ECSS Experience: Preparing a Production MPI Code for Hybrid Execution

Wentao Zhang
Department of Mechanical Science and Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois, U.S.A.

Daniel J. Bodony
Department of Aerospace Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois, U.S.A.

John L. Larson
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois, U.S.A.

Lucas A. Wilson
Texas Advanced Computing Center
The University of Texas at Austin
Austin, Texas, U.S.A.

Introduction This work documents the performance improvements to Codelet-0, a representative subset of an MPI-based Fortran 90 application that investigates the behavior of compressible, viscous gases. The full application uses approximately 15 million Stampede SUs annually among several projects and users. Using the TAU performance tool, we found the Top 5 time-consuming routines and several performance obstacles. Optimization efforts focused on improvements in memory access patterns and data structure layout resulting in a serial 1-core speedup of 2.

Profiling Figure 1a shows the profile, and Figure 2a shows the percentage of peak execution rate of the Top 5 routines before optimization. A TAU performance analysis using hardware counters led us to perform four types of basic optimizations.

Optimizations With the knowledge gained from the previous performance analyses, four types of basic optimizations were performed:

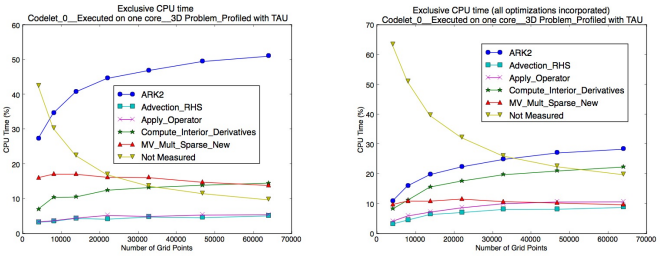
1. **Pointer de-referencing** - Pointers to data arrays in innermost loops were de-referenced outside the loops, thus significantly reducing the number of memory accesses, and also making it easier for the compiler to vectorize the computations.

2. **Algorithm improvement** - The loop order in Compute_Interior_Derivatives was changed to increase the length of inner loop from 5 to 40 to more fully utilize vector instructions and the cache. Sparse matrix-vector multiplication changed from row order to diagonal order.

3. **MV_Mult_Sparse_New** - The loop structure was simplified to reduce the number of indirect memory references.

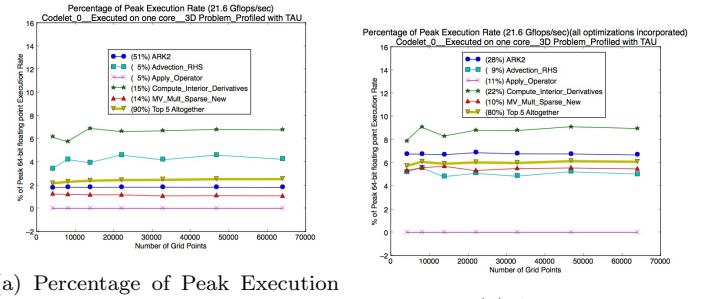
4. **Program data structure** - Large data arrays were redefined as allocatable arrays and referenced directly (rather than indirectly with pointers) allowing the compiler to more easily perform vectorization optimizations.

Results Figure 1b shows the profile, and Figure 2b shows the percentage of peak execution rate of the Top 5 routines after all optimizations were applied. Figure 3 illustrates the resulting Codelet-0 serial speedup of a factor of 2. The Codelet-0 modifications will be incorporated into the full application, and form a new code base for further exploration into optimized multi-core and hybrid parallel executions.



(a) Before (b) After

Figure 1: Percentage of Total Time of Top 5 Routines Before and After Optimization



(a) Percentage of Peak Execution Rate for Top 5 Routines Before and After Optimization

(b) After

Figure 2: Profiling Results for Top 5 Routines in Codelet-0

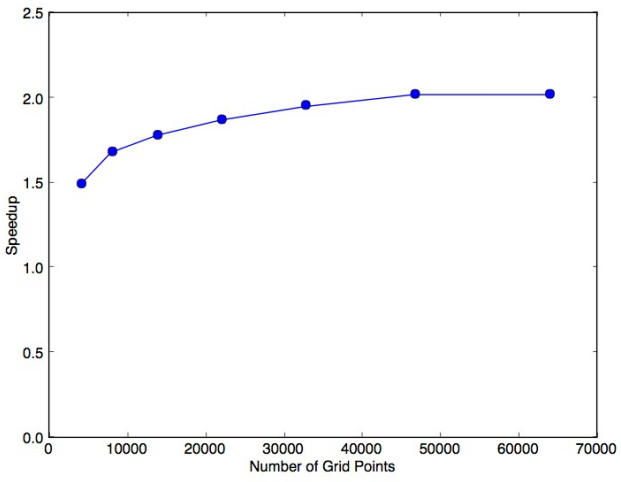


Figure 3: Codelet-0 Speedup after all optimizations