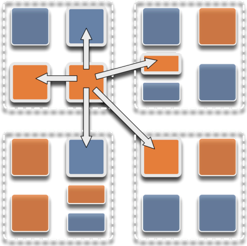
Introducing Overdecomposition to Existing Applications: *PlasComCM* and AMPI



Sam White

Parallel Programming Lab
UIUC

PCI Parallel Computing Institute
CSE Computational Science & Engineering
THE OHIO STATE UNIVERSITY
Center for Exascale Simulation
XPACC
of Plasma-Coupled Combustion

Introduction

How to enable Overdecomposition, Asynchrony, and Migratability in existing applications?

1. Rewrite in a runtime-assisted language
2. Use the parallelism already expressed in the existing code

Adaptive MPI is our answer to 2 above

- Implementation of MPI, written in Charm++

XPACC

XPACC: The Center for Exascale Simulation of Plasma-Coupled Combustion

- PSAAPII center based at UIUC
- Experimentation, simulation, and computer science collaborations

Goals:

- Model plasma-coupled combustion
- Understand multi-physics, chemistry
- Contribute to more efficient engine design

XPACC

What is plasma-coupled combustion?

- Combustion = fuel + oxidizer + heat

Plasma (ionized gas) helps catalyze combustion reactions

- Especially with low air pressure, low fuel, or high winds
- Why? This is not well understood

XPACC

Main simulation code: *PlasComCM*

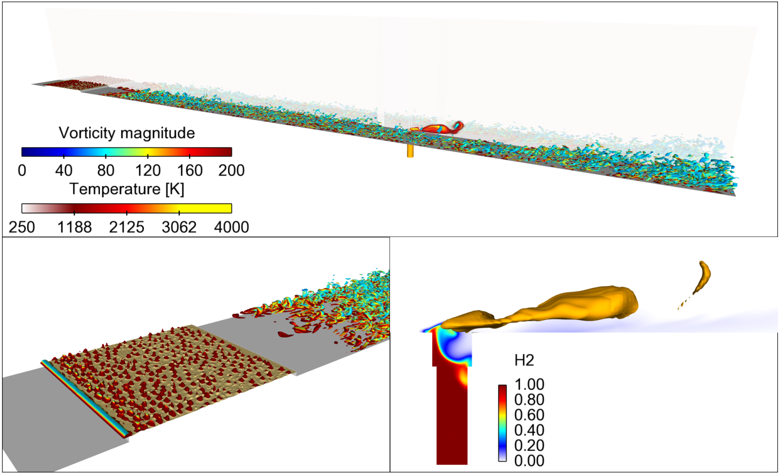
- A multi-physics solver that can couple a compressible viscous fluid to a compressible finite strain solid
- 100K+ lines of Fortran90 and MPI
- Stencil operations on a 3D unstructured grid

XPACC

PlasComCM solves the Compressible Navier-Stokes equations using the following schemes:

- 4th order Runge-Kutta time advancement
- Summation-by-parts finite difference schemes
- Simultaneous-approximation-terms boundary conditions
- Compact stencil numerical filtering

XPACC



Vorticity magnitude
0
40
80
120
160
200
Temperature [K]
250
1188
2125
3062
4000
H2
1.00
0.80
0.60
0.40
0.20
0.00

XPACC

Challenges:

- Need to maintain a “Golden Copy” of source code for computational scientists
- Need to make code itself adapt to load imbalance

Sources of load imbalance:

- Multiple physics
- Multi-rate time integration
- Adaptive Mesh Refinement

Adaptive MPI

Adaptive MPI is an MPI interface to the Charm++ Runtime System (RTS)

- MPI programming model, with Charm++ features

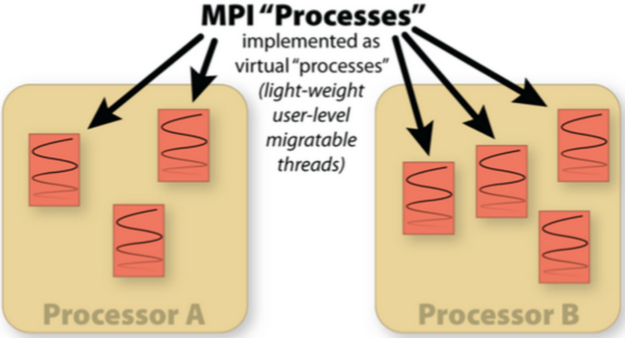
Key Idea: MPI ranks are not OS processes

- MPI ranks can be user-level threads
- Can have many virtual MPI ranks per core

Adaptive MPI

Virtual MPI ranks are lightweight user-level threads

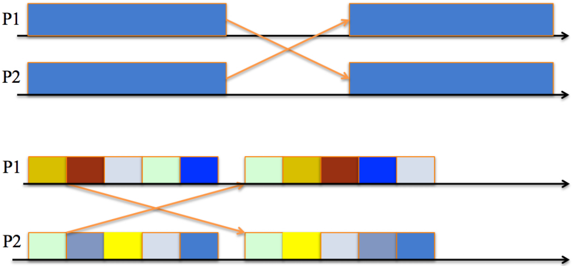
- Threads are bound to migratable objects



Asynchrony

Message-driven scheduling tolerates network latencies

- Overlap of communication/computation



Migratability

MPI ranks can be migrated by the RTS

- Each rank addressed by a global name
- Each rank needs to be serializable

Benefits:

- Dynamic load balancing
- Automatic fault tolerance
- Transparent to user $\rightarrow$ little application code

Adaptive MPI

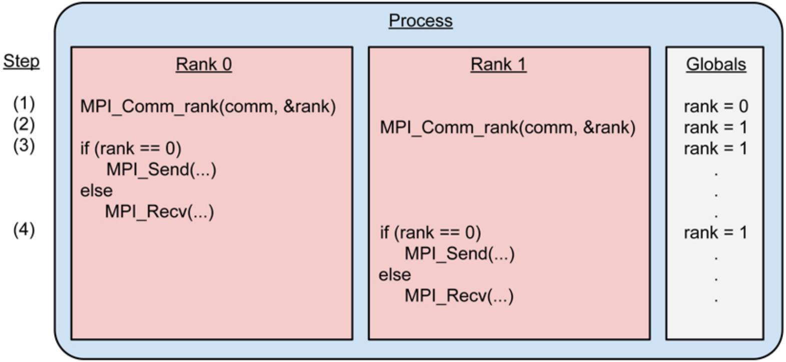
Features:

- Overdecomposition
- Overlap of communication/computation
- Dynamic load balancing
- Automatic fault tolerance

All this with little* effort for existing MPI programs

- MPI ranks must be thread-safe, global variables rank-independent

Thread Safety



- Threads (Ranks 0-1) share the global variables of their process

Thread Safety

Automated approach:

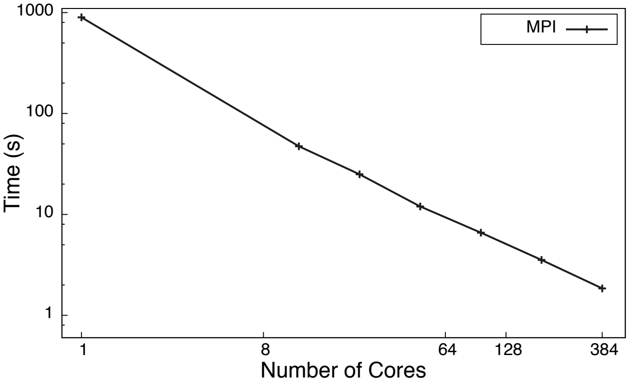
- Idea: Use ROSE compiler tool to tag unsafe variables with OpenMP's Thread Local Storage
- Issue: OpenFortran parser is buggy

Manual Approach:

- Idea: Identify unsafe variables with ROSE tool, transform by hand
- Benefits: Mainline code is thread-safe, cleaner

Results

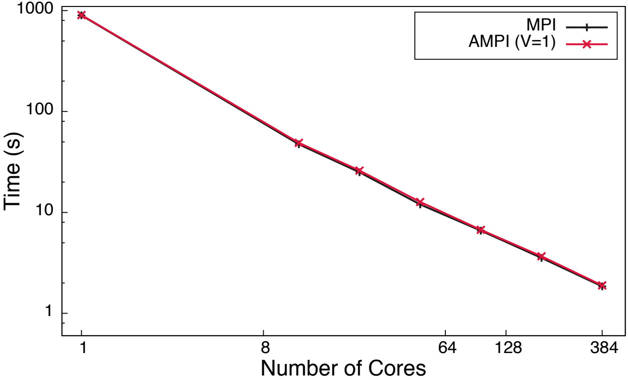
- AMPI virtualization (V = Virtual ranks/core)



- 1.7M grid pts, 24 cores/node of Mustang

Results

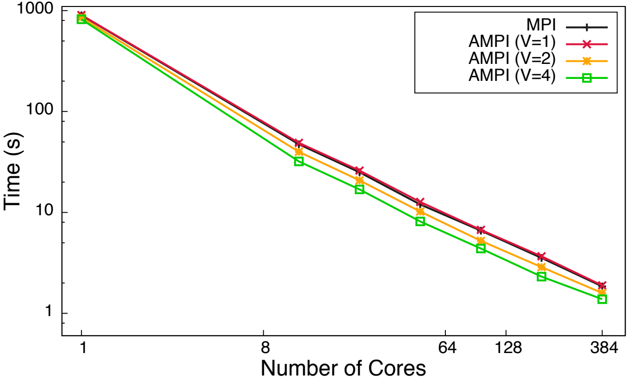
- AMPI virtualization (V = Virtual ranks/core)



- 1.7M grid pts, 24 cores/node of Mustang

Results

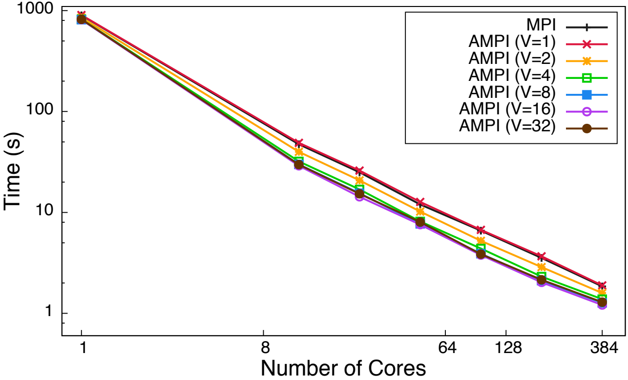
- AMPI virtualization (V = Virtual ranks/core)



- 1.7M grid pts, 24 cores/node of Mustang

Results

- AMPI virtualization (V = Virtual ranks/core)



- 1.7M grid pts, 24 cores/node of Mustang

Results

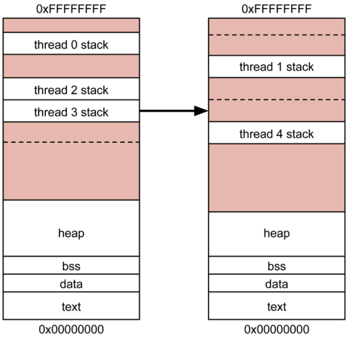
- Speedup on 8 nodes, 192 cores (Mustang)

Virtualization	Time (s)	Speedup
MPI	3.54	1.0
AMPI (V=1)	3.67	0.96
AMPI (V=2)	2.97	1.19
AMPI (V=4)	2.51	1.41
AMPI (V=8)	2.31	1.53
AMPI (V=16)	2.21	1.60
AMPI (V=32)	2.35	1.51

Thread Migration

Automated thread migration: Isomalloc

- Idea: Allocate to globally unique virtual memory addresses
- Issue: Not fully portable, has overheads



Thread Migration

Assisted migration: Pack and UnPack (PUP) routines

- PUP framework in Charm++ helps
- One routine per datatype

Challenge: *PlasComCM* has many variables!

- Allocated in different places, with different sizes
- Existing Fortran PUP interface: pup_ints(array,size)

Thread Migration

New Fortran2003 PUP interface → Auto-generate application PUP routines

- Simplified interface: call pup(ptr)
- More efficient thread migration
- Maintainable application PUP code

Performance improvements:

- 33% reduction in memory copied, sent over network
- 1.7x speedup over isomalloc

Load Balancing

PlasComCM does not have much load imbalance yet

- Algorithmic choice has been constrained by load balance concerns

But we are ready for it:

- With Isomalloc, no AMPI-specific code needed
- Load balancing and checkpoint/restart are just one function call each

Future Work

Demonstrate load balancing:

- Load imbalance slowly being introduced to *PlasComCM*
- AMPI minimizes load balance concerns for scientists

Communication optmizations:

- Halo exchanges within a node could use shared memory
- Node-level shared buffer for transient ghost regions

Summary

Overdecomposition is key to adaptivity and performance

- Adaptive MPI provides low-cost access
- Code transformations can be automated or assisted
- Load balancing, overdecomposition are transparent to users

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0002374.

Questions?