Research Activities on the Single-Chip Cloud Computer

Microprocessors and Digital Systems Laboratory (MicroLab)
School of Electrical and Computer Engineering
National Technical University of Athens (NTUA)

Contents

1 Introduction

This edition includes a summary of the major projects that have occupied the Single-Chip Cloud Computer (SCC) ranging from August 2011 up to March 2013. The SCC is an experimental processor created by Intel Labs. It is a 48-core "concept vehicle" created as a platform for many-core software research [7]. The purpose of the current edition is to familiarize any interested party with the work that has been performed at MicroLab using the aforementioned piece of hardware.

In the current report, we will not be delving into the technical details of the SCC's architecture and programming style. Any interested reader is encouraged to inspect related publications for in-depth understanding of this processor [7, 13]. However, we believe that a brief description is deemed necessary, at least to cover some key terminology: The SCC processor consists of 48 P54C cores that are interconnected with a high speed mesh network. The cores are organized in pairs called *tiles*. SCC cores feature a private memory space like typical systems. However, at the lowest level, a small memory bank is assigned to each core for the submission and retrieval of messages. This memory bank is called Message Passing Buffer (MPB) and enables inter-core communication.

The SCC chip is placed on a board that communicates with a standard server (Management Console Personal Computer – MCPC) through a PCIe cable. The MCPC is responsible for dispatching processes to user-defined cores through `pssh`. Parallel code is written on the MCPC, using functions of a message passing library, called RCCE (pronounced "rocky") [14]. This library enables synchronization between cores, message passing and dynamic frequency and voltage scaling [13].

Each of the following Sections is dedicated to a single research topic that occupied the SCC platform and has been authored by the responsible researcher in each case. Dissemination of the discussed topics to conferences or workshops is cited accordingly. The interested reader can follow these citations for further information about each topic. Alternatively, contacting the responsible researcher is also encouraged. At the end of each Section we list contact information for each researcher.

In the context of our work, the SCC infrastructure is used under the supervision of the Principal Investigator, Dr. Antonis Papanikolaou. The co-authors of this edition would like to thank Prof. Dimitrios Soudris from MicroLab-ECE-NTUA and Prof. Elias S. Manolakos from DIT-UOA, for their additional supervision. Finally, on behalf of all involved researchers, we would like to thank Dr. Xavier Vera and Mr. Enric Herrero of Intel Labs Barcelona for enabling use of the SCC; Intel Labs Braunschweig for providing the related hardware and the Intel MARC communities.

Contact Person: Dimitrios Rodopoulos – PhD Researcher at NTUA/ICCS – drodo@microlab.ntua.gr

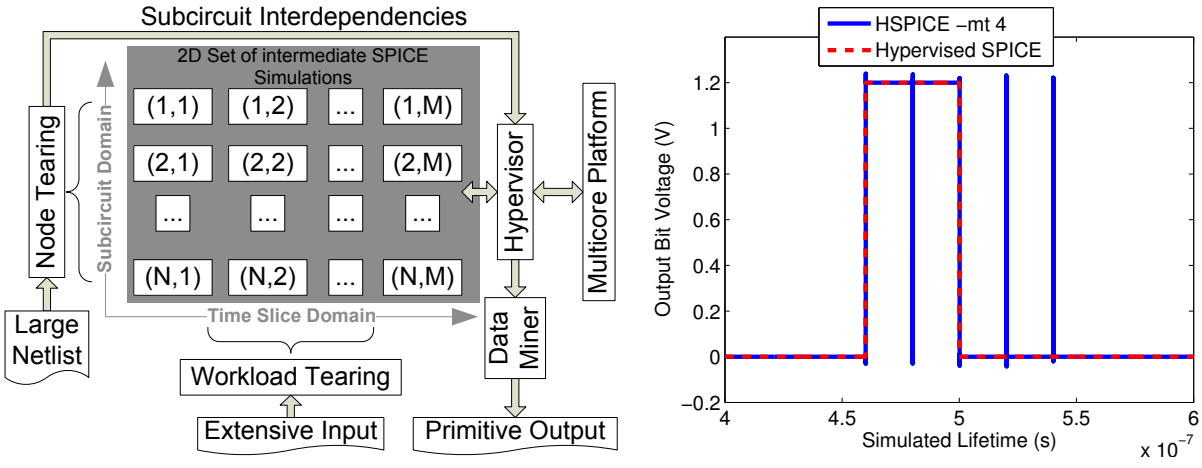
2 Hypervised SPICE for Massive Netlists and Workloads

Abstract: Advances in modern computer architecture have drastically pushed application development to parallel programming. Industry standard SPICE applications have also followed this trend and multi-threaded execution has led to significant advances in the field of circuit simulations limited only by the memory of the respective runtime systems. Therefore there is a need for flexible memory conservative and scalable techniques for parallel application development. This project attempts to attack this issue with the use of a versatile development framework applied in the field of circuit simulations. One of the platforms used to prove the versatility of the framework is the SCC.

Ever since the begining of circuit design there has been a need for circuit simulations. Nowadays digital circuits are an integral part of everyday life. Everything from smartphones to super computers depends on them. Therefore the need for detailed circuit simulations is at it's peak. The field of circuit simulations has constantly grown more computationally intensive over the years. What's more, modern reliability models require massive computational resources in order to complete the simulation.

In order to fill the need for fast and accurate simulations there are many approaches to the optimisation of SPICE applications . Single thread optimisations have been performed over the years on the SPICE kernels that run the simulations [15]. Since multithreaded execution has been introduced developers realised that parallel programming is a promising field and with the advances in modern processors it is only logical that SPICE applications would follow the trend and try to harness the new processing potential [16]. There have also been attempts to run SPICE on specialised hardware such as FPGAs [9] or GPUs [6] with promising results. Many steps have been taken and significant advances have been made, though little is done to minimise memory usage on large netlists and massive workloads, thus many simulations stop to a halt due to bad alloc errors [4].

This project attempts to attack the issue from a different angle. Instead of working in the SPICE kernel source code the framework uses a wrapper and remains agnostic regarding the executable code. In order to achieve that we adopt two methods of data partitioning *Workload* and *Node tearing* [12]. When it comes to input signals one can easily comprehend the concept of *Workload tearing* which consists of signal partitioning on the axis of time. What must be pointed out is that the partitioning method is sophisticated enough to ensure that the accuracy of the simulation is not compromised. On top of that we employ another method of data partitioning referenced as *Node tearing*. The initial netlist is broken down to components that have directed acyclic dependencies between them. The intermediate results are forwarded by the framework among the components. This is represented in a task graph parsed by the framework, enabling the distribution of the tasks described, over a set of execution nodes. Running the simulations independently minimizes the need for massive memory allocation requests thus allowing the simulations to complete.

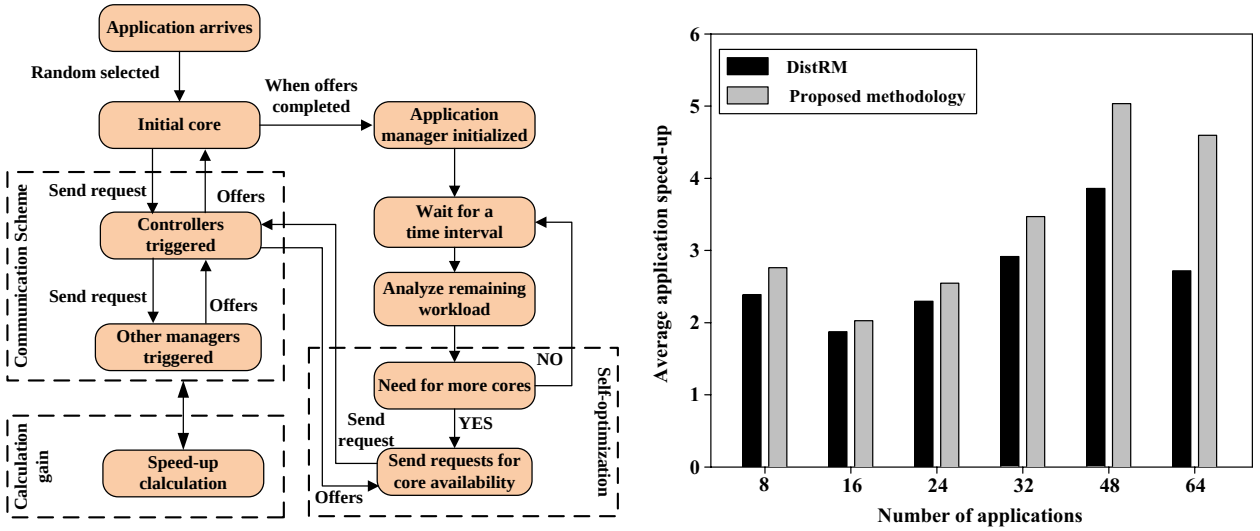


(a) Framework concept

(b) Comparison of results

Figure 2.1: Framework concept and results comparison

Contact Person: Grigorios Lyras – Undergraduate Student at NTUA – greglyras@gmail.com



(a) Overall flow of the proposed methodology

(b) Average application speed-up using the speed-up function presented in [11]

Figure 3.2: Distributed Run-Time Resource Management on Single-Chip Cloud Computer (SCC)

3 Distributed Run-Time Resource Management

Abstract: Prevalent solutions for modern embedded systems and general computing employ many processing units connected by an on-chip network leaving behind complex superscalar architectures. Here, we couple the concept of distributed computing with parallel applications and present a workload-aware distributed run-time framework for malleable applications. The framework is responsible for serving in a distributed way and at runtime, the needs of malleable applications, maximizing resource utilization avoiding dominating effects and taking into account the type of processors supporting platform heterogeneity, while having a small overhead in overall inter-core communication. The framework has been implemented as a run-time service on the SCC and has been compared against a state-of-art run-time resource manager. Experimental results showed that our framework has on average 70% less messages, 64% smaller message size and 20% application speed-up gain.

The run-time resource management paradigm has been revealed as a key challenge to modern multi-core systems and it has become prominent due to the run-time dynamism of modern parallel applications and platforms. In large-scale execution environments such as multi-cluster systems and grids, resource availability may vary due to resource failures and because resources may be added to or withdrawn from such environments at any time thus making the resource allocation problem significant to the overall system performance. In addition, dedicated resources or fixed resource allocation strategies fail to provide the desired high performance that applications expect from large pools of resources.

In this work, we couple the concept of distributed computing with parallel applications and we present a workload-aware distributed run-time framework for malleable applications running on many-core platforms. The proposed framework is based on the idea of local controllers and managers while an on-chip intercommunication scheme ensures decision distribution. The presented framework is responsible (i) for serving, at run-time, the needs of malleable applications, in terms of processing cores; (ii) makes sure that the application will get the optimum number of cores avoiding dominating effects; (iii) it takes into account the type of processors best utilizing any platform's heterogeneity; and (iv) it has a small overhead in overall core intercommunication.

Figure 3.2b presents the average application speed-up using the speed-up function presented in [11]. Speed-up is defined as the ratio of the total number of turnarounds performed for all applications divided by the total workload. The presented framework achieves on average 20% better application speed-up than DistRM. This can be explained by the fact that in the presented framework cores are not disturbed so often by messages during their application execution and thus completing the applications faster. On the other hand, due to the large number of messages sent by the application agents in DistRM, cores stop their functionality more frequently in order to answer to these messages, thus delaying the execution.

Contact Person: Iraklis Anagnostopoulos – PhD Researcher at NTUA/ICCS – iraklis@microlab.ntua.gr

4 SDC Mitigation on SCC Streaming Applications

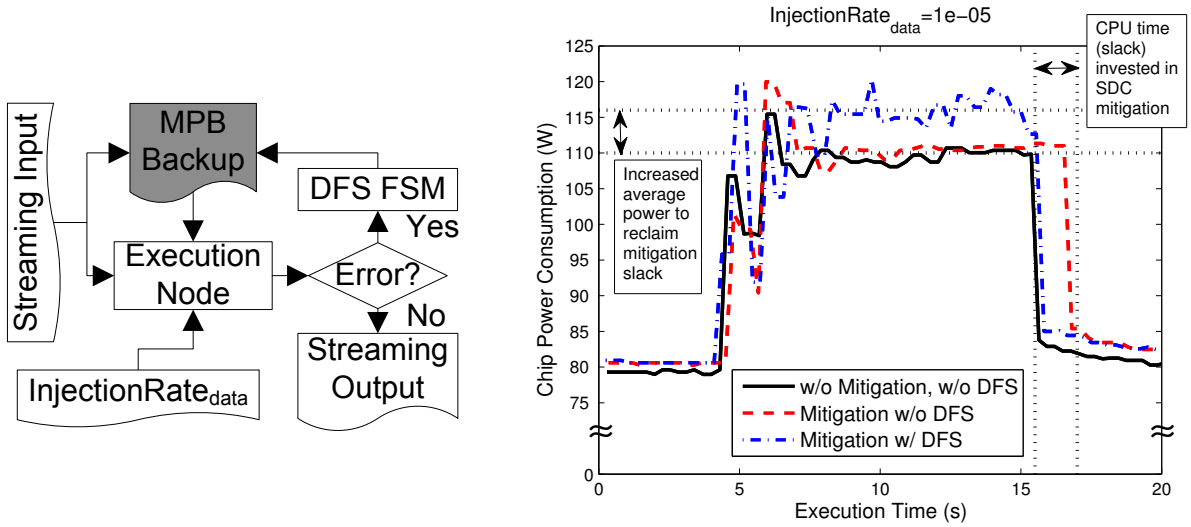
Abstract: This work presents a hybrid HW-SW mitigation scheme against Silent Data Corruptions (SDCs) on the SCC platform. The methodology assumes *error detection* implemented in hardware (HW). Also, the MPBs of each core are assumed to implement both *error detection and mitigation* in HW. Upon error detection, a software (SW) rollback is triggered, while retrieving error-free copies of corrupt data from the closest MPB. Given the context of streaming applications, the processing time invested in mitigation needs to be reclaimed. The dynamic frequency scaling utilities of the target platform are used to reclaim that slack, depending on the level of SDC aggression.

A usual distinction between erroneous states found in digital systems is between permanent and transient errors [22]. We focus on the latter category, namely the corruption of the logic state of a circuit node because of an intermittent phenomenon. Typical examples of such errors are Single- or Multiple-Event Upsets caused by cosmic particles or erroneous logic states due to IR-drops in the power or ground grid of the platform [24]. In the past, transient errors of that sort where exclusively mitigated in HW. However, with decreasing device dimensions, the mitigation overhead purely in HW is becoming too extreme [10].

Our focus is narrowed down to SDCs, which affect the data plane of the target system, without altering the control flow of the executing application. SDCs are injected at runtime in the application's data plane. The intensity of SDCs is abstracted by the $\text{InjectionRate}_{\text{data}}$ metric, representing the probability of a data plane bit being flipped [17]. Assuming binary corruption of separate bits as independent events, we can calculate the probability for a streaming data structure to be corrupt.

The target application of this work is a data-parallelized MJPEG decoder. A single frame decoder [20] occupies a triplet of SCC cores (execution nodes), hence we end up with 16 JPEG decoders decoding in parallel the frames of an MJPEG video benchmark. This being a streaming application, we can identify the data required by each execution node of each decoder instance. An error-free copy of each key data structure is kept in the local MPB once the former is created. That way, assuming that the MPB is HW-protected against transient errors, we can retrieve the error-free copy upon error detection. This retrieval is SW-controlled, by altering the source code of the streaming application (see Figures 4.3a).

Even if demand driven, the SW rollbacks are consuming processing time. Especially in case of increased $\text{InjectionRate}_{\text{data}}$ values, we need to reclaim the processing time invested in mitigation, in order not to affect application performance. We use the Dynamic Frequency Scaling (DFS) of the chip to accelerate a portion of the execution, in order to meet the specification about decoded frames per second. Temporarily increasing the frequency, comes with a power consumption penalty, as illustrated in Figure 4.3b. We have explored the average power impact that slack reclaiming with DFS has under different $\text{InjectionRate}_{\text{data}}$ [18]. Absorbing mitigation overhead using DFS is upper bounded by the highest frequency that the SCC cores can achieve.



(a) Simplified flowchart of the SW-enabled rollbacks upon SDC detection

(b) Impact of reclaiming slack with DFS on the transient power of the SCC: the temporal overhead translates to increased average power

Figure 4.3: Concept and transient power trace for the proposed HW-SW SDC mitigation technique

Contact Person: Dimitrios Rodopoulos – PhD Researcher at NTUA/ICCS – drodo@microlab.ntua.gr

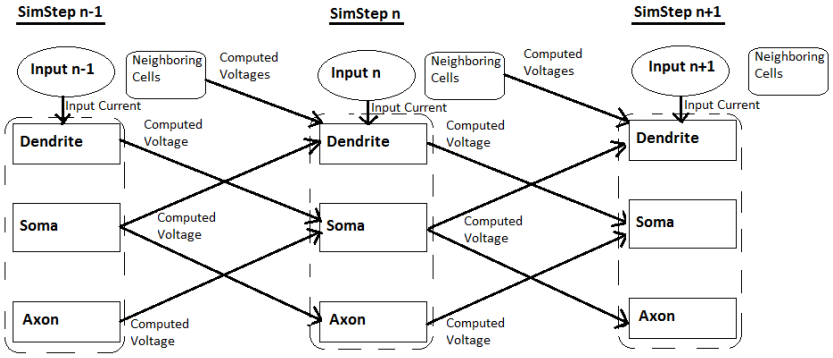


Figure 5.4: Data flow and communication between cells and compartments of the target cell

5 Scalable Simulation of the Inferior Olivary Nucleus

Abstract: This work contains a model of the inferior olive brain cells network, an important subsystem for the functionality of the cerebellum, as depicted in a mesh, with varying network size and level of interconnectivity between the cells. The cell model is comprised of 3 functional units (compartments), representing the dendrite, soma and axon parts of the biological neuron. These units should be mapped to SCC tiles accordingly and communicate with each other. The network is organized as a typical mesh. Using outside stimuli and the previous cell states as input parameters, the voltage output of each cell compartment is computed. Communication and computational costs are prohibitive of simulating large, complicated networks so far, thus efforts to increase simulation speed is of great value.

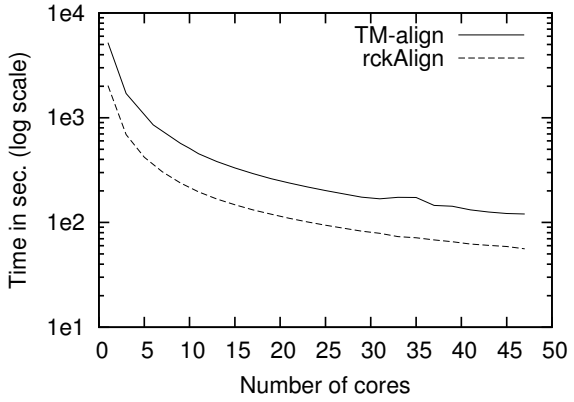
The human brain processes data in a very different way than typical man-made computing systems. Brain cells process data using the propagation of spikes between a network of neuron cells. Information is encoded in the spikes' frequency, amplitude and shape pattern. The exact details of these mechanisms are still heavily debated [23]. The cerebellum has the greatest concentration of neurons in the brain and is imperative for motor learning and synchronization of body movement. In particular, the inferior olive cerebellum subsystem is of great importance in such functions and, thus, a major subject of scientific debate [3]. An application simulating the voltage levels of the olive cells is highly valued and needed, as well as computationally taxing. The porting of a scalable model [1, 8] on a high-computational capacity platform as the SCC is described below.

The inferior olive cells are represented in a mesh of variable dimensions where each element is a cell of the three compartments(or compartments in neuroscience), the soma, the axon and the dendrite. Each mesh element can be fed by an input current as external stimulus. As shown in Figure 5.4, the dendrite is the compartment that operates as the cell input. It models connectivity between neighboring cells (by receiving voltage inputs from other cells) and is fed the previous cell state and external input stimuli. The compartments of a single cell also communicate with each other (some voltage level). After information of the previous step (soma voltages) is gathered and input of the current step is stored in a buffer, the compartment computes all relevant parameters to store their new state and the simulation advances for as many steps as indicated by the simulation process. The voltage outputs of each cell for every simulation step are recorded in an output file.

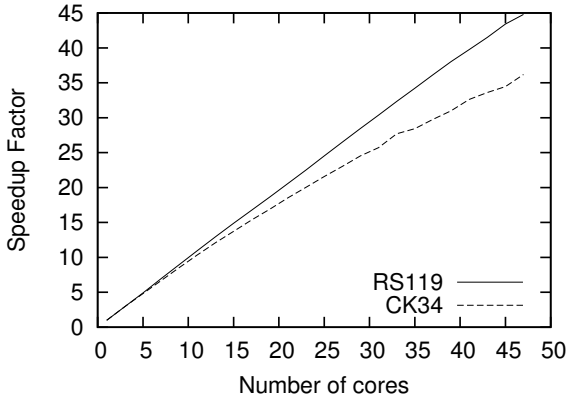
Currently the output evoked by the input stimuli is a certain kind of spike typical of the IO cell (complex spike) that triggers once at a particular time period. The input current values are hard-coded and the correct response values are recorded and compared against a reference output. However, a possible extension of the work will incorporate different input stimuli completely controlled by the user of the application so as to provide more feedback on the behavior of the simulated inferior olive cells. Furthermore, communication between the cells is restricted to immediate neighbors. Higher degrees of interconnectivity can be pursued to replicate real-life cell communication, which will greatly increase the value of the application. In such case, the increase in communication cost needs to be weighed. Thus the computational capacities of the SCC will be greatly tested and exploited, even more than by simply increasing the mesh size.

Since the soma compartment has workload roughly equal to the sum of the dendritic and axonal workloads, the project plans on using SCC tiles: one core handles soma computation and the other one is assigned the dendrite and axon compartments. This will also exploit the locality of two cores belonging to one tile. However, since the dendritic compartment is responsible for communicating with neighboring cells and registering incoming stimuli, if greater amount of inter-connectivity between cells is pursued, the dendrite may exhibit greater workload than the soma and different porting patterns will be explored.

Contact Persons: Giorgos Chatzikonstantis – Undergrad. at NTUA – mightyfelix@hotmail.com
Dr. Christos Strydis – Neuroscience Department, Erasmus Medical Center – c.strydis@erasmusmc.nl



(a) Performance comparison of parallel *rckAlign* with that of distributed TM-align software (C port) for the Chew-Kedem dataset as the number of slave cores used is increasing.



(b) Speedup achieved by *rckAlign* as the number of slave cores is increasing (from 1 to 47) for the Chew-Kedem (CK34) and the Rost-Sanders (RS119) datasets.

Figure 6.5: Performance comparison and speedup achieved by SCC port of TM-align

6 Protein Structure Comparison on Many-Core NoCs

Abstract: This work presents *rckAlign*, an implementation of the popularly used TM-align PSC algorithm, designed for the Single-Chip Cloud Computer (SCC). We developed a skeleton library, *rckskel*, and implemented a master-slaves variant of TM-align to exploit the parallelism offered by the SCC. We observed a 44-fold speedup relatively to a single-core of the SCC. A key aspect of the performance of *rckAlign*, is the almost linear speedup achieved with the number of SCC cores used as slaves. The method presented can easily be applied to other PSC algorithms and extended to running multiple PSC algorithms within the same SCC chip.

A typical task in bioinformatics is comparison of the structure of a protein with a database of known protein structures, one-to-many protein structure comparison (PSC), or when a set of multiple proteins is of interest, comparison of their structures to a whole database of known protein structures, many-to-many PSC. The unit operation in both these forms of PSC is the comparison of structures of a pair of proteins [21]. Computational demand of the one-to-many and many-to-many PSC problem are a result of factors such as: exponential growth of structural proteomics databases, pairwise protein structure comparison is computationally intensive and several pairwise comparison approaches are typically of interest to the researcher [2].

Algorithmic skeletons [5] allow a programmer to develop algorithms without specifying architecture dependent details. By nesting and combining skeletons desired level of parallelism can be introduced into different PSC methods. Using algorithmic skeletons allows programs to fully exploit the parallelism afforded by many-core processor architectures, while retaining the flexibility needed for experimenting with the level of parallelism and shared tasks (such as rotation of structures).

In order to facilitate development of PSC algorithms targeted for the SCC, we built a C library. The library we built is a small parallel programming library which implements algorithmic skeletons - *SEQ, PIPE, MAP, FARM* - in addition to providing convenient wrappers for common RCCE related tasks. Allocation of a core to a job is performed dynamically by the library, based on the number of jobs and the number of cores available for processing. A master-slaves implementation of the C code of TM-align, was developed using the *FARM* skeleton of the *rckskel* library [19].

As shown in Figure 6.5a faster processing times are achieved by *rckAlign* as compared to the distributed port, where a master process is running on the MCPC. There are two main reasons for this behavior: (a) disk access through the Network File System (NFS) creates a bottleneck when multiple processes are trying to access the data, and (b) high environment setup costs incurred while issuing remote processing.

As can be seen in Figure 6.5b an almost linear speedup is achieved by the master-slaves port of TM-align. We also observed that the larger the dataset the higher the speedup observed. These results suggest that many-core NoCs with fast interconnection networks and faster processor cores than the SCC will be ideal candidates for delivering high performance for all-to-all PSC tasks applied to large size protein databases, as needed for combinatorial drug design.

Contact Person: Anuj Sharma – PhD Candidate at UoA – asharma@di.uoa.gr

References

[1] Cerebellar application description. Technical report, Neurasmus – The Erasmus Neuroscience Company, December 2011.

[2] Mario Cannataro, Carmela Comito, Filippo Lo Schiavo, and Pierangelo Veltri. Proteus, a grid based problem solving environment for bioinformatics: Architecture and experiments. *IEEE Computational Intelligence Bulletin 3*, 1:7–17, 2004.

[3] Chris I De Zeeuw, Freek E Hoebeek, Laurens W J Bosman, Martijn Schonewille, Laurens Witter, and Sebastiaan K Koekkoek. Spatiotemporal firing patterns in the cerebellum. *Nature Reviews Neuroscience*, 12(6):327–344, 2011.

[4] N. Frohlich, V. Glockel, and J. Fleischmann. A new partitioning method for parallel simulation of vlsi circuits on transistor level. In *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*, pages 679 –684, 2000.

[5] H. Gonzalez-Velez and M. Leyton. A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers. *Software: Practice and Experiment*, 40(12):1135–1160, November 2010.

[6] Kanupriya Gulati, John F. Croix, Sunil P. Khatr, and Rahm Shastry. Fast circuit simulation on graphics processing units. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, ASP-DAC '09, pages 403–408, Piscataway, NJ, USA, 2009. IEEE Press.

[7] J. Howard, S. Dighe, S.R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V.K. De, and R. Van Der Wijngaart. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *Solid-State Circuits, IEEE Journal of*, 46(1):173 –183, jan. 2011.

[8] Sebastian Isaza. Olivocerebellar hardware modeling in the fpga lab. Technical report, Erasmus MC – Dept. of Neuroscience, June 2012.

[9] Nachiket Kapre. *SPICE2 – A Spatial Parallel Architecture for Accelerating the SPICE Circuit Simulator*. PhD thesis, California Institute of Technology – Pasadena, California, 2010.

[10] Jangwoo Kim, N. Hardavellas, Ken Mai, B. Falsafi, and J.C. Hoe. Multi-bit error tolerant caches using two-dimensional error coding. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 197 –209, dec. 2007.

[11] Sebastian Kobbe et al. DistRM: distributed resource management for on-chip many-core systems. In *Proc. of CODES+ISSS*, pages 119–128. ACM, 2011.

[12] G. Lyras, D. Rodopoulos, A. Papanikolaou, and D Soudris. Hypervised transient spice simulations of large netlists & workloads on multi-processor systems. In *Design, Automation and Test in Europe Conference and Exhibition 2013. Proceedings*, 2013.

[13] T.G. Mattson, R.F. Van der Wijngaart, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe. The 48-core scc processor: the programmer's view. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pages 1 –11, nov. 2010.

[14] Tim Mattson and Rob van der Wijngaart. Rcce: a small library for many-core communication. Technical report, Intel Corporation, 2012.

[15] Laurence W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. PhD thesis, EECS Department, University of California, Berkeley, 1975.

[16] MichałRewieński. A perspective on fast-spice simulation technology. In Peng Li, Lus Miguel Silveira, and Peter Feldmann, editors, *Simulation and Verification of Electronic and Biological Systems*, pages 23–42. Springer, 2011.

[17] D. Rodopoulos, A. Papanikolaou, F. Catthoor, and D. Soudris. Software mitigation of transient errors on the single-chip cloud computer. In *IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, 2012.

[18] Dimitrios Rodopoulos, Antonis Papanikolaou, Francky Catthoor, and Dimitrios Soudris. Hybrid (hw-sw) mitigation of transient errors on the data plane of the single-chip cloud computer. *IEEE Transactions on VLSI Systems (under review)*, 2012.

[19] A. Sharma, A. Papanikolaou, and E. S. Manolakos. Accelerating all-to-all protein structures comparison with tmalign using a noc many-cores processor architecture. *12th IEEE International Workshop on High Performance Computational Biology (accepted)*, 2013.

[20] Sander Stuijk. Design and implementation of a jpeg decoder. Technical report, Technische Universiteit Eindhoven, December 2011.

[21] Jakob Vesterstrøm and William R. Taylor. Flexible secondary structure based protein structure comparison applied to the detection of circular permutation. *Journal of Computational Biology*, 13(1):43–63, 2006.

[22] Qiaoyan Yu and P. Ampadu. Transient and permanent error co-management method for reliable networks-on-chip. In *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, pages 145 –154, may 2010.

[23] Chris I. De Zeeuw, Casper. C. Hoogenraab, S. K. E. Koekkoeka, Tom J. H. Ruigrok, Niels Galjart, and John I. Simpson. Microcircuitry and function of the inferior olive. *Trends in Neurosciences*, 21(9):391–400, 1998.

[24] Chong Zhao, Xiaoliang Bai, and Sujit Dey. Evaluating transient error effects in digital nanometer circuits. *Reliability, IEEE Transactions on*, 56(3):381 –391, sept. 2007.