Simulating Complex Dynamical Systems in a Distributed Programming Environment

E. V. Krishnamurthy[1] and Vikram Krishnamurthy[2]

[1]Computer Sciences Laboratory,
Australian National University, Canberra , ACT 0200, Australia
abk@discus.anu.edu.au

[2]Department of Electrical and Computer Engineering,
University of British Columbia,V6T 1Z4 Vancouver, Canada,
vikramk@ece.ubc.ca

Abstract. This paper describes a rule-based generic programming and simulation paradigm, for conventional hard computing and soft and innovative computing e.g., dynamical, genetic, nature inspired self-organized criticality and swarm intelligence. The computations are interpreted as the outcome arising out of deterministic, non-deterministic or stochastic interaction among elements in a multiset object space that includes the environment. These interactions are like chemical reactions and the evolution of the multiset can mimic the evolution of the complex system. Since the reaction rules are inherently parallel, any number of actions can be performed cooperatively or competitively among the subsets of elements. This paradigm permits carrying out parts or all of the computations independently in a distributed manner on distinct processors and is eminently suitable for cluster and grid computing.

1 Introduction

Most systems we observe in nature are complex dynamical systems that consist of a large number of degrees of freedom. They may contain several inhomogeneous subsystems that are spatially and temporally structured on different scales and characterized by their own dynamics. Such complex systems often exhibit collective ("Emergence") behaviour that is difficult to model. Stochastic and chaotic dynamical systems provide an efficient methodology for modelling and simulation of complex systems by capturing the behaviour of the system at different spatial and temporal scales. The simulation based approach of a stochastic or chaotic dynamical system can be viewed as "*soft computation*", since unlike in conventional computation where exactness is our goal, we allow for the possibility of error and randomness.

This paper describes a generic multiset programming paradigm for the simulation of complex systems. This paradigm permits us to write a generic program [2], [3] called a program shell - that implements the common control structure. It includes a few unspecified data types and procedures that vary from one application to another. Hence, this **U**nified **M**ultiset **S**imulation **P**aradigm (**UMSP**) can be used for all **conventional algorithms** [15], Tabu search, Markov chain Monte Carlo (MCMC), Particle Filters [6], **Evolutionary algorithms**-classifier systems, bucket brigade learning , **Genetic algorithms and Programming** [14], **Immunocomputing**, **Self-**

organized criticality [4] and **Active Walker models** (ants with scent or multiwalker-paradigm where each walker can influence the other through a shared landscape based on probabilistic selection [9], and **Biomimicry** [16]. Also it is applicable to non-equilibrium systems using oscillatory mechanisms involving catalytic reactions - as for example of producing ATP (Adenosine triphosphate) from ATP [8].

Structure of Unified Multiset Simulation Paradigm (UMSP)

The UMSP has the following features:

(i) One or more object spaces (multisets) that contain elements whose information is structured in an appropriate way to suit the problem at hand.

(ii) A set of interaction rules that prescribes the context for the applicability of the rules to the elements of an object space. Each rule consists of a left-hand side (a pattern or attribute) of named objects and the conditions under which they interact, and a right hand side that describes the actions to be performed on the elements of the object space, if the rule becomes applicable based on some deterministic or probabilistic criteria.

(iii) A control strategy that specifies the manner in which the elements of the object space will be chosen and interaction rules will be applied, the kinetics of the rule-interference (inhibition, activation, diffusion, chemotaxis) and a way of resolving conflicts that may arise when several rules match at once.

(iv) A mechanism to evaluate the elements of the object space to determine the effectiveness of rule application (e.g., evaluating fitness for survival).

Thus, UMSP provides a stochastic frame-work of ***"generate and test"*** for a wide range of problems, Yao [19], and Michalewicz and Fogel [14]. Also the system structure of UMSP consisting of components and their interaction is supported by contemporary software architecture design [2].

Computational Features of UMSP

The UMSP has the following computational features:

(i) Interaction -Based: The computations are interpreted as the outcome of interacting elements of the object space that produce new elements (or same elements with modified attributes) according to specific rules. Hence the intrinsic (genotype) and acquired properties due to interaction (phenotype) can both be incorporated in the object space. Since the interaction rules are inherently parallel, any number of actions can be performed ***cooperatively or competitively*** among the subsets of elements, so that the new elements evolve toward an equilibrium or unstable or chaotic state.

(ii) Content-based rule activation: The next set of rules to be invoked is determined solely by the contents of the object space as in chemical reactions.

(iii) Pattern matching: Search takes place to bind the variables in such a way to satisfy the left hand side of the rule. This characteristic of pattern (or attribute) matching makes the UMSP suitable for innovative computing.

(iv) Suitable for deterministic, non-deterministic and probabilistic modes.

(v) Choice of objects, and actions: We can use strings, arrays, sets, trees and graphs, multisets, tuples, molecules, particles and even points, as the basic elements of

computation and perform suitable actions on them by defining a suitable topology, geometry or a metric space.

We describe in Sections 2 and 3, the general properties of rule based paradigms. In Section 4 we give examples for UMSP. Section 5 contains the conclusion.

2 Rule -Based Programming Paradigm

Specification:

The main feature of the rule - based paradigm is the specification of the program:
G(R, A)(M) = If there exists elements a, b, c,.. in an object space M such that an interaction rule R (a, b, c,...) involving elements a , b, c is applicable then
G(R, A)((M- {a , b, c,.. }) + A(a, b , c,...)) else M.

Here M denotes the initial object space with components of appropriately chosen data type. This is a multiset or a bag in which a member can have multiple occurrences, Calude et al. [5]. The operator - denotes the removal (annihilation) of the interacted elements; it is the multiset difference; the operator + denotes the insertion (or creation) of new elements after the action A; this is multiset union of appropriately typed components. Note that R is a condition text (or interaction condition that is a boolean) that is used to check when some of the elements of the object space M can interact. The function A is the action text that describes the result of this interaction. Note that both R and A are exact and deterministic. Testing for R involves a deterministic search, and evaluation of truth or falsity of Boolean predicates. Also actions performed in A are assumed to be exact.

The function R can be interpreted as the query evaluation function in a database M and the function A can be interpreted as the updating function for a set of database instances. Hence, if one or several interaction conditions hold for several non-disjoint subsets of object space at the same time, the choice made among them can be nondeterministic or probabilistic. This leads to *competitive parallelism*. Then the actions on the chosen subset are executed atomically and committed. That is, the chosen subset undergoes an 'asynchronous atomic update'. This ensures that the process of matching and the follow-up actions satisfy the four important properties used in *Transaction Processing* [13] namely, ACID properties: Atomicity (indivisibility and either all or no actions or carried out), Consistency (before and after the execution of a transaction), Isolation (no interference among the actions), Durability (no failure). Once all the actions are carried out and committed the next set of conditions are considered. As a result of the actions followed by commitment, we derive a new database; this may satisfy new conditions of the text and the actions are repeated by initiating a new set of transactions. These set of transformations halt when there are no more transactions executable or the database does not undergo a change for two consecutive steps indicating a new consistent state of the database.

However, if the interaction condition holds for several disjoint subsets of elements in the database at the same time, the actions can take place independently and simultaneously. This leads to *cooperative parallelism*.

Deterministic and Nondeterministic Iterative Computation:
This consists of applications of rules that consume the interacting elements of the object space and produce new or modified elements in the multiset. This is essentially *Dijkstra's guarded command program.* It is well-known that the Guarded command approach serves as a universal distributed programming paradigm for all conventional algorithms with deterministic or nondeterministic components [10] So we will not elaborate on this aspect any further.
Termination: For the termination of rule application, the interaction conditions R have to be designed so that the elements in the object space can interact only if they are in opposition to the required termination condition. When the entire elements meet the termination condition, the rules are not applicable and the computation halts leaving the object space in an equilibrium state (or a fixed point).
Non–termination, instability, chaos: These cases arise when the rules continue to fire indefinitely as in chemical oscillations. Then the object space can be in a non-equilibrium state. It is also possible that the evolution leads to instability and chaos of the deterministic iterative dynamics.

For example, consider the rule-based iterative dynamical system: For X(0) in the range [-1,1], if $X(i) \geq 0$ then $G(X(i+1)) = -2X(i) + 1$; else $G(x(i+1)) = 2X(i)+1$.The rules $X(i)\geq 0$ and $X(i)<0$, are mutually exclusive and non-competitive; they generate a chaotic dynamical system, unstable, having a dense orbit in [-1,1].

3. Stochastic Rule-Based Paradigm

The introduction of stochastic mechanism (randomness) in a rule-based system has several advantages:
(i) It provides ergodicity of search orbits. This property ensures that searching is done through all possible states of the solution space since there is a finite probability that an individual can reach any point in problem space with one jump.
(ii) It provides solution discovery capabilities (as in genetic programming) and enables us to seek a global optimum rather than a local optimum.
(iii) It cuts down the average running time of an otherwise worst–case running time-algorithm. We achieve this gain by producing an output having an error with a small probability.
(iv) Applicable to problems in many discipline; Genetics (genetic algorithms); Thermodynamics (simulated annealing), Statistical Mechanics (Particle transport); Complex Systems (Active -walker, Self-organization and percolation models).

The unified multiset rule-based Simulation paradigm (UMSP) is obtained from the rule-based system described in Section 2, by introducing probabilities for selection to test whether one or more reaction conditions hold for several non-disjoint subsets at the same time. In this case, the choice made among these subsets is determined by a random number generator to randomly select the elements of the multiset with a probability p, test for the reaction conditions, and then perform the required actions.
UMSP is defined by the function:
PG (R (p(i), A) (M) = if there exists elements a,b,c,. .. belonging to an object space M (a multiset) such that R(a, b, c,...) then G(R,A)((M-{a, b, c,.. }) + A(a, b, c,..)) else M where each of the possible number of subsets i that satisfy the conditions R is randomly chosen with an appropriate probability p(i) and the corresponding text

of action A is implemented and the components of the multiset are updated appropriately. Further, if p(i) is not specified in a component program , the choice can be deterministic or nondeterministic. Thus a composite program can contain within itself the deterministic, nondeterministic and probabilistic components.

The implementation of UMSP consists of the following four basic steps:

Step 0: Initialization: Initializing the multiset representing the problem domain.

Step 1: Search: Deterministic or random searching for the candidate elements that satisfy a given rule (interaction condition) exactly or within a probabilistic bound..

Step 2: Rule Application: Carrying out the appropriate actions on these chosen the elements as dictated by the given rule.

Step 3: Stopping: It is typical in probabilistic method, not to explicitly state a stopping criterion. A key reason for this is that the convergence theory can provide only asymptotic estimates, as the number of iterations goes to infinity. However, in practice, we need to choose a suitable stopping criterion for the given problem - otherwise, we may be wasting the resources .

In step 3, we may use various acceptance criteria; these may be involve evaluating an individual element or a selected subset or the whole object space; that is, the evaluation of the object space can take place at different levels of granularity depending upon the problem domain. Also, the acceptance criteria may be chosen dependent or independent of the number of previous trials and the choice of probabilities can remain static or can vary dynamically with each trial. Thus depending upon the evaluation granularity, acceptance criteria and the manner in which the probability assignments are made, we can devise various strategies by suitably modifying the skeletal structure of UMSP. For example one may choose to select a reaction rule from a rule-base probabilistically or vary the frequency of application of competing rules. Also one may carry out any operation probabilistically. UMSP is suitable to optimize the structure of the model used as in Genetic Programming, or optimize its parameters as in Genetic algorithms.

4 Examples for Realisation of UMSP

Practical realisation of the UMSP and application to many different types of algorithms can be achieved through a coordination programming language, Multran, using Multiset and the concept of transactions [13]. Also UMSP can be implemented in the grid and cluster-computing environment using MPI [7,17]. Due to lack of space we can give only two examples.

(i) Swarm and Ant Colony Paradigm

A swarm (flock of birds, ants, cellular automata) is a population of interacting elements that can optimize some global objective through cooperative search of space [9]. Here, individual elements in the multiset are points in space, and change over time is represented as movement of points, representing particles with velocities and the system dynamics is formulated in UMSP using the rules:

1. **Stepping rule**: The state of each individual element is updated in many dimensions, in parallel, so that the new state reflects each element's previous best success ; e.g., ,the position and momentum (velocity) of each particle.

2. **Landscaping rule**: Each element is assigned a new best value of its state that depends on its past best value and a suitable function of the best values of its interacting neighbours, with a suitably defined neighbourhood topology and geometry.

Rule 1 reflects the betterment of the individual, while rule 2 reflects the betterment of the collection of the individuals in the neighbourhood as a whole, by evaluating the relevance of each individual and providing support for its activity. These two rules permit us to model Markovian random walks which is independent of the past history of the walk and non-Markovian random walks, dependent upon past history- such as self-avoiding, self-repelling and active random-walker models. This can result in a swarm (a self - organizing system) whose global nonlinear dynamics emerges from local rules due to stochasticity or chaos introduced by the parameter variation. Also, interesting new properties may show up- low dimensional attractors, bifurcations and chaos and various kinds of attractors having fractal dimensions presenting a swarm -like, flock-like appearances depending upon the Jacobian of the mapping; Wolfram [18].

(ii) Discrete Adaptive Stochastic Optimization

Consider the following discrete stochastic optimization problem. Let $\Theta = \{1,2,...,S\}$ denote a finite set and consider the following problem: Compute

$$\theta^* = \min_{\theta \in \Theta} E\{X_n(\theta)\}$$

where E denotes mathematical expectation and for any fixed $\theta \in \Theta$, $\{X_n(\theta)\}$ denotes a sequence of independent and identically distributed (iid) random variables that can be generated for any choice of $\theta \in \Theta$. If the density function of $X_n(\theta)$ is not known, it is not possible to analytically evaluate the above expectation and hence θ^*. In such a case, one needs to resort to simulation based stochastic approximation to compute the optimal solution θ^*.

A brute force approach of computing the optimal solution to the problem involves exhaustive enumeration over all Θ and proceeds as follows: For each $\theta \in \Theta$ generate a large number N of random samples $X_1(\theta), X_2(\theta), \ldots X_N(\theta)$. Then compute an estimate of $E\{X_n(\theta)\}$ using the sample average (arithmetic mean)

$$G_N(\theta) = (X_1(\theta) + X_2(\theta) + ... + X_N(\theta)/N.$$

By Kolmogorov's strong law of large numbers (which is one of the most fundamental consequences of the ergodic theorem for iid processes), $G_N(\theta) \to E\{X_n(\theta)\}$ with probability one as $N \to \infty$. This and the finiteness of Θ imply that

$$\arg\max_{\theta \in \Theta} G_N(\theta) \to \arg\max_{\theta \in \Theta} E\{X_n(\theta)\} \text{ as } N \to \infty.$$

However, the above brute force procedure is inefficient – evaluating $G_N(\theta)$ at values $\theta \in \Theta$ with $\theta \neq \theta^*$ is wasted effort since it contributes nothing towards evaluating $G_N(\theta^*)$. What is required is an intelligent dynamic scheduling (search) scheme that decides at each time instant which value of θ to evaluate next, given the current estimates, in order to converge to the maximum θ^* with minimum effort.

Here we present a globally convergent discrete stochastic approximation algorithm based on the random search procedures [1,11,12,20]. The basic idea is to generate a homogeneous Markov chain taking values in Θ which spends more time at the global optimum than at any other element of Θ. This consists of the following skeletal structural steps of UMSP .

Step 0: Initialization: At time n=0, select starting point $\theta_0 \in \Theta$ randomly with uniform probability. Set $D_0 = e_{\theta_0}$, where e_i denotes the S dimensional unit vector with 1 in the i th position and zeros elsewhere. Set initial solution estimate $\hat{\theta}_0 = x_0$.

Step 1: Search (Random Sampling): At time n, sample $u_n \in \Theta - \{\theta_n\}$ with uniform distribution.

Evaluate the random sample costs $X_n(\theta_n)$ and $X_n(u_n)$.

Step 2: Rule Application: If $X_n(\theta_n) > X_n(u_n)$ then set $\theta_{n+1} = \theta_n$, else set $\theta_{n+1} = u_n$.

Update duration time vector at time n+1 as $D_{n+1} = D_n + e_{\theta_n}$

Update estimate of maximum at time n as $\hat{\theta}_n = \arg\max_{i \in \{1,2,\ldots,S\}} D_{n+1}(i)$

Step 3: Stopping: Choose stopping criteria appropriately; if not satisfied set $n \to n+1$ and go to Step 1.

Then as proved in [1], under suitable conditions (e.g., if the density function with respect to which the expected value is defined above is symmetric) the estimate $\hat{\theta}_n$ generated by the above random search stochastic approximation algorithm converges with probability one to the global optimum θ^*. It is also shown in [1], that the algorithm is attracted to the global optimum, i.e., the algorithm spends more time at the global optimum than any other candidate value. That is, for sufficiently large n, the duration time vector D_n has it maximum element at θ^*.

The above algorithm has several applications. It can be used to learn the behaviour of an ion channel (large protein molecule) in a nerve cell membrane to estimate the Nernst potential efficiently [11]. In [12,20] its recursive version optimizes the spreading code of a CDMA spread spectrum transmitter over a fading wireless channel.

5 Conclusion

The introduction of stochastic/chaotic mechanisms in a multiset chemical reaction model provides a soft-computational model to study evolutionary biological, chemical and physical systems interacting with the environment. The paradigm described here provides a new environment using a distributed architecture for swarm intelligence, membrane and bio-immunology computing, adaptive stochastic optimisation and self organized criticality. This simulation paradigm is well-suited for cluster and grid computing using MPI.

References

1. S.Andradottir, Accelerating the convergence of random search methods for discrete stochastic optimization, ACM Transactions on Modelling and Computer Simulation, **9**(4) (1999), 349-380.
2. R.Backhouse and J.Gibbons, Generic Programming, Lecture Notes in Computer Science, Vol.2793,Springer Verlag , New York (2003)
3. J.-P.Banatre, D.L. Me'tayer, Programming by Multiset transformation Comm. ACM, **36** , (1993) 98 -111.
4. S.Boettcher and A. Percus, Nature's way of Optimizing, Artificial Intelligence, **119** (2000),275-286,.
5. C.S.Calude et al., Multiset Processing, Lecture Notes in Computer Science, Vol.2235,Springer Verlag, New York (2001)
6. A.Doucet,et al., Particle Filters for State Estimation of Jump Markov Linear Systems, IEEE Trans Signal Processing, **49**, 613-624 (2001)
7. W. Gropp, et al., *Using MPI*, M.I.T Press, Cambridge, Mass,(1992)
8. D.-Q.Jiang,et al.,Mathematical Theory of Nonequilibrium Steady states, Lecture Notes In Mathematics, Vol 1833,Springer Verlag, New York (2004)
9. J.Kennedy, and R.C.Eberhart, Swarm Intelligence,Morgan Kauffman,London (2001).
10. E.V.Krishnamurthy, Parallel Processing, Addison Wesley, Reading., Mass.,(1989)
11. V.Krishnamurthy and S.H.Chung, Adaptive Learning Algorithms for Nernst Potential and I-V curves in Nerve Cell Membrane Ion Channels modelled as Hidden Markov Models, IEEE Transactions NanoBioScience, **2** (4), 266-278 9(2003)
12. V.Krishnamurthy, et al., Adaptive Spreading Code Optimization in Multiantenna Multipath Fading Channels in CDMA, IEEE Conference on Communications, Anchorage, May 2003.
13. W.Ma et al,.,Multran - A coordination Programming Language Using Multiset and Transactions, Proc. Neural, Parallel and Scientific Computing, **1**, 301-304, Dynamic Publishers, Inc.,U.S.A.,(1995).
14. Z. Michalewicz, D.B Fogel, How to solve it : Modern Heuristics , Springer Verlag, New York (2002).
15. V.K.Murthy and E.V. Krishnamurthy, Probabilistic Parallel Programming based on Multiset transformation, Future Generation Computer Systems.**11**(1995)283-293.
16. K.M.Pacino, Biomimicry of bacterial foraging for distributed optimisation and control , IEEE Control System Magazine, **22**(3)(2002),52-68.
17. M.Snir et al.:MPI: The complete Reference, M.I.T.Press,Cambridge, Mass.(1995)
18. S.Wolfram: A New kind of Science, Wolfram Media Inc., Champaign, Ill (2002)
19. X.Yao, The evolution of Evolutionary computation, Lecture Notes in Artificial Intelligence, Vol.2773, 19-20 (2003).
20. G.Yin, et al. Regime Switching Stochastic Approximation Algorithms with application to adaptive discrete stochastic optimization, SIAM Journal of Optimization, to appear.