Master's Thesis Experiment Plan

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

August 21, 2015

1 Summary

In this experiment, we will build a simulation model and measure a network processing system used for stream computation task. The model will be used to understand the possibilities of distributing and load-balancing the stream computation between NPU-units, and thus the measurements will focus on the communication latencies of the units.

2 Goals

The higher level goal of this experiment is twofold: firstly we will build a simulation model of a stream computing system consisting several modern network processing units and event-based programming models; secondly, we measure an NPU hardware to obtain the metrics needed for the model to answer real-life behaviour of such a system. These two goals are built of several subgoals presented below.

As the simulation model will mainly be used to understand the distribution and load-balancing of the computation, the measurements will be focused on the delays in the processing pipeline of a network processing unit. This means the behaviour of the packet throughput time of the unit under varying workloads. The memory latencies will be measured to amplify the model of the actual packet processing phase of the system.

The accuracy and the modularity of the simulation model are important parts of the experiment. The simulation model should replicate real-world hardware precisely enough so that it can be used for further analysis. The modularity means that the important parts of the system, especially the packet scheduling and software applications, can be remodeled without special comprehension about the actual simulator.

As we have limited time resources for the experiment and plenty of metrics that can be measured, we will have to carefully plan the measurements so that they promote both of our main goals.

The system will be modeled by using the in-house PSE simulation tool. One of the minor goals is to gain further understanding about the tool, and further develop it.

3 Modeling methods & materials

The hardware setup that we are interested in, consists of eight 32 MIPS core network processing units, or blades. Due to the black-box-like nature of the setup, the focus of the experiment will be in the simulation of the system. We will build a realistic simulation model of our hardware running OpenEM application for video processing. The parameters, mainly delays, needed for the simulation model will be gathered from several smaller measurement experiments done with the actual hardware setup.

The accuracy of the model must be precise enough so that further studies of the system behaviour can be conducted reliably by the simulation. We will validate and verify the results by comparing the behaviour of the simulation to the real life application.

3.1 Hardware Setup

The blades are connected to each other over ethernet switch. The data packet flow inside each blade can be divided into three main phases namely input phase, sso and core processing phase, and output phase.

Input phase:

1. The Receive Port (RX) forwards the packet to the Input Packet Data (IPD) Unit. IPD processes the packet together with Packet Input Processor (PIP), e.g. parse packets and the fields required for the Work Queue Entry (WQE).
2. Allocate WQE Buffer and Packet Data Buffer from the Free Pool Allocator (FPA) Unit.
3. Write WQE fields and packet data to the WQE Buffer and Packet Data Buffer respectively.
4. Add WQE pointer to the Schedule Synchronization Order (SSO) Unit.

SSO and Core Processing:

1. A processing core requests WQE pointer from the SSO unit. WQE contains pointer to the packet data.
2. The packet data is processed by the core, the data is read and written to L2/DRAM.

3. When the processing is finished, the core sends the Packet Data Buffer pointer and the data offset to the Packet Output Queue in the Packet Output (PKO) Unit.

4. Free the WQE Buffer back to the FPA.

Output phase:

1. PKO DMA's the Packet Data Buffer in to its own memory.

2. PKO does the needed post-processing, e.g. adding checksums, and then sends the packet data to Output Port (TX). Optionally notify the core when the packet was sent.

3. PKO free's the Packet Data Buffer back to FPA.

The delays in the input and output phases are most interesting, and challenging at the same time, for us to understand the inter-unit load-balancing. The input and output phases are mostly hardware accelerated which makes the measurements challenging. The measurements will be varied to be able to statistically model the behaviour. Also, the intra-unit core-to-core delays are interesting.

3.2 OpenEM

Understanding the OpenEM's computation context is crucial when creating the simulation model. When we move the computation of from one blade, we need to know how much, and what kind of, data needs to be transferred in order to continue the computation on another blade.

3.3 Inter-blade Load Balancing

Once we undestand the data needed for the inter-blade load-balancing, we need to understand the different options to model and implement the actual message passing between the blades. At the moment MPI- and future-promise-style constructs both seem valid options. Only those parts of the message passing will be implemented that are needed to measure the metrics for the model.

3.4 Performance Simulation Environment

The system will be modeled using Performance Simulation Environment (PSE). The three main parts of PSE model are workload model, software model and hardware model. The workload model consists of actions which are invoked according to user specified rules, for example probability distributions. These actions are passed to the software model to be processed. Software model utilizes the hardware resources described in the hardware model.

Both, the hardware and software models, are amplified by the parameters gained from the measurements of the system.

4 Measurements

Measurements are needed to gather the parameters to configure the simulation system. As we are interested especially in the inter-unit level challenges of the stream computation, the communication delays are the main focus of our measurements. Also, the intra-unit delays, mainly memory latencies inside the units, are interesting. Blades present the instances of such a network processing units.

The measurements with the blades will be run on top of Linux, which makes the measurements and workload handling easier. However, this also creates overhead in the measurements, and has to be taken into account in the results.

4.1 Inter-unit Communication Delays

The processing chain of the inter-unit communication contains several steps, as presented in the previous chapter. We will not able to measure the delays in each of the hardware parts of the communication chain. Thus, the measurements need to be varied to find out the parameters for the underlying statistical behaviour of the phases.

We will measure the total delay in the output phase and the input phase by using the blade hardware counters. CVMX_CORE_PERF_CLK counter returns the clocked cpu cycles. We can determine the communication delay between the blades by sending data packets from blade A to another blade B. We will save the clock cycle once right before the sending the packet from blade A, and again right after the blade B receives the packet. If we send the packets from a blade back to itself through the switch, the absolute timing will be easier to handle. The parameters for the statistical model will be gathered from several repeated measurements, varying the packet size and count each time.

Cavium SDK offers ready-built interface for the use of performance counters. The complete list of per-core counters can be found from the OCTEON Programmer's Guide, p. 6-14.

At the moment, the traffic generation is planned to be done by using external machine. We will create a local area network between the blades and an external packet generator. The packet generator will send packets, using a networking tool (e.g. netcat or mausezahn), to one of the blades, which then forwards the packets to other blades, and finally back to the traffic generator. The communication times could also be measure by using Multi-core Processor Architecture and Communication (MPAC) benchmarking library. Using MPAC, we would measure the loopback time of the packets, starting from the packet generator through one or more of the blades, and then back to the packet generator.

For this setup, the delays in the actual traffic generator might be very large. We will have to measure the latencies of that machine as well.

The passthrough.c example from the Cavium SDK can be used as a base for this experiment. We might also need to implement some simple application that also includes processing and termination of the packets. With a simple OpenEM queue implementations, we can measure the load-balancing overhead when the amount of data transferred exceeds the computation capacity of the blades.

4.2 Intra-unit Delays

We will also need to measure the delays, mainly memory latencies of a single blade. This will be done by running some multi-threaded application, implemented for example with OpenMP, and measure it's performance, for example with MPAC.

5 Modeling

The model will be created by iterative modeling-measurement-configuration approach. The measurements described in the last chapter are needed to build the model. The modeling software, PSE, consist of three main components, which are discussed below.

5.1 Workload model

The workload for the model will be chosen so that it brings out the real-life characteristics of the load-balancing. There is no need to conduct any special measurements for the workload model, we already know how to create network packet data. Technically the parameters, such as the branching statements, of the model are written in the workload model.

5.2 Software model

The software is modeled using OpenEM type, event based, processing where the inter-blade communication and load-balancing is implemented using for example future-promise or MPI type synchronization constructs.

OpenEM execution objects are presented as separate submodels, where the hardware utilization and workload flow depends on the modeled execution object's logic. The different software queue types will be modeled as resource usage nodes.

5.3 Hardware model

The hardware model will be created based on the previously described network processing device. The parameters and the delays gained from the measurements will be used to present the hardware provision utilized by the software model.

6 Post processing

In the post-processing phase, we will validate and verify the created simulation model, and plan the next steps according to those results. The simulations ran with the model has to produce similar effects as the real system under study. The precision of the simulation model will be measured both qualitatively and quantitatively.

After validation and verification, we will either reiterate the measurement and modeling phase, or conclude the experiment and present the results in the thesis.

7 Schedule

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.