Object-based Over-Decomposition Can Enable Powerful Fault Tolerance Schemes

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

• Appeal for research in leveraging these features in FT research
Parallel Programming Lab - PPL

- [http://charm.cs.uiuc.edu](http://charm.cs.uiuc.edu)
  - Open positions 😊
PPL Mission and Approach

• To enhance *Performance and Productivity* in programming *complex* parallel applications
  – Performance: scalable to thousands of processors
  – Productivity: of human programmers
  – Complex: irregular structure, dynamic variations

• Application-oriented yet CS-centered research
  – Develop enabling technology, for a wide collection of apps.
  – Embody it into easy to use abstractions
  – Implementation: Charm++
    • Object-oriented runtime infrastructure
    • Freely available for non-commercial use

6/6/2009 Dagstuhl Fault Tolerance Workshop
Enabling CS technology of parallel objects and intelligent runtime systems has led to several CSE collaborative applications.

Well-known Biophysics molecular simulations App
Gordon Bell Award, 2002

OpenAtom
Synergy

ChaNGa

Space-Time Meshing

System

NAMD

Issues

Other Applications

Nano-Materials..

Computational Astronomy

Space-Time Simulation

Rocket Simulation
Object based over-decomposition

• **Objects:**
  – Locality of data references is a critical attribute for performance
  – A parallel object can access only its own data
  – Asynchronous method invocation for accessing other’s data

• **Over-Decomposition**
  – the programmer decompose computation into objects
    • Work units, data-units, composites
  – Let an intelligent runtime system assign 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)
Benefits of Object-based overdecomposition

• **Software engineering**
  – Number of virtual processors can be independently controlled
  – Separate VPs for different modules

• **Message driven execution**
  – Adaptive overlap of communication
  – Predictability:
    • Automatic out-of-core
    • Prefetch to *local stores*
  – Asynchronous reductions

• **Dynamic mapping**
  – Heterogeneous clusters
    • Vacate, adjust to speed, share
  – **Automatic checkpointing, more advanced Fault Tolerance schemes**
  – Change set of processors used
  – Automatic dynamic load balancing
  – Communication optimization
Some Relevant Properties of this approach:

Message Driven Execution

Object-based Virtualization leads to Message Driven Execution

Scheduler

Message Q

Scheduler

Message Q
Some Relevant Properties of this approach:

Parallel Composition:
A1; (B || C ); A2

Recall: Different modules, written in different languages/paradigms, can overlap in time and on processors, without programmer having to worry about this explicitly.
Without message-driven execution (and virtualization), you get either:
Space-division
OR: Sequentialization

Diagram of network with nodes A, B, and C connected.
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

6/6/2009 Dagstuhl Fault Tolerance Workshop
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

• 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
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 other processors
    • use buddy’s checkpoint to recreate state of failing processor
    • perform load balance after restart
In-Memory Double Checkpoint/Restart (cont.)

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

![Graphs showing total execution time for different numbers of processors on 100Mbit and Myrinet networks.](image-url)
In-Memory Double Checkpoint/Restart (cont.)

- Comparison to disk-based checkpointing:

![Graph showing comparison of checkpoint overhead with problem size for different storage types.](image)
In-Memory Double Checkpoint/Restart (cont.)

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

![Graph](image-url)

6/6/2009 Dagstuhl Fault Tolerance Workshop
In-Memory Double Checkpoint/Restart (cont.)

- Application Performance:
  - Molecular Dynamics LeanMD code, 92K atoms, P=128
  - Checkpointing every 10 timesteps; 10 crashes inserted:
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 data size:
  - 5-point stencil code in Charm++, IA-32 cluster

![Graph showing evacuation time vs. data size]
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
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 \(t=27s\)
Power

Progress

Time

Normal Checkpoint-Resart method

Progress is slowed down with failures

Power consumption is continuous
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:
  – 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
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
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)
    – I am sure you can think of many new schemes

• We are willing to help
  – (without needing to be co-authors)
  – E.g. a simplified version of Charm RTS for you to use?
Messages

• We have an 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
PPL Funding Sources

• National Science Foundation
  – BigSim, Cosmology, Languages

• Dep. of Energy
  – Charm++ (Load-Balance, Fault-Tolerance), Quantum Chemistry

• National Institutes of Health
  – NAMD

• NCSA/NSF, NCSA/IACAT
  – Blue Waters project (Charm++, BigSim, NAMD), Applications

• Dep. of Energy / UIUC Rocket Center
  – AMPI, Applications

• NASA
  – Cosmology/Visualization