Optimizing data transfers with the NewMadeleine multithreaded communication engine

Pr. Raymond Namyst
Runtime Team
INRIA
Bordeaux 1 University
• Brief Overview of Runtime activities
  - Designing efficient runtime systems for parallel architectures

• The NewMadeleine Communication Engine
  - Going beyond the traditional “send/recv” paradigm
  - Introducing opportunistic, aggressive optimizations
  - Exploiting heterogeneous multi-rails configurations

• Integration into Higher-Level Middleware
  - The PadicoTM Grid middleware
  - Preliminary integration with MPICH2 and YAMPI
About the RUNTIME Team

• Designing efficient runtime systems…
  = approaching hardware’s performance
  …while preserving applications’ portability
  i.e. portability of performance

• Research directions
  – Multithreading on Hierarchical NUMA Multiprocessors
    • Application-guided, topology-aware thread scheduling
    • Marcel, BubbleSched, OpenMP
  – Fast Communications over High Speed Networks
    • Adaptive, “just-in-time” data transfer optimizations
    • NewMadeleine, MPI
  – Communication over Large Scale Grids
    • Efficient, multi-paradigm, multi-protocol communications
    • PadicoTM
The PM2 Software Suite

- POSIX Thread Compatibility layer
- PadicoTM Communications for Grids
- MPICH/Mad Multi-protocols implementation
- MARCEL/BubbleSched Thread scheduler
- NewMadeleine Communication library

- IA32
- IA64
- PPC
- Sparc
- Myrinet
- SCI
- IB
- QD

- GPL License, available on INRIA GForge
  - http://gforge.inria.fr/projects/pm2/
  - http://gforge.inria.fr/projects/padico/
  - http://gforge.inria.fr/projects/mpich-mad/
The NewMadeleine Communication Engine
Beyond strict send/recv interfaces

• Send/Recv oriented interfaces are very good for applications where
  – Communication schemes are regular
    • No unexpected messages
  – Structure of messages is simple
    • No internal dependencies

• Some applications/environments have more requirements
  – Multi-segments messages with dependencies
    • RPC-like interactions (Client/Server applications, SDSM, …)
  – Multiple simultaneous communication flows
    • Composite applications
  – Heterogeneous, multi-rail configurations
    • Bandwidth-oriented applications

• New opportunities for optimization!
Optimization strategies have to be decided at run time

- The “best” optimization strategy depends on
  - the characteristics of the underlying network (and driver)
    - Latency
    - Performance of PIO & DMA transfers
    - Gather/Scatter capabilities
    - Remote DMA capabilities
    - Driver limitations
    - Etc.
  - host architecture
    - Performance of memory copies
    - Performance of the I/O bus
  - applications’ requirements!
    - Communication/computation overlap vs raw performance
    - Etc.
Implementing data transfers efficiently

Transfer time

Message size

PIO + copy

DMA + copy

DMA + RdV
Implementing data transfers efficiently

Transfer time

- PIO + copy
- DMA + copy
- DMA + RdV

Message size
Implementing data transfers efficiently

Sending the 3 chunks sequentially = t₁ + t₂ + t₃
(if no pipelining effects)
Implementing data transfers efficiently

This second strategy is better if
\[ t_1 + t_3 > t_4 + k.(\text{sizeof(chunk 1)} + \text{sizeof(chunk 3)}) \]
The NewMadeleine communication engine

Objectives

- Provide an engine capable of efficiently multiplexing multiple communication flows over multiple networks
  - Centralized view of communication traffic

- Enable aggressive optimizations to be implemented
  - Out-of-order data transfers, coalescing, …

- Think the optimization of communications as the scheduling of threads
  - Packet scheduling directed by NIC activity

- Provide a extensible set of optimization strategies
  - Experiments with various packet scheduling policies

- Enable various API to be implemented
  - send/recv, pack/unpack, MPI
• Application submits requests
  - Non-blocking operation

• Meta-data are associated to data
  - Reordering constraints
  - Tag, seq number, etc

• User communication interfaces
  - Incremental message building interface
  - Send-receive interface
  - Subset of MPI routines: Mad-MPI
Architecture overview: the network layer

- Synchronization with NIC activity
  - Busy NIC → simply gather application requests
  - Idle NIC → invoke the scheduler
    - Analysis of the user request backlog
    - Application of a policy
    - Synthesis of a network request
- Natural building of the backlog
- Other approaches
  - Synchronization with application
  - Optimization right upon the application call
Architecture overview: the packet scheduler

- Provides the next ready-to-send packet
- **Tactic**: basic operation
  - Aggregation, split, reordering, etc
- **Strategy**: combination of tactics
- Combined with any network protocol

In the future:
- Evaluate and compare each strategy
- Select the best one
Evaluation

- Mad-MPI
  - Lightweight implementation of MPI over NewMadeleine
  - Myrinet, Quadrics, Infiniband, SCI

- Hardware platform
  - Dual-core opteron 1.8GHz
  - 1MB L2 cache, 1GB main memory
  - Linux 2.6.17
  - MX/Myri-10G network
  - Elan/QsNet II Quadrics network

- Challenger implementations
  - MPICH-MX, MPICH-Quadrics, OpenMPI-MX 1.1
Overhead on top of Myri-10G
Raw point-to-point ping-pong

Latency: 2.7 µs
Overhead: about 0.5 µs worse than MPICH
up to 0.5 µs faster than OpenMPI

- Cost of added headers
- Cost of the optimizer on a single request

Bandwidth: 1155 MB/s
Loss: Equivalent to MPICH
and 35% better than OpenMPI
Overhead on top of Quadrics
Raw point-to-point ping-pong

Latency: 1.9µs
Overhead: Less than 0.5µs

- Same reasons
  - Added headers
  - A single optimized request

Bandwidth: 835 MB/s
Loss: Equivalent
Aggregation of messages
Ping-pong of different channel segments

On top of Myri-10G
- 8 chunks

- Neither MPICH nor OpenMPI aggregate messages (but pipelining is efficient)
- Using aggregation strategy, NewMadeleine combines the data
- Up to 70% faster

- 16 chunks

- NewMadeleine combines the data
- Up to 70% faster
Aggregation of messages
Ping-pong of different channel segments

On top of Quadrics

- 8 chunks
  - Up to 50% faster

- 16 chunks
Implementing multi-rail support

- **Long fragments**
  - “Heterogeneous” splitting
  - The split ratio is computed according
    - Performance of each network
    - Estimated availability of NICs
  - Network sampling is necessary

- **Short fragments**
  - Aggregation over the fastest NIC
  - Should be balanced
    - Only if multiple cores are available!
Aggregating bandwidth for large segments through adaptive stripping

Taking the performance of each network into account

![Graph showing bandwidth vs. data size]

- **Heterogeneous stripping**
- **Iso stripping**
- **MX only**
- **Quadrics only**
The need for a sophisticated I/O manager

• **Issues**
  - Asynchronous progression of communication
    - Background handling of Rendez-vous transactions
    - Offloading on idle cores
  - Reactivity to high priority I/O events
    - Polling vs Interrupts, low overhead priority management
  - Support of multithreaded middleware

• **PIOMan: the PM2 I/O event manager**
  - manages automatically I/O events
    - uses interrupt and/or polling transparently
    - makes communication progress asynchronously (no thread needed in application)
    - well integrated with threads scheduler (Marcel)
  - available in multiple flavors: no thread / single CPU multithreading / SMP
Integration in higher-level Middleware
The PadicoTM Framework

- Supports various middleware
  - MPI: MPICH, OpenMPI, YAMPI
  - CORBA: omniORB, Mico
  - SOAP, Java VM, DSM, HLA, JXTA, ...
The PadicoTM Framework

- Supports various communication methods
  - API: virtual sockets, Madeleine API, FM
  - filters: compression (LZO, ZIP, BZIP), encryption/authentication (TLS/SSL)
  - connectivity: routing, tunnelling, automatic SSH tunnels, TCP splicing, etc.
The PadicoTM Framework

- Supports various networking technologies
  - Myrinet (MX, GM), Infiniband (OpenIB), QsNet, SCI, TCP/IP
YAMPI over PadicoTM

- **PadicoTM-enabled YAMPI**
  - Make YAMPI take benefit from PadicoTM networking facilities
  - Preliminary prototype
    - Based on YAMPI2 (CVS)
    - Seen as a driver by YAMPI (small: 400 lines of code)
    - “Reasonable” performance, but has obviously to be improved
      - e.g. Infiniband: 8µs, 1300MB/s (Grid’5000@Bordeaux)

- **Scientific challenges**
  - Interaction with multithreading
  - Reactivity and asynchronous communication progress
  - Topology-aware collective communications
MPICH2 over NewMadeleine

- The preliminary version was straightforward to implement
  - MPI_xxx point to point communication are (almost) directly mapped on nmad_xxx
  - The NEMESIS is used for intra-node (shared memory) communication (joint work with Argonne NL)

- Performance is good
  - Latency is only less that 1\(\mu\)s higher, bandwidth is the same

- Scientific challenges
  - Optimization of MPI datatypes management
  - Support for collective operations within NewMadeleine
  - Full support of dynamicity over heterogeneous (or multi-rails) configurations
Future work

• Extend the range of optimizations thanks to multithreading
  – Parallel processing of PIO requests for short messages
  – More generally: offloading on idle cores

• Design heuristics to automatically select a good optimizing strategy
  – Compare strategies with “scores”
  – Combine them?

• Make NewMadeleine NUMA-aware
  – Locality of data \textit{wrt} NICs
  – Threads and tasklets scheduling