Design Automation of Co-Processors for Application Specific Instruction Set Processors

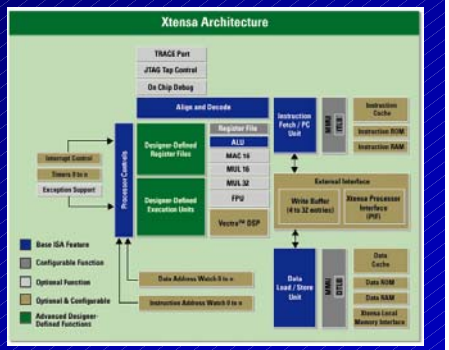
Seng Lin Shee

Outline

1. Introduction
2. Justification & Aims
3. Work done & Accomplishments
4. Current Research
5. Customized Architecture
6. Future work

ASIPs in General

- ASICs vs GPPs situation
- Power & Performance vs Design / Manufacturing Cost
- ASIPs are the hybrid of the two
- Main characteristic: highly configurable
- Consist of a base processor and optional components
- Today's ASIPs are extensible
- Xtensa, Jazz, PEAS-III, ARCtangent, Nios, SP5-flex



Aim

- Automatically create coprocessors for critical loops
- Create coprocessors which acquire small area, power and fast
- Maximize parallelism
- Design the methodology to create coprocessor
- Create estimation methods / ILP formulatiom

Related Work

- [Ernst1993] Hardware software cosynthesis for microcontrollers
  - Standard processor is connected by the main memory bus to a co-processing ASIC/FPGA
  - Disadvantage: only produce a small amount of improvements; no parallelism involved; also degradation in performance
- [Stitt2003] Dynamic Hardware/Software Partitioning: A First Approach
  - Hardware approach to profile program dynamically
  - Synthesize onto FPGA; dynamic partitioning to extract appropriate loop
  - Disadvantage: only small regions of code; single cycle loop body; sequential address of memory block; number of iterations must be predetermined
- CriticalBlue
  - Provides complete methodology with toolset for converting functions to individual coprocessors on the Cascade platform
  - Disadvantage: no parallelism between coprocessor and base processor; coprocessor is a separate component on the bus

Contributions

- Coprocessors are generally separate components from the main processor, connecting via the main memory bus
- My contributions:
  - Coprocessors can operate loops in multiclock cycles
  - Maximum parallelism
  - No limit on loop size
  - Minimize resource usage; reducing area usage
  - Methodology to generate such a coprocessor
  - Reduction in communication overhead
  - Accurate prediction to determine the improvement of the code segment given a certain constraint and architectural configuration

Project Tools

- Rapid Embedded Hardware/Software System Generation [Peddersen J., Shee S. L., Janapsatya A., Parameswaran S.] presented at the 18th IEEE/ACM International Conference on VLSI Design, January 2005
  - Uses ASIPmeister to generate core then adapts the RTL to complete the processor
  - Include and exclude any instructions
  - Automatic generation of Application Specific Instruction Set
  - Implements the Portable Instruction Set Architecture (PISA)
  - Part of the SimpleScalar framework
  - Support for extended instructions
  - Contribution:
    - A full SimpleScalar architecture (integer) processor core (synthesizable into SOC or FPGA for prototyping)
    - A novel approach to generate a processor with various subsets of instructions

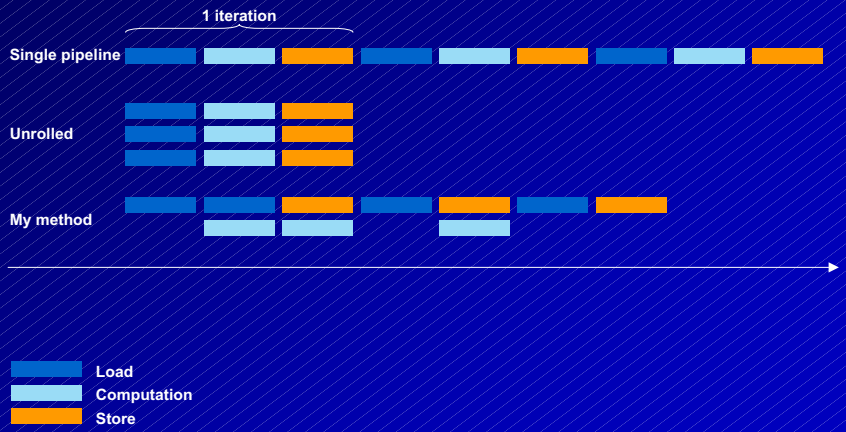
More Tools

- Modified SimpleScalar Toolset to support SYSCALL of SS CPU
  - Take advantage of cache & memory features in SimpleScalar
  - Matches clock cycle count of hardware version
  - Provides memory dump support
- Loop detection software
  - To detect most frequently occurring outer most loops.
  - Refers back to the line numbers in the C source code.
  - “Dynamic Characteristics of Loops”, [Kobayashi M. 1984]
- Memory dump file analyser
- Hot Function Detector
  - Provides the statistics of how much time is spent in each function

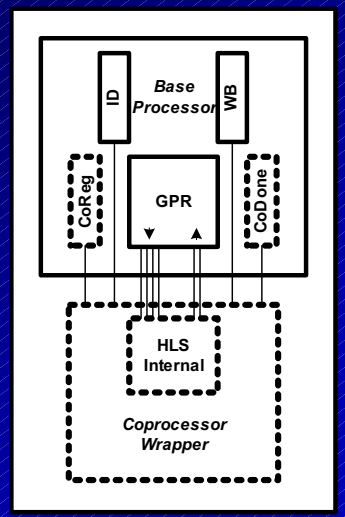
High Level Synthesis Approach

- Previous tools used: SUIF, MACHsuif (particularly for unrolling loops)
- Use SPARK for coprocessor creation (inner control)
  - a C-to-VHDL high-level synthesis framework that employs a set of innovative compiler, parallelizing compiler, and synthesis transformations
  - takes behavioural ANSI-C code as input, schedules it using speculative code motions and loop transformations, runs an interconnect-minimizing resource binding pass and generates a finite state machine for the scheduled design graph. A backend code generation pass outputs synthesizable register-transfer level (RTL) VHDL
- SPARK : A High-Level Synthesis Framework For Applying Parallelizing Compiler Transformations [Gupta2003]

How improvements are obtained



Integration

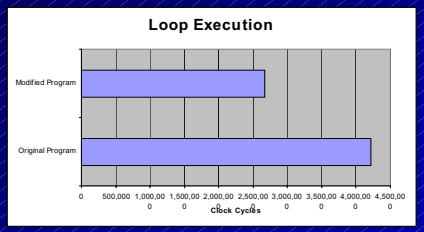


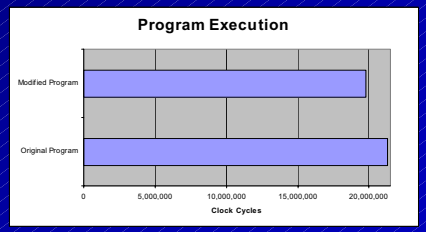
HLS Coprocessor Features

- Register file sharing
- A wrapper to control the execution of inner coprocessor
- SCPR & BCPR Instructions
- Disadvantages:
  - Can only read from destination register after write back stage; latency $\propto$ number pipeline stages
  - Very hard to make loops if input always need to be fetched every time
  - Have to make wrapper all the time just to accommodate SPARK generated component
  - Number of input / outputs = number of arguments

More details

- Detect loop hotspot in cjpeg program
- Created coprocessor using HLS Approach
- Simulated using ModelSim





- Synthesized using tcbn90gwc technology libraries through SYNOPSYS design compiler
- Given a 10ns clock constraint:
- 416.7MHz; 6,199 μm^2; 2,562 NAND gates

Why HLS approach was used

- Used to unroll loops

```
for (i = 0; i < 100; i++)
  g ();
```

```
for (i = 0; i < 100; i += 2)
{
  g ();
  g ();
}
```

- To find out how much parallelism can be obtained
- Parallelism is limited by how many register ports that can be read at any one time
- Area usage & power of register file increases linearly with increasing number of ports

GPR configuration	2 reads 1 write	4 reads 2 writes	5 reads 3 writes	8 reads 4 writes
NAND gates	19,185	27,813	34,101	42,432

- However, loop unrolling will only be beneficial if the fetches / stores are done in parallel
- We need multiple resources, but we only have 1 base processor! Bottleneck!
- No need to fetch data at the last moment

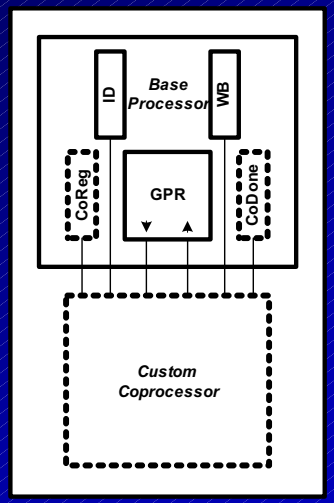
Customized Architecture

- Highly integrated coprocessor architecture
- Something like a coprocessor but integrated within the base processor
- Make full used of unused registers (r8 – r15, r24-r25)
- All calculations in the loop (when possible) are done in coprocessor
- Base processor just fetches the required data from memory and store the result back to memory
- Coprocessor taps into signal to know when data is ready and when to start execution
- Assumptions:
  - No multitasking
  - No preemption, no interrupts
  - Coprocessor does not stall CPU; will already know how long it would take at creation time; use NOPs
- Problems:
  - Latency $\propto$ pipeline stages
  - Not good for loops with short / simple computations

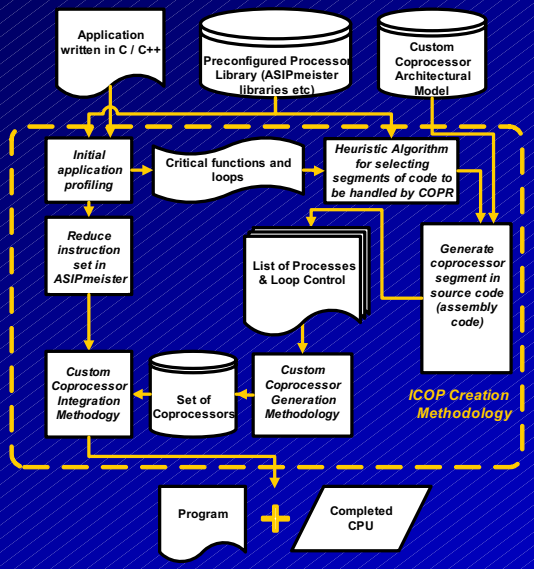
Advantages

- Save register usage
- Fetch data immediately when it is ready at WB stage
- Easy coprocessor task generation; basic block grouping
- Full control of instruction synthesis
- Maximize parallelism; address calculations are also performed
- Memory I/O task given to base processor
- No branch calculations

Customized Coprocessor Integration



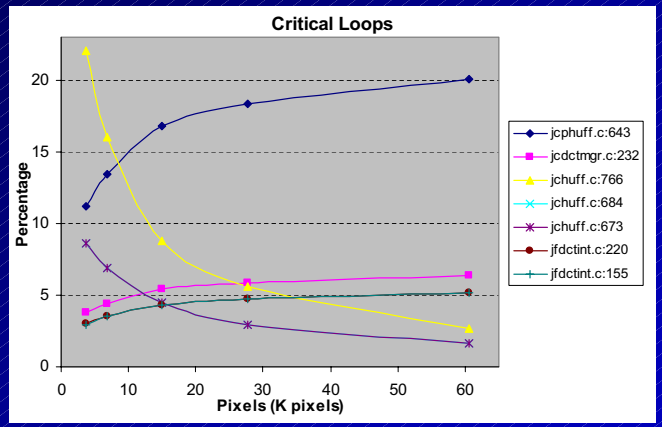
Custom Coprocessor Creation Methodology



Verification Methodology

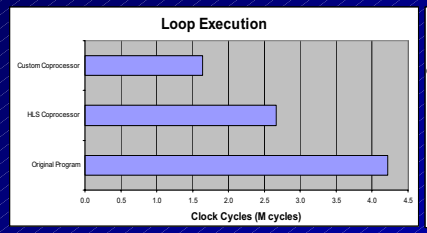
- C/C++ program is run through hardware simulation (ModelSim) and software simulation (SimpleScalar)
- Memory dump file and execution time produced by both simulations should be identical.
- Same method is applied for verification of ICOP architecture
- Sim-hexbin (program developed) is used to obtain output file from dump file for comparison purposes

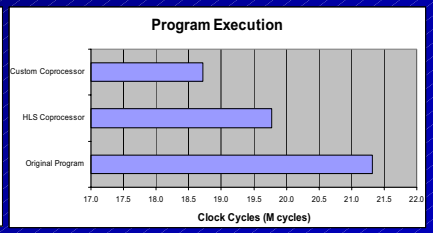
Loop Identification



How did we fair?

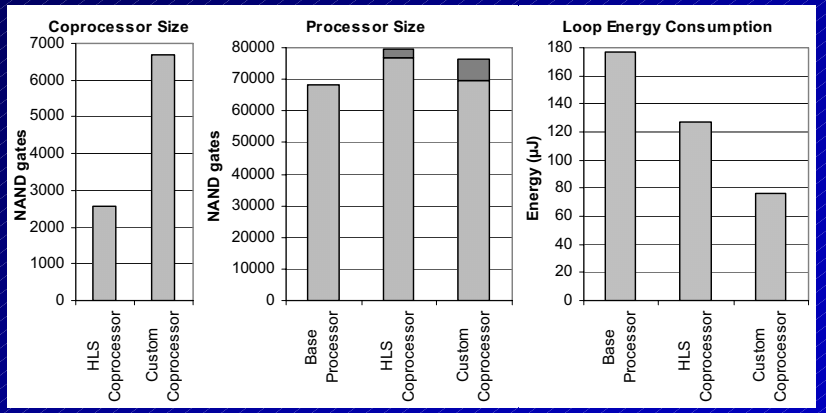
- Detect (same as previous) loop hotspot in cjpeg program
- Created coprocessor using Custom Coprocessor Methodology
- Simulated using ModelSim



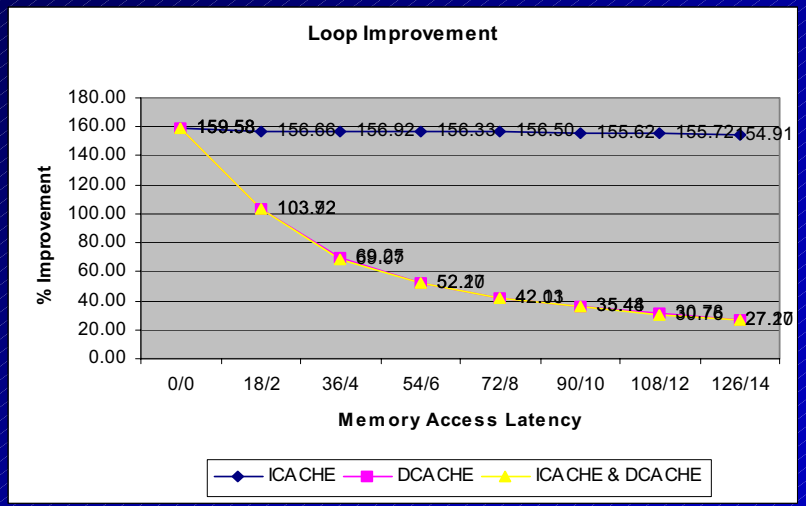


- Synthesized using tcbn90gwc technology libraries through SYNOPSYS design compiler
- Given a 10ns clock constraint:
- 166.9MHz (1 GHz possible); 16,203 μm²; 6,698 NAND gates
- Has potential to acquire less area

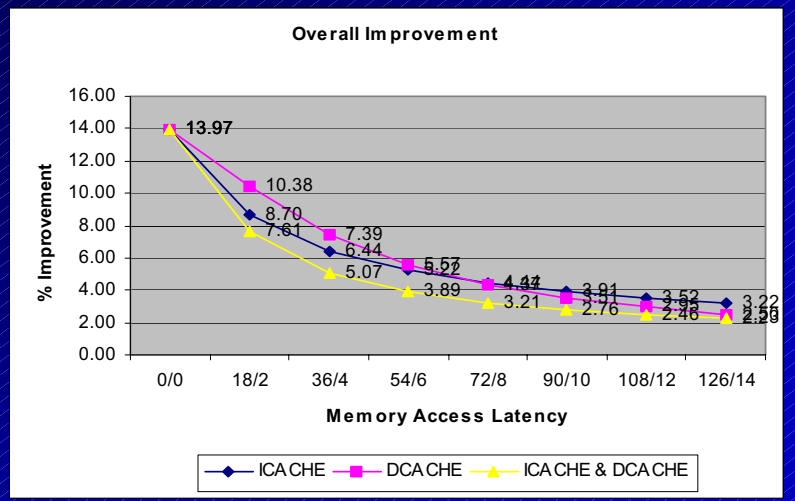
HLS vs Custom Coprocessor



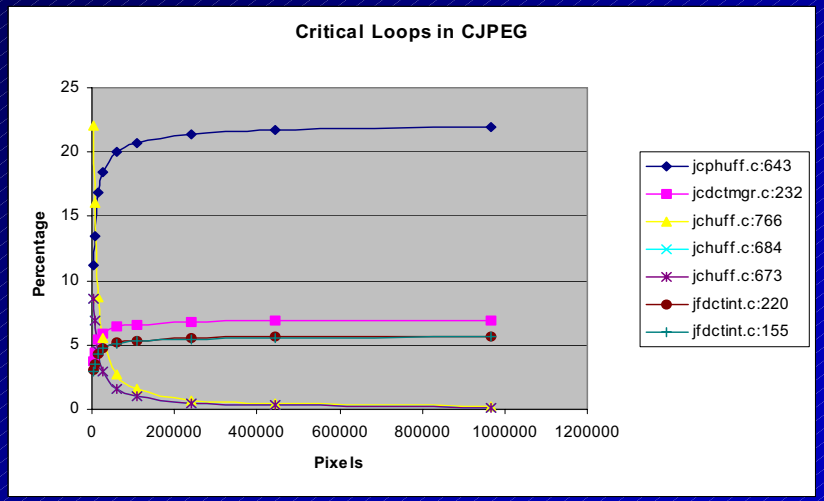
Memory Latency Effect



Memory Latency Effect



Input Data Behaviour



Future Work

- Formalize methodology
- More concrete model of coprocessor
- Model to predict performance improvement
- Able to decide when is ICOP architecture feasible
- Analyze performance improvements on work on a variety of benchmark applications

Thank you