Center for Embedded and Cyber-Physical Systems
University of California, Irvine

A SystemC model for N-body problems and its Parallel Design Space Exploration

Kasra Moazzemi
Rainer Doemer
Aparna Chandramowlishwaran

Technical Report CECS-16-09
November 28, 2016

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

http://www.cecs.uci.edu

A SystemC model for N-body problems and its Parallel Design Space Exploration

Kasra Moazzemi
Rainer Doemer
Aparna Chandramowlishwaran

Technical Report CECS-16-09
November 28, 2016

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

moazzemi@uci.edu
http://www.cecs.uci.edu

Abstract

This project focuses on exploring N-body problems on SystemC simulation environment. The goal is to first evaluate the computation and communication patterns of various N-body problems, then to map these problems into suitable architectures. The flexibility of SystemC environment makes this design space exploration feasible and expendable. The massive parallelism in N-body algorithms make this problem a good candidate for benchmarking parallel SystemC. As an example of N-body problems, we evaluate particle simulation in 2D space and show the scalability of this space simulation in parallel SystemC simulation. The 2D space is divided into smaller areas that each are mapped to one of the computational tiles that will run on one thread. At the end, we evaluate the trend of speedup with respect to the number of threads on several multi-core hosts.

Contents

List of Figures

List of Tables

A SystemC model for N-body problems and its Parallel Design Space Exploration

Kasra Moazzemi
Rainer Doemer
Aparna Chandramowlishwaran
Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
moazzemi@uci.edu
http://www.cecs.uci.edu

Abstract

This project focuses on exploring N-body problems on SystemC simulation environment. The goal is to first evaluate the computation and communication patterns of various N-body problems, then to map these problems into suitable architectures. The flexibility of SystemC environment makes this design space exploration feasible and expendable. The massive parallelism in N-body algorithms make this problem a good candidate for benchmarking parallel SystemC. As an example of N-body problems, we evaluate particle simulation in 2D space and show the scalability of this space simulation in parallel SystemC simulation. The 2D space is divided into smaller areas that each are mapped to one of the computational tiles that will run on one thread. At the end, we evaluate the trend of speedup with respect to the number of threads on several multi-core hosts.

1 Introduction

In recent years, there have been many works on massively parallel applications and the architectures that can support such parallelism. One of the classes of parallel applications is called N-Body problems. In physics, the N-body problem is the challenge of predicting the individual motions of a group of celestial objects interacting with each other gravitationally. Solving this problem has been motivated by the desire to understand the motions of the Sun, Moon, planets and the visible stars. In the 20th century, understanding the dynamics of globular cluster star systems became an important N-body problem. The N-body problem in general is considerably more difficult to solve. One important feature of these problems is how to simulate systems that contain these N-body problems.

The main focus of this project is designing and analysis of high performance architectures for N-body problems. This consists of designing the system in SystemC environment and simulating their performance during execution of large problems with N-body structure. This includes multiple stages of system design, each consisting of various refinement steps. For example, planning the architecture based on the structure of the problem and then mapping small parts of the N-body problem to operational units. This mapping requires analysis on computational requirement of the problem in addition to communication pattern to other units. Simulating performance of architecture with various configurations is expected to show strong and weak points of traditional and proposed architectures in running large-scale scientific problems. This analysis would bring insight on how to design future high performance architectures for such applications. In this work, we will use particle simulation as a well-known N-body problem.

The remainder of this report is organized as fol-

lows: After a brief description of N-body problems in Section 2, we will explain simulation of these problems in Section 3. In Section 4, we then explain about SystemC simulation. Section 5 describes the architecture used in this work. We then list challenges in this domain in Section 6 and then show the result of our Particle simulation in parallel SystemC in Section 7. Finally we'll conclude this report in Section 8.

2 Definition of N-body problems

N-body problem in domain of physics can be defined as the problem of predicting the individual motions of a group of celestial objects interacting with each other gravitationally[1]. Solving this problem has been motivated by the desire to understand the motions of the Sun, Moon, planets and the visible stars. In the 20th century, understanding the dynamics of globular cluster star systems became an important n-body problem[2]. The n-body problem in general relativity is considerably more difficult to solve.

The classical physical problem can be informally stated as: given the quasi-steady orbital properties (instantaneous position, velocity and time)[3] of a group of celestial bodies, predict their interactive forces; and consequently, predict their true orbital motions for all future times

2.1 History

One of the first cases of these type of problems started with curiosity about the orbital positions of planet's orbit. Isaac Newton was able to produce equations to predict planet's motion, but later on he found out these equations are not completely accurate and realized it was because gravitational interactive forces amongst all the planets was affecting all their orbits.

This discovery was the start of shaping what a N-body problem is physically the realization of necessity of knowing gravitational interactive forces have to be known in addition to three orbital positions of planets was the one of the first definitions of a N-body problem.

2.2 General formula

The n-body problem considers N point masses m_i, i=1,2,...,N in an inertial reference frame in three dimensional space R3 moving under the influence of mutual gravitational attraction. Each mass m_i has a position vector $\mathbf{q}_i$. Newton's second law says that mass times acceleration $m_i d^2\mathbf{q}_i/dt^2$ is equal to the sum of the forces on the mass. Newton's law of gravity says that the gravitational force felt on mass m_i by a single mass m_j is given by [9]:

$$\mathbf{F}_{ij} = \frac{Gm_i m_j(\mathbf{q}_j - \mathbf{q}_i)}{\left\|\mathbf{q}_j - \mathbf{q}_i\right\|^3}, \tag{1}$$

where G is the gravitational constant and $\left\|\mathbf{q}_j - \mathbf{q}_i\right\|$ is the magnitude of the distance between $\mathbf{q}_i and \mathbf{q}_j$ (metric induced by the ℓ_1 norm).

Summing over all masses yields the n-body equations of motion:

$$m_i \frac{d^2\mathbf{q}_i}{dt^2} = \sum_{j=1, j\neq i}^{N} \frac{Gm_i m_j(\mathbf{q}_j - \mathbf{q}_i)}{\left\|\mathbf{q}_j - \mathbf{q}_i\right\|^3} = \frac{\partial U}{\partial \mathbf{q}_i} \tag{2}$$

where U is the self-potential energy

$$U = \sum_{1\leq i<j\leq N} \frac{Gm_i m_j}{\left\|\mathbf{q}_j - \mathbf{q}_i\right\|}. \tag{3}$$

3 N-body simulation

The N-body problem can be defined as the problem of simulating the movement of points or particles or bodies under the influence of some type of force. Depending on the type of force, there are numerous applications ranging from astrophysics, molecular dynamics, fluid dynamics, to computer graphics and machine learning. Mathematically, given a system of N source particles, with positions given by Y1,...,Yn and N targets with positions X1,...,Xn we wish to compute the N sums,

$$f(x_i) = \sum_{j=1}^{N} K(x_i, y_i).s(y_j), i = 1, ..., N \tag{4}$$

where f(x) is the desired potential at target point x, s(y) is the density at source point y, and K(x, y) is

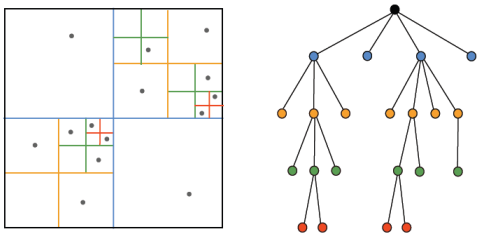

Figure 1: An adaptive quadtree in 2-D with one particle in each leaf node.

an interaction kernel that specifies the physics of the problem. For instance, the single-layer Laplace kernel [14], K(x, y)= $\frac{1}{4pi}\frac{1}{||x-y||}$ might model electrostatic or gravitational interactions.

There are a number of algorithms for computing the potential and its derivative force exerted on the target particles by the source particles. They can be broadly classified into two categories, namely direct and approximation algorithms. Evaluating these sums using a direct algorithm requires O(N^2) operations. However, there are computationally less expensive approximation algorithms which reduce the complexity to O(N logN) and even O(N). One such class of approximation algorithms are called tree-methods, which use a tree data structure as shown in Figure 1 to hierarchically decompose the particles.

4 Parallel SystemC simulation

For the evaluation and design space exploration of cyber-physical and embedded system models, high speed parallel simulation is critical. Traditional Discrete Event Simulation (DES), which is used in reference simulators such as SystemC [10][8], manages a set of explicitly specified concurrent threads by a central scheduler. The simulation is driven by events and time advances (delta- and time-cycles), but follows cooperative multi-threading semantics. Simulation executes sequentially, only one thread at any time. As such, regular DES cannot utilize the parallel processing capabilities of modern multi- or many-core hosts.

In contrast, synchronous Parallel Discrete Event Simulation (PDES) [12][6][5] follows a partial temporal order and executes threads in parallel if and only if they run at the same simulated time. Usually, this approach results in significant speedup compared to traditional DES. Nevertheless, simulation cycles are still absolute barriers which pose an obstacle to increasing the simulation speed for many application models [4].

To overcome these limitations, advanced PDES techniques are needed. Our approach, Out-of-Order Parallel Discrete Event Simulation (OoO PDES) [13] breaks the simulation barrier by using local per-thread time stamps and is known as the fastest technique for ESL simulation today. OoO PDES utilizes advanced compiler technology for static data and event conflict analysis. This allows the simulator to run threads in parallel and out-of order even in different delta and time cycles if there are no conflicts. Thus, the simulator can execute the maximum number of threads in parallel, resulting in highest simulation speed [11] without the loss of accuracy. This technology can also be applied to the use of modern many-core target platforms, such as the latest Intel Xeon Phi Many Integrated Core (MIC) architecture. Here, we can expect multiple orders of magnitude speedup with 100 percent accuracy in the simulation and performance evaluation of novel computing architectures.

5 Architecture model

Our current model simulates a system containing a test bench and a grid of N * N functional units. The goal of the test bench is only to initialize the particle space for simulation and provide each functional unit with a portion of the entire space for simulation. The rest of the simulation is done on functional units called Design Under Test (DUT). A vast number of DUTs and their correspondence to each other makes this structure similar to a distributed system but the flexibility to optimize the DUTs for computing complex particle interactions and implementing a hand tuned communication infrastructure suited for particle simulation makes this system unique. System level design gives us the capability to reduce the cost and increase time-to-market while exploring for architecture with desired performance for such problem.

Figure 2(a) shows a sample space consisting of

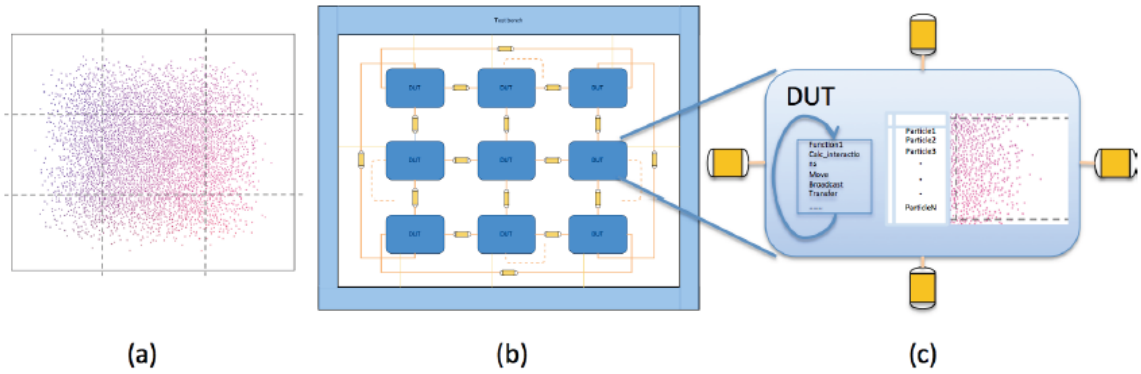

Figure 2: Particle simulation modeling:(a) Particle space divided into nine spaces (b) Sample system with 9 DUTs (c) Schematic of one DUT

thousands of particles. Figure 2(b) describes a system generated with nine DUTs to host the particle space simulation. Each portion of the space will be mapped to one of the DUTs. Figure 2 (c) shows a DUT unit schematic that is connected to its four direct neighbors through reliable channels and have a control connection to the main test bench. This unit will receive particles assigned to it from the test bench at beginning of the simulation. In each iteration, this unit will first analyze the forces toward every particle from all other particles, then apply an advance movement to each particle. This computation has to be done for all particles during one iteration. Afterwards, this DUT will pass the particles exiting its borders to its corresponding DUT. In addition, center of mass in every portion of space can be broadcast to all other DUTs. Due to the fact that this broadcast is an expensive communication it's optional to have this function based on the particle space characteristics.

The simulation time for even one time-step is much longer in comparison to simulating only few milliseconds of particle behavior in real time. This is due to massive parallel movement of particles in space and their interaction with one another. This parallel nature in particle simulation makes it suitable for parallel simulation on a system with large number of cores.

Our current prototype of SystemC model functions up to 64 different DUTs. This will enable us to simulate a reasonably large particle space to capture real-application characteristics.

The next issue in such simulations is the communication cost between cores. The communication between DUTs is handled through reliable channels for passing the particles to other spaces. This communication is required to transfer necessary information about the state of each space and particles to other functional units. This communication can be a bottleneck in most distributed systems. Therefore communication core arrangement such as torus is evaluated for this system and communication primitives used for transferring data, such as single burst transfer, one-to-all broadcast, and all-to-all broadcast, are considered for this problem.

6 Challenges

We encountered several challenges during the modeling of particle simulation. Bellow we explain some of these challenges and the solution we advised for each of them.

6.1 Particle Simulation Modeling

Major issues happen when the simulation moves to multiple functional units or DUTS. Some of these problems are how to transfer the information about the escaping particles to neighbor DUTs, how to realize a particle have to bounce or jump and so on. T

The first problem was the memory management for each DUT. Previously, every DUT had a fixed size

memory allocated for a fixed number of particles. This has changed since particles can escape boundaries of one DUT and move to another unit. To fix this problem we allocated extra memory with static size that can be used to host more particles. In addition to having an advance movement from particles, we also implement a memory management system for each DUT to handle recycling of the extra memory.

If we only broadcast center of mass of each portion of space to other DUTs, we have no knowledge about the particles close to our border on the other neighbor DUTs which have direct impact on particles this side of the border. This would result in inaccuracy when two particles pass one region of the border at the same time from opposite directions. As they are not aware of each others existence, they may even pass through each other on border jump. In addition, only broadcasting center of mass doesn't provide any useful information by itself. In future work we plan to explore the exchange of information between DUT units for each round of simulation.

6.2 Simulation environment issues and our solutions

In this section we will discuss some of the environment related issues we faced during simulation. First issue was memory leak. This problem happened when a big number of particles moves to a specific DUT and it runs out of memory. Solution, the DUT stops the simulation and terminates to avoid memory leaks. Next step would be implementing a recycling function to reuse the memory of the particles that left the DUT.

Another issue was particles crossing the boundaries. Particles in adjacent DUTs which are close to the boundaries don't get effected by each other. This is an important issue specially if the critical effecting distance can reach to other units particles.

Final issue was preprocessing of port binding. In order to generate massive systems with large number of DUTs, it's inefficient to write the complete system manually. One solution to this problem is to first use a preprocessing script to generate the system specification then compile it. Our script works up to 64 cores connected to test bench. The issue appears when number of DUTs go beyond 64. The SystemC compilation fails in port binding stage. Further study shows there is a limit on binding by order which can be resolved by using function "bind" in SystemC.

6.3 Configurations

Here is a summery of the configuration knobs of our modeling system and their effect on particle simulation:

NUM DUT: This parameter specifies the number of computational units instantiated in the simulation. Each DUT will be responsible for part of particle simulation.

DUT ROW and DUT COL: So far the assumption is the arrangement of the DUT is in a square with perfect square number of DUTs. In future we might want to change the arrangement so we can use these configurations to specify number of DUTs in each row and column.

NSTEPS: shows the number of iterations the simulation will run for. We can limit the simulation using this parameter.

SAVE FILE: Setting this parameter to 1 saves the coordinates of particles during simulation. This has some overhead during simulation but can be used to prepare a demo for simulation.

SAVEFREQ: In some cases we may want to avoid saving the coords in every iteration. Save to file will take a major time of the simulation. By changing the value of this step parameter we can save the coords every SAVEFREQ steps.

N: This parameter shows the number of active particles in EACH DUT! In future we may want to change this to contain number of all active particles in the space.

EXTRA SPACE: This parameter specifies extra space allocated in EACH DUT for accommodating the escaping particles from neighbor DUTs.

DELAYS: These set of parameters will change the delays embedded in each part of system to measure the simulation time.

Initialization: Depending on the type of the simulation we might want to give different initialized acceleration and voracity. This might help us to replicate a falling of particles or every particle moving towards center and other cases.

7 Experiments and Results

In this section we describe some of the results from simulating particle simulation with different number of particles in space and division of this space to various numbers of computational units to show the amount of parallelism achieved.

7.1 Communication vs Computation

There are some factors that are important in this exploration. One aspect that we look at is the ratio of computation and communication complexity depending on how large our particle space is and what is the granularity of divided chunks dedicated to each Design Under Test (DUT) unit. We analyze this based on two different scenarios that can later on be mapped to communication model of different N-body problems. First one only communicate to adjacent tiles the information regarding the movement of particles. The second method, each DUT not only communicates with neighbor tiles but also broadcast some information like center of mass or escaping particles to all other tiles. This information can later on be used to decide if tiles in further distance have any force to one tile based on the cut-off measurement in every simulation.

Simulation Specs	4 DUTs	9 DUTs
Execution Time	22.5s	50s
Computation	129444sc_ns	250000sc_ns
Neighbor	6719sc_ns	8847sc_ns
Neighbor+Broadcast	18719sc_ns	29693sc_ns

Table 1: Computation and communication simulation delays in a sequential reference simulation

Table 1 shows the result on 500 iteration of simulation on a simple particle space with 250 particles for each tile. It's important to notice that execution time is measure in time and the delays are measured in simulation time. The reason for this is we wanted to analyze the overhead of communication between DUTs on the amount of computation done. We also notice a large overhead when we add the broadcast method due to blocking transfer between these units.

We can observe using reliable channels implements similar to SpecC channels [7] a synchronization is needed every time two tiles need to communicate. This synchronization in case of broadcast has been done first in row wise fashion then DUTs in the same column exchange data. This method enables us to extend the simulation to any square number of tiles without any need to change the communication protocols.

7.2 Parallel Speedup

Out of order Parallel SystemC [12][5] enables the execution of independent tiles to run in parallel to each other. This would enable every thread to run on a core in concurrent to each other with low synchronization overhead. One of our main focuses in this project is to show the abilities of parallel SystemC in optimizing the simulation of highly parallel systems depending on the hardware they are running on.

We illustrate simulation speedup that parallel SystemC can deliver comparing to sequential SystemC simulation. The experiments are done four different machine to show the scalability of this environment. Table3 shows the speedup of four different machines with various number of cores and threads. Tables below also show individual speedup for each machine.

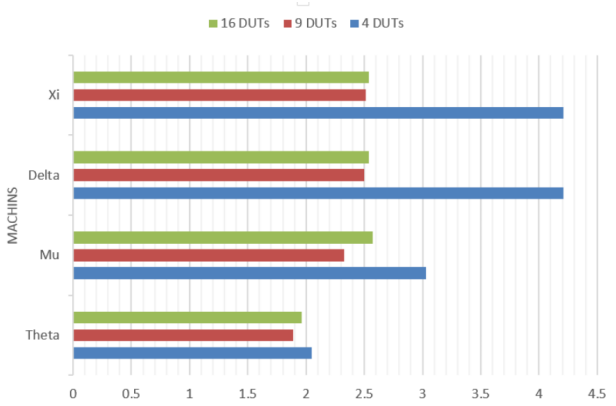

Figure 3: Parallel speed up based on number of machines cores

Machine name	Number of CPUs	Number of cores	Number of threads
Theta	1	2	2
Mu	1	4	1
Delta	1	4	2
Xi	2	6	2

Table 2: Four test machines configurations

Number of tiles	Parallel	CPU usage	Sequential	CPU usage	**Speedup**
4 DUTs	49.97s	177	102.7s	104	2.05
9 DUTs	120s	176	227s	104	1.89
16 DUTs	204.45s	187	401.49s	104	1.96
25 DUTs	319s	188	627	104	1.96

Table 3: Speedup in simulation using parallel SystemC on a 2 core machine

8 Conclusion and future work

In this project we have done a design space exploration on N-body applications. As an example we used particle simulation and demonstrated Parallel SystemC ability to speedup simulation process. Computation and communication patterns of particle simulation throughout various software/hardware mapping. We have divided the particle space into multiple tiles and mapped each tile to a parallel thread. Finally we have evaluated the speedup we can gain from simulating the particle simulation on parallel SystemC. This work can be further expanded to accommodate many other massively parallel applications.

References

[1] Sverre J Aarseth. Gravitational n-body simulations, tools and algorithms. *Cambridge University Pres*, 2003.

[2] Sauer T. D. Alligood, K. T. and J. Yorke. Chaos: An introduction to dynamical systemss. *Springer*, pages 44–46, 1996.

[3] Donald D.; Bate, Roger R.; Mueller and Jerry White. Fundamentals of astrodynamics. *Dover*, 1971.

[4] W. Chen, X. Han, C. W. Chang, G. Liu, and R. Dmer. Out-of-order parallel discrete event simulation for transaction level models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12):1859–1872, Dec 2014.

[5] R. Dmer, W. Chen, X. Han, and A. Gerstlauer. Multi-core parallel simulation of system-level description languages. In *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, pages 311–316, Jan 2011.

[6] R. M. Fujimoto. Parallel discrete event simulation. In *Proceedings of the 21st Conference on Winter Simulation*, WSC '89, pages 19–28, New York, NY, USA, 1989. ACM.

[7] Andreas Gerstlauer, Rainer Dömer, Junyu Peng, and Daniel D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.

[8] IEEE Computer Society, http://www.systemc.org. *IEEE Standard 1666-2011 for Standard SystemC Language Reference Manual*, 2011.

[9] Dan Offin Kenneth Meyer, Glen Hall. *Introduction to Hamiltonian Dynamical Systems and the N-Body Problem*.

[10] Open SystemC Initiative, http://www.systemc.org. *Functional Specification for SystemC 2.0*, 2000.

Number of tiles	Parallel	CPU usage	Sequential	CPU usage	**Speedup**
4 DUTs	19s	305	80	106	4.21
9 DUTs	70s	439	176s	105	2.51
16 DUTs	123	461	313s	105	2.54

Table 4: Speedup in simulation using parallel SystemC on a 4 core machine

Number of tiles	Parallel	CPU usage	Sequential	CPU usage	**Speedup**
4 DUTs	31s	317	94s	105	3.03
9 DUTs	89s	275	208s	104	2.33
16 DUTs	141s	323	363s	104	2.57

Table 5: Speedup in simulation using parallel SystemC on a 4 core hyper threaded machine

[11] Weiwei Chen R. Domer and Xu Han. Parallel discrete event simulation of transaction level models. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2012.

[12] Che wei Chang Weiwei Chen, Xu Han and R. Domer. Advances in parallel discrete event simulation for electronic system-level design. *Design Test, IEEE*, 2013.

[13] Xu Han Weiwei Chen and R. Domer. out-of-order parallel simulation for esl design. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, March 2012.

[14] Lexing Ying, George Biros, and Denis Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2):591–626, 5 2004.

Number of tiles	Parallel	CPU usage	Sequential	CPU usage	**Speedup**
4 DUTs	32s	318	111	105	3.46
9 DUTs	54	519	243	105	4.5
16 DUTs	97	705	423s	104	4.36

Table 6: Speedup in simulation using parallel SystemC on a 6 core hyper threaded machine