Acceleration of Spiking Neural Networks in Emerging Multi-core and GPU Architectures

Mohammad A. Bhuiyan, Vivek K. Pallipuram and Melissa C. Smith

Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634, USA
{mbhuiya, kpallip, smithmc}@clemson.edu

Abstract—**Recently, there has been strong interest in large-scale simulations of biological spiking neural networks (SNN) to model the human brain mechanisms and capture its inference capabilities. Among various spiking neuron models, the Hodgkin-Huxley model is the oldest and most compute intensive, whereas the more recent Izhikevich model is very compute efficient. Some of the recent multi-core processors and accelerators including Graphical Processing Units, IBM's Cell Broadband Engine, AMD Opteron, and Intel Xeon can take advantage of task and thread level parallelism, making them good candidates for large-scale SNN simulations. In this paper we implement and analyze two character recognition networks based on these spiking neuron models. We investigate the performance improvement and optimization techniques for SNNs on these accelerators over an equivalent software implementation on a 2.66 GHz Intel Core 2 Quad. We report significant speedups of the two SNNs on these architectures. It has been observed that given proper application of optimization techniques, the commodity X86 processors are viable options for those applications that require a nominal amount of flops/byte. But for applications with a significant number of flops/byte, specialized architectures such as GPUs and cell processors can provide better performance. Our results show that a proper match of architecture with algorithm complexity provides the best performance.**

I. INTRODUCTION

Research in neuroscience is motivated by the performance of a mammalian brain, which can perform cognitive tasks more reliably and faster than a silicon-based processor. Among various proposed models, the spiking neural network (SNN) models are considered to be some of the most biologically accurate models to characterize neuronal dynamics in the brain [1, 2]. Some of the SNN models have already proven their biological fidelity by accurately reproducing neuronal activity [2]. Izhikevich shows that two of the most biologically accurate models include (in the order of biological accuracy): 1) Hodgkin-Huxley [3] and 2) Izhikevich [4]. Of these, the Hodgkin-Huxley (HH) model is the most compute intensive, while the Izhikevich model is the most computationally efficient. In this work, we have investigated and implemented SNNs of these two models on several processor and accelerator architectures.

Conventional single core processors are limited by serial computation and limited memory bandwidth. Emerging multi-core and GPU architectures can exploit task and thread-level parallelism and therefore are attractive for large-scale SNN simulations. In this work we utilize a character recognition algorithm based on the Izhikevich spiking neuron model presented in a previous paper [5] and scale the SNN from 9264 neurons up to over 5 million neurons, corresponding to an image size of 2400×2400. The primary contributions of this work are:

1) Analysis of the thread-level and data-level parallelization for two spiking neuron models in a character recognition SNN.
2) Network scalability and speedup performance of four architectures (GPU, PS3, Xeon, and Opteron) for SNN models.
3) Analysis of several optimization techniques available for all four architectures as applied to these SNN models.

Our optimized parallel implementations of the models have achieved speedups greater than 110x on the GPU, 80x on Xeon, 70x on PS3, and 60x on AMD over the serial implementation on a 2.66 GHz Intel Core 2 Quad for various image sizes. Speedups observed for the two neuron models were dependent on the ratio of computation to communication and how they were mapped to the architectures.

Section II of this paper provides a background on the spiking neural model and the network, while section III presents the four architectures examined. Section IV discusses optimization techniques and section V presents experimental set up. Section VI and section VII present the parallelization and mapping of the models and results, while section VIII concludes the paper.

II. BACKGROUND

A. *Spiking Neural Models*

Over the last 50 years, several models [6] have been proposed that capture the spiking mechanism within a neuron. In this paper, we examine two of the most biologically accurate spiking neuron models for implementation on multi-core architectures.

The Hodgkin–Huxley (HH) model is considered to be one of the most biologically accurate spiking neuron models. It consists of 4 differential equations and a large number of parameters (25 compared to 4 in the Izhikevich model) describing the neuron membrane potential, activation of Na and K currents, and inactivation of Na currents. The model can describe almost all types of neuronal behavior if its parameters are properly tuned. This model is valuable for the studying of neuronal behavior and dynamics as its parameters are biophysically meaningful and measurable. We have used a 0.01 ms time step for updating the four differential equations to provide sufficient numerical simulation accuracy as described in [20].

Izhikevich proposed a new spiking neuron model in 2003 that is based on only two differential equations and four parameters. The model is attractive for slower computing systems since it requires fewer computations than the Hodgkin Huxley model (13 flops as opposed to 246 flops per neuron update) but can still reproduce almost all types of neuron responses that are seen in biological experiments. In our implementation we have used a 1 ms time step to update the neuron dynamics providing sufficient numerical accuracy as discussed in [2].

B. *The Two-Level Network*

The SNN network used in this study consists of two levels [7], where the first level acts as a collection of input neurons and the second level acts as a collection of output neurons. A binary input image is presented to the first level of neurons, each image pixel corresponding to a separate input neuron. The number of output neurons is equal to the number of training images. Each input neuron is fully connected to all of the output neurons as shown in Fig. 1. Each neuron has an input current that is used to evaluate its membrane potential.

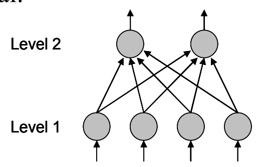

Figure 1. Network used for testing spiking models.

If the membrane potential crosses a certain threshold during a cycle, the neuron is considered to have fired. For a level 1 neuron, the input current is zero if the neuron's corresponding pixel in the input image is "off". If the pixel is "on," a constant current is supplied to the input neuron. A level 2 neuron's overall input current is the sum of all the individual currents received from the level 1 neuron connections. This input current for a level 2 neuron is given by the following equation:

$$I_j = \sum_i w(i,j) f(i)$$

Where, *w* is a weight matrix and *w(i,j)* is the input weight from level 1 neuron *i* to level 2 neuron *j*; and *f* is a firing vector where *f(i)* is 0 if level 1 neuron *i* does not fire and a 1 if the neuron does fire. A single fire at level 2 indicates that an image has been detected.

The elements of the weight matrix *w* are determined through a training process as described in [8]. The network was trained to recognize the 48 different input images given in [8] and later scaled for larger networks. In this study we have accelerated the recognition phase of each network on the processor architectures to be discussed in Section III.

C. *Related Work*

Several groups have studied applications using spiking neural networks. In general, these studies utilize the "*integrate and fire*" model. Thorpe developed the SPIKENET software for simulating spiking neural networks [9]. The system can be used for several image recognition applications, including identification of faces, fingerprints, and video images. Johansson and Lansner developed a large cluster based spiking network simulator of a rodent-sized cortex [10]. They tested a small-scale version of the system to identify 128×128 pixel images. Baig [11] developed a temporal spiking network model based on "integrate and fire" neurons and applied them to identify online cursive handwritten characters. Other applications of spiking neural networks include instructing robots in navigation and grasping tasks [12], recognizing temporal sequences [13], and the robotic modeling of mouse whiskers [14].

At present, several groups are developing biological-scale implementations of spiking neural networks for studying the neuronal dynamics seen in the brain. Nageswaran *et al.* [15] implemented a large-scale spiking neural network using the Izhikevich model in a GPU. They were able to simulate 100K neurons with 50 million synapses achieving a speedup of 27x. The Swiss institution EPFL and IBM are developing a highly biologically accurate brain simulation at the sub-neuron level [16]. They have utilized the Hodgkin Huxley and the Wilfred Rall [17] models to simulate up to 100,000 neurons on an IBM BlueGene/L supercomputer. At the IBM Almaden Research Center, Ananthanarayanan and Modha [18] utilized the Izhikevich spiking neuron models to simulate 10^9 neurons and 10^{13} synapses (equivalent to a cat-scale cortical model) on a 147,456 processor IBM BlueGene/L supercomputer.

III. ARCHITECTURE EXAMINED

Modern processors no longer rely on increasing clock frequencies to improve performance. A widespread trend of exploiting parallelism through multiple cores, and in some cases vector parallelism, has developed as the standard for performance improvement.

A. *Multi-core Architecture*

The Cell Broadband Engine developed by IBM, Sony, and Toshiba [19] is a multi-core processor that heavily exploits vector parallelism. The current generation of the IBM Cell BE processor operates at 3.2 GHz and consists of nine processing cores: a PowerPC based Power Processor Element (PPE) and eight independent Synergistic Processing Elements (SPE). All processors and internal RAM are connected through an Element Interconnect Bus (EIB). The PPE is primarily used for administrative functions while the SPEs provide high performance through vector operations. In the PS3 version of the Cell BE, only six of the eight SPUs are available for computation.

The Intel Xeon 5345 processor-based Dell PowerEdge 1950 examined in this paper contains two 2.33 GHz quad-core Intel Xeon 5345 processors. These processors contain

32 KB level-one cache per core and four cores share an 8 MB level-two cache. The processor can execute vector instructions using the SSE3 instruction set.

The AMD Opteron 2356 processor-based SunFire 2200 examined in this paper contains two 2.33 GHz quad-core AMD Opterons. These processors contain 64 KB level-one cache per core, 512 KB level- two data and instruction cache per core, and a 2MB level-three cache shared by four processing cores. The processor can execute vector instructions using the SSE3 instruction set.

Several software level optimizations useful in application development for these multi-core architectures are discussed in Section IV.

B. *GPU Architecture*

The GPU architecture appears as an array of streaming multiprocessors each containing scalar processors, special function units, double precision units, and shared memory to enable thread cooperation. For our experiments, we have used NVIDIA's Tesla C870 card with compute capability 1.0, which has 16 multiprocessors and 128 cores, 1.5 GB global memory, 64 KB constant memory, 16 KB shared memory and operates at a clock rate of 1.35 GHz. The system provides over 1500 MB/sec host-device transfers for pageable memory and over 60,000 MB/sec device-to-device bandwidth.

IV. OPTIMIZATION TECHNIQUES USED

A. *STI PS3*

The following optimization techniques were explored for the PS3 architecture: a) Multi-Threading (MT), b) SIMD Computation (SC), c) Double Buffering (DB), d) Reducing Conditional Statement (RCS), e) Loop Unrolling (LU), and f) Software Pipelining (SP). In the multi-threaded technique, 6 threads are created for execution on each of the 6 SPUs. In the SIMD optimization technique, four similar floating point operations are carried out using 128-bit registers in one cycle. In the PS3, data transfers are accomplished through a DMA engine where two sets of variables are defined for computation. Double buffering is used to overlap communication with computation allowing one DMA request to transfer data from DRAM to one set of variables, while the SPU is computing on the other set of variables. Since the SPUs of the PS3 do not have branch prediction units, the technique of reducing conditional statements is useful to improve performance. Further performance gains can be achieved if conditional statements are reduced through loop unrolling. We have experimented with different amounts of unrolling in our case studies and found that by unrolling up to 8 loops we achieve a measurable performance gain. Finally, software pipelining is implemented by prefetching the variables that will be used for computation. Software pipelining can achieve good performance if it is combined with loop unrolling since the same variables are used for computation instead of fetching each array element every time it is used.

B. *X86 (Intel Xeon and AMD Opteron)*

On the Intel and AMD architectures, POSIX threading (pthreading) was used to create threads and distribute the computations among threads. Both X86 architectures support the SIMD computation called Streaming SIMD Extension 3 (SSE3). It is essentially the same technique as the SIMD computation used with PS3. Similar software pipelining techniques used in the PS3 optimizations can also be applied in the X86 implementations. Software pipelining shows a significant speedup for both SNN models investigated as will be seen in the results section.

C. *GPU*

NVIDIA's Compute Unified Device Architecture (CUDA) allows the developer to specify a collection of threads, called a *thread block*. These blocks are further divided into groups of 32 threads called *warps;* additionally, these warps are divided into groups of 16 threads called *half warps*. Several concurrent thread blocks can reside on the multiprocessor; the number depends upon the multiprocessor resources such as shared memory available and registers. A kernel launched as a group of thread blocks called a *grid*, is written in CUDA for C and describes the functionality for all the threads created. CUDA offers several optimization techniques that can be employed for optimal performance. These techniques are Memory Level Optimizations, Execution Configuration Optimizations and Instruction Optimizations.

The *Memory Level Optimization* strategies used in our work are *Coalesced Global Memory Acce*sses *(G), Shared Memory* (*S*), and *Texture Memory* (*T)* for the input image. A combination of these optimization techniques were also explored such as *coalesced global memory accesses with texture cache lookup (GT)*, and finally the use of *shared memory with texture cache lookup (ST)*. Global memory offers maximum bandwidth when accesses by threads in a half warp are *coalesced* and hence the hardware can fetch or store the data in one or two transactions. Uncoalesced accesses result in separate transactions for each thread and poor performance. Uncoalesced accesses can be identified using the *CUDA profiler* and can be reduced through kernel modification. Using shared memeory can also reduce uncoalesced accesses. Since shared memory is cached, it is also useful for those applications that involve multiple accesses to data. *Bank conflicts*, which occur when threads in a half warp access the same bank, should be avoided while using shared memory since conflicting accesses are serialized and will lead to poor performance. Finally, *texture memory* is cached and has substantially larger bandwidth than global memory. We have used texture memory bound with global memory for storing input image as an optimization strategy.

Execution Configuration Optimization is an effective way to hide latency on memory bound kernels. To specifying an execution configuration, the developer sets the appropriate number of threads in a block so that the multiprocessor occupancy (Occupancy = # of concurrent warps/maximum # of concurrent warps that can execute on multiprocessor), is maximized. The CUDA occupancy calculator is a useful tool to determine multiprocessor occupancy and the optimal number of threads per block.

The *Instruction Level Optimization* technique examined in our work is the use of fast math functions that are both accurate and involve fewer clock cycles. CUDA also offers the compiler optimization *–use_fast_math* to force compiling arithmetic functions as fast math functions.

V. EXPERIMENTAL SET UP

Specifications for the three multi-core architectures were described in Section III. The PS3s used in this study are running Fedora Core 9 with the gcc compiler version 4.3.0 for PPU and spu_gcc compiler version 4.1.1 for SPU and O3 optimization. The Intel Xeon is running Red Hat 4.1.2 with the icc compiler version 10.1 with O3 and msse3 optimizations. The AMD Opteron is also running Red Hat 4.1.2 but uses the gcc compiler version 4.1.2 with the compiler optimizations O4 and msse3. The GPU platform consists of an Intel Core 2 Quad 2.66 GHz as the host and a Tesla C870 as the GPU accelerator. The SNN networks were developed using CUDA 2.1 installed on Fedora 8, 32-bit.

For each SNN, the number of neurons at each level for different input image sizes is shown in Table I. Two networks (576 neurons in level 1 and 48 neurons in level 2) with two different neuron models were initially developed, tested and trained in MATLAB before being converted to C. The networks are then scaled linearly as seen in Table I to accommodate larger image sizes.

VI. PARALLELIZATION and MAPPING OF MODELS

A. Using Multi-core Architectures

Since the two spiking neuron models (HH and Izhikevich) were used in the same image recognition network structure (see Fig. 1), the parallelization approach for both implementations was the same. The network parallelization and vectorization techniques were used for the all combinations of the network sizes and models as described below.

Network parallelization: All neurons at any particular level of the SNN run in parallel and are independent of each other, allowing the neurons at a given level to be divided evenly across all available processing cores. Each processing core is assigned a set of level 1 neurons and generates a level 1 firing vector after evaluating all neurons in this set.

Evaluation of the level 2 neurons differs between the PS3 and the X86 architectures (Intel Xeon and AMD Opteron). In the X86 architectures, the firing vectors store the index of each level 1 neuron that fired and was evaluated by that processing core. The number of firing vectors is the same as the number of processing cores utilized. Each processing core reads all firing vectors when evaluating the level 2 neurons. This approach enables the level 2 neuron weight computation in training to be simplified by examining only the level 1 neurons that fired.

TABLE I. SPIKING NETWORK CONFIGURATIONS FOR IMAGES

Input image size	**Level 1 neurons**	**Level 2 neurons**	**Total neurons**
96×96	9216	48	9264
192×192	36864	48	36912
240×240	57600	48	57648
384×384	147456	48	147504
480×480	230400	48	230448
720×720	518400	48	518448
960×960	921600	48	921648
1200×1200	1440000	48	1440048
1680×1680	2822400	48	2822448
2160×216	4665600	48	4665648
2400×2400	5760000	48	5760048

In the PS3 architecture, after generating its firing vector, each SPU calculates the current input for level 2 using its firing vector. Thus there will be six vectors, one per SPU, of input current for level 2 where each vector has 48 elements corresponding to the 48 output neurons. These current vectors are then sent to the PPU where they are accumulated to form the total current for each level 2 neuron. Finally, the PPU evaluates the level 2 neurons.

Vectorization: Neurons in a level are independent of each other. Thus, simply evaluating four neuron-level calculations in parallel on each processing core vectorizes the network.

B. GPU

The GPU implementation involves acceleration of the level 1 neurons on the GPU, as it is the most compute intensive of the two levels, and computing the level 2 neurons of the network on the host. The GPU performs the neuron dynamics computation for level 1 and supplies the firing data to the host. The host uses this firing information to compute the neuron dynamics for level 2. The single neuron dynamics in level 1 is computed by a thread on the GPU; the number of threads generated for the GPU and their functionality is described by writing kernels. The optimization techniques described in Section IV are applied and evaluated for performance. We have avoided data transfers between the host and GPU as much as possible. The only frequent communication involved was the transfer of the firing information of level 1 from the GPU to host for each time step.

VII. RESULTS

A. Speedup performance

Performance of the four architectures for the Izhikevich neuron model is shown in Fig. 2. For the GPU implementation, the speedup initially increases with an increase in the number of neurons. The architecture observes only a nominal increase in the speedup beyond 1 million neurons. A maximum speedup of 9.5x was observed for 5.8 million neurons. Initially, the speedup increases due to the increasing image/network size. Referring to the Table II, the flop/byte ratio for the model is only 0.65, and therefore the increasing speedup cannot be sustained as the nominal amount of computations on the GPU is unable to amortize the increased communication time due to larger network sizes. GPUs require a higher flops/byte ratio in order to fully exploit the task and thread level parallelism.

The IBM PS3 demonstrates better performance than the GPU for this model, showing a speedup of 17.3x for 5.8 million neurons. The speedup achieved with the PS3 has a regular performance curve, which increases initially and then saturates at around 17x. The initial increase in speedup can be attributed to the dominance of parallel computation over the fixed overhead cost (such as thread creation and barrier synchronization).

The Xeon speedup increases to about 58x for 1 million neurons and then begins to decrease until it stabilizes at 20x. The speedup diminishes after 1 million neurons because the increasing number of variables required to update the neurons cannot be accommodated in the cache, resulting cache misses. Thus, the communication time will grow faster and dominate the computation time, reducing the speedup.

The speedup performance curve for the AMD Opteron has a shape similar to that of the Intel Xeon with the exception that its peak speedup was only 36x and begins to fall off sooner.

Performance of the four architectures for the HH model appears very different as shown in Fig. 3. Here the GPU clearly outperforms the other architectures. A speedup of 118.8x was observed for 5.8 million neurons. Substantial speedup was expected and seen for the GPU implementation as the model involves a significantly higher flop/byte ratio (6.02 versus 0.65 for the Izhikevich) as shown in Table II.

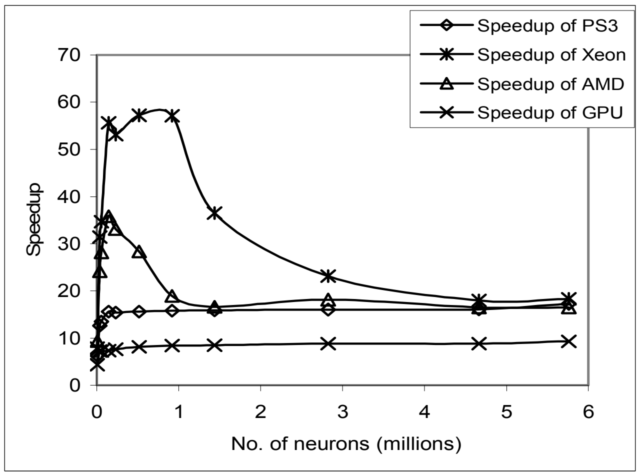

Figure 2. Speed-up performance of four architectures over Intel Core 2 Quad for the Izhikevich model

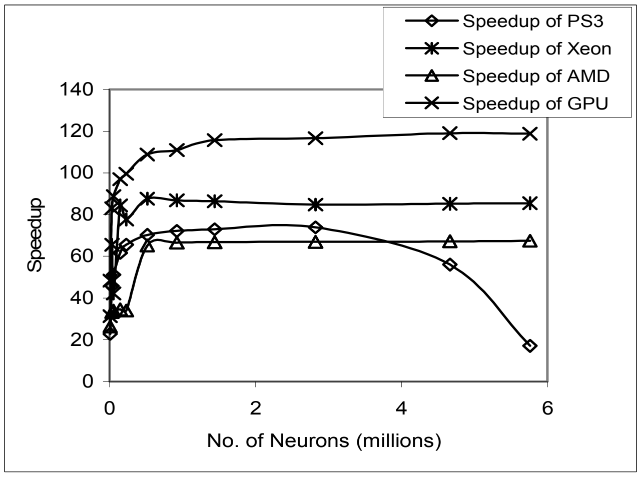

Figure 3. Speed-up performance of four architectures over Intel Core 2 Quad for the HH model

TABLE II. FLOPS AND DATA TRANSFER FOR UPDATING EACH NEURON DYNAMICS

Models	Flops	Bytes transferred	Flops/byte ratio
Izhikevich	13	20	0.65
HH	246	44	6.02

As seen in Fig. 3, the PS3 speedup for the HH model has an initial increase up to 0.5 million neurons and begins to saturate around 2.8 million neurons before finally decreasing. This behavior can be explained through analysis of the cache and main memory sizes. The combined size of the PPU Level 1 and 2 cache is 544 KB and the main memory capacity is 512 MB. The smallest simulated network has 9264 neurons and requires 1.9 MB of data, which is larger than the available caches resulting in cache misses but still providing speedup due to the flops/byte ratio of the model. Scaling the network to 0.5 million neurons increases the data size to 106 MB and we still observe a speedup since the data continues to fit in main memory. Once the data size increases beyond the capacity of main memory, which occurs for a network size of 2.8 million neurons (581 MB), the performance begins to decline. Network sizes of 2.8 million neurons and larger will therefore cause page table misses. This abrupt increase in data communication overhead causes a decrease in speedup as seen in Fig. 3 for networks of 2.8 million neurons and larger.

Finally, the speedup for the Intel Xeon and the AMD Opteron initially grow but saturate at 88x and 68x respectively at 0.5 million neurons for reasons similar to those described above for the PS3.

B. *Effect of Optimization Techniques on Speedup*

B.1. Cell BE (PS3)

In Fig. 4, performance of the PS3 resulting from each of the available optimization techniques for both neuron models is shown. From the figure, we observe that for the Izhikevich model (for both the image sizes of 2400×2400 and

1680×1680), multithreading and SIMD computation techniques contribute the most to the speedup. For the HH model and an image size of 1680×1680, the speedup increases with each additional optimization technique. But for an image size of 2400×2400, the loop unrolling and software pipelining techniques reduce the speedup, because for these optimizations, more instructions must be accommodated in the local store of the SPU resulting in more variables to transfer (i.e. more communication and more overhead).

B.2. X86 (Intel Xeon, AMD Opteron)

For the Intel Xeon implementation of the Izhikevich model with the largest image size of 2400×2400, the SSE and SP optimizations do not contribute significantly to the speedup because of the low flop/byte ratio of the model (see Fig. 5). For the smaller image/network (960×960), the speedup grows almost linearly as the optimizations are added one by one. For the HH model, each optimization contributes significantly to the speedup for both image sizes.

For the AMD Opteron, the same optimization techniques used for the Intel Xeon were applied with similar findings as shown in Fig. 6.

B.3. GPU

In the GPU implementations, experiments were conducted for different memory optimizations with execution configurations assigned as a multiple of the warp size, 32. For each of the two models, CUDA profiler did not report any uncoalesced accesses and hence the coalesced global memory access strategy was successful. No bank conflicts were reported either which ensured the optimal use of shared memory strategy. As seen in Table III, the Izhikevich model performed the best with shared memory optimization and an execution configuration of 128 threads per block. We attribute this behavior to the performance benefits offered by the fast shared memory.

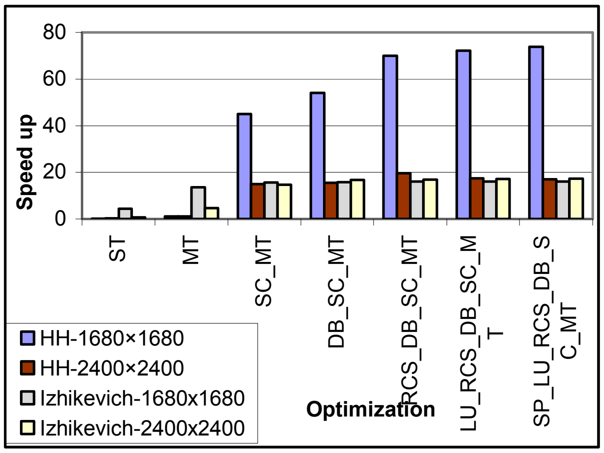

Figure 4. PS3 speedup (6 SPUs) with different optimization techniques applied (ST: Single Thread, MT:Multithread; SC: SIMD computation, DB: Double Buffering, RCS: Reduced Conditional Statement, LU: Loop Unrolling, SP: Software Pipelining)

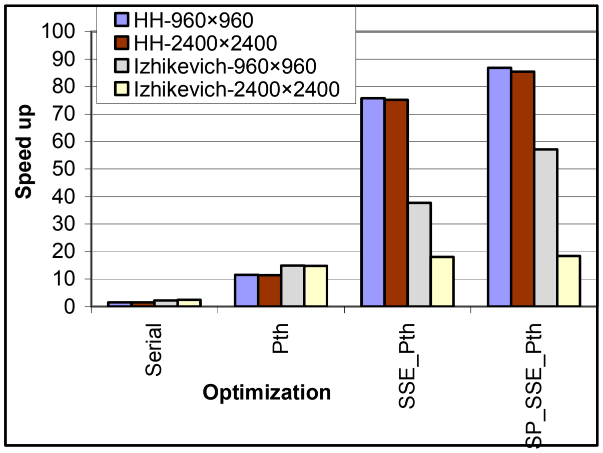

Figure 5. Intel Xeon speedup (8 cores) with different optimization techniques applied for the Izhikevich and HH model (pth: POSIX thread, SSE: Streaming SIMD Extension 3, SP: Software Pipelining)

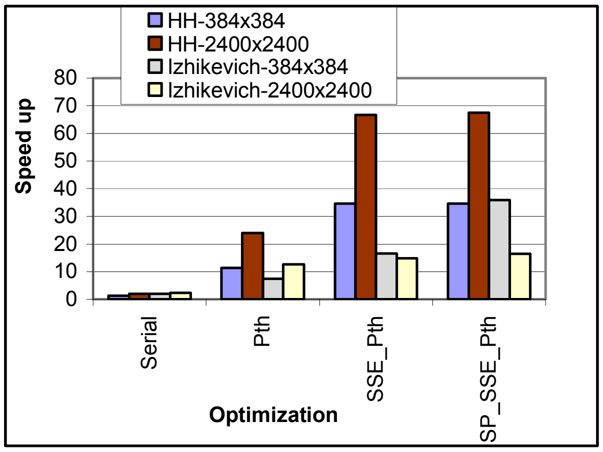

Figure 6. AMD Opteron speedup (8 cores) with different optimization techniques applied for the HH and Izhikevich model

TABLE III. GPU: EFFECT OF MEMORY OPTIMIZATION TECHNIQUES FOR IZHIKEVICH AND HH MODEL FOR IMAGE SIZE 2400x2400*

Memory Optimi-zation	Execution Configuration		Multiprocessor Occupancy		Speedup	
	Izh.	HH	Izh	HH	Izh.	HH
G	256	192	1	0.75	9.33	118.4
GT	256	192	1	0.75	9.27	118.8
S	128	192	0.75	0.50	9.45	116.43
ST	128	192	0.75	0.50	9.29	116.32

*Techniques: G:Global access. GT:Global access with texture cache lookup. S: Shared Memory. ST: Shared Memory with texture cache lookup.

For the Hodgkin Huxley model, as seen in Table IV, coalesced global memory accesses with texture cache lookup (GT) and an execution configuration of 192 performed the best. For this model, G and GT have the highest multiprocessor occupancy, 0.75 compared to 0.5 of S and ST. We attribute the differences in the speedup performance to the difference in the level of multiprocessor occupancy. Also, the fast math functions provided a boost in the performance as shown in Table IV for the HH model with *GT* technique.

TABLE IV. GPU: EFFECT OF USING FAST MATH FOR THE HODGKIN HUXLEY MODEL

Image Size	Speedup without fast math	Speedup with fast math	Improvement (%)
2400	109.62	118.8	8.3
2160	109.69	119.08	8.5
1680	107.63	116.56	8.29
1200	106.2	115.62	8.87
960	103.54	110.98	7.18

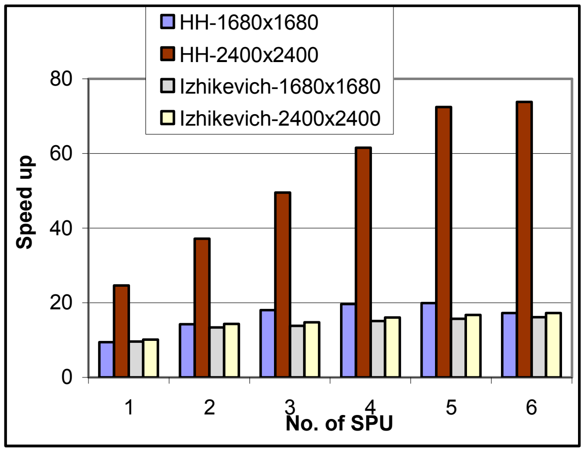

Figure 7. PS3 speedup for increasing number of SPUs for the Izhikevich and HH model

C. *Effect of changing accelerator configuration*

We have investigated the effect of additional threads on speedup. Though it is generally expected that with the addition of cores to the implementation, the speedup will proportionately increase, as seen below it is not always true.

C.1. PS3

Fig. 7 shows the speedup variation in the PS3 as the number of SPUs is varied for both neuron models. We observed from the figure that for the Izhikevich model of network sizes 2400x2400 and 1680x1680, moving from 1 SPU to 2 SPUs, the speedup increases significantly; beyond 2 SPUs, the speedup grows marginally. This behavior occurs because the model requires more communication time than computation time, which cannot be parallelized by adding SPUs. For the Hodgkin-Huxley model with an image size of 1680×1680, it is observed that the addition of SPUs causes the speedup to grow almost proportionately. For the largest image size, 2400×2400, the communication time increases more than computation time and the resulting speedup is slightly lower than that of the smaller image sizes, as discussed in the speedup performance sub-section. It is also notable that the speedup with 6 SPUs is slightly lower than that of 4 or 5 SPUs. This change most likely occurs due to the congestion in the EIB of PS3. The EIB has only four lanes to transfer data between the SPU and DRAM. So when 6 SPUs issue a DMA request for data from the DRAM, the request processing time is increased due to the larger amount of requested data from the DRAM. Thus, repeated DMA requests will be issued, further increasing the communication time and eventually negatively impacting the speedup.

C.2. X86 (Intel Xeon and AMD Opteron)

In Fig. 8, the speedup resulting from the addition of processing cores for both the models is presented. From the figure it is found that for the Izhikevich model with an image size of 960×960, the speedup grows almost proportionally with the addition of processing cores as expected. But for an image size of 2400×2400, the speedup is significantly less and after 2 processing cores, there is almost no gain from additional cores. The primary reason for this behavior is the higher communication time compared to computation time, so the addition of processing cores will not reduce the run time. On the other hand, the Hodgkin-Huxley model is a compute intensive model, and thus, for both the image sizes of 960×960 and 2400×2400, the speedup grows proportionally with the addition of processing cores.

For the AMD Opteron, we have reported results for both models for an image size of 384×384, where the best speedup occurs, and for the largest image size, 2400×2400, in Fig. 9. Except for the smaller image size, we find a similar trend to that of the Intel Xeon and the same explanation can also be applied. For the 384×384 image size with the HH model, the speedup decreases beyond 4 cores due to the additional communication overhead.

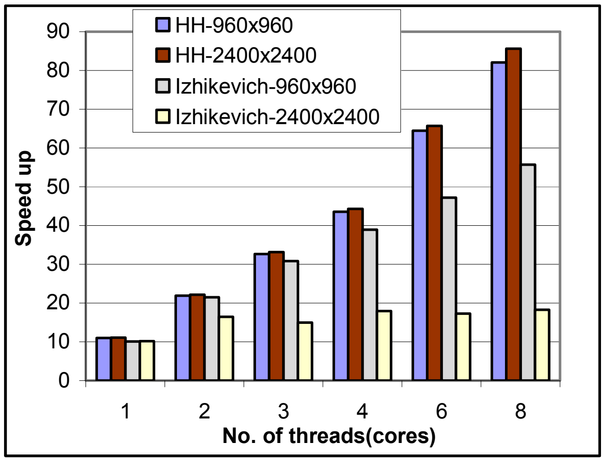

Figure 8. Intel Xeon speedup for increasing number of cores for the Izhikevich and HH model

C.3. GPU

As discussed in Section IV, changing the number of threads per block can alter the execution configuration of the GPU. This modification results in a different multiprocessor occupancy. A higher occupancy may not directly imply better performance, but it can assist in hiding latency. Table V compares the performance of different block configurations with the best optimization strategies for each of the models: shared memory for the Izhikevich model and GT optimization for the HH model. From this table we can infer that block configurations with similar multiprocessor occupancy give similar performance. For the HH model, higher multiprocessor occupancy has shown to be effective in hiding the communication latency and boosting the overall performance. As shown in Table V, a block size of 192 with multiprocessor occupancy of 75% has given the maximum speedup of 118.8x.

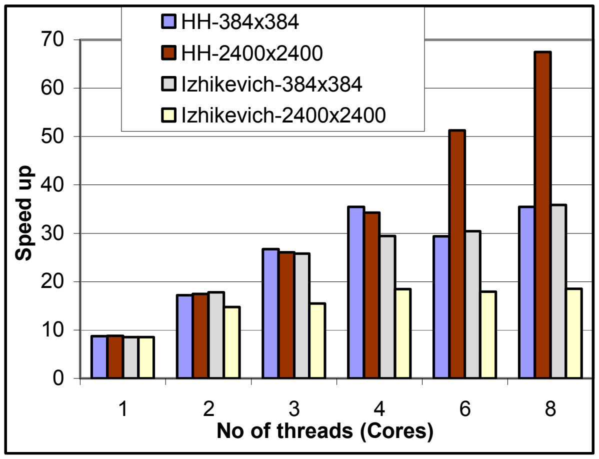

Figure 9. AMD Opteron speedup for increasing number of cores for the Izhikevich and HH model

TABLE V. GPU: SPEEDUP WITH VARYING THREADS PER BLOCK FOR IZHIKEVICH AND HH MODELS WITH IMAGE SIZE 2400×2400

Threads per block		Multiprocessor Occupancy		Speedup	
Izh	HH	Izh	HH	Izh	HH
128	192	0.75	0.75	9.45	118.8
192	224	0.75	0.58	9.43	115.7
256	256	0.75	0.67	9.38	116.2

VIII. CONCLUSIONS

We have implemented two of the most biologically accurate spiking neural models on an NVIDIA GPU, Cell BE, Intel Xeon and AMD Opteron. The best speedup of the Izhikevich model is obtained from the Intel Xeon with all optimizations while the best speedup of HH model is obtained from the NVIDIA GPU with *coalesced global memory access along with texture lookup* for the input image as optimization. Different optimization techniques were explored and analyzed for each combination of the architectures with input image sizes. We have presented the speedup performance dependence on architectures, optimization techniques, data transfer size, and flops/byte. For the Xeon, Opteron and PS3, the best speedups were obtained when all of the optimization techniques were applied. For the GPU implementation this was not the case, as we found the best speedups were obtained when either a single or a combination of two memory optimization techniques were applied. The *CUDA occupancy calculator* and *CUDA profiler* were found to be useful tools in selecting the best configuration and analyzing the results. Our implementation of the models on a variety of architectures shows that architecture and algorithm complexity are closely related and a proper match between the two is necessary to achieve the best performance. Future work will involve exploration of optimization strategies for GPUs with higher compute capability, exploring the simulation of the mammalian scale spiking neuron networks, and a comparison of performance for clusters of these architectures.

REFERENCES

[1] R. Ananthanarayanan and D. Modha, “Anatomy of a Cortical Simulator,” *Proc. Int’l Conference for High Performance Computing, Networking, Storage and Analysis* (SC ‘07), pp. 1-12, Nov. 2007.
[2] E.Izhikevich, “Which Model to Use for Cortical Spiking Neurons?” *IEEE Trans. on Neural Networks*, vol. 15, no. 5, pp. 1063-1070, 2004.
[3] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and application to conduction and excitation in nerve,” *Journal of Physiology*, vol. 117, pp. 500–544, 1952.
[4] E. M. Izhikevich, “Simple Model of Spiking Neurons,” *IEEE Trans. on Neural Networks*, vol. 14, no. 6, pp. 1569-1572, Nov. 2003.
[5] K. L. Rice, M. A. Bhuiyan, T. M. Taha, C. N. Vutsinas, M. C. Smith, “FPGA Implementation of Izhikevich Spiking Neural Networks for Character Recognition”, in *Proc. Reconfig‘09*, Mexico, pp. 451 – 456, Dec. 2009.
[6] E. M. Izhikevich, “Dynamical Systems in Neuroscience,” *MIT press*, Cambridge, Massachusetts, 2007.
[7] A. Gupta, L. Long, “Character Recognition using Spiking Neural Networks,” in *Proc. IJC*NN, pp. 53 – 58, Aug. 2007.
[8] M. A. Bhuiyan, T. M. Taha, and R. Jalasutram, “Character recognition with two spiking neural network models on multi-core architectures,” in *IEEE Proc. CIMSVP*, TN, pp. 29 – 34, Mar. 2009.
[9] A. Delorme and S. J. Thorpe, “SpikeNET: an event-driven simulation package for modeling large networks of spiking neurons,” *Network-computation in neural systems*, vol. 14, no. 4, pp. 613–627, Nov. 2003.
[10] C. Johansson and A. Lansner, “Towards Cortex Sized Artificial Neural Systems,” *Neural Networks*, vol. 20, no. 1, pp. 48–61, Jan. 2007.
[11] A. R. Baig, “Spatial-temporal artificial neurons applied to online cursive handwritten character recognition,” *in Proc. European Symposium on Artificial Neural Networks*, pp. 561–566, Apr. 2004.
[12] C. Panchev and S. Wermter “Temporal sequence detection with spiking neurons: towards recognizing robot language instructions,” *Connect. Sci.*, vol. 18, no. 1, pp. 1-22, 2006.
[13] T. Ichishita, R. Fujii, “Performance evaluation of a temporal sequence learning spiking neural network,” in *Proc. 7th IEEE International Conference on Computer and Information Technology*, pp, 616-620, Oct. 2007.
[14] K-Team, Inc. Online Available: http://www.k-team.com/
[15] J. M. Nageswaran, et al., “A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors,” *Special issue of Neural Network, Elsevier*, vol. 22, no. 5-6, pp. 791-800, July 2009.
[16] H. Markram, “The Blue Brain Project,” *Nature Reviews Neuroscience*, vol. 7, pp. 153–160, 2006.
[17] W. Rall, “Branching dendritic trees and motoneuron membrane resistivity,” *Experimental Neurology*, vol. 1, pp. 503–532, 1959.
[18] R. Ananthanarayanan, S. K. Esser, H. D. Simon, and D. S. Modha, “The Cat is Out of the Bag: Cortical Simulations with 10^9 Neurons, 10^{13} Synapses,” in *Proc. of SC ‘09*, Oregon, Nov. 2009.
[19] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, T. Yamazaki, “Synergistic Processing in Cell’s Multicore Architecture,” *IEEE Micro*, vol. 26 no. 2, pp. 10–24, Mar. 2006.
[20] R. Fox, “Stochastic versions of the Hodgkin-Huxley equations,” *Biophysical Journal*, vol. 72, no. 5, pp. 2068-2074, May 1997.