Master's Thesis Plan

Kristian Hartikainen (222956)
kristian.hartikainen@aalto.fi

August 21, 2015

1 Summary

In this thesis, we hope to understand the applicability of network processing units and event driven programming models in distributed stream computation context. We believe that, by distributing the computation and using task based parallel programming models, both the programmability and the efficiency of such systems can be improved. We hope understand the load-balancing options, that is, mainly the communication latencies and mechanisms, of such systems.

We will measure the performance of a multi-unit network processing system, and build a simulation model based on those measurements. Especially interesting are the delays in the processing pipeline and memory latencies of the system. The simulation model is used to further understand limitations of such a system and its suitability as stream computing platform.

The modeling will be done using Performance Simulation Environment (PSE), based on the values gathered from various measurements. The measurement setup consists of eight 32 MIPS core network processing units, connected to each other over an ethernet switch.

The thesis is done under the Embedded Systems Group in Aalto University with connections to the ParallaX research project. The actual starting point for this thesis is defined by the recent bachelor's theses (Kiljunen, Teemaa) and master's theses (Hanhirova, Saarinen, Rasa), written in the same group.

2 Background

Background for this thesis builds on several aspects. One, the explosive growth of data produced in today's world has changed the nature of computation from traditional Von Neumann type computing to stream computing. At the same time, due to the limitations of hardware manufacturing, the pursuit of performance gains in computing has led to parallelizing computation and use of heterogeneous multi processor hardware. Developing parallel applications has

turned out to be extremely complex, and thus several new methods, such as task based programming, have been presented to aid the problem.

2.1 Hardware Development

For a long time, the performance gains in computing relied on faster and faster CPU clock speeds, optimization of sequential programs, and larger cache sizes. However, the growth in CPU clock speeds has withered due to the physical constrains of chip manufacturing. Heterogeneous MPSoC devices have become an important part of today's efficient computing systems. They have also brought out new challenges in the software development and requirements for the programming models.

2.2 Parallel Programs

The performance gains in today's computing rely heavily on parallelism and concurrency of programs. The parallel computation itself, especially on heterogeneous hardware, is a hard problem due to the non-determinism, communication (memory) delays and complicated memory management. Most widely adopted parallel programming approach, threads, are becoming inefficient and thus new approaches, such as task parallelism, have been developed. This raises the question of how to balance the computation between the computing resources to maximize the utilization.

2.3 Stream Computing

In traditional computing architectures, such as Von Neumann architecture, the computation is based on reading, processing and writing the data located in system's memory. These architectures have their limitations and thus both the academia and the industry are researching alternative ways of computing. One of the alternatives is stream computing, which is present especially in Internet of Things (IoT) applications. In stream computing, the computation is done for data streams flowing through the system, instead of being altered in the memory.

2.4 Distributed Computing

Distributed computing has been used for a long time, especially in scientific computing. However, distributing the stream computation is non-trivial, due to its real time nature. Traditional computing clusters can be virtualized, alleviating the distribution of the computation, by using message passing abstractions such as Message Passing Interface (MPI). However, the real time nature of stream computation makes its distribution difficult. Data processed in stream computation is often dynamic: the stream characteristics are not known in advance,

and the data volume and the computation requirements can change at any point of the computation. Load-balancing of the computation has to be done on the fly, and thus understanding the communication delays and latencies between the units are important.

3 Objectives

The objective of this thesis is to understand the load-balancing options of a dynamic workload of a stream computing system combining both, inter-unit (shared memory) and intra-unit (distributed memory), level parallelism. This objective includes constructing an experiment, that is, measuring and modeling such a system.

We hope to understand how the implemented dynamic load-balancing mechanism performs under the changing volume and requirements of a dynamic data streams, and furthermore, gain directions for further development of such a system. The experiment should reflect a real world situation of a dynamic stream processing, for the results being applicable in the further research in the field.

When distributing the computation between multiple NPU's, the communication latencies play crucial role. The latencies consist essentially from two aspects. Firstly, the speed, that is latency and throughput, of the units. We will statistically determine these for our test system. Secondly, we want to understand how much implicit context information needs to be transferred along with the data, to carry on the computation on another unit. This is clearly dependent on the method used for the distribution and communication.

4 Methods

We will propose a distribution and load-balancing mechanism for event based stream computing system. The mechanisms will be based on existing methods found on literature, and resulting system's behaviour will be demonstrated with an experiment.

The experiment consists of two major parts. First, building a simulation model of a stream computing system, and second, instrumenting and measuring an example hardware. The simulation model will then be used to further demonstrate and understand the possibilities of distributing and load-balancing the stream computation between network processing units.

The actual experiment will not include a full implementation of either event based programming environment or message passing system, but the needed characteristics will be measured by experimenting with isolated parts of those systems.

The results of the experiments will be evaluated both qualitatively and quantitatively. The experiment will be implemented on a system consisting of eight

32 MIPS core network processing units. The example hardware system in use is sort of a black box, and thus our focus will be heavily on the quantitative results and simulation.

More details about the actual methods and experiment can be found from the experiment plan.

5 Material and Resources

5.1 Measurement Hardware and Software

The hardware and software needed to carry out the measurements are listed below.

- **Cavium OCTEON II**: MPSoC blade with a special purpose hardware accelerators and 32 MIPS cores.
- **Cavium OCTEON SDK**: A set of software tools to build applications that run on the OCTEON II blades.
- **Multi-core Processor Architecture and Communication Benchmark Suite (MPAC)**: A benchmarking library for performance characterization of multi-core systems.
- **Wireshark**: An open source network packet analysis tool.

All the measurements will be done on the Cavium OCTEON II blades. OCTEON SDK provides several packet processing examples, which will be used as a base for the actual measurements. The examples include a traffic generator application which will be one potential option, together with Wireshark, when generating test workloads. The SDK provides convenient access to the hardware counters which will be used in the timing.

5.2 Modeling material

The system will be modeled using Performation Simulation Environment (PSE). PSE is a heterogeneous modeling and discrete event simulation environment, based on a Queuing Network Simulation (QSE) tool built at Aalto University. It provides a toolset for analyzing hardware and software co-scheduled manycore systems.

5.3 Reference Material

The reference material, listed below, is not directly related to the measurements or the modeling of the system, but act as the base for understanding the different solutions for the challenges.

- **Open Event-Machine (OpenEM)**: Architectural abstraction and framework of event-driven multicore computation. OpenEM enables efficient writing of dynamically load-balanced multicore applications with a very low overhead run-to-completion principle. OpenEM can be implemented both, on bare hardware as well as on top of an operating system.
- **Message Passing Interface (MPI)**: A specification and de facto standard for a set of message passing libraries. MPI standard defines the syntax and semantics for functions that are useful in a portable message-passing and distributed memory programs.
- **Futures and promises**: Futures and promises are constructs that are used communication and synchronization in concurrent programming.
- **Open Multi-Processing (OpenMP)**: Multi-platform, shared memory multiprocessing API.
- **Intel Data Plane Development Kit (DPDK)**: A set of packet prosessing libraries and drivers.

OpenEM will work as a base programming framework in our study. To understand how OpenEM could be extended to support distributed computation, we will investigate already widely used communication methods such as MPI and future-promise. Intel DPDK can be studied to better understand the data plane functions and data I/O.

5.4 Other resources

Vesa Hirvisalo will work as an advisor for this thesis. Jussi Hanhirova is working on his doctoral thesis on the same area of research. Risto Vuorio is working on his Master's thesis on the similar subject. The supervisor is still yet to be decided.

6 Challenges

We have spent lots of time trying to figure out the problems with the experiment hardware. The OCTEON II blades drop the received packets and deadlock unexpectedly. We are waiting to get answers for these issues in a week or two.

I'm still waiting for the official bachelor's degree graduation, needed to get the master level study plan approved. The graduation date is 31st of August.

7 Work Plan

The work plan for the thesis consists of five major steps, that can be summarized as follows:

0. Study the material needed to create the experiment
1. Build a simulation model of the system
2. Measure the needed attributes from the example hardware and software
3. Plug in the attributes to the simulation model
4. Run simulations
5. Validate that the simulations correspond the hardware system

In step 0 we have studied existing message passing systems (MPI, future and promise constructs), event based programming models (OpenEM) and the example network processing unit. We are currently working on step 1, during which we will build a simulation model with PSE. The high level model has already been built, however the packet scheduling and applications still need to be improved.

In step 2, we will instrument and measure the needed attributes from the example hardware system. Main interest will be on the communication and memory latencies. We will begin the measurements right after we overcome the packet dropping and deadlocking problems of the system.

Then, in step 3, the measurement attributes will be plugged in the simulation model. Finally, the simulations will be run and further tests are done to validate correct behaviour of the simulation model. The experiment will be reiterated, while need be, and the resources are sufficient, starting from step 1.

8 Schedule

Week 21	First iteration of the thesis plan. Studied the material related to simulation and task parallelism.
Week 23	Studied message passing abstractions, mainly MPI. Ran examples to get familiar with the basic structure and send/receive functions. Quick overview of the MPI I/O. Presented these in the study circle. Also spent some time figuring out how the experiment NPU's work.
Week 24	Studied OpenEM. We setup the DPDK based OpenEM implementation together with Risto. However we ran into troubles with the I/O hardware. Also shortly studied the packet_multi_stage.c example from the OpenEM reference implementation and represented it in the study circle.
Week 25	Studied the event and execution object constructs in OpenEM. Got an overview about the context of OpenEM queues and execution objects. Still room for digging deeper in the subject.
Week 26	Spent time figuring out the packet loss problems of the NPU.
Week 27	Still debugging the NPU. Figured out how to access the NPU's performance counters through the SDK. Coded initial examples for those.
Week 28	Measured the simple loopback times through the NPU. Generated data from external machine and measured the packets' transmit and receive times, as the NPU worked as a simple passthrough. Did 100 measurements for several different packet sizes. The NPU still had problems with the packet dropping so all the tests were done using exactly 32 packets. Also figured out the performance counter problem. The performance counters do work in simple-exec mode, and should be activated on Linux-mode.
Week 29	Debugging the NPU. Simple-exec apps loaded from the flash work, meaning that the problems are in the hardware initialization of the SDK.
Week 30	Vacation
Week 31	Managed to get some of the packets through the NPU. Still, some of the packets disappear on their way through. Coded some initial instrumentation code. The NPU's turned out to be challenging to measure due to packet dropping and deadlocking. Coded shell scripts for faster reboot and setup of the measurement system.
Week 32	Studied the switching and routing to better understand what we need to model. Researched the NPU's input and output phases as a background for the simulation model. Started modeling the system with PSE. Simple placeholders for each processing phase in the NPU.
Week 33	Continued tweaking the PSE model. First working model ready. Complete overhaul of the model to better model the scheduling and memory of the system.
Week 34	Rewrote the thesis plan and experiment plan. Continue PSE modeling to get the current version to compile properly.

Week 35	Enhance the PSE model based on the feedback. More accurate model of the scheduling and memory. Figure out how to continue.
Week 36	If we get the NPU's SDK patched, measure the values needed for the model.

9 Dissemination

This thesis is related to the ParallaX research project. The thesis hopefully gives some background to Jussi Hanhirova's doctoral thesis.

Appendices

A Thesis structure

The rough of the thesis structure is presented below. The bullets under the subsections are general ideas of what should be included in each chapter, not subsubsections.

- Abstract
- Preface
- Contents
- Abbreviations

1. Introduction

   Research Problem

   - Distributing stream computing, how to load-balance the computation across the nodes?
   - How long delays occur if the computation is moved from node to another?

   Contributions

   Structure

2. Background

   Parallel Computing

   - Free lunch is over. Why do we parallelize computing in the first place?
   - MPSoC devices
   - From threads to tasks. Why do we need new parallel programming methods such as OpenEM?

   Stream Computing

   - Internet of Things, fog computing.
   - From Von Neumann to Streams. Window-type view of the data. Real-time.

   Distributed Computing

   - Distributed computing is widely used e.g. in scientific computing
   - Message passing

I/O Virtualization

- ??

3. Distributed, Parallel Stream Computing

   Open Event-Machine

   - Event driven parallel programming model
   - Currently no support for distributed computation

   The context of computation

   - What is actually needed to move the computation between the nodes?

   Hardware requirements

   - What is required from the hardware for the distributed computing to be feasible
   - Especially input and output latencies are important

4. Measuring the system

   Measurement Setup

   - NPU nodes
   - Instrumenting the applications.

   Communication Latencies

   - Statistical behaviour of input and output under varying workload
   - Simple tests with message passing. MPI or future-promise

   Memory Latencies

   Results

5. Simulation Model

   PSE - Performance Simulation Environment

   Workload model

   Hardware model

   - Scheduler, input/output, NPU nodes

   Software model

   - Scheduler, OpenEM-type application model

6. Discussion

   Challenge

   Discoveries

   Related Work

   Future Work

7. Conclusions

B Key Literature

[1] Jerry Banks and John S. Carson II. "Introduction to Discrete-event Simulation". In: *Proceedings of the 18th Conference on Winter Simulation*. WSC '86. Washington, D.C., USA: ACM, 1986, pp. 17–23.

[2] Flavio Bonomi et al. "Fog Computing and Its Role in the Internet of Things". In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. Helsinki, Finland: ACM, 2012, pp. 13–16.

[3] Sanjay Chatterjee et al. "Integrating Asynchronous Task Parallelism with MPI". In: *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013*. 2013, pp. 712–725.

[4] Mihai Dobrescu et al. "RouteBricks: Exploiting Parallelism to Scale Software Routers". In: *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*. SOSP '09. Big Sky, Montana, USA: ACM, 2009, pp. 15–28.

[5] N. Egi et al. "Improved Forwarding Architecture and Resource Management for Multi-Core Software Routers". In: *Network and Parallel Computing, 2009. NPC '09. Sixth IFIP International Conference on*. 2009, pp. 117–124.

[6] N. Egi et al. *Understanding the Packet Processing Capabilities of Multi-Core Servers*. Internet Draft. EPFL, 2009.

[7] Richard M. Fujimoto. "Parallel Discrete Event Simulation". In: *Commun. ACM* 33.10 (Oct. 1990), pp. 30–53.

[8] Jussi Hanhirova. "Performance analysis of hardware accelerated scheduling". `https://aaltodoc.aalto.fi/handle/123456789/14571/`. MA thesis. Department of Computer Science, Engineering, Aalto University School of Science, and Technology, Espoo, Finland, 2014.

[9] Simon Hauger et al. "Packet Processing at 100 Gbps and Beyond - Challenges and Perspectives". In: *Photonic Networks, 2009 ITG Symposium on*. 2009, pp. 1–10.

[10] Edward A. Lee. "The Problem with Threads". In: *Computer* 39.5 (May 2006), pp. 33–42.

[11] Joonas Saarinen. "Parallel Processing of Vehicle Telemetric Data". `https://aaltodoc.aalto.fi/handle/123456789/14570/`. MA thesis. Department of Computer Science, Engineering, Aalto University School of Science, and Technology, Espoo, Finland, 2014.

[12] Herb Sutter. "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software". In: *Dr. Dobb's Journal* 30.3 (2005).

[13] N. Varis and J. Manner. "Performance of a software switch". In: *High Performance Switching and Routing (HPSR), 2011 IEEE 12th International Conference on*. 2011, pp. 256–263.

[14] Reinhard Wilhelm et al. "The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools". In: *ACM Transactions on Embedded Computing Systems* (2008), pp. 1–53.