Specialization Is for Insects
Polymorphous Architectures: A Unified Approach for Extracting Concurrency of Different Granularities

Karu Sankaralingam

Computer Architecture and Technology Laboratory
Department of Computer Sciences
The University of Texas at Austin
http://www.cs.utexas.edu/~karu
Technology Trends

• Wire delays
  – Less than 1% of chip reachable in a cycle
  – Architectures must be partitioned

• Power
  – Limits on pipelining reached
  – 12 to 22 FO4 seems optimal

• Processor complexity

Performance must come from concurrency
Application Heterogeneity

- Face recognition, photo search
- Video editing
- Game physics
- Game graphics
- Bio-informatics
Conventional Microarchitectures

- Intel Pentium 4
- Sun Niagara
- IBM Cell
- NVIDIA G40 (graphics chip)

Desktop | Server | Games/Graphics

Tuned to one type of workload
Integrated Heterogeneity

Poor design reuse and complexity
Thesis Contributions

• Architectural polymorphism
  – Application controlled specialization
  – Coarse grain microarchitectural configuration

• Explicit Data Graph Execution ISA
  – Unifying abstraction layer for all types of concurrency

• Distributed microarchitecture design
  – Micronetworks and protocols
  – TRIPS prototype processor
Outline

• Completed in 2003
  – TRIPS architecture and high level microarchitecture design
  – Preliminary concept of polymorphism
  – Application characterization

• Promised in 2003
  – Detailed application characterization
  – Polymorphism mechanisms
  – TRIPS prototype processor
Outline

• Principles of Polymorphism
• EDGE Architectures and TRIPS prototype
  • Instruction-level parallelism
  • Thread-level parallelism
• Data-level parallelism
  – Application characterization
  – Mechanisms
  – Evaluation
• Conclusion
What is Architectural Polymorphism?

The ability to modify the functionality of coarse grain microarchitecture blocks at runtime, by changing control logic but leaving datapath and storage elements largely unmodified, to build a programmable architecture that can be specialized on an application-by-application basis.

- Principles:
  - Adaptivity to different granularities of parallelism
  - Economy of mechanisms
  - Reconfiguration of coarse grain blocks
System Design

• Granularity of processor core

- To first order differentiates application classes
- Instruction-level parallelism (ILP)
- Thread-level parallelism (TLP)
- Data-level parallelism (DLP)

• Granularity of parallelism
- Fewer number of large cores better than more fine grained cores

• Technology constraints
- Modularity, reduced complexity, and energy efficiency
## Taxonomy of Architecture Principles

<table>
<thead>
<tr>
<th>Architecture type</th>
<th>Processing core type</th>
<th>Processor granularity</th>
<th>Configuration granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmable h/w</td>
<td>Homogeneous</td>
<td>Coarse-grain</td>
<td>Coarse-grain</td>
</tr>
<tr>
<td>App. specific h/w</td>
<td>Heterogeneous</td>
<td>Fine-grain</td>
<td>Fine-grain</td>
</tr>
</tbody>
</table>

**Polymorphous Architectures**

| Programmable            | Homogeneous or       | Coarse or fine        | Coarse grain             |
|                         | Heterogeneous        |                       |                          |

**TRIPS and this Dissertation**

| Programmable            | Homogeneous          | Coarse                | Coarse                   |

**FPGA, Piperench, and ASH**

| App. specific h/w       | Homogeneous          | Fine-grain            | Fine grain               |

**Tarantula**

| Programmable            | Heterogeneous        | Coarse-grain          | -                        |
Outline

• Principles of Polymorphism
• EDGE Architectures and TRIPS prototype
• Instruction-level parallelism
• Thread-level parallelism
• Data-level parallelism
  – Application characterization
  – Mechanisms
  – Evaluation
• Conclusion
EDGE: A Class of ISAs for Concurrency

- **Explicit Data Graph Execution**
  - Defined by two key features

1. Block-atomic execution
   - Program graph is broken into sequences of *blocks*
   - Basic blocks, hyperblocks, or something else

2. Blocks encoded as dataflow graphs: Direct instruction communication
   - The block’s dataflow graph is explicit in the architecture
   - Within a block, ISA support for direct producer-to-consumer communication
   - Across blocks, ISA support for named registers
   - Caveat: memory is still a shared namespace
EDGE Architectures and Polymorphism

• The dataflow graph expresses concurrency efficiently

• ILP
  – Blocks express limited parallelism
  – Control speculation in h/w mines more

• TLP
  – Similar to ILP

• DLP
  – Ample parallelism is efficiently encoded
  – RISC: hardware rediscovers parallelism
C to TRIPS Binaries

• Control flow analysis creates hyperblocks
  – [Smith, CGO 2006] and [Maher, MICRO 2006]
• Scheduler assigns instructions to slots
  – ISA defines 128 slots
  – Scheduling is like a microarchitectural optimization
  – [Nagarajan, PACT 2005], and [Coons, ASPLOS 2006]
• Complete software toolchain
  – GNU binutils based
  – TRIPS compiler builds EEMBC and SPEC CPU2000
TRIPS Microarchitecture Principles

• Limit wire lengths
  – Architecture is partitioned and distributed
  – No centralized resources
  – Local wires are short
  – Networks connect only nearest neighbors

• Design for scalability
  – Design productivity by replicating tiles
  – Communication through well-defined control and data networks
TRIPS Processor Organization

- Partition all major structures into banks, distribute, and interconnect
- Execution Tile (E)
  - Instruction and operand storage
- Register Tile (R)
  - Architectural register storage and buffers (32)
- Data Tile (D)
  - Data cache (8KB) and buffers
  - Ordering and miss-handling logic
- Instruction Tile (I)
  - Instruction cache (16KB)
- Global Control Tile (G)
  - Block prediction & resolution logic
## TRIPS Micronetworks and Protocols

<table>
<thead>
<tr>
<th>Micronetwork</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operand n/w: OPN</td>
<td>Pass operands</td>
</tr>
<tr>
<td>Global dispatch n/w: GDN</td>
<td>Dispatch instructions</td>
</tr>
<tr>
<td>Global status n/w: GSN</td>
<td>Block completion information</td>
</tr>
<tr>
<td>Global refill n/w: GRN</td>
<td>I-cache miss refills</td>
</tr>
<tr>
<td>Data status n/w: DSN</td>
<td>Store completion status</td>
</tr>
<tr>
<td>External store n/w: ESN</td>
<td>Store completion status in L2</td>
</tr>
</tbody>
</table>
TRIPS Chip

130 nm 7LM IBM ASIC process
335 mm² die
~170 million transistors

Overall Chip Area:
29% - Processor 0
29% - Processor 1
21% - Level 2 Cache
14% - On-Chip Network
7% - Other

Processor Area:
30% - Functional Units
4% - Register Files & Queues
10% - Level 1 Caches
13% - Instruction Queues
13% - Load & Store Queues
12% - Operand Network
2% - Branch Predictor
16% - Other
Prototype Design

• Design
  – Modularity reduced complexity: Specification → Physical design
  – SoC-like but tiles form one large uniprocessor

• Verification
  – Hierarchical verification (265 bugs total)
    • Tile-level, processor-level, chip-level
  – Performance verification (16 bugs total)

<table>
<thead>
<tr>
<th>Tile</th>
<th>Function</th>
<th>Cell Instances</th>
<th>Array Bits</th>
<th>Size (mm²)</th>
<th>Tile Instances</th>
<th>% Chip Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td>Processor control</td>
<td>51,684</td>
<td>93K</td>
<td>3.1</td>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>RT</td>
<td>Register file</td>
<td>26,284</td>
<td>14K</td>
<td>1.2</td>
<td>8</td>
<td>2.9</td>
</tr>
<tr>
<td>IT</td>
<td>Instruction cache</td>
<td>5,449</td>
<td>135K</td>
<td>1.0</td>
<td>10</td>
<td>2.96</td>
</tr>
<tr>
<td>DT</td>
<td>L1 Data cache</td>
<td>119,106</td>
<td>89K</td>
<td>8.8</td>
<td>8</td>
<td>21.0</td>
</tr>
<tr>
<td>ET</td>
<td>Instruction execution</td>
<td>83,887</td>
<td>13K</td>
<td>2.9</td>
<td>32</td>
<td>28.0</td>
</tr>
<tr>
<td>MT</td>
<td>L2 Data cache</td>
<td>60,115</td>
<td>542K</td>
<td>6.5</td>
<td>16</td>
<td>30.7</td>
</tr>
<tr>
<td>NT</td>
<td>OCN NW interface and routing</td>
<td>23,467</td>
<td>–</td>
<td>1.0</td>
<td>24</td>
<td>7.1</td>
</tr>
<tr>
<td>SDC</td>
<td>DDR SDRAM controller</td>
<td>64,441</td>
<td>6K</td>
<td>5.8</td>
<td>2</td>
<td>3.4</td>
</tr>
<tr>
<td>DMA</td>
<td>DMA controller</td>
<td>30,365</td>
<td>4K</td>
<td>1.3</td>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>EBC</td>
<td>External bus controller</td>
<td>28,547</td>
<td>–</td>
<td>1.0</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>C2C</td>
<td>Chip-to-chip communication controller</td>
<td>47,714</td>
<td>–</td>
<td>2.2</td>
<td>1</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>Totals (for entire chip)</td>
<td>5.8M</td>
<td>11.5M</td>
<td>334</td>
<td>106</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Prototype Design Lessons

+ Clean predicate model and simple block exit path
+ Register renaming design revised, full search done once
+ H/W prototype design helped push s/w toolchain flow
  + Compiler heuristics, register allocator, scheduler

  – Block predictor design complexity $\Rightarrow$ 3-cycles to predict
  – Significant router area (12%), routing logic on critical path
  – LSQ replication consumed significant area
    – Ongoing work addresses this challenge
TRIPS Motherboard

- Size 14” x 17”
- 18 layers
- Host
  - PowerPC 440GP (400 MHz, 3-way superscalar)
- Debug
  - FPGA XC2VP40 (1148 pins)
  - FPGA connectors for external I/O
- Four daughtercards each with 1 TRIPS chip
Outline

• Principles of Polymorphism
• EDGE Architectures and TRIPS prototype
• Instruction-level parallelism
• Thread-level parallelism
• Data-level parallelism
  – Application characterization
  – Mechanisms
  – Evaluation
• Conclusion
Instruction-Level Parallelism

- Control speculation exposes parallelism
- Register renaming and load/store pairs build program level DFG
ILP Results (Microbenchmarks)

Demonstrates potential
Can compiler generate high quality code?
Thread-level Parallelism

• Execution Tiles:
  – Reservation stations divided between threads

• Register Tiles:
  – Register renaming augmented
  – Extra physical register storage for each thread

• Global Tile:
  – Instruction fetch cycles between threads
  – Small amount of block predictor storage added

• Results:
  – High processor utilization: average IPC of 3.0
  – 2X speedup when executing 4 threads
  – Inter-thread contention in general low: ~20%
  – But dominates for highly concurrent programs
Data-level Parallelism

• Many common attributes:
  – High computation intensity and memory b/w
  – Loops executing on parts of memory in parallel

• But,
  – Memory access patterns can vary
  – Loops sizes can vary
  – Control flow can vary

```
for each vertex V {
  for (j = 0; j < V.ntrans; j++) {
    Z = Z + product(V.xyz, M[j])
  }
}
```

Characterize applications by the different parts of the architecture they affect.
Program Attributes: Control

a) Sequential

- Read record
- Instructions
- Write record

- Vector or SIMD control
- Example: single vadd

b) Static loop bounds

- Read record
- Instructions
- Write record

- Vector or SIMD control
- Branching required
- Example: DCT

10

- MIMD control
- Masking required for SIMD architectures
- Example: skinning

c) Data dependent branching

- Read record
- Instructions
- Write record

- MIMD control
- Masking required for SIMD architectures
- Example: skinning
Program Attributes: Memory

• Regular memory
  – Memory accessed in structured regular fashion
  – Example: Reading image pixels in DCT compression

• Irregular memory accesses
  – Memory accessed in random access fashion
  – Example: Texture accesses in graphics processing

• Scalar constants
  – Run time constants typically saved in registers
  – Example: Convolution filter constants in DSP kernels

• Indexed constants
  – Small lookup tables
  – Example: Bit swizzling in encryption
## Benchmark Suite

<table>
<thead>
<tr>
<th>Domain</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multimedia processing</td>
<td>convert, dct, high pass filter</td>
</tr>
<tr>
<td>Scientific computing</td>
<td>fft, LU</td>
</tr>
<tr>
<td>Network processing, security</td>
<td>md5, rijndael, blowfish</td>
</tr>
<tr>
<td>Real-time graphics processing</td>
<td>vertex-simple, vertex-reflection, vertex-skinning, fragment-simple, fragment-reflection, anisotropic-filtering</td>
</tr>
</tbody>
</table>
Benchmark Attributes

![Bar Chart]

- **Memory Attributes**
  - Regular: 5 benchmarks
  - Irregular: 3 benchmarks
  - Scalar constants: 6 benchmarks
  - Indexed constants: 4 benchmarks

- **Control Attributes**
  - No loops: 9 benchmarks
  - Static bounds: 3 benchmarks
  - Variable: 2 benchmarks

- **Computation Attributes**
  - Low ILP: 4 benchmarks
  - Mid ILP: 6 benchmarks
  - High ILP: 4 benchmarks
# Benchmark Attributes

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Computation</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Inst</td>
<td>ILP</td>
</tr>
<tr>
<td>convert</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>dct</td>
<td>1728</td>
<td>6</td>
</tr>
<tr>
<td>highpassfilter</td>
<td>17</td>
<td>3.4</td>
</tr>
<tr>
<td>fft</td>
<td>10</td>
<td>3.3</td>
</tr>
<tr>
<td>lu</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>md5</td>
<td>680</td>
<td>1.63</td>
</tr>
<tr>
<td>blowfish</td>
<td>364</td>
<td>1.98</td>
</tr>
<tr>
<td>rijndael</td>
<td>650</td>
<td>11.8</td>
</tr>
<tr>
<td>vertex-simple</td>
<td>95</td>
<td>4.3</td>
</tr>
<tr>
<td>fragment-simple</td>
<td>64</td>
<td>2.96</td>
</tr>
<tr>
<td>vertex-reflection</td>
<td>94</td>
<td>7.1</td>
</tr>
<tr>
<td>fragment-reflection</td>
<td>98</td>
<td>6.2</td>
</tr>
<tr>
<td>vertex-skinning</td>
<td>112</td>
<td>6.8</td>
</tr>
<tr>
<td>anisotropic-filter</td>
<td>80</td>
<td>2.1</td>
</tr>
</tbody>
</table>
## Benchmark Attributes

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Record size (words)</th>
<th># Irregular memory accesses</th>
<th># Constants</th>
<th># Indexed scalar constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>convert</td>
<td>3/3</td>
<td>-</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>dct</td>
<td>64/64</td>
<td>-</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>highpassfilter</td>
<td>9/1</td>
<td>-</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>fft</td>
<td>6/4</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>lu</td>
<td>2/1</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>md5</td>
<td>10/2</td>
<td>-</td>
<td>65</td>
<td>-</td>
</tr>
<tr>
<td>blowfish</td>
<td>1/1</td>
<td>-</td>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td>rijndael</td>
<td>2/2</td>
<td>-</td>
<td>18</td>
<td>1024</td>
</tr>
<tr>
<td>vertex-simple</td>
<td>7/6</td>
<td>-</td>
<td>32</td>
<td>-</td>
</tr>
<tr>
<td>fragment-simple</td>
<td>8/4</td>
<td>4</td>
<td>16</td>
<td>-</td>
</tr>
<tr>
<td>vertex-reflection</td>
<td>9/2</td>
<td>-</td>
<td>35</td>
<td>-</td>
</tr>
<tr>
<td>fragment-reflection</td>
<td>5/3</td>
<td>4</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>vertex-skinning</td>
<td>16/9</td>
<td>-</td>
<td>32</td>
<td>288</td>
</tr>
<tr>
<td>anisotropic-filter</td>
<td>9/1</td>
<td>≤ 50</td>
<td>6</td>
<td>128</td>
</tr>
</tbody>
</table>
DLP Bottlenecks

- Inst. Fetch
- Registers
- Memory

% Critical cycles

<table>
<thead>
<tr>
<th>Function</th>
<th>Inst. Fetch</th>
<th>Registers</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>convert</td>
<td>35</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>dct</td>
<td>40</td>
<td>15</td>
<td>25</td>
</tr>
<tr>
<td>filter</td>
<td>45</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>frag-refl</td>
<td>50</td>
<td>25</td>
<td>35</td>
</tr>
<tr>
<td>frag-light</td>
<td>55</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>vert-refl</td>
<td>60</td>
<td>35</td>
<td>45</td>
</tr>
<tr>
<td>vert-light</td>
<td>65</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>vert-skin</td>
<td>70</td>
<td>45</td>
<td>55</td>
</tr>
<tr>
<td>blowfish</td>
<td>75</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>md5</td>
<td>80</td>
<td>55</td>
<td>65</td>
</tr>
<tr>
<td>rijndael</td>
<td>85</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>fft</td>
<td>90</td>
<td>65</td>
<td>75</td>
</tr>
<tr>
<td>LU</td>
<td>95</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>Mean</td>
<td>89</td>
<td>75</td>
<td>85</td>
</tr>
</tbody>
</table>
High Level Architecture

I-Fetch

Register file

L1 memory

L2 memory
DLP Attributes and Mechanisms

Regular memory accesses

I-Fetch
Registre file
L1 memory
Software managed cache
DLP Attributes and Mechanisms

Regular memory accesses
Scalar named constants

Register file

I-Fetch

L1 memory

L2 memory
Regular memory accesses
Scalar named constants
Indexed constants
Tight loops
Data dependent branching

DLP Attributes and Mechanisms

Instruction Revitalization

Local program counter control at each ALU

Register file

L1 memory

L2 memory
Instruction revitalization to support tight loops

- Dynamically create a loop engine
- Power savings, I-Caches accessed once
I-Fetch and Control Mechanisms(2)

- Local PCs for data dependent branching
- Reservation stations are now I-Caches

MIMD Execution Array
Results

• Baseline Machine:
  – 4x4 TRIPS processor with a mesh interconnect

• Kernels hand-coded, placed using custom schedulers

• DLP mechanisms combined to produce 3 configurations
  – Software managed cache + Instruction Revitalization (S)
  – Software managed cache + Instruction Revitalization + Operand Reuse (S-O)
  – Software managed cache + Local PCs + Lookup table support (M-D)
  – Of possible 20 these are most meaningful; operand reuse without instruction revitalization does not make sense for example

• Performance comparison against specialized hardware
Evaluation of Mechanisms

S
S-O
M-D

(Inst. revit, op reuse)
(Local PC, lookup table)

Speedup
Comparison to Specialized H/W

• Pick “best” specialized processor for each workload
• Normalize TRIPS clock to specialized processor:
  – Scale both to 10FO4
• Normalize area based on functional units
  – TRIPS is 16-issue, but MPC 7447 is 4-issue
  – Multiply performance of MPC 7447 by 4
• Optimistic scaling of specialized processors
Comparison to Specialized Hardware

![Graph comparing relative performance of different hardware for multimedia, encryption, scientific, and graphics tasks.]

- **Specialized h/w**
- **TRIPS**
- **Scaled specialized h/w**

- **MPC 7447**
- **Imagine**
- **Tarantula**
- **Cryptomaniac**
- **QuadroFX(F)**
- **QuadroFX(V)**

Summary

• Architectural polymorphism implemented using small set of mechanisms
  – Effective thread level parallelism support
  – Detailed analysis of DLP
  – Mechanisms provide competitive performance compared to specialized processors

• EDGE ISA
  – Dataflow graph abstraction

• TRIPS prototype processor
  – Distributed microarchitecture design principles
Conclusions

• Challenges
  – Application heterogeneity
  – Technology limitations (power and wire delays)

• Architectural polymorphism
  – Coarse grain microarchitectural reconfiguration
  – Scalable modular blocks provide scalability

• Future work
  – Compilation for polymorphous architectures
  – Polymorphism to achieve higher power and area efficiency
Publications

- Dataflow Predication, *MICRO 2006*
- Distributed Microarchitectural Protocols in the TRIPS Prototype Processor, *MICRO 2006*
- TRIPS: A polymorphous architecture for exploiting ILP, TLP, and DLP, *TACO 2004*
- Universal Mechanisms for Data-Parallel Architectures, *MICRO 2003*
- Routed Inter-ALU Networks for ILP Scalability and Performance, *ICCD 2003*
- Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture, *ISCA 2003*
- A Design Space Exploration of Grid Processor Architectures, *MICRO 2001*
TRIPS Prototype

- *High level microarchitecture*  
  - Ramdas Nagarajan and Karu Sankaralingam  
- LSQ  
  - Simha Sethumadhavan and Raj Desikan  
- Next block predictor  
  - Nitya Ranganathan  
- ISA design  
  - Ramdas Nagarajan, Robert McDonald, and Karu Sankaralingam  
- *Prototype microarchitecture spec. and modeling*  
  - Ramdas Nagarajan, Haiming Liu, Nitya Ranganathan, Simha Sethumadhavan, Premkishore Shivakumar, Diyva Gulati, Heather Hanson, and Karu Sankaralingam  
- NUCA cache and OCN design  
  - Changkyu Kim and Paul Grat  
- Logic design and verilog  
  - RT, OPN  
- Processor level verification  
- Chip level verification  
- Physical design  
- Fabrication  
- System bringup  

- 2001  
- 2002-2003  
- 2002-  
- 2004  
- 2004  
- 2002-2005  
- 2004-2005  
- 2005  
- 2005  
- 2005-2006  
- 2006  
- 2006-? (😊)
Questions