Programming with Parallel Objects

Session 3: Advanced Programming with Charm++

Esteban Meneses

Advanced Computing Laboratory
Costa Rica National High Technology Center

School of Computing
Costa Rica Institute of Technology

2017

CeNAT
Centro Nacional de Alta Tecnología

ACKNOWLEDGEMENTS

Most of the content of this tutorial was provided courtesy of Prof. Laxmikant V. Kalé from the Parallel Programming Laboratory (PPL) of the University of Illinois at Urbana-Champaign. The PPL originally developed the ideas behind parallel object programming and has maintained the Charm++ code base for more than 20 years.

ADMINISTRATIVIA

- Official Charm++ website:
  http://www.charmplusplus.org/
- Slides and code for this tutorial:
  https://github.com/emenesesrojas/parallel_objects.git
- Questions on parallel objects programming:
  https://ecar2017.slack.com
- Advanced questions on Charm++:
  ppl@cs.illinois.edu
- Official instructor's email address:
  emeneses@cenat.ac.cr

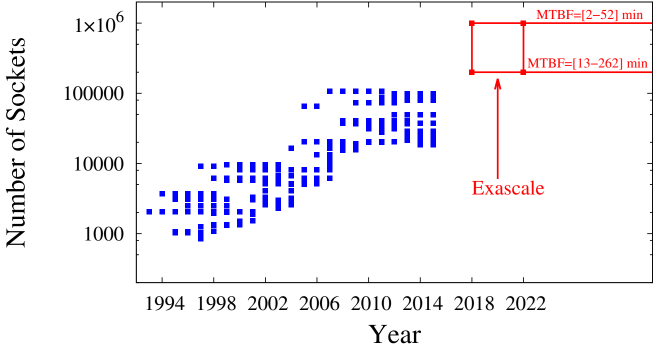
Outline

Checkpointing and Resilience

Performance Analysis

The Reliability Problem

Future supercomputers will fail frequently

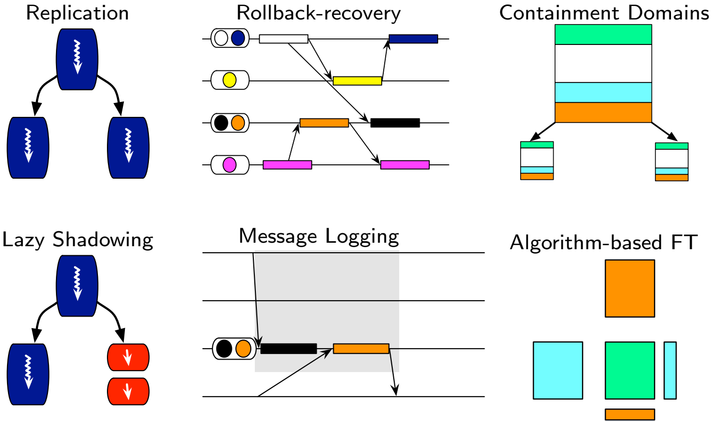


Source: www.top500.org

- *Today, more than 20% of the computing capacity in a large HPC system is wasted due to failures and recoveries* [Elnozahy, 2010]
- *There were 317 independent hardware failures on Titan in 2014* [Meneses, 2015]
- *Insufficient resilience of the software infrastructure would likely render extreme scale systems effectively unusable* [Dongarra, 2011]

The Current Solution

Several strategies make supercomputing reliable



More transparent...Less transparent

Fault Tolerance in Charm++

Approaches based on migratability

- Four Approaches:
  - Disk-based checkpoint/restart
  - In-memory double checkpoint/restart
  - (experimental) Proactive object evacuation
  - (experimental) Message-logging for scalable fault tolerance
- Common Features:
  - Easy checkpoint
  - Migrate-to-disk leverages object-migration capabilities
  - Based on dynamic runtime capabilities
  - Can be used in concert with load-balancing schemes

Split Execution

Checkpointing to the file system

- The common form of checkpointing
  - The job runs for 5 hours, then will continue at the next allocation another day!
- The existing Charm++ infrastructure for chare migration helps
- Just "migrate" chares to disk
- The call to checkpoint the application is made in the main chare at a synchronization point
- Example:

```
CkCallback cb(CkIndex_Hello::SayHi(),helloProxy);
CkStartCheckpoint(''log'',cb);
```

Split Execution

Additional changes to code

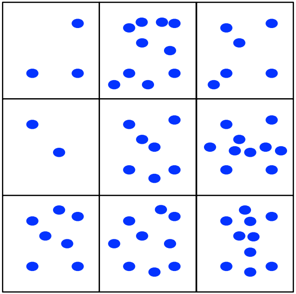
- Main chare must be made migratable
- Groups must be made migratable
- Migratable objects have both, a `pup` method and a migration constructor
- Resuming execution:

```
> ./charmrun hello +p4 +restart log
```

Exercise 3.1

Particle interaction

Add checkpointing capabilities to the particle interaction code on a 2D space:



Complete the code in directory:

`code/particle/skeleton`

of the class repository:

`git clone https://github.com/emenesesrojas/parallel_objects.git`

Do not forget to define variable `CHARMDIR` in the Makefile

Performance Analysis

Using Projections to find bottlenecks

- Instrumentation and measurement
  - Link program with `-tracemode projections` or `-tracemode summary`
  - Trace data is generated automatically during run
  - User events can be easily inserted as needed
- Projections: visualization and analysis
  - Scalable tool to analyze up to 300,000 log files
  - A rich set of tool features: time profile, time lines, usage profile, histogram, and more
  - Detect performance problems: load imbalance, grain size, communication bottleneck, and more

Using Projections

Different types of tools

- Tools of aggregated performance viewing
  - Time profile
  - Histogram
  - Communication over time
- Tools of processor level granularity
  - Overview
  - Timeline
- Tools of derived/processed data
  - Extrema analysis: identifies outliers
  - Noise miner: highlights probable interference

Problem Identification

Procedure

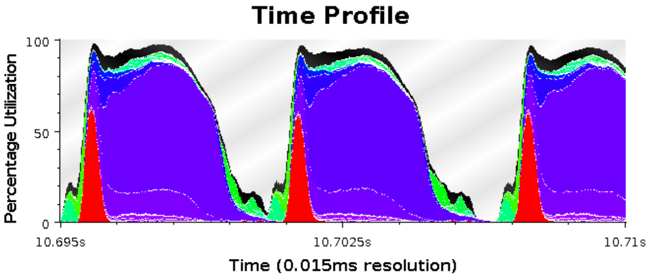
- Load imbalance
  - Time profile: lower CPU usage
  - Extrema analysis tool:
    - Least idle processors
  - Load the over-loaded processors in Timeline
  - Histogram: grain size issues

Example
NAMD

- Molecular dynamics:
  - Trying to identify the next performance obstacle for NAMD
    - Running on 8192 processors, with 1 million atom simulation
    - Jaguar Cray XK6
    - Test scenario: with PME every step

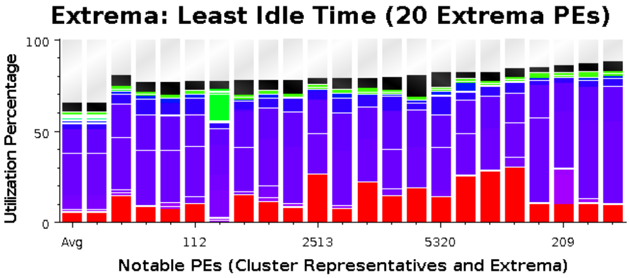
Example

Time profile



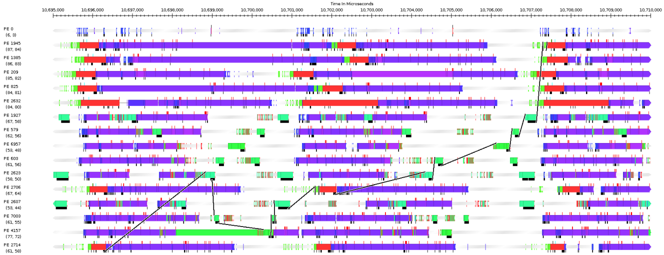
Example

Extrema tool for least idle processors



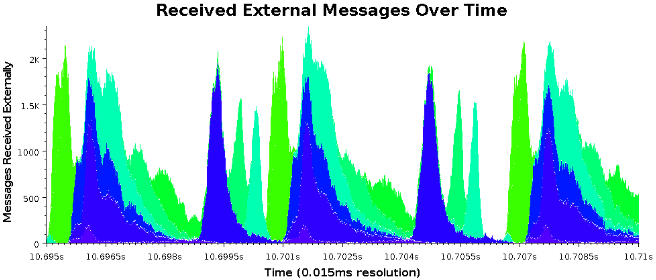
Example

Time lines with message back tracing



Example

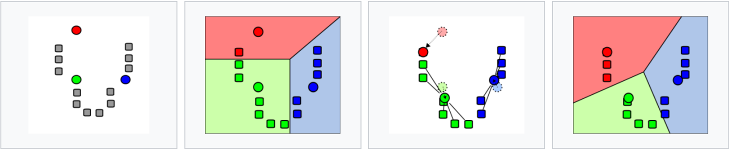
Communication over time for all processors



Project

Parallel K-means

Write a Charm++ parallel program to solve k-means on an *n*-dimensional space:



Complete the code in directory:

`code/kmeans/skeleton`

of the class repository:

`git clone https://github.com/emenesesrojas/parallel_objects.git`

Do not forget to define variable `CHARMDIR` in the Makefile

Project
Specifications

- Data points must be randomly created inside a unit multidimensional cube
- There must be a 1D array of parallel objects
- Each object holds a portion of the data points and computes the clustering
- The last object of the array has at least 10 times more data points than the rest
- Migration of objects must be implemented
- The main chare collects the new computed centers every iteration
- Turn in this project individually
- Send the solution to `emeneses@cenat.ac.cr`

Concluding Remarks

- Reliability is a fundamental concern in making usable supercomputers
- Migratability empowers fault tolerance in Charm++
- Performance analysis is a useful tool to detect performance bottlenecks

Thank You!
Q&A

Centro Nacional de Alta Tecnología