Adaptive Runtime Support for Fault Tolerance

Laxmikant (Sanjay) Kale
Celso Mendes
Esteban Meneses
Presentation Outline

• Object-based decomposition
  • General benefits with Charm++ and AMPI
  • Useful features for Fault Tolerance

• Fault Tolerance in Charm++ and AMPI
  • Checkpoint/Restart
  • Message Logging

• Future directions
Object-based over-decomposition

Objects:
- Locality of data references (performance)
- A parallel object can access only its own data
- Asynchronous method invocation

Over-decomposition:
- Decompose computation into objects
- Work units, data-units, composites
- Let an intelligent RTS assign objects to processors
Charm++

- Multiple “indexed collections” of C++ objects
- Multidimensional
- Dense or sparse
- Object-based Virtualization leads to Message Driven Execution
- Permits to overlap communication with computation
- Programmer expresses communication between objects with no reference to processors
Charm++ (cont.)

User View

System Implementation

CPU A

CPU B

CPU C

Scheduler

Message Queue

CPU

LACSS 2009, Santa Fe
• Each MPI process is implemented as a user-level thread (virtual processor)

• Threads are light-weight, and migratable! (<1 microsecond context switch time, potentially >100k threads per core)

• Each thread is embedded in a Charm++ object (chare)
Fault Tolerance

- Common Features:
  - Based on dynamic runtime capabilities
  - Use of object-migration
  - Can be used in concert with load-balancing schemes
  - Independence on the number of processors

- Four Approaches Available:
  - Disk-based checkpoint/restart
  - In-memory double checkpoint/restart
  - Proactive object migration
  - Message-logging
Disk-Based Checkpoint/Restart

- Similar to traditional checkpoint/restart; “migration” to disk
- Implemented by a blocking coordinated checkpoint: MPI_Checkpoint(DIRNAME)
  + Simple scheme, effective for common cases
  + Virtualization enables restart with any number of processors
- Checkpointing and data reload operations may be slow
- Work between last checkpoint and failure is lost
- Job needs to be resubmitted and restarted
Double In-Memory Checkpoint/Restart

- Avoid overhead of disk access for keeping saved data (allow user to define what makes up the state data)

- Implementation in Charm++/AMPI:
  - Coordinated checkpoint (SYNCFT)
  - Each object maintains two checkpoints:
    - on local processor’s memory
    - on remote buddy processor’s memory
Double In-Memory Checkpoint/Restart (cont.)

- A *dummy* process is created to replace crashed process
- New process starts recovery on other processors
  - use buddy’s checkpoint to recreate state of failing processor
  - perform load balance after restart
Recovery Performance

- Molecular Dynamics LeanMD code, 92K atoms, P=128
  - Load Balancing (LB) effect after failure:

![Graphs showing simulation time per step with and without LB](image-url)
Summary (SYNCFT)

+ Faster checkpointing than disk-based
+ Reading of saved data also faster
+ Only one processor fetches checkpoint across network
- Memory overhead may be high
- All processors are rolled back, despite individual failure
- All the work since last checkpoint is redone by every processor
Message-Logging

- Basic Idea: messages are stored by sender during execution
-Periodic checkpoints still maintained
- After a crash, reprocess “recent” messages to regain state

- Implementation in Charm++/AMPI:
  - New receptions occur in the same order
  - No need to roll back all the processors!
  - Restart can be parallelized
  - Virtualization helps fault-free case as well
Power consumption is continuous.

Normal Checkpoint-Resart method

Progress is slowed down with failures

LACSS 2009, Santa Fe
Our Checkpoint-Restart method
(Message logging + Object-based virtualization)

Faster recovery
Power consumption is lower during recovery
Fault-free Performance

- Test: NAS benchmarks, MG/LU
  - Versions: AMPI, AMPI+FT, AMPI+FT+multipleVPs
Bad scenario

Jacobi 3D (Abe, p=64, n=512, b=64)

Progress (iterations)

Time (seconds)

- NOFT
- SYNCFT
- MLOGFT

Wednesday, October 14, 2009
Memory Consumption

Jacobi 3D (Abe,p=64,n=512,b=64)

MLOGFT
SYNCFT

Progress (iteration)

Memory (MB)

LACSS 2009, Santa Fe
Summary (MLOGFT)

+ No need to roll back non-failing processors
+ Restart can be accelerated by spreading work to be redone
+ No need of stable storage
- Protocol overhead is present even in fault-free scenario
- Increase in latency may be an issue for fine-grained applications
Team-based Message Logging

- Group processors in teams and avoid logging intra-team messages
- Each team recovers as a unit
- Compromise between memory demand and recovery time
- Load balancer in charge of assigning objects to processors
- Cores per node = natural team size

LACSS 2009, Santa Fe
Allocated Memory

Jacobi 3D (Abe, p=64, n=512, b=64)

SIZE = 1
SIZE = 8

Memory (MB)

Progress (iteration)
Proactive Object Migration

- Basic Idea: use knowledge about impending faults
  - Migrate objects away from processors that may fail soon
  - Fall back to checkpoint/restart when faults not predicted

- Implementation in Charm++/AMPI:
  - Each object has a unique index
  - Each object is mapped to a *home* processor
    - objects need not reside on home processor
    - home processor knows how to reach the object
Proactive Object Migration (cont.)

Upon getting a warning, evacuate the processor

• reassign mapping of objects to new home processors

• send objects away, to their home processors
MPI Application Performance

• Sweep3d code, 150x150x150 dataset, P=32, 1 warning

• 5-point stencil code in Charm++, IA-32 cluster
Summary (Proactive)

+ No overhead in fault-free scenario
+ Evacuation time scales well, only depends on data and network
+ No need to roll back when predicted fault happens
- Effectiveness depends on fault predictability mechanism
- Some faults may happen without advance warning
Obstacles to FT on Existing Machines

- Current systems too strict and inflexible
- Entire application is killed when one process dies
- Most MPI implementations behave like this
- True in other scenarios as well (e.g. IBM’s POE+LAPI)
- Typical situation today
  - System software (OS, scheduler) controls the whole machine
  - Job is aborted when something goes bad
  - No option for application to continue running after faults, even for applications that could proceed!
  - But Charm++ net version can handle faults today, and other Charm++ versions can follow a similar scheme
Obstacles to FT on Existing Machines

- Desired scenario
  - System software optionally allows job to proceed beyond faults
  - It must be a community effort: includes vendor participation!

- Broader Need:
  - Scheduler that allows flexible, bi-directional communication between jobs and scheduler
  - Scheduler may notify job to shrink or expand, and job adapts accordingly
  - Job may ask scheduler for more resources when needed, or return partial resources no longer needed
Current PPL Research Directions

• Multiple concurrent failures

• Message-Logging Scheme
  – Decrease latency overhead and memory overhead
  – Stronger coupling to load-balancing
  – Newer schemes to reduce message-logging overhead
    – Team-based: a set of cores are sent back to their checkpoint (Greg Bronevetsky)
    – Implementation of other protocols (Franck Capello)
But, we are not experts in FT

- The message-driven objects model provides many benefits for fault tolerance schemes
  - Not just our schemes, but your schemes too
  - Multiple objects per processor: latencies of protocols can be hidden
  - Parallel recovery by leveraging “multiple objects per processor”
  - Can combine benefits by using system level or BLCR schemes specialized to take advantage of objects (or user-level threads)
Conclusions

• We have interesting fault tolerance schemes (read about them)

• We have an approach to parallel programming
  – That has benefits in the era of complex machines, and sophisticated applications
  – That is used by real apps
  – That provides beneficial features for FT schemes
  – That is available via the web
  – SO: please think about developing new FT schemes of your own for this model

• More info, papers, software: http://charm.cs.uiuc.edu
Acknowledgements

- Dep. of Energy – FastOS Program
- Colony-1 and Colony-2 projects
- Collaborators: ORNL (Terry Jones) & IBM (Jose Moreira)
- Fulbright Scholarship
- Interim support between Colony phases
- NSF/NCSA
  - Deployment efforts specific for Blue Waters
- Machine allocations
  - TeraGrid MRAC – NCSA, TACC, ORNL
  - Argonne Nat. Lab – BG/P
Thank you!
Overflow slides
Well Established Systems

- The Charm++/AMPI model has succeeded in CSE/HPC (because resource management,...)
- 15% of cycles at NCSA, 20% at PSC, were used on Charm++ apps, in a one year period
- So, work on fault tolerance for Charm++ and AMPI is directly useful to real apps
- Also, with AMPI, it applies to MPI applications
Application Performance

- Molecular Dynamics LeanMD code, 92K atoms, P=128
  - Checkpointing every 10 timesteps; 10 crashes inserted:

  ![Simulation time per step](image)
Memory vs Disk

Problem size (MB)
- double in-memory (Myrinet)
- double in-memory (100Mb)
- Local Disk
- double in-disk (Myrinet)
- NFS disk

Checkpoint overhead (s)

LACSS 2009, Santa Fe
Checkpoint Overhead

- 3D-Jacobi code in AMPI, 200 MB data, IA-32 cluster
  - Execution of 100 iterations, 8 checkpoints taken

![Graph showing execution time vs. number of processors for different setups: Normal Charm++/AMPI, FT-Charm++ w/o checkpointing, and FT-Charm++ with checkpointing.](image-url)
Evacuation Time vs Number of Processors

- 5-point stencil code in Charm++, IA-32 cluster
Fast restart performance

- Test: 7-point 3D-stencil in MPI, $P=32$, $2 \leq VP \leq 16$
- Checkpoint taken every 30s, failure inserted at $t=27s$
Protocol Optimization

- Combine protocol messages: reduces overhead and contention
- Test: synthetic compute/communicate benchmark

![Graph showing time per iteration vs. work per iteration per processor]
Evacuation Time vs Data Size

- 5-point stencil code in Charm++, IA-32 cluster