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.