

# Power- and Performance-Aware Architectures

**PhD. candidate:** Ramon Canal Corretger


**Advisors:** Antonio González Colás (UPC)  
James E. Smith (U. Wisconsin-Madison)

Departament d'Arquitectura de Computadors  
Universitat Politècnica de Catalunya



# Introduction

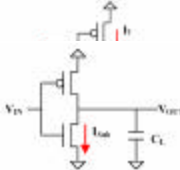

- Increasing miniaturization**
  - Portable handhelds (PDAs, notebooks, cell phones, etc.)
- Increasing computation requirements**
  - Multimedia processing (audio, video)
    - Speech-recognition
    - Audio and video encoding/decoding
  - Wireless communication
- Increasing demand for**
  - Longer battery life
  - Smaller heat dissipation



# Motivation

**Design for low power**

- Becoming a critical design constraint
- Types of energy consumption
  - Dynamic (predominant source for current technology)
    - Charging/Discharging capacitors


# Introduction

**Design goal:**

**POWER EFFICIENCY**


**Metrics used:**

- Activity counts
- Energy consumption
- Energy-delay<sup>2</sup>



# Main Contributions

- Issue logic designs**
  - Dependence-tracking schemes
  - Prescheduling schemes
- Ultra-low power processors**
  - Novel microarchitecture
  - Alternatives for different energy/delay
- Low power in high-performance proc.**
  - Hardware value compression
  - Software value compression




# Issue Logic - Motivation

**Base alternatives**

- In-order (limited performance, low complexity)
- Out-of-order (high performance, huge complexity)
  - Roadmap to high performance
    - Large instruction windows
    - Lots of functional units
  - Associative lookups in issue
    - Due to a broadcast and select mechanism
  - Instruction issue in the critical path
  - Long delays due to hardware size and complexity

**Can we have the best of both?**  
(high performance, low complexity)



## Issue Logic Designs

issue logic

Queue

write

Value

opns

GCL

Future

work

UPC

**Dependence tracking schemes**

- Keep a direct relationship between producers and consumers

**Prescheduling schemes**

- Most latencies are known so schedule instructions so that they are just considered once for select

## Characteristics exploited

issue logic

Queue

write

Value

opns

GCL

Future

work

UPC

**Dependence tracking ® Output register usage<sup>1</sup>**

- 22% of the values produced by the SpecInt95 and
- 25 % of the values produced by the SpecFP95 are read more than once.
  - Can avoid associative lookup if the non-ready instructions are kept in a table index by physical register.

**Prescheduling ® Latency of operations known**

- Every FU latency is known (except for memory accesses)
  - Output register availability time can be computed
  - Instructions can be "tagged" with its issue cycle
  - Assume memory accesses have a fixed latency or a special management

<sup>1</sup> J.L.L. Cruz, A. González, M. Valero, N. Topham. Multiple-Banked Register File Architectures. Proc. of the 27th Int' Symp. on Computer Architecture, June 2000, pp.316-324.

## Experimental Framework

issue logic

Queue

write

Value

opns

GCL

Future

work

UPC

**Simulator**

- based on SimpleScalar
- + issue mechanisms

**Benchmarks**

- Spec95 benchmark suite

**Configuration**

- 8-way issue (4 int + 4 FP)
- ROB of 64 entries

## N-Use Scheme (i)

issue logic

Queue

write

Value

opns

GCL

Future

work

UPC

**Basic Concept**

- Avoid an associative lookup by keeping the instructions with its producers.
- Just N instructions are kept for physical register. When the register becomes available the instruction/s is/are sent to the ready queue.
- Hopefully, most of the instructions will be ready when dispatching or (as the statistics say) within the first N uses of its source operands.

## N-Use Scheme (ii)

issue logic

Queue

write

Value

opns

GCL

Future

work

UPC

**Example**

- LD R1, 0(R4)
- LD R2, 0(R5)
- ADD R3, R1, R2
- MUL R4, R1, R3
- ST 0(R6), R4

R4, R5 and R6 are assumed to be ready

## Prescheduling Schemes (i)

issue logic

Queue

write

Value

opns

GCL

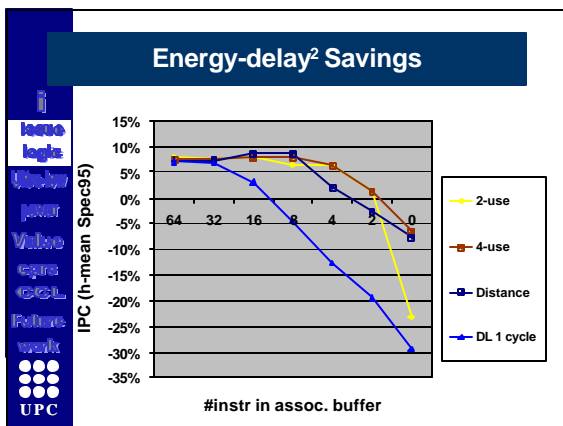
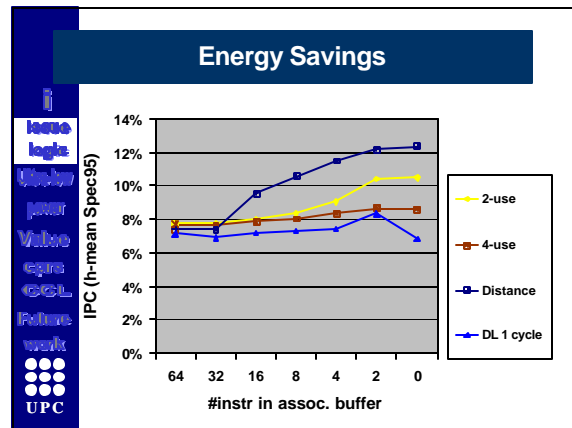
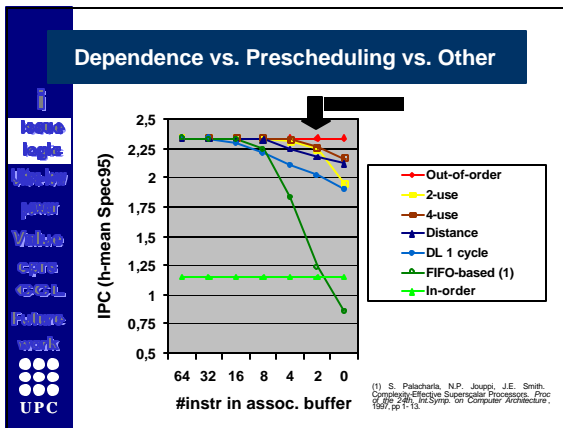
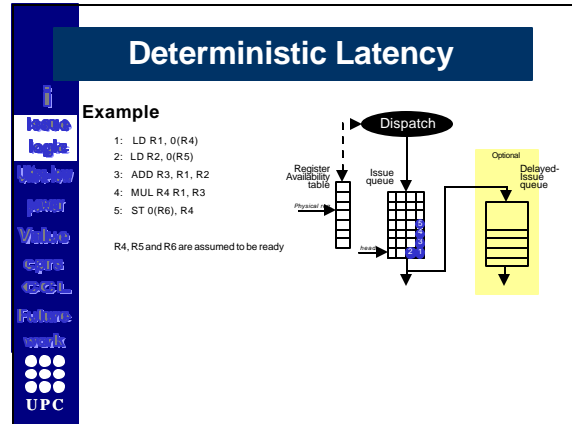
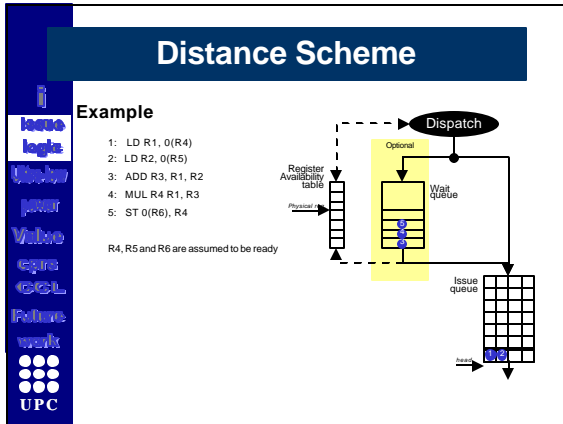
Future

work

UPC

**Basic Concept**

- Schedule instructions according to cycle where the operands will be available and when there is a free functional unit.
- When an instruction knows in which cycle it will be executed it updates the availability cycle of its output register (if any).
- Latency of most operations known
  - Memory accesses are assumed to have an unknown latency ? Distance Scheme
  - Memory accesses are assumed to have a fixed latency ? Deterministic Latency Scheme
- Similar to a "dynamic" VLIW



## Summary

**New alternatives for issue logic**

- Can eliminate associative search in the issue logic (10% IPC loss).
- Similar IPC with one eighth of the associative search.
- Increased energy-efficiency.

# Ultra-Low Power Processors

Data values set minimal activity

## Significance compression

- Effective and simple
- Pipeline wide technique

## Low power pipelines

- Basic pipeline (8-bit wide)
- Balanced pipeline
- Parallel pipeline



# Motivation

## Size of the operands

- Small values are largely used (that fit in a byte)
  - 00 00 00 01 hex
  - FF FF FF FD hex
- Large values may have bytes within that are just a sign extension of the previous one.
  - 00 7F 00 01 hex
  - 01 00 00 00 hex
- Just the significant bytes need to be computed and stored; the rest are redundant

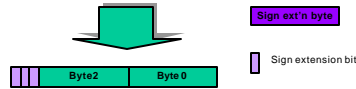


# Significance Compression

Conventional approach



Significance compression approach



Example: 0x00 7F 00 01 ? 101 | -- 7F -- 01  
 1 extension bit per byte. Byte 0 is always computed so no sign extension needed



# Significance Compression

## Size of the operands

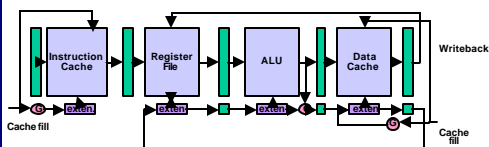
- values read/written into the register file and memory (Avg. Medi abench)

Cases	% of values	Acc	Example (hex)
eeee	61.0	61.0	00 00 00 01
eess	13.6	74.6	00 00 01 00
ssss	12.6	87.2	01 01 01 01
esss	7.4	94.6	00 01 01 01
ssee	1.8	96.4	01 01 00 01
sees	1.5	97.9	01 00 01 01
eses	1.3	99.2	00 01 00 01
sees	0.8	100	01 00 00 01

- Overall
  - 61.0% need 8 bits
  - 15.7% need 16 bits
  - 10.7% need 24 bits
  - 12.6% need 32 bits



# Basic Pipeline



## Five stage pipeline

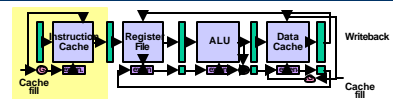
- Fetch, Decode+Read, Execution, Memory, Writeback

## Pipeline extended with extension bits

- Significance compression in all the stages of the pipeline



# Power Savings (i)



## PC increment

- Compute 1 byte at a time
- Typically, carry does not propagate between bytes

- Activity savings 73.3% average



### Power Savings (ii)

**Fetch**

- Compress instructions (from 4 to 3 bytes)
  - Remove fields not used and compress function codes
  - Compress immediates that do not use all the space reserved
- Most used instructions will use 3 bytes
  - 1 bit extra per instruction to distinguish among compressed/uncompressed instructions

- Activity savings 18.2% average

### Power Savings (iii)

**Register file (read)**

- Use significance compression when reading/writing
- Keep extension bits (3)

- Activity savings 46.5% average

### Power Savings (iv)

**ALU**

- Just compute "significant" bytes
- Generate extension bits for the result

- Activity savings 33.2% average

### Power Savings (v)

**Memory (D-cache data)**

- Just send/receive "significant" bytes
- Keep extension bits in the cache

- Activity savings 30.1% average

### Power Savings (vi)

**Memory (D-cache tags)**

- Tag compare with first byte (early miss detection)

- Activity savings 1% average

### Power Savings (vii)

**Register file (write)**

- Just write significant bytes
- Update extension bits

- Activity savings 42.1% average

## Power Savings (viii)

**Summary activity reduction**

Datapath Width	PC incr	Fetch	RF read	ALU	D-cache data	RF write	Latches
8 bits	73.3%	18.2%	46.5%	33.2%	30.1%	42.1%	42.2%
16 bits	46.7%	18.2%	35.9%	22.1%	23.4%	30.3	34.9%

## Low Power Pipelines

### Byte-serial implementation

## Summary

**Significant bytes determine minimal activity**

- For instructions, addresses and data values
- Activity is 30-40% lower than a conventional 32-bit architecture

**Proposed pipeline implementations**

- Several designs proposed
- Trading off power consumption and performance
  - Similar activity reduction levels
  - Minimal activity achieved by the serial organization
    - Less latches
    - Smaller static consumption (fewer and smaller blocks)
  - Maximum performance achieved by the parallel organizations

## Low-Power High-Performance processors

**Hardware value compression**

- Analysis of the best value compression method for a 64-bit wide datapath

**Software value compression**

- Value Range Propagation
- Value Range Specialization

**Cooperative hwd-sw approach**

## Value Compression

**Data size distribution for the Spec Integer 95 benchmarks**

Size in bytes	Percentage of occurrence
1	45%
2	15%
3	5%
4	5%
5	35%
6	2%
7	2%
8	5%

## Value Compression

**Analysis of value compression mechanisms for 64-bit architectures**

- Significance compression
  - Sign extension bytes
- Value compression
  - Leading sign extension bytes
- Zero compression<sup>1</sup>
  - Zero bytes

<sup>1</sup> L. Villa, M. Zhang, and K. Asanovic, "Dynamic Zero Compression for Cache Energy Reduction", in Proc. of the 33rd International Symposium on Microarchitecture, Dec. 2000.

## Significance Compression

32-bit value

Significance compression approach

Signext'n byte

Sign extension bit

## Size Compression

32-bit Value

Size compression approach

Signext'n byte

Sign extension bit

## Zero Compression

32-bit Value

Size compression approach

zero byte

Zero extension bit

## Experimental Framework

**Simulator**

- based on Wattch
- + data width operand gating

**Binary Optimizer**

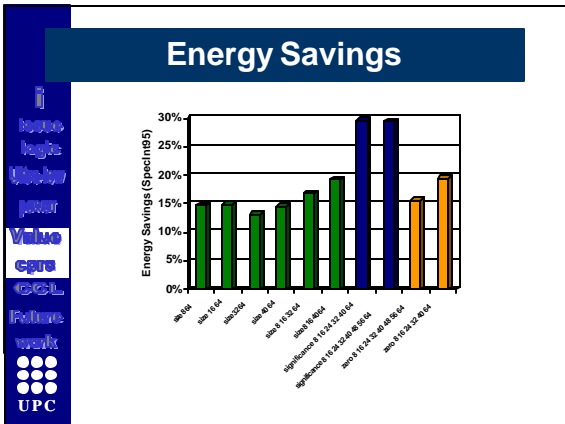
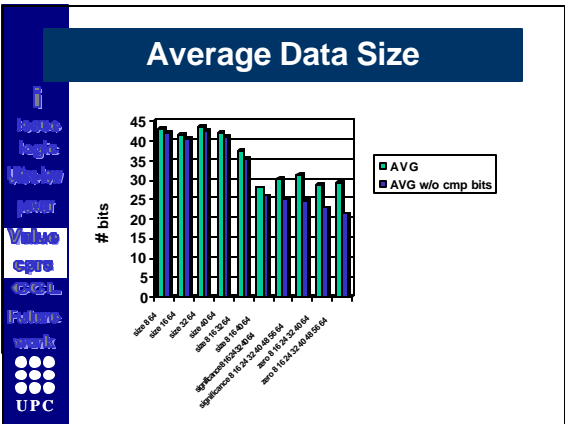
- Alto

**Benchmarks**

- Spec95 benchmark suite

**Configuration**

- 4-way issue
- ROB of 64 entries



## Software Techniques

Value  
Copro  
GCL  
Future  
work  
UPC

### Value Range Propagation

- Propagate at compile-time range of values that each instruction operand might have

### Value Range Specialization

- Using value profiling information, specialize regions of code to a certain range for a given value

## Value Range Propagation

Value  
Copro  
GCL  
Future  
work  
UPC

### Find initial value ranges

- Inst. declared with narrow operands (e.g. add\_short)
- Assignments (e.g. Variable=constant)
- "if" condition statements

### Forward propagation

- Set the range of the output operands
  - Depending on the input operands' ranges and operation

### Backward propagation

- Set the range of the input operands
  - Depending on the output operand's range, the input operands' ranges and the operation

## Value Range Propagation

Value  
Copro  
GCL  
Future  
work  
UPC

### "Useful" Range Propagation

Constrain the range of the values due to their operations

- Logical operations
  - AND R1, 0xFF, R2 (i.e. R1?R2&0xFF)
  - OR R1, 0xFFFFFFFF00000000, R2 (i.e. R1?R2|0xFFFFFFFF00000000)
- Mask operations
- Limited width fields
  - shift amounts

## Value Range Propagation

Value  
Copro  
GCL  
Future  
work  
UPC

### Example of Value Range Propagation

- a0=<NTmin,INTmax>
- a1=<0,0>
- a3=<0,0>
- a2=<NTmin,INTmax>
- a1=<1,1>
- tripcount=100
- a1=<1,100>

```

a0 = 0a
a1 = 0
a3 = a1*4
a2 = a0+a3
mem[a2] = a1
a1 = a1+1
a1 < 100
return
            
```

a3 = <0, 396>      9.

a1in = <0,99>      8.

Original C code:

```

for (i=0; i<100; i++) {
    a[i]=1;
}
            
```

## Value Range Specialization

Value  
Copro  
GCL  
Future  
work  
UPC

### Profiling based compile-time technique

#### Three step process:

- Identification of instructions (candidates) where specialization may be profitable using basic block profiles (i.e. basic block counts)
- Computation of the ranges of values for the candidates with the support of profiling data
- Specialization of the candidates that are deemed profitable

## Value Range Specialization

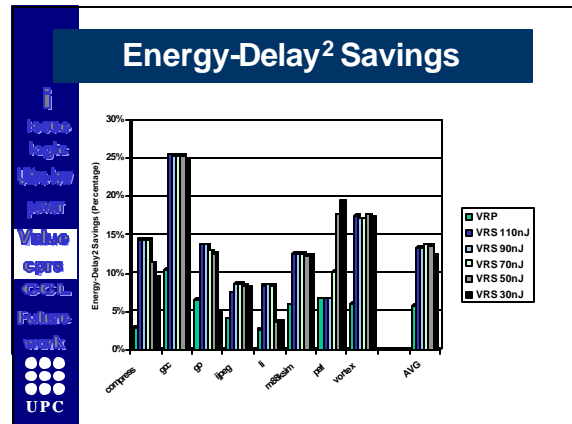
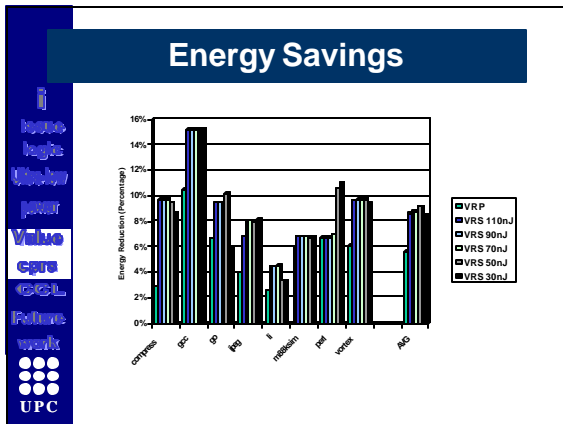
Value  
Copro  
GCL  
Future  
work  
UPC

Candidates

Savings

Specialization

3. Specialization of the candidates that are deemed profitable according to their savings



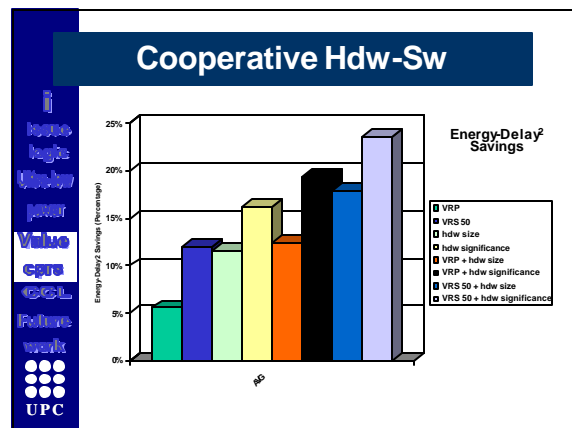
### Cooperative Hdw-Sw

**Advantages Hdw**

- ISA untouched
- Actual values

**Advantages Sw**

- Minimal hdw complexity
- Wider program scope




### Summary

- Value compression shown to be an effective way of reducing power requirements
- Software not so effective but less complex
- Cooperative hdw-sw for more effectiveness

### Conclusions

**Investigated several alternatives to increase power-efficiency**

- Novel issue logic designs that increase the energy -efficiency of this stage of the pipeline
- Novel ultra-low power microprocessor. Several implementations studied for different set of energy/delay tradeoffs.
- Novel software and hardware value compression methods for high-performance processors.



## Publications

**Three issue logic proposals based on two different approaches:**


- Dependence-tracking (ICS'00 and ICS'01)
- Prescheduling (ICS'00 and ICS'01)
- Survey on issue queue designs (IEEE Micro 2003)

**Several ultra-low power pipelines designs**

- Based on significance compression (MICRO-33)

**Low-power high-performance processors**

- Hardware value compression (TR UPC-DAC-2003-32)
- Value Range Propagation (CGO-2004)
- Value Range Specialization (CGO-2004)



## Future Work

**Static Power Consumption**

- Reduce leakage currents

**Temperature-Aware Computing**

- Where temperature sets performance

**System on a Chip**

- Mobile devices where memory and several communication protocols are integrated on a chip



## Q & A