Chapter 8.13
Computing Gamma Calculus on Computer Cluster

Hong Lin
University of Houston-Downtown, USA

Jeremy Kemp
University of Houston-Downtown, USA

Padraic Gilbert
University of Houston-Downtown, USA

ABSTRACT

Gamma Calculus is an inherently parallel, high-level programming model, which allows simple programming molecules to interact, creating a complex system with minimum of coding. Gamma calculus modeled programs were written on top of IBM's TSpaces middleware, which is Java-based and uses a "Tuple Space" based model for communication, similar to that in Gamma. A parser was written in C++ to translate the Gamma syntax. This was implemented on UHD's grid cluster (grid.uhd.edu), and in an effort to increase performance and scalability, existing Gamma programs are being transferred to Nvidia's CUDA architecture. General Purpose GPU computing is well suited to run Gamma programs, as GPU's excel at running the same operation on a large data set, potentially offering a large speedup.

HIGHER-LEVEL PARALLEL COMPUTING: IMPLICIT PARALLELISM

Higher level parallel programming models express parallelism in an implicit way. Instead of imposing programmers to create multiple tasks that can run concurrently and handle their communications and synchronizations explicitly, these models allow programs to be written without assumptions of artificial sequenciality. The programs are naturally parallel. Examples of such kind of models include the Chemical Reaction Models (CRMs) (Banatre & Le Metayer, 1990, 1993), Linda (Carriero & Gelernter, 1989), and Unity (Chandy & Misra, 1988; Misra, 1989). These models are created to address higher level programming issues such as formal program specification, program synthesis, program derivation and verification, and software architecture. Efficient implementation

DOI: 10.4018/978-1-61350-456-7.ch8.13

of these models has limited success and therefore obscures its direct applications in software design (Creveui, 1991; Gladitz, 1996). Despite this limitation, efforts have been made in both academic and industrial settings to avail these models in real-world programming. For example, Unity has been used in industrial software design and found successful; execution efficiency of Linda has been affirmed by experiments and it is implemented by IBM Tuple Space. Recent discussions of these models in multi-agent system design have also been found in literature (Cabri, 2000). In the following discussion, we focus on the Chemical Reaction Models and its applications.

The Chemical Reaction Models describe computation as "chemical reactions". Data (the "solution") are represented as a multiset. A set of "reaction" rules is given to combine elements in the multiset and produce new elements. Reactions take place until the solution becomes inert, namely there are no more elements can be combined. The results of computation are represented as the inert multiset. Gamma is a kernel language in which programs are described in terms of multiset transformations. In Gamma programming paradigm, programmers can concentrate on the logic of problem solving based on an abstract machine and are free from considering any particular execution environment. It has seeded follow-up elaborations, such as Chemical Abstract Machine (Cham) (Berry & Boudol, 1992), higher-order Gamma (Le Metayer, 1994; Cohen & Muylaert-Filho, 1996), and Structured Gamma (Fradet & Le Metayer, 1998). While the original Gamma language is a first-order language, higher order extensions have been proposed to enhance the expressiveness of the language. These include higher-order Gamma, hmm-calculus, and others. The recent formalisms, γ-Calculi, of Gamma languages combine reaction rules and the multisets of data and treat reactions as first-class citizens (Banâtre, Fradet, & Radenac, 2004, 2005a, 2005b). Among γ-Calculi, γ_0-Calculus is a minimal basis for the chemical paradigm; γ_c-Calculus extends γ_0-Calculus by adding a condition term into γ-abstractions; and γ_n-Calculus extends γ_0-Calculus by allowing abstractions to atomically capture multiple elements. Finally, γ_{cn}-Calculus combines both γ_c-Calculus and γ_n-Calculus. For notational simplicity, we use γ-Calculus to mean γ_{cn}-Calculus from this point on.

The paper will be organized as follows. In the second section, we give a brief introduction to γ-Calculus. In the third and the fourth section, we discuss the method for implementing γ-Calculus in IBM Tuple space and in OpenCL, respectively. Experimental results are presented thereafter. We conclude in the last section.

γ-CALCULUS

The basic term of a Gamma program is molecules (or γ-expressions), which can be simple data or programs (γ-abstractions). The execution of the Gamma program can be seen as the evolution of a solution of molecules, which react until the solution becomes inert. Molecules are recursively defined as constants, γ-abstractions, multisets or solution of molecules. The following is their syntax:

M::= 0 | 1 | … | 'a' | 'b' | … ; constants

| γP[C].M ; γ-abstraction

| M_1, M_2 ; multiset

| <M> ; solution

The multiset constructor "," is associative and commutative (AC rule). Solutions encapsulate molecules. Molecules can move within solutions but not across solutions. γ-abstractions are elements of multisets, just like other elements. They can be applied to other elements of the same solution if a match to pattern P is found and condition C evaluates to true and therefore