Productive Parallel Programming with CHARM++

Phil Miller
Department of Computer Science
University of Illinois
mille121@illinois.edu

Keywords: Parallel programming, productivity, load balancing, overdecomposition

1. INTRODUCTION

CHARM++ is a general-purpose framework for developing high-performance parallel applications [1]. Applications written using CHARM++ run at scales spanning mobile devices [2], multi-core processors, multi-processor NUMA woprkstations and servers, networked clusters, and the world's largest supercomputers. A selection of CHARM++ applications in production scientific usage is shown in Table 1. These applications consume the second most execution cycles on NSF computing resources, after MPI.

The key goal of CHARM++ is high scientific productivity. This takes account of the notion that the time to solve a scientific problem with computation includes both development time and execution time. At present, a substantial portion of development time on large-scale parallel applications is directed toward performance optimization and implementing features that are necessary in many such applications, such as load balancing. Because of limitations on developer time and differences in expertise, application-specific renditions of these features tend to be fairly rudimentary.

CHARM++ addresses this challenge through robust *separation of concerns*. Application developers and scientific users are expected to apply their domain knowledge to define scalable parallel algorithms that suit their purpose. The CHARM++ runtime system takes responsibility for ensuring those algorithms execute efficiently, across the full range of platforms. Where necessary, CHARM++ provides plugin interfaces for highly application-dependent tasks (such as mapping and load balancing) while ensuring that application logic is unaffected by changes in these areas.

In light of its widespread use, CHARM++ is maintained as a high-quality software project suitable for production, rather than just CS research. Bugs in the system are tracked and aggressively fixed. Changes are verified through a bundled test suite and continuous integration with several applications. They are also reviewed by a developer other than their author for problems before integration.

HPC 2015, April 12 - 15, 2015, Alexandria, VA, USA

2. TUTORIAL

This short tutorial aims to get new CHARM++ users over the initial hurdles, to enable subsequent independent exploration and development. These hurdles include both conceptual issues, like the elements of a CHARM++ program and design philosophy, as well as mechanical issues like compilation and language syntax. With that foundation, we will discuss the design of a few example applications and usage of key features of the CHARM++ system.

2.1. Download, build, install CHARM++

CHARM++ is distributed in source form on the web at `http://charmplusplus.org/`. Once downloaded, users need merely run `./build` to activate an interactive script that will generate and compile a suitable configuration.

2.2. Design Philosophy of CHARM++ Applications: Overdecomposition

The execution model of CHARM++ is that every processor in the parallel system is assigned a set of objects whose work will be executed on that processor. These objects are activated by the delivery of asynchronous messages through their host processor's scheduling queue. Good CHARM++ applications typically present many objects per processor, an approach that we term *overdecomposition* in contrast to the process-oriented decomposition typical of MPI and PGAS models. These objects are generally representative of units in the problem domain (e.g. matrix blocks, chunks of an interaction pattern, or blocks of a data-parallel structure).

This structure yields several benefits. The first benefit is processor count independence. Since the objects are defined by the input problem and program configuration, and mapped to processors by the runtime system, programs can run on any number of processors. This is a major natural convenience over older-style MPI applications that were often constrained to run on processor counts that were a specific function of the input size, like a power of 2 or a cube.

The next benefit is adaptive, asynchronous execution. With multiple objects on each processor, only one must be ready to execute at a time to ensure that they don't run out of work and go idle. The runtime system can fully overlap communication to and from each object with computation on others. The relative priority of different units of work can also be adapted dynamically, either based on application hints or runtime instrumentation and control mechanisms.

Table 1. A selection of applications built on CHARM++. Scaling of "$> 350k$ cores" indicates effective use of the full Blue Waters Cray XE6 system at NCSA. SLOC are measured using David A Wheeler's 'SLOCcount'

Application	Domain	SLOC	Scalability (cores)
NAMD	Molecular biophysics	118k	$> 350k$
ChaNGa	Cosmology & Astrophysics	74k	$> 350k$
EpiSimdemics	Agent-base epidemiology	N/A	$> 350k$
OpenAtom	Material science	97k	$40k$
Enzo-P/Cello	Cosmology & Astrophysics	52k	$32k$
FreeON/SpAMM	Quantum Chemistry	111k	$24k$
ROSS	Discrete Event Simulation	N/A	$16k$
SDG	Engineering mechanics	N/A	$10k$
ClothSim	Cloth Physics with Rigid Bodies	N/A	768
JetAlloc	Stochastic mixed-integer program optimization	N/A	120

Given overdecomposition and the *migratability* of the objects, the runtime system is further empowered to provide efficient execution. Based on measured or predicted work required on each object, the mapping of objects to processor can be adjusted dyanmically to provide *load balance*. CHARM++ provides both the instrumentation and migration infrastructure to enable this, and a suite of plug-in strategies to generate new mappings. Various strategies consider a wide range of factors beyond simply load, such as communication locality, processor speed, and interference from other processes and co-located VMs on a common host.

With the serialization mechanisms necessary to migrate objects for load balancing purposes, CHARM++ is also able to take snapshots of those objects for *fault tolerance* purposes. At its simplest, this consists of an application-agnostic pure userspace for checkpointing to files on stable storage and restarting from those files later. The system can restart applications on a different number of nodes and processes than were used in the run that generated a checkpoint, granting substantial flexibility. In supportive environments, CHARM++ also provides more sophisticated mechanisms that implement online fault tolerance, that allow applications to continue running through the loss of one or even several nodes and the occurrence of data corruption. These include in-memory checkpoints, message logging, and cross-checked replicated execution.

2.3. MPI Interoperation

Code written in CHARM++ and MPI can interoperate within a single parallel job [3]. This provides application developers with substantial flexibility. It means that individual modules of an application can be written in one model or the other as appropriate. In particular, it allows developers writing primarily in one model to take advantage of pre-existing libraries written in the other. It also enables incremental migration between the two different environments, converting individual components as the need arises rather than porting an entire codebase in a single step.

REFERENCES

[1] B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni, L. Wesolowski, and L. Kale, "Parallel Programming with Migratable Objects: Charm++ in Practice," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. New York, NY, USA: ACM, 2014.

[2] Touchpress Limited. Molecules by Theodore Gray. [Online]. Available: https://itunes.apple.com/us/app/molecules-by-theodore-gray/id923383841?mt=8

[3] N. Jain, A. Bhatele, J.-S. Yeom, M. F. Adams, F. Miniati, C. Mei, and L. V. Kale, "Charm++ & MPI: Combining the best of both worlds," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (to appear)*, ser. IPDPS '15. IEEE Computer Society, May 2015, lLNL-CONF-663041.

Biography

Phil Miller is a PhD candidate in the Department of Computer Science at the University of Illinois at Urbana-Champaign, focusing on the development of asynchronous parallel algorithms. He is a member of the Parallel Programming Laboratory, directed by Professor Laxmikant (Sanjay) Kalé. Phil has been a core developer of CHARM++ for several years, and managed feature and patch releases spanning versions 6.3.0 through 6.6.0. He has given presentations and tutorials on CHARM++ at numerous conferences and national laboratories.

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois.