

Powering Smart Home intelligence using existing entertainment systems

Markus Scholz¹, Gesine Flehmig², Hedda R. Schmidtke¹, and Gerhard H. Scholz²

¹ Karlsruhe Institute of Technology, TecO, Karlsruhe, Germany

² Leipziger Institut für Präventivmedizin GmbH, Leipzig, Germany

Abstract—Smart Homes and Smart Environments are classically centrally organized and utilize methods from machine learning and artificial intelligence. In recent years, continuous progress has been made and the technology has reached a level where the deployment becomes feasible on a larger scale and in everyday settings, raising the question where the central system should be deployed. In this paper, we propose the use of existing entertainment systems as the Smart Home controller. In a case study we examine the performance of the multi-core Cell processor of the Sony PlayStation 3 for training artificial feed forward neural networks using specifically adapted parallel training strategies. The evaluation of these strategies shows a gain in speed of up to 6.6 over a sequential implementation on a single processing element of the Cell. Based on these findings and related work we suggest that home entertainment systems should be considered as possible powerful, deployment platforms for future Smart Home systems.

I. INTRODUCTION

Smart homes aim to create an environment for their inhabitants that improves comfort and productivity. The state of the home is thereby recognized and changed using sensors and actuators which are controlled by the home gateway. The traditional Smart Home is implemented using a centralized architecture in which this home gateway/controller is the central decision point. Hence, the system on which control, detection and intelligent algorithms reside. The progress of research in the Smart Home domain has advanced this technology to a point where it can leave the test beds and be deployed in real home environments.

In this context, the question arises where such a central Smart Home system could be located and how this system should be designed in order to cope with more complex algorithms, e.g. for activity recognition of residents[16], [8]. While cloud computing based solutions would offer enough processing resources to facilitate even the most complex algorithms this would probably be controversial with respect to user privacy. On the other hand, the employment of smartphones could work around this issue but lead to problems regarding computing speed and availability. Thus, a system that is permanently available and offers enough computing power would be preferable.

We believe that home entertainment systems can provide such a platform. These systems are widespread, always activated or in standby. They provide a stable development base and are usually interconnected with the local network in the user's home. These systems are further outfitted with general purpose interfaces using which Smart Home sensors

and actuators could be accessed. As these systems are designed to deliver a high quality immersive gaming experience they are also typically outfitted with powerful processing units. While the possibility of developing applications for these systems was restricted in previous system generations, for all of the three major systems (Nintendo Wii, Microsoft Xbox 360 and Sony PlayStation 3) there currently exist solutions to develop and deploy custom applications.

In this paper, we give a brief overview over these major home entertainments systems. We further report from a case study for the Cell processor which is embedded into the PlayStation 3 and which potentially offers the highest computing performance. In this case study we demonstrate how artificial neural network training speed can be improved by developing a specifically adapted version for this CPU. We continue with a discussion of the achieved performance of the presented neural network simulator, the related work and the expected performance and eligibility of the other two consoles. We conclude the paper with the suggestion that the inclusion of entertainment systems in future Smart Home architectures could be a worthy approach.

II. HOME ENTERTAINMENT SYSTEMS

The development of home entertainment systems and especially game consoles has reached its seventh generation today. Compared to the first generations which were introduced between 1970 and 1980 there has been a consolidation in the number of available systems (from around 10 to three) and a strong growth in distribution (from ca. 10 to 180 million sold units).

The three major systems which are dominating the markets today are the Nintendo Wii, the Microsoft Xbox 360 and the Sony PlayStation 3 (PS3). Apart from the Wii which gained its high sales numbers mostly due to a novel user interface concept, the two other systems both offered a better price to computing power ratio than PC systems when they were released[18]. For instance, the theoretical peak performance of the PlayStation 3's Cell CPU of 204.8 giga floating operations per second (GFlops) is still nearly the double of the peak performance of a recent Intel Core i7-970 (94.0GFlops). This is also supported by studies in which the recent gaming systems were employed in order to build a cheap high performance cluster[7] or to improve calculation speed of complex computational tasks[18]. Further developments that could make these systems interesting for Smart Home environments are

advances regarding network connectivity, availability of developer tools and low level system access, as well as accessibility of interfaces like USB to connect arbitrary hardware. In the following, a brief overview of the Wii and the Xbox 360 is given in respect to those attributes. For the PlayStation 3 more details are provided in order to understand the performed case study. A summary on all three systems is given in table I.

Table I: Relevant attributes of the three dominating entertainment systems

	Wii	Xbox 360	PS3
sold units as of Dec.2010 (mil)	85	50	47.5
theoretical peak performance (GFlops)	2.9	115.2	204.8
network connectivity built-in	yes	yes	yes
official development support	no	yes	yes
unofficial development	yes	yes	yes

A. Nintendo Wii

The Nintendo Wii, introduced in 2006, seems significantly underpowered compared to its competitors. However, a year after its release it became the market leader of its console generation. Until today around 85 million units have been sold. The Wii has 24MB of main RAM, 512MB flash memory, an IBM PowerPC based CPU running at 730MHz, an ATI graphics processing unit (GPU) running at 243MHz with 3MB internal and 64MB shared memory and an integrated 802.11b/g wireless LAN device. The Wii has an additional ARM11 SoC which is also running at 243MHz and handles I/O, security and boot up of the system. Further integrated hardware components are a Bluetooth transceiver, a SD card memory bay, two USB2.0 connectors, and a DVD ROM. As official resources about the hardware specifications and performance are sparse, the theoretical peak performance of the Wii can be only roughly estimated to around 2.9GFlops¹.

Developing and deploying applications for the Wii requires the installation of an unofficial application called home brew channel for which a development kit is available². Applications developed with the WiiBrew development kit can access all devices, periphery, network and attached input devices (Wimotes, Balance Board, etc.). Thus, also the communication with USB based devices, e.g. IEEE 802.15.4, to read out sensors and actuators from the Smart Home environment would be possible.

B. Microsoft Xbox 360

The Xbox 360 by Microsoft was introduced in 2005 and was sold 50 million times until December 2010. The Xbox 360 is based on a specially designed 3.2 GHz PowerPC Tri-Core Xenon processor which is tightly coupled to a 500MHz ATI Xenos GPU[6]. The theoretical peak performance has been estimated to 115.2GFlops[18]. Depending on the model the system features up to five USB slot, an Ethernet jack, and

a DVD drive. An additional wireless LAN USB device is available. Development for Xbox 360 is officially supported by Microsoft through the XNA framework. The application deployment can be realized using the Xbox LIVE Marketplace. Previously, this was only accessible by game development companies, but it was changed with the “XNA Creators Club” in which even hobbyists may upload and distribute their games to the Xbox community, if they agree to a \$99 per year charge (a free academic license is available)³. Alternatively, there exists also a home brew community for the Xbox which provides unofficial software to allow arbitrary development for the platform.

C. PlayStation 3

The PlayStation 3 was released in 2006. Since then it has been sold 47.5 million times worldwide. It features a modified version of the IBM Cell processor which runs at 3.2 GHz and a Nvidia GPU running at 550MHz with 256MB GDDR3-VRAM. The system RAM is 256MB. The system is further outfitted with a Blu-ray Disc player, up to 4 USB2.0 slots, a Bluetooth 2.0 transceiver, gigabit Ethernet and Wi-Fi networking. Some versions also have integrated flash card readers. The system performance of the Cell processor in the PS3 provides a theoretical peak performance of 204.8GFlops making this system the most powerful of all three consoles. However, the special architecture of the processor requires programming paradigms which are quite different from general purpose processor development.

With the first release of the system, Sony has encouraged the optional use of an “Other OS” i.e. the use of a different operating system in parallel to the standard entertainment system firmware. As the Linux kernel supports the Cell, Linux is typically run as “Other OS”. While in the Linux system all interfaces can be accessed normally, direct access to the GPU is denied using a hardware security mechanism. This restricts the computing potential of the “Other OS” as the GPU could complement the Cell’s processing power. Nevertheless, the “Other OS” functionality has enabled users to create powerful parallel applications for the PS3, e.g. a cheap supercomputer cluster build from PS3s[7]. The development for the PS3 is simplified through a special Cell SDK which was created by IBM and which provides a tool chain and libraries[7], [20].

Despite these developments, Sony has recently removed the “Other OS” functionality and there are currently multiple law suits filed related to the removal. Hence, to develop software for the PS3 it is currently necessary to use the unofficial home brew software or to use an old firmware. The home brew software is also based on Linux and provides full access to eight of the originally nine cores of the Cell processor and to the graphics driver hardware.

Due to the strong potential of the PS3 for Smart Home environments, especially in respect to processing power, we have conducted a case study for accelerating training of artificial neural networks (ANNs) on the Cell processor. Neural network training or similar algorithms are regularly encountered in Smart Home architectures[1], [14]. Additionally, training

¹<http://jonon.gs/blog/computers/gflop-comparison-table-of-cpus-and-gpus/>, last accessed on April 10th, 2011

²<http://wiibrew.org>, last accessed on: April 10th, 2011

³<http://create.msdn.com/>

neural networks may become very expensive in terms of computation time, for instance when using large networks, datasets or when employing a heuristic wrapper based feature selection. To provide a basis for understanding the implementation we will now briefly introduce the Cell’s architecture.

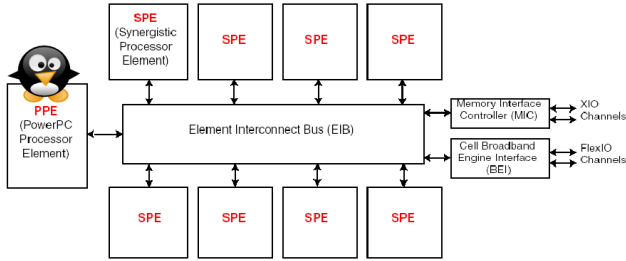


Figure 1: Scheme of the Cell processor, modified from [20].

The Cell’s architecture is shown in fig.1. It is composed of nine processing elements of two different types. The two elements are the PowerPC processing element (PPE) and the synergistic processing element (SPE). While there are 8 SPE cores and 1 PPE core incorporated into the Cell, only six of the eight SPEs can be utilized in the PS3 when using the “Other OS” functionality.

The PPE is a general-purpose PowerPC architecture compatible 64-bit RISC processor. The PPE incorporates a vector/SIMD⁴ extension called AltiVec which implements 32 128-bit vector registers and a mature vector instruction set. The task of the PPE is running the operating system and managing the SPEs.

The SPE is a 32-bit vector/SIMD processor using its own machine language optimized for data-rich operations. It does not have direct access to the main memory, instead it is provided with its own small local memory of 256KB called local store (LS). The SPE can transfer data in-between LS and main memory parallel to the execution of program code (DMA transfer). The maximum amount of data for one DMA transfer is 16 KB.

PPE and the SPEs are connected through a high bandwidth bus (EIB). Access to main memory is provided by the memory interface controller (MIC). It is further possible to connect two Cell processors together using the Cell Broadband Engine Interface (BEI).

III. IMPROVING TRAINING SPEED OF NEURAL NETWORKS

A. Selecting parameters for fast neural network training

The authors of [5] provide a comparison of the most commonly used feed forward connected artificial neural network (ANN) training algorithms on some of the datasets from the UCI Machine Learning Repository. An algorithm which nearly always converges to a low error in minimal time is the Levenberg-Marquardt algorithm (LM)[15]. However, this algorithm has a memory requirement of $T \times N$ where T is the number of patterns and N is the number of weights. Considering the architecture of the Cell processor, on the distributed processing elements there is only limited memory

available. Hence, selecting an algorithm which converges fast but requires only limited memory is preferable. Therefore the resilient back propagation learning algorithm (RP)[10] was selected for implementation. This algorithm only needs fourfold the space of the original back propagation (BP) algorithm. Further optimizations were the implementation of the improved hyperbolic tangent activation function (tanh2) and the Nguyen-Widrow weight initialization. A detailed description of their implementation can be found in [11].

B. Parallelization of neural networks

In [21], four parallelization strategies for ANNs were described: training session parallelism, exemplar parallelism, node parallelism and weight parallelism.

In training session parallelism several neural networks with different configurations (e.g. node count, layer count, learning rate, weight initialization, etc.) are trained independently on different processing elements (PEs). Finally, the best performing network is selected.

In exemplar parallelism the training set is distributed across the processing elements. Then each of these PEs trains an identical neural network on this subset of the training data. After these patterns have been processed the PEs transfer their weight change averages to a central sink which adapts the ANN weights and sends them back to the PEs for the next training iteration.

Node parallelism exploits the parallelism in the structure of the neural network. Hence, neural network nodes of a layer are distributed across the processing elements. As the number of nodes in a network may outnumber the available processing elements several nodes may be allocated to a single processing element. Depending on the capabilities of the processing elements distributing multiple nodes to the processing elements should also be considered in order to reduce the number of messages necessary for synchronization.

In weight parallelism the input signals to the neurons on the next layer are calculated in parallel on the processing elements. Hence, the weighting of the outputs of the previous layer is parallelized. The summation is then realized using an appropriate communication scheme. This approach is probably most useful for an array of a large number of PEs with reduced capabilities and high speed interconnections.

In [9], these four methods were evaluated on a cluster with 32 computers. Based on their reports we selected node and exemplar parallelism for implementation on the Cell. While training session parallelism is an interesting strategy it is no specific study of a parallel algorithm for neural network training. On the other hand, weight parallelism seems much to fine grained and would probably need to many synchronization messages to allow an efficient implementation. While node parallelism also needs a considerable larger amount of messages between processing elements (PE) than exemplar parallelism, it is expected that the efficient ring communication bus of the Cell reduces latency and makes this a worthy parallelization candidate.

As it has been shown that ANNs with one hidden layer can approximate arbitrarily well any functional continuous

⁴Single Instruction Multiple Data

mapping if the number of hidden nodes is large enough [2] we will further only regard parallelization of ANNs with two layers of weights i.e. three layer of nodes.

a) *Node parallelism (NP)*: The basic NP allocates every single neuron to a node of the cluster i.e. every cluster node computes a single activation function. Due to inefficiency of the basic approach (similar to weight parallelism a large number of messages is generated) a number of neurons of the same layer is usually mapped to one processing element along with the associated connection weights of the neuron. Hence, processing elements also multiply their input values by the corresponding connections weights and sum them together. NP with a tree based broadcast method in a cluster computer, as reported in [9], needs the following number of messages for one presentation of all dataset rows to the network and the corresponding weight adaptation (epoch):

$$\#Messages = T * \left((P-1) + \sum_{i=1}^{L-2} (P * \log_2(P) + 2 * (P-1)) \right) \quad (1)$$

In this formula L represents the number of ANN layers (including input and output layer), P is the number of processing nodes and T is the number of training patterns. The formula represents the following process: a central control PE distributes the error vector to the processing elements and, after updating and computing, the PEs sending their error contribution back to the control PE.

b) *Exemplar parallelism (EP)*: In EP the training set is distributed across PEs with identical ANNs, i.e. identical weight matrices and topology. In contrast to NP, the ANN connection weights are not adapted after presentation of each dataset row (online/incremental training) but after presentation of all rows of the dataset to the network (offline/batch training)[2]. Typically, this is realized as follows: for each dataset row which is propagated through the network, i.e. classified, the calculated error gradients are stored and accumulated in special weight change matrices. After each epoch the weight change matrices of all processing elements are transferred to the control PE where they are averaged and used to adapt the original weight matrices (training). These are then transferred back to the other PEs in order to replace the previous weight matrices so that all processing elements train identical networks. Hence, we can deduce the following formula for the number of messages for EP.

$$\#Messages = 2 * (P-1) \quad (2)$$

As stated before, the number of messages necessary for the training compared to NP is considerably smaller as there are only two broadcast messages per processing element necessary per epoch.

IV. IMPLEMENTATION

The implementation was realized in three steps: development of a sequential algorithm on the PowerPC processing element of the Cell processor, replacement of the sequential instructions by single instruction multiple data vector instructions (SIMD/Altivec) and distribution of the SIMD code over

the processing elements. While the description of the sequential algorithm is omitted, we have described the conversion of sequential instructions to SIMD instructions (SIMDization) and the implementation of the two parallelization approaches.

A. SIMDization

In this step all computations in the sequential algorithm were modified to support four operands of the 32-bit float type at once by issuing the corresponding 128-bit Altivec/SIMD operations. To allow an efficient use of the SIMD instructions, it was necessary to adapt the layout of the data so that the operands were stored in consecutive order. For this reason the weight matrices were transposed into a row-major order, i.e. matrices were stored row wise in memory. Among other modifications special effort was put into the weight update in the backpropagation phase of the training. As the network topology was defined to be a feedforward neural network with two layers of weights, the backpropagation routine is also realized in two steps, one for each layer of weights. In the first part of the routine the gradients for the weights between hidden and output layer are summed together and the delta values are computed. In the second part the gradients for the weights between input node and hidden node layer are accumulated. Besides the implicit speed up due to the use of the SIMD instructions (four instructions are processed at once) it is also noteworthy that the number of branches compared to the scalar implementation is reduced by a factor of 16.

B. Node parallelism

NP was realized as depicted in figure 2. In contrast to the NP implementation described in [9], in this implementation the associated connection weights were also distributed. Hence, in an ANN with a single hidden layer of neurons these are the connection weights from input to hidden and from hidden to output layer. The number of messages of this approach is given in formula 3. Please note that one-time data movements, i.e. steps 1a-2b in figure 2, have been omitted.

$$\#Messages_{epoch} = B_{loads} * P + T * 3 * P (+P) \quad (3)$$

In this equation $B_{loads} = \frac{\#Dataset\ rows}{B_{size}} + 1$ describes the number of transfers necessary for the assigned dataset rows from RAM to each SPE P , until the Buffer of $B_{size} = \frac{\#Dataset\ rows}{0.5 * freespace}$ is filled. Multiple transfers may be necessary as the maximal DMA transfer size is restricted to 16KB.

The $T * 3 * P$ part of the equation represents the learning phase. Hence, this includes the transfer of the partial results with the associated mean squared error (MSE) as shown in figure 2, step 2f. It further includes a continue or termination signal from the PPE (fig. 2, step 1g or 1h, respectively). If a continue signals has been received the summed up partial values of the output nodes for every dataset row are gathered (fig. 2, step 2g).

Finally, the optional part $(+P)$ describes the sending of the partial weight matrices from all SPEs back to RAM if one of the termination conditions is triggered (MSE or defined number of epochs reached).

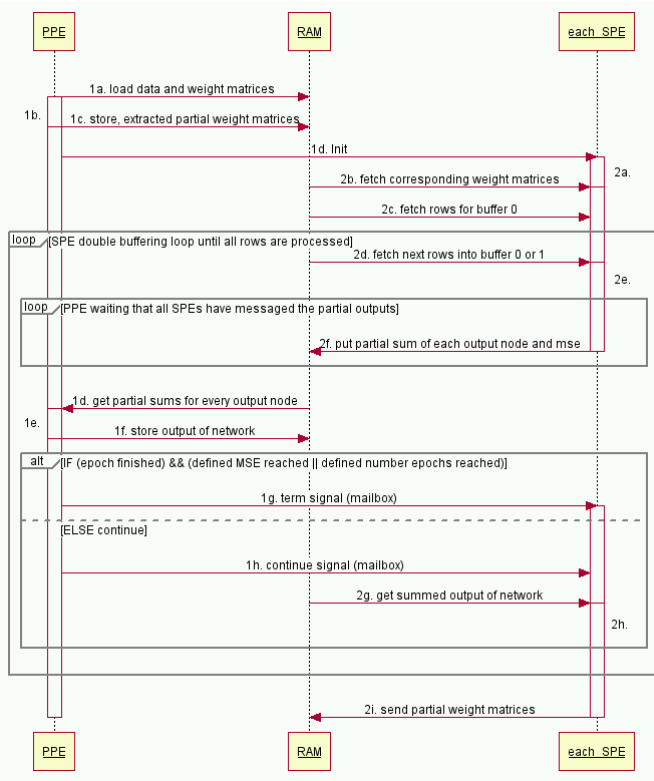


Figure 2: Sequence diagram of NP. Main entities in the figure: the PPE (power processing element of the Cell), the RAM (main RAM of the Cell) and each_SPE (each participating synergistic processing element of the Cell with its own private memory). All each_SPE objects execute the same code although operating on different connection weights and neurons.

C. Exemplar parallelism

EP was implemented as shown in figure 3. The number of messages of this approach is given in the following formula. Please note that one-time data movements, i.e. steps 1a-1c in fig.3, have been omitted.

$$\#Messages_{epoch} = P + B_{loads} * P + 2 * P \quad (4)$$

As before, P described the number of processing elements which are used (excluding the PPE). Hence, the first P corresponds to the retrieval of the updated weight matrices (fig.3, 2b). As in the previous equation B_{loads} denotes the transfer of the current weight matrices and double buffering. The $2 * P$ of the equation denote the transfer of the accumulated gradients and training set errors (fig.3, 2f) and the termination or continue notification sent by the PPE. Differences to formula 2 result from the double buffering and the notification signal at the end of the transfer.

V. EVALUATION

For the evaluation of the parallelization modes we define the speed up $S(P) = \frac{T(1)}{T(P)}$ where $T(1)$ is the computation time with a single thread and $T(P)$ is the time needed using P threads. For both parallelization strategies speed up vs. dataset

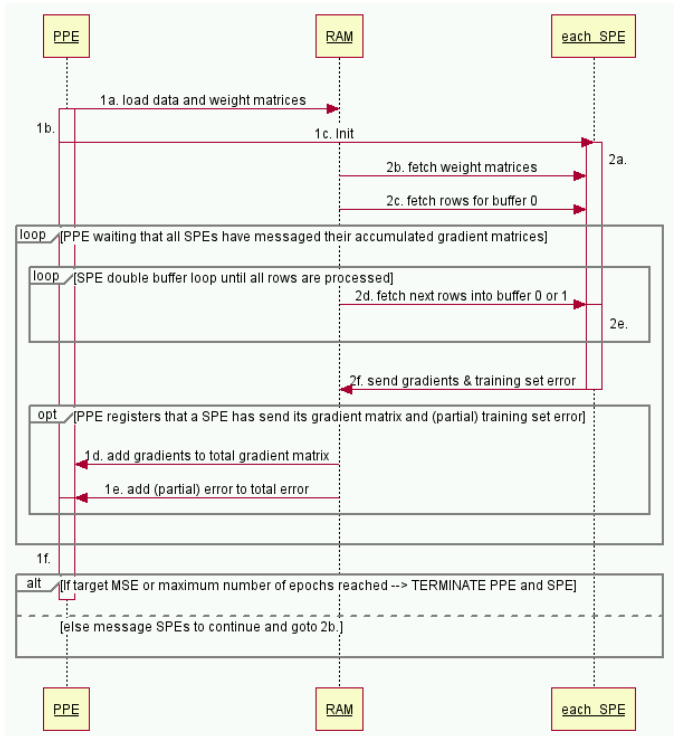


Figure 3: Sequence diagram of EP. Main entities in the figure: the PPE (power processing element of the Cell), the RAM (main RAM of the Cell) and each_SPE (each participating synergistic processing element of the Cell with its own private memory). All each_SPE objects execute the same code although operating on different dataset rows.

size and speed up vs. network size was evaluated. Following [9] the evaluated neural network topologies were set using $N - \frac{3}{4}N - \frac{1}{2}N$. Thereby N denotes the number of input nodes $\frac{3}{4}N$ is the number of hidden nodes and $\frac{1}{2}N$ is the number of output nodes. Both parallelization strategies were evaluated using different numbers of synergistic processing elements ($P = 2, 4, 8, 16$).

A. Evaluation of node parallelism on the Cell

For network scaling vs. speed up the following network topologies were evaluated: $N = 12, 25, 37, 50, 62, 75, 87, 100, 150, 200, 250, 300$. For each of the evaluated topologies the dataset size was 1000 rows. The corresponding performance curve is shown in the upper chart in fig. 4.

We found that the NP-4 (node parallelism utilizing 4 synergistic processing elements and the PPE) with $N = 200$ delivers the best speed up of 1.34. Due to memory restrictions on the SPE nodes the size of networks that could be tested was restricted, e.g. using only 2 SPEs the maximal network size that could be simulated was $N = 120$, while NP-16 allowed the evaluation of a network of $N = 300$. While it could be expected that NP would not outperform EP (see next section), the current implementation of this parallelization strategy is not satisfying and worse than expected. Despite the fast hardware interconnections on this architecture, the number of computations necessary, even with the largest network, did

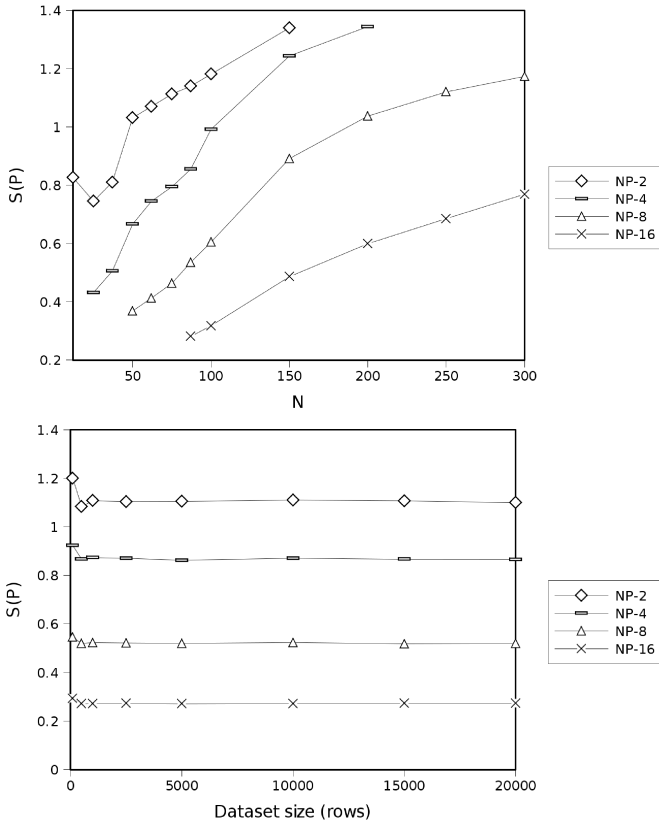


Figure 4: Projection of speed up for node parallelism (NP). In the upper figure the speed up of the NP with 2, 4, 8 and 16 SPEs vs. network size is shown. Evaluated network topologies were $N = 12, 25, 37, 50, 62, 75, 87, 100, 150, 200, 250, 300$ with dataset size set to 1000. In the lower figure speed up with 2, 4, 8 and 16 SPEs vs. dataset size is shown. Evaluated datasets were of sizes 100, 500, 1000, 2500, 5000, 10000, 15000 and 20000 with network size set to 87.

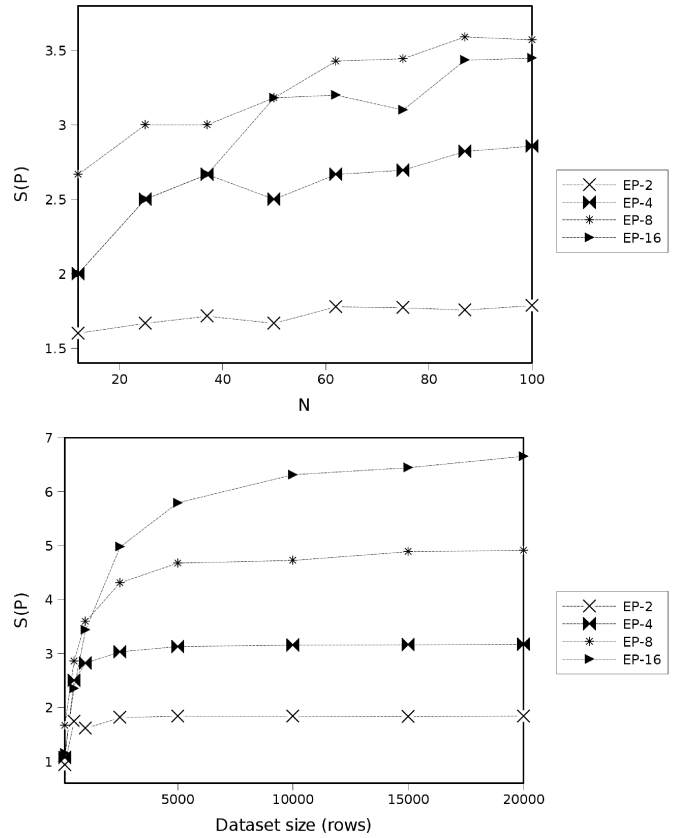


Figure 5: Projection of speed up for exemplar parallelism (EP). In the upper figure the speed up with 2, 4, 8 and 16 SPEs vs. network size is shown. Evaluated network sizes were 12, 25, 37, 50, 62, 75, 87 and 100. Not all network sizes compared to NP could be tested due to memory limitations. In the lower figure the speed up of EP vs. dataset size scaling is depicted. Evaluated dataset sizes were 100, 500, 1000, 2500, 5000, 10000, 15000 and 20000 with a network size was set to 87.

not require enough computing time to compensate for the large number of synchronization messages.

For scaling dataset size vs. speed up ($N = 87$) the best speed up achieved was 1.2 with 2 SPEs (fig. 4, lower chart). Only a marginal correlation between speed up and size of the dataset was found.

B. Evaluation of exemplar parallelism on the Cell

Top speed up for network size scaling (fig. 5, above) was 3.4 using EP-8 (exemplar parallelism utilizing 8 synergistic processing elements and the PPE) with a network size of $N = 100$. EP can provide a considerable speed up for network size scaling. However, due to fact that the network weight adaption and gradient summation is based on the PPE probably some of the performance gained over single SPE mode is lost. Another limitation is that network size cannot be increased further with the current implementation.

Exemplar parallelism yields a good speed up when scaling dataset size (fig. 5, below). Top speed up was 6.6 over the single SPE mode for a network size of $N = 87$ and a dataset size of 20000 using 16 SPEs. As reported by [9], EP exhibits

a strong correlation between performance and dataset size. However, while their limiting factor is communication time this is not the limitation here. In fact, the time needed for DMA transfers is much smaller than the time the SPEs must wait until the PPE has adapted the weight matrices.

VI. DISCUSSION AND RELATED WORK

This paper is based on the vision of using existing home entertainment systems as central controllers for Smart Home installations. Entertainment systems are usually switched on or in standby mode and have powerful processors to support even complex machine learning algorithms. To show the potential of this idea we have described the three dominating systems on the market today. We have further conducted a case study on an IBM dual Cell blade which incorporates the same processor as the Sony PlayStation 3. In this case study we have described and implemented two of four parallelization strategies for ANN training on this processor.

While both strategies show a noticeable speed up compared to a sequential implementation, the gained performance due

to exemplar parallelism (EP) is greater than for node parallelism (NP). For EP on the other hand the coordinating power processing element (PPE) was identified as a bottleneck. This could be improved by shifting the gradient summation from the PPE to a SPE. Further improvements would be the employment of 16 bit fixed point values. This would allow a doubling of the execution speed for SIMD as more values could be processed in a single SIMD instruction (4 with 32 bit floats vs. 8 when using 16 bit values). This could further enable the training of larger networks.

Additionally, it is emphasized that the two parallelization strategies utilize different training schemes. While node parallelism employs an incremental training, in which the ANN's weights are updated after the presentation of each pattern, exemplar parallelism employs batch or offline training in which the networks' weights are adapted after the presentation of a set of patterns. The size of the set is thereby determined by the number of used processing elements and the number of overall patterns. According to [2], incremental training may be more efficient for highly redundant data and can escape local minima. On the other hand, batch learning is known to create ANN classifiers with better generalization properties[3]. Bishop [2], also states that the advantages of incremental training over batch training may be reduced if the weight adaptation is done more frequently and not only after presenting the complete set of training patterns to the network.

In the presented EP implementation this is the case as the complete training dataset is distributed across the processing elements and weight changes are computed on these subsets of the training data. The update frequency could be further increased, e.g. by requiring the weight update after a customizable number of patterns. However, this should be considered carefully as it will increase the number of synchronization messages and thus, will reduce speed up.

In summary, it was found that the worst cases for both parallelization strategies here are similar to [9]. For EP this worst case is a small dataset and a large network, while for NP the worst case is a small network and a large dataset. As a general rule for this implementation we suggest to use EP as long as the network size allows it, otherwise NP should be employed.

The potential of the Cell for machine learning, was also investigated by other researchers. The authors of [12] could show a 55x speed up of the training of spiking neural networks for image recognition over a serial implementation on the PPE. In [13], researchers showed that the distributed calculation of a correlation matrix gives a speed up of up to 4.5 for 8 SPEs for large matrices (2000x2000). In [19], the authors presented a parallel implementation of a Kohonen SOM on a single PS3 and a cluster of PS3s. For the experiments based on a single PS3 they reported a speed up of ca. 6 when using 6 SPEs. They further observed a slight dependence of speed up and Kohonen map size.

Based on these positive results we consider the PS3 a worthy platform for use as a Smart Home controller. However, researchers interested in evaluating this system for their Smart Home architecture should currently use an old firmware version in order to use the "Other OS" functionality or use

home brew software channels. Additionally, the employment of home brew software could yield further performance improvements as the graphics processing unit (GPU) can be directly accessed.

While the Wii input devices and Wii console have been used for scientific research in the past[17], the low performance and the lack of official support by the manufacturer may not generally suggest it's use as a Smart Home controller. However, the console can be programmed using unofficial development kits which provide full hardware access. Hence, using the most wide spread console system today as a central Smart Home control device is not denied.

For the Xbox 360, [4] showed the feasibility of implementing a linear genetic programming algorithm for solving classification and regression problems using XNA. For their implementation, they employed the Xbox 360 Tri-Core PowerPC CPU and the GPU. While their implementation could not outperform a desktop system, they conclude that from a practical programming and future hardware point of view, it is worthwhile to use the Xbox 360 as an evolutionary computation general purpose programming GPU platform. Additionally, the XNA platform will soon incorporate support for the novel Kinect device. This poses an opportunity for novel Smart Home services, making the Xbox 360 into an attractive Smart Home controller platform.

VII. CONCLUSION

In this paper we showed that home entertainment systems like the Sony PlayStation 3, hold the potential to bring complex and computing intense algorithms to the end user. Due to their high grade of distribution, such entertainment systems may be found in residences of many potential users, they are connected to the local area network and are often continuously on. Furthermore, they provide a constant hardware configuration and will even provide a stable base system in different countries. They usually have accessible device interfaces such as USB which can be used to integrate Smart Home appliances, sensors/actuators or wireless transceivers. Development kits are available for all of the three major systems, although sometimes only unofficially. Another advantage could lie in the user interface of the systems. As this is already known by the user an additional learning phase could be avoided. Thus, in conclusion we suggest to thoroughly consider existing home entertainment systems as central building blocks for future Smart Home architectures.

VIII. ACKNOWLEDGMENT

We express our gratitude to Werner Kriechbaum of IBM Boeblingen Lab for supervision of the diploma thesis [11] in which the Cell based neural network simulator was developed and benchmarked. We would further like to thank IBM Deutschland GmbH for providing the hardware to develop and test the simulator. This work was further partially funded by the German Federal Ministry of Education and Research through the landmark project.

REFERENCES

- [1] Rezaul Begg and Rafiul Hassan. Artificial Neural Networks in Smart Homes. pages 146–164. 2006.
- [2] Bishop CM. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
- [3] Rumelhart DE, Hinton GE, and Williams RJ. Learning internal representations by error propagation. In Rumelhart DE, McClelland JL, et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, 1987.
- [4] Wilson G and Banzhaf W. Linear genetic programming GPGPU on Microsoft's Xbox 360. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*, pages 378–385, 2008.
- [5] Demuth HB, Beale M, and Hagan MT. *Neural Network Toolbox User's Guide*. The MathWorks, 2007a edition, Mar 2007.
- [6] Andrews J and Baker N. Xbox 360 system architecture. *IEEE Micro*, 26:25–37, 2006.
- [7] Kurzak J, Buttari A, Luszczek P, and Dongarra J. The PlayStation 3 for High-Performance Scientific Computing. *Computing in Science and Engineering*, 10:84–87, 2008.
- [8] Hsu K, Chiang Y, Lin G, Lu C, Hsu JY, and Fu L. Strategies for Inference Mechanism of Conditional Random Fields for Multiple-Resident Activity Recognition in a Smart Home. In Garcia-Pedrajas N, Herrera F, Fyfe C, Benitez JM, and Ali M, editors, *Trends in Applied Intelligent Systems - 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2010, Cordoba, Spain, June 1-4, 2010, IEA/AIE*, volume 6097 of *Lecture Notes in Computer Science*. Springer, 2010.
- [9] Pethick M, Liddle M, Werstein P, and Huang Z. Parallelization of a Backpropagation Neural Network on a Cluster Computer. In Gonzalez T, editor, *Parallel and Distributed Computing and Systems, IASTED PDCS, Marina del Rey, USA*, volume 1. IASTED/ACTA Press, 2003.
- [10] Riedmiller M and Braun H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE Intl. Conf. on Neural Networks*, pages 586–591, San Francisco, CA, 1993.
- [11] Scholz M. Development of an artificial neural network on a heterogeneous multicore architecture to predict a successful weight loss in obese individuals. Diploma thesis, Chair of Bioinformatics, University of Leipzig, 2008.
- [12] Bhuiyan MA, Jalsutram R, and Taha TM. Character recognition with two spiking neural network models on multicore architectures. In *Computational Intelligence for Multimedia Signal and Vision Processing, 2009. CIMSVP '09. IEEE Symposium on*, volume 30, pages 29–34, April 2009.
- [13] Jaiswal MK. Acceleration of Correlation Matrix on Heterogeneous Multi-Core CELL-BE Platform. *International Journal of Advanced Science and Technology*, 27, 2011.
- [14] Sawsan MM, Ahmad L, and Langensiepen C. Occupancy Pattern Extraction and Prediction in an Inhabited Intelligent Environment Using NARX Networks. In *Proceedings of the 2010 Sixth International Conference on Intelligent Environments, IE '10*, pages 58–63, Washington, DC, USA, 2010. IEEE Computer Society.
- [15] Hagan MT and Menhaj M. Training feed-forward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, 1994.
- [16] Brdiczka O, Reignier P, and Crowley J. Detecting Individual Activities from Video in a Smart Home. In Bruno Apolloni, Robert Howlett, and Lakhmi Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 4692 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007.
- [17] Rehrig R, Marks J, Janansky S, Aiello L, Kiamilev F, and Barner K. Repurposing Commodity Hardware for use as Assistive Technologies. In *Proceedings of RESNA 2010 Conference, International Symposium On Quality Of Life Technology*, 2010.
- [18] Scarle S. Implications of the Turing completeness of reaction-diffusion models, informed by GPGPU simulations on an Xbox 360: Cardiac arrhythmias, re-entry and the Halting problem. *Computational Biology and Chemistry*, 33(4):253 – 260, 2009.
- [19] McConnell SM. Self-Organizing Maps on the Cell Broadband Engine Architecture. *Journal of Physics: Conference Series*, 256(1):012013, 2010.
- [20] Sony, Toshiba, and IBM. *Cell Broadband Engine Programming Tutorial*. IBM Systems and Technology Group, 2.1 edition, Mar 2007.
- [21] Nordström T and Svensson B. Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing*, 14(3):260–285, 1992.