Master's Thesis Plan

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

June 29, 2015

1 Summary

This document is meant to cover all the general stuff that needs to be considered before and during writing my master's thesis. In addition to the technical planning of the thesis, this plan also includes the general matters that might affect the thesis writing.

In the background section I present the outline for the larger research area where the thesis sets in. In the followed three chapters I present the goals for this thesis, the methods used to achieve those goals, and the material to be used. In the last chapters, I will go through the general matters possibly affecting the thesis writing, such as the material and resources in use, general challenges, the actual work plan, and the dissemination. In the appendix I will also present a scaffolding for the actual thesis structure, and the key literature for the thesis.

2 Background

Parallel computation and heterogeneous MPSoC devices are important parts of today's efficient computing systems. However, they also bring out new challenges in the software development and requirements for the programming models.

The scalability of the parallel computation itself, especially on heterogeneous hardware, is a hard problem due to the non-determinism of parallel computation, communication delays and complicated memory management. 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.

Hardware threads are static, meaning that they cannot adapt to the dynamic changes of the workload. Also, the hardware threads cannot be migrated between the computing nodes, and thus binding the computation to the hardware threads forces the computation to be done on the specific computing node. This is why the traditionally used parallel programming models, that are based on thread parallelism, don't meet needs of streaming computing of dynamically changing workflow.

Traditional computing clusters and inter-node computations can be virtualized by using message passing abstractions such as MPI. However the real time nature of the streaming computation brings on problems to this kind of inter-node computation. The research problem of this thesis is the balancing and transferring of the dynamic workload between multiple stream computing nodes.

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 for the Embedded Systems Group.

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-node (shared memory) and intra-node (distributed memory), level parallelism. This objective includes constructing an experiment, that is, planning and designing, implementing, and analyzing an example of such a load-balancing 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.

4 Methods

The load-balancing of the dynamic data-stream will be transformed into a task scheduling problem. This will be achieved by extending OpenEM type of processing queue mechanism by MPI's inter-node task scheduling. The actual experiment will not include a full implementation of either OpenEM or MPI, but their 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 8 Cavium OCTEON II blades, each consisting of 32 MIPS cores. Because the hardware system is sort of a black box, our focus will be heavily on the quantitative results and simulation. Program analysis will be done both statically and dynamically. The main simulation tool will be Performation Simulation Environment (PSE). The analysis results will be compared to the actual measurements of the experiment system.

5 Material and 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. Hardware and frameworks to be used in the thesis is listed below.

- Cavium OCTEON II: MPSoC blade with aspecial purpose hardware accelerators and 32 MIPS cores.
- Intel DPDK: a set of packet prosessing libraries and drivers.
- MPI (Message Passing Interface): De-facto standard for message-passing system to write portable message-passing programs.
- OpenEM (Open Event-Machine): Architectural abstraction and framework of event-driven multicore computation.
- OpenMP (Open Multi-Processing): Multi-platform, Shared memory multiprocessing API, a thread-parallelism implementation.
- PSE (Performance Simulation Environment): A toolset for analyzing hardware and software co-scheduled manycore systems.

5.1 Open Event Machine

Open Event Machine (OpenEM), originally developed by Nokia Solutions and Networks (NSN) is a framework and programming model for a task (event) based processing on a multicore platforms. It 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 harware as well as on top of an operating system.

OpenEM applications are constructed from three main parts, namely execution objects (EO), events, and event queues. Events are the main communication blocks, the data, of OpenEM. Execution object is a run-to-completion object, or a function, that is run using the event as the argument. Event queues are used to store the events between execution objects. The scheduling order of the events, from the queues to the EO's, are handled by the OpenEM scheduler. Niether the application nor the EO's are tied to a specific core. The dynamic load-balancing is achieved by processing EO's with the events on any available core at the time.

The user is able to guide the scheduling of the events through the EO's coremask parameters, and queue's priority and type parameters. The coremask parameter for the EO's specifies which cores each the EO is allowed to be run on (no restrictions by default). The queue type can be atomic, parallel, or parallel ordered, and the priority order of the queues can be specified by the priority parameter. In addition to the user specified parameters, the scheduling is guided by the locality of the EO, meaning that tasks to be processed on the same EO are more likely to be scheduled consecutively in cases where the other priorities are equal. Also, the task order is maintained through the queue in a FIFO principle.

5.2 Message Passing Interface

Message Passing Interface (MPI) is 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.

5.3 Performation Simulation Environment

Performation Simulation Environment (PSE) is a heterogeneous modeling and simulation environment, based on a Queuing Network Simulation (QSE) tool built at Aalto (TKK).

6 Challenges

The Cavium OCTEON II hardware used in the experiments is going to be hard to analyze.

I haven't officially received my Bachelor's degree, since I'm one course short from it. The official degree is needed to get the Master's thesis topic approved by the school.

However, this shouldn't be a problem, as I have discussed with the student counselor Elsa Kivi-Koskinen, and the course is under progress. I have done the course exam already, and right after I finish the course project, it's done from my part. Then the Bachelor's degree still have to be approved by the school council.

7 Work Plan

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

0. Study the load-balancing mechanisms in OpenMP and MPI systems
1. Design a corresponding mechanism for our experiment hardware
2. Plan how to implement and test such a mechanism for our experiment hardware

3. Implement a bare load-balancing mechanism for the experiment
4. Run the experiment using the implemented mechanism
5. Model the experiment using a simulator

In step 0 we will study the load-balancing mechanisms in OpenMP and MPI systems. In step 1 we will design how to build such a asynchronous task parallelism with distributed-memory parallelism system on top of OpenEM, instead of OpenMP. OpenEM at the moment does not include an MPI or inter-node support.

In step 2, we will plan how to isolate a suitable part of the system for our experiments needs. The time resources for this thesis are limited, and thus the implementation of a complete model is unfeasible. We will isolate a part of the model, such that we can implement and test the performance of its bare load-balancing mechanism.

Next, we will implement the load-balancing mechanism of the model. That is, we will implement a very bare queue mechanism, such that the workdload can be scheduled between the computing nodes, without using OpenEM, OpenMP or MPI. After that, we will test and measure the implementation. Finally, we will model the mechanism using a simulator software, and interpret the results. The experiment will be reiterated, while need be and the resources are sufficient, starting hopefully from step 2.

8 Schedule

Week 23	Studying of the load-balancing in OpenMP/MPI systems. Figure out how to generate the workload for the experiment. Start modeling the PSE simulation. Figure out the contstraints of Cavium OCTEON II.
Week 24	Continue studies of the load-balancing systems. Designing/planning of the load-balancing mechanism for our needs. Continue modeling.
Week 25	Start the concrete implementation of the experiment.
Week 26	Implementing. Test the hardware counters of Cavium OCTEON II.
Week 27	First implementation of the experiment finished. Vesa and Jussi will be on a vacation and thus those vacation weeks have to be well planned. If analysis and the interpretations of the results are not finished, as is likely, then plan the exact steps for these. Plan for the next iteration.
Week 28	Work on the analysis.
Week 29	Analysis.
Week 30	Interpretation of the results for the experiment.
Week 31	Focus on writing the first iteration of the thesis.
Week 32	Thesis writing. If Vesa and Jussi are back from vacation, evaluate the experiment results. Work on the experiment according to the results.
Week 33	Thesis writing. Work on the experiment according to the results.
Week 34	Thesis writing. Work on the experiment according to the results.
Week 35	Absolute deadline for the first iteration of the experiment.
Week 36	Figure out how to continue.

9 Dissemination

This thesis is related to the ParallaX research project. Also, the thesis hopefully gives some background to Jussi Hanhirova's doctor's thesis.

Appendices

A Thesis structure

- Abstract
- Preface
- Contents
- Abbreviations

1. Introduction
   - Research Problem
   - Contributions
   - Structure
2. Background
   - Heterogeneous systems
   - Intra-node parallelism
   - Inter-node parallelism
3. Analysis of Parallel Programs
   - The goal of the Analysis
   - Tools used
4. Experiments
   - Setup
   - Results
   - Conclusions
5. Discussion
   - Challenge
   - Discoveries
   - Related Work
   - Future Work
6. Conclusions

B Key Literature

Cavium	OCTEON Programmers Guide - The Fundamentals
Sutter, H.	The free lunch is over
Chatterjee & al.	Integrating Asynchronous Task Parallelism with MPI
E.A. Lee.	The Problem with Threads. doi:10.1109/MC.2006.180
Bonomi & al.	Fog computing and its role in the internet of things. doi:10.1145/2342509.2342513
Nokia Solutions and Networks	Open event machine.
Teemaa, Taavi	Rinnakkaisuuden mallit ohjelmointikielissä
Kiljunen, Olli	Tehtävärinnakkaisuus ohjelmoinnissa
Hanhirova, Jussi	Performance analysis of hardware accelerated scheduling
Saarinen, Joonas	Parallel Processing of Vehicle Telemetric Data
Rasa, Marko	
Wilhelm et al.	The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools
Banks et al.	Introduction to Discrete-Event Simulation
Gustavsson et al.	Timing Analysis of Parallel Software Using Abstract Execution
Fujimoto et al.	Parallel Discrete-Event Simulation