

LA-UR-03-1164

Approved for public release;
distribution is unlimited.

c.1

Title:

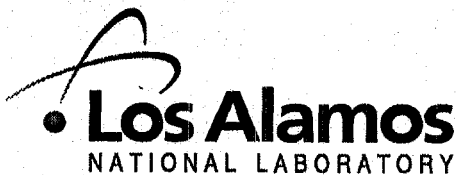
A PRELIMINARY STUDY OF
MOLECULAR DYNAMICS ON
RECONFIGURABLE COMPUTERS


Author(s):

Christopher Wolinski
Frans Trow
Manya Gokhale

Submitted to:

EXSA 103



Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by  University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Form 836 (8/00)

99

A Preliminary Study of Molecular Dynamics on Reconfigurable Computers

Christophe Wolinski
Los Alamos National Laboratory
Los Alamos, NM, U.S.A.
IRISA, IFSIC France

Frans Trouw
Maya Gokhale
Los Alamos National Laboratory
Los Alamos, NM, U.S.A.

Abstract

In this paper we investigate the performance of platform FPGAs on a compute-intensive, floating-point-intensive super-computing application, Molecular Dynamics (MD). MD is a popular simulation technique to track interacting particles through time by integrating their equations of motion.

One part of the MD algorithm was implemented using the Fabric Generator (FG)[11] and mapped onto several reconfigurable logic arrays. FG is a Java-based toolset that greatly accelerates construction of the fabrics from an abstract technology independent representation. Our experiments used technology-independent IEEE 32-bit floating point operators[4] so that the design could be easily re-targeted. Experiments were performed using both non-pipelined and pipelined floating point modules.

We present results for the Altera Excalibur ARM System on a Programmable Chip (SoPC), the Altera Stratix EP1S80, and the Xilinx Virtex-II Pro 2VP50. The best results obtained were 5.69 GFlops at 80MHz(Altera Stratix EP1S80), and 4.47 GFlops at 82 MHz (Xilinx Virtex-II Pro 2VP50). Assuming a 10W power budget, these results compare very favorably to a 4Gflop/40W processing/power rate for a modern Pentium, suggesting that reconfigurable logic can achieve high performance at low power on floating-point-intensive applications.

1 Introduction

The goal of our work is to evaluate the performance of modern reconfigurable logic arrays on compute-intensive, floating-point-intensive algorithms. The algorithm chosen, Molecular Dynamics, is a computer simulation technique whereby the time-varying interactions of particles are followed by computing each particle's equations of motion.

In the realm of Molecular Dynamics (MD), simulation is a common computational technique used in the physical sciences to bridge the gap between the individual and collective properties of atoms[9]. It is possible to write down a description of the forces between atoms as a function of their separation, but there are no analytical methods for extrapolating this information to predict the properties of for instance a protein. This problem is dealt with by solving the classical equations of motion for the atoms using numerical methods.¹

In this work, we used Mindy[2], a simulation code for sequential computers developed by the University of Illinois. Mindy can be used to study the interactions of large biomolecular systems.

2 Related Work

Molecular Dynamics

Application of the MD method has always been constrained by the limitations imposed by computational speed constraints. In the late seventies and eighties, most of the then cutting-edge work was carried out on the Cray series of supercomputers, as the MD problem is inherently vectorizable. Some initial work was carried out to map the problem onto parallel architectures

¹There are also approaches based on quantum mechanics but these are not considered here.

such as the ICL DAP, and the Thinking Machines CM-5. With the advent of fast workstations, some of this work moved to RISC processors, and currently the approach is to use clusters of commodity RISC processors connected with low latency network connections. One of the limitations of the cluster approach is the requirement that the nodes must communicate the positions of the particles on a regular basis, and this has led to application specific coarse-graining of the algorithm.

Another approach that continues to be tried is the development of application specific integrated circuits (ASICs), but these suffer from the problem of almost instant obsolescence and high cost. A current example is the Molecular Dynamics Machine at the Institute of Physical and Chemical Research (RIKEN) in Japan, which was developed in conjunction with IBM[7].

The ultimate goal of our work is to gain a significant fraction the performance of an ASIC, without the prohibitively high cost and inherent obsolescence of that approach.

Floating Point on Reconfigurable Computers

In the realm of floating point implemented on reconfigurable logic, there has been considerable interest in obtaining efficient floating point performance on reconfigurable computers (RC). Data intensive signal and image processing applications, which have been successfully mapped to RC, are usually developed in floating point, and it has been necessary to translate to fixed point in order to obtain high performance on RC. An early study [8] revealed that customized floating point representations could be efficiently be implemented on RC, but that standard representations consumed excessive area and/or time. More recent evaluation [3] has demonstrated 95 MFlops floating point multiplication on Xilinx Virtex E using a format optimized for DSP calculations. A proprietary floating point library for the Xilinx [6] shows average performance of 180 MFLOPS for IEEE-754-compliant variable wordlength floating point arithmetic cores. However, when floating point is just one aspect of a complex application, the results may not bear out as well. For example, [5] concludes that "despite the sophistication of today's FPGA architecture and EDA tools, floating-point precision is still not feasible in FPGA-based Artificial Neural Networks."

Thus the purpose of our study is not only to quantify the performance of floating point operators, but to examine the performance of a complex floating-point-intensive application on reconfigurable computers.

3 Molecular Dynamic Algorithm

The MD method is based on a repetitive process to solve the equations of motion for typically up to a few million atoms in simple systems, and tens of thousands of atoms in the more complex simulations of solvated proteins and model biological membranes. A variety of interatomic potentials are used, and these typically encompass a short-range repulsive term, medium range attractive term, and a long-range electrostatic component. The heart of the calculation requires that

- all permutations of the interatomic distances be calculated,
- the forces evaluated from the analytical interatomic potentials, and
- a net force calculated on each atom reflecting its interaction with the surroundings.

This net force is then used in a predictor corrector algorithm to move the positions of the particles forward in time by a small timestep, δt . This timestep is typically small and on the order of a femtosecond as the algorithm does not take into account the fact that all of the other atoms move at the same time, which in turn changes the forces on each atom. A typical finite difference method used for the integration step (predictor corrector) is the Verlet algorithm:

$$r_i(t + \Delta t) = 2r_i(t) - r_i(t - \Delta t) + a_i(t) \times \Delta t^2 \quad (1)$$

where $r_i(t)$ is the position of particle i at time t , and $a_i(t)$ is the net force on atom i at time t arising from all of the other atoms. This equation is usually applied in two half steps, where the positions (p) and velocities (v) are incremented, followed by a calculation of the forces at the new position, and finally an update of the velocities as shown in Equations 2 through 4:

$$r_i(t + \Delta t) = r_i(t) + \Delta t \times v_i(t) + 1/2 \Delta t^2 \times a_i(t) \quad (2)$$

$$v_i(t + \Delta t) = v_i(t) + 1/2 \times \Delta t \times a_i(t) \quad (3)$$

calculation of forces at new positions, $r_i(t + \Delta t)$

$$v_i(t + \Delta t) = v_i(t + 1/2 \times \Delta t) + 1/2\Delta t \times a_i(t + \Delta t) \quad (4)$$

Although the calculation of the forces is the most compute intensive part of the calculation, it is also the most complicated. The full MD algorithm requires features such as periodic boundary conditions, nearest image convention, neighborhood tables, with an escalating complexity as the size of the problem increases. We anticipate that the traditional MD algorithms will need to be adapted to the FPGA architecture, and so a direct comparison with conventional microprocessors is non-trivial. As this is a floating point intensive code, our initial work has evaluated the effectiveness of a FPGA to carry out the Verlet algorithm, which encompasses some of the typical computational effort required in a molecular dynamics loop.

4 Polymorphous Computing Fabric and Fabric Generator Toolset

In this study, we wanted to experiment with different implementations on different technologies. In order to easily generate and evaluate many variations of the basic algorithm, we used the notion of a polymorphous fabric [10], a cellular array with regular communication patterns both within the array as well as to an attached processor. Each version of the algorithm was designed as a “fabric” consisting of simple, inter-connected computational datapath cells, each with an optional local memory.

In our model, the collection of local memories forms a (dual-ported) global memory that can be loaded and examined from the attached processor on the SoPC. The cells composing the fabric need not all be the same – a fabric may contain groups of homogeneous cells, as illustrated in Figure 1, in which different cell types are distinguished by different names (“Send,” “Rec,” “P,” and “Ele”). Each datapath cell may have its own controller, or alternatively, a group of identical cells may share a controller. Many different sorts of communications patterns may be realized within a single fabric. The fabric can be considered as a processing memory and presents a standard memory to the embedded processor.

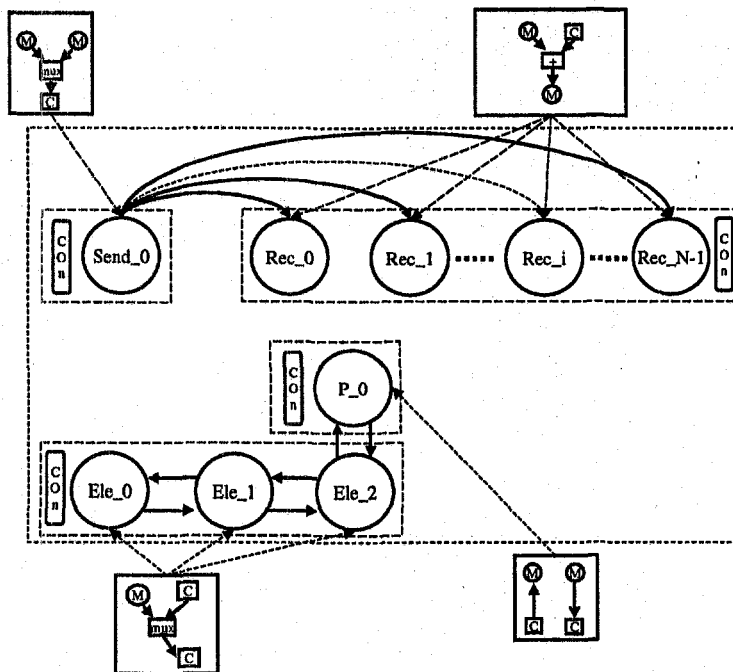


Figure 1. Example Fabrics

We have built a Fabric Generator FG [11] to help accelerating construction of the fabrics. The FG library contains classes to

- define a module

```

for (j=0; j<natoms; j++)
{
    pos[j] += dt*vel[j] + 0.5*dt*dt*f[j]*imass[j];
    vel[j] += 0.5*dt*f[j]*imass[j];
}

```

Figure 2. C code for Verlet Algorithm

- create a datapath of interconnected modules
- instantiate cells consisting of datapaths with associated sequencers
- create a fabric of interconnected cells

The fabric designer writes a Java program and calls methods provided by the library to define and instantiate modules. In addition, modules may be defined in VHDL using technology-specific components, if desired, and instantiated in the fabric program. Modules can be instantiated to build a datapath. Next, cells may be defined. A cell contains a datapath and specifies a sequencer (controller). Cells may share a sequencer (SIMD mode) or each cell may have a unique sequencer (MIMD mode). A fabric contains a collection of cells and sequencers. The fabric program calls a method to generate a sequencer description file. After the fabric designer has microcoded a low level control sequence for a datapath, an assembler generates a state machine. The Controllers, Fabric, and Datapaths in the Figure are then synthesized through the standard CAD tool chain. The result is a component with standard memory interface which can be connected to any processor.

5 Reconfigurable Computer Implementation

Figure 2 shows the Mindy [2] loop computing equations 3 and 4. Our first step was to tile the loop so that each “processor” on the RC computes a subset of the atom positions and velocities. The tiled version of the loop is shown in Figure 3.

```

int M = natoms/N; where N is a number of cells
for (i=0; i<N-1; i++)
{
    for (j=0; j<M-1; j++)
    {
        k = M*i+j;
        pos[k] += dt*vel[k] + dt*f[k]*M[k]; where M[k]=0.5*dt*imass[k]
        vel[k] += f[k]*M[k];
    }
}

```

Figure 3. Tiled Verlet Loop

Using the FG toolset, two fabrics were initially implemented on the Altera Excalibur ARM (see 4), both utilizing multi-cycle IEEE 32-bit floating point modules. One fabric uses non-pipelined floating point modules, while the second uses pipelined modules. In this experiment we wanted to find out which fabric has a better performance/space ratio.

Each fabric is composed of a collection of N cells, and each cell realizes the inner loop from figure 3 for M number of atoms. By virtue of the fabric model, on the Altera Excalibur ARM, the ARM processor has direct access to all data memories.

Since each computational pipeline is independent, communication channels between cells are not required for either design. In the case of the first fabric, each cell has separated controller because the *stop* signals emitted by the non-pipelined adders are asynchronous between cells, and thus must be caught by each cell’s controller. For the second fabric, since the floating point modules are synchronized, a single controller can be used for all the cells.

Figure 4 shows the data-path of each fabric. In data path 1, the registers A, B, C, D and E are used to enable the pipelined execution of the graph composed of non-pipelined multi-cycles floating points modules. The module *synch* in data-path 1 synchronizes the *stop* signals coming from the floating point adders. The output *synch* signal determines the *end* of the

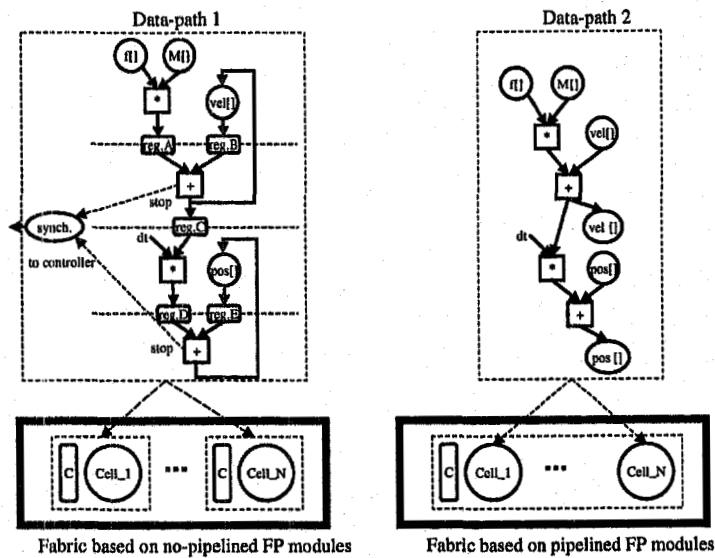


Figure 4. Fabrics implementations.

pipeline stage composed of several clock cycles. In data-path 2, the synchronization module is not required since the floating point modules are pipelined, resulting in a simpler implementation.

The fabrics were implemented on the Altera ARM Excilibur system (see Figure 5). In this system, the user logic can communicate with the ARM processor through the *dual-port memory* or the *bridge*. We have shown in our previous work [1] that the communication through the dual-port memory is 10 times faster than the bridge. For this reason a DMA component was introduced between the fabric and the dual port memory to make communication transparent between the ARM processor and the fabric.

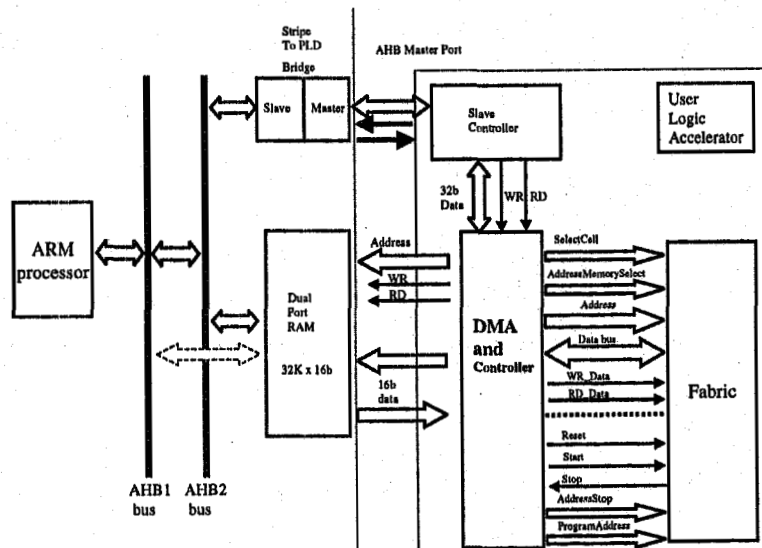


Figure 5. Implementation on ARM Altera Excilibur SoPC.

Figure 6 shows performance and the resources consumed for each fabric with N=5.

ARM Excalibur Altera SoPC	Fabric based on no-pipelined <i>FP</i> modules	Fabric based on pipelined <i>FP</i> modules
performance	132 MFlops	560 MFlops
latency	~24 cycles, new data every ~8 cycles	22 cycles, new data every 1 cycle
logic elements in gates	26025	25220
logic elements in %	65 %	63 %
memories in blocks	20 * (128x32)	30 * (128x32)
memories in %	25 %	37.5%
performance/space	5	22

Figure 6. Results of Implementation on SoPC.

Figure 6 shows that the pipelined fabric is 4.2 times faster than the non-pipelined one. Both fabrics consume approximately the same number of resources. The non-pipelined fabric used simpler floating point units than the pipelined one, but its cell's data-path must integrate supplementary registers and a synchronization module to pipeline the overall execution. In addition, each cell of the non-pipelined fabric has a separate controller. The only disadvantage of the pipelined fabric is that it requires a larger number of memories.

We then mapped the pipelined fabric, which has the better performance/space ratio, onto the Altera Stratix EP1S80 and the Xilinx Virtex-II Pro 2VP50. The results are presented in Figure 7.

	Altera Stratix EP1S80	Xilinx Virtex-II Pro 2 VP50
number of fitted cells	20	15
clock frequency	80 MHz	82 MHz
performance	5.69 GFlops	4.47 GFlops
used logic resources in %	89 % †	61 %
number of logic cells in %	100 % †	131 %
number of multipliers in %	100 % †	61 %
memories in %	100 % †	58 %

†The number of resources on the Altera Stratix EP1S80 are considered as 100 % to give a relative comparison of two circuits.

Figure 7. Results of implementation on EP1S80 and 2VP50 circuits.

In the case of the Altera Stratix EP1S80, the fabric composed of the N=20 cells fit the component, while in the case of the Xilinx Virtex-II Pro 2VP50, only N=15 cells would fit. This was due to limitations in routing connections between the distributed memories and the multipliers. Even though the number of available resources exceeded the needs of the design, only 15 cells could be successfully routed.

The clock speeds obtained are comparable. However, since the Stratix could fit more cells, it achieves 5.69GFlops, while the Virtex-II Pro delivers 4.47GFlops.

6 Conclusion

In this work, we have conducted a preliminary study of the capabilities of platform FPGAs to run a compute-intensive, floating-point intensive kernel. The selected loop is part of a larger Molecular Dynamics simulation. It implements Verlet's algorithm, the predictor corrector position update calculation. The study shows that the Altera Stratix and Xilinx Virtex-II Pro are capable of an impressive floating point rate on this kernel, even when non-optimized, technology-independent floating point modules are used. The performance/watt – a factor of 10X – compared to Pentium class processors make reconfigurable computers even more attractive.

However, for the MD application, the position update, which we have benchmarked, is only one part of the overall computation. The other aspects include inter-atomic distance calculations and force calculation, which are also extremely compute-intensive. Current algorithmic methods to perform these calculations involve complex data structures and linked list traversals. Thus we foresee that the traditional MD methods will need to be adapted to map efficiently onto reconfigurable

fabric-based architectures. Our future work lies in developing representations and algorithms for MD that can be mapped onto simple, pipelined computational data-paths.

References

- [1] M. Gokhale, J. Frigo, K. McCabe, J. Theiler, C. Wolinski, and D. Lavenier. Experience with a hybrid processor: K-means clustering. *Journal of Supercomputing*, 2003.
- [2] J. Gullingsrud. Mindy - a minimal molecular dynamics program. <http://www.ks.uiuc.edu/Development/MDTools/mindy/>, 2001.
- [3] J. Dido, N. Geraudie, I. Loiseau, O. Payeur, Y. Savaria, and D. Poirier. A lexible floating-point format for optimizing data-path and operators in fpga dsps. *FPGA*, February 2002.
- [4] M. Leeser, M. Estlick, N. Kitaryeva, J. Theiler, and J. Szymanski. Applying Reconfigurable Hardware to Segmentation for Multi-spectral Imagery. In *HPEC 2000*, Boston, MA, Sept. 2000.
- [5] K. R. Nichols, M. A. Moussa, and S. M. Areibi. Fleasibility of floating-point arithmetic in fpga artificial neural networks. *MAPLD*, September 2000.
- [6] Quixilica. Quixilica floating point cores. www.quixilica.com/pdf/qx.fpl.pdf, 2002.
- [7] RIKEN. Molecular dynamics machine. <http://atlas.riken.go.jp/mdm/>, 2002.
- [8] N. Shirazi, A. Walters, and P. Athenes. Quantitative analysis of floating point arithmetic on fpga based custom computing machines. *IEEE International Conference on FPGAs for Custom Computing Machines*, April 1995.
- [9] M. A. D. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, 1987.
- [10] C. Wolinski, M. Gokhale, and K. McCabe. A new polymorphous computing fabric. *IEEE Micro*, Sept. 2002.
- [11] C. Wolinski, M. Gokhale, and K. McCabe. Rapid construction of reconfigurable computing fabrics for systems on a programmable chip. *HPCA/SSRA*, Feb. 2003.