Exposed Datapath Implementation Survey
What is Exposed Datapath?

**GPP Processor**
- No Information about the datapath
- Programmer has no possibility to influence the architecture

**Exposed Datapath Processor**
- With explicit commands the programmer can influence the architecture
- show more details of the datapath to the software
- fine-grained control over the datapath in the software
- eliminate redundant data movement
  - -> improvement in performance and energy efficiency.
Three exposed datapath architectures

• Transport Triggered architecture
  • FlexCore architecture
  • Wide SIMD architecture
Transport Triggered Architecture

- TTA programs: data transports between RFs and FUs
- operations are side-effects of the FU, triggered by the “triggering port”
Transport Triggered Architecture

- interconnection network connects all inputs and outputs
- the programmer decides which connection is enabled or disabled
- Adding an FU requires only the connectivity with the most benefit
- No limitations of how many inputs and outputs
Transport Triggered Architecture

- customization of the interconnection network
- data transports directly from one function unit to another (software bypassing)

-> big performance increases
TTA: From C to ASP

1. HLL source code + ADF with minimal resources -> compiler

2. Simulate the Program and customize:
   a) add/remove FUs and RFs
   b) add/remove connections on the Interconnect
   c) Simulate again

3. HDL generation: choose implementations for each component in the ADF

4. RTL implementation: generate the interconnect and the control unit

5. Generate the encodings for the instructions

6. Platform integrator: connect external components and create project files for 3rd party synthesis tools
Custom Operations

• the best speed-up: exploiting a priori knowledge on the algorithm and find a self-defined operation

• create a compiler definition for the operation
  – how many input and output
  – if it has some internal states
  – if it needs access to memory
  – a C/C++ simulation behavior

• add a FU which implements the operation to the processor architecture
  – a new compile-simulate-analyze iteration
Performance of a TTA-processor

- Nios II/f processor, synthesized on an Altera Stratix II FPGA
- MicroBlaze with 3-stage pipeline, running on a Xilinx Virtex 5 family FPGA
- MicroBlaze with 5-stage pipeline, running on a Xilinx Virtex 5 family FPGA.

- CHStone benchmarking suite
  (for measuring the efficiency of HLS tools)
Performance of a TTA-processor

- AES: Advanced encryption standard
- BLOWFISH: Data encryption standard
- JPEG: JPEG image decompression
- SHA: Secure hash algorithm
Three exposed datapath architectures

- Transport Triggered architecture
  - FlexCore architecture
  - Wide SIMD architecture
The FlexCore consists of four datapath units
(more units can be added)
all units are connected to a flexible, fully connected interconnect
FlexCore architecture

- whole data-path is controlled by a 91-bit wide control word  
  (programmer is able to handle the information flow)
- dynamic routing of a result from one datapath unit to the next
- The N-ISA includes the bits controlling the interconnect.
- In each cycle, an N-ISA word controls all units in the datapath as well as the interconnect
• The FlexCore N-ISA control word has 91 bit:
  – Interconnect (24 bits)
  – PC (37 bits)
  – D (Data buffers, 2 bits)
  – LS (Load/Store, 5 bits)
  – A (ALU, 5 bits),
  – Register (18 bits)

The Interconnect (24 bits):
• \( n \): number of control-bits
• \( N \): inputs of the Network (8)
• \( M \): output of the network (7)

\[
n = N \times \log_2 M
\]
Extensions for the FlexCore

- Adding application-specific units: adding more ports to the interconnect and increasing the N-ISA control word
- Example: a multiplier.
  - A 32-bit multiplier is connected to the interconnect
  - Two inputs, two outputs and no control signals
  - The Extended control word: 108-bits.
Generate Code for the FlexCore

• compile high-level code down to N-ISA, thus enables the pipeline length and structure to be changed as often as needed, and allows for programs to use the datapath units in any order.

• translation every single GPP instructions to N-ISA code with four cycles:
  – The first cycle uses the immediate port and the read ports of the Register File
  – The second cycle uses the ALU and RegA
  – The third cycle uses the Load/Store Unit and RegB
  – The fourth cycle uses the write port of the Register File.

• scheduled as tightly as possible with as much overlapping as possible
  – Resource conflicts while overlapping are not a problem
  – Data dependencies are more important: use forwarding

• Generate the overlap:
  – begin without overlap, then with one cycle overlap, then with two cycles overlap, ...
  – Better speedup with changing the order of instructions, but much more complicated
  – performance-critical regions must be scheduled manually
Performance of the FlexCore

- benchmarks from the Embedded Microprocessor Benchmark Consortium (EEMBC):
  - Fast Fourier Transform (FFT)
  - Autocorrelation (Autocor)
  - Viterbi Decoder (Viterbi)
  - High pass grey-scale (RGBHPG) filter

- processors architectures:
  - GPP: The raw output of the FlexCore compiler, without a Interconnect (like a GPP)
  - Exposed GPP: most frequent instructions (inner loops) scheduled manually, using just a few communication paths
  - FlexCore : same instructions like the Exposed GPP, but with the full interconnect of the FlexCore
## Performance of the FlexCore

<table>
<thead>
<tr>
<th>Datapath</th>
<th>Autocor</th>
<th>FFT</th>
<th>RGBHPG</th>
<th>Viterbi</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPP</td>
<td>1.5k (100%)</td>
<td>58k (100%)</td>
<td>3.4M (100%)</td>
<td>268k (100%)</td>
</tr>
<tr>
<td>Exposed GPP</td>
<td>1.3k (88%)</td>
<td>47k (80%)</td>
<td>3.2M (93%)</td>
<td>268 (100%)</td>
</tr>
<tr>
<td>FlexCore</td>
<td>1.0k (67%)</td>
<td>37 (64%)</td>
<td>2.8M (82%)</td>
<td>243 (90%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Datapath</th>
<th>Power (mW)</th>
<th>Timing (ns)</th>
<th>Area (mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPP</td>
<td>6.57 (100%)</td>
<td>2.25 (100%)</td>
<td>0.35 (100%)</td>
</tr>
<tr>
<td>Exposed GPP</td>
<td>6.57 (100%)</td>
<td>2.25 (100%)</td>
<td>0.35 (100%)</td>
</tr>
<tr>
<td>FlexCore</td>
<td>8.07 (123%)</td>
<td>2.49 (111%)</td>
<td>0.40 (115%)</td>
</tr>
</tbody>
</table>
Three exposed datapath architectures

• Transport Triggered architecture
  • FlexCore architecture
  • Wide SIMD architecture
Wide SIMD Architecture

- Two parts: a control processor (CP) and a one dimensional array of processing elements (PEs).
- All the PEs are controlled by the CP
- Based on a 24-bit RISC-like ISA with a 4-bit Header. This leads in a size for a 2-issue VLIW instruction-packet with 56 bits (28-bit CP instruction + 28-bit PE instruction).
- A neighborhood network connects the PEs
- A Bypass connects all PEs with the central CP: broadcast data to all PEs.
Wide SIMD Architecture

- The first operand of each instruction comes from either the local RF, the Bypass or the communication network.
- Two bits in each instruction are used to indicate which source it should use.
- Each instruction can control the result:
  - No write-back: the result is only available at the FU output, so itself and the neighbors can read it.
  - To the WB stage: the result is written to the pipeline register, but not to the RF, so every PE can read it from the pipeline.
  - To RF: the result is stored into the normal RF.
• It is also possible to use the proposed architecture as an accelerator for a different processor.
  – slave-interface
  – direct-memory-access (DMA)
Code Generation

- OpenCL program
- Divide it in work-groups with a number of work-items in it
- The different work-groups are executed sequentially
- Every work-item is independent
- The single work-items in a work-group are mapped to the PE array
Code Generation

• generate for the slave-system
• requires the complete sources of the host-processor and the slave-device
• The host processor controls the program launching.
• For each program, the host processor sends the launch parameters to the accelerator (via the DMA controller)
• After the execution, the result is copied out
Performance of the Wide SIMD Architecture

• Four programs:
  – MAdd (Matrix addition)
  – FIR (5-tap FIR filter on 1-D data stream)
  – Sobel (3x3 Sobel edge detection filter)
  – Transpose (Square matrix transpose)

• three different architectures:
  – 128-PE SIMD processor with automatic bypassing (SIMD)
  – 128-PE SIMD processor with explicit bypassing (SIMD-b)
  – 4-stage 32-bit RISC processor as the reference (RISC)

• The codes were written in OpenCL and compiled off-line. The sequential code for the RISC processor is written in C.
Performance of the Wide SIMD Architecture

speedup

energy-consumption
Thank you for your attention!