DFT Compiler for Custom and Adaptable Systems

Paolo D’Alberto

Electrical and Computer Engineering
Carnegie Mellon University
Personal Research Background

- Theory of computing
- Embedded and High Performance Computing
- Compiler:
  - Static and Dynamic
- Algorithm Engineering

Timeline:
- Padova
- Bologna
- UC Irvine
- CMU
Problem Statement

♦ Automatic DFT library generation across hardware and software with respect to different metrics
  - Time (or performance Operations per seconds)
  - Energy (or energy efficiency Operations per Joule)
  - Power

Requires the following

♦ Software (code generation or selection)
♦ Hardware (HW generation or selection)
♦ Software/Hardware Partitioned
♦ Demonstrate Performance and Energy Efficiency
♦ Demonstrate automatic generation
SPIRAL’s Approach

- One infrastructure for SW, HW, SW/HW
- Optimization at the “right” level of abstraction
- Conquers the high-level for automation
Performance/Energy Optimization of DSP Transforms on the Intel XScale Processor

Do Different Architectures need different Algorithms?
Motivation (Why XScale ?)

♦ XScale processor deploys a interesting ISA
  - Complex Instructions: pre-fetching, add-shift …
  - Only Integer Operations

♦ XScale is an re-configurable architecture
  - architecture = MEMORY + BUS + CPU
  - Memory: $\tau = 99, 132$ and $165$MHz
  - BUS: $\tau \alpha/2$ MHz where $\alpha = 1,2,$ and $4$
  - CPU: $\tau \alpha \beta$ MHz where $\beta = 1, 1.5, 2$ and $3$

♦ $(\alpha, \beta, \tau)$ we can tune/change at run time by SW

♦ 36 possible architectures
  - 4 Recommended (by the manual),
  - 13 Investigated (13 is a lucky number in Italy)
13 different architectures
- Which one to use, which one to write code for?
- We write and test codes for the fastest CPU?
- Fastest Bus? Memory? Slowest?

SPIRAL: Re-configurable SW
- DSP-transforms SW generator for every architecture
- Targeting and deployment the best SW for a HW

SPIRAL: Inverse Problem by fast Forward Pro
- Given a transform/application, what is the best code

SPIRAL: Energy Minimization by dynamic ada
- Slow-down when possible (w/o loss of performance)
- Using simple and fast to apply techniques
Related Work

♦ Power/Energy Modeling (each system)
  - To drive the architecture selection [Contreras & Martonosi 2005]

♦ Compiler Techniques
  - Static decision where to change the architecture configuration
  - [Hsu & Kremer 2003, Xie et al. 2003]

♦ Run time Techniques
  - By dynamically monitoring the application and changing architecture
  - [Singleton et al. 2005]

♦ SW adaptation
  - The code adapts to each architecture

♦ One code fits all
  - IPP
Feedback/SPIRAL Approach

- Given a transform
- Choose a HW configuration
- Select an algorithm
- Determine an implementation
- Optimize the code
- Run the code
- Dynamic Programming/Others
- Prune the search space
  - HW, Algorithm, Implementation
**SPIRAL Approach**

- Same infrastructure for SW, HW, SW/HW
- Optimization at the “right” level of abstraction
- Complete automation: Conquers the high-level for automation

**Transform**

- Ruletree
- SPL
- Σ-SPL

**Tools** (traditional human domain vs. available)

- Problem specification
- Easy manipulation for search
- Frequency/parallel/streaming opts.
- Loop optimizations

**Problem Specification**

-DFT<sub>256</sub>
-DFT<sub>256</sub>
-DFT<sub>16</sub>

-...(I₄ ⊕ J₄)D₈(F₂ ⊕ I₄)...

-...∑<sub>j=0</sub> Sₙ(j) mₙ n Cₙ Gₙ(j) mₙ n

**Machine Code**

- SW (C/Fortran)
- SW vector/parallel
- SW/HW partitioned

**Netlist**

- HW (RTL Verilog)
Our Approach: From Math to Code -- Static

♦ We take a transform $F$
  - E.g., $F = \text{DFT}$, $\text{FIR}$, or $\text{WHT}$ …

♦ We annotate the formula with the architecture information

$$[F]_{497-1/3-1/3}$$

♦ We generate code
  - The switching point are transferred to the code easily
Our Approach: From Math to Code -- Dynamic

♦ We take a transform (WHT: consider the input as a matrix):

\[ WHT_{2n} = (WHT_{2k} \otimes I_{2m})(I_{2k} \otimes WHT_{2m}), \quad n = k + m. \]

WHT on the column \quad WHT on the rows

♦ We annotate the formula with the architecture information

\[
\begin{align*}
[(WHT_{2k} \otimes I_{2m})_{497-1/3-1/3}] & \cdot [(I_{2k} \otimes WHT_{2m})_{530-1/4-1/2}]
\end{align*}
\]

E.g., Poor cache use \quad E.g., Good cache use
fast Memory reads/writes \quad fast CPU and Bus

♦ We generate code

- The switching point are transferred to the code easily, through the rule tree
- Very difficult to find these from the code directly
Dynamic Switching of Configuration

- Red = 530-132-256
- Blue = 497-165-165
- Grey = no work

Ruletree = Recursion strategy

Execution

530-132-265 497-165-165 530-132-265 497-165-165

Time
Setup: Code Generation for XScale

SPIRAL Host
(Linux/Windows)
♦ Formula generation
♦ Rule tree
♦ Code optimization
♦ Code generation
♦ Cross compilation
♦ Feedback timing to guide the search

XScale (Linux)
♦ Timing
♦ Feedback to spiral

We measure Execution Time, Power, Energy of the entire board and compare vs. IPP
DFT: Experimental Results

IPP 4.1

SPIRAL

Better
FIR 16 taps: Experimental Results

IPP 4.1

SPIRAL

Better
WHT: Experimental Results

- If the switching time is less than 0.1ms we could improve performance using 530-1/4-1/2 and 497-1/3-1/3
- Switching between 398-14-1/3 and 497-1/3-1/3 improves energy consumption 3-5%
DFT Compiler for CPU+FPGA

What if we can build our own DFT co-processor?
SW/HW Partitioning Backgrounder

Basic Idea

♦ Fixed set of compute-intensive primitives in HW
  ⇒ performance/power/energy efficiency
♦ Control-intensive SW on CPU
  ⇒ flexibility in functionality
♦ E.g. support a library of many DFT sizes efficiently
Core Selection: Rule-Tree-Based Partitioning of DFT

- Recursive structure of DFT
- Map a set of fixed size DFTs (that fits in HW)
- Rest is performed in generated SW
- Few HW modules can speed up an entire library

HW/SW Partitioning: how to choose?
Related Work

♦ The problem we try to solve is NP-Hard [Arato et al. 2005]
  - Optimization of metrics with area constraints

♦ SpecSyn (and other System Level Design Languages)
  - Specification of the behavior and partitioning of a system

♦ Heuristics for the partitioning problem
  - Kavalade and Lee 1994, Knerr et al. 2006

♦ Solution for specific problems
  - JPEG Zhang et al. 2005 or reactive systems Weis et al. 2005

♦ Compiler Tools
  - Hot-spot determination Stitt et al. 2004
  - Pragmas NAPA C 1998

♦ Architectures specification
  - Garp 1997
SPIRAL’s Approach

- One infrastructure for SW, HW, SW/HW
- Optimization at the “right” level of abstraction
- Conquers the high-level for automation

Diagram:
- DFT
- Ruletree
- SPL
- Σ-SPL
- SW (C/Fortran)
- SW vector/parallel
- SW/HW partitioned
- HW (RTL Verilog)

Transformations:

- Problem specification
- Easy manipulation for search
- Vector/parallel/streaming optimizations
- Loop optimizations
Virtual Cores by a HW interface

- A DFT of size $2^n$ can be used to compute DFTs of sizes $2^k$, for $k < n$
- Virtual core for $2^k$ using a $2^n$ core:
  - Up-Sampling in time, periodicity in frequency
  - Same latency as real $2^n$ core
  - Same throughput as real $2^k$ core, for $n-c \leq k < n$
Experiment: Partitioning Decision

♦ HW:
  - E.g., DFT cores for sizes 64, 512
  - E.g., Virtual DFT cores for sizes 16, 32, 128, and 256

♦ SW:
  - Generated library for 2-power sizes
One Real HW Core DFT

“HW only”

HW speeds up SW

Performance [Maps] vs. Problem size

- **Software only vs. SW + HW core DFT**
- Early ramp-up and peak, 1.5x – 2.6x speed-up
One Real HW Core DFT\textsubscript{512}

- Software only vs. SW + HW core DFT\textsubscript{512}
- Slower ramp-up but better speed-up
Two Real HW Cores: $DFT_{64}$ and $DFT_{256}$

- DP search: Software only vs. SW + HW (both, $DFT_{512}$ only, $DFT_{64}$ only)
- For a library: 2 cores combine early ramp-up with high speed-up
Performance

- Software only vs. SW + 2 HW cores
- Clear winner for each size, but for whole library?
- This is performance, but what about power/energy?
Energy/Power:

- **Software only vs. SW + 2 HW cores**
- **Clear winner for each size, but for whole library?**
- **What about performance/power trade-off?**
Error Analysis

SNR

Average Error

Maximum Error

Better
Conclusions

♦ We can evaluate different performance metrics for different transforms and architectures
  - Different applications need different architectures
  - Different architectures need different algorithms

♦ We can evaluate the dynamic effects of configuration manipulation
  - XScale
    ▪ No performance improvement (unless the switching time is <0.1ms)
    ▪ Energy efficiency may improve by 3-5% using the current switching
  - SW/HW
    ▪ Large difference in using different configurations

♦ Spiral approach simplifies the search, the code generation and the code evaluation
SPIRAL Team

HW

SW

Carnegie Mellon