Challenges of MPSOC Communication, Computation and Design Flow

Prof. Jari Nurmi
Tampere University of Technology
Institute of Digital and Computer Systems
P.O.Box 553, FIN-33101 Tampere
FINLAND
Email: jari.nurmi@tut.fi

Outline

Ways to address the application-specific requirements in MPSOC computation

How to combine Network-on-Chip and computation efficiently

A bit on the role(s) of reconfigurability

How should MPSOCs be designed

Application-specific processing power is needed

Just put there more (general-purpose) processors!

Or use more specialized solutions like
- Configurable cores (Xtensa)
- Application-specific processors (CoWare LISA)
- Reconfigurable cores (XiRISC)
- Accelerators (Coffee + MILK + BUTTER)

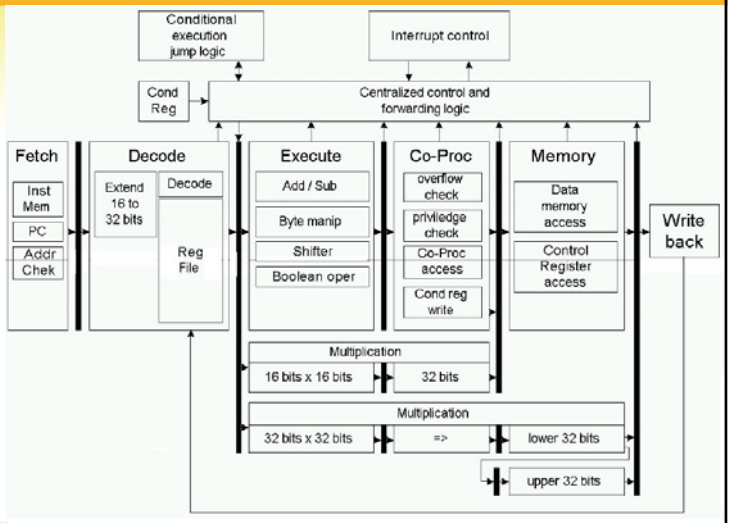
All of the latter mean that we end up using heterogeneous multiprocessor configurations

Coffee RISC Core ™

Open-source hardware (BSD licence variation)

Tools open-source software (mostly GPL and LGPL licences)

Available at **coffee.tut.fi**



Integrated Floating-Point capability

MILK Floating-Point co-processor

Detached from the main computation flow → inefficience

Now Milk merged with Coffee (→ Cappucino ? ☺)

Single-issue multi-commit architecture (result register locking mechanism)

The Past: RAA

Reprogrammable Algorithm Accelerator

A MIMD style array of simple processors

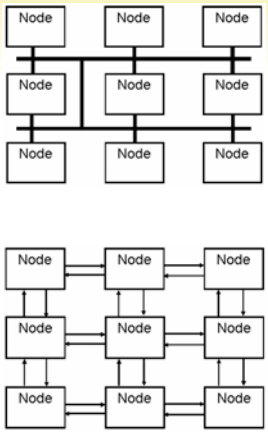
Can be programmed/configured by a host processor

- Individual node
- Row
- Column
- Rectangular region

at a time

PEs communicate using FIFOs

Each PE has two output FIFOs to avoid congestion



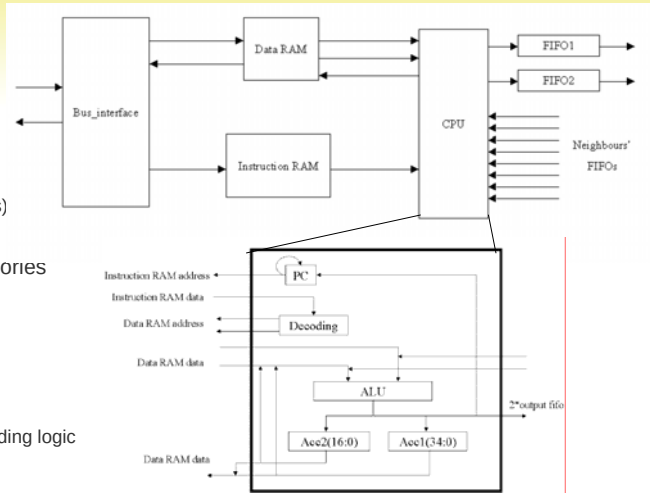
RAA processing node

Host interface on left

FIFO interfaces on right
- 2 output FIFOs
- 8 input FIFOs
  (output FIFOs of
  4 nearest neighbors)

Data and instruction memories
- 16-bit or 8-bit data

DSP CPU
- ALU
- 2 accumulators
- Sequencing and decoding logic



RAA context memories

Multiple contexts to hide reconfiguration latency

Host interfaces, PC, accumulators, memories duplicated for each additional configuration

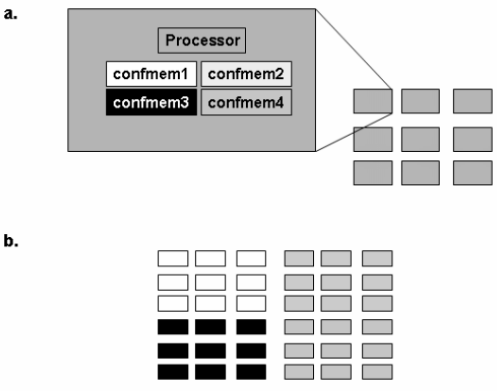
FIFOs are not duplicated but shared between different contexts (configurations)
- Context identifier in each FIFO entry
- Deadlocks avoided by swapping adjacent entries (alternating odd and even pairing) in the FIFO if the first entry does not belong to the context in execution

Virtualizing array dimensions using multiple contexts
- Multiple configurations can be used to represent parts of a larger array
- Binary compatibility between different array sizes achieved!

RAA proved to be a working solution for general-purpose acceleration

Just 10s of code lines for a single PE typically

Tools to program require ”assembly” level entry



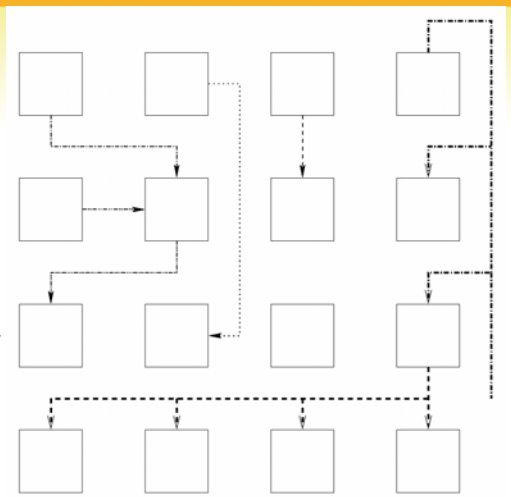
Virtual array of size 4x is only about 2x the area but has about 1/4x the speed → area/speed trade-off

And now: BUTTER (is BETTER)

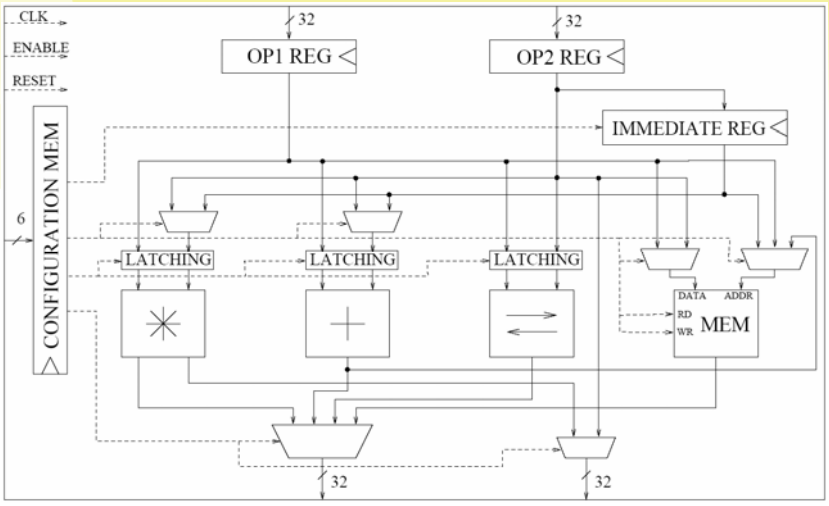
BUTTER is a N x M array of reconfigurable floating-point units

Flexible interconnect schemes between the PEs

Dedicated input and output in addition to the system bus (or network!) interface which is mainly used for configuration purposes



BUTTER processing element



BUTTER first results and the competition

	Logic cells	Speed (MHz)	area (μm^2)	area (Kgates)	Speed (MHz)
Butter	66976	65	2858818	476470	280
Morphosys	n.a.	n.a.		597696	100
Adres	n.a.	n.a.	7000000	116666	400
montium	n.a.	n.a.	2200000	366666	65

Size and speed figures in 0.13 um ASIC technology and on Altera Stratix2 FPGA for a 8 x 4 BUTTER instance

TABLE II
CLOCK CYCLES FOR DCT ON DIFFERENT PLATFORMS

Algorithm	Butter	M1	REMARC	Pentium	TMS320C55x
DCT	18	21	54	240	320

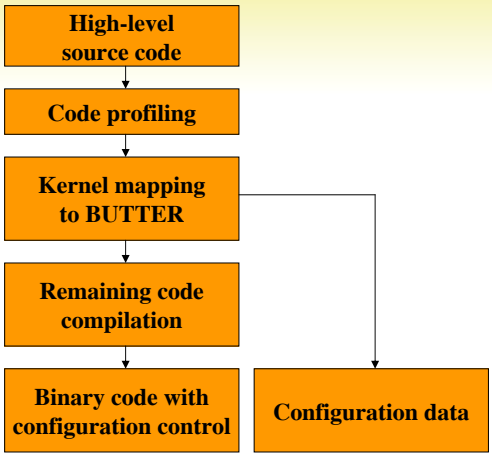
DCT/IDCT mapping results

TABLE III
CLOCK CYCLES FOR IDCT ON DIFFERENT PLATFORMS

Algorithm	Butter	XiRISC	TMS320C55x
IDCT	54	960	753

The design flow for acceleration must be automated!

The target flow:



The book project (completion 4/2007)

Introduction (**J. Nurmi**, TUT)
Processor architecture fundamentals revisited (**J. Nurmi**, TUT)
Beyond the Valley of the Processors (fallacies and pitfalls in processor design) (**S. Leibson**, **G. Martin**, Tensilica)
Processor design flow (**J. Nurmi**, TUT)
General-purpose embedded processor core design (**J. Kylliäinen**, **J. Nurmi**, TUT)
DSP processor design space (**G. Frantz**, TI)
VLIW processors for high-end DSP processing (**C. Panis**, Catena Radio Design)
Customizable processors and processor customization (**S. Leibson**, Tensilica)
Reconfigurable processor architectures (**F. Campi**, ARCES)
Co-processor approach to accelerating multimedia applications (**C. Brunelli**, **J. Nurmi**, TUT)
Designing processors for FPGAs (**J. Ball**, Altera)
Protocol processor design issues (**S. Virtanen**, UTU)
Stream processors (**A. Agarwal**, **R. Rabbah**, MIT)
Java co-processor design (**T. Säntti**, **J. Tuominen**, **J. Tyystjärvi**, **J. Plosila**, UTU)
On-chip multi-core processors (**J. Goodacre**, ARM)
Processor clock generation and distribution (**S. Rusu**, Intel)
Asynchronous and self-timed processor design (**S. Furber**, **J. Garside**, U Manchester)
Application-specific processor design tools (**A. Hoffmann**, CoWare)
Early estimation models of processors (**T. Nurmi**, UTU, **T. Ahonen**, **J. Nurmi**, TUT)
High-level simulation models (**S. Virtanen**, UTU, **S. Määttä**, **J. Nurmi**, TUT)
Programming tools for reconfigurable processors (**C. Mucci**, **F. Campi**, ARCES, **C. Brunelli**, **J. Nurmi**, TUT)
Future directions in processor design (**J. Nurmi**, TUT)

Chapter on **processor testing**, anyone?

Network-on-Chip

Buses do not scale well

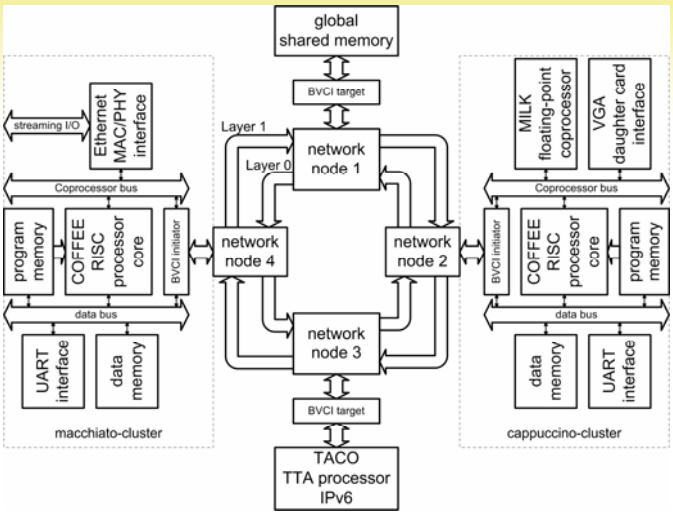
NoC provides higher bandwidth

Early NoC schemes include e.g.
- xPipes (University of Bologna et al)
- Nostrum (KTH, VTT)
- SPIN (LIP6 Paris)
- Proteo (TUT)
- XGFT (TUT et al)

Overview in SoC 2005 keynote
”*NoC will never completely replace buses*” (Nurmi, SoC 2005)

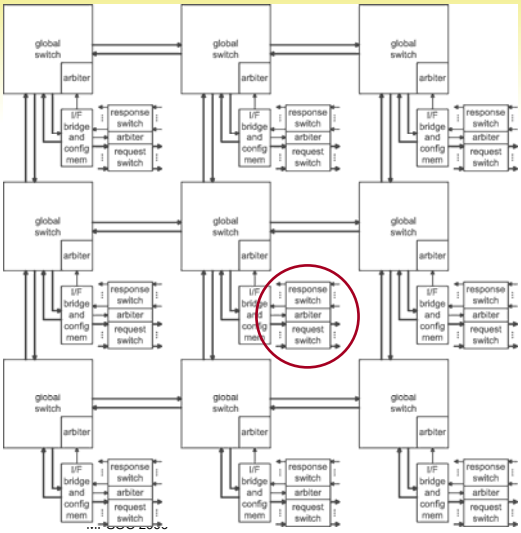
The Past: PROTEO NoC

- Divided roles (initiator/target)
- Complex interface logic between the NoC and local bus
- Bus mastership required to deliver incoming packets
- Slave devices shared over the network (slow, large network)



How about replacing the bus: Hierarchical NoC scheme

- Local buses replaced by memory-mapped switch cluster
- N masters and M slaves requires N x (M+1) switches
- Bus bottleneck avoided
- Guaranteed service can be provided
- Programmable priority scheme (relative priorities)
- Programmable configuration lifetime and fast context switching (page pointer set externally)
- Pipelined accesses
- Small and fast switches achievable (one-hot selection)



Switch implementation results

90 nm technology, T=125°C, Vcc=0.95V, 5 metal layers, wire load model for over 100K gate blocks

32-bit data, 16-bit address, 4 byte enables, write enable, valid, 16-bit routing field

Node NxM /RQ /RSP	Optimized for	Levels of logic	Latency (ps)	Gate count (NAND2)	Leakage (nW)
Global 5x5 /70b	speed	5	1343	11734	510
Local 2x5 /54b /32b	speed	7	959	4588	229
Global 5x5 /70b	area	6	3860	1980	78
Local 2x5 /54b /32b	area	7	3191	1108	37

Reconfigurable NoC

What the reconfiguration can be used for in networks?

Remember: most networks provide inherent redundancy

So, you can improve manufacturability/yield by reconfiguring the network, in a similar manner as hard-disks and memory chips are ”repared” by configuring the address logic in case of bad blocks.

Fault detection and repair (FDAR) systems can also detect transient faults and recover the network operation by reconfiguration

- separate diagnostics mode
- health monitoring and automatic repair in user mode

FDAR used successfully for Mesh and XGFT networks at TUT with just about 10-15% area overhead

More work on NoC manufacturability, testing, verification needed

The design flow for networks must be automated!

Network generation and optmization is an additional high-level task before (or partially interleaved) with the normal SoC design flow

Currently we have a network generation and optimization tool (OIDIPUS) at TUT which works currently well for hierarchical rings (PROTEO) finding optimum placements within the ring for the interconnected blocks

AFAIK, nobody has a comprehensive tool to generate arbitrary networks from the communication specifications

Getting into the best suited network implementation requires exploring different topologies in addition to ”turning the knobs” of a single topology to their correct values

Feedback to computation: The Latency

Network latency added to the memory latency

In DRAM access the network latency is not significant

In large distributed shared memory type of multiprocessors the network latency may start to dominate the access latency that a single processor experiences

The latency may be from 10’s to several 100’s of cycles in a high-end embedded processor

(The network latency in the hierarchical NoC e.g. in 4 x 4 clusters
→ max 6 x global + 2 local = 10 ns + the memory latency)

Ways to fight impacts of latency in MPSOC

Latency-tolerant multi-issue processors have been proposed
- (Very) large instruction issue window
- Large load and store queues

enabling continuously running independent instructions also during miss handling

Normally required that
- Loads take place in program order
- Stores must snoop the load queue for possible violations

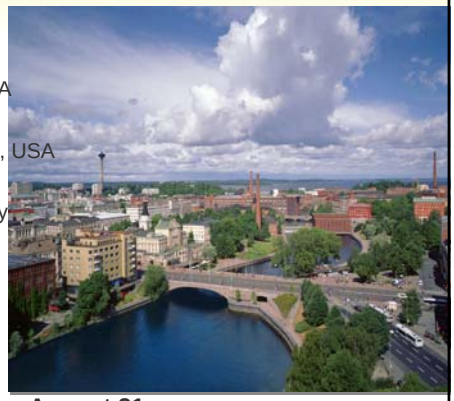
which will complicate large load and store queue implementations

Thus two-level load and store buffers proposed
- second level set-associative load buffer
- hierarchical store queue or store-redo log based algorithm without CAM *

To have less dependences and better PE utilization, simultaneous multi-threading can be used in the multi-issue cores

SoC 2006, Tampere, Finland, Nov. 14-16

Theme of the year: "SoC Design Methodology and Tools"
(The design flow for ... must be automated ☺)
Tutorial on SystemVerilog on Monday November 13
Invited talks
- Prof. Arvind, MIT, USA
- Prof. Shuvra Bhattacharyya, U Maryland, USA
- Prof. Jean-Luc Dekeyser, USTL, France
- Dr. John Glossner, Sandbridge Technologies, USA
- Dr. Yervant Zorian, Virage Logic, USA
- Dr. Leandro Soares Indrusiak, TUD, Germany
- Steve Leibson, Tensilica, USA
- N.N., Nokia, Finland
- M.M., CoWare, USA



About 40 **contributed scientific papers**
Exhibit – some free space still available...
Industry track papers extended deadline: August 21

PhD theses of 2005-2006 from my group

David Sigüenza Tortosa, *PROTEO: The Development of a Practical Network-on-Chip*

Tapio Ristimäki, *Reconfigurable IP Blocks: a MIMD Approach*

Lasse Harju, *Programmable Receiver Architectures for Multimode Mobile Terminals*

Heikki Kariniemi, *On-line Reconfigurable Extended Generalized Fat Tree Network-on-Chip for Multiprocessor System-on-Chip Circuits*

Tapani Ahonen, *Designing Network-Based Single-Chip System Architectures*

Many more are in the pipeline, working with Coffee, BUTTER, design tools, etc.

Industrial cooperation and cofunding are warmly welcome!

Conclusions: My view of dream MPSOC

Multiple heterogeneous cores (e.g. Coffee/Cappucino, Butter) with local memories

Hierarchical NoC

Reconfiguration used for improving yield and fault tolerance

All designed from high-level design entry