31st Annual International Conference of the IEEE EMBS
Minneapolis, Minnesota, USA, September 2-6, 2009

SIMENGINE: A Low-Cost, High-Performance Platform for Embedded Biophysical Simulations

Randall K. Weinstein, Christopher T. Church, Carl S. Lebsack, Joshua E. Cook, and Michael E. Sorensen
Simatra Modeling Technologies
Atlanta, Georgia 30308
Email: info@simatratechnologies.com

Abstract—**Numerical simulations of dynamical systems are an obvious application of high-performance computing. Unfortunately, this application is underutilized because many modelers lack the technical expertise and financial resources to leverage high-performance computing hardware. Additionally, few platforms exist that can enable high-performance computing with real-time guarantees for inclusion into embedded systems — a prerequisite for working with medical devices. Here we introduce SIMENGINE, a platform for numerical simulations of dynamical systems that reduces modelers' programming effort, delivers simulation speeds 10–100 times faster than a conventional microprocessor, and targets high-performance hardware suitable for real-time and embedded applications. This platform consists of a high-level mathematical language used to describe the simulation, a compiler/resource scheduler that generates the high-performance implementation of the simulation, and the high-performance hardware target. In this paper we present an overview of the platform, including a network-attached embedded computing device utilizing field-programmable gate arrays (FPGAs) suitable for real-time, high-performance computing. We go on to describe an example model implementation to demonstrate the platform's performance and describe how future development will improve system performance.**

I. INTRODUCTION

Numerical simulations of dynamical systems are an indispensable research tool for fields as diverse as biology, electronics, economics, and climatology. These fields all use numerical simulations to explain how systems behave and to guide the development of new technologies and products. The structure of these models naturally lends itself to implementation in high-performance computing applications[1], [2], [3].

Unfortunately, high-performance computing is underutilized because of the difficulty and expense of creating high-performance simulations. Scientists and engineers who develop models and simulations ("modelers") often lack the technical expertise to build robust, high-performance implementations of their models. Although modelers understand the math and the science underlying the construction of their models, they are frequently unfamiliar with the nuances of high-performance programming — such as threading, vectorization, and resource scheduling — that produce high-performance simulations. As a result, modelers must rely on others to create high-performance code for them, adding an undesirable delay and additional cost to their simulations. Finally, many modelers lack the financial resources to acquire or maintain large, high-performance computing systems.

Additionally, real-time simulations that can be used to "close the loop" with the real world are becoming increasingly important in basic research, laboratory instrumentation, and clinical applications. Traditional hardware used for high-performance computing, such as computer clusters, supercomputers, and graphics processing units (GPUs), lack real-time interfaces and scheduling capabilities and hence can not be used in for real-time applications. In addition, these systems are often too large, too costly, or consume too much power to be included in embedded applications. On the other hand, hardware used for embedded computing, such as programmable logic controllers (PLCs), digital signal processors (DSPs), and field-programmable gate arrays (FPGAs), either lack the computational horsepower required for complex numerical simulations or are prohibitively difficult to program for modelers without experience in hardware design.

To address these problems, we have developed SIMENGINE, a platform for numerical simulation of dynamical systems. Although we initially developed this platform as a tool for computational biologists, the design of SIMENGINE is such that it can be applied to a broad range of applications in the natural and engineering sciences. This platform is composed of three components: a high-level mathematical description language that enables modelers to describe their simulations in straightforward terms, a compiler that translates the simulation description into high-performance code, and a high-performance hardware target. The goal of the SIMENGINE platform is to enable high-performance simulation for individual modelers. Therefore, our current development has focused on low-cost, low-power hardware systems that can be easily installed and maintained by the end user, such as multi-core processors, GPUs, and FPGAs.

Here, we discuss the ability of SIMENGINE to target *FPGAs*: reconfigurable semiconductor devices containing many discrete programmable logic blocks that can be configured to perform a set of custom computations. In effect, a custom processor is created for each model, providing a performance gain 10–100× that of a standard single core general-purpose microprocessor, along with substantial benefits for power consumption[4]. Additionally, FPGAs are frequently used as embedded computing devices, and have been shown to be an ideal target for high-performance, real-time computing[2]. Although FPGAs are notoriously difficult to program, the SIMENGINE platform automatically translates high-level model descriptions into FPGA-compatable code, substantially reducing the programming effort on the part of the modeler.

978-1-4244-3296-7/09/$25.00 ©2009 IEEE

II. Platform Architecture

The SIMENGINE platform is built on three components: a model description language, a compiler, and a high-performance hardware device. Together, these three components enable modelers to easily describe their models and simulations, automatically build a high-performance simulation engine from their model description, and execute that simulation engine on a customized, high-performance processor.

A. Model Description Language

The front-end to the SIMENGINE platform is a domain-specific, dynamical system language (DIESEL) that allows modelers to describe their systems in straightforward mathematical terms. DIESEL is a fully-featured functional and object-oriented programming language. It incorporates a framework of data types based on concepts familiar to modelers, such as states, parameters, constants, and outputs, and also includes language primitives for differential and difference equations.

A DIESEL model description defines states in terms of equations that characterize the evolution of the system from one simulation iteration to the next. Parameters are input quantities whose value is defined by the user at runtime. Constants are unchanging input quantities defined at compile time. Outputs may correspond to states, or they may represent *intermediate quantities*, which are computed from states, parameters, and constants.

Figure 1 shows an example of DIESEL code implementing the Lorenz attractor, a chaotic dynamical system described by the following equations:

$$\frac{dx}{dt} = \sigma(y - x) \quad (1)$$

$$\frac{dy}{dt} = x(\rho - z) - y \quad (2)$$

$$\frac{dz}{dt} = xy - \beta z \quad (3)$$

```
model Lorenz
  state x (-100 to 100 by 1e-8) = 0
  state y (-100 to 100 by 1e-8) = 0
  state z (-100 to 100 by 1e-8) = 0

  parameter sigma (0 to 20 by 0.1) = 10
  parameter rho (0 to 30 by 0.1) = 28
  parameter beta (0 to 10 by 0.01) = 8/3

  equations
    x' = sigma*(y-x)
    y' = x*(rho - z) - y
    z' = x*y - beta*z
  end

  solver = forwardeuler(0.001)
end
```

Fig. 1. An example of DIESEL code representing the equations governing the Lorenz attractor.

B. Compiler

The SIMENGINE compiler translates a DIESEL model description into high-performance code. The compiler operates in two phases: a parsing phase that interprets the DIESEL model syntax into a flexible, graph-based *internal representation*, and an optimizing phase that translates the internal representation into efficient, targeted code, creating a custom simulation engine for the user's model (see Figure 2).

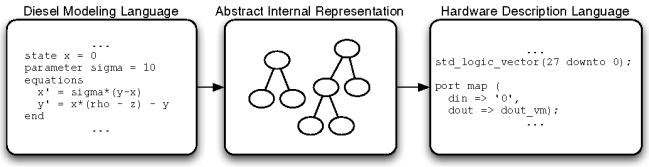

Fig. 2. Schematic depiction of the flow of data through the SIMENGINE compiler.

The parsing phase of the SIMENGINE compiler translates a DIESEL model description into an abstract representation, suitable for targeting various simulation platforms. The compiler interprets the model description, producing a list of differential and difference equations that define the changes in state values over time. The parser transforms differential equations and higher-order difference equations into first-order difference equations. These equations make up the abstract model representation in the form of directed flow graphs.

In the optimization phase, several transformations are performed on the internal representation. For example, the optimizer consolidates redundant expressions, simplifies algebraic expressions, and removes constant evaluations from the iteration loop. Additional optimizations may be performed depending on the final hardware target. For instance, when targeting an FPGA, the optimizer converts numerical operations to use fixed-point data, constructs look-up tables for estimating bounded functions, and applies an heuristic mapping algorithm to optimally allocate hardware resources.

At the end of the optimization phase, the SIMENGINE compiler generates targeted excution code for one or more simulation engines. For instance, the generated code may include a Verilog implementation of the engine for the FPGA simulation target as well as C code implementing a software simulation. The compiler also generates the communications channel between the simulation engine and the client user interface. Additionally, when targeting the FPGA hardware device, the compiler automatically uses the Xilinx ISE[1] toolkit to synthesize the Verilog implementation into a bitstream and to program the bitstream onto the FPGA.

C. Hardware Platform

The FPGA hardware platform described herein is the prototype of a network-attached computing appliance currently under development. (Figures 3 and 4) This device contains two Xilinx Virtex-4 family FPGAs[2]. The primary FPGA is the

[1]ISE is a registered trademark of Xilinx, Inc.

[2]Xilinx and Virtex-4 are registered trademarks of Xilinx, Inc.

Fig. 3. The computation FPGA is a Xilinx Virtex-4 XC4VSX35 FPGA. The host FPGA is a Xilinx Vertex-4 XC4VFX12 FPGA, incorporating a PowerPC 405 core and a Gigibit Ethernet interface. The two FPGAs are interconnected via a 2 Gbps proprietary interface.

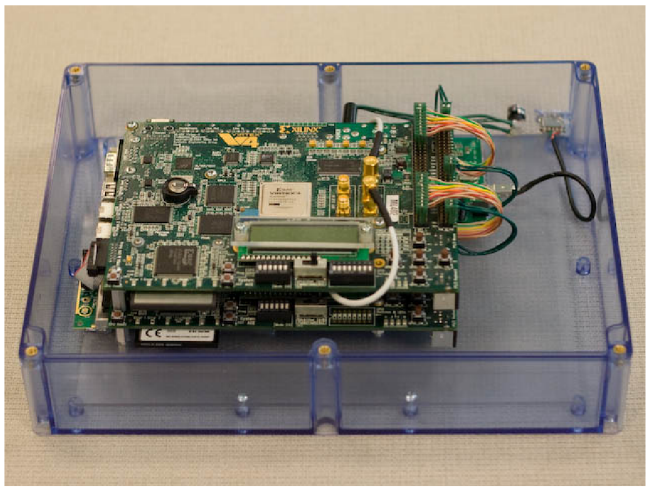

Fig. 4. The prototype hardware device.

computation FPGA, which implements the high-performance simulation engine. The computation FPGA contains 192 18-bit by 18-bit signed multipliers, 192 18-bit by 1024 element block RAMs, and 15,360 basic logic blocks. Each basic logic block contains two binary look-up tables (LUTs) and two single-bit registers. The second FPGA is the *host FPGA*, which provides the communications link between the computation FPGA and the simulation client, where the user issues simulation commands and receives output data. The host FPGA incorporates a PowerPC core running an embedded OS, which manages a Gigabit Ethernet interface between the device and the simulation client. The computation FPGA and host FPGA are connected via a proprietary high-speed data interconnect. The total cost of the hardware platform is less than a high-end computer workstation, placing it well within the financial reach of most researchers.

In an embedded configuration, digital I/O ports can be exposed for direct data interfaces between the model controller of the computation FPGA and external sensors, filters, or data converters. The efficient implementation of a model may execute much faster than the actual process being simulated. In this case, the simulation rate of the engine may then be tuned by adding delays or integrator enables in order to match the simulation's time scale to the computation time scale, enabling the potential for real-time interfacing.

III. Performance Results

We now illustrate the performance of the SIMENGINE platform through a case study of a stomatogastric (STG) neural model[5]. This section first describes the STG model, then describes its implementation using the SIMENGINE platform, and then discusses model performance on the FPGA hardware target.

A. STG model description

The STG model represents a neuron from the stomatogastric ganglion of the lobster. The behavior of the STG neuron model is characterized by the shape of the voltage trajectory of the membrane potential. The STG model has three main behavior types: silent, spiking and bursting. One spiking and two bursting behaviors are illustrated in the voltage traces found in Figure 5. These behaviors are distributed across the parameter space and can be further subdivided based on resting potential, spike frequency, spikes per burst, and burst area.

The STG model was chosen for this case study as it is characteristic of many conductance-based physiologically-grounded neural models. Full details of the model description and the full equations can be found in [5].

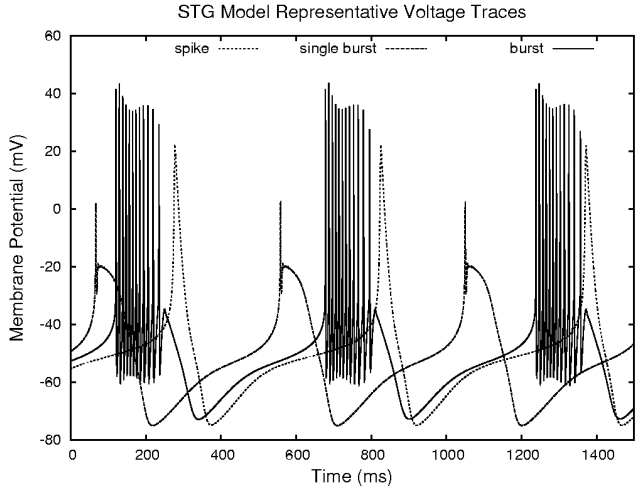

Fig. 5. Distinct model behaviors of the STG model. Traces taken from FPGA hardware using the SIMENGINE platform.

B. STG model implementation

Using DIESEL the entire STG model is represented with only sixty lines of code, compared to nearly one thousand lines of C++ used in the original study. This reduction in code size represents a substantial decrease in programmer overhead, allowing modelers to more easily implement complex models in less time than with traditional approaches.

This small code size is enabled by several features of the DIESEL language and the SIMENGINE platform. First, the DIESEL language was designed to provide a terse representation of mathematical equations that resemble their canonical forms. Second, the language is designed around dynamical

system models and provides the use of libraries of commonly used constructs such as integration methods. Finally, the SIMENGINE platform automatically produces all of the *glue* code and logic to enable the user to interact and communicate with a running model.

C. STG Results

The performance of the DIESEL STG model on the SIMENGINE FPGA platform is more than adequate for real-time computation. With a time step of 0.05 ms (the time step used in [5]), the computation FPGA can be run at 80 MHz. It takes 41 cycles to compute a single iteration of the model. The data interface to the FPGA imposes a minimal performance hit to the simulation rate, and as a result the resulting simulation can be run at 97.6× the speed of real time. For comparison, the same model implemented in C by SIMENGINE on a 2.5 GHz Intel processor executes at only 6.7× real time. The FPGA therefore provides a nearly 15-fold improvement in performance over a conventional microprocessor, with the added benefit that the FPGA can be readily incorporated into a embedded system for interfacing with physical systems. Additionally, we expect that continued development of the SIMENGINE platform will yield improved performance, producing FPGA-based simulations with greater than a 100-fold gain in performance over a microprocessor. From previous studies of hand-built FPGA-based simulations[2], [3], these expected improvements are quite realistic.

The FPGA implementation of the STG model produces results representative of those found in the original study. Example activity taken from the FPGA is shown in Figure 5. There are, however, certain differences between FPGA and microprocessor-based implementation, which can cause the results to diverge slightly from the original study. These are:

- *Fixed-point numerics*: FPGAs used fixed-point representations of numbers rather than the floating-point representation used in general-purpose microprocessors. SIMENGINE automatically generates appropriate fixed-point ranges and precisions for the FPGA-based implementation to minimize the difference between floating- and fixed-point simulations.
- *Look-up tables*: Functions that are difficult to compute, such as transcendental functions, are approximated using look-up-tables. This technique is similar to the technique used in many microprocessor-based simulators to improve simulation speeds [6], but will be slightly different than when the functions are explicitly solved.
- *Method of integration*: The forward-Euler integration method was used for the STG's membrane potential on the FPGA, whereas in the original study the exponential-Euler method was used. This change caused slight changes in behavior at the boundary between different activities (such as the boundary between spiking and bursting behavior).

Altogether, the differences in behavior between the fixed-point and floating-point simulations were minimal. Furthermore, like the original model, the auto-generated FPGA simulation engine produces output behaviors similar to those observed empirically from biological neurons. Future improvements to the SIMENGINE platform will further minimize the differences between the FPGA-based simulation and its floating-point counterpart.

IV. CONCLUSION

The SIMENGINE platform is the first platform designed to map high-level descriptions of dynamical systems automatically to an FPGA implementation. FPGAs are an ideal platform for realizing complex, computationally-intensive applications for both high-performance computing and embedded systems. By utilizing an FPGA, biophysical models can be simulated at orders of magnitude greater rates than with conventional processors and at significantly less cost than multiprocessor clusters or supercomputers. Furthermore, real-time guarantees available through FPGAs provide a path for embedded simulations of models that are too complex or have time scales too fast for traditional real-time OS or DSP technologies.

The results presented here used an FPGA from the Virtex-4 family of Xilinx FPGAs. Future development will update the FPGA hardware target to use either Virtex-5 or Virtex-6 FPGAs, which will provide improved performance and capacity. These benefits will allow for more complex and time-critical simulations to be implemented on the hardware target. Additionally, built-in GPIO banks will provide additional expandability and interfacing options.

V. ACKNOWLEDGEMENTS

Thanks to Dr. Robert Lee of Emory University Dept. of Biomed. Eng. for his advice on this project. This work was supported through NINDS SBIR award NS057859.

REFERENCES

[1] R. Weinstein, M. Reid, and R. Lee, "Methodology and Design Flow for Assisted Neural-Model Implementations in FPGAs," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 15, no. 1, pp. 83–93, 2007.

[2] E. Graas, E. Brown, and R. Lee, "An FPGA-based approach to high-speed simulation of conductance-based neuron models," *Neuroinformatics*, vol. 2, no. 4, pp. 417–435, 2004.

[3] M. Sorensen and S. DeWeerth, "Functional Consequences of Model Complexity in Rythmic Systems: I. Systematic Reduction of a Bursting Neuron Model," *Journal of Neural Engineering*, vol. 1, no. 4, pp. 179–188, 2007.

[4] D. Thomas, L. Howes, and W. Luk, "A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation.," in *Procedding of the ACM/SIGDA international Symposium on Field Programmable Gate Arrays.*, pp. 63–72, 2009.

[5] A. Prinz, C. Billimoria, and E. Marder, "Alternative to hand-tuning conductance-based models: construction and analysis of databases of model neurons," *Journal of neurophysiology*, vol. 90, no. 6, pp. 3998–4015, 2003.

[6] J. Bower and D. Beeman, *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* Springer, 2003.