Recent Developments in Adaptive MPI

Sam White

Overview

- Introduction to AMPI
- Recent Work
  - Communication Optimizations
  - Automatic Global Variable Privatization

Charmworks PPL UIUC

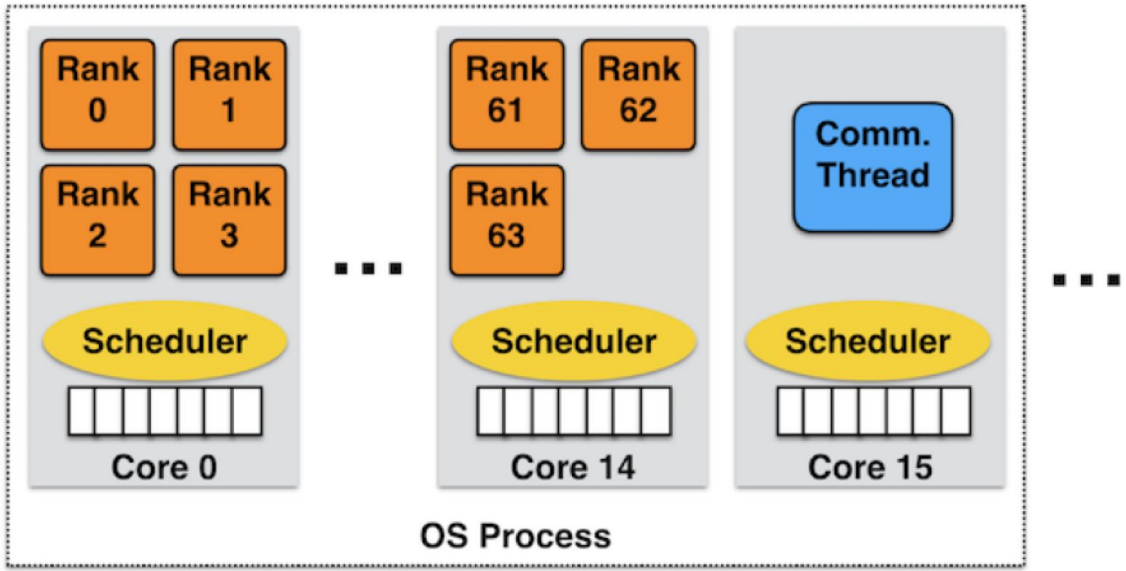
Motivation

- Variability in various forms (SW and HW) is a challenge for applications moving toward exascale
  - Task-based programming models address these issues
- How to adopt task-based programming models?
  - Develop new codes from scratch
  - Rewrite existing codes, libraries, or modules (and interoperate)
  - Implement other programming APIs on top of tasking runtimes

Charmworks PPL UIUC

Background

- AMPI virtualizes the ranks of MPI_COMM_WORLD
  - AMPI ranks are user-level threads (ULTs), not OS processes



Background

- AMPI virtualizes the ranks of MPI_COMM_WORLD
  - AMPI ranks are user-level threads (ULTs), not OS processes
  - Cost: virtual ranks in each process share global/static variables
  - Benefits:
    - Overdecomposition: run with more ranks than cores
    - Asynchrony: overlap one rank's communication with another rank's computation
    - Migratability: ULTs are migratable at runtime across address spaces

Charmworks PPL UIUC

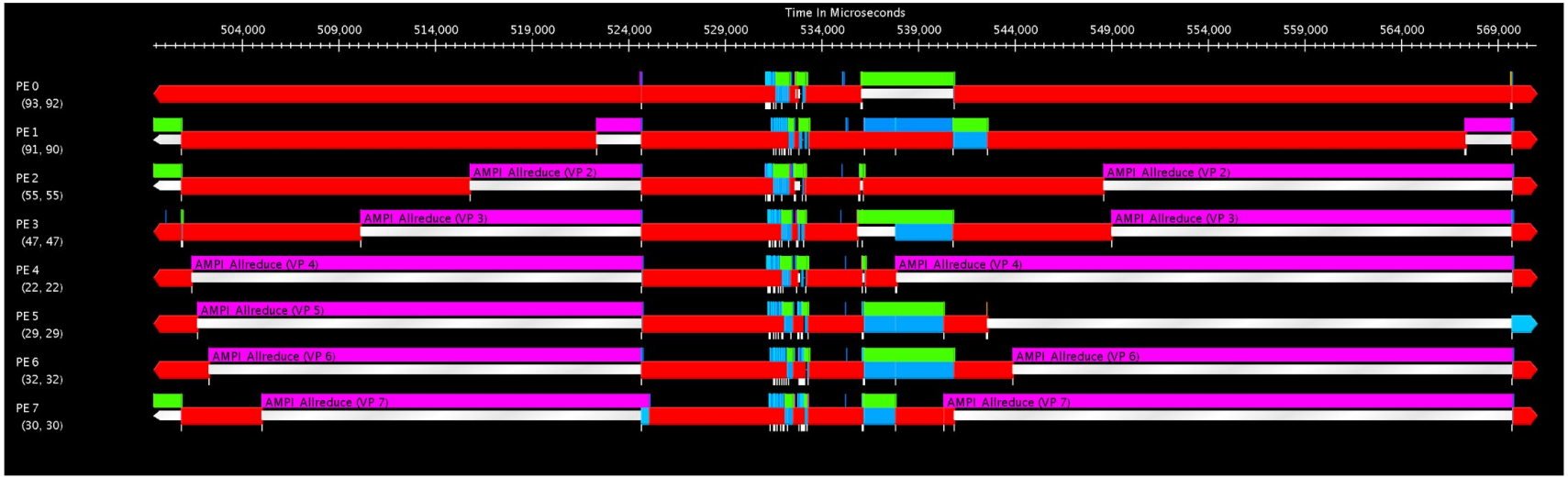
AMPI Benefits

- Communication Optimizations
  - Overlap of computation and communication
  - Communication locality of virtual ranks in shared address space
- Dynamic Load Balancing
  - Balance achieved by migrating AMPI virtual ranks
  - Many different strategies built-in, customizable
  - Isomalloc memory allocator serializes all of a rank's state
- Fault Tolerance
  - Automatic checkpoint-restart within the same job

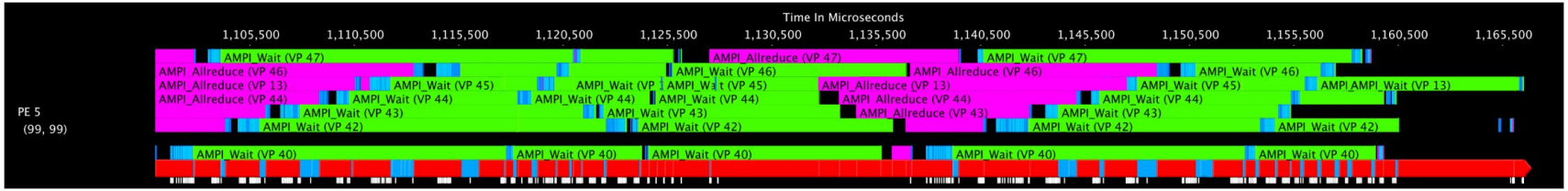
Charmworks PPL UIUC

AMPI Benefits: LULESH-v2.0

No overdecomposition or load balancing (8 VPs on 8 PEs):

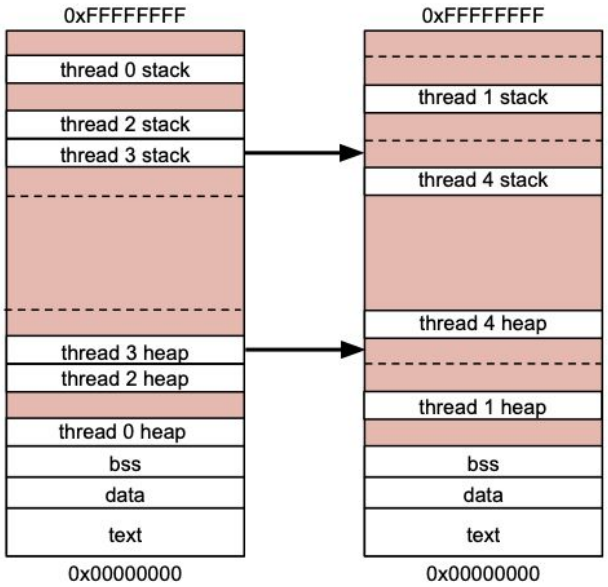


With 8x overdecomposition, after load balancing (7 VPs on 1 PE shown):



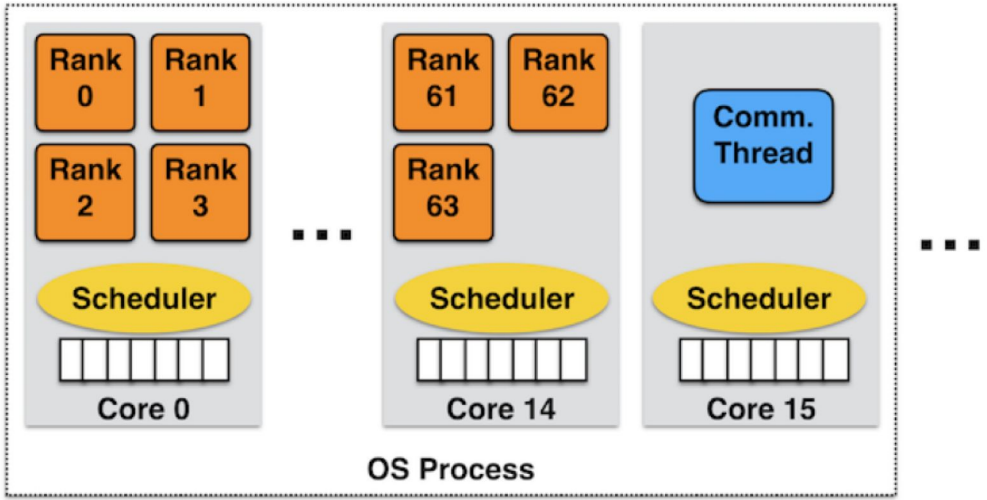
Migratability

- Isomalloc memory allocator *reserves* a globally unique slice of virtual memory space in each process for each virtual rank
- Benefit: no user-specific serialization code
  - Handles the user-level thread stack and all user heap allocations
  - Enables dynamic load balancing and fault tolerance



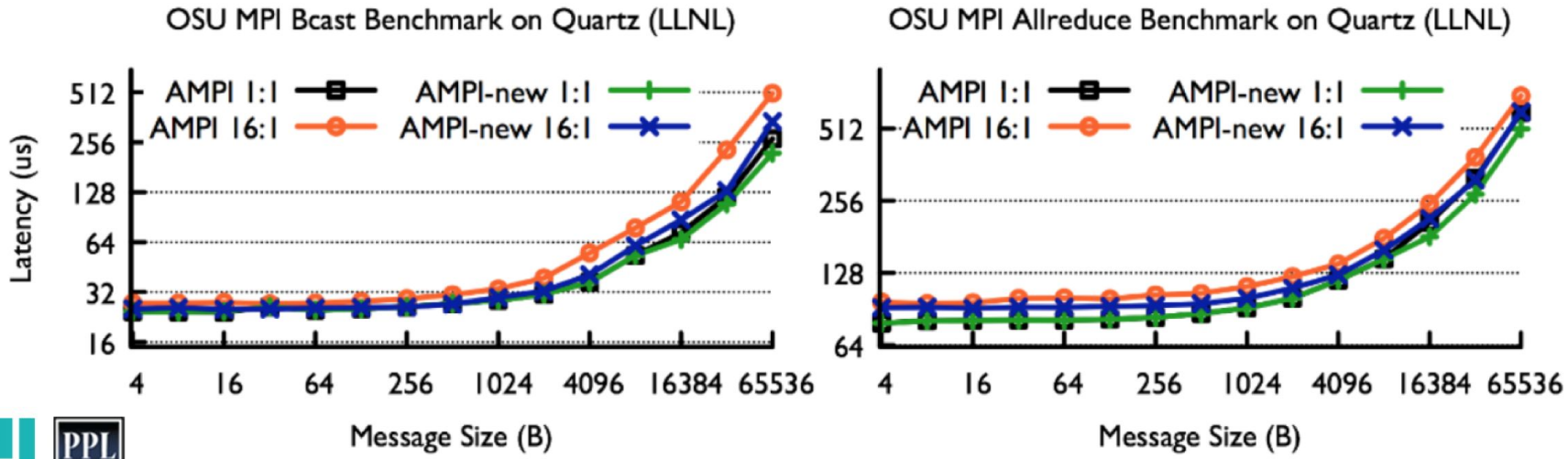
Communication Optimizations

- AMPI exposes opportunities to optimize for communication locality:
  - Multiple ranks on the same PE
  - Many ranks in the same OS process



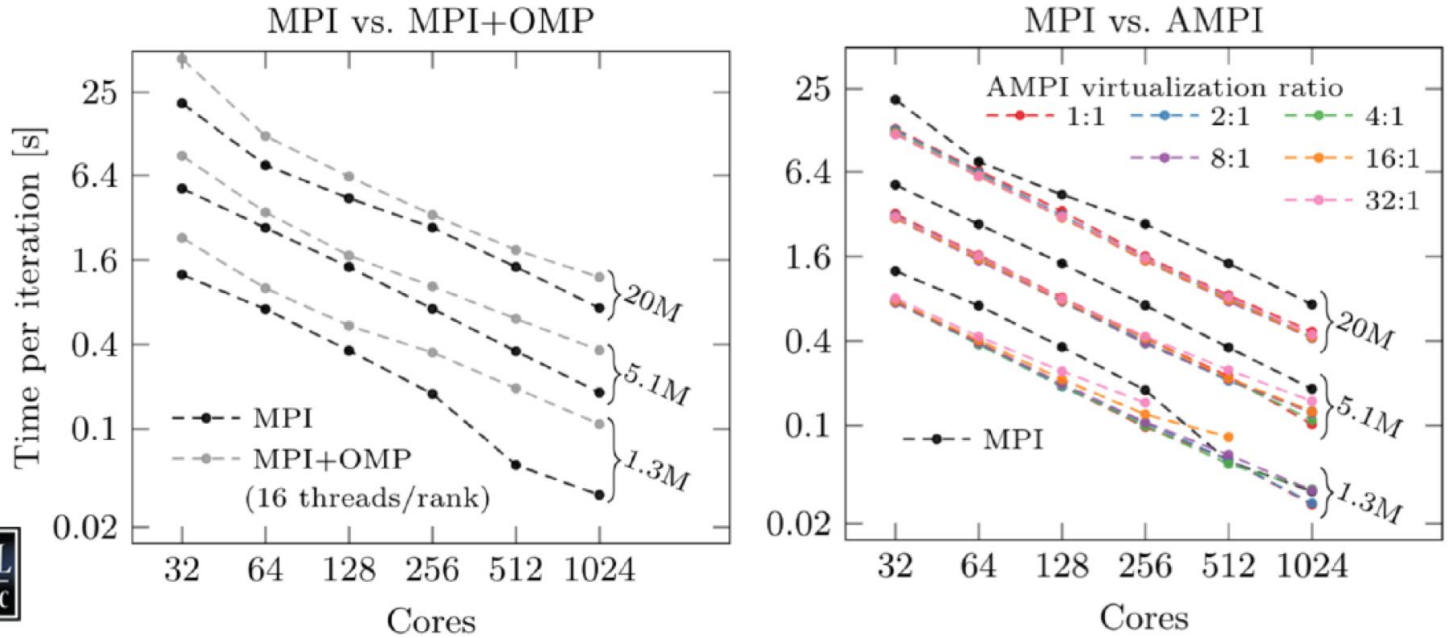
Communication Optimizations

- New virtualization-aware collective implementations avoid *O(VP)* message creation and copies
  - Next: further shared-memory awareness



Communication Optimizations

- Application study: XPACC's *PlasCom2* code
  - Now seeing AMPI outperform MPI (+OMP) even without LB



Privatization Problem

Illustration of unsafe global/static variable accesses:

```
int rank_global;

void func(void)
{
  MPI_Comm_rank(MPI_COMM_WORLD, &rank_global);

  MPI_Barrier(MPI_COMM_WORLD);

  printf("rank: %d\n", rank_global);
}
```

MPI Output:
0
1
2
3

AMPI Output:
3
3
3
3

Charmworks PPL UIUC

Privatization Methods

- Existing methods
  - Manual refactoring
    - Developer encapsulates mutable global state in structures, passes around the stack
    - Portable, but can take days/weeks of developer effort
  - Thread-local storage segment pointer swapping (TLSglobals)
    - Only need to tag variable declarations, not accesses
    - Now works on LInux and MacOS with GCC and recent Clang
- In-Development methods
  - Process-in-Process library integration (PiPglobals)
  - File-system Globals (FSglobals)
  - Clang/libtooling-based source-to-source transformation

Charmworks PPL UIUC

Conclusion

- AMPI is increasingly valuable for a growing set of applications
  - Not just those with load imbalance
- Recent work spans the full stack of AMPI
  - Conformance to the MPI-3.1 standard
  - Communication performance improvements in AMPI/Charm++/LRTS
  - More automated tooling for conversion of legacy code
  - Working closely with more application developers

Charmworks PPL UIUC

Questions?