Thesis Plan
Streaming Parallelism On A Multicore DSP
Week 22

Risto Vuorio (84525R)
risto.vuorio@aalto.fi

May 22, 2015

Contents

1 Summary

This thesis investigates stream processing on a multicore DSP featuring hardware accelerated scheduling. The research platform is the Texas Instruments implementation of Open Event Machine which utilizes hardware accelerated scheduling through Multicore Navigator.

In this thesis: 1. A realistic application implementing stream processing is created using OpenEM on a Texas Instruments multicore DSP. 2. The application is tested on a instrumented platform to get real world measurements of the execution. 3. Static analysis is used to estimate the execution. 4. Dynamic analysis is conducted on the application to further understand the performance and study the analysability of such applications.

2 Background

This thesis builds upon research conducted in the Embedded Systems Group at Aalto University and the international research community. Risto has already been introduced to some key articles, tools and experiments on the field. Further background investigation will be conducted in parallel with the experiments but mostly after the practical experiments have been completed.

Taavi Teemaa has a good introduction to streaming parallelism in his B.Sc work. Dynamic Data flow on multicore processors needs to be further studied. Programmability of frameworks providing dataflow actors: OpenMP good, RVC-CAL bad

Stream Processing is a programming paradigm that aims to simplify the design of parallel programs. In stream processing computations happen in kernel functions. The kernel functions are applied upon each element of the input stream. Stream Processing is a high level programming model and can be implemented in a multitude of ways. Stream Processing is called uniform if all data is processed by all kernels. (NVIDIA Streaming Multiprocessor references this type of streaming.)
http://en.wikipedia.org/wiki/Stream_processing

Stream Computing is often considered in IoT or Industrial Internet context. In Stream Computing data is not in the memory as in V. Neumann architecture but in a "stream" that passes through the process. A problem or a task involving lots of continuous I/O without necessarily knowing the end of the input is a dataflow computing problem.

Flow, a high-level concept where computations happen when dependencies are updates. For example excel implements kind of flow computation.

Dataflow Programming models a program as a directed graph of data flowing between operations. Operations have explicitly defined inputs and outputs. Operations are carried out as soon as the inputs to that particular operation become valid. The explicit dependencies between the operations make Dataflow

Programming well suited for parallel processing because when the operation becomes ready it can be executed any time on any hardware resource without further need for considering its dependencies.
http://en.wikipedia.org/wiki/Dataflow_programming

Dataflow Architecture is a hardware architecture in which the execution of instructions is based on the availability of input arguments. The execution order of the instructions is thus indeterministic.
http://en.wikipedia.org/wiki/Dataflow_architecture

Dataflow (computing) is a software architecture based on the Flow concept. Writing software in dataflow pattern helps reduce coupling related code in programs, meaning programmers need to write less code to explicitly update values based on their dependencies.
http://en.wikipedia.org/wiki/Dataflow
http://en.wikipedia.org/wiki/Kahn_process_networks

Actor model Actors can be thought of as similar to OOP objects but with special emphasis on parallelism. Anything actors do can be done in parallel with other actors. In actor model all computations are done in actors. Actor model is based on the idea of actors sending messages to each other. As a response to receiving a message an actor can create more actors, send messages, change how to respond to next message etc. Actors can send messages at any time, but the receiver can respond whenever it is ready.
http://en.wikipedia.org/wiki/Actor_model

3 Objectives

The objective of this thesis is to study the utilization of hardware accelerated scheduling on a multicore DSP in handling dynamically changing workloads. An application is implemented which demonstrates efficient dynamic scheduling for stream processing. The application needs to resemble real world applications run on similar devices to ensure the applicability of the results.

For example a DSP faces a situation where it is processing N video streams at time A. At time B there are M video streams. The problem this thesis solves is how to reallocate the computation resources on the DSP dynamically so that the variable amount of streams can be processed efficiently.

4 Method

In this thesis an experiment on implementing and analyzing a stream processing program is conducted. The program is implemented on Texas Instruments (TI) hardware using Open Event Machine (OpenEM). The program will be implemented in a dataflow pattern.

In OpenEM the general problem of locking in concurrent programming is transformed in to a scheduling problem. Whenever two threads need the same resource they are scheduled sequentially so that they don't try access the resource simultaneously.

When OpenEM schedules an event, it allocates the resources for that event and lets it run to completion. The next event utilizing the same resources might be something completely different depending on which events are ready for execution. There are no local buffers tied to certain memory locations, as all resource allocation and event execution is scheduled on the fly. This thesis studies the memory behavior in such dataflow context, through the application implemented and the analysis conducted on it.

The execution of the program will be statically analyzed using a suitable tool. The current candidate for analysing the execution is implemented in a tracing library provided by Texas Instruments.

The execution will be dynamically analyzed by measuring the real system. In addition a simulation model of the system will be constructed. The execution of the simulation model will be monitored. Results from the simulation model and the real system will be compared.

4.1 Construction

An actor based implementation of Sobel filter will be used as base for the workload construction. This implementation will be rebuilt on top of OpenEM preserving the actors but changing the details to match the runtime. The structure of the application construction is presented in the following graph.

```
*************************************************
|     Video processing workload in Actor Model   |
*************************************************
 . . . . . . . . . . . OpenEM . . . . . . . . . . .
*************************************************
|           Texas Instruments Hardware           |
*************************************************
```

Another filter such as Canny will be constructed following a similar pattern. The initial actor based implementation will be constructed in PREESM. The PREESM implementation will be converted to OpenEM.

The two (or more) filters are used to simultaneously process two or more streams on the same DSP. The simple implementations (Actors, PREESM) will be scheduled with hand derived static schedule. The benchmarking setup will be constructed so that the number of streams can be changed dynamically.

The performance of the OpenEM based version will be compared to the statically scheduled version under different load conditions. Dynamically changing numbers of streams are especially interesting for the goals of this thesis. Static loads will be studied to find out the overhead of dynamic scheduling in OpenEM versus the static scheduling.

4.2 Analysis

Before choosing the exact analysis tools to be used, more information needs to be gathered about the tools and the workload to be analyzed. There is some internal interest in using the PSE tool for dynamic analysis and since Jussi Hanhirova has been exploring the tool and the using it to analyze applications for TMS320C6678 it seems like a logical choice on the Dynamic analysis side.

Before execution: static analysis <- WCET (Actor Accelerator Function)

After Execution: dynamic analysis <- scheduling (PSE)

5 Material / Resources

5.1 Workload

The program to be analyzed needs to be similar to real-world programs executed on the platform to get generally useful results. Too simple workload wouldn't sufficiently utilize the hardware accelerated scheduler and therefore wouldn't help us understand the memory behavior of the applications and the capabilities of the analysis tools in analyzing hardware accelerated scheduling on a multicore platform.

Care must be exercised in selection of the workload to not implement a too complex load. Too complex load couldn't be analyzed with the analysis tools we are hoping to use.

The workload will consist of two or more simple filters applied to varying number of video streams. Candidates for suitable filters are Sobel, Canny and similar. The workload will be implemented in two versions: 1. Simple actor based implementation built using PREESM which is scheduled statically according to hand derived schedule. 2. OpenEM adaptation of the simple implementation. OpenEM provides dynamic scheduling.

5.2 PREESM

PREESM is a research tool for prototyping actor model software on multi-core platforms. PREESM features graphical interface for construction of the actor network, the inter core communication and the code generation for target platform. PREESM supports code generation for TMS320C6678 and provides example implementation for Sobel filter.

5.3 TMDSEVM6678L with TMS320C6678 multi-core DSP processor

Hanhirova's `TECHNICAL REPORT TI TMS320C6678 evaluation board` document explains the hardware and related software well. The C6678 belongs to Texas Instruments Keystone family of DSPs.

The TMS320C6678 has eight C6678 - also known as CorePac - processors on it.

The processors are pipelined with the pipeline capable of dispatching 8 instructions every cycle. The processors also have vector processing units for 32, 16, 8 bit instructions.

In the Keystone devices the shared memory controller doesn't maintain memory coherency between the Cores. The coherency is managed by the application running on the devices. For example in [http://link.springer.com/chapter/10.1007%2F978-3-642-40698-0_9#page-1] OpenEM is utilized for the communication needed for memory coherency.

5.4 Analysis

The current candidate for analysing the execution is the CToolsLib Common Platform Tracers (CPTLib) tool which outputs the assembly instructions in order executed by the program and calculates cycle counts for them.

PSE - more info after the PSE introduction on week 23.

The workload execution will be measured dynamically using performance counters available in the TI hardware. The performance counters can be accessed through CPTLib.

5.5 Multicore programming

The TMS320C6678 has multiple different models of inter-core communication: Explicit through Inter Processor Communication, Multicore navigator and SRIO (Serial RapidIO).

Understanding of the capabilities of the multicore platform is vital for successful experiments. In the OpenEM version of the workload the multicore communication will be handled using Multicore Navigator through the TI OpenEM implementation. The explicit Inter Processor Communication will be utilized in the simple Actor version of the workload.

5.6 Code Composer Studio v5 / v6

Code Composer Studio is Eclipse based IDE for developing for TI hardware. CCS features a wide variety of debug, trace and system analysis tools.

5.7 OpenEM

Open Event Machine (OpenEM) is a framework of an event driven multicore optimized processing originally developed at NSN. In this thesis OpenEM will be used as the platform for parallelism. First item in the work plan is to get familiar with OpenEM in general and the Texas Instruments OpenEM implementation. TI OpenEM 1.0.0.2 is the newest OpenEM implementation available. (if anyone finds anything newer let me know.)

The Texas Instruments OpenEM implementation is a parallel runtime providing inter core communication, event scheduling, event queues and other required components for parallel processing applications. OpenEM is Operating System independent runtime framework. The TI OpenEM white paper states the following "OpenEM is able to operate in a heterogeneous OS environment: some cores may have SW running on the bare metal, others may have SW running on an OS and still others may have SW running on another OS."

5.8 People

Vesa Hirvisalo - instructor

Jussi Hanhirova - sort of internal client / co-researcher

Kristian Hartikainen - research on similar subjects

6 Challenges

Risto is currently an IEM (TuTa) student. In a discussion with Kivi-Koskinen Elsa who handles master's program studies at CSE I was assured that getting a supervisor will not be a problem before I'm officially CSE student. The thesis topic can be officially approved only after I've started at CSE which will mean the beginning of September.

It is currently unknown how difficult it will be to implement and analyze a suitable workload with the TI implementation of OpenEM. How well do OpenEM queues, multicore navigator and dynamic workloads play together? The number of events to be queued is unknown at compile time.

7 Work Plan

7.1 First iteration

set-up -> construct -> analyze -> repeat

Setting up the platform has already started. Compilation, execution and limited analysis possible on the TMS320C6678. Setting up will continue with getting more familiar with OpenEM, building and executing some OpenEM applications on the device.

More efforts on the analysis tools are needed but since the platform provides wide analysis support, more information about the specific needs is necessary before exploring analysis further.

7.2 More iterations

Depending on the findings from the first iteration further iterations may be needed.

8 Schedule

Summer - Complete the first iteration of the contribution.

2015-06-01	Construction planning complete
June week 1	Implement a simple (nonparallel) version of the workload
June week 2	Parallelize the workload using OpenEM
June week 3	Implement performance analytics in the application
June week 4	Tune the OpenEM implementation to handle dynamic workload
2015-06-30	Construction complete
2015-07-01	Analysis begins: configuring / implementing the instrumentation
2015-07	Vesa Hirvisalo vacation
2015-07-31	Analysis complete
2015-07-XX	Meeting on next steps
2015-08-01	Beginning of the second iteration (if needed)
2015-08-01	Focus on writing the thesis
2015-08-31	Potential second iteration complete

Fall - Further iterations, thesis writing, editing

9 Dissemination

ParallaX2 / ParallaX3 OpenEM streaming vs. PSE

Appendices

A Thesis structure

- Abstract
- Preface
- Contents
- Abbreviations

1. Introduction
   - Research Problem
   - Contributions
   - Structure
2. Background (split in to logical subsections)
   - Industrial Internet
   - Streams
   - DSPs
   - Actor Model
   - Dataflow (computing, programming)
3. Hardware Accelerated Scheduling
   - Purpose and background
   - Multicore Navigator in TMS320C6678
4. OpenEM
   - OpenEM structure
   - Texas Instruments OpenEM implementation
5. Analysis of Parallel Programs
   - Static Analysis
   - Dynamic Analysis
   - Tools
6. Experiments
   - Setup
   - Results
   - Conclusions

7. Discussion
   - Challenge
   - Discoveries
   - Related Work
   - Future Work
8. Conclusions

B Key Literature

General Background

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

D. Perino & al. a content router for high-speed forwarding on content names

Zhou & al. Scheduling of Parallelized Synchronous Dataflow Actors

Bachelor's theses

Teemaa, Taavi Rinnakkaisuuden mallit ohjelmointikielissä

Kiljunen, Olli Tehtävärinnakkaisuus ohjelmoinnissa

Master's theses

Hanhirova, Jussi Performance analysis of hardware accelerated scheduling

Saarinen, Joonas Parallel Processing of Vehicle Telemetric Data

Rasa, Marko

Texas Instruments Documentation

SPRUGR9H KeyStone Architecture Multicore Navigator User's Guide

SPRUGW0C TMS320C66x DSP CorePac User Guide

SPRUGZ2A KeyStone I Architecture Debug and Trace User Guide

TMS320C6678 datasheet

Dataflow & Actors

Lee, Seshia Introduction to Embedded Systems

OpenEM

Open Event Machine Library API Spec (TI)

Open Event Machine library User Guide (TI)

F. Moerman. Open event machine: A multi-core run-time designed for performance

Stotzer & al. OpenMP on the Low-Power TI Keystone II ARM/DSP System-on-Chip

Static and Dynamic Analysis

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