Lessons learned from MPI

Patrick Geoffray
Opinionated Senior Software Architect
patrick@myri.com
GM design

• Written by hardware people, pre-date MPI.

• 2-sided and 1-sided operations:
  – All asynchronous.
  – All contiguous only.
  – All operates on registered buffers.

• Matching on 5+1 bits.

• Receive-ready required:
  – If receive not ready -> nack -> try again with exponential back-off.

• Thread-safe but not thread-safe.

• No support for high level protocol progression.
MPICH1-GM stack

- Applications
- MPI API
- Convenience MPI functions
- ADI
- CH2
- CH_GM
- CH_MPL
- CH_P4
- CH_NX
- GM
- shmem/self
EAGER protocol

- Pools of pre-allocated, pre-registered, aligned and identical send buffers and receive buffers.
- Memory copy to a send buffer, send to posted receive buffer, memory copy to the user-space buffer when appropriate.
small messages: EAGER protocol

node A

mpi_send

gm_send

Network

node B

MPI_recv complete
or
unexpected

size \leq 32k
small messages: EAGER protocol

node A

mpi_send

gm_send

Network

node B

MPI_recv complete or unexpected

gm-ack

size <= 32k
large messages : RENDEZ-VOUS protocol

- Synchronization via exchange of small GM messages.
- Memory registration on sender and receiver sides.
- One-sided communication for data.
large messages: RENDEZ-VOUS protocol

node A

mpi_send

gm_send

gm_put

Network

enveloppe

RTS

DATA

done notify

node B

gm_send

MPI_recv complete

size >= 32k
large messages: RENDEZ-VOUS protocol

node A

mpi_send

gm_send

gm_put

Network

enveloppe

RTS

data

done notify

node B

gm_send

MPI_recv complete

www.myri.com

© 2005 Myricom, Inc.
Very large messages: RENDEZ-VOUS protocol

- Pipelined to overlap registration and communication.
Registration

- Zero-copy relies on DMA/physical addresses being known for user area
- Message buffers should be gm_registered (costly)
- No explicit registration in MPI, registration reuse is up to implementation:
  - pin memory for each communication
  - or do on-demand registration: maintain a registration cache
    requires tracking address space changes: malloc(), sbrk(), mmap() overloading and tricks.
- The registration cache efficiency can be a major factor in overall app performance
GM Design review (MPI)

- Connectionless.
  - Native shared receive queue. Good !

- Explicit memory registration.
  - Assume overhead out of critical path. Bad !

- Receive-ready.
  - Assume application cooperation. Bad !

- In-order delivery.
  - Last byte written last. Bad !

- Reliability at NIC level.
  - NIC-level ack/timer/resend. Good and Bad !
Myrinet Express

• Low level interface for 3rd and 4th generation Myrinet hardware:
  – MX-2G: D, E, F NICs
  – MX-10G: 10G NICs

• Key design elements:
  – Low latency / low overhead.
  – Optimized for MPI.
  – No explicit registration.
  – Support for progression.
  – Fault-tolerant
  – Extensible (collective communications, etc).
**MX protocol**

- MX packet headers contain:
  - 64-bit matching information: no need for MPI encapsulation.
  - Source id/destination id: drop misrouted packets.
  - Session id: protect jobs, basic security.
  - Process-to-process sequence number: can receive out of order at the NIC level.
  - Message type:
    - Tiny
    - Small
    - Medium
    - Medium pipelined
    - Large

- MX posted receives contain:
  - 64-bit matching information/
  - 64-bit matching mask.
Tiny messages

Driver

Lib | App
---|---
Request | Data

PIO

 NIC

Fabric

Driver

Lib | App
---|---
Copy | Data

DMA

NIC
Small messages

Driver

<table>
<thead>
<tr>
<th>Lib</th>
<th>App</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Data</td>
</tr>
</tbody>
</table>

PIOs

NIC

Receive queue

Event queue

DMAs

Copy

Data

Fabric

NIC

www.myri.com

© 2005 Myricom, Inc.
Medium messages

Driver

Lib

App

Copy

Request

Data

PIO

DMAs

Send queue

NIC

Receive queue

Event queue

Fabric

Driver

Lib

App

Copy

Data

DMAs

NIC

© 2005 Myricom, Inc.
Large messages

Driver

Lib

App

NIC

Fabric

Registration

Rendezvous

Data

Matching

Buffer

www.myri.com

© 2005 Myricom, Inc.
Large messages
MPI communication (non) overlap

MPI_ISend

Computation

MPI_Wait

Polling

Request

Ack

Data

Ack

Comp.

Polling

MPI_Wait

MPI_IRecv

Receiver

Sender
Design trial and error: matching in the NIC

• Pros:
  – NIC is always ready: perfect overlap of communication and computation for large (rendez-vous) messages.
  – Offload matching overhead.

• Cons:
  – Need to re-order messages: MPI matching is always in order!
  – Need to scan list of posted receives linearly:
    • Wildcards
    • 0.5 us overhead for empty receive list.
  – Need to have a single point of matching:
    • shared memory device ≠ network device
  – Need to post the receives close to the matching point:
    • Atomicity of list insertion/removal ⇒ over PCI bus
MX design

• Progression thread: helper thread to provide host cycle when the application is not calling MX functions
  – MX is thread-safe
  – Overlap communication with computation for large messages (without affecting small messages performance): once per 10ms max, only for rndv-messages.
  – handle resources starvation situations.
  – Do not require explicit host involvement to progress the protocol (MPI requirement).

• NIC is stateless and connectionless:
  – Can reboot NIC while application is running (with ack in the lib)
  – Native shared receive queue.
  – Do not require on-demand connection allocation scheme.
MX design

• Ack at the lib level: move the reliability from the NIC to the host
  – Piggybacking of acks on return traffic.
  – Acks aggregation (reordering of messages in the host)
  – Flow control (throttle fast sender against slow receiver)
  – Can add checksum computation for end-to-end (application-to-application) reliability.

• No explicit memory registration:
  – Memory registration is in the critical path!
  – Tricky to avoid (rcache, tricks, etc)
  – Low-overhead implementation possible.
  – Non-pipelined overlap possible.

  – User-level RDMA is not worth it.
MX design

• In-order matching but not delivery:
  – Can receive out of order from hardware.
  – Reordering by the host (lot of cycles, lot of buffers)

• Native support for unexpected messages:
  – It happens a lot.
  – Reduce unexpected overhead (optimistic storage).
MPICH-MX

- MPI_Isend => mx_isend
- MPI_Irecv => mx_irecv
- MPI_Issend => mx issend
- MPI_Test => mx_test

- One-to-one correspondence between MX messages and MPI-messages
- MPI layer still adds communicators, datatypes, collectives
- => Having all the core functionality done previously by MPICH-GM directly inside the MX core allows more flexibility in the implementation and better performance for a lot of tasks.
MPICH1-GM stack
MPICH1-MX stack

- applications
- MPI API
- convenience MPI functions

- CH_MX
- ADI
- CH2
- MX
- CH_MPL
- CH_P4
- CH_NX
Route dispersion (I)

- CLOS topology: multiple paths between peers (for each bisection, there is a set of routes that is contention-free).
- Route dispersion: use multiple routes when sending messages to the same destination.
Route dispersion (II)

• Route dispersion statistically decreases contention in the fabric: traffic is spread on multiples links, probability of link sharing is less.
  – Route dispersion does not improve point to point bandwidth if no contention.
• Route dispersion may hide bad links: spreading the packets on several paths increases robustness to single link failure.
• Route dispersion creates disorder: packets can pass each other when sent on different paths. Bad if firmware expects order to detect packet loss, or to deliver messages in order to the host.
• Route dispersion dramatically increases the size of the route table.
• Route dispersion uses routes given by the mapper/FMS: good route selection and load balancing still very important.
Route dispersion (III)

- **Round-robin** route dispersion: switch to a new path for a particular destination for every fragment/packet sent on the wire.
  - Pros: simple to implement, somewhat deterministic, link failure resilience.
  - Cons: create disorder, statistic contention reduction is average.

- **Adaptive** route dispersion: switch to a new path for a particular destination only if contention is detected by back-pressure.
  - If packet takes longer than normal to be injected in the fabric.
  - Pros: best contention reduction, limit disorder, fall back on round-robin scheme on worst case.
  - Cons: no link failure resilience, harder to implement.
Conclusion

• MPI momentum is huge:
  – Do not port MPI on top of interconnect.
  – Port interconnect under MPI.

• Scalability is not hard:
  – basic rules.

• Not a link problem:
  – Throwing bandwidth at it is not a solution, it’s makeup.
Myri-10G NICs

These are 10-Gigabit Ethernet NICs. Use them with your favorite brand of 10-Gigabit Ethernet switch and Myricom’s bundled driver, and you will see 9.6 Gbits/s (netperf) TCP/IP data rate (Linux, Opterons). Jumbo frames are supported.

These NICs are 8-lane PCI Express, all based on the Myricom Lanai-Z8E chip.

10GBase-CX4
10GBase-R
XAUI over ribbon fiber

These are 10-Gigabit Myrinet NICs. When connected to a 10G Myrinet switch, and when using MX software, you will see performance metrics of:

- 2.2µs MPI latency
- 1.2 GBytes/s data rate
- Very low host-CPU utilization

Myricom
www.myri.com
© 2005 Myricom, Inc.