Scientific Report

First name / Family name	Malik Khan
Nationality	Pakistani
Name of the *Host Organisation*	NTNU, Trondheim, Norway
First Name / family name of the *Scientific Coordinator*	Anne Elster
Period of the fellowship	01/11/2013 to 31/10/2014

I – SCIENTIFIC ACTIVITY DURING YOUR FELLOWSHIP

1. **Improvement of Real-time behavior of the Snow Simulation application:**

The Snow Simulation application, constructed at NTNU over a period of 5-6 years, utilizes the parallel computing powers offered by modern GPUs. The application uses SOR solver for calculating
the wind field and was originally implemented on the pre-Fermi series of NVIDIA GPUs, using CUDA. Currently, the application uses the modern Kepler GPUs from Nvidia, with latest CUDA versions. This means that the application performance can receive a boost by the use of new architecture and new software and programming features. The snow simulation application consists of many highly parallel algorithms and method implementations, which are suitable for getting benefit from throughput oriented architectures like modern GPUs.

I worked with the snow simulation application, trying to improve its real-time behavior. For such an application, the visible effects of improvement in performance are generally judged by the real-time simulation experience, made possible with higher frame rates. To be able to achieve this objective, I collaborated with a couple of other students in a team to identify opportunities for improvement in overlapping the communication and computation by the use of streams in CUDA implementation of the application. This was followed by the related task of ensuring asynchronous compute and rendering capability of the simulator. group to try and improve the real-time behavior of the snow simulation application developed over the years in this lab. The other main focus was to optimize the movement of data structures from the compute GPUs to the rendering GPUs. In this regard, a detailed study was done on the kind of data structures communicated across from the compute GPUs to the renderer GPUs, to figure out some of the data structures which were being unnecessary moved around and some which were not needed for the purpose of simulation.

The need for a good simulation is satisfied with only a limited number of data structures defining a limited set of data. Some of the detailed data does not improve the simulation visibly, but adds a lot of processing and communication overhead. These limitations were optimized in our study we were able to get some improvement on the frame-rates.

We are working on writing a journal paper for the Snow Simulation application. I

collaborated with another Postdoc in writing about the different aspects of snow simulation application, where I concentrated on the part where the application uses GPUs to improve performance. This potential publication will serve as a reference point for the snow simulation application, as it has seen many iterations of development and improvements without much written material that explains its workings, wholistically. There is some pending work on the mathematical modeling of the application, which will complete and hopefully improved performance of the application, completing the picture for the purpose of a publication.

2. **Parameterization of Heterogeneous Systems:**

The landscape of multicore/ manycore architectures is changing rapidly which is presenting new challenges for the application developers. Additionally, the systems having 2 or more than two devices having multi/manycores have become a norm in recent times. There are a lot of devices capable of providing high-performance through their immense computing capabilities and they are now interacting with each other. For example, multi-core CPUs now are augmented with one or more than one modern many-core GPUs or *number-crunching cores* in building powerful systems. This kind of "heterogeneous" architectures present tremendous computing capacity and theoretical peaks ideal to achieve high-performance. However, harnessing that power requires deep understanding of all the different architectures and their possible interaction mechanisms.

Developing applications for such complex architectures is a daunting task in itself, however, it is made exceptionally difficult if the design of the target system is expected to change relatively quickly, in terms of number and nature of different devices. It means that the programming techniques and the optimizations considered for a certain architecture may or may not work for even the modified version of the same architecture. This makes the construction of efficient libraries, scientific codes and HPC applications that are general enough and remain relevant for a reasonable amount of time, a challenge. A collection of techniques known as "auto-tuning techniques" have emerged to address this challenge. However, there are two ways we can go about getting to the best results for a certain task on a target architecture. One way is to arrive at a collection of different but equal and correct implementations of computations, which can then be all evaluated to arrive at a high-performing version of the computation. This process, called *empirical*

auto-tuning, is slow but provides the maximum performance only depending on how detailed is the process of creating a collection of implementations. Such frameworks are often difficult to maintain through the generations of similar architectures and need tuning for the new target architecture. Another approach can be used to *predict* the performance of a specific computation, by using some combination of predefined parameter values and constraints, defined as a predictor model. The main difference between these two approaches, among others, is that empirical autotuning returns a definite and single best performing computation variation, whereas predictor may identify an area of a certain size in a search space of computation variations, which can then be evaluated to find the best variation. It must be noted that if the empirical autotuning process is augmented with an efficient search-space management mechanism it can improve upon its speed of producing results while on the other hand if model for the predictor is sound enough, it can achieve speed and accuracy at the same time.

Our motivation. with this idea, is to use empirical evaluation/auto-tuning and predictive methods to parameterize the heterogeneous architectures using multi-core CPUs and multiple many-core GPUs, in a way that our model may be able to provide the application developers key information on improving the performance of their applications. There are several factors that affect the performance that can be achieved in a system with heterogeneous system architecture. The partitioning of the computations on such architectures, needs to be done in a way that maximum resources are utilized to achieve maximum performance through maximum utilization of the available resources. A minor variation in the computation decomposition parameter values can lead to significant performance changes.

An important future of this particular research work is in the proposed Cloud-Lightning project funded by the EU. The most important factor of this project, in our context, is the feature “Code for one, Code for all”, which really means that same code should be able to run on multiple architectures in the heterogeneous systems.

Research At INRIA (RUNTIME LAB): In the context of our idea with parameterization of heterogenous systems, one important aspect is to be able to run on multiple platforms which include, MIC, GPUs, Parrallela boads etc. With this context, the collaboration we were able to do at INRIA, during my REP visit, is very helpful. The importance of this work is that it integrates frameworks that are developed at two different levels of the

application execution stack, the code compilation/ generation and the runtime.

Motivation: Many researchers have explored the areas of code optimizations, both at the compiler level and at the runtime level. We have the examples of CUDA-CHILL and STARPU, which are frameworks developed in the respective levels by the participants of this study. We strive to put an additional dimension to both the individual framework. For CUDA-CHILL, it will be an extra dimension of being able to run its generated code on multiple many/multi-core devices in a heterogeneous system. For STARPU, it will simply be the availability of a set of computations, which are autotuned and optimized for high performance for a certain computation, ready to be deployed. This will help STARPU in the execution of a computation, as it does not now have to try low-peforming tasks. We intend to use the output of CUDA-CHILL as an input to STARPU and show a proof of concept of the integration. We use SGEMM as the simple example for our prototype. No particular performance improvement was recorded as part of this short study, but we believe that with a wider range of computations and extensive set of experiments, we can prove the benefits of such an arrangement.

Background and Implementation : StarPU is a task programming library for hybrid architectures. The application provides algorithms and constraints for CPU/GPU implementations of tasks and a graph of tasks, using either the StarPU's high level GCC plugin pragmas or StarPU's rich C API. StarPU handles run-time concerns task dependencies, optimized heterogeneous scheduling, optimized data transfers and replication between main memory and discrete memories, optimized cluster communications. Rather than handling low-level issues, programmers can concentrate on algorithmic concerns.

CUDA-CHILL is a compiler framework which is designed to support autotuning, which employs empirical techniques to evaluate a set of alternative mappings of computation kernels and select the mapping that obtains the best performance. We feel that the SGEMM example serves as a prototype of the integrated working of CUDA-CHILL and STARPU. However, the goals were not to compare against CUBLAS's SGEMM which has another advanced version that has shown to perform way better than any other code generation software's result. However, for us the important thing is that we have a framework in which we can generate optimized code for problems which do not have any high-performing hand-tuned library version available. This work will go a long way in

achieving that purpose.

We are working on transforming the work with STARPU and CUDA-CHILL in a publication for the conference with showing more scientific computations improving performance from the use of autotuning based compiler framework and a powerful runtime system.

Implementation of FTDT on a Distributed Multi-GPU system: Finite-Difference Time-Domain (FDTD) is a popular technique for modeling computational electrodynamics, and is used within many research areas, such as the development of antennas, ultrasound imaging, and seismic wave propagation. Simulating large domains can however be very compute and memory demanding, which has motivated the use of cluster computing, and lately also the use of Graphical Processing Units (GPUs). I have co-supervised the work completed as a master thesis in the summer of 2014. The implementation developed in this work, uses a decomposition approach as opposed to scheduling, which allows for larger domains to be divided among multiple execution units. It supports the use of both a CPU and several CUDA capable GPUs on a single system, in addition to multi-node execution through the use of the Message Passing Interface (MPI). A discussion of the differences between the CUDA capable GPU architectures, and how they affect the performance of the FDTD algorithm, is also included. The results shows a performance increase of 66% when simulating large domains on two GPUs, compared to a single GPU. Using the CPU in addition to one or two fast GPUs is shown to give a slight improvement, but the main advantage is the possibility to simulate larger domains. Results from multi-node executions are also included, but they refer to meager performance due to limitations of a 100 Mbit/s Ethernet used to connect nodes.

We have put together the research work in a paper submitted to IPDPS2015, which is a prestigious conference in the area. The work presented in this paper includes a working FDTD decomposition implementation, that can be executed on a cluster of heterogeneous systems with a multi-core CPU, and one or several CUDA capable GPUs. It is also written with the intention that it should be easily extend-able to work with non-CUDA capable GPUs. This implementation is an improved and novel approach to implement FDTD, based on the work by Skomedal [1] and Andersson[2].

3. **Miscellaneous Activities:**

a) In our own time, I have worked with my advisor to write a CUDA cookbook for intermediate to advanced users based on codes developed at the HPC Lab by the students of Dr. Anne Elster over the years. We have finished almost half of the work on the book chapters.

b) I have worked with multiple students, both PhD and Master level, in multiple projects to collaborate and guide CUDA/ compiler specific features in their respective applications.

II – PUBLICATION(S) DURING YOUR FELLOWSHIP

1. *Parallelizing FTDT for Distributed Heterogeneous Multi-GPU Systems, Eirik Myklebost, Andreas Skomedal, Malik M. Khan, Anne C. Elster,* 29th IEEE International Parallel & Distributed Processing Symposium 2015, (IPDPS-2015) .
2. Integration of STARPU and CUDA_ChiLL, Project Technical Report submitted to supervisors at INRIA and NTNU.

III – ATTENDED SEMINARS, WORKHOPS, CONFERENCES

1. SICS Software Week 7-9 October 2014, Stockholm, Sweden. (Coinciding with REP to SICS).

IV – RESEARCH EXCHANGE PROGRAMME (REP)

1. SICS, Sweden. (October 6th - October 10th)
2. INRIA, France. (October 26th – October 31st)