A new Approach to MPI Collective Communication Implementations

T. Hoefler[1,2], J. Sqyures[3], G. Fagg[4], G. Bosilca[4], W. Rehm[2], A. Lumsdaine[1]

[1]Open Systems Lab
Indiana University

[2]Computer Architecture Group
Technical University of Chemnitz

[3]Cisco Systems
San Jose

[4]University of Tennessee
Dept. of Computer Science

2nd Austrian Grid Symposium - DAPSYS'06

Innsbruck, Austria, 22nd September 2006

Outline

Outline

Known Problems

- huge number of different collective algorithms and implementations
- hardware-dependent collective implementations
- no framework that offers run-time selection exists
- selection of optimal algorithm not trivial, because
  - depends on MPI-parameters (size, comm)
  - decision in critical path
  - different implementations only work for certain parameters
  - every process has to chose the same (runtime-decision)

Predictive Performance Models

Prediction is Possible

LogP-Family (LogGP) predicts accurately

- **L** hardware latency
- **o** host overhead (can be divided into o_r and o_s)
- **g** gap between consecutive messages (bw limiting)
- **G** gap between each byte of a message
- **P** number of processes

Collective Operations

All collective operations based on point-to-point messages can be predicted with Log(G)P!

Predictive Performance Models

Prediction is Possible

LogP-Family (LogGP) predicts accurately

- L hardware latency
- o host overhead (can be divided into o_r and o_s)
- g gap between consecutive messages (bw limiting)
- G gap between each byte of a message
- P number of processes

Collective Operations

All collective operations based on point-to-point messages can be predicted with Log(G)P!

Further Problems

- LogP vs. HW optimized implementations
- $\Rightarrow$ need common denominator
- seconds to assess running time
- HW implementations have to offer predictive models
- bypassing must be possible (optimized impl.)

State of the Art

- most impl. use suboptimal hard-coded switching points (MPICH(2), MVAPICH, LAM/MPI, Open MPI)
- "tuned" Open MPI component experiments with dynamic selection with a fixed set of algorithms (no HW optimization)
- Open MPI allows coarse grained third party coll modules
- $\Rightarrow$ no flexible selection framework available yet

Open MPI

⇒ merged FT-MPI, LA-MPI, LAM/MPI, PACX-MPI

- implements MPI-2
- support for different networks (TCP, GM, MX, MVAPI, OpenIB, Portals, SM)
- modular framework architecture
  - some frameworks: PML, BTL, COLL ...
  - easy addition of new ideas
  - clearly defined interfaces
  - binary modules (vendor)

Goals of our Design

⇒ redesign of collv1 framework in Open MPI 1.0/1.1

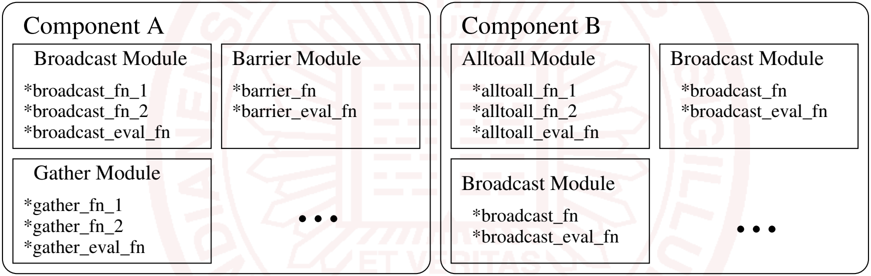
- enable fine-grained selection
- efficient run-time decision
- bypassing/fast-pathing
- modular approach/third party (binary) modules
- automatic usage of best available module

Outline

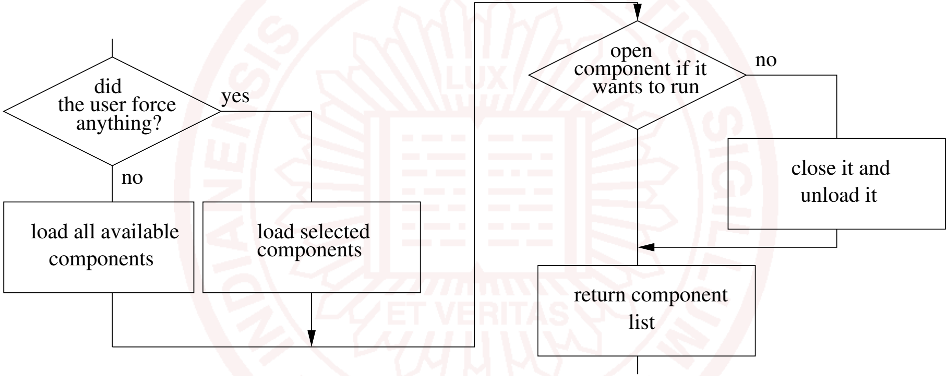
Terms

- **component** functionality without resources provided by implementer
- **module** communicator specific instance of a implementation
- **query** request to a component to return comm specific modules
- **implementation** implementation of a collective operation
- **opaque functions** non-visible functions in coll. modules

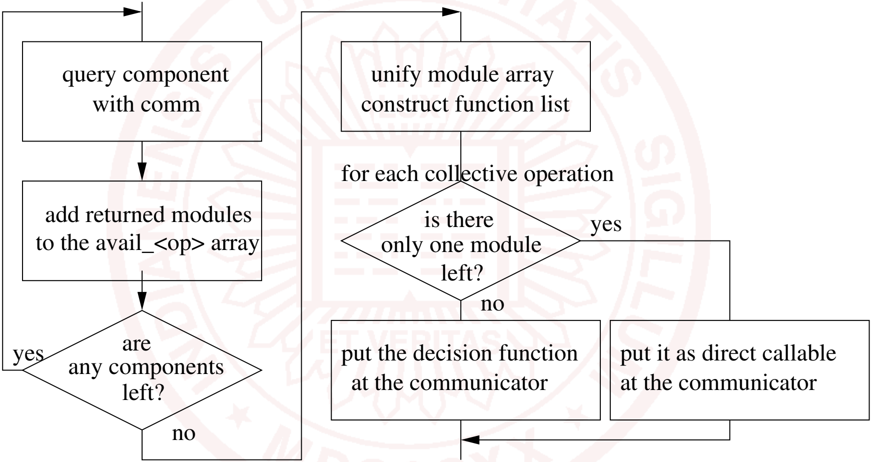
Software Architecture



Actions During MPI_INIT



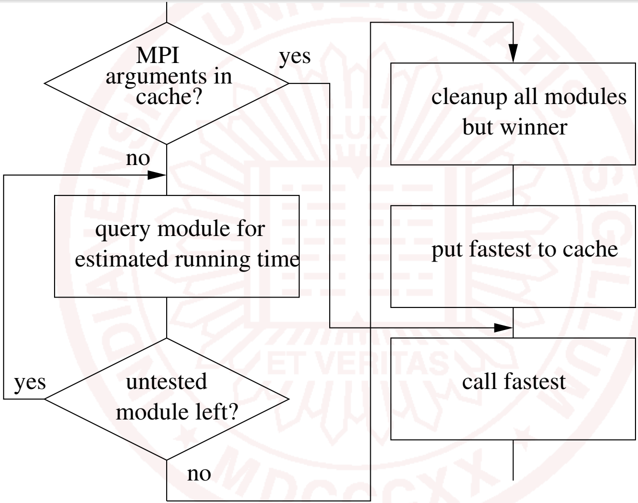
Actions During Communicator Construction



Architecture

- all returned modules are attached to the communicator
- each module offers an evaluation function
- eval. function returns pointer to fastest function and an estimated time
- estimation up to implementer (model, previous benchmark, ...)

Invocation


MPI arguments in cache?
yes
no
query module for estimated running time
yes
untested module left?
no
cleanup all modules but winner
put fastest to cache
call fastest

Decision Overhead

Cache Hit

access in a hash-table

Cache Miss

depends on number of modules

- query each module
- returns model or benchmark-based prediction

Cache Friendliness

- ABINIT/Band: 295/16 (94.6%)
- ABINIT/CG: 53887/75 (99.9%)
- CPMD: 15428/85 (99.4%)

Decision Overhead

Cache Hit

access in a hash-table

Cache Miss

depends on number of modules

- query each module
- returns model or benchmark-based prediction

Cache Friendliness

- ABINIT/Band: 295/16 (94.6%)
- ABINIT/CG: 53887/75 (99.9%)
- CPMD: 15428/85 (99.4%)

Outline

Conclusions and Future Work

Conclusions

- easy, flexible and reliable scheme
- optimized for common case
- uses ”argument-locality”

Future Work

- implement system in Open MPI
- analyze more applications

Conclusions and Future Work

Conclusions

- easy, flexible and reliable scheme
- optimized for common case
- uses ”argument-locality”

Future Work

- implement system in Open MPI
- analyze more applications