BLUE WATERS

SUSTAINED PETASCALE COMPUTING

Writing Parallel Libraries with MPI
-- The Good, the Bad, and the Ugly --

Torsten Hoefler

With input from Bill Gropp and Marc Snir

Keynote at EuroMPI 2011, Sept 21st 2011, Santorini, Greece

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

Outline

- Modular programming basics
- Modular distributed memory programming
  - A taxonomy for parallel libraries
- MPI's loosely synchronous model
  - The Good
  - The Bad
  - The Ugly
- Guidelines and best practices

Modular Programming Basics

- Modular programming is important for:
  - Code reuse (even buy and sell)
  - Smaller scope for optimizations
  - Code exchange (clear interfaces)
  - Performance portability
  - Separation of concerns (implementation, testing)
- Libraries are the "de-facto" standard for modular programming ☺
  - Found to improve productivity and reduce bugs

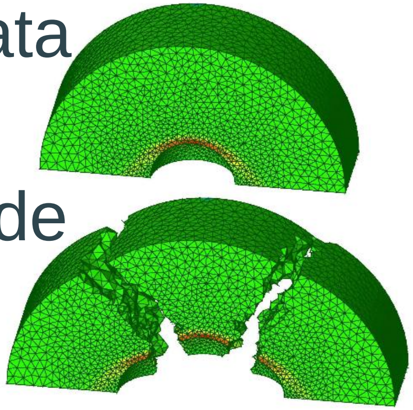
T. Korson, J.D. McGregor: Technical criteria for the specification and evaluation of object-oriented libraries

Component-based Software Engineering (CBSE)

- Program by composing large-scale components
- Desirable attributes of a library:
  - Wide domain coverage
  - Consistency, robustness
  - Easy-to-learn, easy-to-use, intuitive
  - Component efficiency
  - Extensibility, integrability
  - Well-supported

Distributed CBSE?

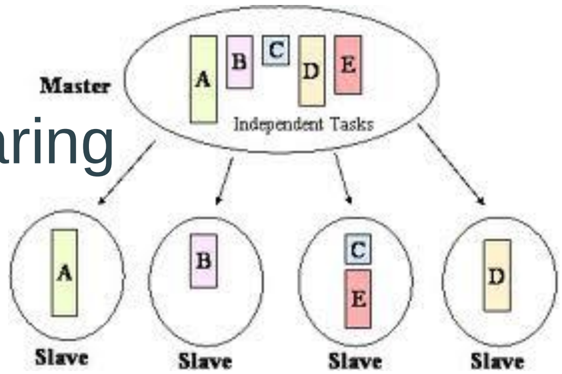
- Needs to control multiple resources (PEs)
- Learn from the Eiffel language:
  - Classes – organize components around data structures and not action structures
  - Information hiding – export facilities, but hide internal structures (avoid “cross talk”)
  - Assertions – characterize semantics
  - Inheritance – module inclusion and subtyping
  - Composability – performance composability and functional orthogonality



Meyer, B.: Lessons from the Design of the Eiffel Libraries

Spatial Resource Sharing

- Serial libraries: only temporal resource sharing
  - Assuming "enough" memory
- Parallel libraries: also spatial resource sharing
  - E.g., master/worker



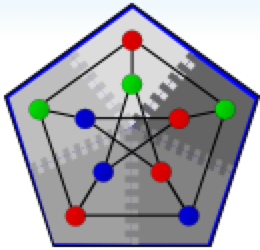
- Main library types:
  1. Spatial (use some processes to implement services, leave other processes to user, e.g., ADLB)
  2. Collective, loosely-synchronous (called "in order" but not synchronous from a static process group, e.g., PETSc)
  3. Collective, asynchronous (called from a static process group but work asynchronously, e.g., libNBC)

A Taxonomy for Parallel Libraries

1. Computational libraries
   - Full computations, often domain-specific, e.g., PETSc, ScaLAPACK, PBGL, PPM
2. Communication libraries
   - Provide (high-level) communication functions, e.g., libNBC, AM++
3. Programming model libraries
   - Specialized (limited) programming model, e.g., ADLB, AP
4. System and utility libraries
   - Interface architectural subsystems, e.g., LibTopoMap, HDF5, Boost.MPI, C# MPI bindings, pyMPI …

T. Hoefler, M. Snir: Writing Parallel Libraries with MPI – Common Practice, Issues, and Extensions

Example Computational Libraries



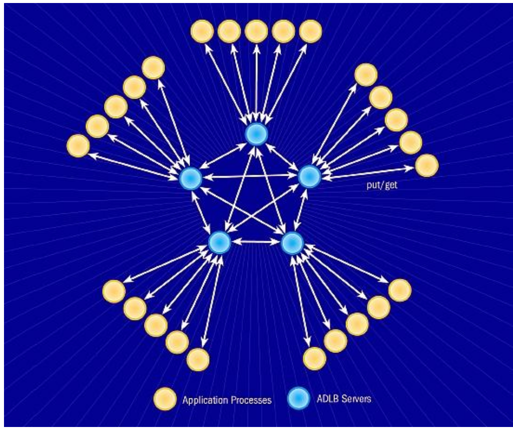
- PETSc
  - Offers algorithms and data structures
  - Scoped with MPI communicators (duped/isolation)
  - Hides communication (uses advanced features)
  - Nonblocking interface (VectScatter{Begin,End}())
- PBGL (Parallel Boost Graph Library)
  - Implements graph algorithms and data structures
  - Generic C++, lifting from sequential algorithms
  - Scoped in process group (e.g., MPI process group)
  - Distributed property map and queue hide comms.

Example Computational Libraries

- PMTL (Parallel Matrix Template Library)
  - Distributed vectors and matrices for linear algebra
  - Completely hides communication
  - Topology mapping (MPI-2.2)
- PPM (Parallel Particle Mesh)
  - Domain decomposition and automatic communication
  - High-level application-oriented interface

Example Programming Model Library

- ADLB (Asynch. Dynamic Load Balancing)
  - Offers a simplified programming model
  - Highly scalable master/worker computations
  - Spatial decomp. (master/worker)
  - User controls workers (with tasks)
- AP (Active Pebbles)
  - Data-driven, fine-grain anon. objects
  - User supplies message handlers and distribution objects
  - Object-based addressing, coalescing and routing



Example Communication Libraries

- LibNBC (nonblocking collectives)
  - Adds support for NBC to MPI-1.0
  - Threaded and “manual” progression
    - Asynchronous and loosely synchronous model
  - Standardized in MPI-3.0
- AM++
  - Support for Active Messages
  - Generic C++, vectorizable handlers!
  - Full functionality (e.g., comm. from handlers)

Example System/Utility Libraries

- LibTopoMap (Topology Mapping)
  - Supports scalable topology mapping for MPI-1.0
  - Provides new comm. with optimized rank order
  - User needs to re-distribute data
  - Standardized in MPI-2.2
- HDF5
  - Abstract data model for storing and managing data
  - Heavily uses datatypes and MPI-IO

The HDF Group

MPI and Libraries (The Good)

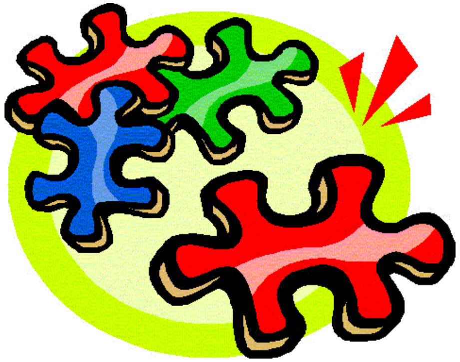
- Communication Contexts
  - Spatial and temporal isolation “comm. privatization”
  - Scope for collective communications
  - → MPI Communicators (and process groups)
- Virtual Topologies
  - Domain-specific process naming
  - Extends the one-dim. naming of process groups
    - Arbitrary Cartesian or general graph

MPI and Libraries (The Good)

- Attribute Caching
  - Associate state with communication objects
    - Communicators, windows, data types
  - Concept of inheritance (copy functions)
- Data types
  - Interface to exchange layouts of data structures
    - Between libraries and users
  - Provide privatization (dup) and (de)serialization

MPI and Libraries (The Good)

- MPI's Modular Design
  - The standard itself is modular
  - Sections can be implemented as separate libraries
    - Collectives
    - Nonblocking collectives
    - Topologies
    - I/O
  - Encourages external communication libraries (e.g., LibNBC)



T. Hoefler et al.: Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI

Where it breaks - initialization (The Bad)

- Imagine:

```
int main() {
  LibA_Init()
  LibB_Init()

  /* use libs */

  LibA_Finalize()
  LibB_Finalize()
}
```

```
LibA_Init() {
  int flag;
  MPI_Initialized(&flag);
  if(!flag) MPI_Init(NULL,NULL);
}
```

```
LibB_Init() {
  int flag, reqd=MPI_THREAD_MULTIPLE, p;
  MPI_Initialized(&flag);
  if(!flag)
    MPI_Init_thread(NULL,NULL,reqd,&p);
}
```

Where it breaks - initialization (The Bad)

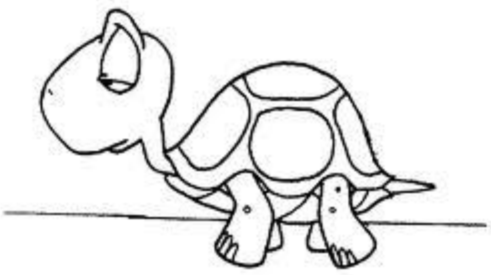
- Imagine:

```
int main() {
  LibB_Init()
  LibA_Init()

  /* use libs */

  LibA_Finalize()
  LibB_Finalize()
}
```

```
LibA_Init() {
  int flag;
  MPI_Initialized(&flag);
  if(!flag) MPI_Init(NULL,NULL);
}
```



```
LibB_Init() {
  int flag, reqd=MPI_THREAD_MULTIPLE, p;
  MPI_Initialized(&flag);
  if(!flag)
    MPI_Init_thread(NULL,NULL,reqd,&p);
}
```

Where it breaks – info objects (The Bad)

```
int main() {
  MPI_Info info; /* =no_locks; */
  MPI_Win_create(..., info, comm, &win);
  /* One-Sided Communication */
  LibA_BuildOctTree(win, comm);
  MPI_Win_free(&win);
}
```

```
void LibA_BuildOctTree(win, comm) {
  MPI_Win_lock(type, 0, 0, win);
  /* One-Sided Communication */
  MPI_Win_unlock(0, win);
}
```

- MPI_INFO
  - Info key/value pairs can be attached to several objects (e.g., windows)
  - Influences performance or correctness
- Requires at least an info query mechanism!

Reentrant Libraries (The Bad)

```
int main() {
  /* init */
  int tid, bsize=N/num_threads;
  LibA_Init();
  #pragma omp parallel private(tid)
  {
    tid = omp_get_thread_num();
    LibA_CalcRange(tid*bsize,
          (tid+1)*bsize, comm);
  }
  LibA_Finalize();
}
```

```
void LibA_Init() {
  int flag;
  MPI_Initialized(&flag);
  if(!flag) MPI_Init(NULL,NULL);
}
```

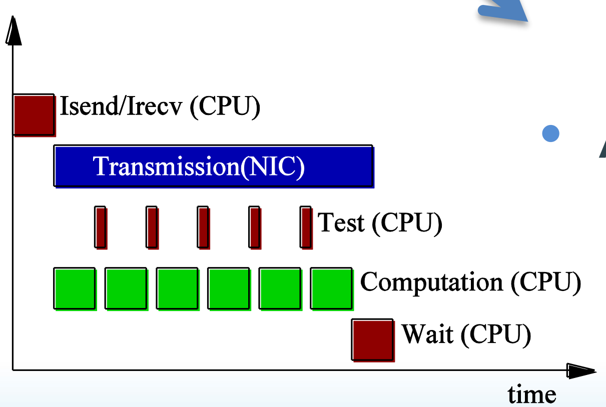
```
void LibA_CalcRange(begin,
          end, comm) {
  /* init and calculate */
  MPI_Allreduce(..., comm);
}
```

Reentrant Libraries (The Bad)

- Libraries create their private communication context
  - Allows for only one invocation per communicator
  - → nonreentrant libraries
- Techniques to make them reentrant
  - Barrier/lock before and after invocation
  - Several dup'd communicators (cf. stack)
  - Special messaging protocol
    - No wildcards, no cancel

Nonblocking Library Progress (The Bad)

```
int main() {
  /* init */
  LibA_Init();
  LibA_Icomm(tid*bsize,
     (tid+1)*bsize,  comm, &handle);
  /* independent computation */
  LibA_Wait(&handle);
  LibA_Finalize();
}
```



- Manual progress
  - User transfers control
    - Progress call!
  - Supported by global progression rule in MPI
- Asynchronous progress
  - No user interaction, finishes autonomously

T. Hoefler, A. Lumsdaine: Message Progression in Parallel Computing - To Thread or not to Thread?

Nonblocking Library Progress (The Bad)

- MPI has a global progress rule
  - Libraries need progress, elegant to hook into MPI
  - Generalized requests associate MPI requests with library state (good!)
  - BUT: require asynchronous libraries in MPI-2.2 (bad!)
- Simple solution discussed in EuroMPI'07
  - Define a user-progress function to be called by MPI
  - [still no proposal for MPI-3.0?]

R. Latham et al.: Extending the MPI-2 Generalized Request Interface

Nonblocking Libraries – init (The Bad)

- Blocking Comm_dup
  - Cannot implement fully nonblocking library!

```
void LibA_Icomm(begin, end,  comm, &handle);
 /* initialize */
 MPI_Attr_get(comm, keyval, &mycomm, &flag);
 if(!flag) {
  MPI_Comm_dup(comm, &mycomm);
  MPI_Attr_put(comm, keyval, mycomm);
 }
}
```

  - Ugly fix: initialize library for each communicator ☹

T. Hoefler et al.: Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI

Complex Communications (The Bad)

- User-defined collective reductions
  - Cannot handle user-defined operations!
  - Fixed in MPI-2.2 (reduce_local)
- Limited tag-space
  - Library must only support 32k tags
  - Stacked libraries may want to use sub-space of tags!
  - Hard to implement “MPI-compliant” libraries!

Complex Communications (The Bad)

- Quiz: what's wrong with this code:

Rank 0	Rank 1
MPI_Send(..., 1, 99, comm);	MPI_Recv(..., 0, 99, comm);
OF_Recv(...);	OF_Send(...);

Complex Communications (The Bad)

- MPI_Send may not send immediately!

Rank 0	Rank 1
MPI_Send(..., 1, 99, comm);	MPI_Recv(..., 0, 99, comm);
OF_Recv(...);	OF_Send(...);

- Synchronization outside of MPI
  - Good source of deadlocks (missing MPI progress)
  - E.g., if libraries are tuned for low-level transports

T. Hoefler, A. Lumsdaine: Optimizing non-blocking Collective Operations for InfiniBand

Other Issues (The Bad)

- No const-correctness
  - No specified contracts for C bindings
- Cannot nest split file I/O
  - What if a library already started an operation?
  - Cf. Edgar's talk on nonblocking I/O on Monday!
- Finalize can only be called once
  - MPI_Initialized() does not suffice
  - Race-conditions for multi-threaded libraries!
  - Solution: ref-counting (proposal for MPI-3)

Hybrid Programming (The Ugly)

- Is this correct?

Rank 0	Rank 1
MPI_Send(..., 1, 99, comm); →	MPI_Recv(..., 0, 99, comm);
upc_all_reduceD(...); ↔	upc_all_reduceD(...);

- What about the following?

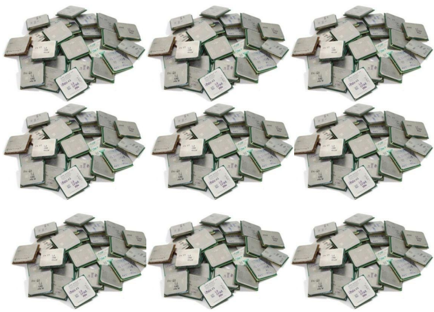
Rank 0	Rank 1
MPI_Bcast(..., 0, comm); →	MPI_Bcast(..., 0, comm);
upc_all_reduceD(...); ↔	upc_all_reduceD(...);

Hybrid Programming (The Ugly)

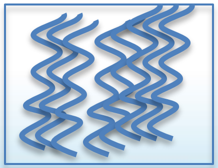
- Mixing MPI with other programming models is rather unspecified
  - Seems straight forward
  - Dangerous (and rare) pitfalls
  - → looks harmless but is dangerous!
- Often conservative programming model
  - Barrier, switch model, barrier, slow
- Complex interaction with threads

MPP

Many Core



Many Thread



J. Dinan et al.: Hybrid Parallel Programming with MPI and Unified Parallel C

Thread-safe Message Probing (The Ugly)

- Probe is important for dynamic applications
  - E.g., active messages in message-driven algs.
- Issues with threading (discussed last year)
- Two threads can probe/receive concurrently
- Shared “MPI state” leads to wrong matching
- Fix on the way for MPI-3.0 (passed)
  - See EuroMPI’10 publication
  - Was hard to communicate, even to the experts!

T. Hoefler: Efficient MPI Support for Advanced Hybrid Programming Models

Control Transfer (The Ugly)

- Threaded libraries may consume PEs
  - Potentially shared with application threads
- How is control passed to a threaded library?
- Four scenarios:
  1. ST app calls ST lib (trivial)
  2. ST app calls MT lib (library is only consumer)
  3. MT app calls ST lib (requires synchronization)
  4. MT app calls MT lib (requires resource management)

Thread Resource Management

- State of the art:
  - Ad-hoc: Query the number of CPUs and pin threads
  - OS: time sharing (thread scheduling, low performance)
- Library issues:
  - Space sharing (one library may not "own" all cores)
  - How to broker resources (cores) among all clients?
    - E.g., polling threads vs. compute threads
- OS-based core allocation (e.g., Lithe)

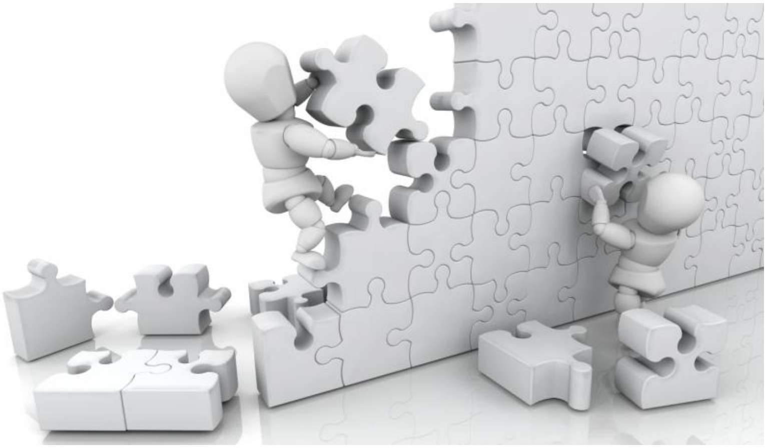
H. Pan et al.: Lithe: enabling efficient composition of parallel libraries

Communication Endpoints (A Solution)

- Observation:
  - Running one MT MPI process per node cannot exploit full communication potential
  - But shared memory is useful
- Solution (MPI-3.0 proposal):
  - Introduce multiple MPI endpoints per process
  - Threads can “grab” endpoints
  - MPI-3.0 endpoints act like MPI-2.2 processes

M. Snir: Endpoints Proposal for MPI-3.0

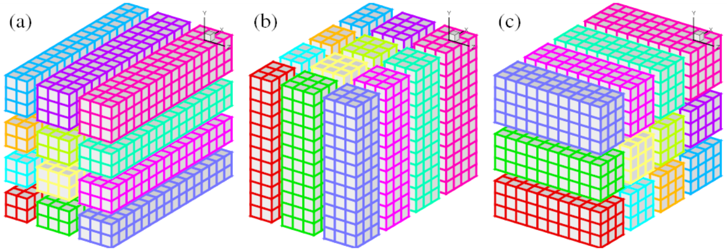
Library Developer's Best Practices



- Use communicators for:
  - Message privatization
  - Spatial decomposition
  - State caching (attributes)
  - Passing library state (exclusively)
- Handle (MPI) errors internally (error handlers), provide library-specific messages
- Initialization can be done explicitly or implicitly
  - Dup has issues with nonblocking libraries!

Do's and don'ts!

- Don't use MPI_COMM_WORLD
  - Hinders future extensions / avoid globals!
- Don't synchronize at entry/exit
  - Costs performance
- Use overlapping communicators if necessary!
  - E.g., 2D-decomposed FFT
- Think about progress
  - "Manual" vs. "asynchronous"



Thanks and Summarizing!

- Modular software development is important
  - Isolate end-users from MPI-complexity (datatypes, topology mapping, …), cf. DSL
- MPI offers good support (other programming environments/languages need to learn)
  - Some environments are lacking
  - Some MPI facilities are dangerous
- But: common pitfalls
  - May be addressed in MPI-3.0 (join us!)
  - Standardize best practices!
  - Come to IMUDI'11! ☺

Collaborators, Acknowledgments & Support

- Thanks to:
  - Bill Gropp, Marc Snir

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

- Sponsored by

BLUE WATERS
SUSTAINED PETASCALE COMPUTING

Sandia National Laboratories