Programming for Modern Architectures in the CHICOMA Hydrocode

LA-UR-13-26808

Multimat 2013

JG Wohlbier[a], LD Risinger[a], TR Canfield[b], MR Charest[c], NR Morgan[c], JI Waltz[c]

Los Alamos National Laboratory

[a] Computer, Computational, and Statistical Sciences Division

[b] Theoretical Division

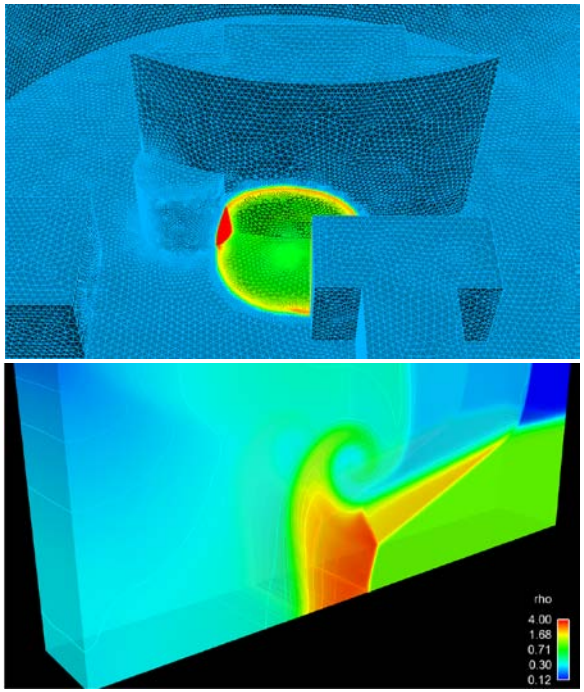
[c] X-Computational Physics Division

Outline

- CHICOMA overview
- Hardware considerations
- Programming models
- Algorithms
- Results
- Conclusions

CHICOMA, a 3D unstructured mesh code

- **CHICOMA supports Eulerian, Lagrangian, and ALE hydro on unstructured tetrahedral meshes with AMR**
  - The Eulerian-AMR algorithm is substantially verified
  - The Lagrangian algorithm is tentatively verified
  - ALE + multi-materials are in progress
- **Multiple strategies for advanced architectures are under investigation**
  - Graph theoretic optimization
  - High-order methods
  - Heterogeneous work sharing models

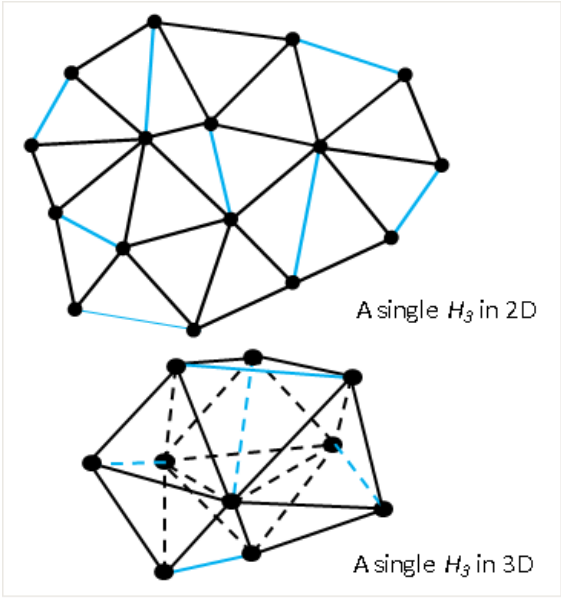


Our goal is to produce innovative, high-performance 3D multi-physics software applicable to a range of scientific problems.

CHICOMA

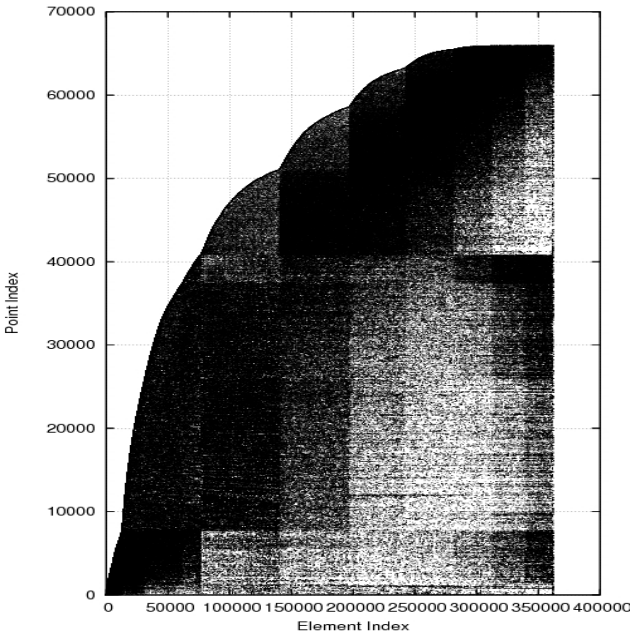
- **CHICOMA has been designed for high performance on homogeneous multi-core architectures**
  - OpenMP + compiler vectorization
  - Edge coloring to avoid write contention on threaded edge loops with point updates
  - Wavefront application to improve data locality

Resolution (μm)	OpenMP Threads	Tetrahedra	Runtime (min)	Cycles	Grind Time (μs/element/cycle)
240	4	362363	1.5	464	2.14
120	8	2898904	19	1294	2.43
60	12	23191232	404	3998	3.14

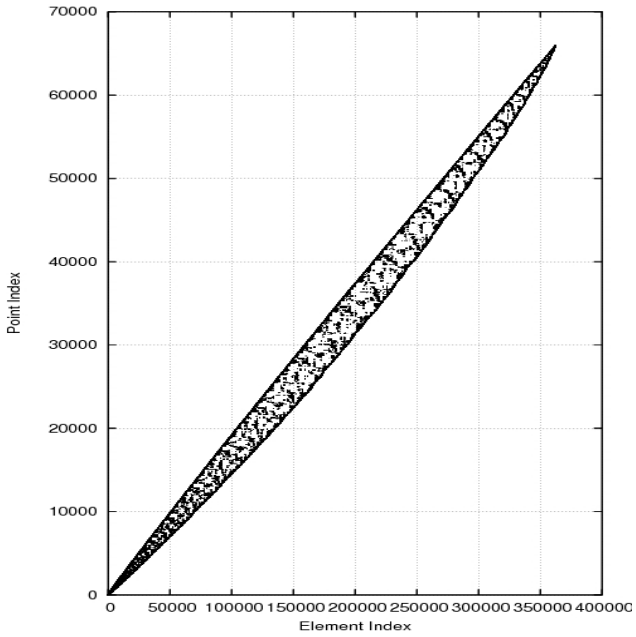


CHICOMA

- **Net effect of renumbering with wavefront reduces "bandwidth" by O(100) and smooths memory access patterns**
- **Example: Point-element connectivity for a Sedov mesh with 363k tetrahedra**



▪Original mesh: ϕ = 65894



▪Optimized mesh: ϕ = 6296

Hardware Considerations

- **Three chips of interest**
  - "Traditional" : Intel Xeon E5-2650
  - "Future" : Nvidia C2075 GPU
  - "Future" : Intel MIC, Xeon Phi 5110p

Processor	Cores / threads	SIMD width (bits)	GF/s (double)	GF/s (single)	Memory Bandwidth (GB/s)	Thermal Design Power (W)	Cost ($)
Xeon E5-2650	8/16	256	128	256	51	95	1100
Nvidia Processor **C2075**	448	NA	515	1030	144	225	1800
Xeon Phi 5110p	60/240	512	1011	2022	320	225	2600

Hardware Considerations

- **Why do we care?**
  - HPC Industry trend is increased Flops/Watt
  - Cannot make an exascale machine by bolting together Xeons

Processor	GF/s (double)	Thermal Design Power (W)	Cost ($)	Flops/Watt (GF/W)	Flops/Cost (GF/$)	CHICOMA wall time (s)	Kernel wall time (s)
Xeon E5-2650	128	95	1100	**1.4**	**0.12**		
Nvidia Processor **C2075**	515	225	1800	**2.3**	**0.29**		
Xeon Phi 5110p	1011	225	2600	**4.5**	**0.39**		

Programming models

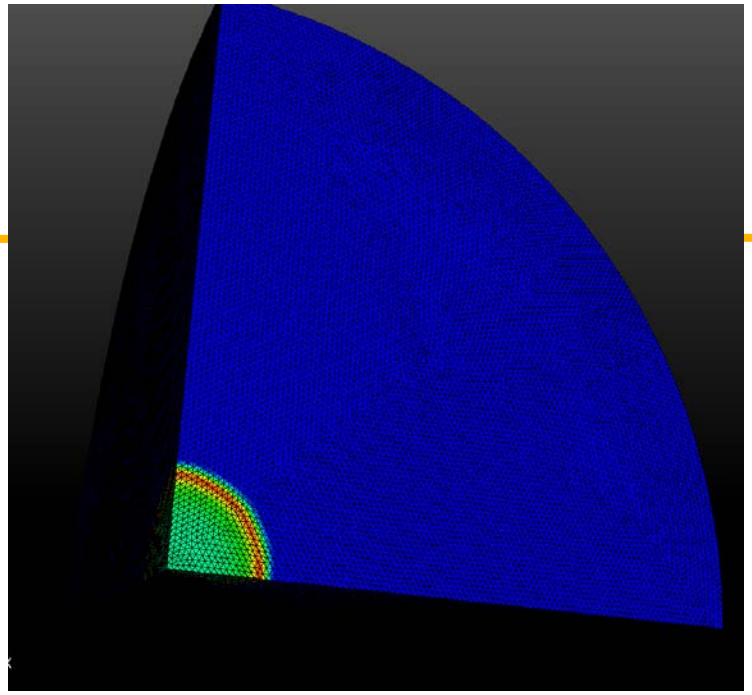
- **We are using multiple programming models**
  - OpenMP with compiler vectorization (CPU + MIC)
  - CUDA (Nvidia GPU)
  - OpenCL (CPU's, GPU's, MIC, FPGA's)
- **CHICOMA was developed primarily with OpenMP + compiler vectorization**
  - Many of the algorithms work very well, e.g., color groups, multi-level OpenMP parallelism
- **GPU computing on unstructured meshes is challenging**
  - GPU's are often "all or none"
  - Increased thread level parallelism does not necessarily map nicely onto data structures that work well for OpenMP + vectorization
- **Language portability**
  - CUDA for Nvidia GPU's only
  - OpenCL promises hardware portability but is very low level
    - SDK's for CPU's and MIC do not support 64 bit atomics

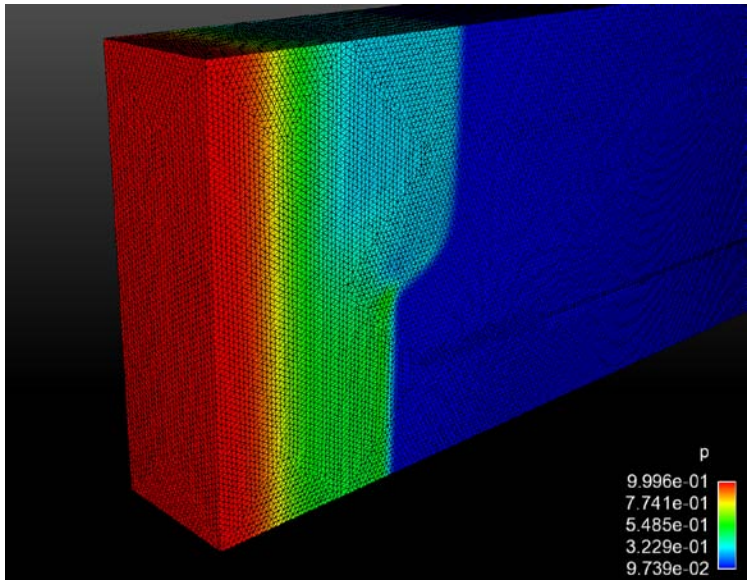
Algorithms Studied

- **Gradient**
  - Loop over edges, gather/scatter to points
  - Very few flops per memory access
- **Muscl solver**
  - Edge loop getting gradients from points and storing in edge array
  - More flops per memory access, but still not huge
- **Riemann solver**
  - Edge loop on muscl unknowns scattered back to points

Results

- **Sedov : 100 cycles on mesh with**
  - Npoint = 505,724
  - Nelement = 2,898,904
  - Nboundary = 45,786
  - Nedge = 3,450,411
  - Nface = 5,843,592
- **Triple Point : 100 cycles on mesh with**
  - Npoint = 347,811
  - Nelement = 1,960,914
  - Nboundary = 45,073
  - Nedge = 2,353,795
  - Nface = 3,966,899





Results: Xeon E5-2650

- **“Large edge color groups” used to avoid write contention on points**
- **“Small color group option” (not shown) gets 7-8x scaling, but not thread safe for small number of zones per thread**
  - Changed to large color groups for thread safety on Intel MIC
- **Large color groups to be replaced by point based method which will recover 7-8x scaling**

Sedov 2.9M elements		
Threads	CHICOMA wall time (s)	Kernel wall time (s)
1	520	426
8	111	83
	4.7x	5.1x

Triple Point 1.9M elements		
Threads	CHICOMA wall time (s)	Kernel wall time (s)
1	367	252
8	87	49
	4.2x	5.1x

Results: Nvidia C2075

- **Started with data layout coming from edge coloring and used atomic operations to handle write contention**
- **GPU kernels invert loop such that unknowns updated for large groups of edges rather than updating all unknowns per edge**
- **Discovered access to global memory was VERY slow**
  - Host side data layout was all unknowns per edge (column major) rather than all edges per unknown (row major)
  - Indirect addressing into points had far from ideal data locality
- **Skipped coloring algorithm on the host to get ideal data locality for points**
  - Branching due to atomic operations became unbearable due to good locality
  - “Whack a mole”
- **Switched to point based algorithm with host memory layout to maximize global memory access**
  - Results in branch divergence on variable numbers of edges per point
  - Best performance so far

Results: Nvidia C2075

- **With the changes above the speed improved substantially**
  - Sedov before: 289s
  - Sedov after: 187s
- **Not reporting OpenCL numbers**
  - OpenCL implementation does not have optimizations done in CUDA
  - … and is presently broken

Before: Sedov 2.9M elements		
Threads	CHICOMA wall time (s)	Kernel wall time (s)
1	289	204

Before: Triple Point 1.9M elements		
Threads	CHICOMA wall time (s)	Kernel wall time (s)
1	271	137

After: Sedov 2.9M elements		
Threads	CHICOMA wall time (s)	Kernel wall time (s)
1	187	109

After: Triple Point 1.9M elements		
Threads	CHICOMA wall time (s)	Kernel wall time (s)
1	207	76

Results: Xeon Phi 5110p

- **Run in "native" mode**
  - Other modes available: "offload mode," OpenCL
- **OpenMP implementation for native mode "just works" via cross compile**
  - If you have a well optimized OpenMP code the MIC won't be a complete disaster

Sedov 2.9M elements		
Threads / threads per core	**CHICOMA wall time (s)**	**Kernel wall time (s)**
30		
60/1		
120/2		
240/4		

Triple Point 1.9M elements		
Threads / threads per core	**CHICOMA wall time (s)**	**Kernel wall time (s)**
30	409	222
60/1	308	145
120/2	325	131
240/4	481	208

Results: Xeon Phi 5110p

- **Have not done detailed analysis for MIC**
- **Estimate parallel fraction from Amdahl's law**
  - CHICOMA ~ 93%
  - Extrapolate to N = 240 for P = 0.93
    - **S ~ 13.5**
  - Increasing P is essential
    - If P = 0.99 and can scale on MIC, S ~ 71
- **Amdahl's Law**
  - S = speed up
  - P = parallel fraction
  - N = number of parallel units

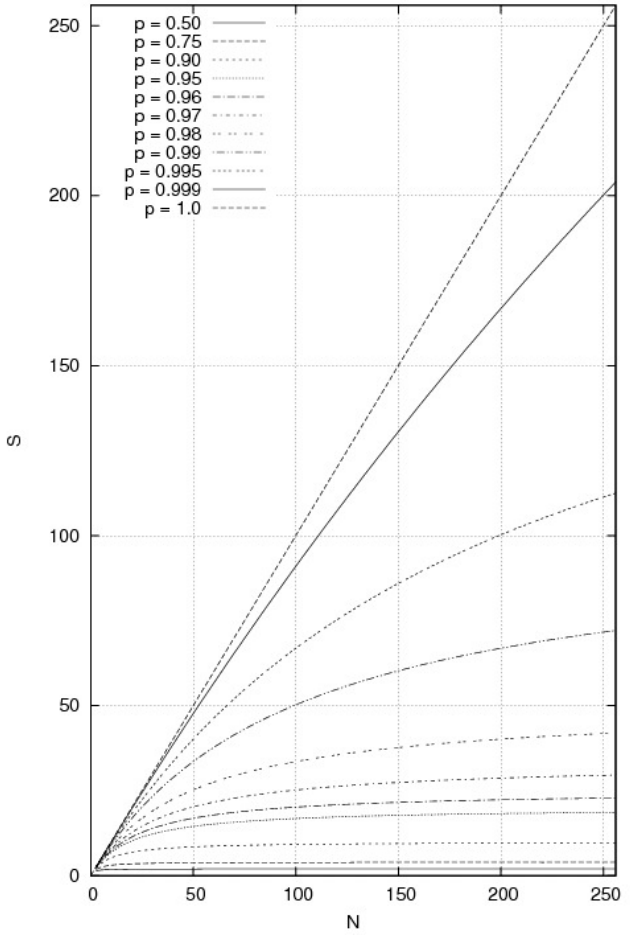
N	time (s)	S	P_{est} (%)
1	**857**	**1**	**NA**
2	459	1.87	92.9
4	267	3.21	91.8
8	161	5.32	92.8
16	106	8.08	93.5
32	76	11.3	94.1
60	68	12.6	93.6
120	69	12.4	92.7

$$S = \frac{1}{1 - P\left(1 - \frac{1}{N}\right)} \qquad P = \left(1 - \frac{1}{S}\right)\left(1 - \frac{1}{N}\right)^{-1}$$

Results: Xeon Phi 5110p

- **Previously saw for OpenMP on "traditional" chips**
  - P ~ 0.98
- **Increasing P is essential once past N = O(10)**
- **Change to point based loops from edges with coloring may increase P due to fewer OpenMP barriers**

$$S = \frac{1}{1 - P\left(1 - \frac{1}{N}\right)}$$



Results

- **We have work to do for "future architectures"**
- **Wall time increases with Flops/Watt**
  - Want opposite trend!

Sedov 2.9M elements							
Processor	**GF/s (double)**	**Thermal Design Power (W)**	**Cost ($)**	**Flops/Watt (GF/W)**	**Flops/Cost (GF/$)**	**CHICOMA wall time (s)**	**Kernel wall time (s)**
Xeon E5-2650 (8 cores)	128	95	1100	**1.4**	**0.12**	**111**	**83**
Nvidia Processor **C2075 (1 host core)**	515	225	1800	**2.3**	**0.29**	**187**	**109**
Intel MIC pre-production	1011	225	2600	**4.5**	**0.39**	**NDA**	**NDA**

Results

- **We have work to do for "future architectures"**
- **Wall time increases with Flops/Watt**
  - Want opposite trend!

Triple Point 1.9M elements							
Processor	**GF/s (double)**	**Thermal Design Power (W)**	**Cost ($)**	**Flops/Watt (GF/W)**	**Flops/Cost (GF/$)**	**CHICOMA wall time (s)**	**Kernel wall time (s)**
Xeon E5-2650 (8 cores)	128	95	1100	**1.4**	**0.12**	**87**	**49**
Nvidia Processor **C2075 (1 host core)**	515	225	1800	**2.3**	**0.29**	**207**	**76**
Xeon Phi 5110p (60 cores)	1011	225	2600	**4.5**	**0.39**	**308**	**145**

Conclusions

- **CHICOMA performs well on current hardware due to built-in design for OpenMP threads**
  - Edge coloring to avoid point write contention
- **Hardware is trending toward higher Flops/Watt via more on chip parallelism**
  - GPU – O(10,000) parallel threads
  - MIC – O(1,000) parallelism via O(100) threads X O(10) SIMD vector widths
- **The status quo will not be viable in the future**
- **GPU implementations in CHICOMA progressing**
- **Intel MIC**
  - Nice because it works “out of the box” with OpenMP + vectorization model, and “only slows code down by a factor of 3.5x”
  - Further study required to determine performance bottlenecks
  - Will be studying ways to increase parallel fraction