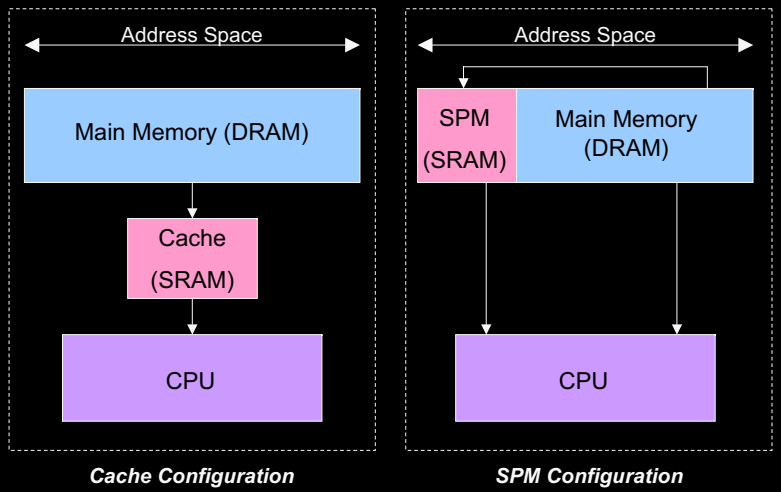
Hardware Managed Scratchpad for Embedded Systems

Ben Rudzyn

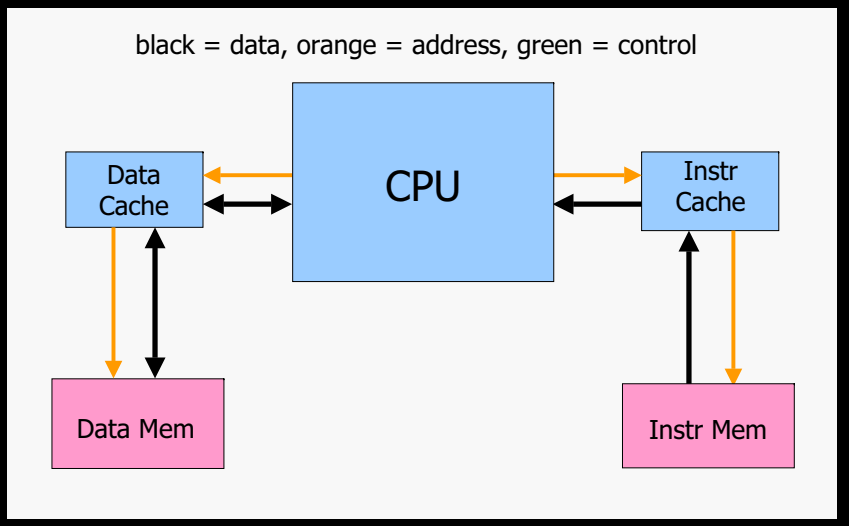
Introduction

- Major concern
  - Energy consumption, execution time
- Embedded Systems
  - Profiling advantage
- What is a Scratchpad Memory (SPM)
  - Array of SRAM cells
  - No extra bits or tags
- How can it be utilised?

Introduction

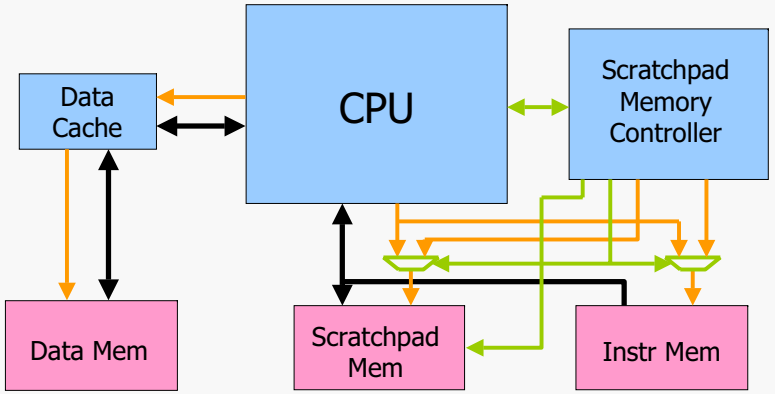


Approach



Approach

black = data, orange = address, green = control



Approach

- SimpleScalar simulator, with syscall modification

CPU	6 stage, statically scheduled, single instruction pipeline
Functional units	1 integer ALU, 1 integer multiplier, 1 integer divider
Functional unit latencies	All single cycle
Instruction cache	2 Kb (256 entries), direct mapped, 1 word block (64 bit), 1 cycle hit latency, 2 cycle miss latency (plus mem delay)
Instruction SPM	8 Kb (1024 entries), 1 cycle hit latency
Instruction main memory	8 Mb SDRAM (10ns), simplified burst mode 10-1-1-1*, 4 word line size
Data cache	2 Kb (512 entries), direct mapped, 1 word block (32 bit), 1 cycle hit latency, 2 cycle miss latency (plus mem delay)
Data main memory	8 Mb SDRAM (10ns), simplified burst mode 10-1-1-1*, 4 word line size

- Accuracy within 0.05% of hardware simulations for large programs (> 1million cycles)

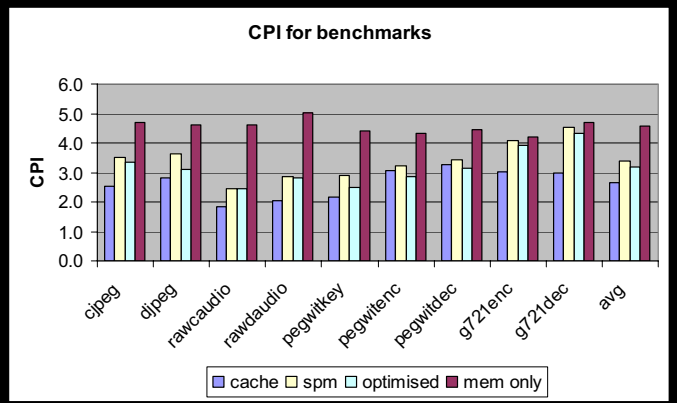
Software Modification

```
        li      $4,0x3b9aca00
        move    $16,$0
$L295:
        sll     $2,$16,2
        addu    $2,$2,$17
        .set    noreorder
        lw      $3,0($2)
        #nop
        .set    reorder
        beq     $3,$0,$L294
        slt     $2,$4,$3
        bne     $2,$0,$L294
        beq     $16,$8,$L294
        move    $4,$3
        move    $7,$16
$L294:
        addu    $16,$16,1
        slt     $2,$16,257
        bne     $2,$0,$L295

        bltz    $7,$L284
        sll     $5,$8,2
        addu    $6,$5,$17
```

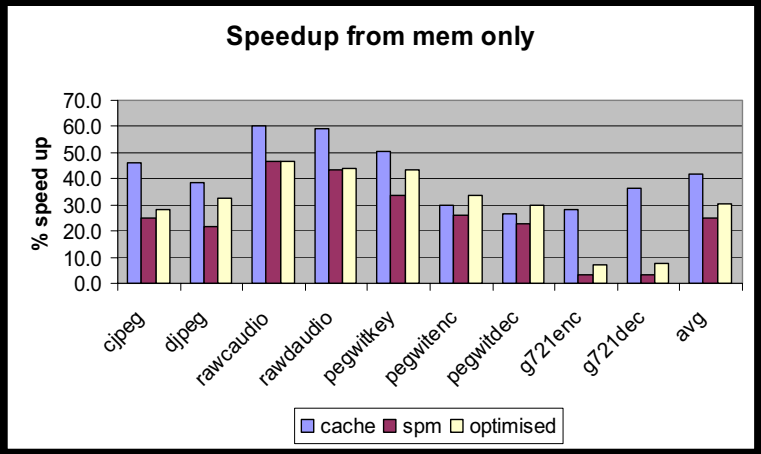
```
        li      $4,0x3b9aca00
        move    $16,$0
        smi     0x12
        j       0x0f000118
$L295:
        sll     $2,$16,2
        addu    $2,$2,$17
        .set    noreorder
        lw      $3,0($2)
        #nop
        .set    reorder
        beq     $3,$0,$L294
        slt     $2,$4,$3
        bne     $2,$0,$L294
        beq     $16,$8,$L294
        move    $4,$3
        move    $7,$16
$L294:
        addu    $16,$16,1
        slt     $2,$16,257
        bne     $2,$0,$L295
        j       $end12

$end12:
        bltz    $7,$L284
        sll     $5,$8,2
        addu    $6,$5,$17
```

Results

- 4 test cases: mem, cache, spm, optimised



Results



Problems

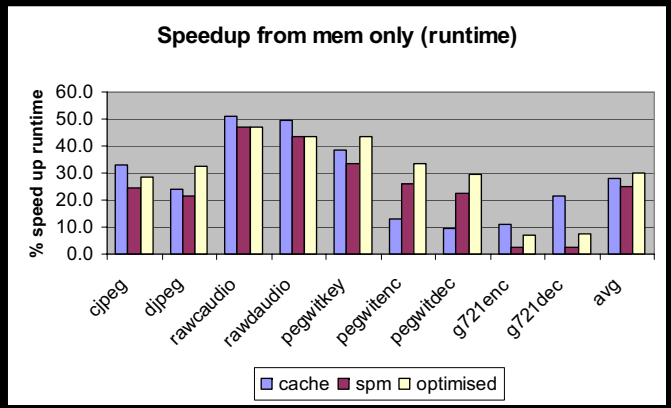
- Overhead
  - SPM fill vs cache miss
  - SPM fill vs cache hit (common case)
  - SPM hit vs cache hit (optimised case)
- Loop size
  - Three extra instructions (*smi, jump, jump*)
  - Number of iterations
- Loop structure
  - *if* statements
  - loop in loop
- Optimised case
  - Saves on SPM fill overhead
  - Still 2 instructions

Limitations

- Library functions
  - Can't be copied at the moment
  - Account for 30% of execution time (adpcm)
- Hand maintenance
  - Instruction insertion
  - Update Controller
  - *jump* addresses
- Size of SPM
  - Optimised case only

Saviour

2 Kb cache → 1.3 ns (760 MHz) 8 Kb SPM → 1.05 ns (950 MHz)



Future Work

- Software compiler
  - Automatically insert *smi* instructions (13 → 70)
  - Automatically update the Controller
  - Automatically update *jump* addresses
- Better procedure to locate blocks and loops of interest
- Optimisation mark II
  - Modify the optimised *smi* placement scheme
  - Use an extra SPM register
  - Allows > 1024 to be stored in the SPM

Conclusion

- Not overly promising so far
- Potential room for improvement through automation

- Next step:
  - Calculate energy consumption
  - Energy profile of the hardware model