to 4016
- 5.2 Tflops at 4016

Programming Models

- To date, have never found an architecture where pure MPI was not the best performing approach
  - Large shared memory nodes (SGI Altix, IBM P5)
  - Dual core, dual cpu commodity machines/clusters
- However, often MPI-OMP strategy is required to access all cores/cpus
- Problems to be addressed:
  - Shared memory benefit of OMP not realized
  - Sequential MPI Send-Recv penalty
  - Thread-safe issues
  - May be different → 1M cores

High Order Methods

- Higher order methods such as Discontinuous Galerkin best suited to meet high accuracy requirements
  - Asymptotic properties
- HOMs scale very well on massively parallel architectures
- HOMs reduce grid generation requirements
- HOMs reduce grid handling infrastructure
  - Dynamic load balancing
- Compact data representation (data compression)
  - Smaller number of modal coefficients versus large number of point-wise values

Single Grid Steady-State Implicit Solver

- Steady state

$$\boxed{R_p(U_p) = S_p}$$

- Newton iteration

$$\left[\frac{\partial R_p}{\partial U_p}\right]^n \Delta U_p^{n+1} = S_p - R_p(U_p^n)$$

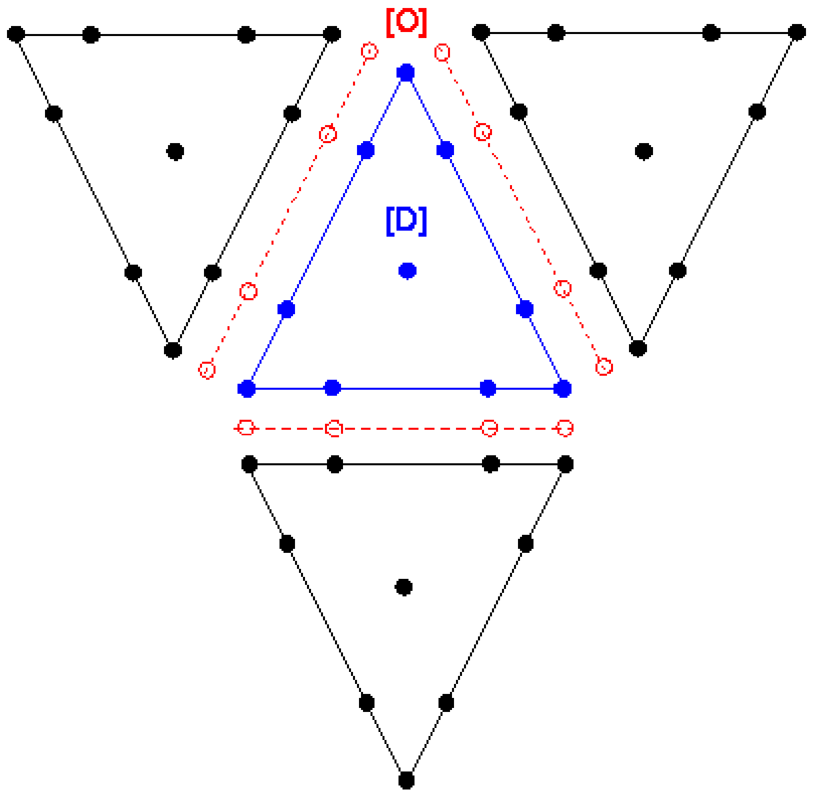
- Non-linear update

$$U_p^{n+1} = U_p^n + \Delta U_p^{n+1}$$

- [D] is Jacobian approximation

- Non-linear element-Jacobi (NEJ)

$$\boxed{\Delta U_p^{n+1} = \left[D_p^n\right]^{-1}\left(S_p - R_p(U_p^n)\right)}$$



p	Size of [D]
1	5 x 5
2	20 x 20
3	50 x 50
4	100 x 100
5	280 x 280
6	420 x 420

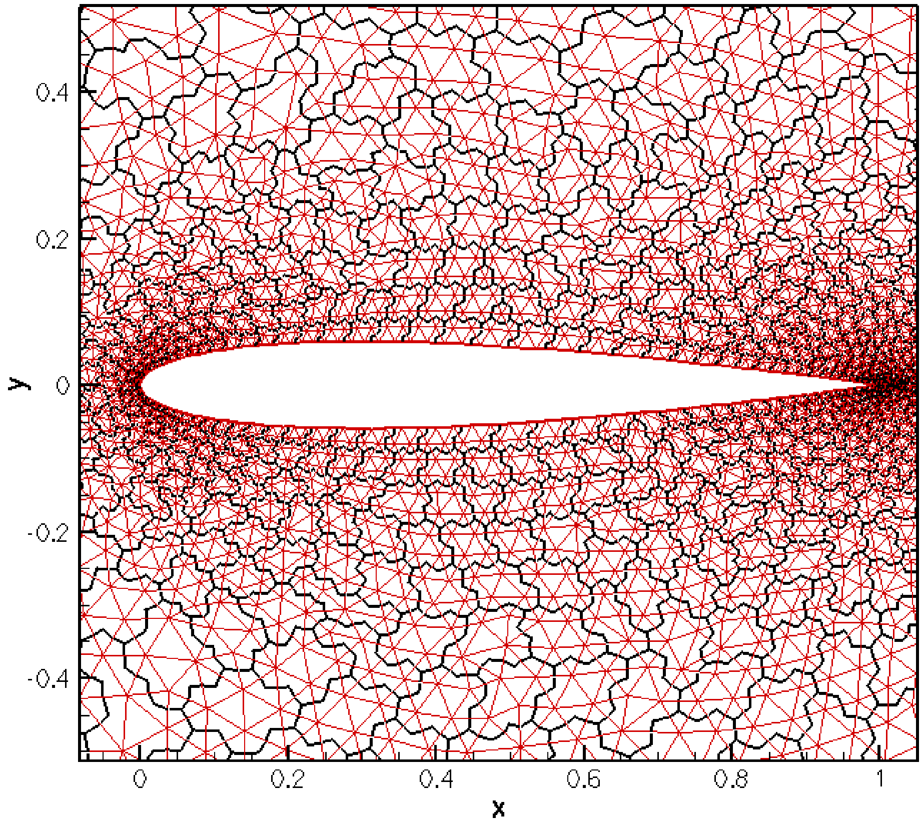
The Multigrid Approach: *p*-Multigrid

- *p*-Multigrid (Fidkowski et al., Helenbrook B. and Mavriplis D. J.)
  - ➢ Fine/coarse grids contain the same number of elements
  - ➢ Transfer operators almost trivial for hierarchical basis
  - ➢ Restriction: Fine -> Coarse: p = 4 → 3 → 2 → 1
    - ✓ Omit higher order modes
  - ➢ Prolongation: Coarse -> Fine
    - ✓ Transfer low order modal coefficients exactly
    - ✓ High order modal coefficients set to zero
  - ➢ For *p* = 1 → 0
    - ✓ Solution restriction: average
    - ✓ Residual restriction: summation
    - ✓ Soution prolongation: injection

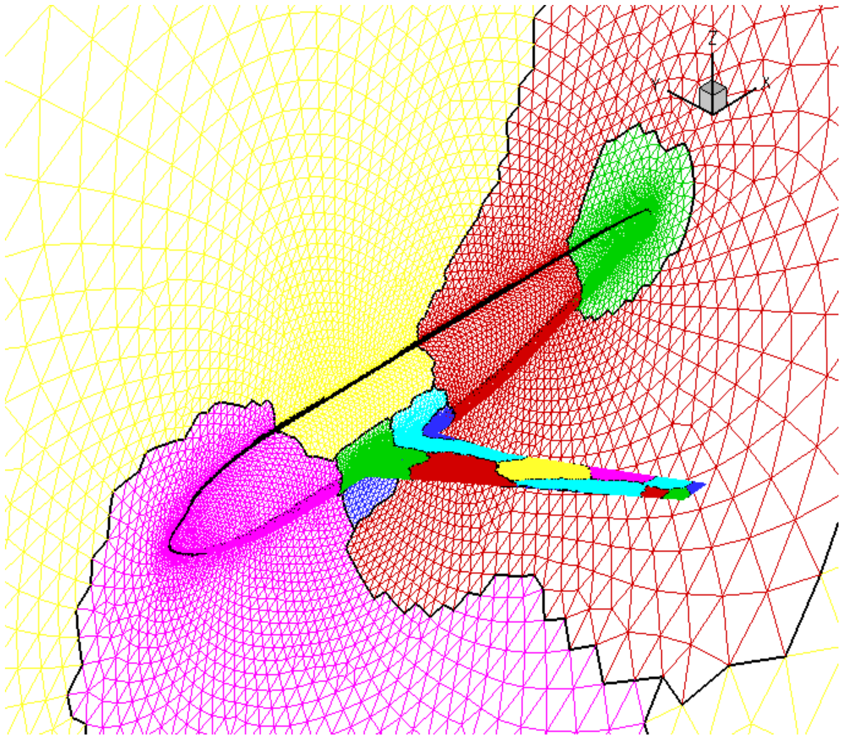
The Multigrid Approach: *h*-Multigrid

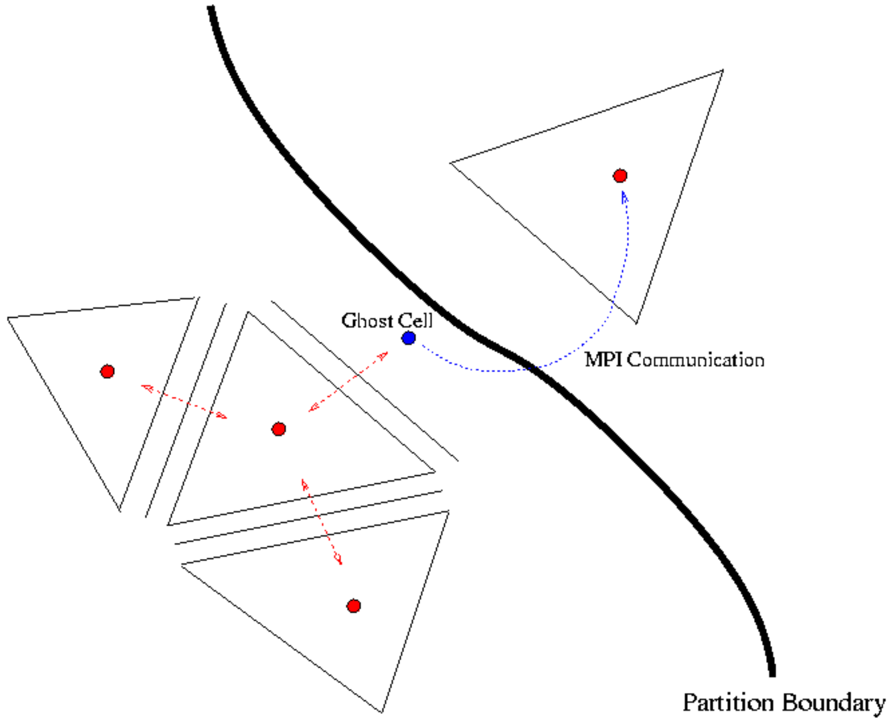
- *h*-Multigrid (Mavriplis D. J.)
  - Begins at p=0 level
  - Agglomeration multigrid (**AMG**)

- <u>*hp*</u>-Multigrid strategy:
  - Non-linear multigrid (**FAS**)
  - Full multigrid (**FMG**)



Parallel Implementation





- MPI buffers
  - Ghost cells
  - *p*-Multigrid
  - *h*-Multigrid (AMG)

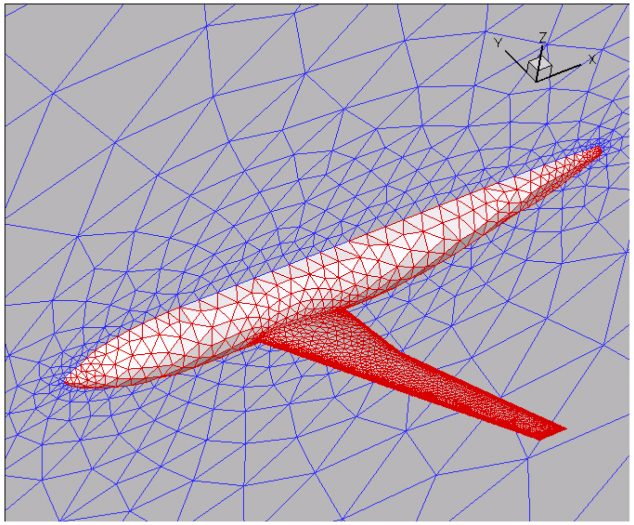
Parallel hp-Multigrid Implementation

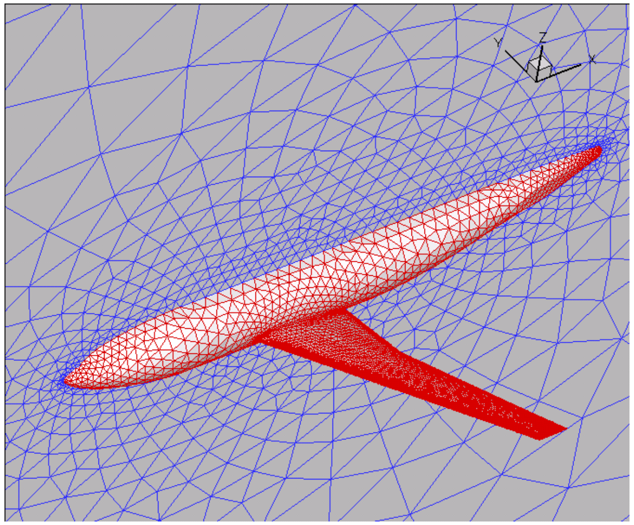
- *p*-MG
  - Static grid
  - Same MPI communication for all levels
  - No duplication of computation in adjacent partition
  - No communication required for restriction and prolongation

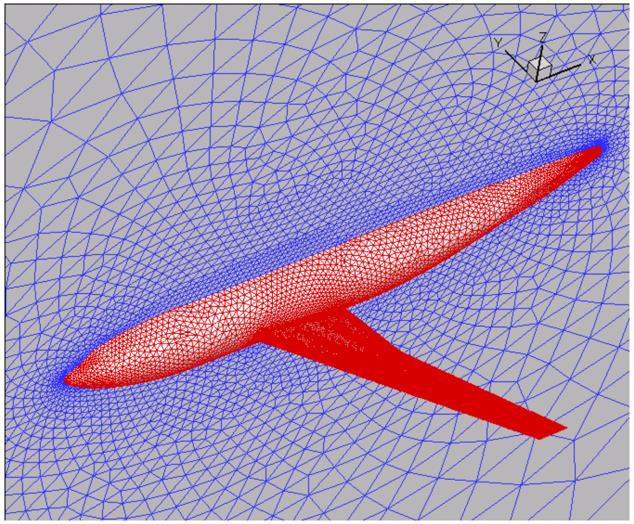
- *h*-MG
  - Each level is partitioned independently
  - Each level has its own communication pattern
  - Additional communication is required for restriction and prolongation
  - But h-levels represent almost trivial work compared to the rest

- Partitioning and communication patterns/buffers are performed sequentially and stored a priori (pre-processor)

Complex Flow Configuration (DRL-F6)



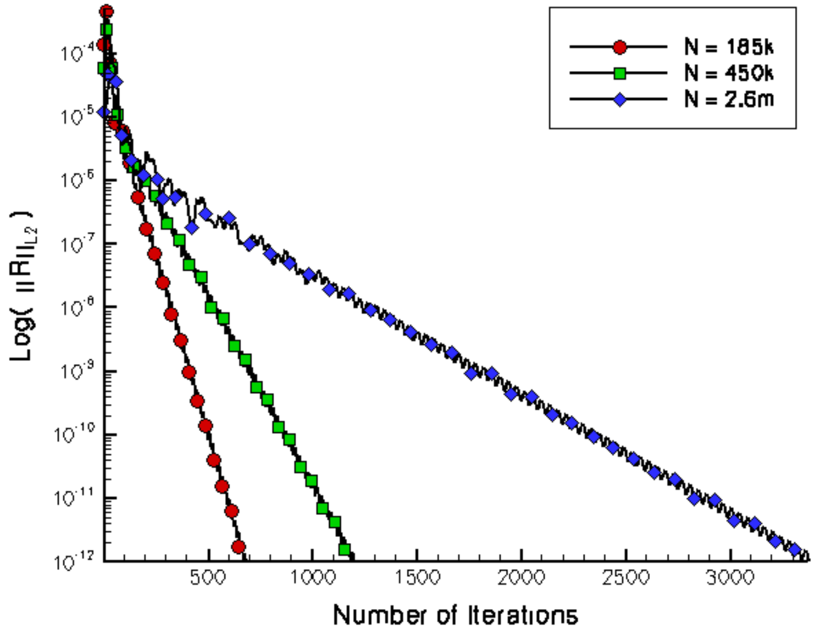


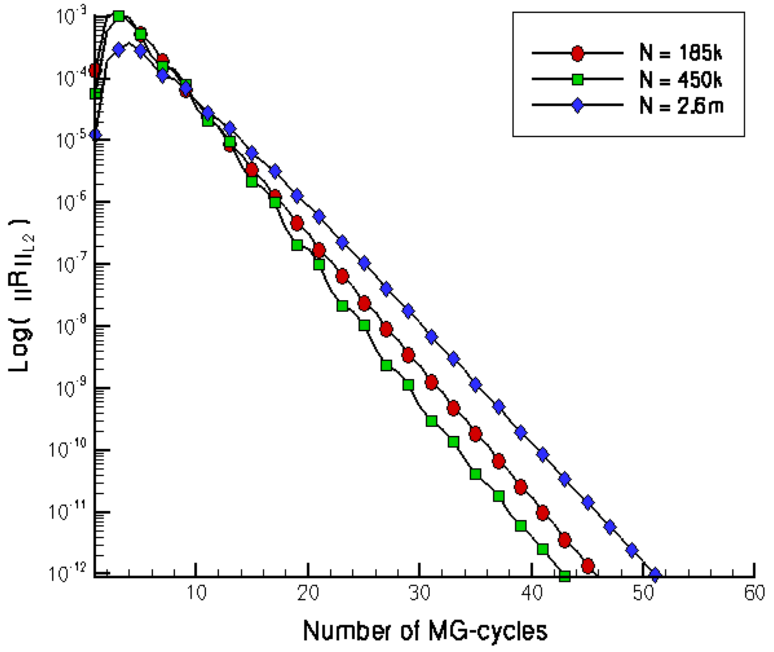


Case	N	AMG-levels
1	185k	4
2	450k	5
3	2.6m	6

- ICs: Freestream Mach=0.5
- *hp*-Multigrid
  - **qNJ** smoother
  - *p*=[0...4]
  - V-cycle(10,0)
  - FMG (10 cyc/level)

Results: p=0

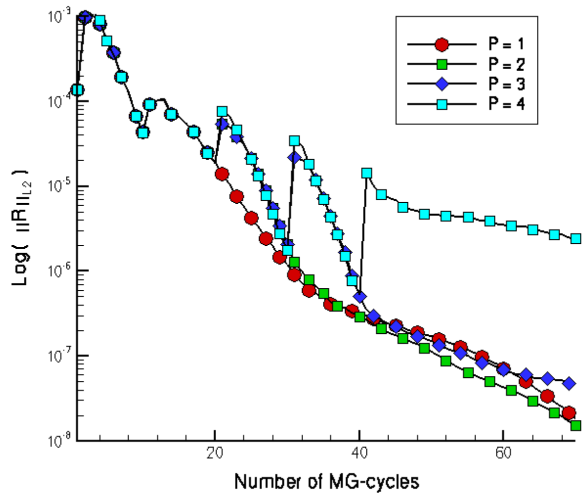




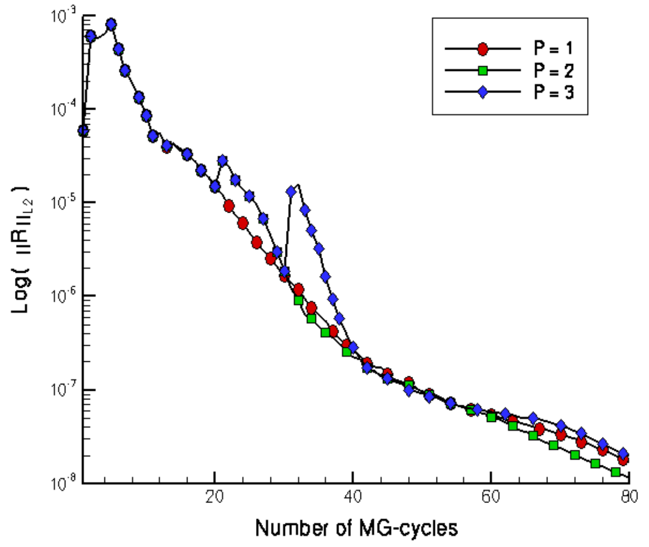
N	Single grid	AMG	AMG-levels
185k	679	46	4
450k	1200	43	5
2.6m	3375	51	6

hp-Multigrid: *p*-dependence

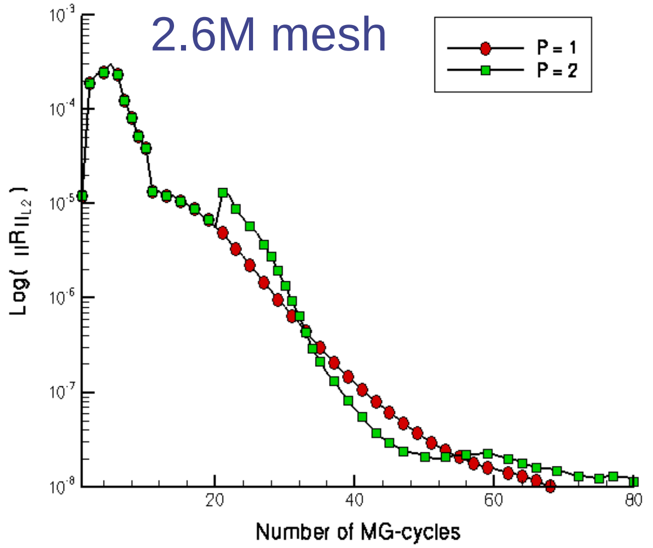
185K mesh

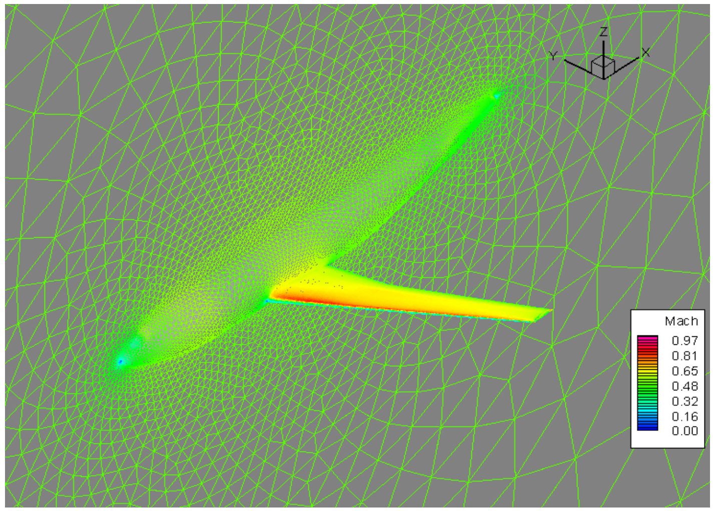


450K mesh

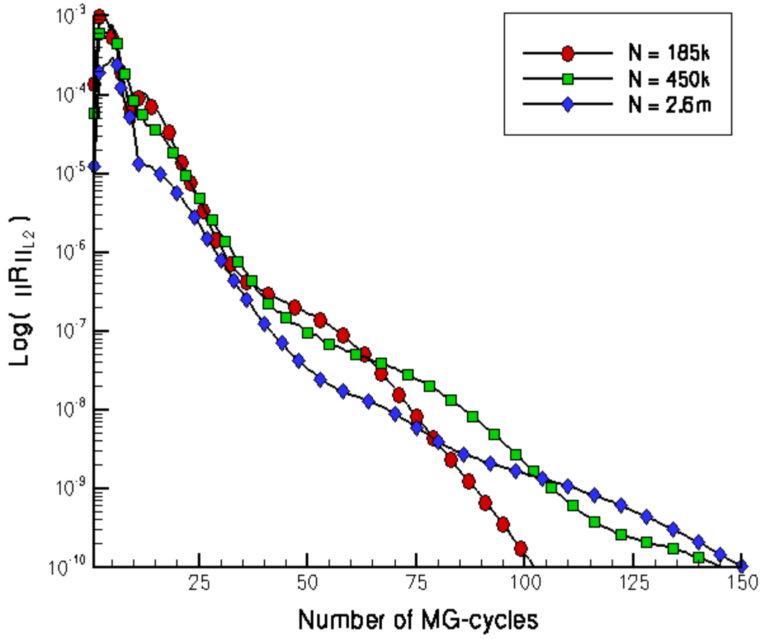


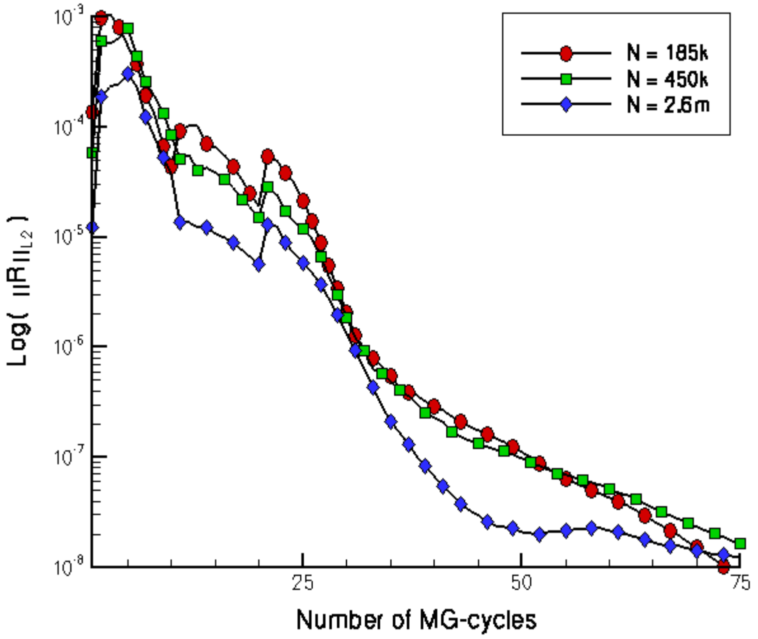
2.6M mesh





hp-Multigrid: *h*-dependence



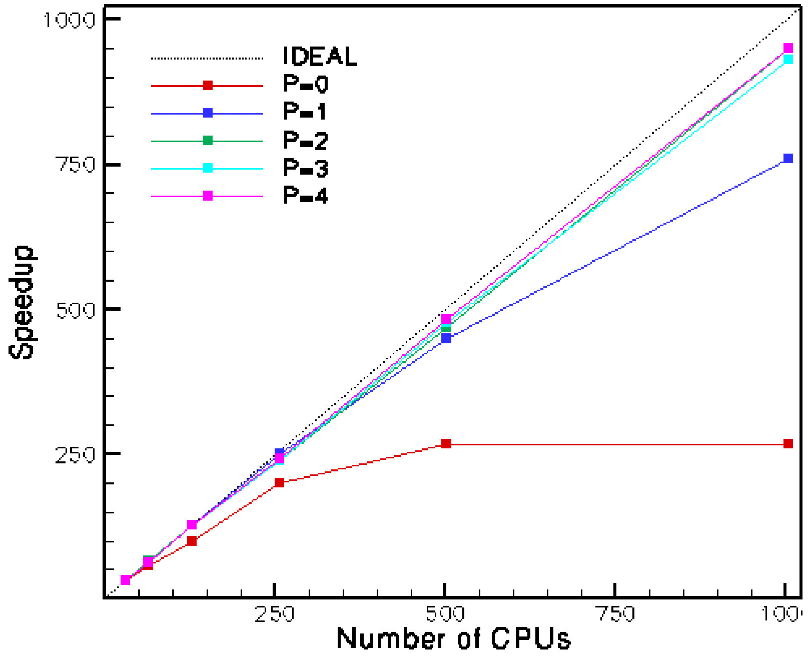


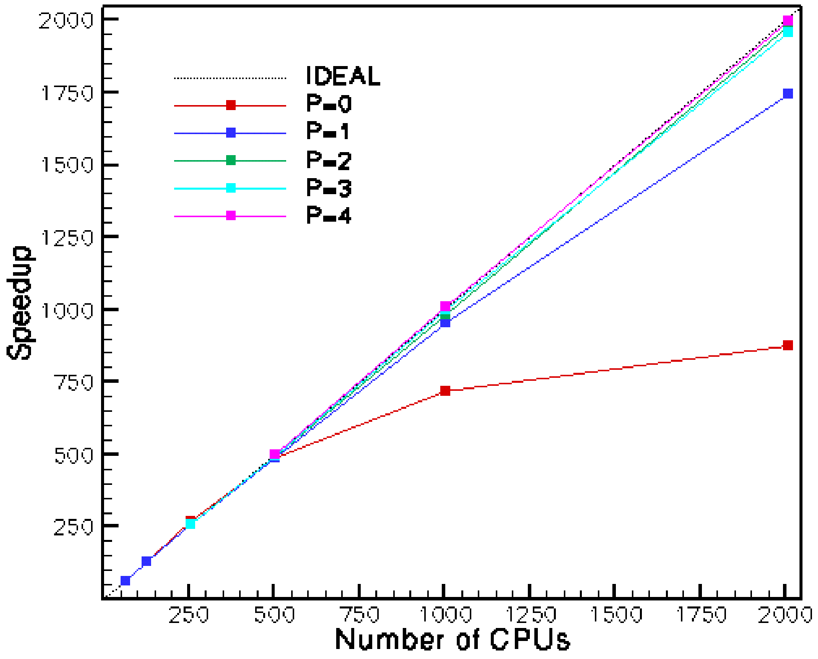
- $p = 1$
- $p = 2$

Parallel Performance: Speedup (1 MG-cycle)

- N = 185 000

- N = 2 600 000





- $p=0$ does not scale
- $p=1$ scales up to 500 proc.
- $p>1$ scales almost optimal

- $p=0$ does not scale
- $p=1$ scales up to 1000 proc.
- $p>1$ ideal scalability

Concluding Remarks

- Petascale computing will likely look very similar to terascale computing:
  - MPI for inter-processor communication
  - Perhaps hybrid MPI-OMP paradigm
- Can something be done to take advantage of shared memory parallelism more effectively ?
  - MPI still appears to be best
  - 16 way nodes will be common (quad core, quad cpu)
- Previously non-competitive methods which scale well may become methods of choice
- High-order methods (in space and time)
  - Scale well
  - Reduce grid infrastructure problems
  - Compact (compressed) representation of data