

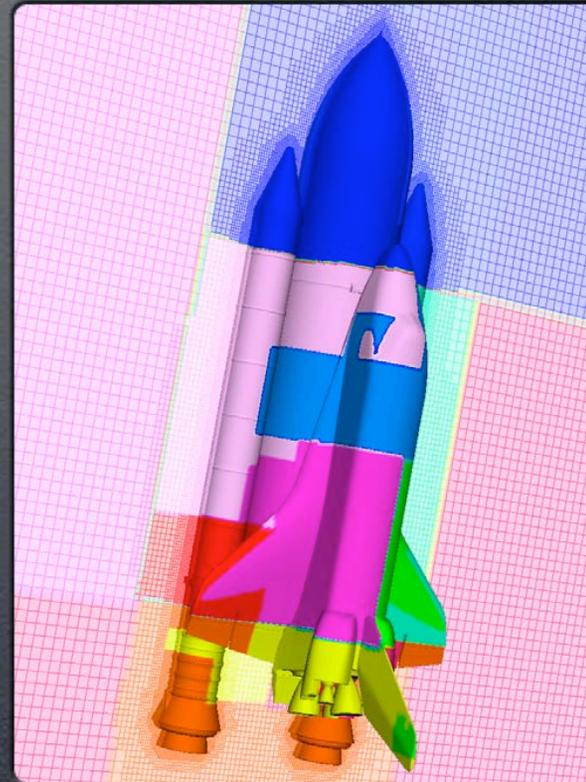
Performance of a New CFD Solver Using a Hybrid Programming Paradigm

M. Aftosmis

NASA Ames Research Center
Moffett Field, CA

M. Berger

Courant Institute
New York University, NY



SIAM Conference on

Parallel Processing for Scientific Computing

San Francisco, CA

February 25-27, 2004





Motivation

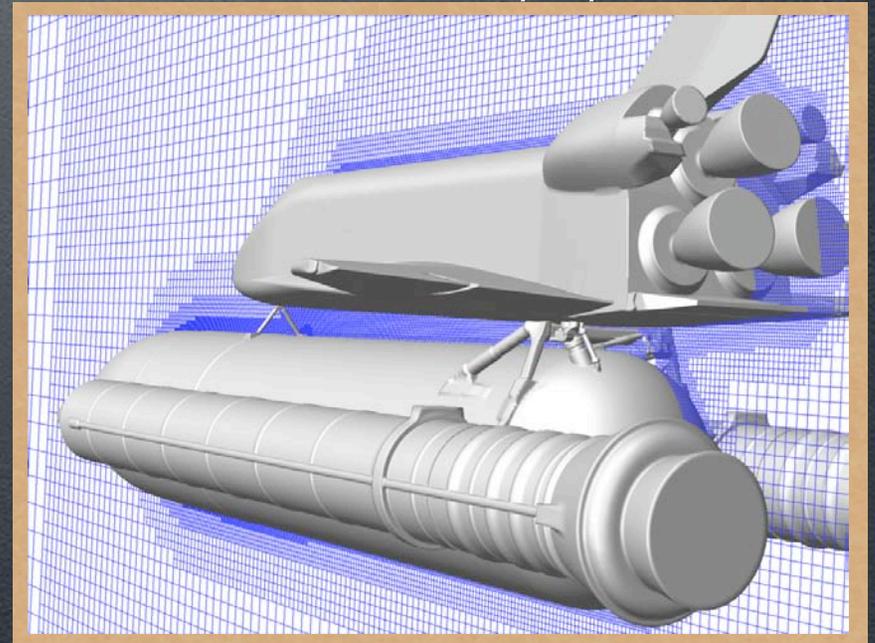
Cartesian methods offer solutions at all levels of fidelity

Support clustered PCs to supercomputers

- OpenMP for quick development and shared-memory systems
- MPI on distributed clusters
- Attention to memory locality

Hybrid programming paradigm

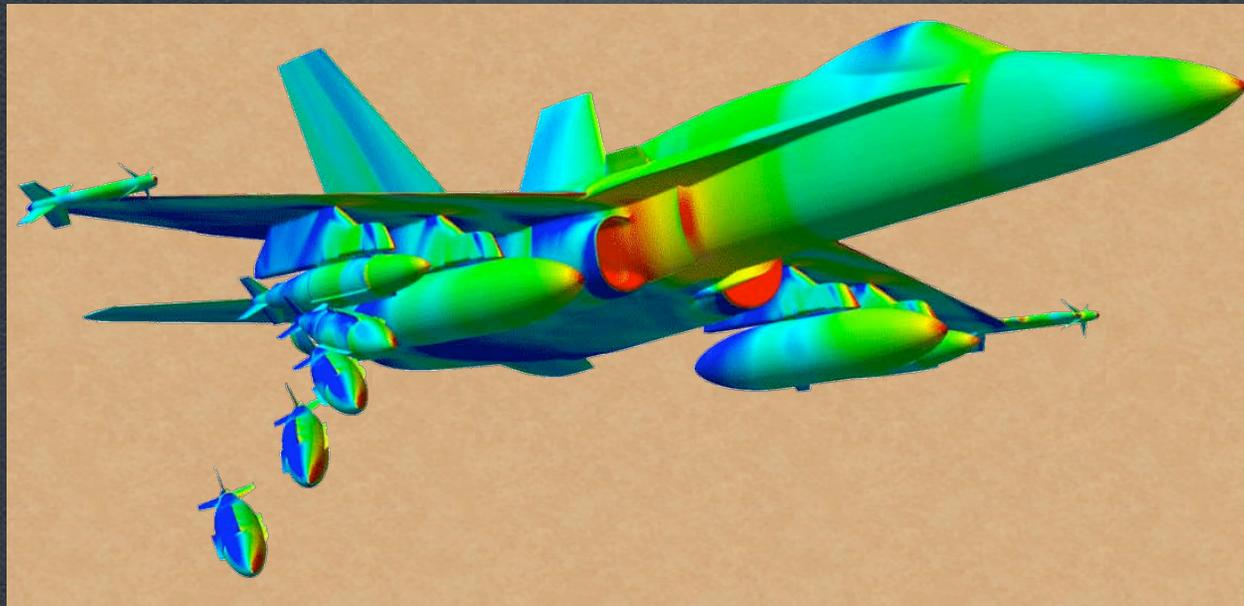
- Domain decomposition - Each CPU integrates its subdomain
- Explicit exchanges between subdomains
- Develop in OpenMP





Outline

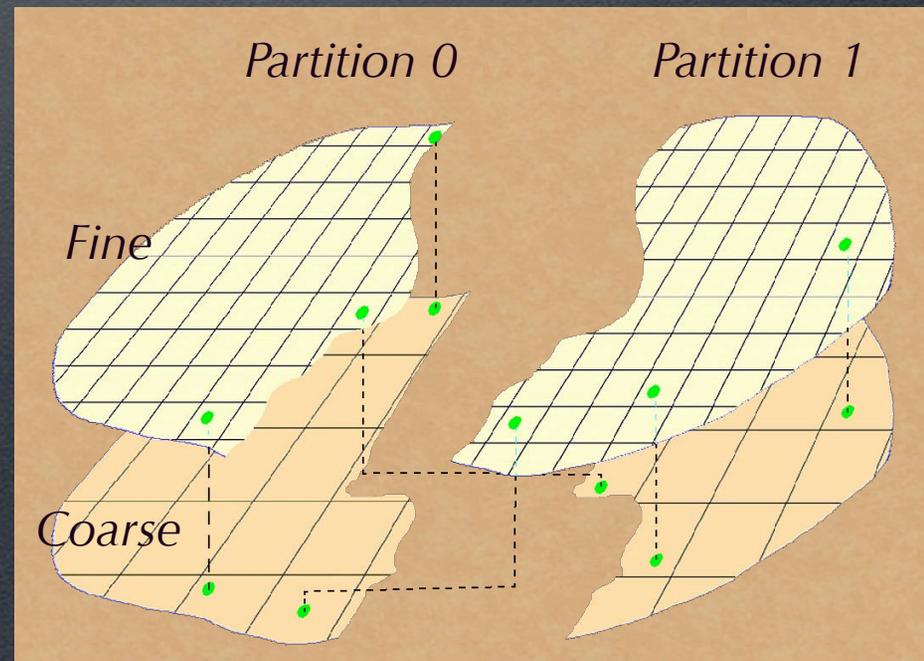
- Basic Approach
- Space-filling-curves (SFC)
 - Multigrid / Domain decomposition
- Conversion to MPI
- Parallel scalability
- Summary





Basic Approach

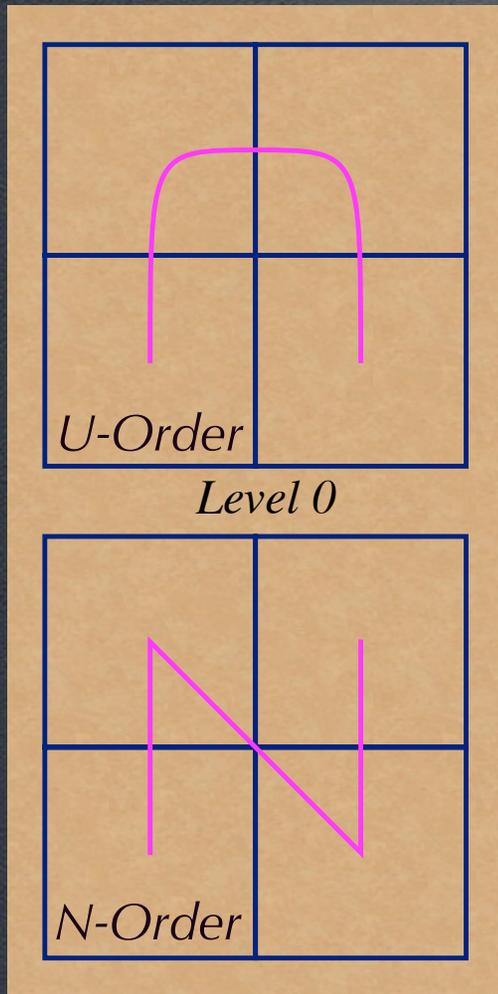
- Explicit Euler solver with multigrid acceleration
- Subdomains reside on processor local memory
 - Each subdomain has own local grid hierarchy
 - Exchange via structure copy (OpenMP), send/receive (MPI)
 - Restrict use of OpenMP constructs
- Common code base
- Customize
 - Initial distribution
 - Boundary exchanges
 - Gather/Scatter





Space-Filling-Curves

Peano-Hilbert and Morton Ordering in 2D



Peano-Hilbert: U-Order

- Basic building block is *U-shaped* curve visiting each 2x2 block
- Subsequent levels replace each quadrant with U-shaped curves

Morton: N-Order

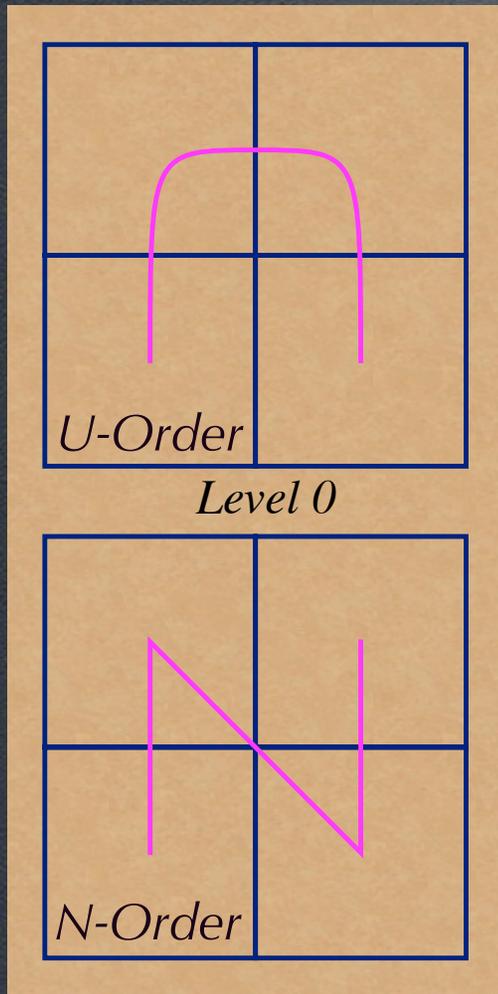
- Basic building block is *N-shaped* curve visiting each 2x2 block
- Subsequent levels replace each quadrant with N-shaped curves

(Salmon94, Griebel96, Baden96, Plimpkin98, Behrens00, ...)



Space-Filling-Curves

Peano-Hilbert and Morton Ordering in 2D

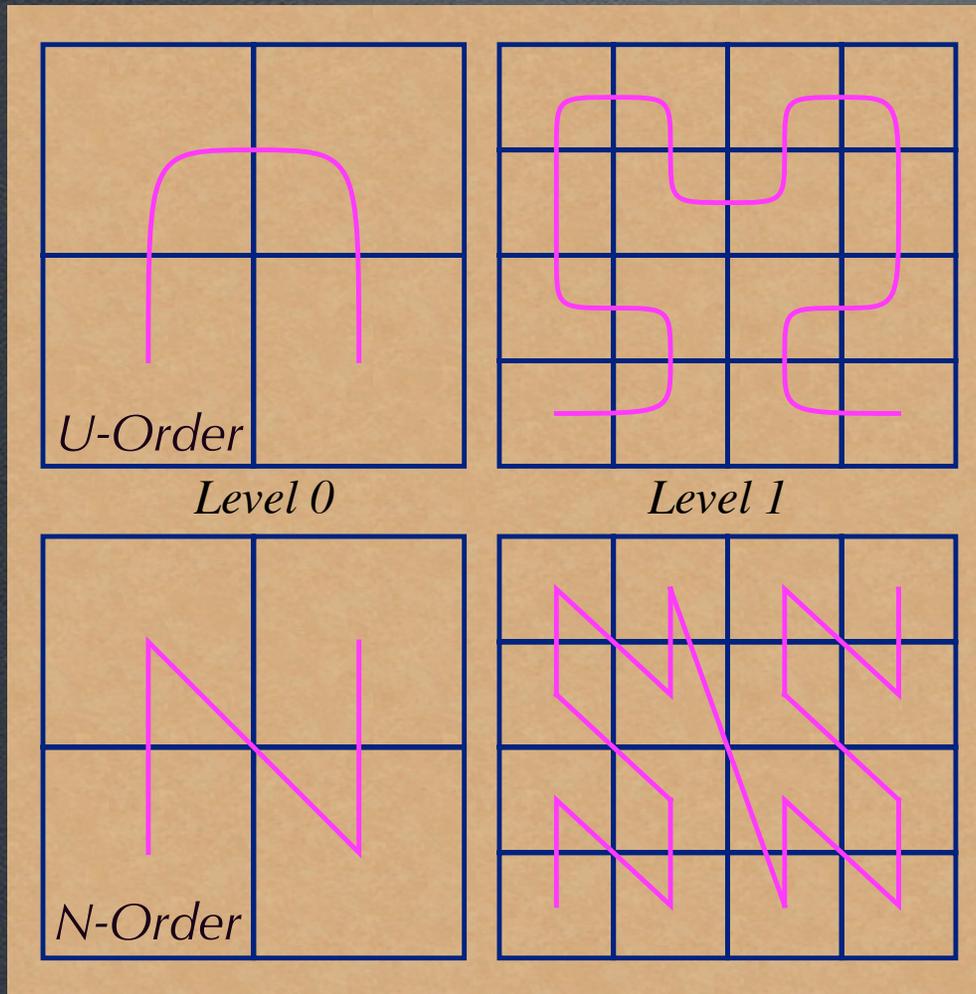


(Salmon94, Griebel96, Baden96, Plimpkin98, Behrens00, ...)



Space-Filling-Curves

Peano-Hilbert and Morton Ordering in 2D

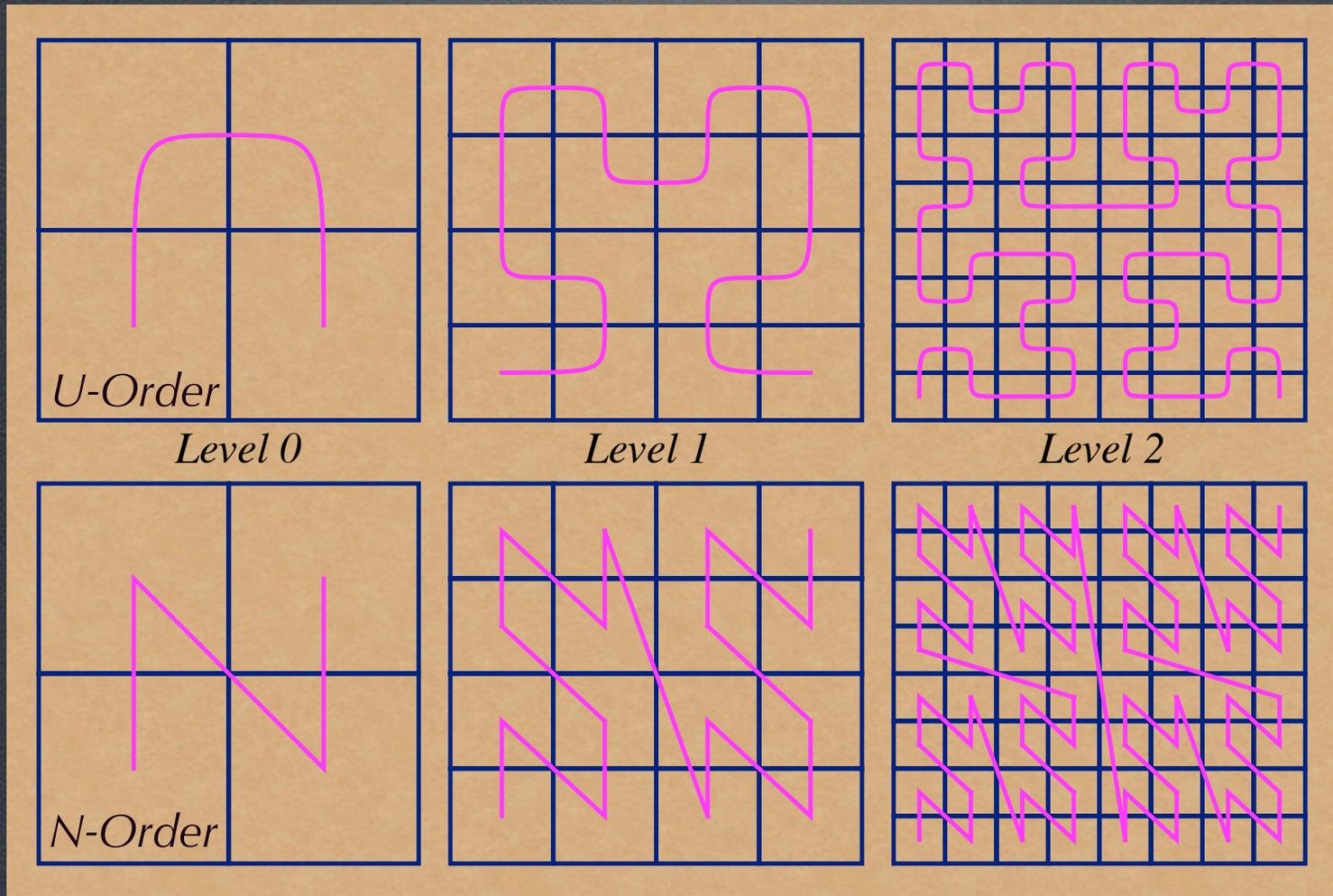


(Salmon94, Griebel96, Baden96, Plimpkin98, Behrens00, ...)



Space-Filling-Curves

Peano-Hilbert and Morton Ordering in 2D

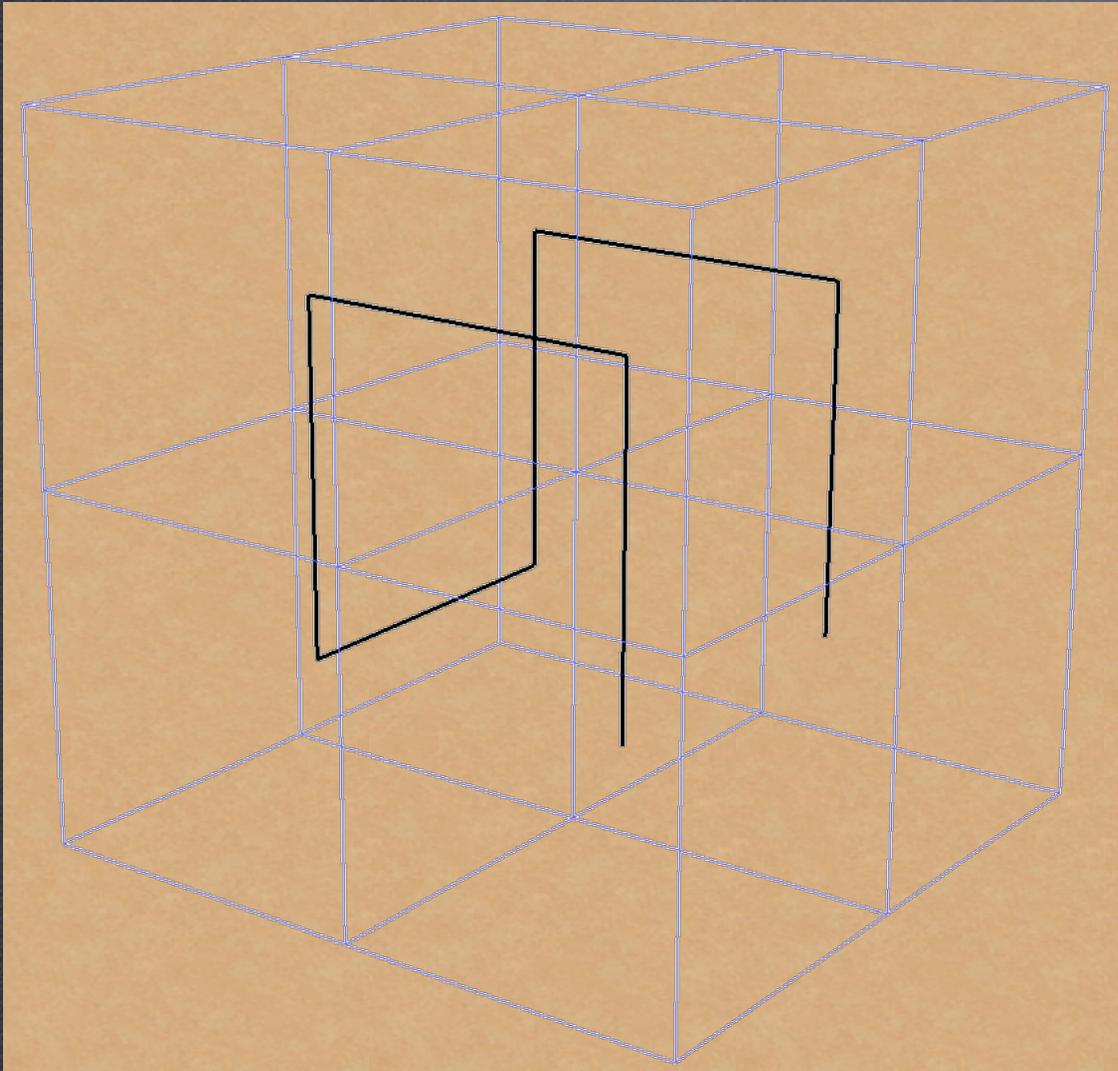


(Salmon94, Griebel96, Baden96, Plimpkin98, Behrens00, ...)



Space-Filling-Curves

Peano-Hilbert and Morton Ordering in 3D

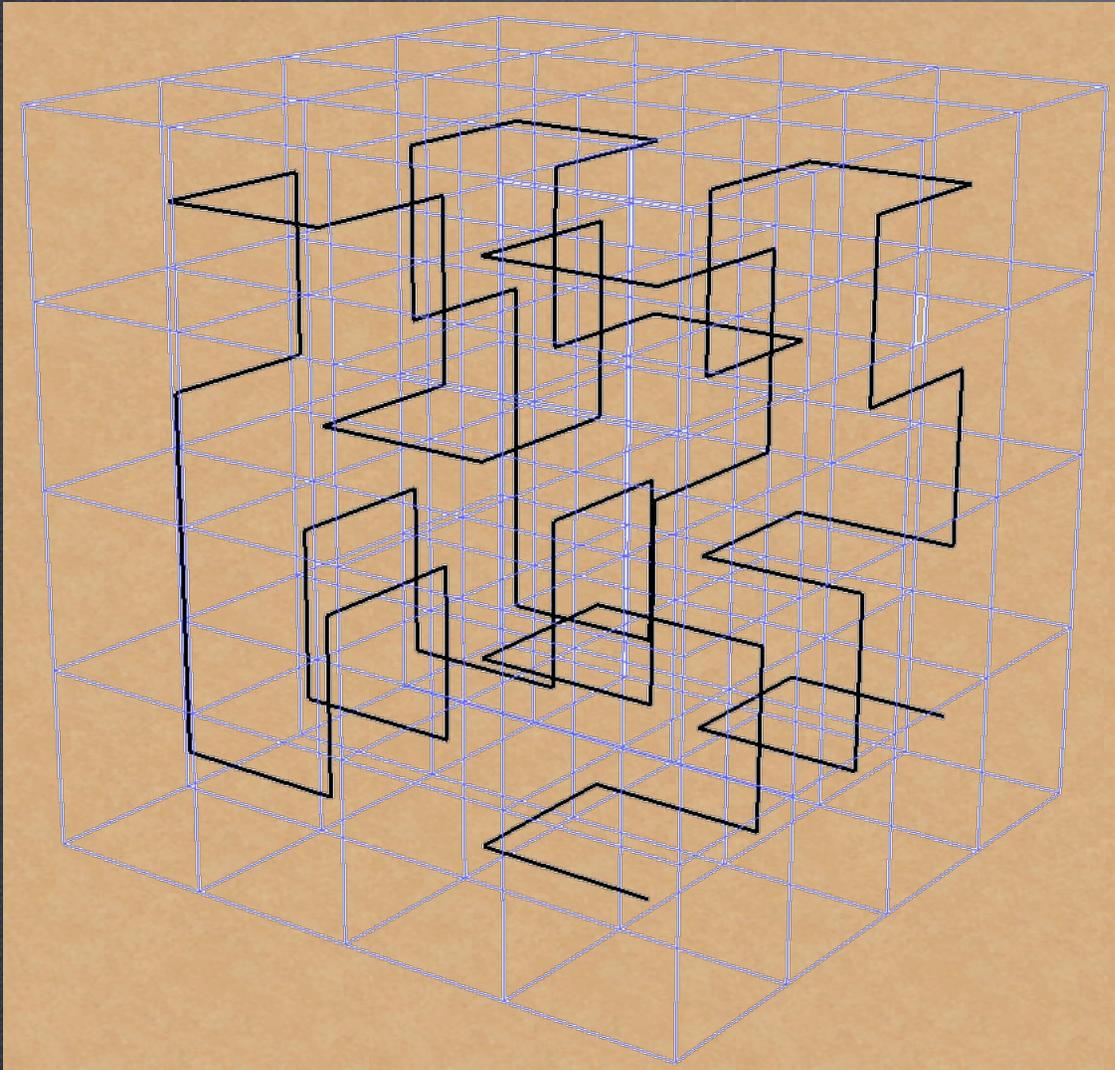


- Extend to 3D with additional U-shaped turns
- Basic building block is $2 \times 2 \times 2$



Space-Filling-Curves

Peano-Hilbert and Morton Ordering in 3D



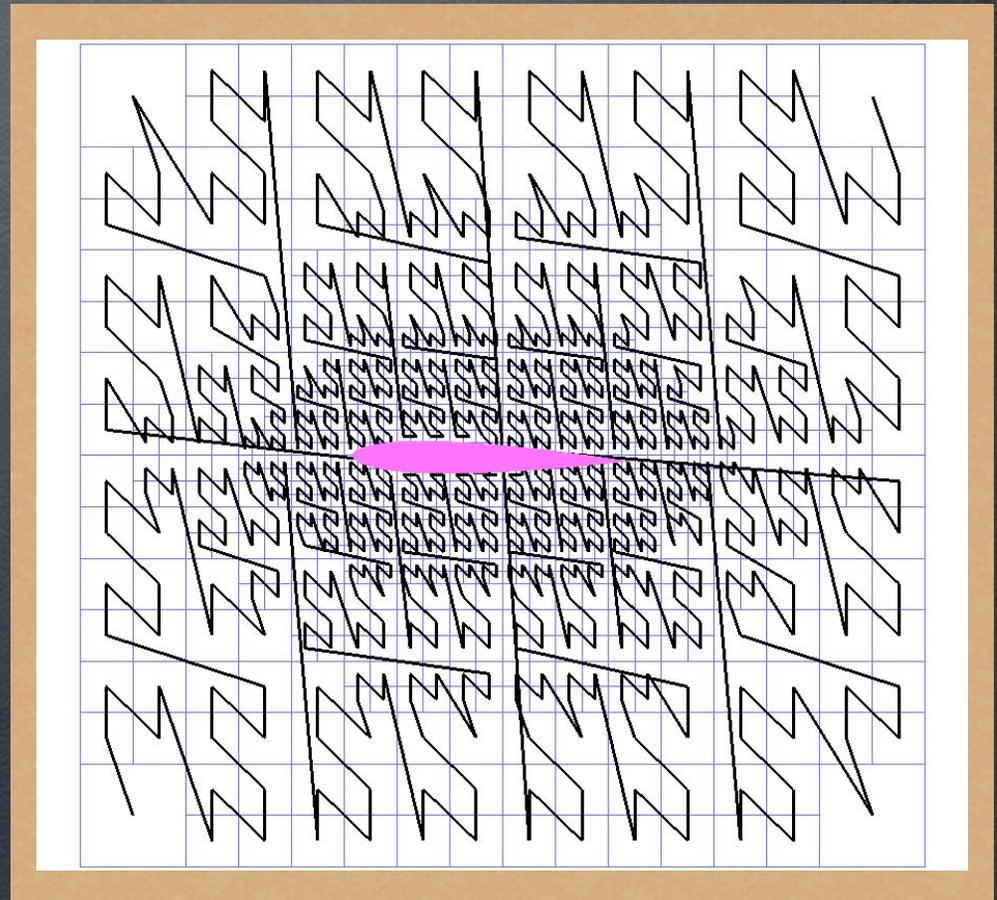
- Extend to 3D with additional U-shaped turns
- Basic building block is $2 \times 2 \times 2$
- Refine by replacing segments with basic building block
- At high enough resolution each voxel is visited by curve



Space-Filling-Curves

Ordering adaptively refined meshes

- Finest cell in mesh defines the dimension of voxel to be visited by SFC.
 - Coarser cells are collections of voxels.
- Compute $M(i)$ or $H(i)$ in each cell and sort using SFC index as sorting key
- Runtime using quicksort is $O(N \log N)$
 - Approx: 4 sec./1M cells on 2Ghz Pentium 4.

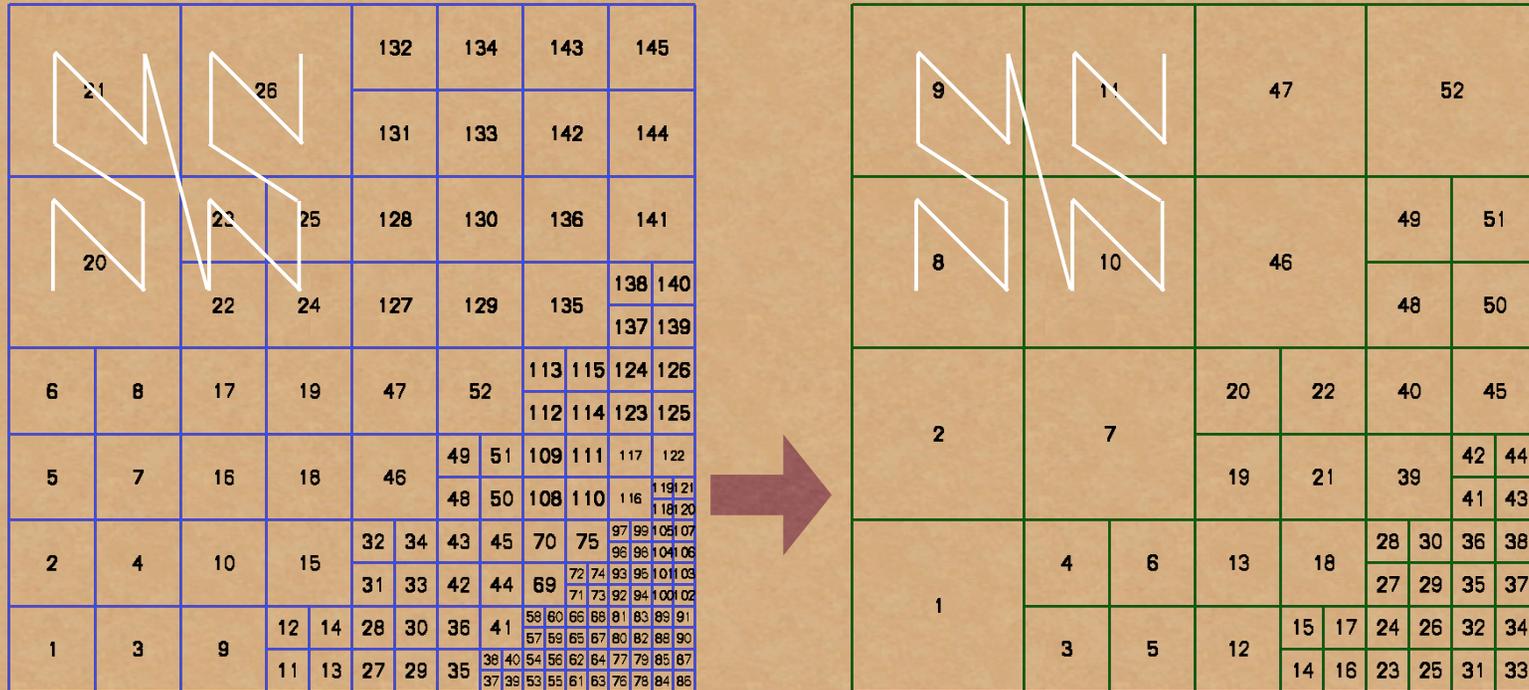


N-ordered adaptively refined mesh around NACA 0012



Mesh Coarsening

Simple traversal of cells in SFC order



- Since the curve is hierarchal by construction, simply traverse, collecting children as you go.
- Cells are siblings if $M(i)$ $H(i)$ of parent are the same



Mesh Coarsening

Fine Mesh

21		26		132	134	143	145											
				131	133	142	144											
20		23	25	128	130	136	141											
		22	24	127	129	135	138	140										
6	8	17	19	47	52	113	115	124	126									
						112	114	123	125									
5	7	16	18	46	49	51	109	111	117	122								
					48	50	108	110	116	119	121	120						
2	4	10	15	32	34	43	45	70	75	97	99	105	107					
				31	33	42	44	69	72	74	93	95	101	103				
1	3	9	12	14	28	30	36	41	58	60	66	68	81	83	89	91		
			11	13	27	29	35	38	40	54	56	62	64	77	79	85	87	
									57	59	65	67	80	82	88	90		
									37	39	53	55	61	63	76	78	84	86

Subsequent coarse grids automatically generated in SFC order!



Mesh Coarsening

Fine Mesh

21		26		132	134	143	145															
				131	133	142	144															
20		23	25	128	130	136	141															
		22	24	127	129	135	138	140														
						137	139															
6	8	17	19	47	52	113	115	124	126													
						112	114	123	125													
5	7	16	18	46	49	51	109	111	117	122												
					48	50	108	110	116	119	121	123										
2	4	10	15	32	34	43	45	70	75	97	99	109	107									
				31	33	42	44	69	72	74	93	95	101	103	96	98	104	106				
1	3	9	12	14	28	30	36	41	58	60	66	68	81	83	89	91						
			11	13	27	29	35	38	40	54	56	62	64	77	79	85	87	57	59	65	67	80
										97	99	109	107	96	98	104	106					
										71	73	92	94	100	102							
										53	55	61	63	76	78	84	86					

First Coarsening

9		11		47		52							
8		10		46		49	51						
						48	50						
2	7	20	22	40	45								
		19	21	39	42	44							
						41	43						
1	4	6	13	18	28	30	36	38					
	3	5	12	15	17	24	26	32	34				
						14	16	23	25	31	33		

Subsequent coarse grids automatically generated in SFC order!



Mesh Coarsening

Fine Mesh

21		26		132	134	143	145		
				131	133	142	144		
20		23	25	128	130	136	141		
		22	24	127	129	135	138	140	
						137	139		
6	8	17	19	47	52	113	115	124	126
						112	114	123	125
5	7	16	18	46	49	51	109	111	117
						119	121		
						48	50	108	110
						116			118
2	4	10	15	32	34	43	45	70	75
						97	99	108	107
						96	98	104	106
						31	33	42	44
						69	72	74	93
						71	73	92	94
						100	102		
1	3	9	12	14	28	30	36	41	58
						59	60	66	68
						61	63	81	83
						67	69	80	82
						88	90		
						38	40	54	56
						62	64	77	79
						85	87		
						97	99	53	55
						61	63	76	78
						84	86		

First Coarsening

9		11		47		52			
8		10		46		49	51		
						48	50		
2		7		20	22	40	45		
				19	21	39	42	44	
						41		43	
1		4	6	13	18	28	30	36	38
						27	29	35	37
		3	5	12	15	17	24	26	32
						14	16	23	25
						31	33		

Second Coarsening

5		20		22					
		19		21					
2		4		10		16	18		
						15	17		
1		3		7	9	12	14		
				6	8	11	13		

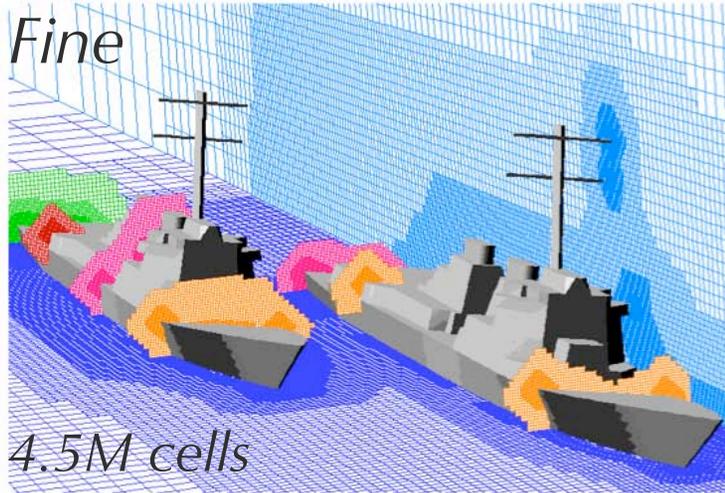
Subsequent coarse grids automatically generated in SFC order!



Mesh Coarsening

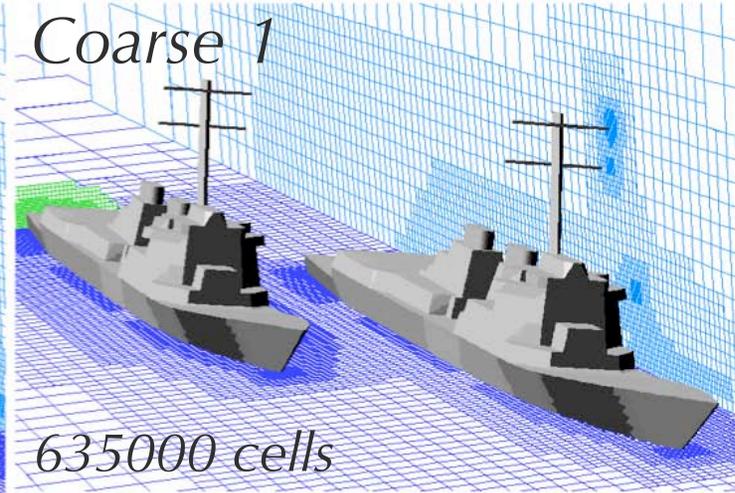
3D mesh coarsening example

Fine



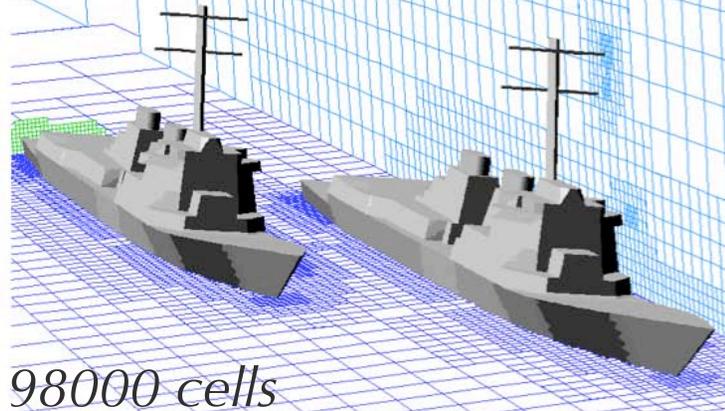
4.5M cells

Coarse 1



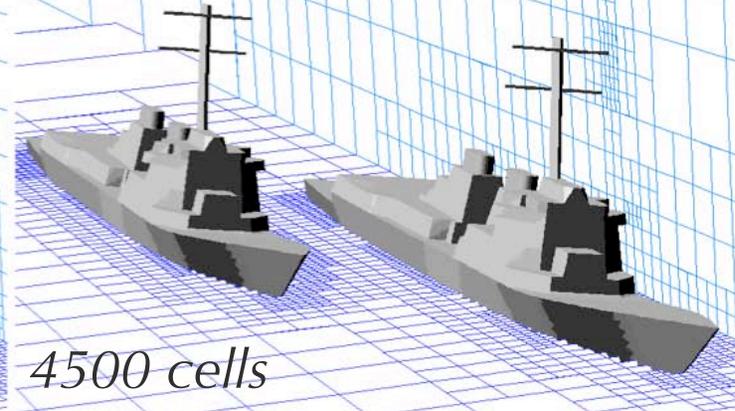
635000 cells

Coarse 2



98000 cells

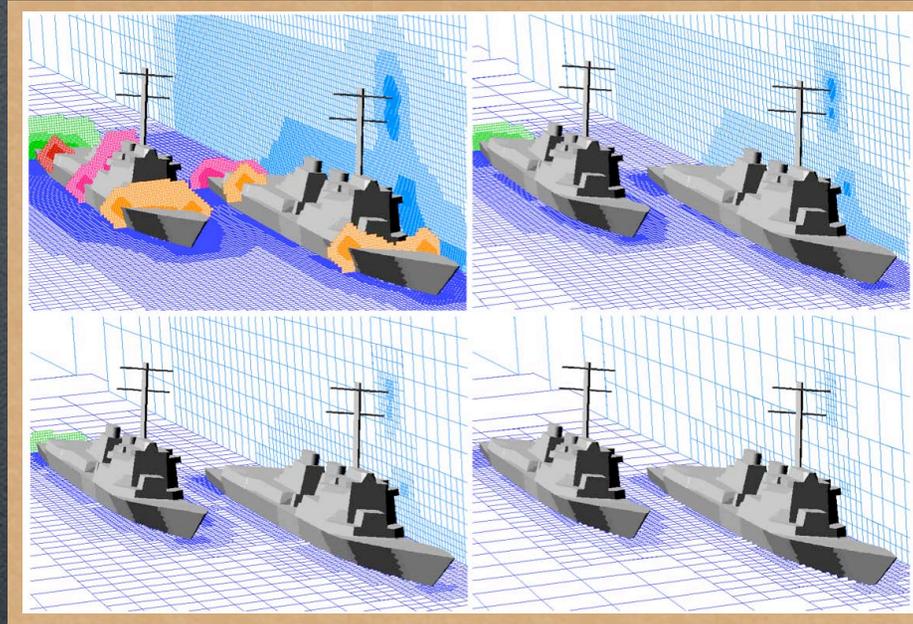
Coarse 4



4500 cells



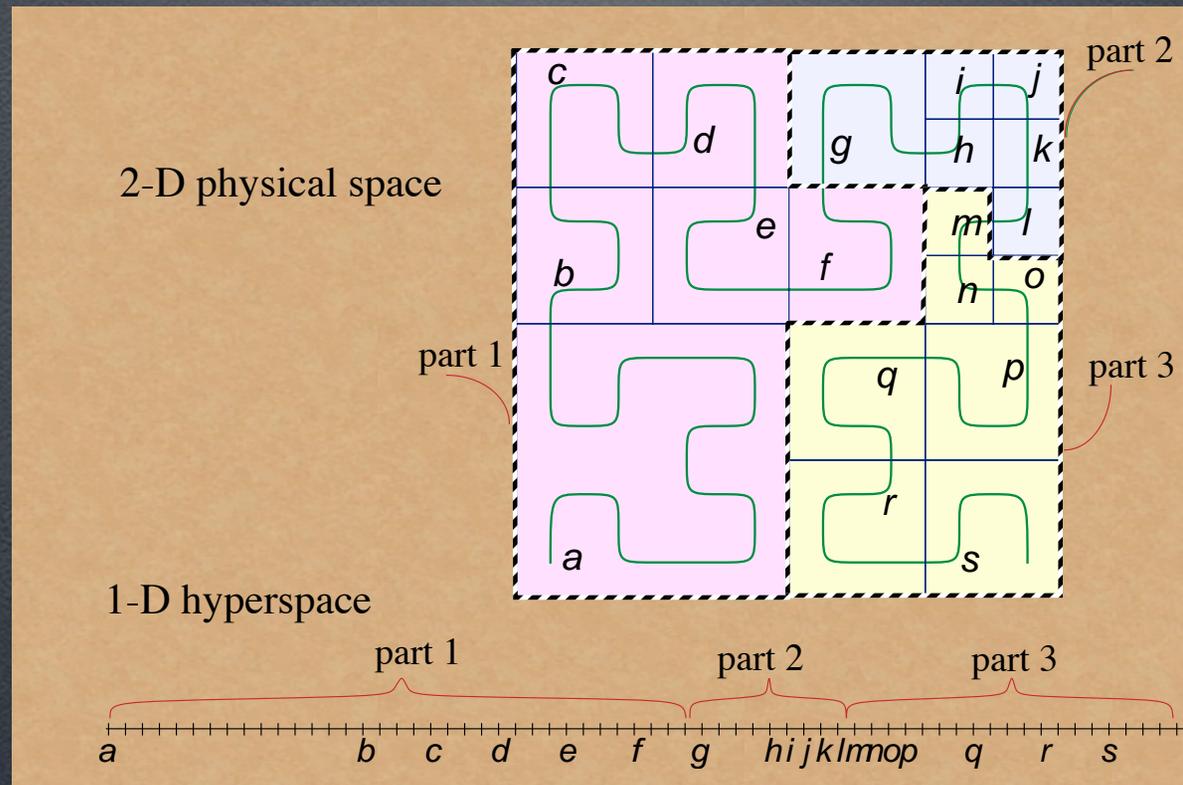
Mesh Coarsening



- Perfect coarsening is 8:1
 - In practice, ratios near 7 on examples with real geometry
 - Coarse meshes produced in SFC order automatically - coarsen again
- Coarsening alg. has linear-complexity from SFC ordered mesh
 - One sweep to coarsen cells
 - One sweep for 1-irregularity rule
 - This example takes under 15 sec. on 2Ghz Pentium 4.



Domain Decomposition

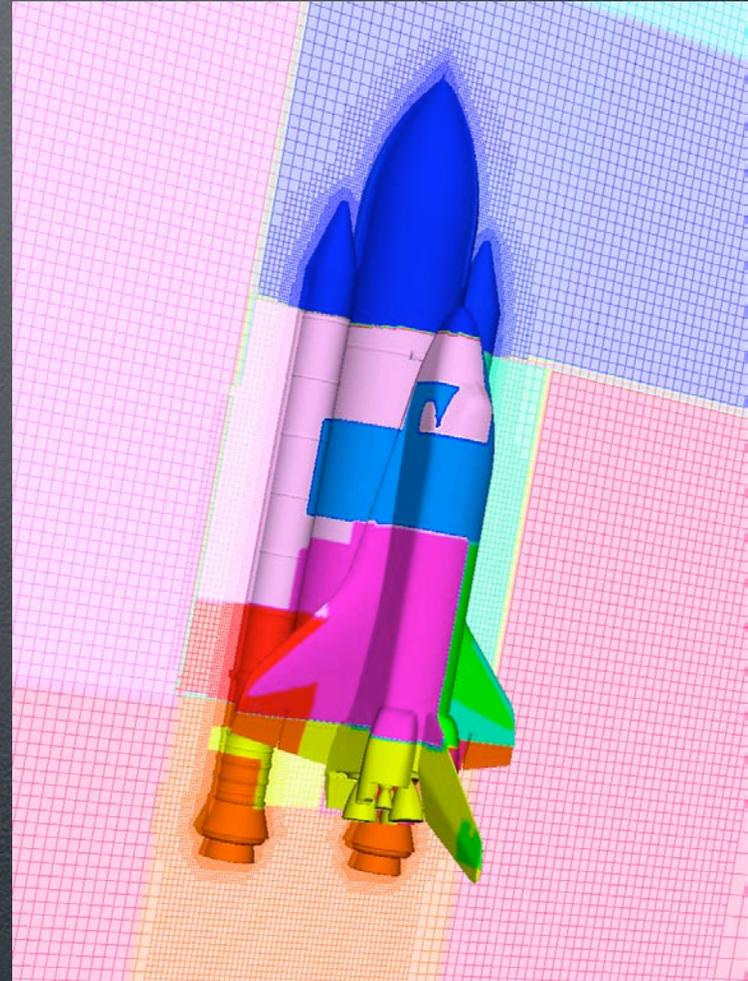


- Place partition boundaries in 1-D hyperspace of U
- Hierarchical structure of SFC gives partition quality competitive with other popular partitioners
- Simple scan of U -ordered mesh – permits variable work-per-cell



Domain Decomposition

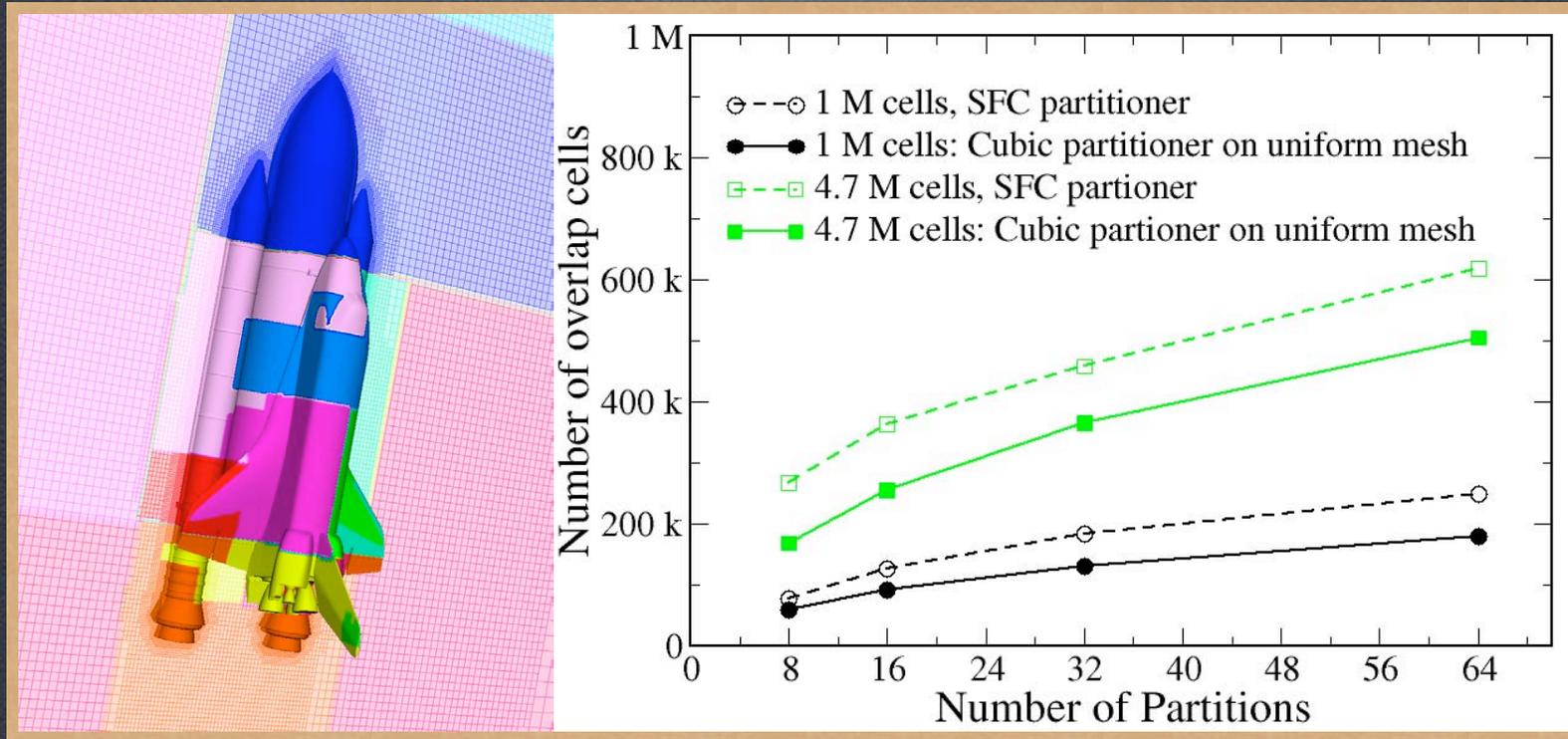
- SSLV with 4.7 M cells, cut-cells weighted 2.1x for load-balance
- Partitions largely rectilinear
- Partition on-the-fly from single U -ordered mesh on disk
 - Runtime partitioning
 - Restart on different # of CPUs
- Coarse grids partition with same SFC
 - Also on-the-fly
 - Load-balance each independently
 - Good overlap b/c same SFC





Domain Decomposition

Partition Quality

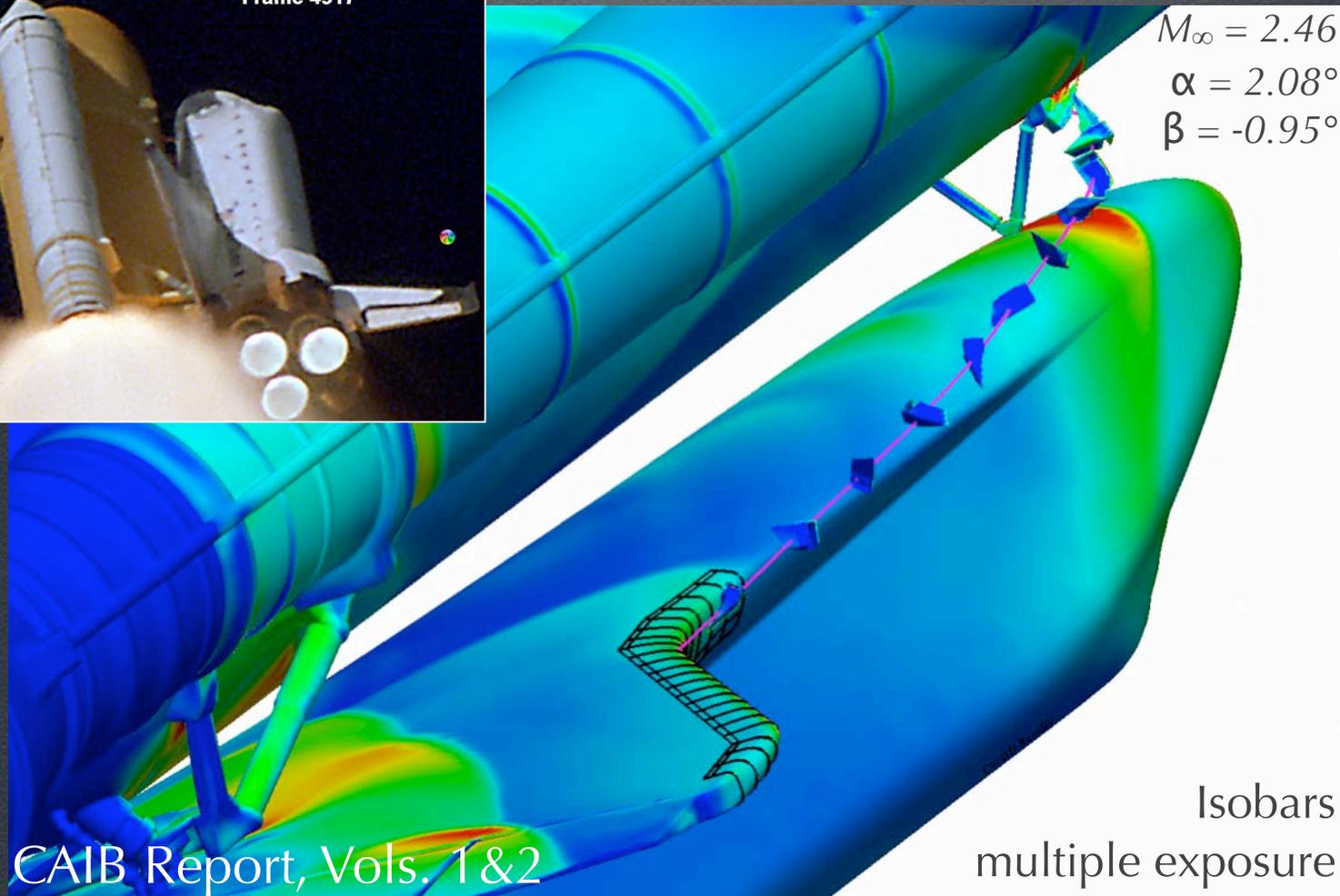
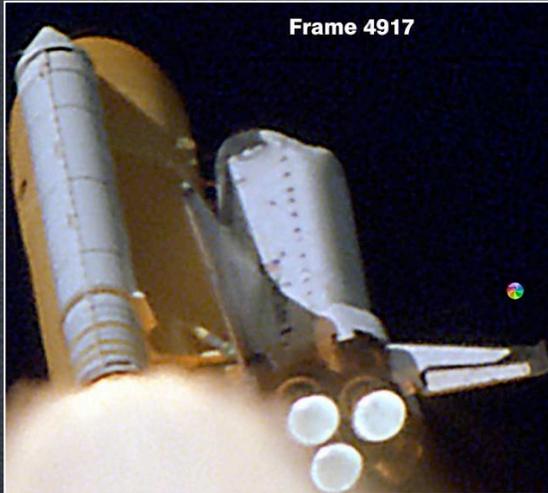


- Compare communication with idealized cubic partitioner over range of CPUs
- Surface-to-volume tracks idealized behavior
- Partitioning minimizes communication enough to give near ideal parallel scalability



Domain Decomposition

STS-107 Ascent debris, moving body 6-DOF, with adaptive mesh

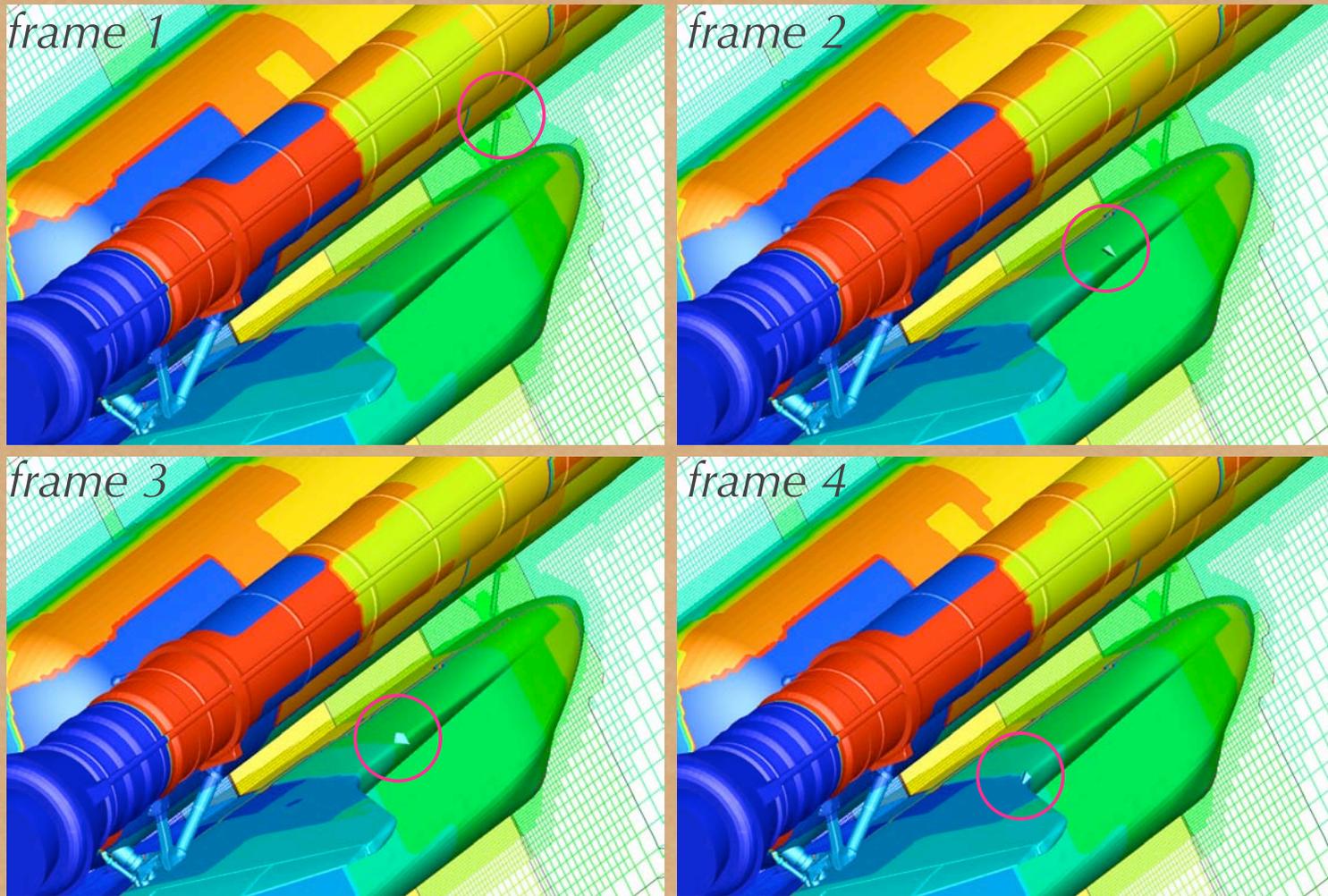


CAIB Report, Vols. 1&2



Domain Decomposition

STS-107 Ascent debris, mesh partitioning

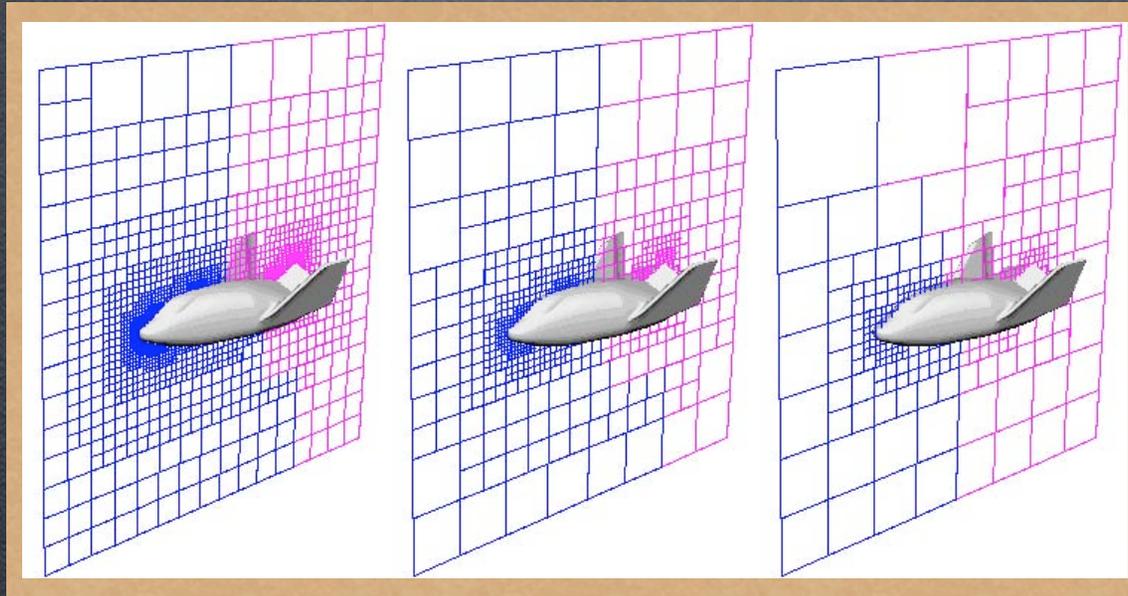


Load balanced partitioning almost unchanged despite mesh adaptation to motion.



Domain Decomposition

Subdomain overlap in multigrid hierarchy



- Intergrid transfer operators in multigrid introduce comm. between *every cell* in hierarchy
- Good overlap between corresponding subdomains on coarse and fine meshes minimizes off-processor bandwidth requirements
- In this example, only 4% of fine cells restrict to a different partition, tabulated results in *AIAA 2004-1232*



Conversion to MPI

- Parallel regions:
 - OpenMP - `#pragma omp parallel`
 - MPI - all CPUs move through identical code, insert `MPI_Barrier()` where OpenMP threads fork.
- Exchange Routines:
 - OpenMP - structure copy using shared memory
 - MPI - `pack-send-receive-unpack` exchange buffers
- Reductions:
 - Different implementations in OpenMP or MPI.
- Multigrid transfer across partitions
 - OpenMP - do directly with shared memory
 - MPI - explicit transfer if restrict/prolong off subdomain
- I/O and initial problem layout different



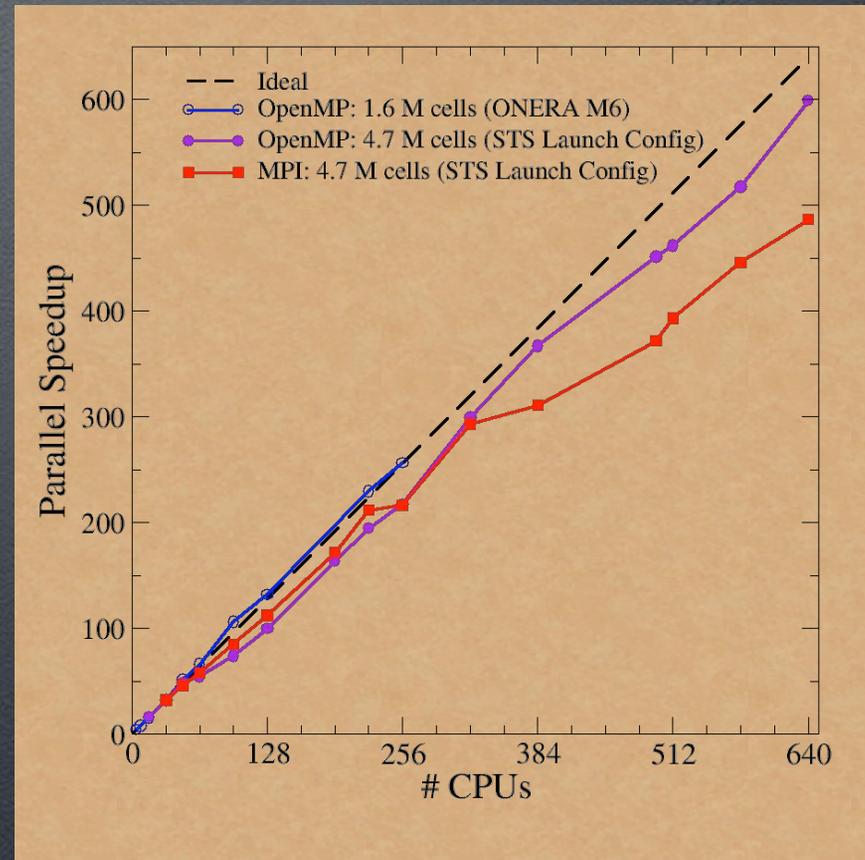
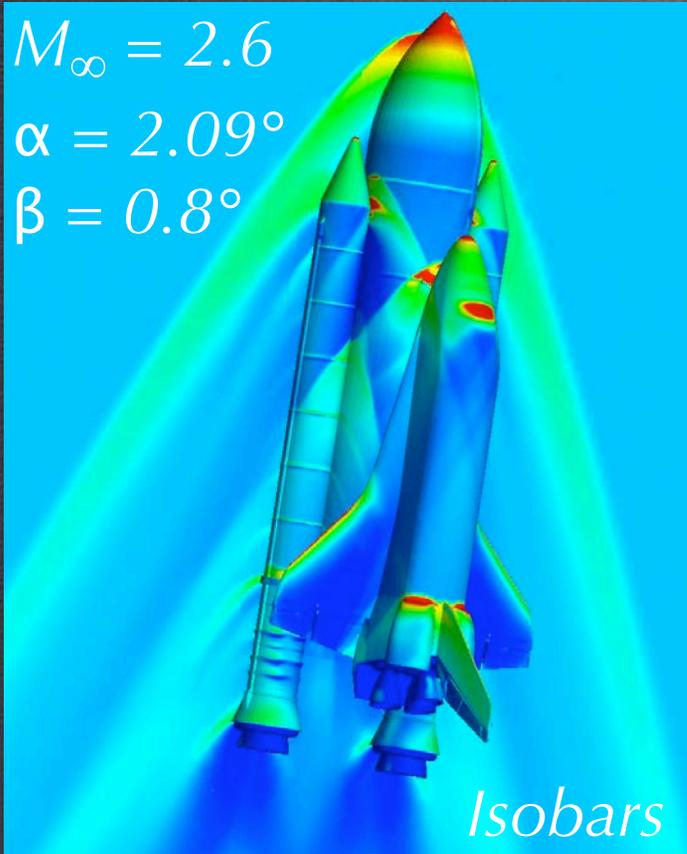
Scalability and Performance

Single grid, 4.7M cell mesh SSLV example – Origin 3000

$$M_\infty = 2.6$$

$$\alpha = 2.09^\circ$$

$$\beta = 0.8^\circ$$



- Origin 3000: Speedup of 599 (OpenMP) and 486 (MPI) on 640 CPUs
- 4.7M cell mesh has only ~7350 cells on each partition



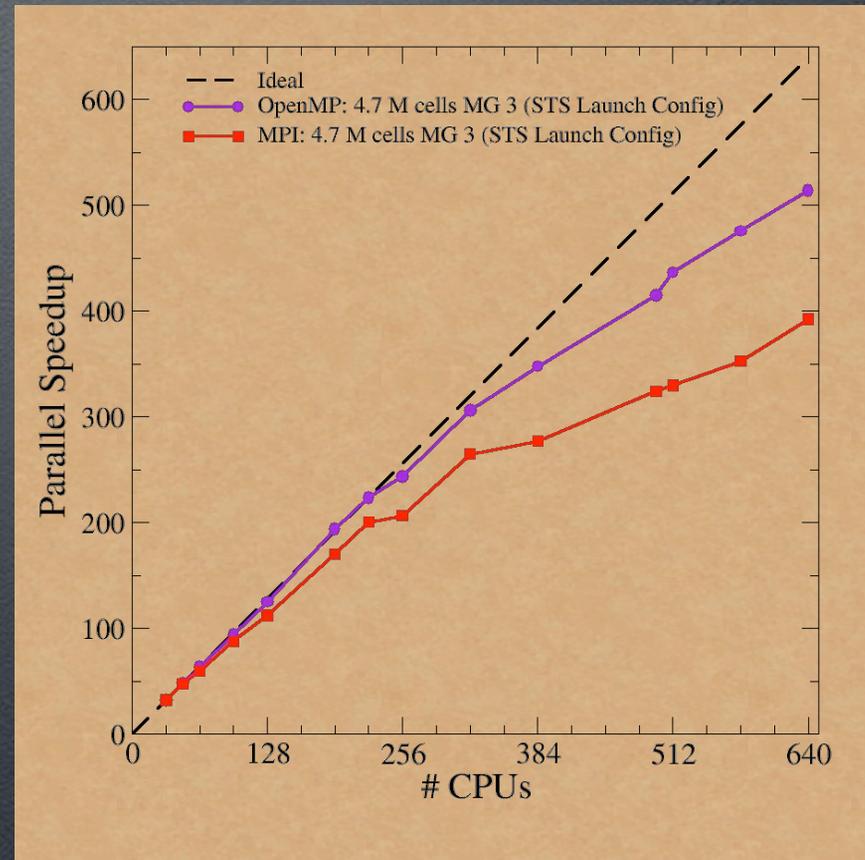
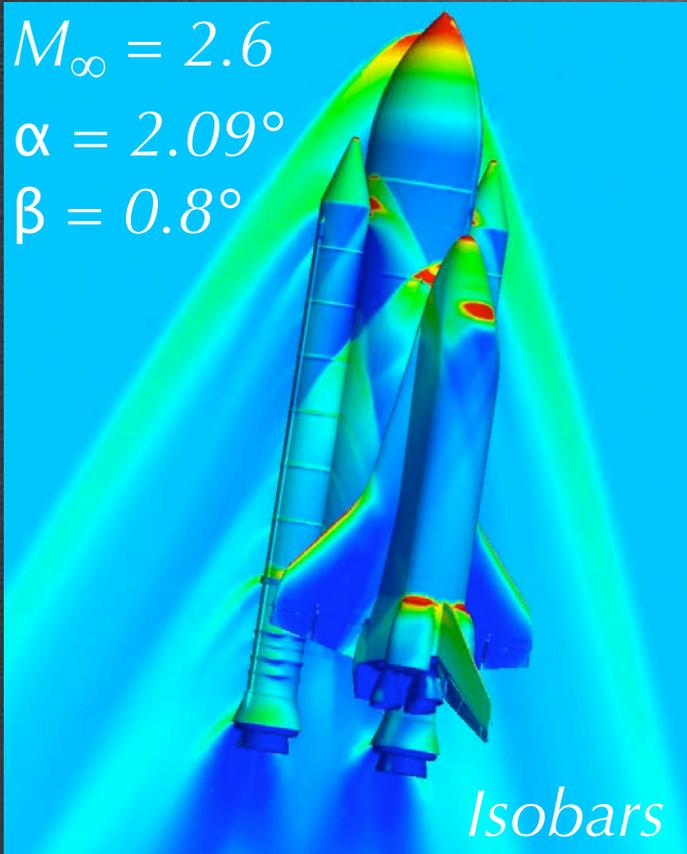
Scalability and Performance

Multigrid, 4.7M cell mesh SSLV example – Origin 3000

$$M_\infty = 2.6$$

$$\alpha = 2.09^\circ$$

$$\beta = 0.8^\circ$$



- Origin 3000: Speedup of 512 (OpenMP) and 392 (MPI) on 640 CPUs
- First and second coarse meshes have only 1100 and 180 cells/cpu



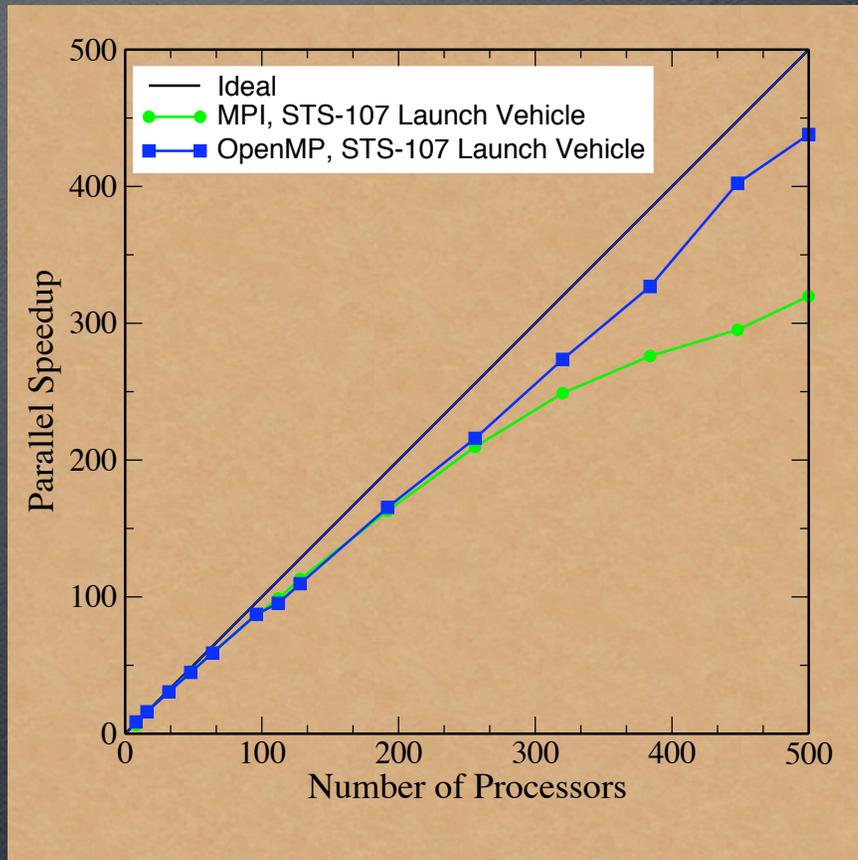
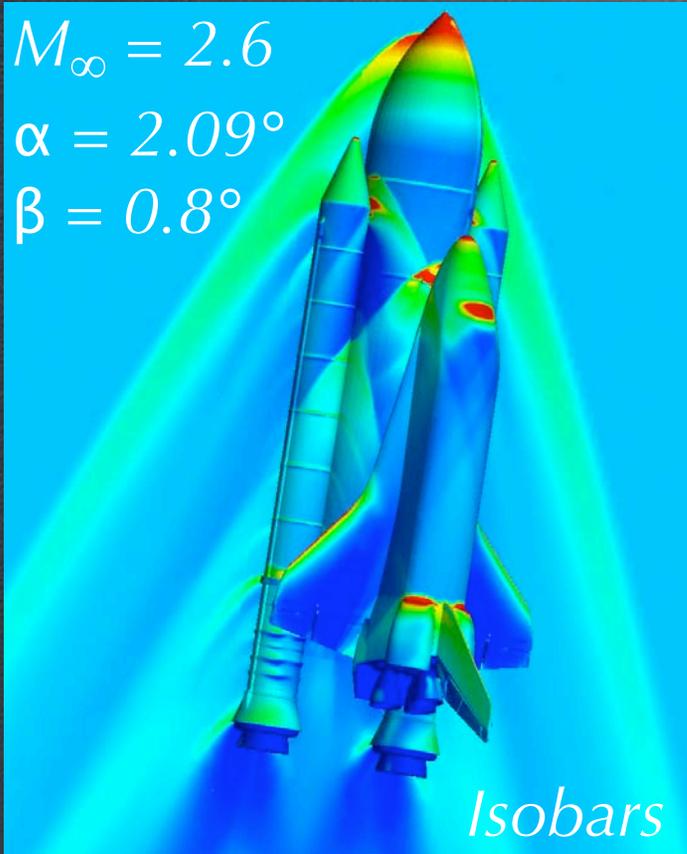
Scalability and Performance

Single Grid, 4.7M cell mesh SSLV example – SGI Altix 3000

$$M_{\infty} = 2.6$$

$$\alpha = 2.09^{\circ}$$

$$\beta = 0.8^{\circ}$$

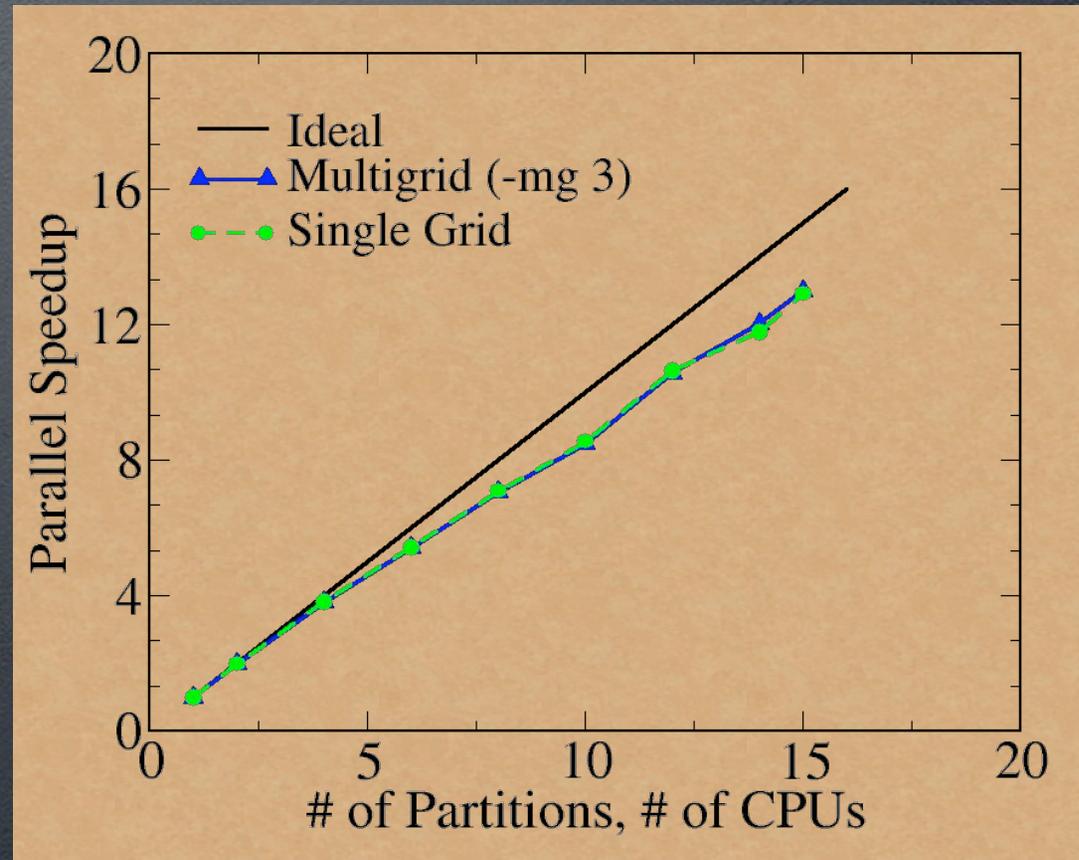
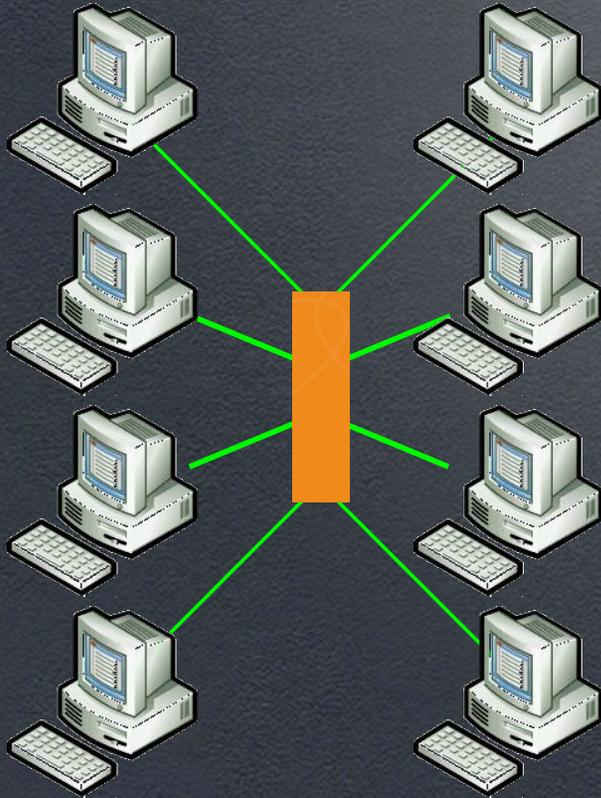


- Altix 3000: Speedup of 440 (OpenMP) and 320(MPI) on 512 CPUs
- Each Altix CPU (1.5Ghz Itanium 2) is ~3x faster than Origin
- Machine is still new, so further gains expected (MPI especially)



Scalability and Performance

Cluster performance, 16 2.4Ghz Pentium 4, with Gig-E switch



- Cluster: Speedup of 13 on 15 CPUs with simple Gig-E unmanaged switch interconnect.
- Each CPU is about 3-4x faster than an Origin CPU



Summary & Future Work

- Very good scalability on wide range of platforms 500-600/640 on Origin 3000, 320-440/512 on SGI Altix, 13/15 on Pentium 4 cluster.
 - Architect code for distributed memory, but use direct access in OpenMP.
 - *Restrict* reliance on OpenMP constructs
 - *Isolate* non-local memory references
- Relatively painless port to MPI
 - Common code base (instruction path ~15% different)
 - Rewrite MPI backends for exchange, setup and I/O
 - Additional work for pack-send-receive-unpack results in scalability almost as good as OpenMP
- Actively seeking DVSM to evaluate OpenMP code on clusters