CHIMAERA: A High-Performance Architecture with a Tightly-Coupled Reconfigurable Functional Unit

Kynan Fraser

- DISCLAIMER

This presentation is based on a paper written by Zhi Alex Yi, Andreas Moshovos, Scott Hauck and Prithiviraj Banerjee. The paper is as named in the title.

All proposals, implementation, testing, results and figures and tables have been done by the aforementioned peoples.

These slides however have been produced by me for educational purposes.

Outline

- Background
- Introduction
- Chimaera architecture
- Compiler support
- Related work (not covered)
- Evaluation
  - - methodology
  - - modelling RFUOP latency
  - - RFUOP analysis
  - - working set of RFUOP's
  - - performance measurements
- Summary

Background

Customized vs Flexibility
Benefits vs Risks

Reconfigurable solution??
Multimedia platforms

Introduction

CHIMAERA
Reconfigurable hardware and compiler

- Coupled RFU (Reconfigurable Functional Unit)
- Implements application specific operations
- 9 inputs to 1 output
- Fairly simple compiler

Introduction – Potential Advantages

- Reduce execution time of dependent instructions
  - tmp=R2−R3; R5=tmp+R1
- Reduce dynamic branch count
  - if (a>88) a=b+3
- Exploit subword parallelism
  - a = a + 3; b = c << 2 (a,b,c halfwords)
- Reduce resource contention

Chimaera Architecture

- Reconfigurable Array
  - programmable logic blocks
- Shadow Register File
- Execution Control Unit
- Configuration Control and Caching Unit

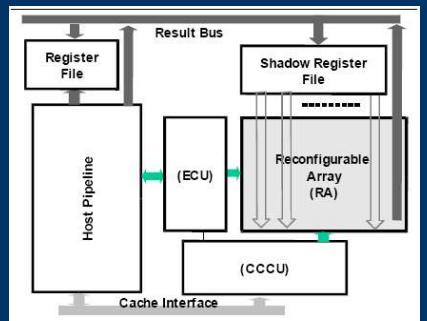

Figure 1: *Overview of the Chimaera Architecture*

Chimaera Architecture

- RFUOP unique ID
- In-order commits
- Single Issue RFUOP scheduler
- Worst case 23 transistor levels (for one logic block)

10 bits	*5 bits*	*17 bits*
RFUOP opcode	Destination register	RFUOP number

(a)

32 bits	*32bits*	*1674 bits*	...	*1674 bits*
Input register vector	# of rows	Row 1 configuration	...	Row n configuration

(b)

Figure 3: *(a) RFUOP instruction format. (b) RFUOP configuration data layout.*

	r1 + r2	r1 + r2 << 2	r1 + (r2 & 5)	if (r1>r2) r5=r3+r4
Critical path (transistors)	19	20	19	23
RA rows	1	1	1	2
Dataflow graph height	1	2	2	2
Latency (processor cycles)	1.58	1.67	1.58	1.96

Table 1: *Critical path through the RFU's RA for some operations assuming 12 transistor levels per processor cycle.*

Compiler Support

- Automatically maps groups of instructions to RFUOP's
- Analyses DFG's
- Schedules across branches
- Identifies sub-word parallelism (disabled in this case due to endangered correctness)
- Look later at how many can instructions actually map to RFUOP's

Related Work

- We are looking at it
- Read section 4 for more information

Configuration

Component	Configuration
	Superscalar Core
Branch predictor	*64k GSHARE*
Scheduler	*Out−of−order issue of up to 4 operations per cycle, 128 entry re−order buffer (RUU), 32 entry load/store queue(LSQ)*
Functional units	*4 integer ALUs, 1 integer MULT, 4 FP adders, 1 FP mult/div*
Functional unit latencies	*Integer ALU 1, integer MULT 3, integer DIV 12, FP adder 12, FP MULT 4, FP DIV 12, load/store 1*
Instruction cache	*32kb Direct−Mapped, 32−byte block, 1 cycle hit latency*
Data cache	*32kb Direct−Mapped, write−back, write−allocate, non−blocking, 32−byte blocks, 1 cycle hit latency*
L2 cache	*Unified 4−way set associative, 128k byte, 12 cycles hit latency*
Main memory	*Infinite size, 100 cycles latency*
Fetch Mechanism	*Up to 4 instructions per cycle*
	Reconfigurable Functional Unit
Scheduler	*8 entries. Each entry corresponds to a single RFUOP* *Single Issue, Single Write−back per cycle.* *An RFUOP can issue if all its inputs are available and no other instance of the same RFUOP is currently executing.*
Functional Unit / RA	*32 rows. Each RFUOP occupies as many rows as instructions of the original program it replaced (pessimistic)* *Only a single instance of each RFUOP can be active at any given point in time.*
Configuration Loading	*1−st level configuration cache of 32 configuration rows (32 x 210 bytes).* *Configuration loading is modeled by injecting accesses to the rest of the memory hierarchy. Execution stalls for the duration of configuration loading.*
RFUOP Latency	*Various model simulated. See Section 5.1.1.*

Table 2: *Base configuration for timing experiments.*

Evaluation - Methodology

- Execution driven timing
- Built over simplescalar
- ISA extension of MIPS
- RFUOP's appear as NOOP's under MIPS ISA
- Previous slide configuration used

Benchmark	*Description*	*Inst. Count*
	MediaBench Benchmarks	
Mpegenc	Mpeg encoder	1139.0 M
G721enc	CCITT G.721 voice encoder	309.0 M
G721dec	CCITT G.721 voice decoder	294.0 M
Adpcm enc	Speech compression	6.6 M
Adpcm dec	Speech decompression	5.6 M
Pegwitkey	Pehwit key generation. Pegwit is a public key encryption and authentication application.	12.3 M
Pegwitenc	Pegwit encryption	23.9 M
Pegwitdec	Pegwit decryption	12.5 M
	Honeywell Benchmarks	
Comp	Image compression	34.1 M
Decomp	Image decompression	32.7 M

Table 3: *Benchmark characteristics.*

Evaluation – Modelling RFUOP Latency

- First row modelled on original instruction sequence critical path
- Second row modelled on transistor levels and delays

Original Instruction–based Models						
Model	C	2C	3C	1	2	N
CPU cycles	c	2*c	3*c	1	2	n
Transistor–Level–based Models						
Model	P24_0		P24_1	P12_0	P12_1	
CPU cycles	⌈t/24⌉		⌈t/24⌉+1	⌈t/12⌉	⌈t/12⌉+1	

Table 4: *RFUOP latency models. "c" is the critical path length of the original dataflow graph an RFUOP replaces. "n" is the number of the original instructions replaced by each RFUOP. "t" is the number of transistor levels in an RFUOP.*

Evaluation – RFUOP Analysis

- Total number of RFUOP's per benchmark
- Frequency of instruction types mapped to RFUOP's

Bench	IC	Red.	Branch		Add/Sub		Logic		Shift	
	Opt.		Orig.	Opt.	Orig.	Opt.	Orig.	Opt.	Orig.	Opt.
Adpcmenc	81%	34%	27%	37%	41%	31%	10%	46%	15%	46%
Adpcmdec	53%	58%	30%	59%	29%	57%	18%	77%	14%	72%
Mpegenc	90%	12%	17%	13%	47%	19%	0%	0%	3%	31%
G721enc	94%	8%	22%	4%	41%	5%	3%	32%	12%	35%
G721dec	92%	9%	23%	5%	41%	5%	3%	32%	11%	41%
Pegwitkey	85%	22%	15%	16%	37%	33%	13%	3%	11%	67%
Pegwitenc	85%	22%	15%	16%	37%	33%	12%	2%	10%	67%
Pegwitdec	85%	22%	15%	16%	37%	33%	13%	3%	11%	67%
Honeyenc	83%	28%	13%	18%	51%	36%	1%	0%	9%	88%
Honeydec	88%	21%	13%	0%	47%	27%	0%	51%	10%	82%
Average	84%	22%	12%	18%	41%	28%	7%	25%	10%	60%

Table 5: *Global Instruction Count Statistics.*

Evaluation – RFUOP Analysis

- Look at how many instructions replaced by RFUOP
  - dest = src1 op src2 op src3
  - 3/4 input/1 output most
- Look at critical path of instructions replaced

	1	2	3	4	5	6	7	...	16	17
adpcmenc	0%	60%	27%	0%	0%	13%	0%	0%	0%	0%
adpcmdec	0%	50%	0%	0%	0%	25%	0%	0%	0%	25%
mpegenc	0%	0%	0%	32%	0%	31%	0%	0%	37%	0%
g721enc	0%	52%	0%	16%	0%	32%	0%	0%	0%	0%
g721dec	0%	55%	0%	15%	0%	30%	0%	0%	0%	0%
pegwitkey	0%	57%	20%	0%	4%	0%	20%	0%	0%	0%
pegwitenc	0%	55%	21%	0%	2%	0%	21%	0%	0%	0%
pegwitdec	0%	57%	20%	0%	4%	0%	20%	0%	0%	0%
honeyenc	0%	70%	30%	0%	0%	0%	0%	0%	0%	0%
honeydec	0%	58%	33%	9%	0%	0%	0%	0%	0%	0%
Average	0%	51%	15%	7%	1%	13%	6%	0%	4%	3%

Table 6: *RFUOP distribution in terms of original instruction count. Range shown is 1 to 17 instructions (columns ommitted have 0% in all rows).*

	1	2	3	4	5	6	7	8
adpcmenc	39%	47%	0%	13%	0%	0%	0%	0%
adpcmdec	0%	50%	0%	25%	0%	0%	0%	25%
mpegenc	0%	0%	0%	32%	0%	69%	0%	0%
g721enc	0%	68%	0%	16%	0%	16%	0%	0%
g721dec	0%	70%	0%	15%	0%	15%	0%	0%
pegwitkey	0%	57%	24%	20%	0%	0%	0%	0%
pegwitenc	0%	57%	23%	21%	0%	0%	0%	0%
pegwitdec	0%	57%	23%	20%	0%	0%	0%	0%
honeyenc	0%	83%	17%	0%	0%	0%	0%	0%
honeydec	0%	58%	36%	6%	0%	0%	0%	0%
Average	4%	55%	12%	17%	0%	10%	0%	3%

Table 7: *RFUOP distribution in terms of the critical path of the original dataflow graph. Range shown is 1 to 8 instructions.*

Evaluation – Working set of RFUOP's

- Larger working set = more stalls to configure
- Maintaining 4 MRU almost no misses
- 16 rows sufficient

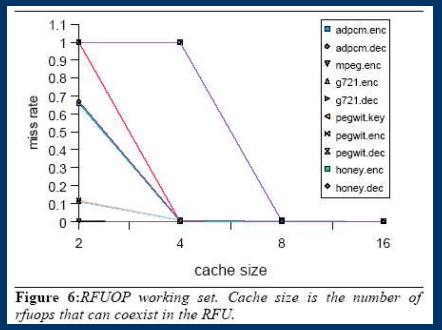

Figure 6: *RFUOP working set. Cache size is the number of rfuops that can coexist in the RFU.*

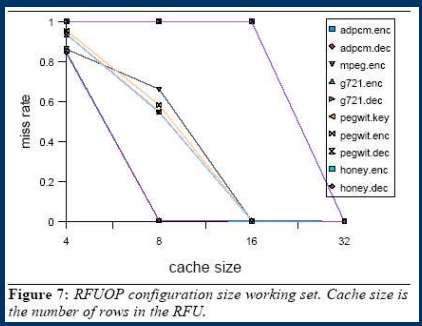

Figure 7: *RFUOP configuration size working set. Cache size is the number of rows in the RFU.*

Evaluation – Performance Measurements

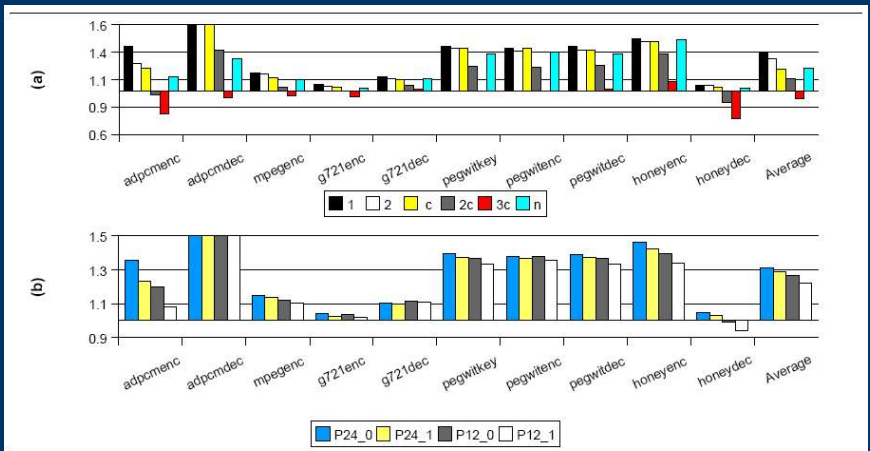

Figure 8: *Relative performance over the 4−way base configuration. (a) Original−instruction−based timing models (The adpcm.dec bars for 1, 2 and C are truncated. The measurements are 3.52, 2.51, and 2.55). (b) Transistor−level−based timing models (The adpcm.dec bars are truncated. The measurements are 2.88, 2.97, 2.56, 2.31 from left to right).*

Evaluation – Performance Measurements

- Performance under original instruction timing latencies (4 issue)
- Latency of 2C or better still give speed of 11% or greater, 3C not worthwhile
- 3C not worthwhile (only speedup under one benchmark)
- N model improves performance overall
  - due to decreased branches and reduced resource contention

Evaluation – Performance Measurements

- Performance under transistor timing (4 issue)
- Improvements of 21% even under most conservative transistor timing
- Performance in optimistic models close to 1-cycle model (upper bound)

Evaluation – Performance Measurements

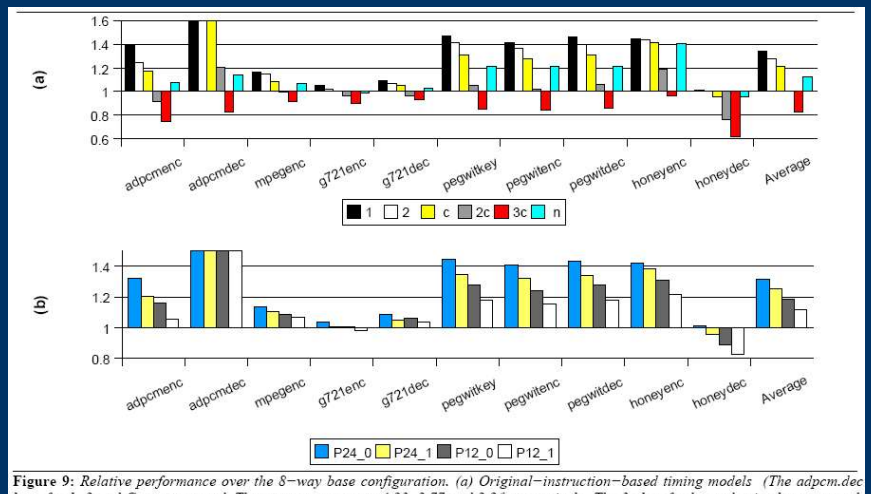

Figure 9: *Relative performance over the 8−way base configuration. (a) Original−instruction−based timing models (The adpcm.dec bars for 1, 2 and C are truncated. The measurements are 4.23, 2.77 and 2.26 respectively. The 3c bar for honeydec is also truncated. The number is 0.62). (b) Transistor−level−based timing models (The adpcm.dec bars are truncated. The measurements are 3.59, 3.35, 2.26 and 2.04 from left to right).*

Evaluation – Performance Measurements

- Performance with 8 issue
- Only improvements with C, 1, 2 and N timing
- Relative improvements (to 4 issue) small
- Reason: Because limited to one RFUOP issue per cycle

Evaluation – Performance Measurements

Bench	IC	Red.	Branch		Add/Sub		Logic		Shift	
	Opt.		Orig.	Opt.	Orig.	Opt.	Orig.	Opt.	Orig.	Opt.
Adpcmenc	81%	34%	27%	37%	41%	31%	10%	46%	15%	46%
Adpcmdec	53%	58%	30%	59%	29%	57%	18%	77%	14%	72%
Mpegenc	90%	12%	17%	13%	47%	19%	0%	0%	3%	31%
G721enc	94%	8%	22%	4%	41%	5%	3%	32%	12%	35%
G721dec	92%	9%	23%	5%	41%	5%	3%	32%	11%	41%
Pegwitkey	85%	22%	15%	16%	37%	33%	13%	3%	11%	67%
Pegwitenc	85%	22%	15%	16%	37%	33%	12%	2%	10%	67%
Pegwitdec	85%	22%	15%	16%	37%	33%	13%	3%	11%	67%
Honeyenc	83%	28%	13%	18%	51%	36%	1%	0%	9%	88%
Honeydec	88%	21%	13%	0%	47%	27%	0%	51%	10%	82%
Average	84%	22%	12%	18%	41%	28%	7%	25%	10%	60%

Table 5: *Global Instruction Count Statistics.*

Evaluation – Performance Measurements

- Strong relationship between performance improvement and branches replaced by RFUOP's
- Benchmarks with lowest branch reduction have lowest speedup
- Even under pessimistic assumptions Chimaera still provides improvements

Summary

- Seen the CHIMAERA architecture
- The C compiler that generates RFUOP's
- Maps sequences of instructions into single instruction (9input/1output)
- Can eliminate flow control instructions
- Exploits sub-word parallelism (not here)
- 22% on average of all instructions to RFUOP's
- Variety of computations mapped
- Studied under variety of configurations and timing models

Summary

- 4 way: average 21% speedup for transistor timing (pessimistic)
- 8 way: average 11% speedup for transistor timing (pessimistic)
- 4 way: average 28% speedup for transistor timing (reasonable)
- 8 way: average 25% speedup for transistor timing (reasonable)