Universidade do Minho – Departamento de Informática

Power Aware Techniques: Extensions to ISAs

Eva Oliveira

ICCA 04

Introduction: The decade of Power Aware

- A wide range of current and new technologies employ low-power systems
- In the last few years, hardware was largely improved to reach better (low) energy levels

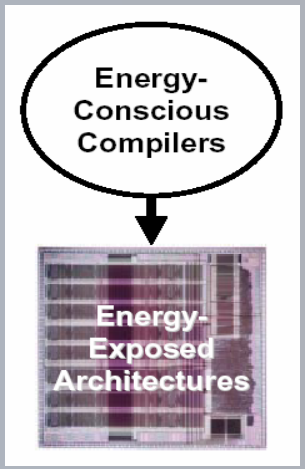
- Now is time to look for compilers that solve the challenging problem of creating power efficient and high-performance software

Where does the power go?

- Implementations of modern RISC/VLIW ISAs perform a large number of microarchitectural operations for each instruction
  - For integer add instruction on 5-stage RISC pipeline only ~2% of energy is the 32-bit adder circuit itself
  - Rest includes cache tags and data, TLBs, register files,pipeline registers, exception state management
- No incentive to **expose** these microarch ops in a purely performance-oriented ISA

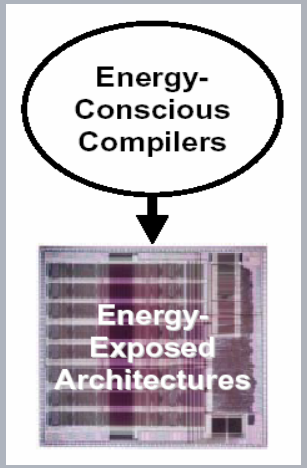
Energy-Exposed ISA's

- An Investigation of processor power consumption must be performed at the most elementary level – the instruction level
- Some authors proposed energy-exposed hardware-software interfaces to give software more fine-grain control over energy consuming microarchitectural operations



Energy-Exposed ISA's

- A RISC microprocessor was modified to support the three techniques proposed
- Compiler algorithms were developed to target the enhanced instruction set



Energy-Exposed techniques

- Software restart markers: improves exception state managment, dividing the instruction stream into restartable regions
- Bypass latches: eliminate register files traffic
- Tag unchecked loads and stores : optimize the hardware tag check time by eliminating it

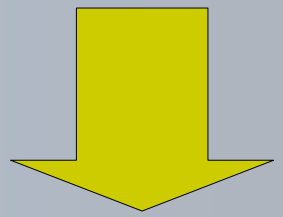
Software Restart Markers

- Current pipelined machines invest significant energy in preserving precise exception semantics
- Instructions results are buffered before committed in order, requiring register renaming logic to find the correct value for new instructions
- Even a simple five-stage RISC pipelined has a bypass network

Software Restart Markers

- Actual machines provide some mechanisms to manage exceptions
- Precise exceptions are supported in a pipelined machine
  - hardware must either buffer updates in form of future files until all possible exceptions have cleared, or...
  - save old machine state in history buffers, so that it can be recalled when exceptions are detected

Software Restart Markers



- These schemes add additional exceptions state managment energy overhead to the executions of all instructions

Software Restart Markers: Compiler Analysis

- **SRM** reduce energy cost of exceptions managment by requiring software to explicity divide the instruction stream into restartable regions

Bypass Latches

- Half of the values written to the register file are used exactly once, usually by the instruction executed immediatly after the one producing the value.

```
lw    r1, (r3)        load value
add r1, r1 ,  1       increment
```

Bypass Latches

- Giving software explicity control of the bypass latches, it is possible to reduce the register file traffic considerably

```
lw,   RS, (r3)
add SD, RS, 1
```

Same performance, but writes and reads have been avoided and replaced with accesses to the bypass latches

Bypass Latches

- **Reduced register file activity**
  - only write to bypass latches, not regfile
  - reduce reads from reg file on avergae 28%
- On average, 34% of all writes are eliminated

Tag-Unchecked Loads and Stores

- Memory system, including caches, consumes a significant fraction of system power
- Tag check in the primary data cache is one significat source of energy consumption

Direct addressing allows software to cache data without the hardware performing a cache tag check

Tag-Unchecked Loads and Stores

- The compiler often knows when the program is accessing the same piece of memory. Don't check the cache tags for the second access
- **HW challenge — make this path low power**
- SW challenge — find the opportunities for use.
  - Compiler algorithms for C languages
- **Interface challenge — minimize ISA changes, don't disrupt HW, don't expose too much HW detail.**

Compiler Algorithm (C)

- Loop unrolling to increase aligned references
- An array of 64 bits-data and the cache line size of 32 bytes

```
for(i=0; i<N; i++) {
   A[i] = 0;
}
```

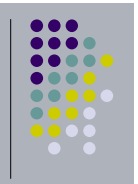
```
for(i=0; i<N; i++) {
   if(&A[I] % line_size == 0)
      break;
   A[I] = 0;
}
for(; i<N; i += 4) {
   A[i + 0] = 0; A[i + 1] = 0;
   A[i + 2] = 0; A[i + 3] = 0;
}
```

- Data cache energy reduction 8.7 - 40%

Conclusions

- Instructions perform many hidden microarchitectural operations as they execute
- Compile-time analysis can statically determine that much of the work is unnecessary
- By providing an energy-exposed instruction set, this analysis information can be transmitted to the hardware to save energy without impacting performance

Conclusions



- Software restart markers reduce this overhead by enabling the introduction of temporary state that does not have to be saved and restored across exceptions.
- Exposed bypass latches are an example of allowing software to make use of temporary state to avoid microarchitectural operations at run time; in this case register file reads and write are statically eliminated.

Conclusions

- Tag-unchecked loads and stores are an example which use compile time analysis to access the cache with direct address registers instead of costly tag checks.