The Next MPI challenge(s)

George Bosilca
Has MPI really failed?

- Difficult to define a success metric
- Failure metric
  - How many MW were lost due to MPI?
- Success metric
  - How many people get a job based on MPI skills?
- How much breakthrough science came to light due to MPI?
Thread based MPI
MPI Processes

- MPI is process based, threads are external entities outside of MPI knowledge
- Point-to-point communications between threads are possible by crafting special tags
- Collectives are process based, one process participate in the collective once
- Threads fight for messages instead of collaborating
- Different approach than TMPI and AMPI
• What if: **MPI Threads**

• MPI became threads based, i.e. each thread get a rank

• Each thread is allowed to behave as a MPI process today

• We can use a thread based programming approach, mixed with message synchronization and collective communication

• Stay as close as possible to the current MPI standard
What if: MPI Threads

MPI became threads based, i.e. each thread gets a rank.

- Stay as close as possible to the current MPI standard (Nx1 is a standard MPI application)
- **MPI_COMM_WORLD** is still the same

We can use a thread based approach, mixed with standard MPI programming and message synchronization.
MPI_Init_thread

• mpiexec -np NxM ...

• will start N processes and notify them that each will have at most M threads

• Extend the standard with MPI_COMM_LOCAL including all M local threads

• Each thread is required to call MPI_Init_thread to set its rank in the MPI_COMM_LOCAL
MPI ranks

• MPI_COMM_LOCAL is a fully featured intra-communicator
  • process based communicator vs. thread based communicator

• It can be used by any communicator creation function

• If any doubts about the rank of the thread in a communicator creation, the order will be based on the rank in the local communicator.
Receive Rules

• On the process based communicator such as MPI_COMM_WORLD all threads can match a receive

• On all mixed communicators the receive are named by rank (thread)

• Similar rules applies for collective communications, i.e. a process can participate multiple times in a collective.
PLASMA
PLASMA: Tile Algorithms

Block algorithms – LAPACK

Tile algorithms – PLASMA
PLASMA: DAG Scheduling

Cholesky
6 x 6

QR
6 x 6
• acyclic representation of the algorithm as a directed graph with procedures attached to the nodes
• nodes are annotated with the list of input and output parameters
• special node for conditionals, loops and collective

Tiles for QR Factorization
Challenges

- DAG construction and exploration
- initial approach: static partitioning and dynamic scheduling in each sub-domain
- “sliding window” approach
- Dynamic scheduling: trade between data reuse and aggressive pursuit of the critical path
SPMD/MPMD

- Some dependencies will point to local variables, while others point to external data.
- Communications are implicit, and the scheduler can extract them from the DAG.
- Potential for overlapping communications and computations.
Early results

ASMA & ACML BLAS
ML Cholesky
LL Cholesky
PACK & ACML BLAS

QR — quad–socket, dual–core Opteron
PLASMA & ACML BLAS
ACML QR
MKL QR
LAPACK & ACML BLAS

Problem size vs. Gflop/s
**Scheduling: Cholesky**

TBB: nested parallelism

SMPSs:
- arbitrary DAG,
- dynamic scheduling,
- data renaming

Current PLASMA scheduler
The runtime system

- Resource constraints
- Automatic Resource Management
- Asynchronous Task Executions
- Implicit communications
- Collective Communications
- Dynamic multi-level scheduling
- Fault Tolerance
Low level DAG device

- Tasks: send, receive, op
- Horizontal arrow: concurrent execution
- Vertical arrow: sequential execution
- Dash line: multi dependencies
Pipelined Binary Reduce

- Created at the user level
- Executed by the lowest level
- Small overhead
- No interruptions
- Asynchronous
- Report on completion
FT-MPI
Why?

• A lack of fault tolerant programming paradigms

• MPI is the de-facto programming model for parallel applications

• MPI Standard: “Advice to implementors: A good quality implementation will, to the greatest possible extent, circumvent the impact of an error, so that normal processing can continue after an error handler was invoked.”
How?

- Define the behavior of MPI [state] in case an error occurs
- Give the application the possibility to recover from a node-failure
- A regular, non fault-tolerant MPI program will run using FT-MPI
- Follows the MPI-1 and MPI-2 specification as closely as possible (e.g. no additional function calls)
- On error user program must do something (!)
Recovery modes

• ABORT, BLANK, SHRINK and REBUILD

• REBUILD: a new process is created, and it will return MPI_INIT_RESTARTED_PROC from MPI_Init

• BLANK: dead processes replaced by MPI_PROC_NULL, all communications with such a process succeed, they do not participate in the collectives

• two sub-modes: local and global
Communications modes

- **RESET**: the epoch should match in addition to the MPI matching requirements
- **CONTINUE**: only MPI matching
Shallow Water (PSTSWM) & HPL

32 nodes with Gigabit

HPL

PSTSWM
Disks

Diskless

Checkpointing

4 available processors

Add a fifth and perform a checkpoint (Allreduce)

Ready to continue

Failure

Ready for recovery

Recover the processor/data
Diskless Checkpointing

• How to checkpoint?
  • either floating-point arithmetic or binary arithmetic will work
  • If checkpoints are performed in floating-point arithmetic then we can exploit the linearity of the mathematical relations on the object to maintain the checksums

• How to support multiple failures?
  • Reed-Salomon algorithm
  • support p failures require p additional processors (resources)
PCG

- Fault Tolerant CG
- 64x2 AMD 64 connected using GigE

<table>
<thead>
<tr>
<th>Prob #</th>
<th>Size of the Problem</th>
<th>Num. of Comp. Procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>164,610</td>
<td>15</td>
</tr>
<tr>
<td>#2</td>
<td>329,220</td>
<td>30</td>
</tr>
<tr>
<td>#3</td>
<td>658,440</td>
<td>60</td>
</tr>
<tr>
<td>#4</td>
<td>1,316,880</td>
<td>120</td>
</tr>
</tbody>
</table>

Performance of PCG with different MPI libraries:

For each problem, the code is generated and executed every iteration.
PCG

Checkpoint overhead in seconds

<table>
<thead>
<tr>
<th>Time</th>
<th>Prob #1</th>
<th>Prob #2</th>
<th>Prob #3</th>
<th>Prob #4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ckpt</td>
<td>2.6</td>
<td>3.8</td>
<td>5.5</td>
<td>7.8</td>
</tr>
<tr>
<td>2 ckpt</td>
<td>4.4</td>
<td>5.8</td>
<td>8.5</td>
<td>10.6</td>
</tr>
<tr>
<td>3 ckpt</td>
<td>6.0</td>
<td>7.9</td>
<td>10.2</td>
<td>12.8</td>
</tr>
<tr>
<td>4 ckpt</td>
<td>7.9</td>
<td>9.9</td>
<td>12.6</td>
<td>15.0</td>
</tr>
<tr>
<td>5 ckpt</td>
<td>9.8</td>
<td>11.9</td>
<td>14.1</td>
<td>16.8</td>
</tr>
</tbody>
</table>
AFTB concept in example

Perform in parallel:

\[ z = x + y \]

AFTB concept in example

Compute in parallel the checksum of x and y
We're ready to proceed with the sum.
Compute in parallel the sum of $x$ and $y$ simultaneously can compute the of the checkpoint.
ABFT summary

- Relies on floating-point arithmetic
- Exploit the checksum processor
- Stable algorithms exist for any linear operation:
  - AXPY, SCAL (BLAS1)
  - GEMV (BLAS2)
  - GEMM (BLAS3)
  - LU, QR, Cholesky (LAPACK)
  - FFT
\[ A_F = \begin{pmatrix} A \\ C_C^T A \end{pmatrix} \begin{pmatrix} A_C \end{pmatrix} \begin{pmatrix} C_C^T A_C \end{pmatrix} \] and \[ B_F = \begin{pmatrix} B \\ C_C^T B \end{pmatrix} \begin{pmatrix} B_C \end{pmatrix} \begin{pmatrix} C_C^T B_C \end{pmatrix} \]

\[ \begin{pmatrix} A \\ C_C^T A \end{pmatrix} \begin{pmatrix} B \\ B_C \end{pmatrix} = \begin{pmatrix} AB \\ C_C^T A B \end{pmatrix} \begin{pmatrix} ABC_C \\ C_C^T A B C_C \end{pmatrix} = (AB)_F \]
The overhead:
- 2p-1 extra processes for p2
- one extra process need to receive the data for the rows and columns

Conclusion: a very scalable approach, more processors means less overhead
Failure Overhead

- FT-MPI will take care of fault management.
- Once the new process using the MPI_COMM_WORLD communicator restarts, we have to rebuild the communicators.
- Then we have to retrieve the data from the checkpoint processor.
Processor type: Opteron 2.2 GHz
Processor theoretical peak: 4.4 GFlops/sec
Number of application processors: 712
System theoretical peak (computational nodes): 3.13 TFlops/sec
Number of shared-memory application nodes: 356
Processors per node: 2
Physical memory per node: 6 GBytes
Usable memory per node: 3-5 GBytes
Switch Interconnect: InfiniBand
Switch MPI Unidirectional Latency: 4.5 µsec
Switch MPI Unidirectional Bandwidth (peak): 620 MB/s
Global shared disk GPFS Usable disk space: 30 TBytes
Batch system: PBS Pro
PBLAS vs. ABFT BLAS (0 failure)
Weak scalability
Strong Scalability
Conclusion

- Data-flow programming models an interesting alternative

- Fault tolerance is a requirement

- FT-MPI approach a viable possibility with algorithms already available

- The future of MPI is decided now!
• MVAPICH over Infiniband

• FT-MPI over socket on Infiniband
• The algorithm maintain the consistency of the checkpoints of the matrix C naturally

\[
\frac{2n^1}{p} y + 2(n + 2\sqrt{p} - 3)(\frac{n}{\sqrt{p}}\beta) \quad \text{PDGEMM-SUMMA}
\]

\[
\frac{2n(n + nloc)^2}{p} y + 2(n + 2\sqrt{p} - 3)(\frac{n + nloc}{\sqrt{p}}\beta) \quad \text{ABFT-PDGEMM-SUMMA}
\]