Parallel Programming with Charm++

Phil Miller

October 31, 2014

There are several challenges in programming applications for large supercomputers: exposing concurrency, data movement, load imbalance, heterogeneity, variations in application's behavior, system failures etc. Addressing these challenges requires more emphasis on the following important concepts during application development: overdecomposition, asynchrony, migratability, and adaptivity. At the same time, the runtime systems (RTS) will need to become introspective and provide automated support for several tasks, e.g. load balancing, that currently burden the programmer.

This tutorial is aimed at exposing the attendees to the above mentioned concepts. We will present details on how a concrete implementation of these concepts, in synergy with an introspective RTS, can lead to development of applications that scale irrespective of the rough landscape. We will focus on Charm++ as the programming paradigm that encapsulates these ideas, and use examples from real world applications to further the understanding.

Charm++ provides an asynchronous, message-driven programming model via migratable objects and an adaptive RTS that guides execution. It automatically overlaps communication, balances loads, tolerates failures, checkpoints for split-execution, interoperates with MPI, and promotes modularity while allowing programming in C++. Several widely used Charm++ applications thrive in computational science domains including biomolecular modeling, cosmology, quantum chemistry, epidemiology, and stochastic optimization. After the tutorial, attendees will be:

- Familiar with the keys ideas of overdecomposition, asynchrony, migratability, and adaptivity.
- Knowledgeable about the state of the art techniques and their utility in adaptive RTS.
- Able to express their problems as interacting overdecomposed objects, e.g. parallel objects in Charm++.
- Able to understand and address the performance characteristics of their Charm++ applications, using both simple manual instrumentation and the Projections performance visualization and analysis tool.
- Capable of identifying the features that Charm++ provides to improve parallel programming productivity, and determine which are applicable to their applications and avoid scalability bottlenecks
- Able to integrate MPI and Charm++ code in mixed-paradigm applications

Targeted Audience, Content Level, and Prerequisites

This tutorial will benefit three broad classes of attendees:
1. Experienced application developers who would like to reassess the challenges of scaling their codes to future and current large-scale machines, the approaches taken, and the software framework they use;
2. Numerical and computational scientists who would like to develop new applications with complex structures and dynamic behaviors, as well as those who would like to examine the performance characteristics of their existing applications in light of the tools and techniques presented in the tutorial;
3. Educators who are developing instruction curricula for parallel programming, and would like to cover a range of practical systems used in the field.

We expect that the audience will be comfortable with programming in C++ and have some background knowledge of parallel computing. The presented material will have a broad appeal – we anticipate that 50%

of the material will be beginner-level, 30% intermediate-level, and 20% intended for parallel programming experts. The intermediate level content will follow directly from the basic content that we will present. The advanced content will be structured so that less experienced attendees will understand its motivations and recognize its relevance should a need arise in practice, so that they can revisit the appropriate tutorial material after they have gained more experience.

Attendees should ideally bring a laptop with a C++ compiler installed on which to follow along, but this is not required.

Outline of Contents

1. Introduction
   - Object Design: decomposition in terms of problem and computational techniques, not execution resources
   - Execution Model: asynchronous, message-driven execution
2. Basics of object level programming
   - Hello World - simple examples and basic syntax
   - Collection of Objects
3. Benefits of Charm++: automatic overlap of communication & computation; modularity; automatic & adaptive resource management
4. Mechanics
   - Basic Charm++ Syntax: object and interface definition, construction, messaging, collectives
   - Compilation and execution: compiler wrapper and process launching
   - Control Flow with Structured Dagger (SDAG)
5. Task Parallelism: granularity, recursive and unstructured problem decomposition
6. Application Design Examples: case studies from applications such as NAMD, ChaNGa, OpenAtom, RocStar, Fractography, EpiSimdemics
7. Performance Tuning and Visualization Tools: timing, tracing, Projections tool
8. Dynamic Load Balancing
   - Strategy Selection: Comprehensive vs refinement, communication consciousness, application-specificity.
   - Automatic and adaptive frequency determination
9. Checkpointing and Fault Tolerance
   - Filesystem checkpointing: Automatic split execution of long runs across short jobs, restarting after system failure, restarting on different numbers of cores.
   - Online automatic failure recovery mechanisms: double in-memory/disk checkpointing.
10. Interoperability with MPI: division of the parallel system in time and/or space
11. Debugging: gdb, valgrind, specialized memory management, CharmDebug
12. Optimization Techniques
    - Messages: avoiding the costs of copying or marshalling data
    - Groups and NodeGroups: sharing data between objects on a processor or shared-memory node
    - Entry Method options and attributes
    - Array Sections
    - Shared Memory