Scalable Fault Tolerance Schemes using Adaptive Runtime Support

Laxmikant (Sanjay) Kale
http://charm.cs.uiuc.edu
Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana Champaign
Presentation Outline

• What is object based decomposition
  – Its embodiment in Charm++ and AMPI
  – Its general benefits
  – Its features that are useful for fault tolerance schemes

• Our Fault Tolerance work in Charm++ and AMPI
  – Disk-based checkpoint/restart
  – In-memory double checkpoint/restart
  – Proactive object-migration
  – Message-logging
Object based over-decomposition

- Programmers decompose computation into objects
  - Work units, data-units, composites
  - Decomposition independent of number of processors
  - Typically, many more objects than processors

- Intelligent runtime system assigns objects to processors

- RTS can change this assignment (mapping) during execution
Object-based over-decomposition: Charm++

- Multiple “indexed collections” of C++ objects
- Indices can be multi-dimensional and/or sparse
- Programmer expresses communication between objects – with no reference to processors

System implementation

User View
Object-based over-decomposition: AMPI

- Each MPI process is implemented as a **user-level thread**
- 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)
Some Properties of this approach Relevant to Fault Tolerance

- Object-based Virtualization leads to *Message Driven Execution*
- Dynamic load balancing by migrating objects
- No dependence on processor number:
  - E.g. 3D cube of objects, can be mapped to a non-cube number of processors
Charm++/AMPI are well established systems

• The Charm++ 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
Fault Tolerance in Charm++ & AMPI

• Four Approaches Available:
  a) Disk-based checkpoint/restart
  b) In-memory double checkpoint/restart
  c) Proactive object migration
  d) Message-logging: scalable rollback, parallel restart

• Common Features:
  – Based on dynamic runtime capabilities
  – Use of object-migration
  – Can be used in concert with load-balancing schemes
Disk-Based Checkpoint/Restart

• Basic Idea:
  – Similar to traditional checkpoint/restart; “migration” to disk

• Implementation in Charm++/AMPI:
  – Blocking coordinated checkpoint: MPI_Checkpoint(DIRNAME)

• Pros:
  – Simple scheme, effective for common cases
  – Virtualization enables restart with any number of processors

• Cons:
  – Checkpointing and data reload operations may be slow
  – Work between last checkpoint and failure is lost
  – Job needs to be resubmitted and restarted
SyncFT: In-Memory double Checkpoint/Restart

• Basic Idea:
  – 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
  – Each object maintains two checkpoints:
    • on local processor’s memory
    • on remote buddy processor’s memory
  – A dummy process is created to replace crashed process
  – New process starts recovery on another processor
    • use buddy’s checkpoint to recreate state of failing processor
    • perform load balance after restart
Jacobi 3D (Abe,p=64,n=512,b=64)
Jacobi 3D (Abe,p=64,n=512,b=64)

Progress (iterations)

Time (seconds)

NOFT

SYNCFT
In-Memory Double Checkpoint/Restart (cont.)

- Comparison to disk-based checkpointing:

![Graph showing comparison between in-memory and disk-based checkpointing methods for different problem sizes.](image)

8/13/2009 HPC Resilience Workshop DC
In-Memory Double Checkpoint/Restart (cont.)

- **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)

8/13/2009  HPC Resilience Workshop DC
In-Memory Double Checkpoint/Restart (cont.)

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

![Graph showing simulation time per step](image)
In-Memory Double Checkpoint/Restart (cont.)

• Pros:
  – Faster checkpointing than disk-based
  – Reading of saved data also faster
  – Only one processor fetches checkpoint across network

• Cons:
  – Memory overhead may be high
  – All processors are rolled back, despite individual failure
  – All the work since last checkpoint is redone by every processor

• Publications:
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
  – Upon getting a warning, evacuate the processor
    • reassign mapping of objects to new home processors
    • send objects away, to their home processors
Proactive Object Migration (cont.)

- Evacuation time as a function of \#processors:
  - 5-point stencil code in Charm++, IA-32 cluster
Proactive Object Migration (cont.)

- Performance of an MPI application
  - Sweep3d code, 150x150x150 dataset, P=32, 1 warning

![Graph showing performance over iterations with and without load balance, warning, and load balancing points.](image-url)
Proactive Object Migration (cont.)

• **Pros:**
  – 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

• **Cons:**
  – Effectiveness depends on fault predictability mechanism
  – Some faults may happen without advance warning

• **Publications:**
  – Chakravorty, Mendes & Kale: HiPC, Dec.2006
  – Chakravorty, Mendes, Kale et al: ACM-SIGOPS, April 2006
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:
  – Since the state depends on the order of messages received, the protocol ensures that the new receptions occur in the same order
  – Upon failure, roll back is “localized” around failing point: no need to roll back all the processors!
  – With virtualization, work in one processor is divided across multiple virtual processors; thus, restart can be parallelized
  – Virtualization helps fault-free case as well
Message-Logging (cont.)

- Fast restart performance:
  - Test: 7-point 3D-stencil in MPI, P=32, $2 \leq VP \leq 16$
  - Checkpoint taken every 30s, failure inserted at $t=27s$
Normal Checkpoint-Resart method
Progress is slowed down with failures
Power consumption is continuous

8/13/2009
HPC Resilience Workshop DC
Our Checkpoint-Resart method

(Message logging + Object-based virtualization)

Progress is faster with failures

Power consumption is lower during recovery
Message-Logging (cont.)

- Fault-free performance:
  - Is ok with large-grain, but significant los with fine-grained
  - Test: NAS benchmarks, MG/LU
  - Versions: AMPI, AMPI+FT, AMPI+FT+multipleVPs
Message-Logging (cont.)

- Protocol Optimization:
  - Combine protocol messages: reduces overhead and contention
  - Test: synthetic compute/communicate benchmark
Jacobi 3D (Abe, p=64, n=512, b=64)
Jacobi 3D (Abe, p=64, n=512, b=64)
Jacobi 3D (Abe, p=64, n=512, b=64)
Message-Logging (cont.)

• **Pros:**
  – No need to roll back non-failing processors
  – Restart can be accelerated by spreading work to be redone
  – No need of stable storage

• **Cons:**
  – Protocol overhead is present even in fault-free scenario
  – Increase in latency may be an issue for fine-grained applications

• **Publications:**
  – Chakravorty & Kale: IPDPS, April 2007
  – Chakravorty & Kale: FTPDS workshop at IPDPS, April 2004
Current PPL Research Directions

• Message-Logging Scheme
  – Decrease latency overhead in protocol
  – Decrease memory overhead for checkpoints
  – Stronger coupling to load-balancing
  – Newer schemes to reduce message-logging overhead
    • Clustering: a set of cores are sent back to their checkpt
      – Greg Bronevetsky’s suggestion
    • Other collaboration with Franck Capello
Some external Gaps

• Scheduler that won’t kill a job
  – Broader need: a scheduler that allows flexible bi-directional communication between jobs and scheduler

• Fault prediction
  – Needed if proactive Fault Tolerance is of use

• Local disks!

• Need to better integrate knowledge from distributed systems
  – They have sophisticated techniques, but HPC metrics and context is substantially different
Messages

• 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
• And please pass the word on: we are hiring