Optimizing Communication on Blue Waters

Torsten Hoefler

PRAC Workshop, Oct. 19th 2010
“Hottest” Optimizations on Blue Waters

- Serial optimizations (e.g., Vectorization)
- Hybridization (Threads + MPI)
- Communication/Computation Overlap
- Collective Communication (incl. Sparse Colls)
- MPI Derived Datatypes
- Topology Optimized Mapping
- One-Sided (maybe)
In This Talk: Communication Optimization

- Serial optimizations (e.g., Vectorization)
- Hybridization (Threads + MPI)
- Communication/Computation Overlap
- Collective Communication (incl. Sparse Colls)
- MPI Derived Datatypes
- Topology Optimized Mapping
- One-Sided (maybe)

mostly serial
conceptually simple
not clearly defined yet
Is Optimization X Relevant To My Application?

• … at scale? - well, we don’t know
  • If you know that it’s irrelevant: go, have a coffee now 😊

• Three ways to find out
  • Educated Guessing (based on mental model)
    • Very powerful and often accurate
  • Simulation (problematic, will hear more later today)
    • Very accurate but limited
  • Analytic Performance Modeling
    • Relatively accurate, often relatively simple

➢ Excellent middle ground!
High-level Performance Modeling Overview

- Platform or System Model (Hardware, Middleware)
- Application Model (Algorithm, Structure)
- Performance Model
- System Expertise
- Application Expertise
Example 1: 2d FFT

- Relatively simple kernel (square box only)
  - dominated by data movement, computation is free

1. perform $N_x/P$ 1-d FFTs in $y$-dimension ($N_y$ elements each)
2. pack the array into a sendbuffer for the all-to-all (A)
3. perform global all-to-all (B)
4. unpack the array to be contiguous in $x$-dimension (each process has now $N_y/P$ $x$-pencils) (C)
5. perform $N_y/P$ 1-d FFTs in $x$-dimension ($N_x$ elements each)
6. pack the array into a sendbuffer for the all-to-all (D)
7. perform global all-to-all (E)
8. unpack the array to its original layout (F)
Educated Guess: What Matters for 2D-FFT?

- No detailed model available (yet)!
  - Lots of experience and previous analysis!
- Communication/Computation Overlap
  - Suggestion: Nonblocking Alltoall
    - Outside the scope of this talk!
- MPI Derived Datatypes
  - Eliminate Pack/Unpack Phase (>50%)
- Topology Optimized Mapping
  - Only in higher-dimensional decompositions
Example 2: MIMD Lattice Computation

- Gain deeper insights in fundamental laws of physics
- Determine the predictions of lattice field theories (QCD & Beyond Standard Model)
- Major NSF application
- Challenge:
  - High accuracy (computationally intensive) required for comparison with results from experimental programs in high energy & nuclear physics
Model-Driven Optimization: What Matters?

- NCSA’s MILC Performance Model for Blue Waters
  - Predict performance of 300000+ cores
  - Based on Power7 MR testbed
  - Models manual pack overheads
    - >10% pack time
      - >15% for small L
Chapter 2

MPI Derived Datatypes
Quick MPI Datatype Introduction

- (de)serialize arbitrary data layouts into a message stream
  - Contig., Vector, Indexed, Struct, Subarray, even Darray (HPF-like distributed arrays)
    - Recursive specification possible
  - *Declarative* specification of data-layout
    - “what” and not “how”, leaves optimization to implementation (*many* unexplored possibilities!)
  - Arbitrary data permutations (with Indexed)
Datatype Terminology

• **Size**
  • Size of DDT signature (total occupied bytes)
  • Important for matching (signatures must match)

• **Lower Bound**
  • Where does the DDT start
    • Allows to specify “holes” at the beginning

• **Extent**
  • Size of the DDT
    • Allows to interleave DDT, relatively “dangerous”
What is Zero Copy?

• Somewhat weak terminology
  • MPI forces “remote” copy

• But:
  • MPI implementations copy internally
    • E.g., networking stack (TCP), packing DDTs
    • Zero-copy is possible (RDMA, I/O Vectors)
  • MPI applications copy too often
    • E.g., manual pack, unpack or data rearrangement
    • DDT can do both!
Purpose of this Talk

- Demonstrate utility of DDT in practice
  - Early implementations were bad → folklore
  - Some are still bad → chicken+egg problem
- Show creative use of DDTs
  - Encode local transpose for FFT

- Details in Hoefler, Gottlieb: “Parallel Zero-Copy Algorithms for Fast Fourier Transform and Conjugate Gradient using MPI Datatypes”
2d-FFT State of the Art
2d-FFT Optimization Possibilities

1. Use DDT for pack/unpack (obvious)
   - Eliminate 4 of 8 steps
     - Introduce local transpose

2. Use DDT for local transpose
   - After unpack
   - Non-intuitive way of using DDTs
     - Eliminate local transpose
The Send Datatype

1. Type_struct for complex numbers
2. Type_contiguous for blocks
3. Type_vector for stride
   - Need to change extent to allow overlap (create_resized)

- Three hierarchy-layers
The Receive Datatype

- Type_struct (complex)
- Type_vector (no contiguous, local transpose)
  - Needs to change extent (create_resized)
2D-FFT: Experimental Evaluation

- Odin @ IU
  - 128 compute nodes, 2x2 Opteron 1354 2.1 GHz
  - SDR InfiniBand (OFED 1.3.1).
  - Open MPI 1.4.1 (openib BTL), g++ 4.1.2
- Jaguar @ ORNL
  - 150152 compute nodes, 2.1 GHz Opteron
  - Torus network (SeaStar).
  - CNL 2.1, Cray Message Passing Toolkit 3
- All compiled with “-O3 –mtune=opteron”
Strong Scaling - Odin (8000²)

 Scaling stops w/o datatypes

• 4 runs, report smallest time, <4% deviation

Reproducible peak at P=192
Strong Scaling – Jaguar (20k²)

- Scaling stops w/o datatypes
- DDT increase scalability
Negative Results

• Blue Print - Power5+ system
  • POE/IBM MPI Version 5.1
  • Slowdown of 10%
  • Did not pass correctness checks 😞

• Eugene - BG/P at ORNL
  • Up to 40% slowdown
  • Passed correctness check 😊
MILC Communication Structure

- Nearest neighbor communication
  - 4d array → 8 directions
  - State of the art: manual pack on send side
    - Index list for each element (very expensive)
    - In-situ computation on receive side
- Multiple different data access patterns 😞
  - su3_vector, half_wilson_vector, and su3_matrix
  - Even and odd (checkerboard layout)
  - Eight directions
  - 48 contig/hvector DDTs total (stored in 3d array)
- Allreduce (no DDTs, nonblocking alreduce is investigated!)

T. Hoefler : Optimizing Communication on Blue Waters
MILC: Experimental Evaluation

- Weak scaling with $L=4^4$ per process
  - Equivalent to NSF Petascale Benchmark on Blue Waters
- Investigate Conjugate Gradient phase
  - Is the dominant phase in large systems
- Performance measured in MFlop/s
  - Higher is better 😊
MILC Results - Odin

- 18% speedup!
MILC Results - Jaguar

- Nearly no speedup (even 3% decrease) 😞
Chapter 3

Topology Mapping
• LL Topology
  • 24 GB/s
  • 7 links/Hub
  • Fully connected
  • 8 Hubs

Source: B. Arimilli et al. “The PERCS High-Performance Interconnect”
• LR Topology
  • 5 GB/s
  • 24 links/Hub
  • Fully connected
  • 4 Drawers
  • 32 Hubs

Source: B. Arimilli et al. "The PERCS High-Performance Interconnect"
• D Topology
  • 10 GB/s
  • 16 links/Hub
  • Fully connected
  • 512 SNs
  • 2048 Drawers
  • 16384 Hubs

Source: B. Arimilli et al. “The PERCS High-Performance Interconnect”
Topology Mapping

• Some simple observations
  1. A node is a clique with 48 GiB/s
  2. A drawer is a clique with 24 GiB/s
  3. D is faster than LR, but there are more LR links!
  4. Everything else is complicated 😊

• If I were you, I’d let others deal with this mess
  • Specify communication topology to the runtime
    • MPI-2.2 Cartesian or scalable graph communicator
      • Hoefler et. al: “The Scalable Process Topology Interface of MPI 2.2”
    • This is safe, talking with IBM about more options
2D Example: Process-to-Clique Mapping

- Trivial linear default mapping
- With 4 processes per node:
  - 6 internal edges
  - 10 remote edges
- Wrap-around
  - Looses two internal edges
  - Unbalanced communication
Optimized 2D Process-to-Clique Mapping

- Optimal mapping
  - cf. Lagrange multiplier
  - 8 internal edges
  - 8 remote edges

- Similar for 4d mapping
  - 16 cores, linear: 30 internal, 98 remote edges ($L_z > 16$)
  - optimal sub-block: $\sqrt[4]{16} = 2 \cdot 2 \cdot 2 \cdot 2$
  - $\frac{1}{2}$ remote edges
FFT Topology Mapping

- Only useful in 2D (or higher) decomposition
- Map all-to-all communicators onto cliques
  - Node, Drawer, (D-clique?), … not trivial
  - Could specify a fully connected graph topology
    - Not sure if this would work too well (needs experiments)
- Maybe adapt decomposition to network structure
  - Square might not be always optimal
  - Needs information about topology
    - We’re working on this …
Map Irregular Structures

• Both MILC and FFT are very regular

• Many codes (AMR, etc.) are not!
  • Only beneficial if communication pattern is somewhat persistent!
  • The scalable graph topology interface provides opportunities for irregular applications!
  • Helps even more if communication is unbalanced
    • Will map heavy communication to fast links!
Encouraging Simulation Results

• Simulate mapping of Sparse MatVec from UFL collection (nlpkkt240)
  • Heuristic Optimization Technique
  • Reduces Congestion up to 80%
  • Greedy strategy computes mapping in ~0.8s for 1024 cores
Takeaways, Questions & Discussion

• Performance Modeling can guide optimizations!
• Serial optimizations & Overlap are most important
• Derived Datatypes and Topology Mapping are often neglected!
  • They have high potential!
  • But implementations need to improve
• We’re working on this with IBM

Datatype benchmarks: http://www.unixer.de/research/datatypes/
Acknowledgments & Support

• Thanks to (alphabetically)
  • Greg Bauer, Steven Gottlieb, William Gropp, Jeongnim Kim, William Kramer, Marc Snir

• Sponsored by

Datatype benchmarks: http://www.unixer.de/research/datatypes/