Transport Triggered Architectures
Principles and Consequences

Seminar on TTAs
November 25-26, 2002
Tampere

Henk Corporaal
h.corporaal@tue.nl
TUEindhoven
IMEC Leuven
Topics

- Trends and problems
- Architecture design spectrum
- Exploiting Instruction Level Parallelism
- Transport Triggering
- Design Space Exploration
- Can we do better?
- Future and Conclusion
Topics

- Trends and problems
- Architecture design spectrum
- Exploiting Instruction Level Parallelism
- Transport Triggering
- Design Space Exploration
- Can we do better?
- Future and Conclusion
Trends in computers
1990 view

From Hennessy and Patterson, Comp. Arch. 1st ed, 1990
Trends in computers

Supercomputers: 35%/year

Mainframes: 55%/year

Minicomputers: 55%/year

Microprocessors: 55%/year

Tampere TTA seminar 2002
Post PC era

Mainframe

0.01/pers

Compute
Power

PC

1/person

+DSP

+Communications

+ Ambient
Intelligence

Embedded Syst.

>100/person


Tampere TTA seminar 2002
Intelligence, where?

- PDA
- Credit card
- Webpad
- Head-mounted displays
- Hand computer
- Keyboard
- Organizer
- Camera
- Medical sensors
- Watch
What kind of processors?

Realization space is huge

- Cost: $1000 - $0.1 \(O(10^4)\)
- Power: 100 W - 10 mW \(O(10^4)\)
- Performance: 100 Gops - 10 Mips \(O(10^4)\)

high-end | low-end
Design productivity gap

Process technology +58%

HW design productivity +21 %

SW productivity +8 %
Computational efficiency gap

Intrinsic computational efficiency

[Roza]

Feature size [µm]

[MOPS/W]

Tampere TTA seminar 2002
Problem statement

We need an processor architecture which is:
- flexible, scalable, simple and quick to design,
  programmable, low power, high computational efficiency,
  easy to integrate in embedded system

We need a design environment for
- semi-automatic code generation
- simulation and testing
- semi-automatic processor realization

Solution: TTA framework
Topics

- Trends and problems
- Architecture design spectrum
- Exploiting Instruction Level Parallelism
- Transport Triggering
- Design Space Exploration
- Can we do better?
- Future and Conclusion
Architecture design spectrum (parallelism view)

Four dimensional architecture design space: I,O,D,S

- Operations/instruction ‘O’
- Data/operation ‘D’
- Instructions/cycle ‘I’
- Superpipelining degree ‘S’

VLIW, Superpipelined, Superscalar, RISC, SIMD, CISC
Architecture design spectrum

Which choice: I, O, D, or S? A few remarks:

- I: instructions / cycle
  - Superscalar / Dataflow: limited scaling due to complexity
  - MIMD: do it yourself (yet)

- O: operations / instruction
  - VLIW: good choice if binary compatibility not an issue
  - Speedup for all types of applications
Architecture design spectrum

- **D**: data/operation
  - SIMD / Vector: application has to offer this type of parallelism
  - may be good choice for multimedia
  - dense instruction format

- **S**: pipelining degree
  - Superpipelined: cheap solution
  - however, operation latencies may become dominant
  - unused delay slots increase
**Architecture design spectrum**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>I</th>
<th>O</th>
<th>D</th>
<th>S</th>
<th>Mpar</th>
</tr>
</thead>
<tbody>
<tr>
<td>CISC</td>
<td>0.2</td>
<td>1.2</td>
<td>1.1</td>
<td>1</td>
<td>0.26</td>
</tr>
<tr>
<td>RISC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>VLIW</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1.2</td>
<td>12</td>
</tr>
<tr>
<td>Superscalar</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>4.8</td>
</tr>
<tr>
<td>Superpipelined</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Vector</td>
<td>0.1</td>
<td>1</td>
<td>64</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>SIMD</td>
<td>1</td>
<td>1</td>
<td>128</td>
<td>1.2</td>
<td>154</td>
</tr>
<tr>
<td>MIMD</td>
<td>128</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>154</td>
</tr>
<tr>
<td>Dataflow</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>24</td>
</tr>
</tbody>
</table>

*Mpar is the amount of parallelism to be exploited by the compiler / application!*
Topics

- Trends and problems
- Architecture design spectrum
- Exploiting Instruction Level Parallelism
- Transport Triggering
- Design Space Exploration
- Can we do better?
- Future and Conclusion
Mapping / Scheduling: placing operations in space and time

d = a * b;
e = a + d;
f = 2 * b + d;
r = f - e;
x = z + y;

Data Dependence Graph (DDG)
How to map these operations?

Architecture constraints:
• One Function Unit
• All operations single cycle latency

Cycle 1
1. *

Cycle 2
2. *

Cycle 3
3. +

Cycle 4
4. +

Cycle 5
5. -

Cycle 6
6. +

Tampere TTA seminar 2002
How to map these operations?

Architecture constraints:
- One Add-sub and one Mul unit
- All operations single cycle latency

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Mul</th>
<th>Add-sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>+</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td>+</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
There are many mapping solutions

![Diagram](Tampere TTA seminar 2002)
A processor: how does it look inside?
General organization of an ILP architecture

Instruction memory

Instruction fetch unit

Instruction decode unit

FU-1
FU-2
FU-3
FU-4
FU-5

Register file

Data memory

CPU

Pipeline

Ifetch  Decode  R-read  Execute  Execute  R-write

Tampere TTA seminar 2002
Instruction Level Parallelism

- ILP (*Instruction Level Parallel*) architectures very popular
  - Flexible & Scalable
- However: Fundamental problems of current ILP architectures
  - Control complexity in Superscalars / Dataflow
  - Scaling problems
    - number of ports on register file
    - bypass complexity
  - Flexibility problems
    - can we plug in arbitrary functionality?
General organization of an ILP architecture

Instruction memory → Instruction fetch unit → Instruction decode unit → Bypassing network → Register file → Data memory

CPU

FU-1 → FU-2 → FU-3 → FU-4 → FU-5

Control problem

O(N^2)  O(N) - O(N^2)  
With N function units

Tampere TTA seminar 2002
Bypassing requires many busses, multiplexors, and dynamic matching of source and destination id’s (associative matching).
VLIW evaluation

Strong points of VLIW:
- Scalable (add more FUs)
- Flexible (an FU can be almost anything; e.g. multimedia support)

Weak points:
- With N FUs:
  - Bypassing complexity: $O(N^2)$
  - Register file complexity: $O(N)$
  - Register file size: $O(N^2)$
- Register file design restricts FU flexibility

Solution: *mirror programming paradigm*
Solution

Mirroring the Programming Paradigm

TTA: Transport Triggered Architecture
Topics

- Trends and problems
- Architecture design spectrum
- Exploiting Instruction Level Parallelism
- Transport Triggering
- Design Space Exploration
- Can we do better?
- Future and Conclusion
Transport Triggered Architecture

General organization of a TTA

- Instruction memory
- Instruction fetch unit
- Instruction decode unit
- Bypassing network
- Register file
- FU-1
- FU-2
- FU-3
- FU-4
- FU-5

CPU

Data memory
TTA structure; datapath details

Data Memory

Load/store unit
Load/store unit
Integer ALU
Integer ALU
Float ALU

Integer RF
Float RF
Boolean RF
Instruct. unit
Immediate unit

Instruction Memory

Socket
Function unit design

- Pipelining model:
  - Hybrid
  - TVTL
  - SVTL

- Standard interface:
  - FU_ready
  - Result_ready
  - Global_lock
TTA hardware characteristics

- Modular: building blocks easy to reuse
- Very flexible and scalable
  - easy inclusion of Special Function Units (SFUs)
- Very low complexity
  - > 50% reduction on # register ports
  - reduced bypass complexity (no associative matching)
  - up to 80% reduction in bypass connectivity
  - trivial decoding
  - reduced register pressure
  - easy register file partitioning (a single port is enough!)
TTA software characteristics

- More difficult to schedule!
- But: extra scheduling optimizations

That does not look like an improvement!??!
TTA specific optimizations

Bypassing can eliminate the need of RF accesses

Example:

\[
\begin{align*}
    r1 & \rightarrow \text{add.o1}, & r2 & \rightarrow \text{add.o2}; \\
    \text{add.r} & \rightarrow r3; \\
    r3 & \rightarrow \text{sub.o1}, & 95 & \rightarrow \text{sub.o2} \\
    \text{sub.r} & \rightarrow r4;
\end{align*}
\]

Translates into:

\[
\begin{align*}
    r1 & \rightarrow \text{add.o1}, & r2 & \rightarrow \text{add.o2}; \\
    \text{add.r} & \rightarrow \text{sub.o1}, & 95 & \rightarrow \text{sub.o2}; \\
    \text{sub.r} & \rightarrow r4;
\end{align*}
\]
Scheduling example

VLIW

```
add r3, r1, r2
sub r4, r1, 99
```

TTA

```
r1 -> add.o1, r2 -> add.o2
add.r -> sub.o1, 95 -> sub.o2
sub.r -> r4
```
Register file port pressure

Read and write ports required

ILP degree

Read ports

Write ports
Register efficiency

Impact of decreasing number of registers

- Integrated allocation
- Early allocation

Cycle count increase [%]

Number of registers

Tampere TTA seminar 2002
Register file measures

Energy/access = (c1 + c2 \cdot N_{\text{regs}}) (c3 + c4 \cdot N_{\text{ports}})

T_{\text{access}} = c1 + c2 \cdot N_{\text{regs}} + c3 \cdot N_{\text{ports}}

Area = N_{\text{regs}} (1 + c1 \cdot N_{\text{ports}})^2
Code generation trajectory

- Application (C)
- Compiler frontend
- Sequential code
- Compiler backend
- Parallel code
- Sequential simulation
- Profiling data
- Parallel simulation
- Input/Output

- Frontend: GCC or SUIF (adapted)
TTA compiler characteristics

- Handles all ANSI C programs
- Region scheduling scope with speculative execution
- Using profiling
- Software pipelining
- Predicated execution (e.g. for stores)
- Multiple register files
- Integrated register allocation and scheduling
- Fully parametric
- Clustered architectures
Code:

A;  
If cond  
Then B  
Else C;  
D;  
If cond  
Then E  
Else F;  
G;

CFG: Control Flow Graph

A

B  C

D

E  F

G
Scheduling scopes

Trace  Superblock  Decision tree  Hyperblock/region
Code movement (upwards) within regions

Legend:
- Copy needed
- Intermediate block
- Check for off-liveness
- Code movement

source block

destination block
Topics

- Trends and problems
- Architecture design spectrum
- Exploiting Instruction Level Parallelism
- Transport Triggering
- Design Space Exploration
- Can we do better?
- Future and Conclusion
Design of embedded architectures

We have described a

- Templated architecture
- Parametric compiler exploiting specifics of the template

Problem:

*How to tune a processor architecture for a certain application domain?*
Example SoC: Intelligent datalogger

Features:
• mixed signal
• special FUs
• on-chip RAM and ROM
• operates stand alone
• core generated automatically
Mapping applications to processors

Move framework

- User interaction
- Architecture parameters
- Optimizer
- Parametric compiler
- Hardware generator
- Feedback

Pareto curve (solution space)

exec. time

Parallel object code

Tampere TTA seminar 2002
Exploration: resource reduction
Exporation: connectivity reduction

- Reducing bus delay
- FU stage constrains cycle time
- Critical connections disappear

Number of connections removed

Execution time

Tampere TTA seminar 2002
Exploration: resulting architecture

Architecture for image processing
- Note the reduced connectivity
Topics

- Trends and problems
- Architecture design spectrum
- Exploiting Instruction Level Parallelism
- Transport Triggering
- Design Space Exploration
- Can we do better?
- Future and Conclusion
Can we do better

How?
- Transformations
- SFUs: Special Function Units
- Multiple Processors
Transforming the specification

Based on associativity of + operation
\[ a + (b + c) = (a + b) + c \]
Transforming the specification

d = a * b;
e = a + d;
f = 2 * b + d;
r = f – e;
x = z + y;

r = 2*b – a;
x = z + y;
Changing the architecture
adding SFUs: special function units

4-input adder
why is this faster?
Changing the architecture
adding SFUs: special function units

In the extreme case put everything into one unit!

Spatial mapping
- no control flow

However: no flexibility / programmability !!
SFUs: fine grain patterns

- Why using fine grain SFUs:
  - Code size reduction
  - Register file #ports reduction
  - Could be cheaper and/or faster
  - Transport reduction
  - Power reduction (avoid charging non-local wires)
  - Supports whole application domain!

Which patterns do need support?
- Detection of recurring operation patterns needed
SFUs: fine grain patterns

General pattern & subject graph
- multi-output
- non-tree
- operand and operation nodes
SFUs: covering results
Bridging computational efficiency gap

[Graph showing computational efficiency as a function of feature size.]

- Intrinsic computational efficiency
- Application specific
- Domain specific
- Energy eff. processor

Tampere TTA seminar 2002
Multiple processors

Why splitting your program into threads?

- Combine fine (ILP) and coarse grain parallelism
- Avoid ILP bottleneck
- Multiprocessor solution may be cheaper
  - More efficient resource use
- Wire delay problem → clustering needed!
Instant frequency tracking example
Results of automatic partitioning loop-nests

![Bar chart showing speedup for different benchmarks across 1, 2, 3, and 4 processors. Benchmarks include arfreq, g722, instf, interp3, mulaw, music, radproc, rfast, rtpse, mpeg2enc. The y-axis represents speedup, and the x-axis represents different benchmarks.]
Topics

- Trends and problems
- Architecture design spectrum
- Exploiting Instruction Level Parallelism
- Transport Triggering
- Design Space Exploration
- Can we do better?
- Future and Conclusion
Future

Do we run out of steam?

Performance

Year

35%/y

7%/y

55%/y

Processing

Memory

Performance gap
Future: more bad news!
Delay: Gate vs. Interconnect
Solution

Locality, Locality & Locality

- Local data
  - Elaborate on-chip memory hierarchy
  - IMEC’s data transfer and storage exploration (DTSE) methods
- Local processing
  - Clustering, Multiple processors
- Local communication and synchronization
  - Short wires

⇒ Use Networks on Chip
Networks on Chip

On-chip Network

NoC platform transaction layer abstraction

Application

Network interface
Networks on Chip

On-chip Network

Network interface

NoC realization

Tampere TTA seminar 2002
Merger of computation and communication

**SNDs: Smart Networked Devices**

Computation and communication platform

---

**Virtual Machine**

- Protocols for Multimedia (mpeg21) Network
- RTOS
- HW-SW library

---

radio

programmable hardware

accelerator hardware

reconfig. hardware
Trends: SND keywords

**Past**
- homogeneous
- single node
- SW reconfigurable
- store-and-forward
- best-effort
- time-sharing
- wired
- somewhere

**Future**
- heterogeneous: NoC
- distributed application
- computing, storage, bandwidth
- SW and HW reconfigurable
- store-compute-forward
- QoS
- scalability, graceful degradation, ..
- time-and-power sharing
- wireless
- everywhere
Network of communicating IP blocks

Possible blocks:
- 32-bit RISC microprocessor ~ 40 Kgates
- MPEG decoding ~ 100 Kgates
- Wavelet filtering ~ 40 Kgates
- CDMA, OFDM, ..
- SRAM blocks
- SDRAM blocks
- FPGA blocks
- ....................
- **TTA cores**
Conclusion

- Billions of embedded processing systems
  - how to design these systems quickly, cheap, correct, low power,.... ?
  - what will their processing platform look like?

- TTAs are certainly a candidate
  - extremely flexible building block
  - very scalable
  - semi-automatic design space exploration
  - automatic processor generation
  - low cost, low power

*do not pay for hardware if you can do it by software*