The End of ISA Design

*Power Tools for Optimal Processor Generation*

Hot Chips 16
August 22-24, 2004

David Goodwin
Chief Engineer
Instruction Set Architecture

- Contract between software and hardware
- Semiconductor and software progress cause ISA design to evolve
- But always constrained by conflicting goals
  - Generality for future (unknown) applications
  - Improve performance, code-size, area, power for known applications
High performance requirements
  • Wide variety of specialized ISAs

Application must be specifically targeted to exploit ISA
  • C/C++ compiler cannot efficiently generate code for peculiarities of ISA
  • Developer must often use C intrinsic functions or assembly

ASIP better than a coprocessor
  • No data transfer / communication overhead
  • No function unit or memory duplication
  • ASIP based on general core
Meta-ISA: A Better Target for Application Development

XPRES Compiler: new commercially available tool for automatic ASIP generation

**Exposes meta-ISA that supports varying degrees of**
- Instruction-level parallelism
- Data parallelism
- Pipeline parallelism

**Generates variety of ISAs to exploit parallelism**

```c
int main()
{
    int i;
    short c[100];
    for (i=0; i<N; i++)
    {
```
Meta-ISA Parallelism

Maximum parallelism: $L \times M \times N$

Meta-ISA:
- **VLIW**
  - Instruction size
  - Number of operation slots per instruction
  - Operation mix in each slot
- **SIMD**
  - Types of vector operations
  - Number of vector elements
  - Number and size of vector register files
- **Fusion**
  - Number and type of composing operations
  - Number of inputs and outputs
  - Latency

Multi-issue instruction

$L$ operations packed in one long instruction

$M$ copies of storage and function

Register and constant inputs

Fused operation

$N$ dependent operations implemented as single fused operation

Register and constant inputs

SIMD operation

$reg$, $reg$, $reg$, $const$, $op$, $op$, $op$, $reg$
Attributes of Meta-ISA
- Exploits data parallelism with vector (SIMD) operations
- Exploits instruction parallelism with multiple issue instructions
- Exploits pipeline parallelism with long latency fused operations

Application developers focus on clean algorithm implementation
- Expose parallelism
- Don’t tune for peculiarities of specific ISA
- No intrinsics or assembly

Application code more directly implements algorithm
- Easier implementation and maintenance
- Easier debugging
XPRES Backend: Implementing an Application-Specific Processor

XPRES uses Xtensa processor generator flow as backend for creating ASIPs

Application-Specific ISA
Configuration selection and custom-instruction description (TIE)

* TIE: Tensilica Instruction Extension Language

Complete Hardware Design
Verilog/VHDL RTL, EDA scripts, test suite

Use standard techniques and libraries for any IC fabrication process

Customized Software Tools
C/C++ compiler
Debuggers
Simulators
RTOSes

Complete Hardware Design
Verilog/VHDL RTL, EDA scripts, test suite

ALU
DSP
Cache
OCD
Register File
FPU

Xtensa Processor Generator
Implementing a Family of Meta-ISA ASIPs Using XPRES

Application Source Code

int main()
{
    int i;
    short c[100];
    for (i=0;i<N;i++)
    {

XPRES Compiler

        addi    a3, a4, a7
        add8x16 v3, v3, v7
        ld8x16  v7, a2, 4

ASIP Implementations

Generation

Use

int main()
{
    int i;
    short c[100];
    for (i=0;i<N;i++)
    {

Same or Related Application Source Code(s)

        addi    a3, a4, a7
        add8x16 v3, v3, v7
        ld8x16  v7, a2, 4

Customized Software Tools

Application Binary

Copyright © 2004, Tensilica, Inc.
Profile / Analyze application
- Identify performance critical functions / loops
- Collect operation mix, dependence graphs
- Provide feedback to user – code transformations to better target meta-ISA (esp. vectorization)

Generate sets of ISA extensions
- Each set implements some dimension of meta-ISA
- Evaluate each set across all functions / loops
- Performance and cost estimates allow exploration of large design space

Collect ISA extensions that together provide maximum performance improvement
- Each collection of ISA extensions forms ASIP
- Generate family of ASIPs for varying hardware cost budget
XPRES Output Showing Application Performance vs. Hardware Cost
Example: Family of ASIPs

```c
for (i=0; i<n; i++)
    c[i] = (a[i] * b[i]) >> 4
```

Total core size (130 nm) = 0.25 mm² (base Xtensa) + gates / 125,000 mm²
Meta-ISA supports ILP with multiple-issue
- 32-bit or 64-bit instructions
- Each instruction can contain 1-15 slots
- Each slot can contain arbitrary mix of operations

Implemented in Xtensa using FLIX (Flexible Length Instruction Xtensions)
- Similar to VLIW without code size increase

Xtensa C/C++ compiler exploits automatically
- Software pipelining for loops
- List scheduler for other code
XPRES: Instruction Parallelism (exploited using FLIX)

Original C Code

```c
for (i=0; i<n; i++)
    c[i] = (a[i] * b[i]) >> 4
```

64-bit 3-issue FLIX

```assembly
format f 64 { s0, s1, s2 }
slot_opcodes s0 { L32I, S32I }
slot_opcodes s1 { SRAI, MULL }
slot_opcodes s2 { ADDI }
```

64-bit 2-issue FLIX

```assembly
format f 64 { s0, s1 }
slot_opcodes s0 { L32I, S32I }
slot_opcodes s1 { ADDI, SRAI, MULL }
```

- Performance
  - Increased IPC
- Hardware cost
  - Replicated function units
  - Additional register file ports

Generated Assembly
(1 iteration in 3 cycles, 2.6x)

```assembly
loop:
{ l8ui a4, a11, 12;  addi a13, a13, 4 }
{ l8ui a2, a9, 12;  mul16u a3, a3, a2 }
{ l8ui a14, a9, 4;  srai a12, a12, 4;  addi a14, a14, 1 }
```

(2 iterations in 8 cycles, 2x)

```assembly
loop:
{ l8ui a8, a9, 0;  mul16u a12, a10, a8;  addi a9, a9, 1 }
{ l8ui a10, a11, 0;  nop;  addi a11, a11, 1 }
{ s8i a13, a14, 508;  srai a13, a12, 4;  addi a14, a14, 1 }
```

...
Results: Instruction Parallelism

Estimated Area (Gates)

Application Speedup

- idwt
- autocor
- rgbcmyk
- fft-radix4
- gsm encode
- gsm decode
- compress95
- djpeg
- m88ksim
- li
Meta-ISA supports data parallelism with SIMD operations

- Vectors of length 2, 4, 8, and 16 (up to 256 bits)
- Support for unaligned vector loads and stores
- Vector C-operators, MIN, MAX, ABS, reductions
- Vector user-defined operations

Xtensa C/C++ compiler exploits automatically

- Automatic loop vectorization
- Source attributes and compiler options specify aliasing and alignment directives to enable additional vectorization opportunities
**XPRES: Data Parallelism (exploited using SIMD)**

Original C Code

```c
for (i=0; i<n; i++)
    c[i] = (a[i] * b[i]) >> 4
```

Vector Length 8

- regfile vr8x8 64 4 v
- regfile vr16x8 128 2 x
- operation ashri16x8 { out vr16x8 a, ... }

Vector Length 2

- regfile vr8x2 16 4 v
- regfile vr16x2 32 2 x
- operation ashri16x2 { out vr16x2 a, ... }

- **Performance**
  - Increased computation bandwidth
- **Hardware cost**
  - Replicated function units
  - Vector register file(s)

Generated Assembly

(8 iterations in 6 cycles, 10.6x)

```assembly
loop:
    liu8x8 v0,a8,8
    ashri16x8 x0,x0,4
    liu8x8 v1,a9,8
    cvt16x8sr8x8s v2,x0
    mpy8r16x8u x0,v0,v1
    siu8x8 v2,a10,8
...```

• Performance
  • Increased computation bandwidth
• Hardware cost
  • Replicated function units
  • Vector register file(s)
Results: Data Parallelism

![Graph showing application speedup vs. estimated area (gates). The graph includes data points for various applications: idwt, autocor, rgbcmyk, fft-radix4, gsm encode, gsm decode. Each application has a distinct line color and marker.](image)

- **idwt**
- **autocor**
- **rgbcmyk**
- **fft-radix4**
- **gsm encode**
- **gsm decode**
Exploiting Pipeline Parallelism

Meta-ISA supports pipeline parallelism with fused operations

- Fused operation composed of two or more other operations, plus possibly constant values
- Latency of fused operation usually less than combined latency of composing operations
- Limits on input/output operands, number of composing operations, hardware cost, max latency, etc.
- Graphical support for manual fused operation generation

Tradeoff performance and generality

- Performance: large fusions w/ fixed constants
- Generality: smaller fusions, fewer fixed constants

Xtensa C/C++ compiler exploits automatically

- Fused operations automatically replace sequences of composing operations
XPRES: Pipeline Parallelism (exploited using Fusion)

Original C Code

```
for (i=0; i<n; i++)
    c[i] = (a[i] * b[i]) >> 4
```

Performance
- Decrease instruction count
- Decrease computation latency

Hardware cost
- Logic to share function units
- Register file ports

Generated Assembly
(1 iteration in 5 cycles, 1.6x)

```
loop:
  18ui   a12,a11,0
  18ui   a13,a10,0
  addi.n  a10,a10,1
  addi.n  a11,a11,1
  fusion.mull6u.srai.s8i.addi a9,a12,a13
```
Results: Pipeline Parallelism

Estimated Area (Gates)

Application Speedup

- idwt
- autocor
- rgbcmyk
- fft-radix4
- gsm encode
- gsm decode
- compress95
- djpeg
- m88ksim
Exploiting Multiple Types of Parallelism

XPRES combines FLIX, SIMD, and Fusion

Original C Code

```c
for (i=0; i<n; i++)
    c[i] = (a[i] * b[i]) >> 4
```

FLIX and Vectorization (16 iterations in 4 cycles, 32x)

```asm
{ cvt16x8sr8x8s v2,x1;     liu8x8 v0,a8,8;     mpy8r16x8u x2,v1,v2 }
{ si8x8 v2,a10,8;           liu8x8 v1,a9,8;     ashri16x8 v1,v0,4 }
{ siu8x8 v3,a10,16;         liu8x8 v2,a9,8;     ashri16x8 x3,x2,4   }
{ cvt16x8sr8x8s v3,x3;     liu8x8 v1,a8,8;     mpy8r16x8 x0,v0,v1 }
```

FLIX, Vectorization, and Fusion (16 iterations in 3 cycles, 42.6x)

```asm
{ si8x8 v2,a10,8;           liu8x8 v0,a8,8;     fusion.mpy.ashri.cvtx8 v2,v0,v1 }
{ siu8x8 v5,a10,16;         liu8x8 v1,a9,8;     fusion.mpy.ashri.cvtx8 v5,v3,v4 }
{ liu8x8 v3,a8,8;           liu8x8 v4,a9,8;     nop     }
```
Results: Instruction and Data Parallelism

![Graph showing application speedup against estimated area (gates) for various applications. The graph includes applications such as idwt, autocor, rgbcmyk, fft-radix4, gsm encode, and gsm decode.]
Results: Instruction, Data, and Pipeline Parallelism

- idwt
- autocor
- rgbcmyk
- fft-radix4
- gsm encode
- gsm decode
Conclusion

XPRES exposes meta-ISA, a better target for embedded application development

- Instruction, data, and pipeline parallelism
- Developers focus on clean algorithm development, not particulars of specific ISA

Implementations of meta-ISA span range of hardware costs

- Early feedback on potential performance and hardware cost for system applications / tasks
- Quick exploration of performance / cost tradeoffs
Manual Fusion Selection

Dataflow Graphs (1 of 1)

Manual Fusions

Est. Area: <none>  Est. Latency: <none>
Name: 

Fusion Evaluation:
The manual fusion is valid.

Estimated Area: 474 gates
Estimated Latency: > one cycle

Vector Length: Scalar

Evaluate  Add  Clear

OK  Cancel