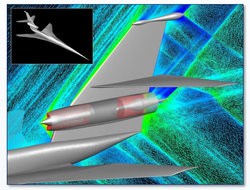
Unstructured-Grid CFD Algorithms on Many-Core Architectures

Aaron Walden,[1] Eric J. Nielsen,[1] Mohammad Zubair,[2] John C. Linford,[3] John G. Wohlbier,[4] Justin P. Luitjens,[5] Jason Orender,[2] Izaak Beekman,[3] Samuel Khuvis,[3] and Sameer S. Shende[3]

[1]NASA Langley Research Center, Hampton, VA 23681 USA [2]Old Dominion University, Norfolk, VA 23529 USA [3]ParaTools, Inc., Eugene, OR 97405 USA [4]HPCMP PETTT, Engility Corp., West Bethesda, MD 20817 [5]NVIDIA Corp., Salt Lake City, UT 84101

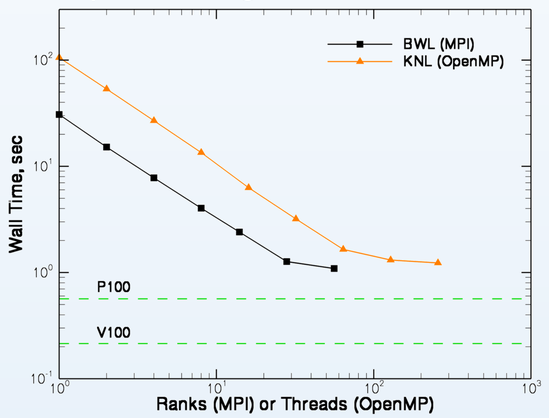
In the field of computational fluid dynamics, the Navier-Stokes equations are often solved using an unstructured-grid approach to accommodate geometric complexity. Furthermore, turbulent flows encountered in aerospace applications generally require highly anisotropic meshes, driving the need for implicit solution methodologies to efficiently solve the discrete equations. These approaches require frequent construction and solution of large, tightly-coupled systems of block-sparse linear equations.



Exascale simulations impose dramatically lower power and memory-per-core constraints. Legacy multi-core MPI codes may be rendered woefully inefficient by the coming paradigm shift. To prepare NASA Langley Research Center's FUN3D CFD solver for the future HPC landscape, we port two representative kernels to Intel Xeon Phi Knights Landing (KNL) and NVIDIA Pascal (P100) and Volta (V100) GPUs. We compare these shared-memory results with multi-core Intel Xeon Broadwell (BWL), where we exclusively use MPI for parallelism to represent conventional FUN3D.

Kernel 1: Compressible Viscous Flux Jacobians

Optimized Single Node Performance



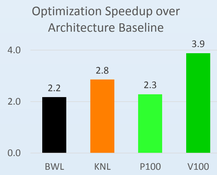
Hardware:
- **BWL**: Dual Socket Intel Xeon E5-2680v4 (28 cores)
- **KNL**: Intel Xeon Phi 7230 in flat quadrant mode
- **P100**: NVIDIA Pascal PCIE
- **V100**: NVIDIA Volta SXM

Speedup over BWL	
KNL	0.7
P100	1.5
V100	3.9

- Kernel 1 constructs majority of the matrix (CSR format) that is input to Kernel 2
- Physics kernel: Cell-based arithmetic with long dependency chains, weak vectorization potential, and highly irregular write pattern
- Race conditions for matrix updates if using shared memory (P/V100/KNL)
- GPUs have hardware-supported atomics, for KNL we use greedy cell coloring
- Coloring exacerbates the already irregular access pattern (>30% slower on KNL)
- MPI decomposition may imbalance workload over ranks (BWL)

BWL/KNL Optimization

Using conventional FUN3D Fortran as baseline:
- Factor algebra to avoid recomputation
- Store computed addresses in lookup table
- Hard-code loop extents
- Prefetch data to reduce memory latency

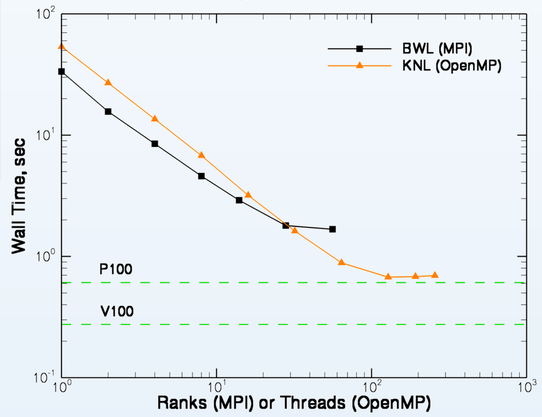


P100/V100 GPU Optimization

Using a CUDA C++ port of the BWL/KNL optimized Fortran as baseline:
- Use hardware-supported atomics for matrix updates
- Change parallelization from 1 thread per cell to X threads per cell
  - Use X threads across inner loops that are parallelizable
  - More active threads in memory-heavy portions of the code
  - Coalesces memory access patterns
  - Reduces register and shared memory pressure, increasing occupancy
- Collapse nested loops and parallelize across X threads
  - Improves thread utilization
  - Reduces communication through redundant computation
- Cache control state in shared memory
  - Add reductions across X threads in shared memory
- Refactor to reduce divergence
- Auto-tune block sizes (X,Y) and launch bounds

Kernel 2: Multicolor Point-Implicit Linear Solver

Optimized Single Node Performance



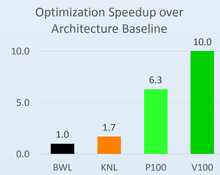
Input Grid:
- 5,634,274 grid points
- 14,718,107 tetrahedra
- 6,108,374 prisms
- 37,617 pyramids
- Mem footprint < 16 GB

Speedup over BWL	
KNL	2.5
P100	2.8
V100	6.1

- Kernel dominated by block-sparse matrix-vector product with low arithmetic intensity
- Matrix rows colored such that no adjacent grid points map to the same color
- Rows ordered in memory by color, contiguous access to most data
- Rows consist of some number of 5x5 dense blocks
- Input/output vectors indirectly addressed
- MPI communication of halo values between colors

KNL Optimization

Using conventional FUN3D Fortran as baseline:
- Rewrite kernel using AVX-512 vector intrinsics
- Vectorize over 3 5x5 blocks, 94% efficiency
- Prefetch next row's data into L2
- Prefetch current row's data into L1

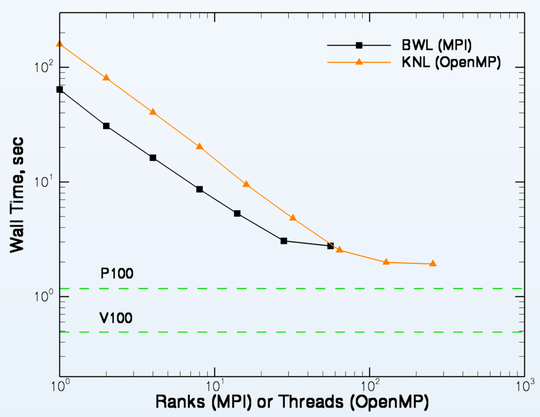


P100/V100 GPU Optimization

Using a *cuBLAS*/*cuSPARSE* library-based implementation as baseline:
- Rewrite custom CUDA C++ solver
- Process a 5x5 block with first 25 threads of a single warp of X threads
- Process rows of one color with Y thread blocks
- Aggregate partial sums of matrix-vector product using shuffles
- Store all intermediate results and diagonal block in shared memory
- Auto-tune block sizes (X,Y) and launch bounds

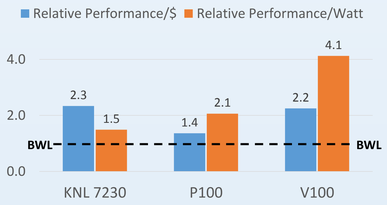
Kernels 1 and 2 Combined

Aggregate Single Node Performance



Speedup over BWL	
KNL	1.3
P100	2.1
V100	5.2

- Combined kernels account for ~70% of FUN3D run-time for common cases
- Accompanying chart shows a comparison of performance per dollar MSRP and performance per watt vs BWL across the three many-core architectures



Conclusions

- Many-core may offer up to 2× more performance per dollar MSRP and up to 4× more performance per watt than multi-core for FUN3D
- GPUs' flexible parallelization is advantageous for matrix construction
- Sparse linear algebra speedups are consistent with the higher main memory bandwidth of many-core architectures
- KNL benefits greatly from explicit prefetching and vectorization
- GPUs require CUDA rewrite to achieve acceptable performance; OpenACC is inadequate for these kernels
- On KNL, coloring bests OpenMP atomics for thread safety
- Hardware-supported atomic operations excel on GPUs

Acknowledgments

This work was developed in part by the User Productivity Enhancement, Technology Transfer and Training (PETTT) Project No. PETA-KY07-001 and by DoE SBIR Grants DE-SC0009593 and DE-SC0017183. The authors would like to acknowledge the computational resources and PETTT software support from the DoD High Performance Computing Modernization office under Contract No. GS04T09DBC0017. Support from the NASA Langley Research Center Comprehensive Digital Transformation initiative and the Revolutionary Computational Aerosciences sub-project within the NASA Aeronautics Research Mission Directorate is also acknowledged. This work was in part developed at the 2017 ORNL Hackathon at NASA, which was a collaboration between and used resources of both the National Aeronautics and Space Administration and the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory. Oak Ridge National Laboratory is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. The authors also wish to acknowledge Dr. Larry Davis of the DoD HPCMP Office for hardware resources used in this study.