Simplifying Parallel Programming for CSE Applications Using a Multi-Paradigm Approach
Project NSF 0833188
Annual Report: Sep.2011 to Aug.2012

1 Project Participants

1.1 Funded Personnel

- Prof. Laxmikant V. Kalé (PI), Prof. David Padua (co-PI), Prof. Vikram Adve (co-PI)
- Anshu Arya (Graduate Student): 9/17/11 - 12/31/11
- Aaron Becker (Graduate Student): 8/16/11 - 8/15/12
- Ehsan Totoni (Graduate Student): 8/16/11 - 10/15/11
- JoAnne Geigner (Information Specialist): 25% 8/16/11 - 8/15/12
- Ramprasad Venkataraman (Research Programmer): 10% 8/16/11 - 3/15/12
- Pritish Jetley (Graduate Student): 11/16/11 - 2/15/12
- Jonathan Lifflander (Graduate Student): 1/1/12 - 2/15/12
- Graham Evans (Graduate Student): 2/16/12 - 5/15/12
- Adarsh Keshan (Undergraduate Student): 5/10/12 - 7/24/12

1.2 Unfunded Collaborators

- Phil Miller (Graduate Student): University of Illinois, Dept. Computer Science

2 Activities and Findings

2.1 Research Activities – 2011/2012

2.1.1 Charj

In the past year, we have extended and improved Charj, our high-level language and associated compiler infrastructure that targets the Charm++ runtime system. Charj aims to simplify the task of writing programs that use the Charm++ infrastructure and enhance programmer productivity by eliminating redundant code, facilitating the use of multiple programming models within a single application, and taking advantage of domain-specific knowledge to power model-specific compiler optimizations.

We have improved support for embedding multiple programming models within Charj programs, including support for partitioned global address space programming via the multiphase shared arrays programming model. We have also added support for heterogeneous programming in Charj, allowing programmers to write applications that run on both traditional multicore hardware and floating point accelerator architectures such as GPUs without requiring separate code for each architecture.

Using Charj, we developed multiple small applications that have similar computational characteristics as larger, production-scale parallel applications: a Lennard-Jones molecular dynamics application, an LU decomposition matrix factorization application, and a Jacobi relaxation application. These applications were smaller than their Charm++ equivalents by 27.2% to 56.7% in terms of token count and by 11.8% to 57.1% in terms of lines of code without sacrificing performance relative to the Charm++ implementations.

One of the factors that contributes to the decrease in line and token count is the concise syntax we have developed for sequential arrays in Charj. Our array implementation allows the programmer to slice an n-dimensional array, the runtime copying the necessary region instead of the programmer. This reduces potential programmer error and also enables layout optimizations that the runtime can perform depending on access patterns.

2.1.2 Charisma

Charisma is a higher-level notation that allows for the expression of static data flow programs in an elegant, concise and productive manner. It separates parallel algorithmic structure from the sequential details of implementation, thereby

fostering code readability and maintainability. Recently, we added three new syntactic constructs to Charisma, to obtain a language that has a much broader scope of expression. These constructs are:

1. *Publisher-directed communication patterns*. The communication pattern is specified in the form of data dependencies between published and consumed values. To allow for greater generality and efficiency, we allow the index expressions of published values to be any expression, whereas the index expression of consumed values may only be (projections of) the identity expression on object array indices.

2. *Index subspaces*. Ability to apply actions on predicated subsets of object arrays.

3. *Value range publication*. Support for publication of *ranges* of parameter values in addition to individual values and anonymous sets of values.

Given that they represent a significant syntactic departure from the previous version, we call the new language Charisma 2.0. The new language allows for the expression of algorithms in which communication pattern instances are tied to the dynamic, run-time variables such as for/while-loop indices. Examples are: Parallel prefix sum, Butterfly FFT, Gauss-Seidel relaxation, Dense LU decomposition, Multigrid solvers and an ab initio quantum/molecular dynamics simulator called OpenAtom.

2.1.3 Divide-and-Conquer

We have constructed an abstraction for the fine-grained expression of divide-and-conquer algorithms in a productive manner. Our framework addresses an important and significant gap in the literature: *generative recursion* requires both task- and data-parallel constructs, but for distributed memory machines, only the question of task-parallelism has been studied in the past. Through our work, we provide constructs that enable the efficient exploitation of data-parallelism on distributed memory machines. In particular, we have created the *DivConArray*, which is a global-data-view that supports operations that are important for divide-and-conquer algorithms, namely map/reduce, read/write, and data redistribution. The associated runtime system component automatically manages the agglomeration of operations and communication to ensure an efficient implementation, while giving the programmer the convenience of expressing his/her algorithm in

a fine-grained manner. The framework has a separate component that performs dynamic task agglomeration, thereby providing automatic grain size control.

2.1.4 Generic Programming

Charm++ has long supported generic programming through C++ class templates for defining parallel objects. However, the lack of orthogonal support for method templates and SDAG coordination code in templates limited expressiveness. We have now extended template support to individual methods and to SDAG. This support is used in the production application OpenAtom.

2.1.5 HPC Challenge Award

The project personnel, in collaboration with other members of the Parallel Programming Laboratory, have implemented the complete set of benchmarks from the HPC Challenge suite in Charm++ and SDAG. This builds on the previous year's effort of developing and tuning CharmLU, which implemented one member of the suite. In the course of this effort, tools and libraries in the Charm++ stack were improved to meet the needs demonstrated by these benchmarks.

CharmLU was further used to produce a detailed study of mapping techniques for dense linear algebra routines as relates to modern multicore supercomputer and cluster nodes.

2.2 Findings – 2011-2012

Our major findings in this period (September 2011 to August 2012) were:

- **Charj:** The use of a programming language specific to the message-driven programming model can result in significantly shorter applications with equivalent performance and functionality. It also provides an opportunity for improved warning and error messages based on semantic knowledge of the programming model, reducing the chances for model-specific errors. Furthermore, it exposes opportunities for compiler optimization that cannot be performed by compilers targeted at sequential languages because of their model-specific nature.

- **Charisma:** The addition of three simple constructs, namely (1) *publisher-directed* communication patterns, (2) index subspaces and (3) parameter value range publication significantly broaden the scope of expression of Charisma, allowing us to write programs that could not otherwise have been written in the language.

- **Divide-and-Conquer:** A framework that automates task grain size control and provides dynamic agglomeration of data-parallel operations for efficient communication can improve parallel performance significantly. The framework engenders productivity by allowing the natural, fine-grained of divide-and-conquer algorithms.

- **Generic Programming:** More thorough support for template metaprogramming in Charm++ enables clearer, more concise expression of application logic in the OpenAtom code. For example, algorithms that can operate in terms of various numeric types (single versus double precision, real versus complex) now only need to be expressed once, with the attendant benefits for readability and correctness.

- **HPC Challenge:** The suite of developed benchmarks achieved admirable performance on multiple platforms, with substantially less code than reference implementations. The necessary code also exhibited better separation of concerns between algorithmic logic necessary for correctness and tuning logic necessary for performance. The separation was measured not just through the volume and structure of the resulting code, but also through

version control logs recorded over the course of development. These benchmarks were submitted to the HPC Challenge Competition at Supercomputing 2011, where they were awarded a Class 2 (Productivity) prize for overall performance. Additionally, CharmLU was used to study the performance effects of various mapping schemes on performance.

2.3 Research Training and Development

2.4 Outreach Activities

The annual workshop on Charm++ and its applications had another edition in May 2012. During this workshop, some of the results from our Charm++ developments were presented. We expect to continue conducting the Charm++ workshop every year. All the materials related to the workshop are freely available from our website, including slides and archived audio/video from the presentations.

In addition, Phil Miller delivered an introductory tutorial on Charm++ programming to the UIUC chapter of SIAM. This tutorial provided exposure to the Charm++ stack for large scale parallel computing to participants at many levels (undergraduate, graduate, research staff) working in a variety of disciplines (CS, engineering, and sciences).

2.5 Publications and Products

- Jonathan Lifflander, Phil Miller, Ramprasad Venkataraman, Anshu Arya, Terry Jones, and Laxmikant Kale: "Mapping Dense LU Factorization on Multicore Supercomputer Nodes", International Parallel and Distributed Processing Symposium (IPDPS), May 2012, Shanghai China.

- Phil Miller, Aaron Becker, and Laxmikant Kale: "Using shared arrays in message-driven parallel programs", Parallel Computing Volume 38 Issue 1-2, January, 2012.

- Aaron Becker: "Compiler Support for Productive Message-Driven Parallel Programming", Ph.D. Thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 2012.

- Pritish Jetley and Laxmikant Kale: "Optimizations for Message Driven Applications on Multicore Architectures", 18th annual IEEE International Conference on High Performance Computing (HiPC), December, 2011.

2.6 Internet Dissemination

The following products have resulted from this project so far:

1. Enhancements to the Charm++ infrastructure, available at `http://charm.cs.illinois.edu`

2. Charm++ Workshop materials, available at
   `http://charm.cs.illinois.edu/charmWorkshop`

3 Contributions

3.1 Within Discipline

This project is developing a new approach to parallel programming that includes development of multiple, individually incomplete, programming models. Each model simplifies parallel programming while still covering significant categories of applications. This collection of interoperable models provides a powerful environment for developing future petascale applications.

3.2 Outside Discipline

Software for development of Scientific and Engineering applications can make a significant impact on society through better understanding of physical phenomenon and improved design of engineered artifacts. The results of this project, including the new parallel programming models being developed, will lead to effective use of the petascale computing facilities being developed and deployed nationally.

3.3 Contributions to Human Resources Development

This project is training the next generation of computer scientists in the art of using parallel computers and in developing programming techniques for large systems. It is giving our graduate students an excellent environment for the use of parallel machines and for the development of languages aimed at future machines. The software developed in this project is freely distributed via the Internet. By accessing this distribution, students from various application areas should be able to develop parallel applications, in general, with an improved development environment.

3.4 Contributions to Resources for Research and Education

The free availability of Charm++ allows researches from various areas to have a powerful software platform for the development of parallel applications. Also, for

some researchers who already created parallel versions of their codes via MPI, the use of AMPI allows such applications to benefit from recent Charm++ enhancements. Meanwhile, the various parallelization paradigms enabled by Charm++ provide a practical tool for students to grasp basic concepts of parallel computing and parallel performance optimization.

4 Plans for the Remainder of the Project

We will continue to develop Charj, both to allow the expression of the full range of Charm++ programs and to develop optimizations that are not feasible without compiler assistance. We will investigate ways to use Charj to simplify shared address space programming using the Charm++ programming model by applying a capability system to memory shared between communicating objects. We will investigate the use of compiler analysis to improve the efficiency of checkpointing and process migration in Charj programs. We will also develop Charj applications which explore common Charm++ application needs in an effort to identify more areas where Charj can enhance programmer productivity and to demonstrate the efficacy of Charj in simplifying the creation of parallel message-driven applications.

Similarly, we will further develop our abstractions for divide-and-conquer parallelism, aiming to apply automatic grain size control and agglomeration in a distributed memory context. Our goal is to demonstrate the utility of this framework for generative recursion by developing scalable applications that make use of it. We will also continue to refine and improve the existing SDAG and Charisma programming models.