

Performance Enhancements to PlascomCM

Lucas A. Wilson

XSEDE ECSS Symposium

Jan. 19, 2016

Outline

- Background
- Data Encoding
- Vectorization
- Heterogeneous Domain Decomposition
- Summary
- Future Plans

Where Did We Start?

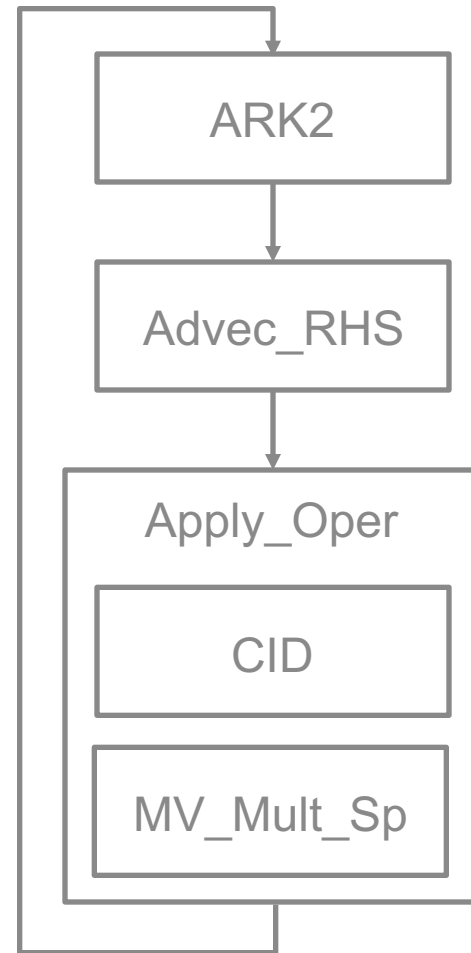
BACKGROUND

Background

- PlascomCM
 - Fortran90+MPI
 - Investigate the behavior of compressible, viscous gases with a focus on turbulence and generated sound.
 - PI: Daniel Bodony, UIUC
- ECSS project currently in 4th year
 - PY 1&2: Restructuring code and data encoding to enable vectorization on Sandy Bridge (SNB)
 - PY 3: Convert homogeneous domain decomposition to heterogeneous domain decomposition for symmetric execution on Knights Corner (KNC)
 - PY 4 (Current): Preparing for transition to Knights Landing (KNL)

Structure of Code

- ARK2
 - RK4 Integration
- Advection_RHS
 - Matrix-Vector Product
 - Store RHS
- Apply_Operator
 - MPI driver routine
 - Compute_Int_Deriv
 - N^3
 - MV_Mult_Sparse
 - N^2



Improving Single-task Performance

DATA ENCODING

Data Encoding

- Original code used standard compressed sparse row (CSR) representation for stencil matrix
 - For 5-point stencil, this allowed only 5 iterations within inner loop (multiplication)
 - Also, only 5 contiguous memory address lookups
 - 1 4-wide double-precision vector operation plus a scalar peel
- Modified code uses a customized diagonal representation, since the regular stencil is naturally clustered around the diagonal
 - Multiplication inner loop length (and contiguous memory accesses) increased from 5 to 40
 - 10 4-wide double-precision vector operations on SNB
 - Allows for 5 complete L1 cache lines per inner loop iteration
- **39% performance improvement**

CSR vs. Diagonal Storage

	Row	Diagonal
Load Instructions	2.66×10^9	2.48×10^9
FLOPs	2.30×10^9	2.30×10^9
Main Mem. Hits	0.00%	0.00%
L3 Hits	0.18%	7.60%
L2 Hits	2.25%	4.65%
L1 Hits	97.57%	87.75%
Time (seconds)	1.65	1.19
Mem. Loads/FLOP	1.15	1.08
% of Peak	6.47	8.99

Improving Single-task Performance

VECTORIZATION

Vectorization

- Original code used multiple compound types for grid encoding
 - Prevents vectorization due to non-obvious contiguous data access

```
do i = 1, grid%nCells
    state%time(i) =
        state%timeOld(i) +
        c_vec(rkStep) *
        state%dt(i)
end do
```

Vectorization

- Static in-loop structure dereferences were aliased to expose contiguous memory accesses to the compiler
- 50% improvement in full code wall time
 - Intel14 w/ O2
 - No vectorization

```
time => state%time
timeOld => state%timeOld
dt => state%dt

do i = 1, grid%nCells
    time(i) = timeOld(i) +
                c_vec(rkStep) *
                dt(i)
end do
```

Vectorization

- With contiguous accesses exposed within inner loops, SIMD directives were inserted
- Further 30% improvement in full code wall time

```
time => state%time
timeOld => state%timeOld
dt => state%dt

!DIR$ SIMD
do i = 1, grid%nCells
    time(i) = timeOld(i) +
                c_vec(rkStep) *
                dt(i)
end do
```

Enabling Symmetric MPI Execution on Host and Phi

HETEROGENEOUS DOMAIN DECOMPOSITION

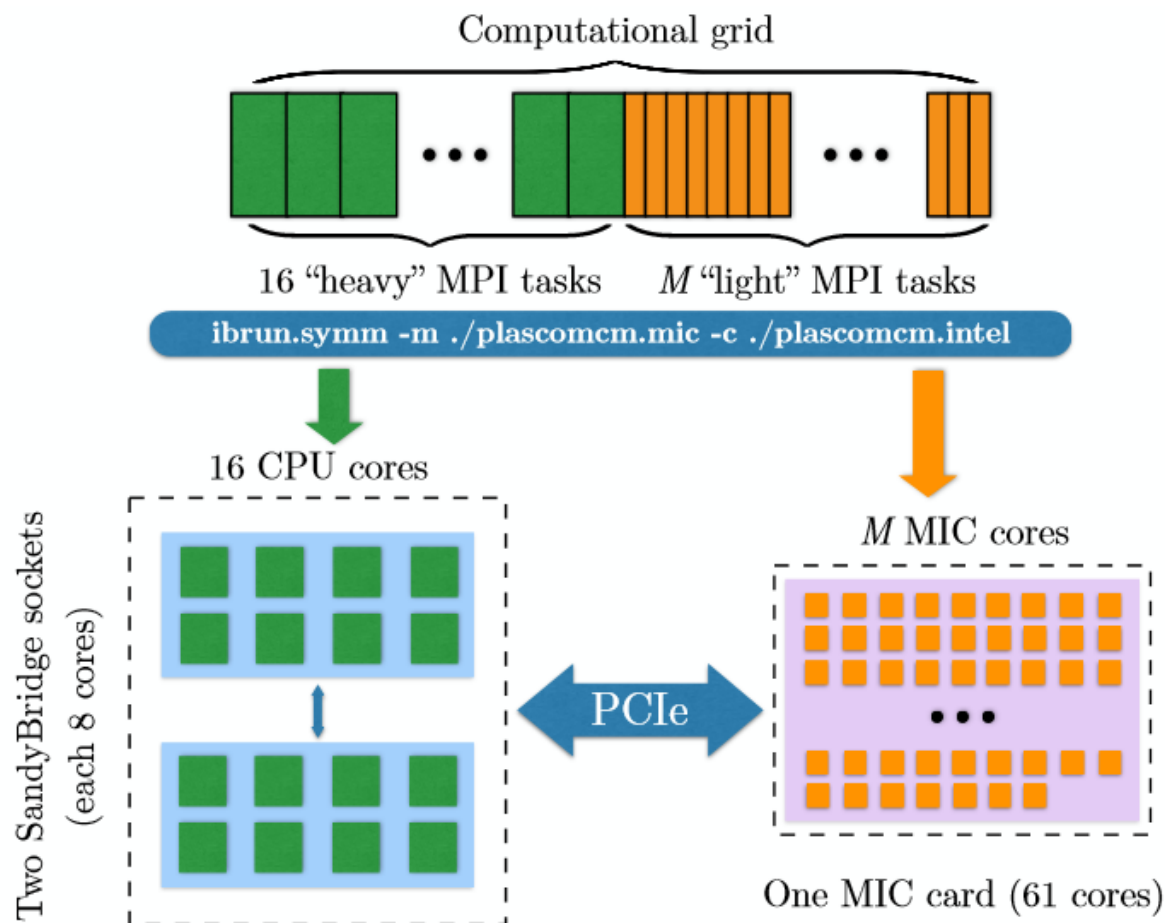
Heterogeneous Decomposition

- The original code used a homogeneous 1D domain decomposition to distribute work across MPI tasks.
 - Advantages:
 - Simple
 - Works well for homogeneous resources
 - Drawback:
 - Memory footprint per task did not allow many MPI processes to execute on KNC (4-8)

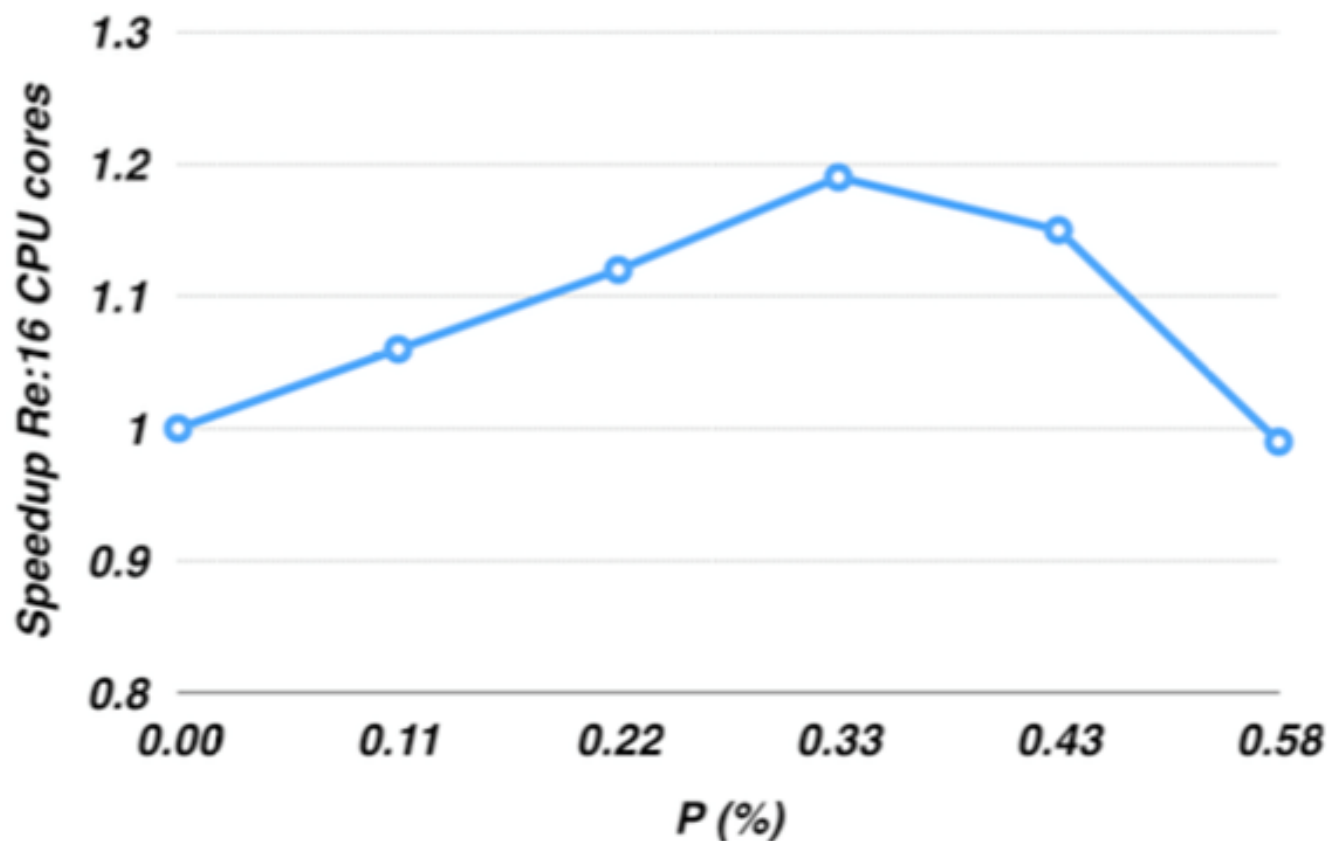
Heterogeneous Decomposition

- To enable more processes per KNC card, a 1-D heterogeneous decomposition was used
 - n “Heavy” MPI tasks for the SNB
 - m “Light” MPI tasks for the KNC

Heterogeneous Decomposition

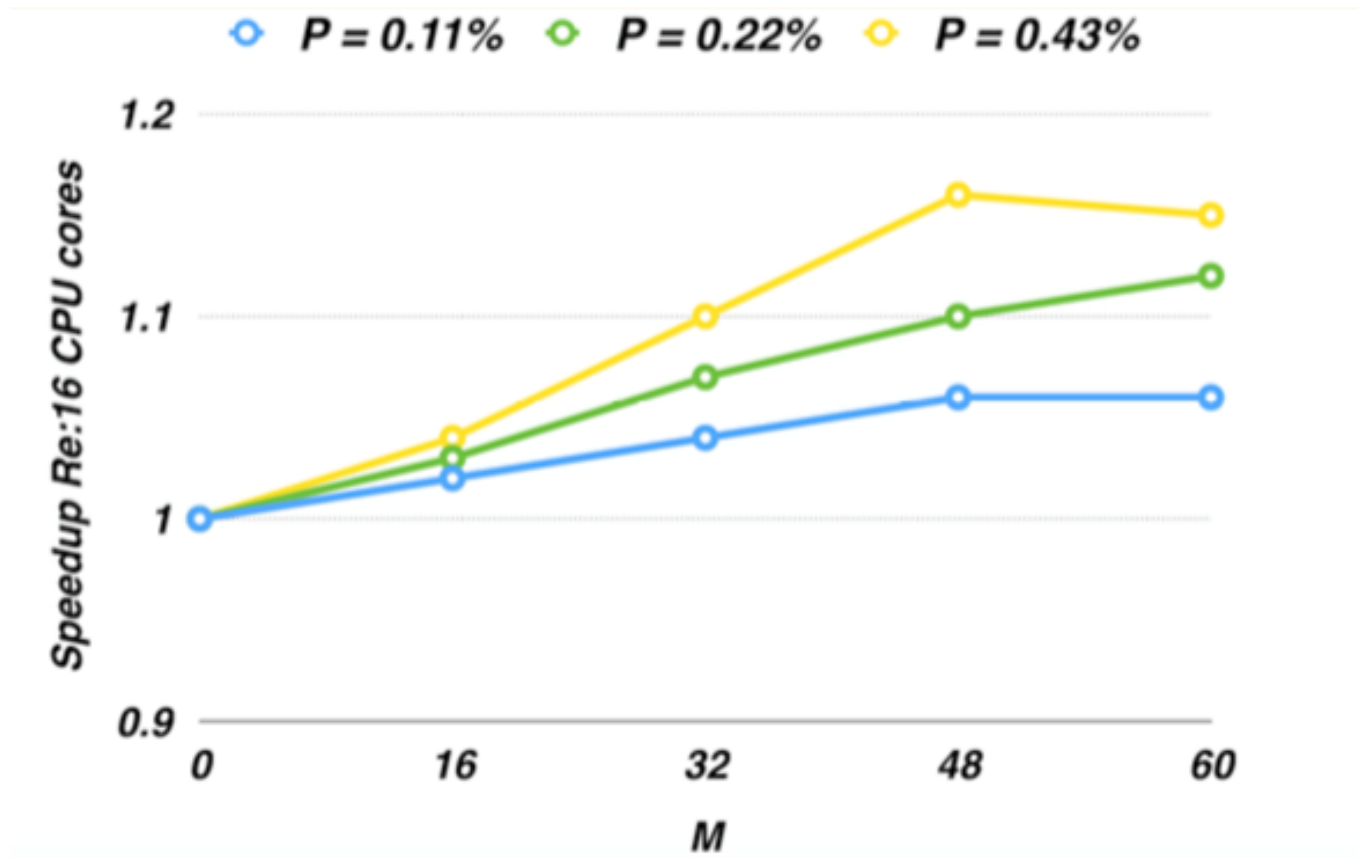


Heterogeneous Decomposition



20% Speedup with 60 KNC tasks executing 33% of domain

Heterogeneous Decomposition



Speedup achieved by varying tasks per KNC with different workload amounts

Heterogeneous Decomposition

- Roadblocks to Scalability:
 - Boundary condition routines are mostly sequential
 - Performance on KNC is poor
 - Code does not intelligently decide which MPI tasks are “heavy” and which are “light”
 - Possible solution: execute pieces of the domain containing boundary elements on SNB, and interior pieces of the domain on KNC

Effect of Improvements to Date

SUMMARY

Summary

- So far, the performance of PlascomCM has been improved by > 2.3 times
 - Data representation changes
 - Inner loop changes to expose vectorization and reduce indirect memory accesses
 - Decomposition changes to allow for more efficient execution on lightweight KNC cores
- At current allocation rates for this project, ***potential savings of 12-14 million SUs per year***

Moving to KNL

FUTURE PLANS

Future Plans

- Slow boundary condition execution time on KNC is due to mostly sequential nature of algorithm
 - KNC is poor for sequential algorithms:
 - 1 GHz
 - In-order
 - No branch prediction

Future Plans

- KNL architecture *may* address many of the root causes of poor boundary condition performance
 - Wait and see

Lucas A. Wilson
lwilson@tacc.utexas.edu

For more information:
www.tacc.utexas.edu

