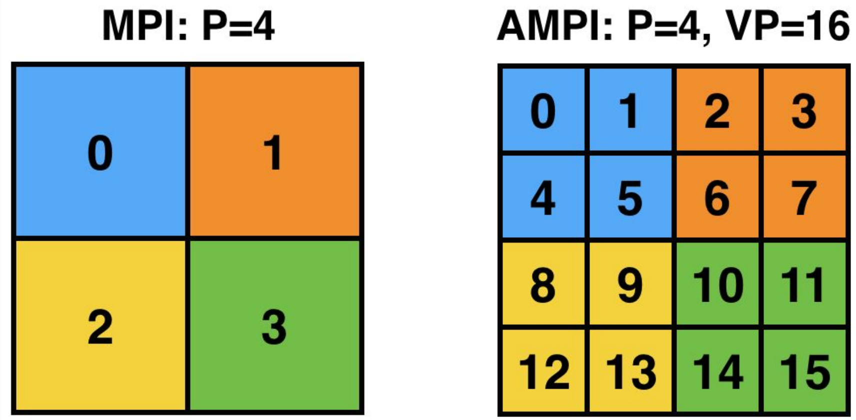
CharmMPI: From Research Code to Production Workhorse

Evan Ramos,
Charmworks, Inc.

Recap: What does CharmMPI do?

- Create MPI ranks as user-level threads that coexist in OS processes
  - Within-node communication is low-latency and high-bandwidth
  - Idle ranks can yield to other ULTs with work to do
- Allocate each rank's call stack and heap at a deterministic location
  - Can migrate ranks across network or snapshot them to disk
- Measure time spent in each rank & redistribute them to balance load
  - Allows iterations to complete faster, **decreasing total run time**



How is CharmMPI's functionality achieved?

- CharmMPI is a *Charm++* program and the ULTs are chares
- ULTs provided by *uFcontext*, using *boost.context* ASM underneath
- Deterministic memory positioning provided by *Isomalloc*
  - Call stacks allocated manually as part of startup procedure
  - Runtime heap operations (malloc/free, new/delete) intercepted
- What else is part of a program's state?

The Privatization Problem

```
int rank_global;

void print_ranks(void)
{
  MPI_Comm_rank(MPI_COMM_WORLD, &rank_global);
  MPI_Barrier(MPI_COMM_WORLD);
  printf("rank: %d\n", rank_global);
}
```

- Time-consuming and difficult to fix codebase manually
- Automated solution preferable

TLSglobals (2010)

- Thread-Local Storage Segment Pointer Swapping
  - Add *thread_local* tag to global variable declarations and definitions (but not accesses)
  - Supported with migration on Linux (GCC, Clang 10+), macOS (Apple Clang, GCC)
  - O(1) context switching cost
  - Good balance of ease of use, portability, and performance
  - Still requires manual changes, just not intensive refactoring
  - Clang/libtooling-based C/C++ automated transformer created at Charmworks
  - Supported on x86/x86_64, AArch64 and POWER support in progress (2021)

PIEglobals (2020)

- Position-Independent Executable (PIE) Runtime Relocation
  - *ampicc*, *ampif90*, etc. build the MPI program as a PIE shared object
  - PIE binaries store and access globals relative to instruction pointer
  - CharmMPI processes the shared object at program start:
    - *dlopen*: dynamically load shared object once per OS process
    - Walk ELF (Executable and Linkable Format) header: list program segments in memory
    - Duplicate code & data segments for each virtualized rank w/ Isomalloc
    - Update PIC (Position-Independent Code) relocations to point to new privatized addresses
    - Calculate privatized location of entry point for each rank and call it
  - Result: global variables become **privatized** and **migratable** with **no changes**

PIEglobals: Advancements in 2021

- Shared objects opened once per logical node instead of per rank
  - Critical to avoid crashes in glibc due to interaction of dlopen and pthreads
- Automatically combined with TLSglobals whenever available
  - Prevents issues due to preexisting *thread_local*s and system libraries with TLS
- Added rank tracking infrastructure to AMPI's Charm implementation
  - Necessary for user-defined reductions: function pointers differ by rank
- Validated on ARM and POWER architectures
- Merged to Charm's *main* branch

CharmMPI with PIEglobals: Successes

- miniGhost
- Nekbone
- MFEM
- Laghos
- Continued collaboration with major ISV on an industrial FEA code

CharmMPI with PIEglobals: Frontiers

- OpenFOAM
- mpi4py

CharmMPI Development History

- Adaptive MPI began as research in PPL @ UIUC around 2001
  - Continued work until present, with more focused effort beginning in 2014
- Charmworks awarded DOE SBIR in 2017
  - Phase II grant concluded in 2021
  - Made robust, standard compliant, and improved performance

What steps were originally needed to use AMPI?

- Edit your code's build system to point toolchain to *ampicc* full path, or pass as parameters to configure step
- Handle globals
- Handle entry point
  - C/C++: *#include "mpi.h"* before *main*
  - Fortran: Rename *program XYZ* to *subroutine MPI_Main*
- Want migration and load balancing? Don't forget to link with *-module CommonLBs -memory isomalloc* (editing Makefile, or passing parameters)
- Don't forget to run with *+isomalloc_sync ++no-va-randomization* or migration could fail
- Learn how to use charmrun

CharmMPI's Ease of Use Improvements

- Goal: less to explain, less to remember, fewer barriers to entry
- *ampicc* automatically passes *-memory isomalloc -module CommonLBs*
- *+isomalloc_sync* on by default and implementation cleaned up
- Disable ASLR by default: keep code pointers on call stack deterministic
- Added directory containing unadorned *mpicc*, for use like Modules
    - *cd "netlrts-linux-x86_64/bin/ampi" && export PATH="$(pwd):$PATH"*
- Added *AMPI_BUILD_FLAGS* environment variable to simplify passing *-tlsglobals*, *-pieglobals*, etc. to *ampicc*
- PIEglobals can use *main* as an entry point (C, C++, **and Fortran!**)
- Added *+n* argument to charmrun to specify node count directly
- CharmMPI Onboarding Tutorial published
    - https://github.com/UIUC-PPL/charm/wiki/CharmMPI-Onboarding-Tutorial

CharmMPI's Robustness Improvements

- Portability with Cray, macOS, shared objects, Clang solidified
- Replaced Isomalloc's bespoke mempool with glibc's dlmalloc
- Isomalloc rewritten to divide address space by logical unit (MPI rank), not Charm processing element (PE), avoiding contention
- Isomalloc now wraps more heap APIs (posix_memalign, aligned_alloc)
- Isomalloc uses in-place network transfers when available, avoiding memory usage spikes during migration and potential out-of-memory
- Use a Fortran entry point for Charm when running Fortran code
- *AMPI-only* build target to specifically tailor Charm++ configuration
- charmc and charmrun now support arguments with spaces
- conv-core, conv-util, conv-partition, conv-ldb, conv-machine, tmgr, and hwloc_embedded all combined into one libconverse.a/.so

Case Study: LAMMPS on CharmMPI

- Upstream replaced unsafe *strtok* function with custom C++ parsing
- Accepted patch to fix a remaining thread-safety issue in regex parsing
- With above, **rank virtualization successful**
- Migration faced obstacle of stale *stdio.h* file handles after migration
- **Solution:** Intercept & proxy *FILE** APIs, reopen and seek at destination

```
/* mpi.h: */
#include <stdio.h>

FILE* ampi_fopen(const char* filename, const char* mode);
int ampi_fclose(FILE* stream);
size_t ampi_fread(void* ptr, size_t size, size_t nmemb, FILE* stream);
size_t ampi_fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);
/* ... */
#define fopen ampi_fopen
#define fclose ampi_fclose
#define fread ampi_fread
#define fwrite ampi_fwrite
/* ... */
```

evan@hpccharm.com

Charmworks