Thesis Experiment Plan OpenEM Based Video Filtering on Texas Instruments TMS320C6678 Week 27

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

June 29, 2015

Contents

1 Dynamic Workload Experiment on TMS320C6678

1.1 Outline

This document describes the construction of an dynamic workload experiment using OpenEM. OpenEM implementation on Texas Instruments TMS320C6678 will be used as the runtime system. The workload will be a simple video filtering application consisting of couple of filters and dynamic stream bitrate.

The implementation steps of the experiment are represented in the following figure.

Iteration 1 - Initial Implementation and Measurements

Filter application in PREESM

Convert Filter application to OpenEM

Instrumentation of both applications

Select a suitable workload for the applications

Measure applications

Model the OpenEM application with PSE

Compare measurements with the PSE estimates

Iteration 2 - Refining the Experiment

Compare the model behavior to the expected OpenEM behavior

If the model behavior matches the expected OpenEM behavior but not the real application behavior study the cause

Does the application utilize OpenEM according to specification?

Does OpenEM specification match the implementation on the platform

If faults in either model or the application are found fix them

1.2 Goals

The objective of this experiment is to understand the behavior and performance of Texas Instruments implementation of Open Event Machine in stream computation. The objective is achieved by comparing the performance of an application implemented using OpenEM to the performance of a similar application implemented using a simpler multicore runtime (PREESM). The OpenEM runtime system utilizes two performance enchancing features the simpler runtime does not, namely hardware accelerated inter-core communication and dynamic scheduling. The hypothesis is that OpenEM will achieve better performance under dynamic workload. Performance difference is expected to be smaller for static workloads.

The second part of the experiment is the construction of a simulation model. The performance estimates from the constructed simulation model will be compared to the performance of the real world application. The objectives of the comparison are to help better understand the OpenEM platform and to assess the utility of simulation and modeling in analysis of stream computation.

2 Texas Instruments TMS320C6678

2.1 Overview

The hardware platform used in this experiment is Texas Instruments TMD-SEVM6678L TMS320C6678 Evaluation Module. The device belongs to the Keystone I family of multicore DSPs. The device has eight C6678 dsp cores. Each core can dispatch eight instructions every cycle. Communication between cores can be handled through hardware accelerated packet communication channel called Multicore Navigator or through shared memory using user defined locking scheme. For more details see the technical report by Hanhirova and Texas Instruments manuals.

Hanhirova, Jussi: TECHNICAL REPORT TI TMS320C6678 evaluation board characteristics, tools and analysis mechanisms for the purpose of static analysis
Texas Instruments: Multicore Fixed and Floating-Point Digital Signal Processor, tms320c6678.pdf

2.2 Multicore Navigator

The Multicore Navigator provides high speed packet communication on the device. The Multicore Navigator provides a hardware queue manager called the Queue Manager Subsystem (QMSS), packet DMA (PKTDMA) and multicore host notifications via interrupt generation.

In the Keystone I architecture there are two Packed Data Structure Processors (PDSP) in the QMSS. The PDSPs execute a firmware that can perform QMSS related functions. In this experiment firmware provided in the TI OpenEM implementation will be used. The OpenEM runtime system uses hardware queues to accelerate inter-core communication. Firmware specific to OpenEM handles the hardware queue management.

The packets consist of descriptors and payload data. The packets reside in the memory and pointers to the packets are passed from core to core using the Multicore Navigator.

Texas Instruments: KeyStone Architecture Multicore Navigator, sprugr9h.pdf

2.3 Development Environment

Software is developed for the device using an IDE from Texas Instruments called Code Composer Studio. CCS version 5.2.1 is used. The Software Development Kit used for development for the device is called Multicore Software Development Kit (MCSDK). MCSDK version 2.1.2.5 is used.

CCS features a variety of debug, trace and analysis tools but to access performance counters in the hardware, libraries external to MCSDK are required. The needed trace libraries are distributed under collective name of CToolsLib. The newest versions of the libraries supporting the device are used.

CToolsLib download https://gforge.ti.com/gf/project/ctoolslib/frs/?action=index

3 Filters

A realistic workload application with the capability for processing dynamic input is designed for the experiment. DSPs are commonly used for video stream processing. Processing video streams is a suitable workload for parallelization and dynamic number of inputs. Video filtering is a common and fairly simple task in video stream processing. For these reasons a video filtering application is selected as the workload for the experiment.

The application consists of two common filters. The first is a sobel filter that is used in edge detection. The second is a gaussian filter that has a wide range of uses. Both filters are used in canny edge detectors which are realistic applications for multicore DSPs. The real world users of the studied device would probably implement a complete edge detector rather than parts of it. However the workload in this experiment is deliberately kept simple to make it well analyzable and reasonably simple to implement.

The filters themselves are based on convolving the image data with a filter matrix and are thus straightforward to implement.

Gaussian Filter http://en.wikipedia.org/wiki/Gaussian_filter
Sobel Operator http://en.wikipedia.org/wiki/Sobel_operator
Canny Edge Detector http://en.wikipedia.org/wiki/Canny_edge_detector

4 PREESM

Actor Model is selected as the top-level programming model for the workload application. Actor Models are specifically designed for parallel programming and fit the OpenEM defined application layout well. Prototyping Actor Model programs is straightforward due to available graphical tools. Actor Model rapid prototyping tool PREESM will be utilized in the experiment to build a simple version of the workload. PREESM was selected over comparable tools for the following reasons: 1. PREESM supports TMS320C6678 as a compilation target. 2. PREESM is under active development.

PREESM version 2.1.3 and Eclipse Luna Service Release 2 (4.4.2) will be used in the experiment.

Pelcat M. et. al. Preesm: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming 10.1109/EDERC.2014.6924354

5 OpenEM

5.1 Open Event Machine

Open Event Machine is a runtime system for multicore platforms originally developed by NSN for the network dataplane. OpenEM is used to write dynamically load balanced applications with run-to-completion principle.

The key concepts of OpenEM are events, execution objects, queues and the scheduler. **Event** is the unit of communication in OpenEM. Events are typically used to carry data to be processed but can be data-less tokens as well. The event descriptors are allocated at the initialization of the Queues. **Execution objects** encapsulate the algorithm to execute when an event is received. **Queues** connect events (data) and execution objects (algorithms). Each queue is associated with one execution object and all queued events will be processed by this execution object. **Scheduler** moves allocated events to queues. **Dispatcher** is called by the user in a dispatch loop. Dispatcher calls the 'receive' function of the execution object of the connected queue.

5.2 Texas Instruments Implementation of OpenEM

The Texas Instruments implementation of Open Event Machine runtime system is included in the MCSDK (chapter 2.3). OpenEM version 1.0.0.2 is used in the experiment.

The OpenEM library is OS agnostic: the OpenEM dispatcher can be called from bare metal software or from an OS thread. The TI implementation of OpenEM Leverages the Multicore Navigator for hardware queues and packet communication. The scheduler always runs on a PDSP core and is defined by firmware specific to OpenEM.

The OpenEM execution model is described in figure 1. The event descriptors are allocated at OpenEM initialization in the free pool. The application allocates events from the free pool using *em_alloc*. Events can be allocated either at runtime or at initialization. After the allocated event has been populated it is sent to a queue using *em_send*. The queues are associated with specific execution objects at initialization and therefore they determine which EO will process the events the queues. The scheduler running on a PDSP core (chapter 2.2) manages the queues (virtual queues mapped to QMSS hardware queues). The scheduling operations are carried out on an explicit request from a DSP core when a call to *ti_em_preschedule* is made. When one of the cores mapped to a specific execution object finishes executing its current event it makes a call to *ti_em_dispatch_once* which triggers a call to the *receive* function of the execution object. The execution object processes the data and makes a call to *em_free* to return the event to its free pool.

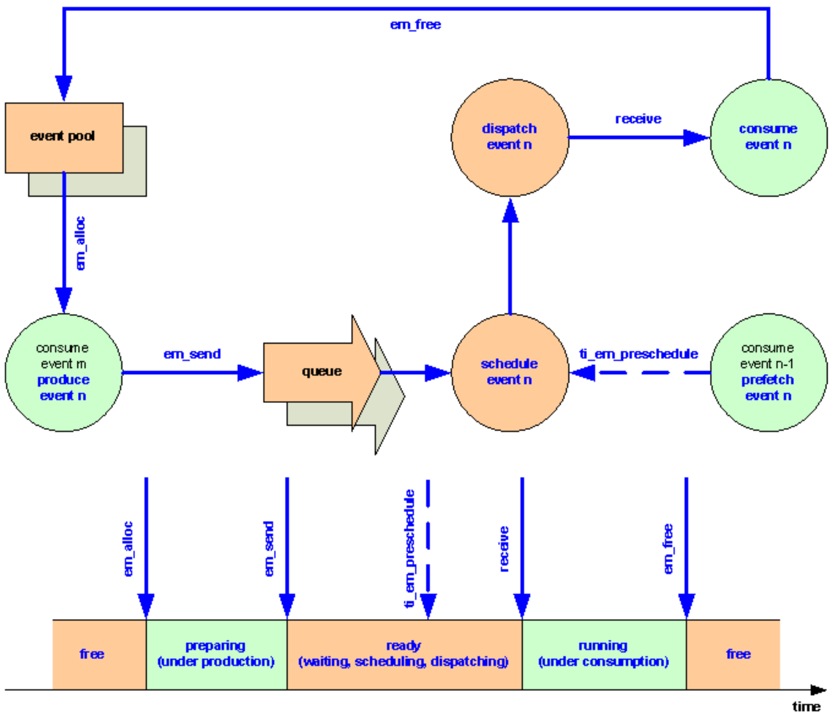

Figure 1: The OpenEM execution model

The concept of queue groups is used to specify which cores can execute which EOs. Queues are associated with EOs and each queue belongs to a queue group. Queue groups define which cores can execute the events from the queues in the group using a core mask.

The scheduler running on a PDSP core considers four scheduling criteria: **Priority**, **Atomicity**, **Locality** and **Order**. Each queue has a priority. The scheduler will always select events from queues with the highest priority. Queues are either atomic or parallel. Events from parallel queues can be scheduled on as many cores as are available for the queue. In contrast no event from an atomic queue can be scheduled while another event from the same queue is executing. The scheduler tries to schedule events from certain queues on the same cores if possible. The last criterion for scheduling is order. If multiple events are eligible for scheduling, the event that has been in a queue longest will be scheduled.

Texas Instrument OpenEM White Paper, ti.openem.white.paper.pdf
Texas Instrument OpenEM User Guide, ti.openem.user.guide.pdf
Texas Instrument OpenEM Api Guide, ti.openem.api.guide.pdf

6 Measurements

6.1 Desired Metrics

Runtime performance measurements are carried out on both of the implemented applications. The high level metric used is video frames per second. Finer grained metrics are used to understand the details of the Texas Instruments OpenEM implementation. Clock cycle accurate metrics are needed for the construction of the PSE simulation model. The selection of the metrics is easier with the initial construction of the simulation model completed.

An area of interest for the finer grained metrics are the costs associated with the OpenEM runtime model. Even though the PREESM generated application is not an optimized application it will hopefully show the performance behavior to be expected from a simpler, non-hardware accelerated multicore implementation. Relating the PREESM application behavior to the OpenEM application behavior gives us a good view to the performance of the OpenEM runtime.

Quantitative Metrics

FPS - video Frames Per Second. How many frames of each stream per second the application is capable of processing.

Core Utilization - How many instructions were dispatched on each core per clock cycle. Measuring core utilization will help us understand the efficiencies of the scheduling strategies.

Time Per Actor / Execution Object - Which parts of the program the execution is spending the most time at.

Other - Other quantitative metrics are implemented if required for the construction of the simulation model.

6.2 Implementation of the metrics

The frames per second metric is straightforward to implement by measuring the number of complete frames processed per time unit.

Fine grained metrics will be implemented using Texas Instruments tools provided in CCS and CToolsLib. CToolsLib provides an API for accessing performance counters of the different hardware components and exporting the performance data. CCS also provides a profiling tool called System Analyzer which might be useful for some measurements as it has a relatively simple interface for performance measurements on the device.

6.3 Measurement Parameters

Dynamic workload conditions are emulated by repeating the measurements with different parameters. To keep things simple video streams are not dynamically switched at runtime. The measurement parameters are presented in the listing.

> **Measurement Parameters**
>
> **Video Frame Size -** The workloads are differentiated by changing the frame size of the video streams.
>
> **Number of Frame Slices -** The video frames are sliced in to a number of slices. The unit of data to be processed is one frame slice, therefore the number of slices affects the granularity of the schedule.
>
> **PREESM Core Mapping -** The PREESM application is measured with different core mappings of the actors to find out how large an effect the static mapping has under dynamic workload.
>
> **OpenEM Core Masks -** The OpenEM application is measured with different core masks of the Execution Objects to learn more about the behavior of the scheduler.

The parameters in the listing and many other parameters could be used as factors for different measurement setups but the utility of additional factors needs to be evaluated before the final selection is done. If too many factors are selected, conducting the measurements will become very time consuming and pontentially generate unnecessary data that only complicates the analysis. The final factors of the measurement are selected after the implementation of iteration 1 is complete.

7 Construction

7.1 Video Filter Application Using PREESM

An actor network is constructed in PREESM that represents the video filter application. A network sketch is presented in Figure 2. To keep the model simple and the program well analyzable all of the processing paths in the network are independent. The shared dependencies in the sketch (marked with dashed lines) are constants related to the video stream and will change in the actual application.

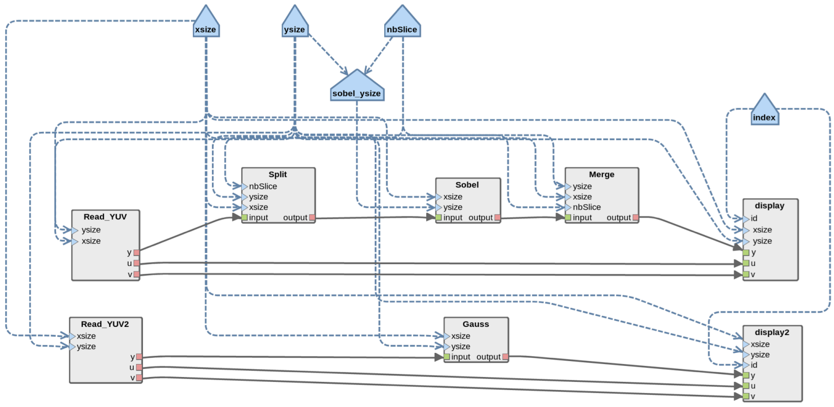

Figure 2: A Sketch of The Video Filter Actor Network

The first actor on each of the processing paths loads the video frames from memory and passes them to splitting actors. The splitting actor splits the frames to a suitable number of splices to enable processing of the same video stream on multiple cores. The filter actor follows the splitting actor. Partial frames filtered in the filter actor are merged back to whole frames in the merge actors. The last actors on each processing path are sink actors that collect data from the execution and frees resources.

The filter actors are run on all cores following a manually created schedule. The manual schedule determines the execution in the simpler runtime system and is thus very important for the experiment results. The details of the schedule are determined according to the workload cases generated.

7.2 Video Filter Application Using OpenEM

The video filter application first constructed as a PREESM actor model, will be implemented using OpenEM. The mapping from the actors in the actor model to the execution objects in OpenEM is expected to be one-to-one. An execution object is allocated for each actor in the actor model. The events passed between the actors encapsulate the data in the video streams.

In contrast to the PREESM application where the schedule has to be changed for each streaming case, the OpenEM implementation will not be changed for the different workloads.

The specifics of setting up the execution objects, number of queues attached to them, the priority of the queues and other specifics depend heavily on the final workload application and thus are not further explained here. The details are also subject to change based on the initial measurements to make the workload better reflect a realistic application.

7.3 Workload

The workload application needs to be kept simple to keep it analyzable. Complex parts of real applications unrelated to the parallel execution such as I/O are omitted from the workload implementation. Without I/O available the video streams need to be preloaded to the memory. CCS provides tools for preloading.

The purpose of the experiment is to understand the OpenEM behavior under dynamic load. Organizing the workload so that the number of streams could actually be dynamically changed at runtime will not be necessary if the cost of switching between streams can be estimated. A number of static loads will be constructed and measured on both of the applications.

8 Performance Simulation Environment

8.1 Performance Simulation Environment

Performance Simulation Environment (PSE) is a resource network based simulation environment that can be used to construct models of applications at different levels of abstraction. If a low abstraction level simulation model is desired, specific measurements of the runtime are needed. The measurements conducted on the second iteration will depend on the needs of the simulation model.

8.2 Modeling the workload using PSE

The main objective of this experiment is to understand OpenEM as a computation platform for stream computation. Analysis of stream computation is complicated especially on load balanced platforms such as TI implementation of OpenEM. One method of analysis is modeling and simulation of the application. A simulation model of the OpenEM application is constructed firstly to improve understanding of the OpenEM runtime system and secondly to evaluate modeling and simulation as a method of understanding stream computation.

9 Results

Two applications and a simulation model are implemented for the experiment. The simulation model is potentially interesting for future research on the subject. Representing the simulation model can be compactly done by providing the reader with pictures of the resource network accompanied with description of the model. A simplified representation of the PREESM actor model could be also included.

As there are many measurement setups generating a lot of data, the representation of the measurements needs to be carefully designed. The initial idea is to use tables listing the different application setups on one axis and the stream setups on the other axis to present data from a single measurement. Key measurements will be visualized using suitable graphs.

Table 1: Preesm Measurement Setups

	PREESM 4/4	PREESM 2/7	PREESM 8/8
Stream 1/1			
Stream 1/2			
Stream 1/3			

Table 2: OpenEM Measurement Setups

	OpenEM 1	OpenEM 2	OpenEM 3
Stream 1/1			
Stream 1/2			
Stream 1/3			