ITMO

Computer Systems Design
Lesson 2
Non-vN architectures.
Domain-specific architectures

Alexander Antonov, Assoc. Prof., ITMO University

Hangzhou, 2025

Outline the lesson

- Limitations of vN architectures
- Architectures with programmable data flows
- Microprogrammable and reconfigurable architectures
- Domain-specific architectures
- Systems on chip

What people don’t like in vN architecture

- Program counter as a global “sequencing” mechanism

  Hampers concurrency

- “Memory/power/ILP walls” *(D. Patterson)*

  Computations appear to be disintegrated with memory

  Caches can hide latency, but are resource-hungry

  Complexity increase cannot be unlimited

- “Education wall” *(R. Hartenstein)*

  Abstraction from HW-friendly data streams is dominant

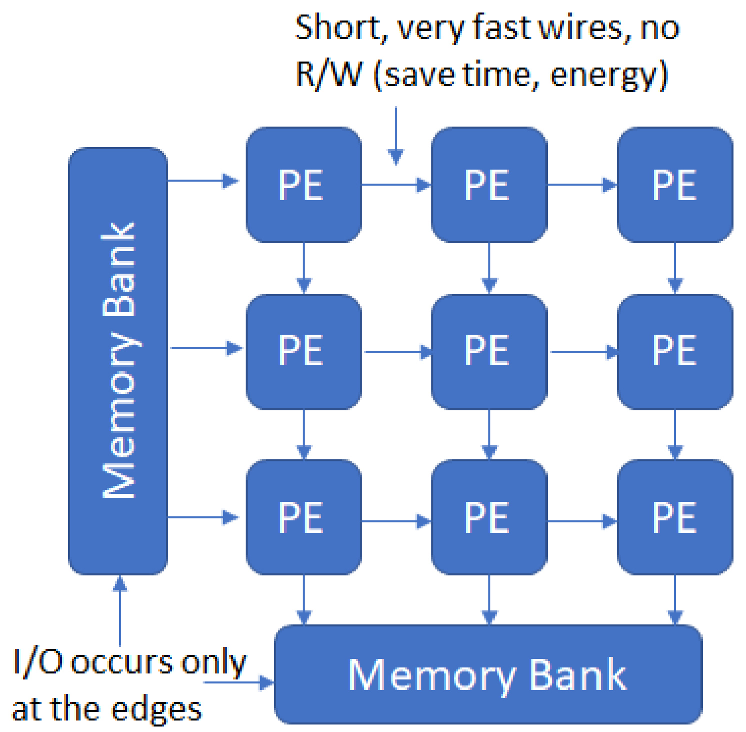
  Almost all SW stacks are built with vN architecture in mind

  Alternatives are almost not considered

Non-vN architectures attempt to avoid common limitations

Architectures with programmable data flows

Systolic array (1943…)

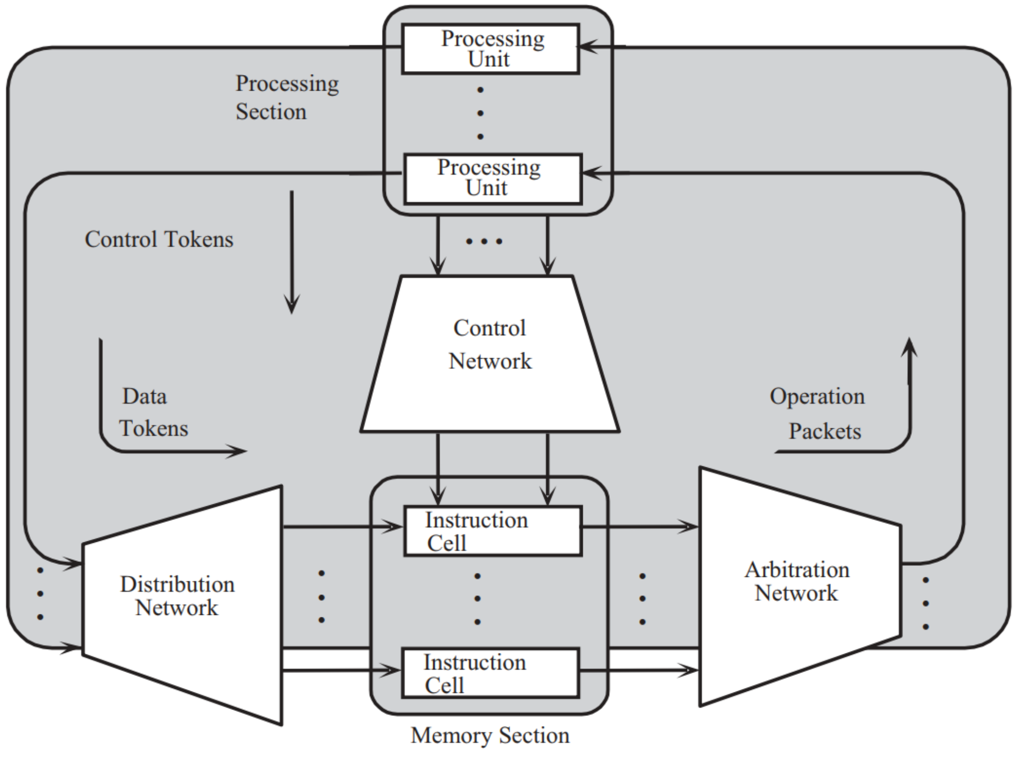


Short, very fast wires, no R/W (save time, energy)
Memory Bank
PE
PE
PE
PE
PE
PE
PE
PE
PE
I/O occurs only at the edges
Memory Bank

- Massively parallel arrays of primitive processing elements, PEs (or data processing units, DPUs), triggered by data arrival
- No intermediate storage in memory
- Interconnection may be hard-wired or microcoded
- Data propagates through the structure as synchronous wave (“*systolic blood pressure*”)

Commonly used nowadays for matrix multiplication in neural network processors

Dataflow architecture: MIT Static Model (1975)

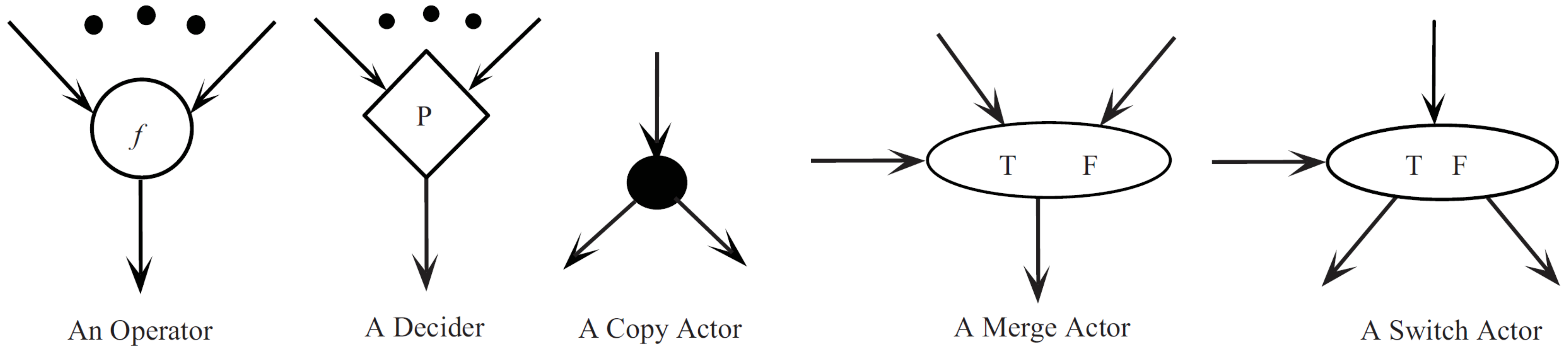


- No control flow
- No strict ordering and program counter: instructions (“nodes”) execute once operands are ready (explicit parallelism)
- No addressable RAM – computed data (“tokens”) are routed directly to consumer instructions
- ***Static***: one instruction cell = one activation

  no dynamic loops and recursion
- Machine language – dataflow graph

Ali R. Hurson, Krishna Kavi. Dataflow Computers: Their History and Future. Wiley Encyclopedia of Computer Science and Engineering. 2007

Basic dataflow graph primitives

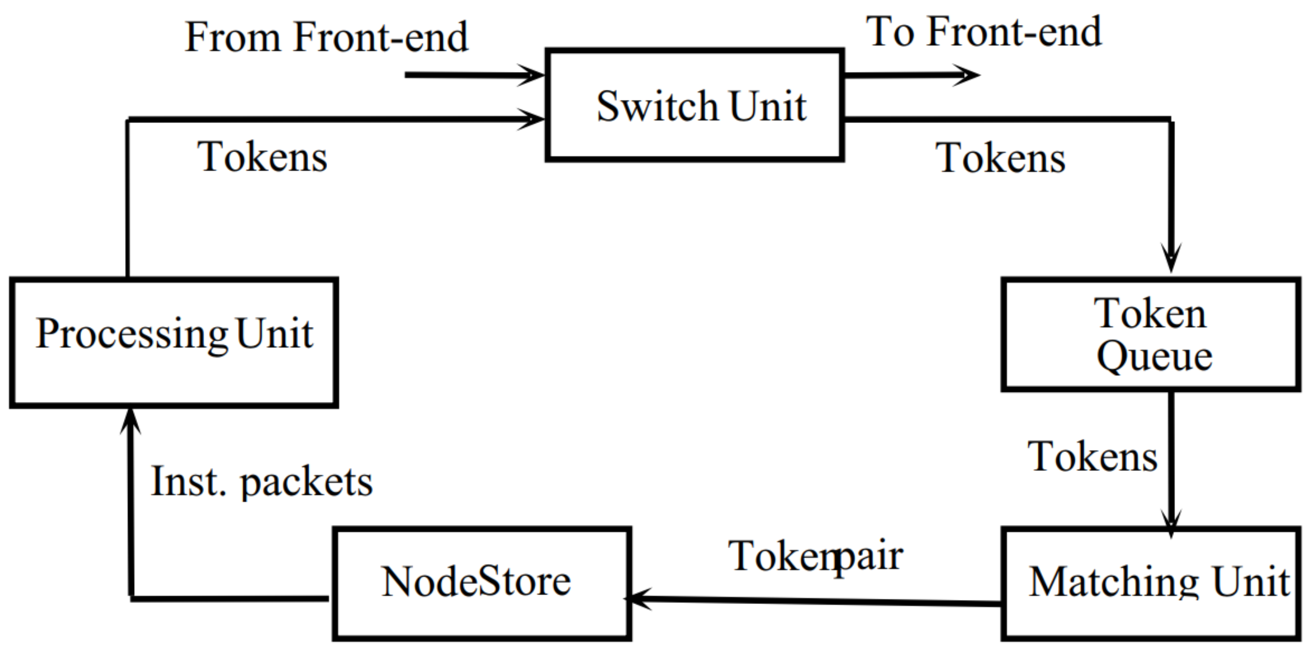


Language examples: *VAL, Id (Irvine dataflow), SISAL*.

Concepts closely relate to functional programming:

- no side effects
- single assignment (data token = unique value)
- locality of effect (no redundant dependencies)

Dataflow architecture: Manchester Dynamic Model (1990)



- Data tokens are ***tagged*** to identify context
  enables dynamic loops and recursion
- ***Dynamic***: several node instances can be fired in run-time
- Only tokens with identical tags are matched and make node executable
- ***Actual problem: deadlocks***

Common drawbacks of dataflow architectures

- Associative token matching in matching unit for the whole graph

  Bad for area and latency

- Long datapath: arbitration network + processing units + matching …

  RAW dependencies limit performance

- Difficult programming of complex, irregular algorithms

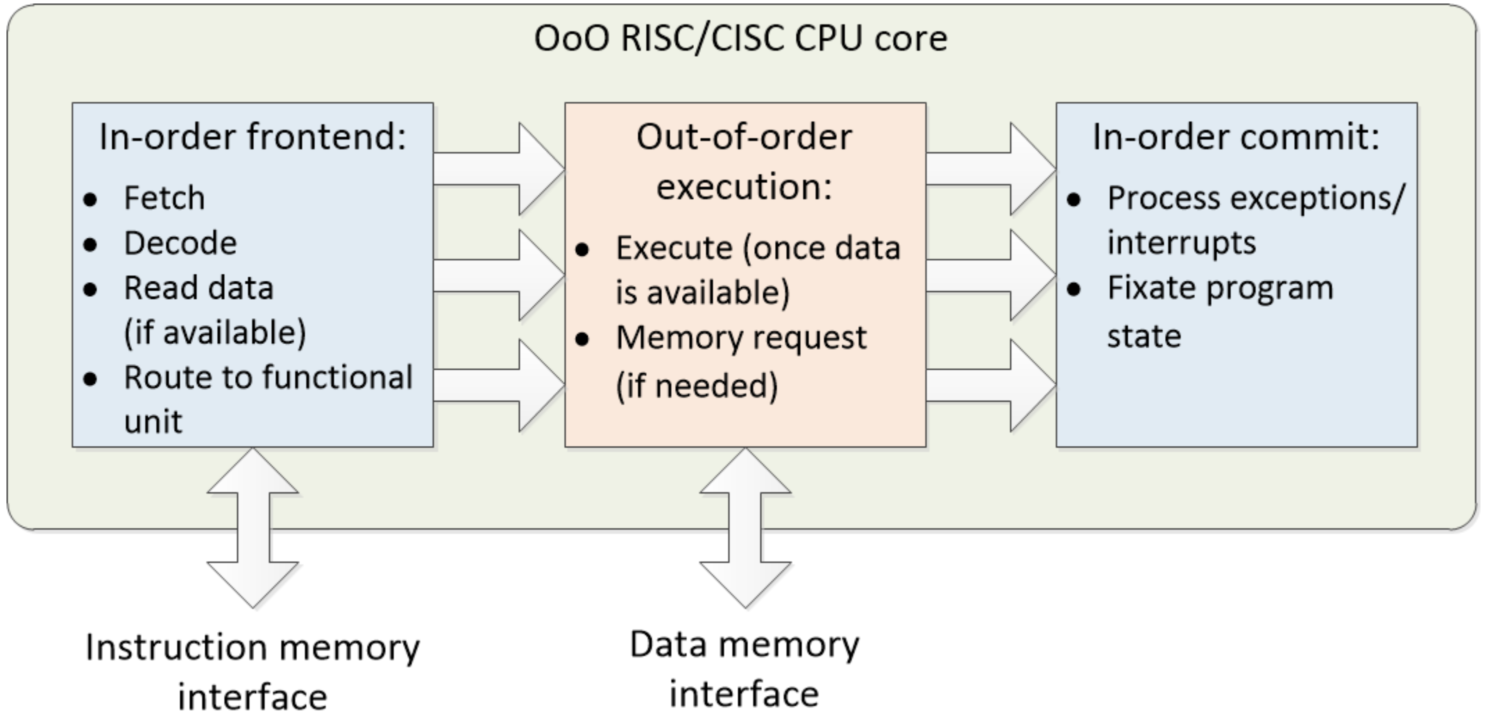
  Loops, recursion, pointer processing, …

- Difficult implementation of complex data structures

- Difficult deadlock avoidance in dynamic models

However …

General concept of out-of-order (OoO) processor structure


OoO RISC/CISC CPU core
In-order frontend:
• Fetch
• Decode
• Read data (if available)
• Route to functional unit
Out-of-order execution:
• Execute (once data is available)
• Memory request (if needed)
In-order commit:
• Process exceptions/ interrupts
• Fixate program state
Instruction memory interface
Data memory interface

Dataflow-like structure – implemented inside superscalar OoO processors

(more details in Lecture 10)

Microprogrammable and reconfigurable architectures

ITMO

Processors with microcode (1940s …)

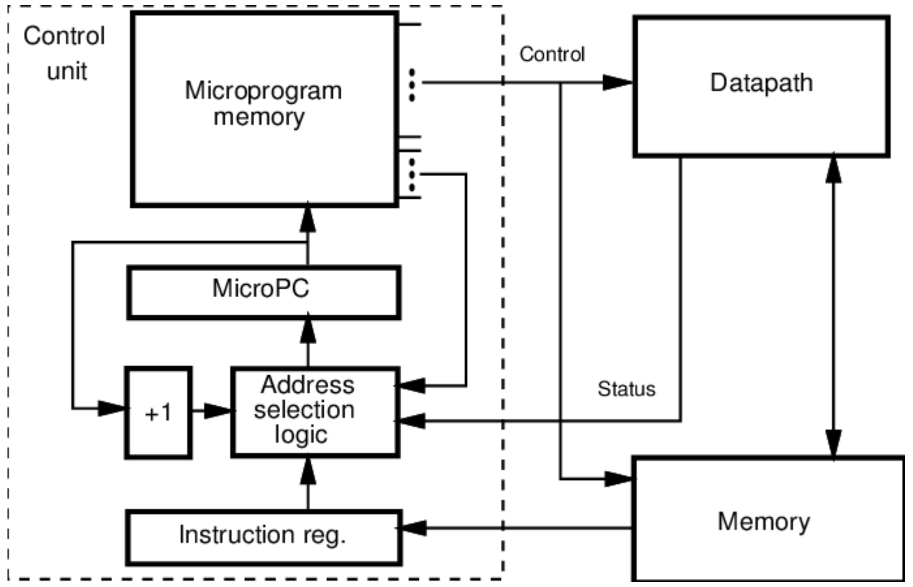
Complex instructions (e.g. CISC ones) can be translated to the series of ***simpler*** internal ones, ***residing in programmable memory*** and ***controlling individual hardware signals***

Can offer multiple virtues:

- Simplification of processor design
- Support of new instructions can be added
- Can emulate other architectures
- Patchable errors

But has drawbacks:

- Frequency slowdown (sequencing, memory, muxes)
- Inconvenient for pipelined execution



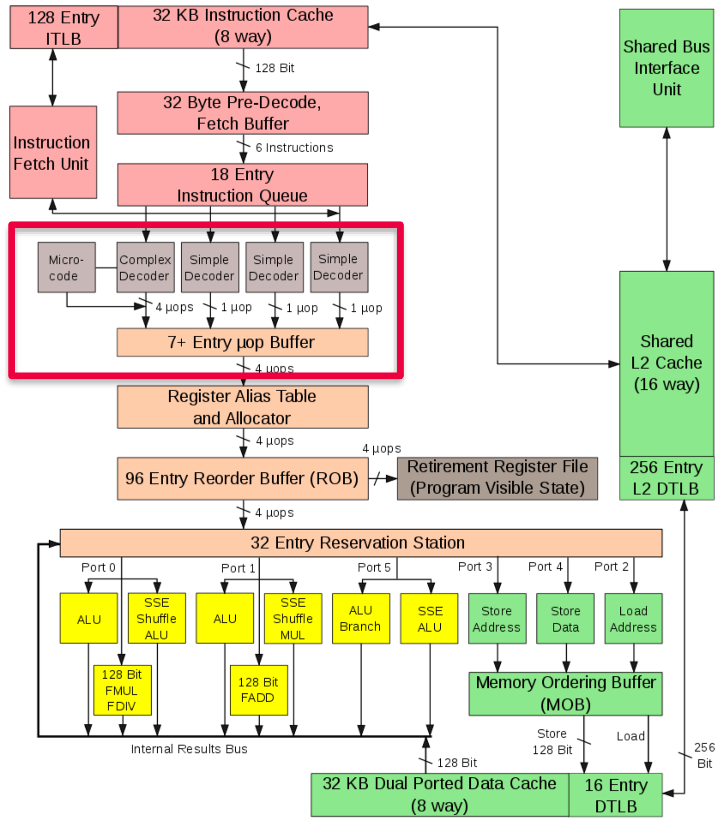
Modern microprogramming (e.g. Intel’s uops)

Modern high-performance processors (even RISC ones!) often have microcode for complex instructions

Complex instructions are translated to series of simpler ones (uops) that are managed ***by a hardwired pipeline***

Uops can be generated ***both from memory*** or ***by a hardwired decoder***

Sometimes referred to as “CISC-to-RISC” which is incorrect * – uops contain microarchitecture-specific information

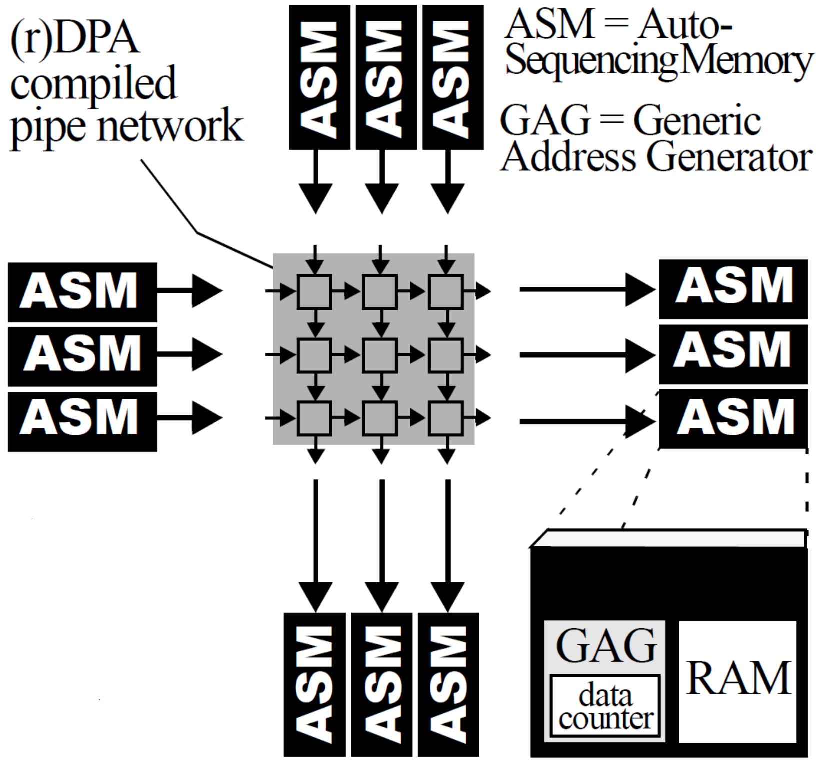


Intel Core 2 Architecture

* *https://www.quora.com/Why-are-RISC-processors-considered-faster-than-CISC-processors/answer/Bob-Colwell-1*

ITMO

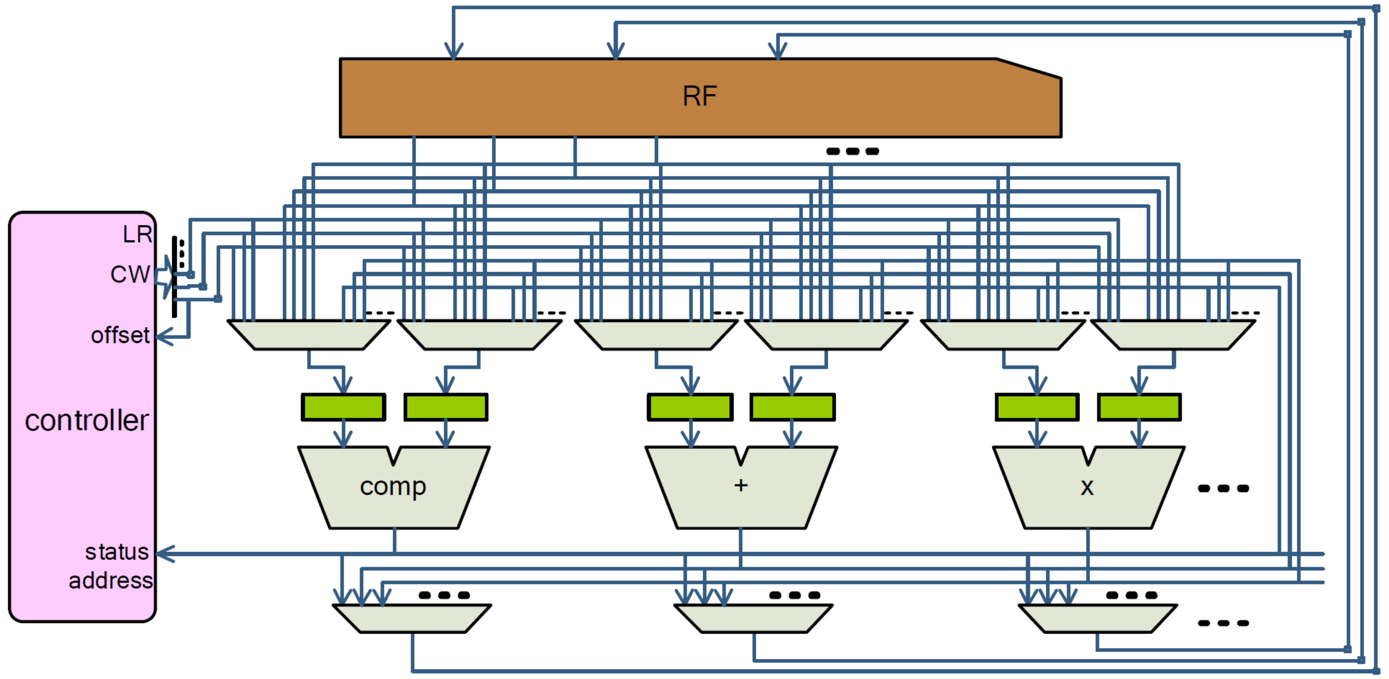
Anti-machine/Xputer (R. Hartenstein, 1990)


(r)DPA compiled pipe network
ASM = Auto-Sequencing Memory
GAG = Generic Address Generator
ASM
GAG
data counter
RAM

- **Anti-machine** – "counterpart of von Neumann machine"
- Focus: move from instruction streams programming to **data streams** programming (software -> "flowware")
- No program counter, replaced with **data counters**
- Programmer configures naturally parallel reconfigurable DataPath Array (rDPA) and data sequencers (ASM)

R. Hartenstein: The von Neumann Syndrome; SAMOS, Greece, July 2006.

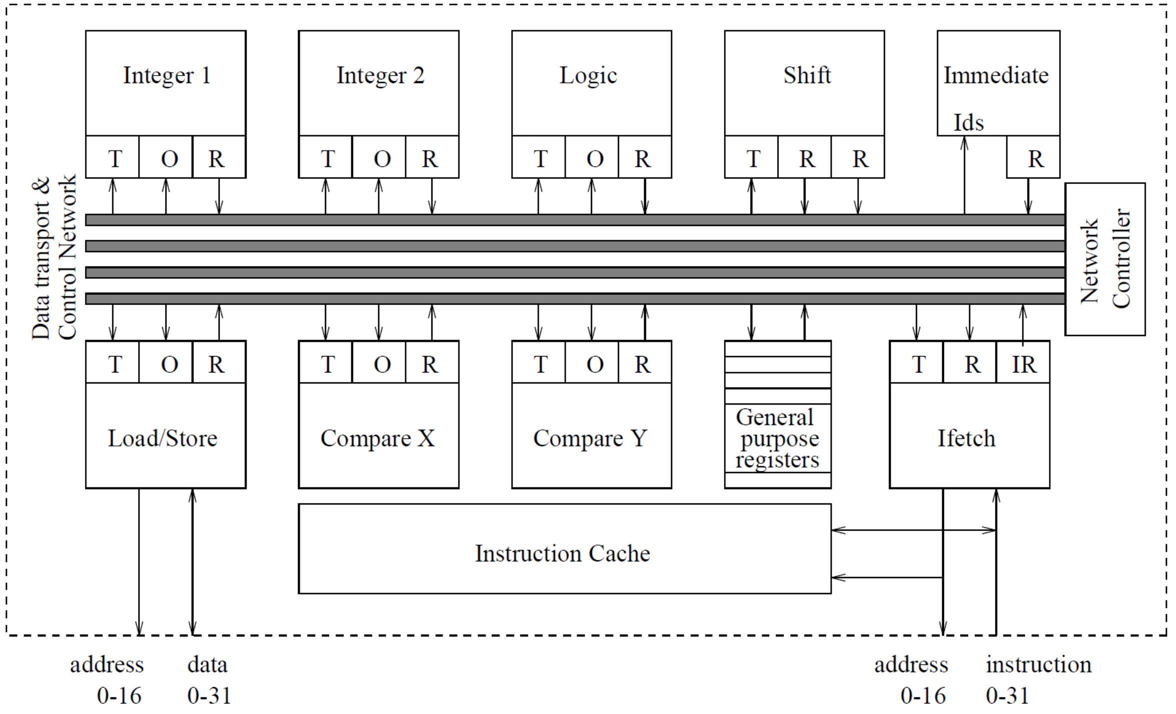
NISC: No Instruction Set Computer



- Processor datapath is explicitly ***exposed*** to software ("nanocode")
- Rarely used connections are trimmed
- Dynamic dependency identification and bypassing HW logic is trimmed

B. Gorjiara and D. Gajski, "Automatic architecture refinement techniques for customizing processing elements," 2008 45th ACM/IEEE Design Automation Conference, 2008, pp. 379-384

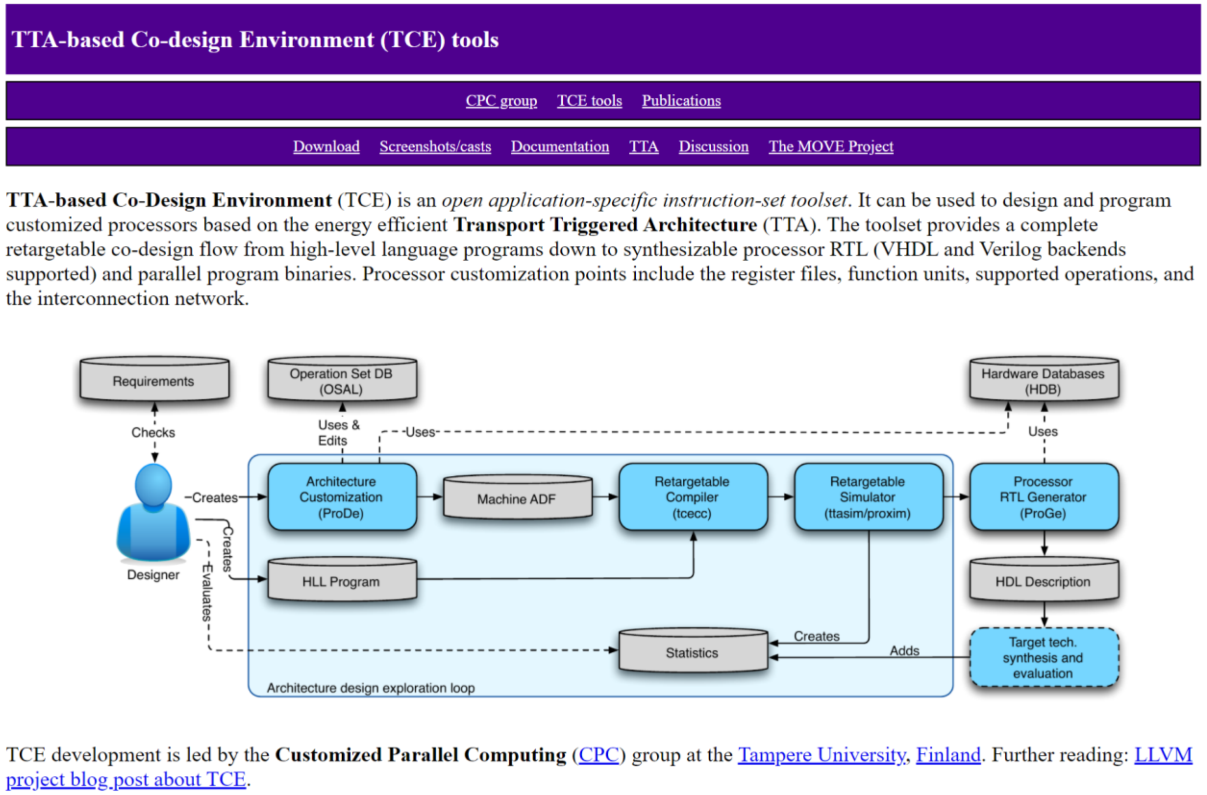
TTA: Transport Triggered Architecture



- Multiple FUs are connected to common bus infrastructure
- FUs have special ports for operands and results
- One of input FU ports is "triggering" – computations start when written

H. Corporaal, "Design of transport triggered architectures," Proceedings of 4th Great Lakes Symposium on VLSI, 1994, pp. 130-135

Example: TTA-based Co-design Environment (TCE)

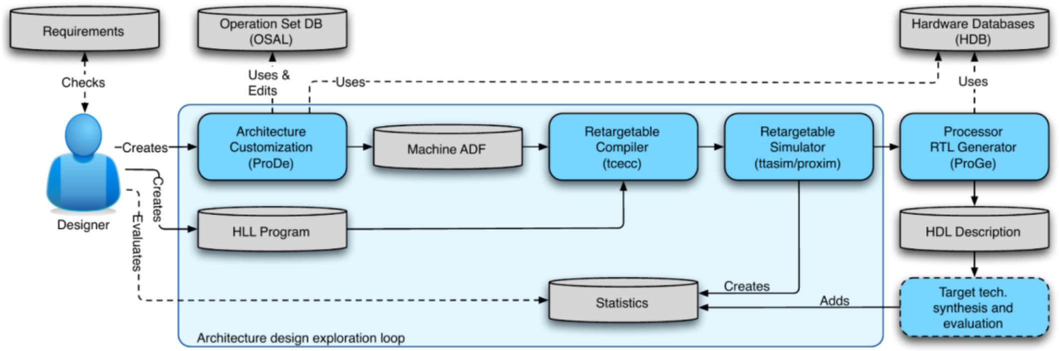


TTA-based Co-design Environment (TCE) tools

CPC group TCE tools Publications

Download Screenshots/casts Documentation TTA Discussion The MOVE Project

TTA-based Co-Design Environment (TCE) is an *open application-specific instruction-set toolset*. It can be used to design and program customized processors based on the energy efficient **Transport Triggered Architecture** (TTA). The toolset provides a complete retargetable co-design flow from high-level language programs down to synthesizable processor RTL (VHDL and Verilog backends supported) and parallel program binaries. Processor customization points include the register files, function units, supported operations, and the interconnection network.


Requirements
Operation Set DB (OSAL)
Hardware Databases (HDB)
Checks
Uses & Edits
Uses
Uses
Designer
Creates
Creates
Evaluates
Architecture Customization (ProDe)
Machine ADF
Retargetable Compiler (tcecc)
Retargetable Simulator (ttasim/proxim)
Processor RTL Generator (ProGe)
HLL Program
HDL Description
Creates
Adds
Statistics
Target tech. synthesis and evaluation
Architecture design exploration loop

TCE development is led by the **Customized Parallel Computing** (CPC) group at the Tampere University, Finland. Further reading: LLVM project blog post about TCE.

URL: http://openasip.org/
Repo: https://github.com/cpc/tce

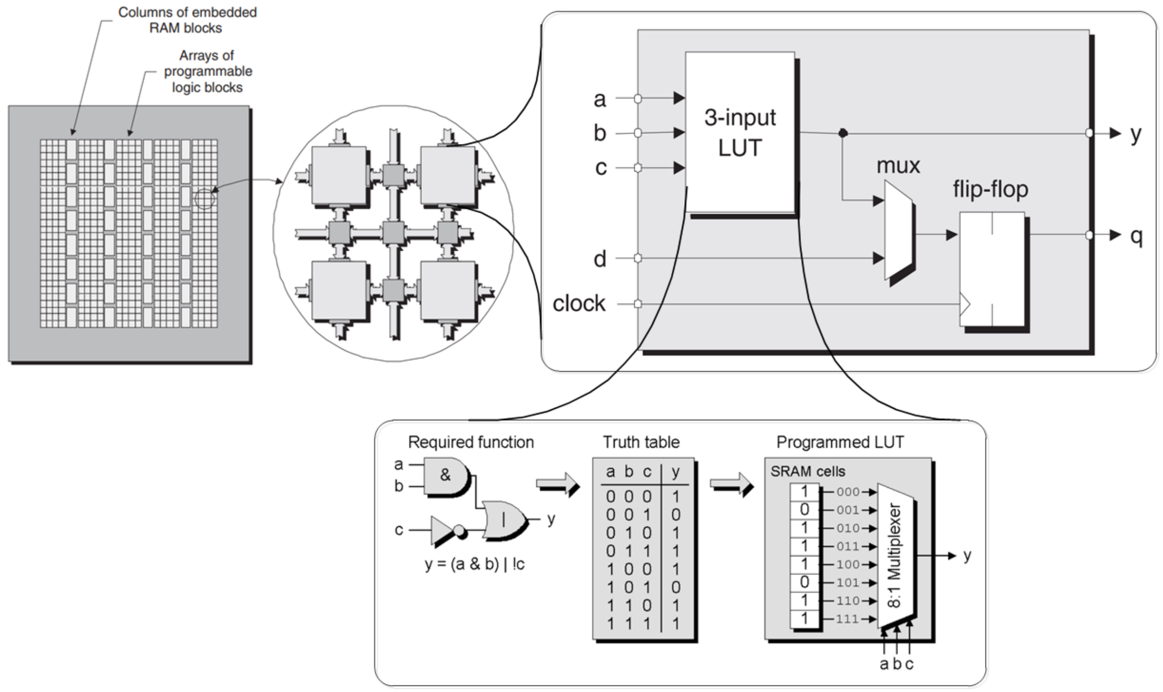
FPGA: Field Programmable Gate Array (1985 …)

Mesh of programmable elements providing precise software-defined control of hardware signals transformation and interconnection

Allow to implement custom digital logic

Can also contain:

- clock managers
- embedded RAM (BRAM)
- multipliers
- high-speed transceivers
- DAC/ADC
- CPU cores
- hard accelerators
- etc. …

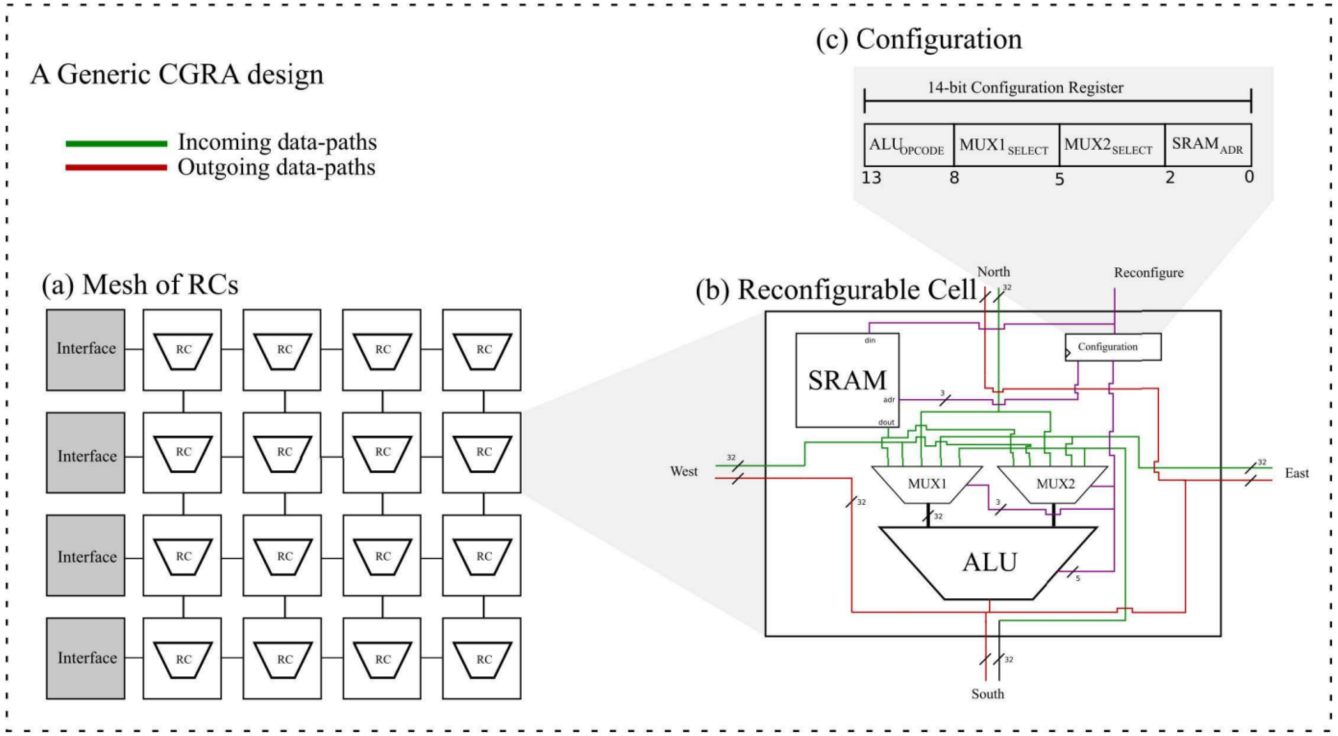


a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

C. Maxfield, The Design Warrior's Guide to FPGAs: Devices, Tools and Flows, 2002

ITMO

CGRA: Coarse-Grained Reconfigurable Array (1990s …)



- Mesh of processing elements

  Reconfigurable Cells, RCs

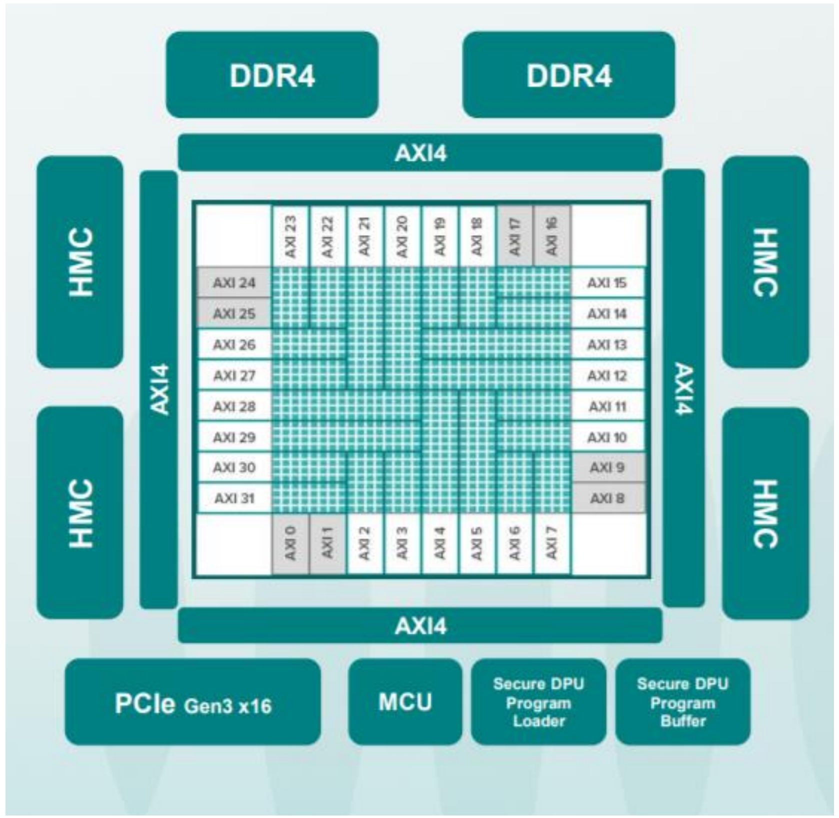
- RCs: simpler than CPU cores, but more complex than FPGA logic cells

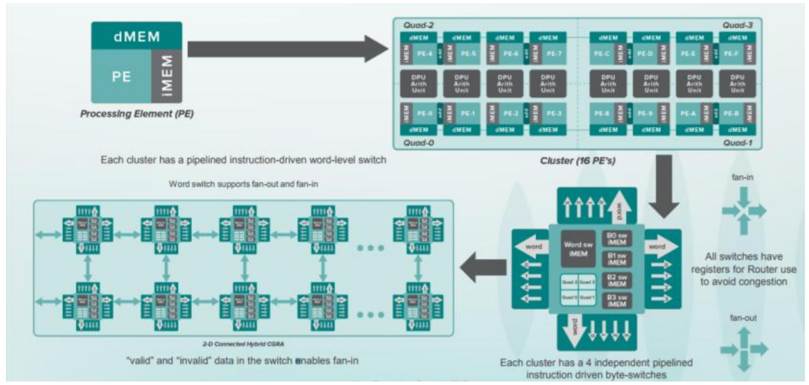
- Less flexible that FPGAs, but work on higher frequencies

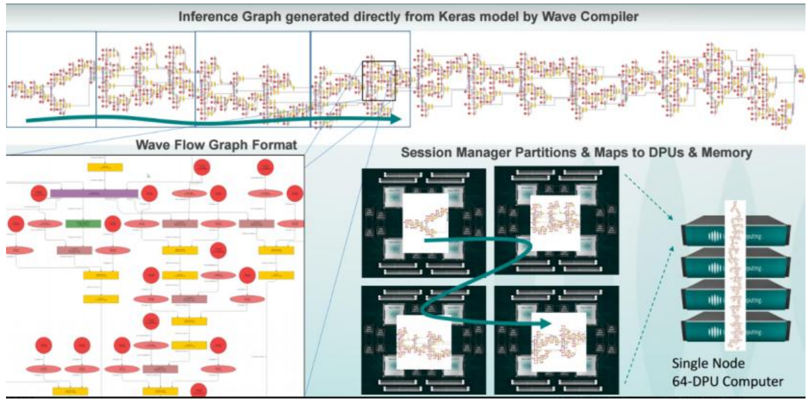
Considered efficient for neural networks processing

A. Podobas, K. Sano and S. Matsuoka, A survey on coarse-grained reconfigurable architectures from a performance perspective", IEEE Access, Vol. 8, 2020.

Commercial CGRA example: Data Processing Unit (Wave Computing)




dMEM
PE
iMEM
Processing Element (PE)
Quad-2
Quad-3
Quad-0
Quad-1
DPU Arith Unit
Cluster (16 PE's)
Each cluster has a pipelined instruction-driven word-level switch
Word switch supports fan-out and fan-in
2-D Connected Hybrid CGRA
"valid" and "invalid" data in the switch enables fan-in
word
Word sw iMEM
B0 sw iMEM
B1 sw iMEM
B2 sw iMEM
B3 sw iMEM
fan-in
All switches have registers for Router use to avoid congestion
fan-out
Each cluster has a 4 independent pipelined instruction driven byte-switches


Inference Graph generated directly from Keras model by Wave Compiler
Wave Flow Graph Format
Session Manager Partitions & Maps to DPUs & Memory
Single Node
64-DPU Computer

https://www.nextplatform.com/2017/08/23/first-depth-view-wave-computings-dpu-architecture-systems

Common drawbacks of software micro-management

- Extra HW overhead

  Extra multiplexers, registers, RAMs, programming mechanisms, …

- Large binaries

  Binaries define in detail what happens in HW

- Weak architecture/microarchitecture decoupling, no binaries interoperability

  Binaries are tied tightly to certain HW implementations

- Complex, highly specialized tooling

  Low level of hardware abstraction

- Strict synchronization, poor adaptability to dynamism of computational process

  Variable-latency instructions, branch prediction fails, cache misses, shared resources conflicts, multicore and I/O communication, …

Can be suitable for specific applications, but for now multicore CISC/RISC are dominating as General-Purpose Processors (GPP)

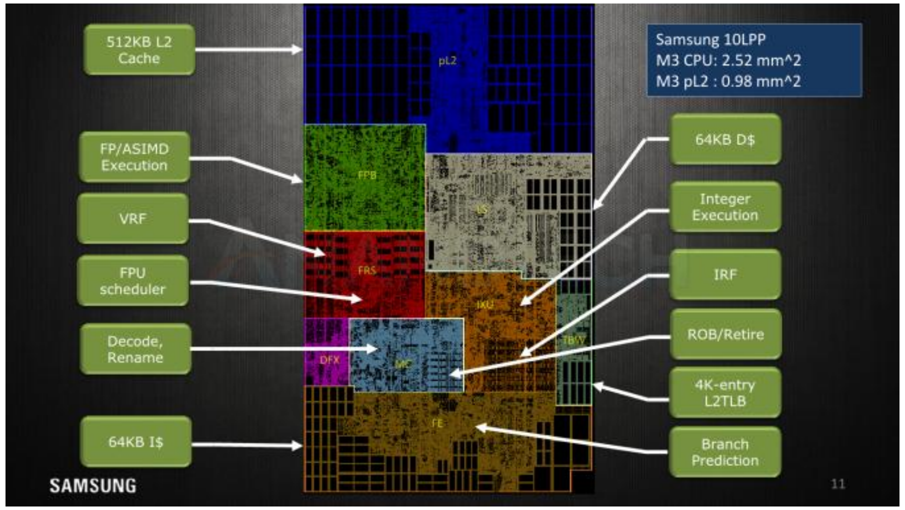
Domain-specific architectures. Systems-on-chip

Modern general-purpose CPU cores layouts

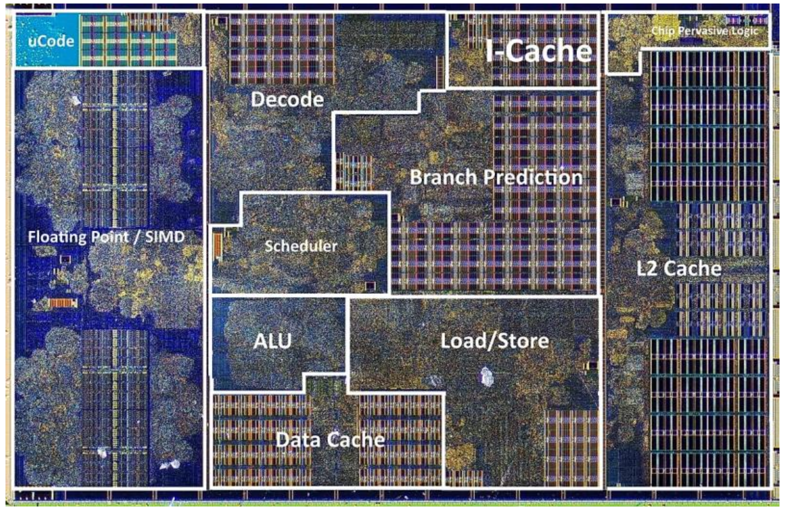
"Computational" blocks (execution units) occupy only a (small) ***fraction*** of CPU chip area

Majority of chip area is occupied by ***speculative runtime management/optimization engines***

branch prediction, out-of-order execution schedulers, caches, etc.



Samsung Exynos M3 CPU (2018)*



AMD Zen 2 CPU (2019)**

** – Andrei Frumusanu. Hot Chips 2018: Samsung's Exynos-M3 CPU Architecture Deep Dive, August 20, 2018,*
https://www.anandtech.com/show/13199/hot-chips-2018-samsungs-exynosm3-cpu-architecture-deep-dive/3

**** – https://www.flickr.com/photos/130561288@N04/49045449908/

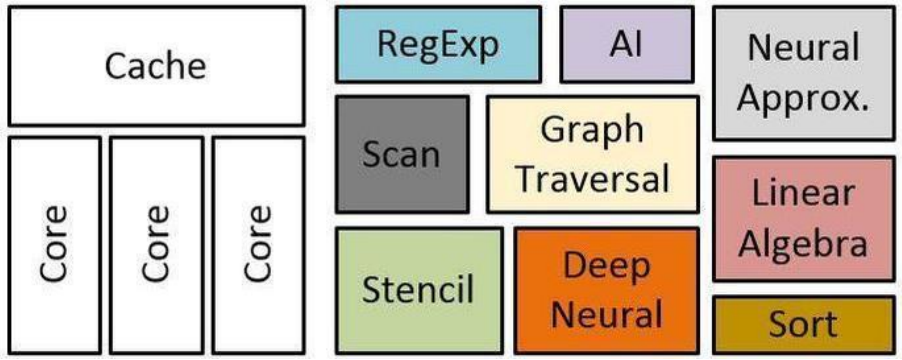
Domain-specific architectures (DSA)

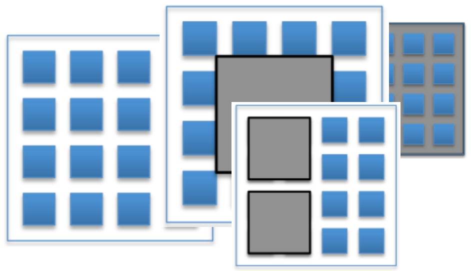
Strategy: offload ***critical functions*** to ***domain-specific architectures***

Use GPPs for the rest of (non-critical) workload

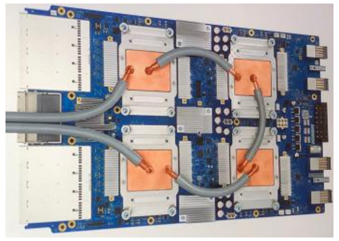
Key DSA concept:
shifting computational process optimization
from run-time to design-time

Less management-related runtime work →
more raw power, better performance,
power savings



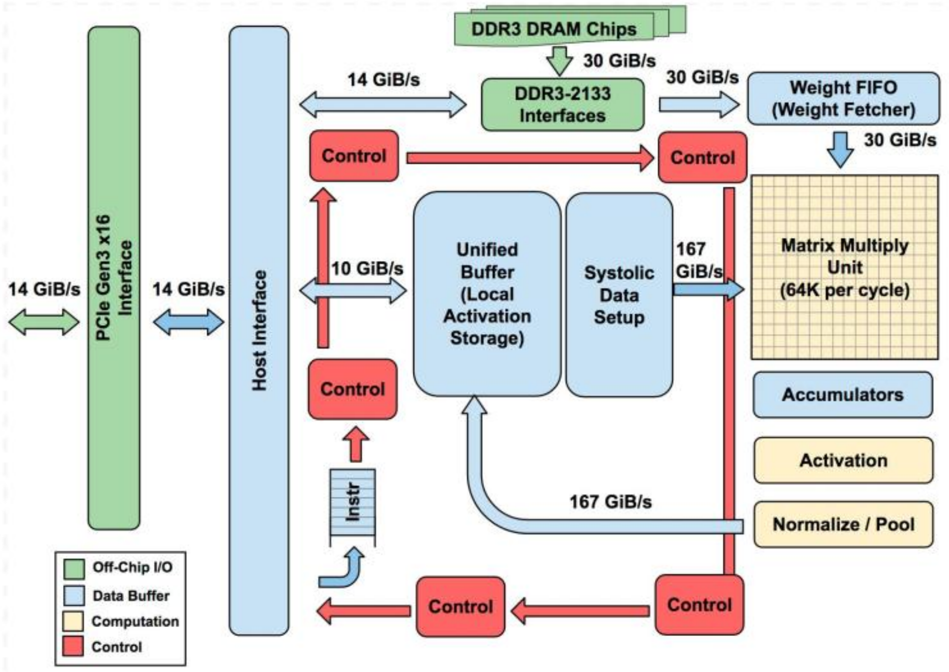


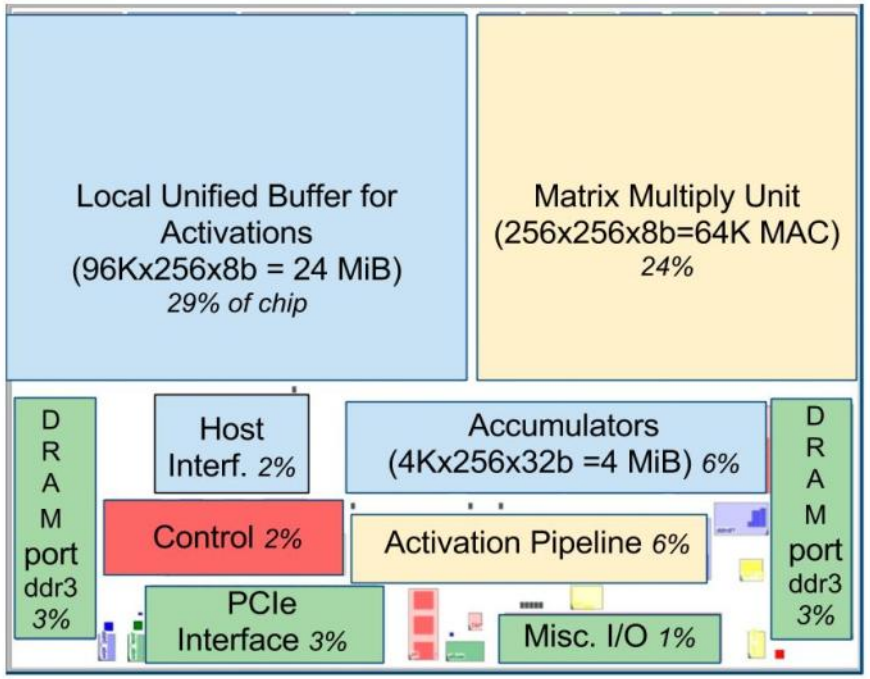
ITMO



DSA example: Google TPU

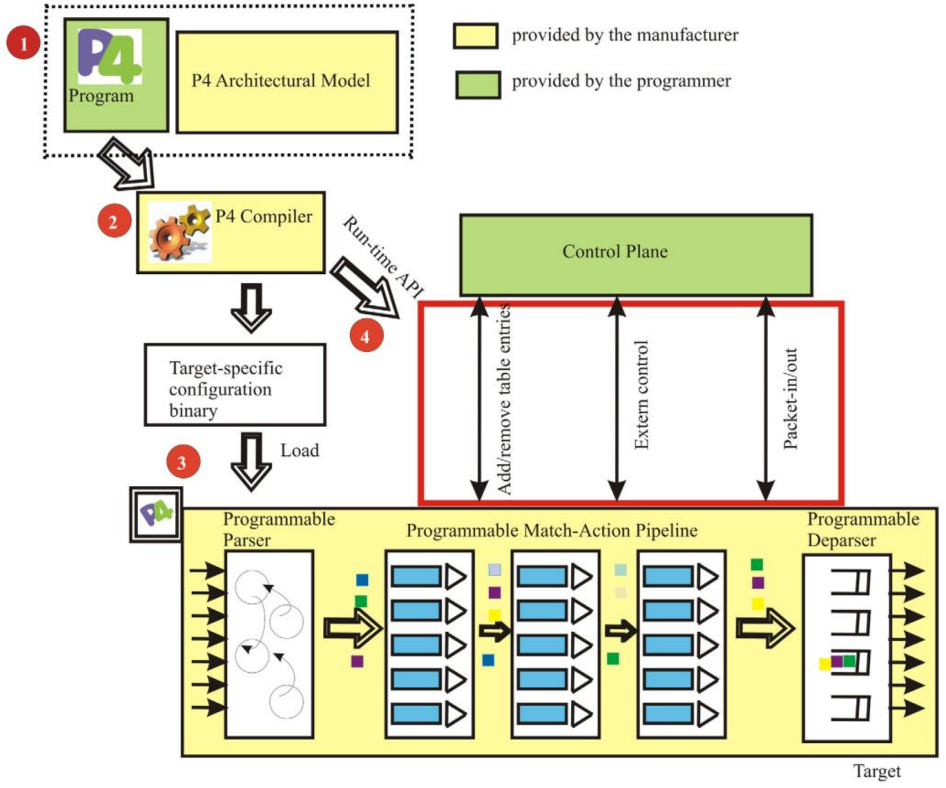
Coprocessor for ***neural network inference*** and (from 2nd gen) ***training:***




Local Unified Buffer for Activations (96Kx256x8b = 24 MiB) 29% of chip
Matrix Multiply Unit (256x256x8b=64K MAC) 24%
DRAM port ddr3 3%
Host Interf. 2%
Accumulators (4Kx256x32b =4 MiB) 6%
DRAM port ddr3 3%
Control 2%
Activation Pipeline 6%
PCIe Interface 3%
Misc. I/O 1%

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017

DSA example: P4-programmable NICs



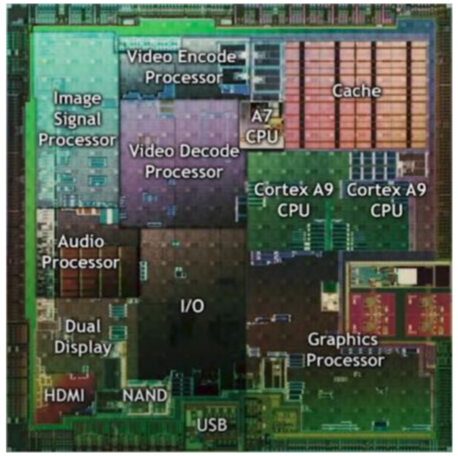
Sukhveer Kaur, Krishan Kumar, Naveen Aggarwal, A review on P4-Programmable data planes: Architecture, research efforts, and future directions, Computer Communications, Vol. 170, 2021, P. 109-129

Systems-on-chip (SoC)

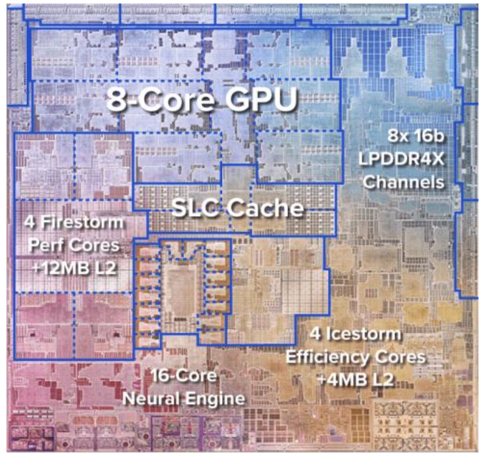
- ***System on chip*** - an integrated circuit that integrates all or most components of a computer or other electronic system

  *CPUs, **custom application-specific accelerators (Graphics/Radio/Neural Processing Units)**, I/O controllers, memory, analog*

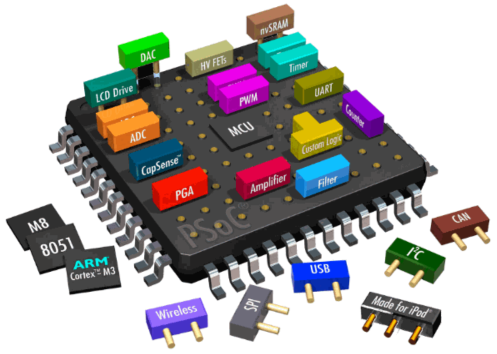
- ***System on chip*** – single-chip computer, designed primarily from reusable logic blocks (semiconductor intellectual property cores, ***IP cores***)



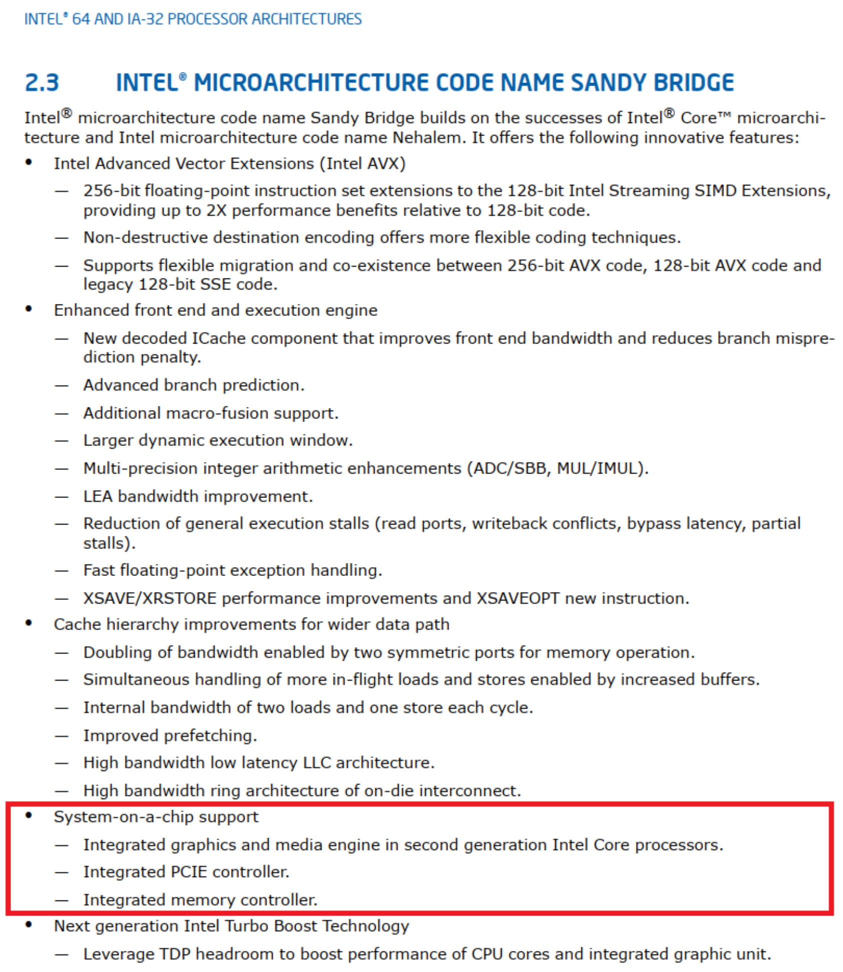
NVIDIA Tegra SoC


8-Core GPU
8x 16b LPDDR4X Channels
SLC Cache
4 Firestorm Perf Cores +12MB L2
4 Icestorm Efficiency Cores +4MB L2
16-Core Neural Engine

Apple M1 SoC



Intel CPUs as SoCs: Sandy Bridge (2011)

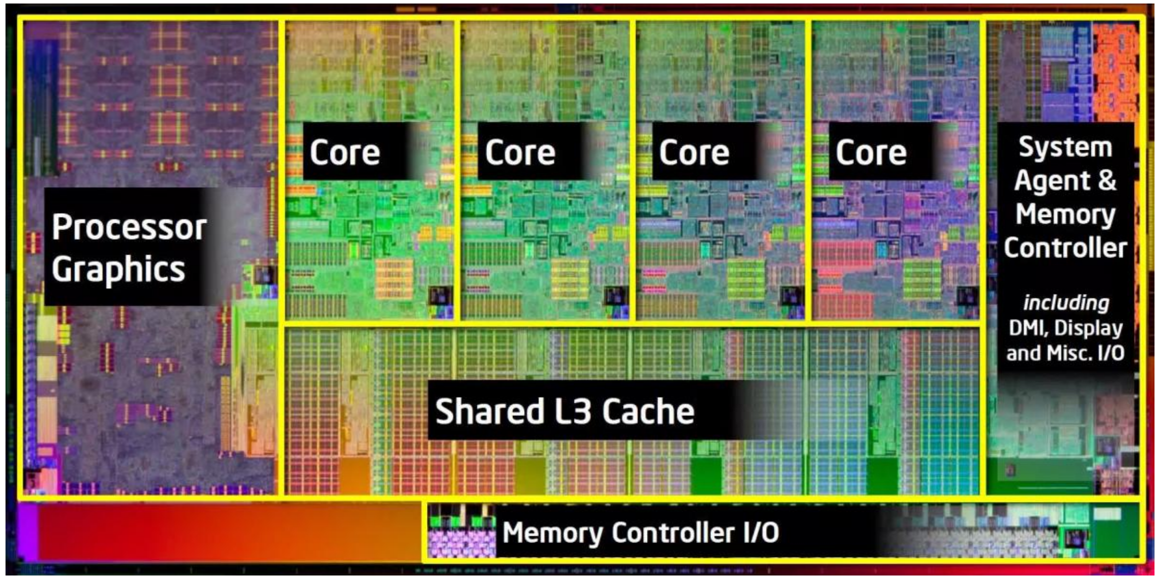


INTEL® 64 AND IA-32 PROCESSOR ARCHITECTURES

2.3 INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

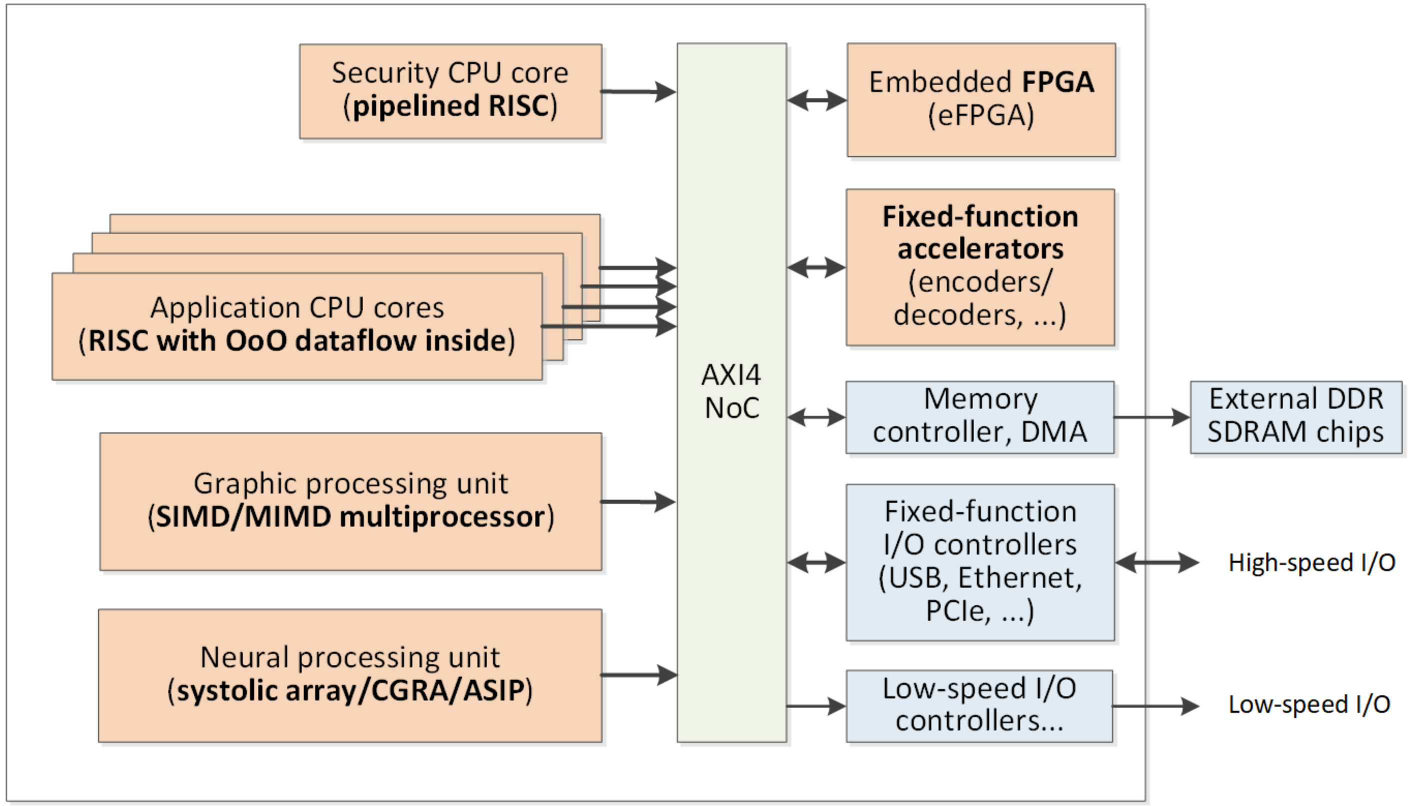
Intel® microarchitecture code name Sandy Bridge builds on the successes of Intel® Core™ microarchitecture and Intel microarchitecture code name Nehalem. It offers the following innovative features:

- Intel Advanced Vector Extensions (Intel AVX)
  - — 256-bit floating-point instruction set extensions to the 128-bit Intel Streaming SIMD Extensions, providing up to 2X performance benefits relative to 128-bit code.
  - — Non-destructive destination encoding offers more flexible coding techniques.
  - — Supports flexible migration and co-existence between 256-bit AVX code, 128-bit AVX code and legacy 128-bit SSE code.
- Enhanced front end and execution engine
  - — New decoded ICache component that improves front end bandwidth and reduces branch misprediction penalty.
  - — Advanced branch prediction.
  - — Additional macro-fusion support.
  - — Larger dynamic execution window.
  - — Multi-precision integer arithmetic enhancements (ADC/SBB, MUL/IMUL).
  - — LEA bandwidth improvement.
  - — Reduction of general execution stalls (read ports, writeback conflicts, bypass latency, partial stalls).
  - — Fast floating-point exception handling.
  - — XSAVE/XRSTORE performance improvements and XSAVEOPT new instruction.
- Cache hierarchy improvements for wider data path
  - — Doubling of bandwidth enabled by two symmetric ports for memory operation.
  - — Simultaneous handling of more in-flight loads and stores enabled by increased buffers.
  - — Internal bandwidth of two loads and one store each cycle.
  - — Improved prefetching.
  - — High bandwidth low latency LLC architecture.
  - — High bandwidth ring architecture of on-die interconnect.
- System-on-a-chip support
  - — Integrated graphics and media engine in second generation Intel Core processors.
  - — Integrated PCIE controller.
  - — Integrated memory controller.
- Next generation Intel Turbo Boost Technology
  - — Leverage TDP headroom to boost performance of CPU cores and integrated graphic unit.



Typical modern high-performance SoC internals

- High-performance SoCs contain multiple computational units with various organization
- These units might have different degree of specialization and require different approaches to programming



Security CPU core (**pipelined RISC**)
Application CPU cores (**RISC with OoO dataflow inside**)
Graphic processing unit (**SIMD/MIMD multiprocessor**)
Neural processing unit (**systolic array/CGRA/ASIP**)
AXI4 NoC
Embedded **FPGA** (eFPGA)
Fixed-function accelerators (encoders/ decoders, ...)
Memory controller, DMA
External DDR SDRAM chips
Fixed-function I/O controllers (USB, Ethernet, PCIe, ...)
High-speed I/O
Low-speed I/O controllers...
Low-speed I/O

ASIP: Application-Specific Instruction Processor

Thank you for the lesson!

Alexander Antonov, Assoc. Prof., antonov@itmo.ru

Hangzhou, 2025