Vectorization and Mapping of Software Defined Radio Applications on GPU Platforms

Shuvra S. Bhattacharyya
Professor, ECE and UMIACS
University of Maryland at College Park
ssb@umd.edu, http://www.ece.umd.edu/~ssb

With Contributions from G. Zaki, W. Plishker, C. Clancy, and J. Kuykendall

GPU Summit at the UMD/NVIDIA CUDA Center for Excellence,
College Park MD, October 27, 2014
Outline

• Introduction

• Contribution:
  Novel Vectorization and Mapping Workflow.

• Evaluation

• Summary
Outline

• Introduction

• Contribution: Novel Vectorization and Mapping Workflow.

• Evaluation

• Summary
DSPCAD Methodologies: Computer-Aided Design (CAD) for Digital Signal Processing (DSP) Systems

Applications and Tasks [Bhattacharyya 2013]

Image: medical, computer vision, feature detection, etc.

- Imaging device
- Data preprocessing
- Image reconstruction
- Post reconstruction
- Advanced image analysis
- Image visualization

Video: coding, compression, etc.

- Color processing
- Prediction
- Transformation & Quantization
- Entropy Coding

Audio: sample rate conversion, speech, etc.

- Audio device
- Data preprocessing
- Feature extraction
- Data postprocessing

Platforms

- Programmable DSP
- GPU
- FPGA
- Microcontroller

Wireless communication systems

- Source encoding
- Channel encoding
- Digital modulation
- D/A conversion
- RF Back-end
Motivation

• Diversity of platforms:
  – ASIC, FPGA, DSPs, GPUs, GPP

• Complex application environments
  – GNU Radio

• Exposing parallelism
  – Task, Data and Pipeline

• Difficult Mapping Problem

• Multi-objective (throughput, Latency)
Background: GNU Radio

- A software development framework that provides software defined radio (SDR) developers a rich library and a customized runtime engine to design and test radio applications [Blossom 2004]
DSP-oriented Dataflow Models of Computation

- **Application is modeled as a directed graph**
  - Nodes (actors) represent functions of arbitrary complexity
  - Edges represent communication channels between functions
  - Nodes produce and consume data from edges
  - Edges buffer data *logically* in a FIFO (first-in, first-out) fashion

- **Data-driven execution model**
  - An actor can execute whenever it has sufficient data on its input edges.
  - The *order in which actors execute is not part of the specification.*
  - The order is typically determined by the compiler, the hardware, or both.

- **Iterative execution**
  - Body of loop to be iterated a large or infinite number of times
DSP-oriented Dataflow Graphs

- Vertices (actors) represent computational modules
- Edges represent FIFO buffers
- Edges may have delays, implemented as initial tokens
- Tokens are produced and consumed on edges
- Different models have different rules for production (SDF $\rightarrow$ fixed, CSDF $\rightarrow$ periodic, BDF $\rightarrow$ dynamic)
Dataflow Production and Consumption Rates

- \( p_{x,y} \) denotes the number of tokens produced onto edge \( e_x \) by the \( y \)th firing of its source actor (for \( y = 1, 2, \ldots \)).

- Similarly, \( c_{x,y} \) denotes the number of tokens consumed from edge \( e_x \) by the \( y \)th firing of its sink actor.
Dataflow Graph Scheduling

• Assigning actors to processors, and ordering actor subsets that share common processors
• Here, a “processor” means a hardware resource for actor execution on which assigned actors are time-multiplexed
• Scheduling objectives include
  – Exploiting parallelism
  – Buffer management
  – Minimizing power/energy consumption
Background: Contemporary Architectures

Vector Operations in General Purpose Processors (GPPs)

Graphics Processing Units (GPUs)
Primary Contribution

A novel workflow for scheduling SDF graphs while taking into account
- Actor execution times.
- Efficient vectorization.
- Heterogeneous multiprocessors.

Demonstration system
- Applications described in a domain specific language.
- Systematic integration of precompiled libraries.
- Targeted to architectures consisting of GPPs and GPUs.
Previous Work

• Automatic SIMDzation [Hormati, 2010]
  - Based on StreamIT compiler.
• Hierarchical models for SDR [Lin, 2007]
  - Targeted towards special architectures.
• Multi-processor scheduling [Stuijk, 2007]:
  - Formulation towards special objectives.
• Vectorization [Ritz, 1992]:
  - Single processor block processing optimization.
Outline

• Introduction

• Contribution: Novel Vectorization and Mapping Workflow.

• Evaluation

• Summary
DIF-GR-GPU Workflow

- Start from a model-based application description.
- Use tools to optimize scheduling, assignment.
- Generate an accelerated system.
Workflow Goals

• Adequately make use of all sources of parallelism in order to utilize the underlying architecture.

Sources of parallelism:
  – Data Parallelism (prod and cons in SDF)
  – Task Parallelism (implicit in DFG)
  – Pipeline Parallelism (Looped schedules)
SDF Scheduling Preliminaries

- An SDF graph $G = (V,E)$ has a valid (periodic) schedule if it is deadlock-free and is sample rate consistent (i.e., it has a periodic schedule that fires each actor at least once and produces no net change in the number of tokens on each edge).

- For each actor $v$ in a consistent SDF graph, there is a unique repetition count $q(v)$, which gives the number of times that $v$ must be executed in a minimal valid schedule.

Some Possible Schedules: (1) AABAB (2) AAA BB

$q(A) = 3$ $q(B) = 2$
DIF-GR-GPU: Dataflow Scheduler

Objective:
- Optimize exploitation of data and pipeline parallelism
  → Higher throughput.

Flat Schedule: Executes an SDF graph as a cascade of distinct loops with no inter-actor nesting of loops.

Vectorization of a schedule S: A unique positive integer $B$, called the blocking factor of S, such that S invokes each actor $v$ exactly $(B \times q(v))$ times.

Original SDF Graph

Corresponding BPDAG, $B = 10$
DIF-GR-GPU Workflow

- Start from a model-based application description
- Use tools to optimize scheduling, assignment.
- Generate an accelerated system.

GNU Radio

Data Flow Graph
Throughput, Latency Constraints

Dataflow Scheduler

Application Graph

Multiprocessor Scheduler

Mapping and Ordering Schedule

GNU Radio Engine

Library of Actor Implementations

Final Implementation
Heterogeneous Multiprocessor Scheduler

- **Objective**: Utilize available multiprocessors in the platform.

- Architecture Descriptions: The platform is described by a set \( P \) of processors and a set \( B \) of all to all communication buses.

- Execution times depend on the blocking factor.

- Every processor is assumed to have a shared memory.
Scheduler Inputs

- **Architecture description**: set $P$ of processors and a set $B$ of communication buses.

- **Application description**: The application model (input BPDAG) consists of a set $T$ of tasks, and a set $E$ of edges.

- **Task and edge profiles**: These profiles are described by two functions:
  - $RTP(t \in T, p \in P) \rightarrow R$ defines the execution time of task $t$ on processor $p$,
  - $REB(e \in E, b \in B) \rightarrow R$ defines the execution time of edge $e$ on bus $b$.

- **Dependency analysis**: Task $t_1$ is said to be dependent on task $t_2$ if there is a path that starts at $t_1$ and ends at $t_2$. If no such path exists between $t_1$ and $t_2$, then they are called parallel tasks. A similar concept can be applied to edges.
Multiprocessor Scheduler

• The basic scheduler functionality is to
  – Map every task to a given processor.
  – Order the execution of parallel actors assigned to the same processor.
  – “Zero” out the communication cost of collocated dependent actors.

• The scheduler objective is:
  Minimize the latency $L_B$ of $B$ graph iterations.
MLP formulation

• Why?
  – Offline analysis of SDF graphs.
  – Coarse grain nature of SDF graphs.
  – Solver gives a bound from optimal solution.

• Basic Variables:
  – **Mapping:** $X_T[t, p] = 1$ if task $t$ is assigned to processor $p$; $X_T[t, p] = 0$ otherwise.
  – **Ordering:** For all parallel tasks $t_1, t_2$ that are assigned to the same processor $Y_T[t_1, t_2] = 1$ if $t_1$ is ordered before $t_2$; $Y_T[t_1, t_2] = 1$ otherwise.
  – **Running time:** $RT[t] =$ actual (platform dependent) execution time of the task $t$ depending on its mapping.
  – **Start time:** $ST[t]$ is the start time for execution of task $t$. 
MLP formulation (continued)

- **Constraints:**
  - Assignment: \( \forall t \in T, \sum_{p \in P} XT[t, p] = 1 \) and \( \forall e \in E, \sum_{b \in \beta} XE[e, b] = 1 \)
  - Dataflow dependency: \( SE[e] \geq ST[src(e)] + RT[snk(e)] \)
  - Zero cost communication:
    \[
    \begin{align*}
    ST[t_1] &\geq ST[t_2] + RT[t_2] - K(1 - YT[t_1, t_2]) - K \times ZT[t_1, t_2] \\
    ST[t_2] &\geq ST[t_1] + RT[t_1] - K \times YT[t_1, t_2] - K \times ZT[t_1, t_2]
    \end{align*}
    \]

- **Objective:** Minimize \( M \)
  \[
  \forall t \in T, M \geq ST[t] + RT[t]
  \]
Outline

• Introduction

• Contribution:
  Novel Vectorization and Mapping Workflow.

• Evaluation

• Summary
DIF-GR-GPU Workflow

- Start from a model-based application description using the dataflow interchange format (DIF)
- Use tools to optimize scheduling, assignment.
- Generate an accelerated system.
GPU interface GRGPU [Plishker 2011]

- GNU Radio
- NVIDIA CUDA Libraries

**GRGPU**

- `device_work()`
- C++ Block with call to GPU Kernel (.cc)
- GPU Kernel in C++ (.cc)
- CUDA Synthesizer (nvcc)
- CUDA Libraries
- libtool
- Standalone Python package

- Source
- H2D
- FIR (GPU accelerated)
- D2H
- Sink
MP-Sched Benchmark (GNU Radio)

# of Stages

SRC

FIR

FIR

FIR

FIR

SNK

FIR

FIR

FIR

FIR

FIR

FIR

FIR

FIR

# of Pipelines
Realization of a 2x5 MP-Sched Graph

Application: 2x5 mp-sched graph
Platform: 1 GPP (Intel XeonCPU 3GHz), 1 GPU (a NVidia GTX 260), and a PCI
Blocking factor B : 2048.

<table>
<thead>
<tr>
<th>Amount of improvement</th>
<th>All GPP</th>
<th>All GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytical</td>
<td>55%</td>
<td>19%</td>
</tr>
<tr>
<td>Empirical</td>
<td>39%</td>
<td>21%</td>
</tr>
</tbody>
</table>
Latency vs. Throughput Trade-offs

Each point an optimized assignment for each blocking factor.
MLP Solver Running Time

Table 1. Solver results for different mp-sched graphs.

<table>
<thead>
<tr>
<th>Graph Size</th>
<th>Plat. Desc.</th>
<th>Lat</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPES</td>
<td>STAGES</td>
<td>GPPs</td>
<td>GPUs</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

- Problem written in MathProg.
- Solved using the IBM ILOG CPLEX optimizer
- On Intel Core 2 Duo processor at 3 GHz.
Outline

• Introduction

• Contribution: Novel Vectorization and Mapping Workflow.

• Evaluation

• Summary
Summary

• Diversity of platforms:
  – ASIC, FPGA, DSPs, GPUs, GPP

• Complex application environments
  – GNU Radio

• Exposing parallelism
  – Task, Data and Pipeline

• Difficult Mapping Problem

• Multi-objective (throughput, Latency)
To Probe Further …


To Probe Even Further

- Foreword by S. Y. Kung
- Part 1: Applications
- Part 2: Architectures
- Part 3: Programming and Simulation Tools
- Part 4: Design Methods

Acknowledgements

• This research was supported in part by the Laboratory for Telecommunications Sciences.

• For more details on this project, other projects in the Maryland DSPCAD Research Group, and associated publications: http://www.ece.umd.edu/DSPCAD/home/dspcad.htm.
References 1


References 2