Redundant Execution of HPC Applications with MR-MPI

Christian Engelmann and Swen Böhm
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Motivation

• Large-scale 1 PFlop/s systems are here
  – #1 NSCT Tianhe-1A: 2.566 PFlop/s, 186,368 cores
  – #2 ORNL Jaguar XT5: 1.759 PFlop/s, 224,162 cores
  – #3 NSCS Nebulae: 1.271 PFlop/s, 120,640 cores
  – #4 GSIC Tsubame 2.0: 1.192 PFlop/s, 73,278 cores
  – #5 LBNL Hopper: 1.054 PFlop/s, 153,408 cores
  – #6 CEA Tera 100: 1.050 PFlop/s, 138,368 cores
  – #7 LANL Roadrunner: 1.042 PFlop/s, 122,400 cores

• The trend is toward even larger-scale systems
  – End of processor frequency scaling ➔ Node/core scaling
## Proposed Exascale Initiative Road Map
(one of many versions)

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2011</th>
<th>2015</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System peak</strong></td>
<td>2 Peta</td>
<td>20 Peta</td>
<td>100-200 Peta</td>
<td>1 Exa</td>
</tr>
<tr>
<td><strong>System memory</strong></td>
<td>0.3 PB</td>
<td>1.6 PB</td>
<td>5 PB</td>
<td>10 PB</td>
</tr>
<tr>
<td><strong>Node performance</strong></td>
<td>125 GF</td>
<td>200 GF</td>
<td>200-400 GF</td>
<td>1-10 TF</td>
</tr>
<tr>
<td><strong>Node memory BW</strong></td>
<td>25 GB/s</td>
<td>40 GB/s</td>
<td>100 GB/s</td>
<td>200-400 GB/s</td>
</tr>
<tr>
<td><strong>Node concurrency</strong></td>
<td>12</td>
<td>32</td>
<td>O(100)</td>
<td>O(1000)</td>
</tr>
<tr>
<td><strong>Interconnect BW</strong></td>
<td>1.5 GB/s</td>
<td>22 GB/s</td>
<td>25 GB/s</td>
<td>50 GB/s</td>
</tr>
<tr>
<td><strong>System size (nodes)</strong></td>
<td>18,700</td>
<td>100,000</td>
<td>500,000</td>
<td>O(million)</td>
</tr>
<tr>
<td><strong>Total concurrency</strong></td>
<td>225,000</td>
<td>3,200,000</td>
<td>O(50,000,000)</td>
<td>O(billion)</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>15 PB</td>
<td>30 PB</td>
<td>150 PB</td>
<td>300 PB</td>
</tr>
<tr>
<td><strong>IO</strong></td>
<td>0.2 TB/s</td>
<td>2 TB/s</td>
<td>10 TB/s</td>
<td>20 TB/s</td>
</tr>
<tr>
<td><strong>MTTI</strong></td>
<td>days</td>
<td>days</td>
<td>days</td>
<td>O(1 day)</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>6 MW</td>
<td>~10MW</td>
<td>~10 MW</td>
<td>~20 MW</td>
</tr>
</tbody>
</table>
Resilience Issues in Extreme-scale HPC

- Significant growth in component count (up to 50x nodes expected) results in correspondingly higher error rate
- Smaller circuit sizes and lower voltages increase soft error vulnerability (bit flips caused by thermal and voltage variations as well as radiation)
- Hardware fault detection and recovery is limited by power consumption requirements and production costs
- Heterogeneous architectures (CPU & GPU cores) add more complexity to fault detection and recovery
- Power management cycling decreases component lifetimes due to thermal and mechanical stresses
Risks of the Business as Usual Approach

- Increased error rate requires more frequent checkpoint/restart, thus lowering efficiency (application progress)
- Memory to I/O ratio improves due to less memory/node, but concurrency for coordination and scheduling increases significantly (up to 50x nodes, 444x cores)
- Current application-level checkpoint/restart to a parallel file system is becoming less efficient and soon obsolete
- Missing strategy for silent data/code corruption will cause applications to produce erroneous results or hang
Objectives

• Address the deficiencies of recovery-oriented HPC
• Instead of rollback recovery, focus on redundancy
• Center on a software-only approach at the system software layer that is completely transparent
• Aim at redundancy at the Message Passing Interface layer
Related Work (1/2)

• Modular redundancy has been used to ensure availability and reliability in information technology, aerospace and command & control systems for decades

• Our 2009 availability analysis made the case for redundancy in HPC
  – Compute node mean-time to failure (MTTF) can be lowered by a factor of 100-1,000 for dual redundancy and by 1,000-10,000 for triple redundancy without lowering overall system MTTF
  – Compute node MTTF can be lowered by a factor of 1,000-1,0000 for dynamic dual redundancy and by 10,000-100,000 for dynamic triple redundancy
Related Work (2/2)

• Redundant execution with rMPI
  – Using the MPI profiling layer PMPI to intercept all MPI calls from an application and to hide all redundancy mechanisms
  – Similar, parallel effort to MR-MPI that has different features and limitations

• Redundant execution with VolpexMPI
  – MPI library implemented from scratch that supports redundancy in its communication layer
  – Similar, parallel effort to MR-MPI that has different features and limitations

• Redundant Execution with MMPI
  – Recent study of MPI redundancy protocols
Technical Approach

• Aim at MPI redundancy similar to rMPI/VolpexMPI

• Redundant execution of MPI processes:
  – On the same processor
  – On different processors
  – On different compute nodes

• Input replication and output comparison between the MPI library and the app

• Fault model: fail-stop

• MPI/platform independent
Design (1/2)

- Utilize PMPI to intercept MPI calls from the app. and to hide redundancy
- MRMPI is a C library with 53 C/Fortran MPI calls
- To run an application with redundancy, simply:
  - Link it with libmrmpi
  - Execute: `mpirun -np <nprocs> <application> -mrmpi-np <vprocs>`
Design (2/2)

- \( r \times m \) native MPI ranks:
  - \( r \) ranks visible to the app.
  - \( m \): replication degree
  - \(<nprocs> = r \times m\)
  - \(<vprocs> = r\)

- Message replication

- Master-failover for non-determinism, e.g. for MPI_Wtime()

- Partial replication is supported statically
Implementation (1/2)

• MR-MPI core relies on redundant non-blocking p2p communication
  ∵ Tracking of redundant MPI communication requests

• Blocking MR-MPI p2p calls use the non-blocking calls

• Non-determinism:
  ∵ Lowest replica executes first and then tells others
  ∵ Next replica in-line takes over if lowest fails
  ∵ MPI_Wtime(), MPI_Probe(), MPI_Test…()

• Collectives use MR-MPI p2p calls and the recently added MPI_Reduce_local()
  ∵ Linear implementations

10th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN), Innsbruck, Austria, Feb. 15-17, 2011
Implementation (2/2)

- MPI communicators & groups:
  - Fixed mapping of ranks
  - Utilize native MPI calls and maintain mapping
  - MPI_Comm_split(), MPI_Group_intersection(), ...

- Fortran calls use C variants and MPI_..._f2c/c2f()
  - Support for 32-bit Fortran and 64-bit C compiler mix

- GNU autotools-based distribution

- Fully documented source, including Doxygen

- Tested on/with:
  - Mac OS X 10.5 and Linux
  - Open MPI 1.5rc3
  - NPB 3.3.1
  - GNU C and Fortran comp.
Experimental Results

• Test environment #1:
  – 32-node (32-core) cluster
  – Intel Pentium 4 2GHz per node
  – 0.75GB RAM/node
  – Fast Ethernet (100Mbps)
  – Ubuntu 8.04 32-bit Linux
  – Without swap
  – Open MPI 1.5.rc3
  – p2p latency benchmark
  – NAS parallel benchmark suite 3.3.1

• Test environment #2:
  – 16-node (128-core) cluster
  – Two 4-core AMD Opteron 2378 2.4GHz per node
  – 8GB RAM/node
  – Gigabit Ethernet (1Gbps)
  – Ubuntu 10.04 64-bit Linux
  – With swap
  – Open MPI 1.5.rc3
  – NAS parallel benchmark suite 3.3.1
Results: P2P Latency #1

- High impact of message replication on p2p latency
- MRMPI-internal overlap due to non-blocking calls: 4B-4KB
Results: NAS Parallel Benchmark Suite #1

- No overhead for embarrassingly parallel (EP) benchmark
- 70-90% overhead for communication-heavy integer sort (IS) and fast Fourier transform (FT) under DMR
Results: NAS Parallel Benchmark Suite #2

- With increasing per-node core counts, the overhead is increasing as well
- No overhead for embarrassingly parallel (EP) benchmark
Conclusions and Future Work

• Presented the MR-MPI approach for HPC redundancy

• In contrast to rMPI, MR-MPI does not rely on a specific MPI library, such as MPICH

• MR-MPI distinguishes itself from VolpexMPI by not reimplementing the MPI layer

• The performance results clearly show the negative impact of the $O(m^2)$ messages between replicas

• MR-MPI and rMPI recently joined efforts: redMPI

• Recently added output comparison for soft error detection

• Further planned work focuses on better redundancy protocols and on POSIX file I/O under redundancy
Questions?