Domain specific computing architectures

Asif Ali Khan

Fall Semester 2024

Department of Computer Systems Engineering

UET Peshawar, Pakistan

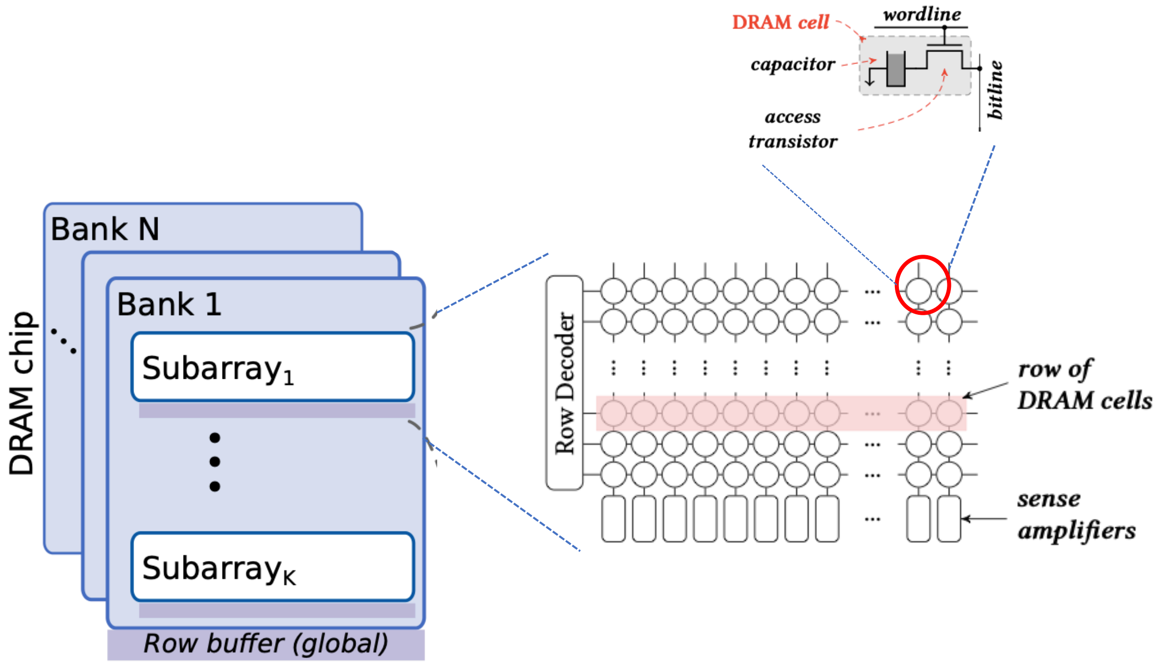
16 Oct 2024

Recap: Overview of Von-Neumann architectures

- Key Components
  - CPU
  - Memory
  - I/O
  - Bus (communication channel)
- Key Concept
  - Stored program concept
- Von Neumann Bottleneck
  - Performance is limited by the shared bus

Recap: Main memory subsystem

- ❑ Consists of channels, ranks, and banks
- ❑ Each bank consists of one or more subarrays
- ❑ A subarray is a grid of wordlines (rows) and bitlines (cols)
- ❑ To access a row, the previously opened row must be precharged first and then the new row needs to be activated



Sishadri et al, MICRO 2017
Lima et al., Arxiv, 2024

Instruction set architecture (ISA)

- The set of instruction a processor can execute

Instruction set architecture (ISA)

- The set of instruction a processor can execute
- An interface between hardware and software

Instruction set architecture (ISA)

- The set of instruction a processor can execute
- An interface between hardware and software
- ISA Types:
  - CISC: Richer instruction set, slower execution per instruction; variable instruction size
  - RISC: Simpler and fixed-size instructions, faster execution (typically 1 per clock cycle)

Instruction set architecture (ISA)

- The set of instruction a processor can execute
- An interface between hardware and software
- ISA Types:
  - CISC: Richer instruction set, slower execution per instruction; variable instruction size
  - RISC: Simpler and fixed-size instructions, faster execution (typically 1 per clock cycle)
- Examples
  - X86 (CISC), ARM (RISC)

Programmability of Von Neumann Systems

- The system only understands low-level machine code
  - Extremely hard (if not impossible) to write manually

Programmability of Von Neumann Systems

- The system only understands low-level machine code
  - Extremely hard (if not impossible) to write manually

- Assembly language raises the abstraction to human-readable form

```
MOV R1, #5   ; Load value 5 into register R1
ADD R2, R1, R3   ; Add contents of R1 and R3, store in R2
```

Programmability of Von Neumann Systems

- The system only understands low-level machine code
    - Extremely hard (if not impossible) to write manually
- Assembly language raises the abstraction to human-readable form

```
MOV R1, #5   ; Load value 5 into register R1
ADD R2, R1, R3   ; Add contents of R1 and R3, store in R2
```

- High-level languages (C/C++, python etc.) abstract from hardware details
- Require high-level compilers to lower to the low-level machine code
    - gcc, g++, clang etc.

Compiler overview

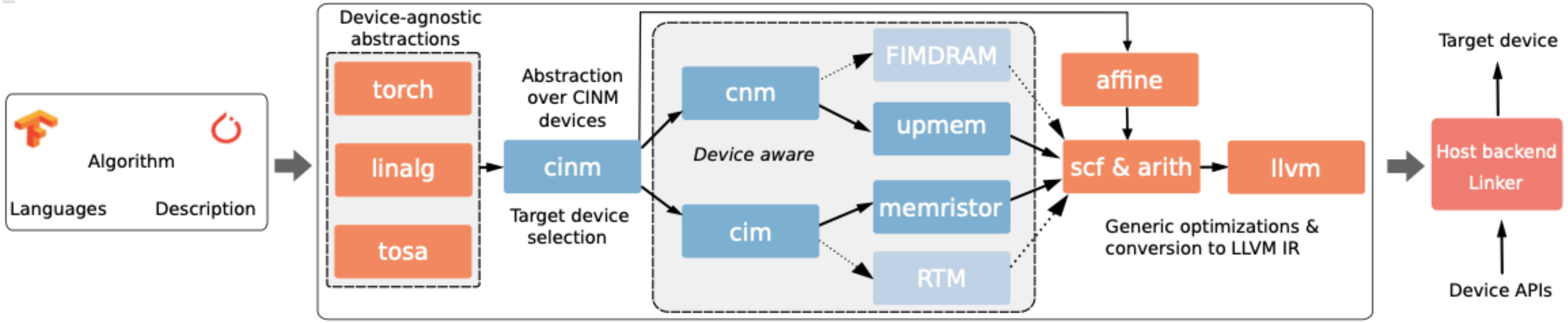
- The compiler takes care of:
  - Lowering the high-level input program into machine code
  - In the process, performs a series of optimizations

Compiler overview

- The compiler takes care of:
  - Lowering the high-level input program into machine code
  - In the process, performs a series of optimizations
- Typically is defined by three submodules:
  - Front-end: Takes care of syntax and semantics analysis
  - Middle-end: Performs a bunch of optimizations, e.g., loop optimizations
  - Back-end: Takes care of register allocation, and hardware specific code generation

Compiler overview

- Typically is defined by three submodules:
  - Front-end: Takes care of syntax and semantics analysis
  - Middle-end: Performs a bunch of optimizations, e.g., loop optimizations
  - Back-end: Takes care of register allocation, and hardware specific code generation



Khan et al, ASPLOS 2024

Compiler overview

- The compiler takes care of:
  - Lowering the high-level input program into machine code
  - In the process, performs a series of optimizations
- Typically is defined by three submodules:
  - Front-end: Takes care of syntax and semantics analysis
  - Middle-end: Performs a bunch of optimizations, e.g., loop optimizations
  - Back-end: Takes care of register allocation, and hardware specific code generation
- Resource for more details:
  - *Compilers: Principles, Techniques, and Tools (the Dragon Book)*

Domain-specific computing architectures/accelerators

Domain specific architectures

- Specialized computing systems for a particular domain of applications

Domain specific architectures

- Specialized computing systems for a particular domain of applications
- DSAs are everywhere and do the heavy-lifting but are mostly invisible
  - E.g., the GPUs, CODECs etc in your phones

Domain specific architectures

- Specialized computing systems for a particular domain of applications
- DSAs are everywhere and do the heavy-lifting but are mostly invisible
  - E.g., the GPUs, CODECs etc in your phones
- Different from ASIC (which is often used for a single function)
  - Still programmable

Domain specific architectures

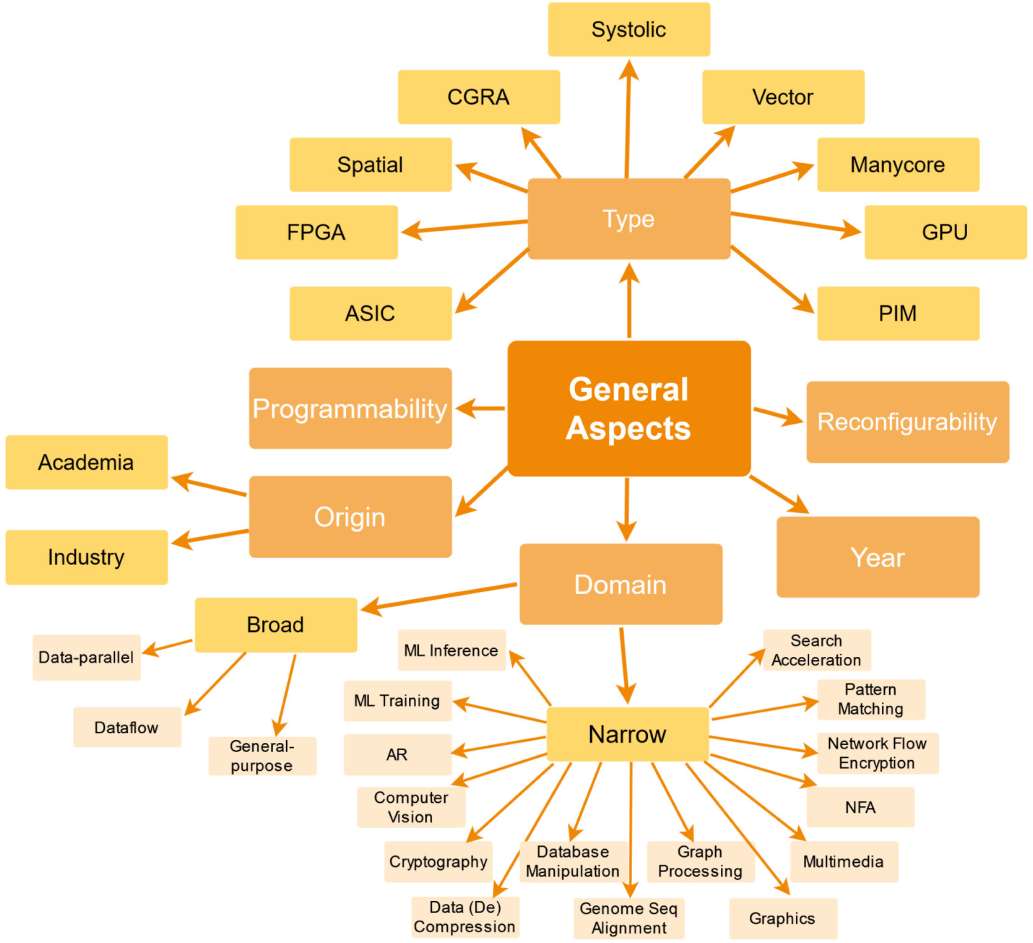
- Specialized computing systems for a particular domain of applications
- DSAs are everywhere and do the heavy-lifting but are mostly invisible
  - E.g., the GPUs, CODECs etc in your phones
- Different from ASIC (which is often used for a single function)
  - Still programmable
- Examples: GPUs, TPUs, DPUs, NPUs, QPUs etc.

Domain specific architectures

- Specialized computing systems for a particular domain of applications
- DSAs are everywhere and do the heavy-lifting but are mostly invisible
  - E.g., the GPUs, CODECs etc in your phones
- Different from ASIC (which is often used for a single function)
  - Still programmable
- Examples: GPUs, TPUs, DPUs, NPUs, QPUs etc.

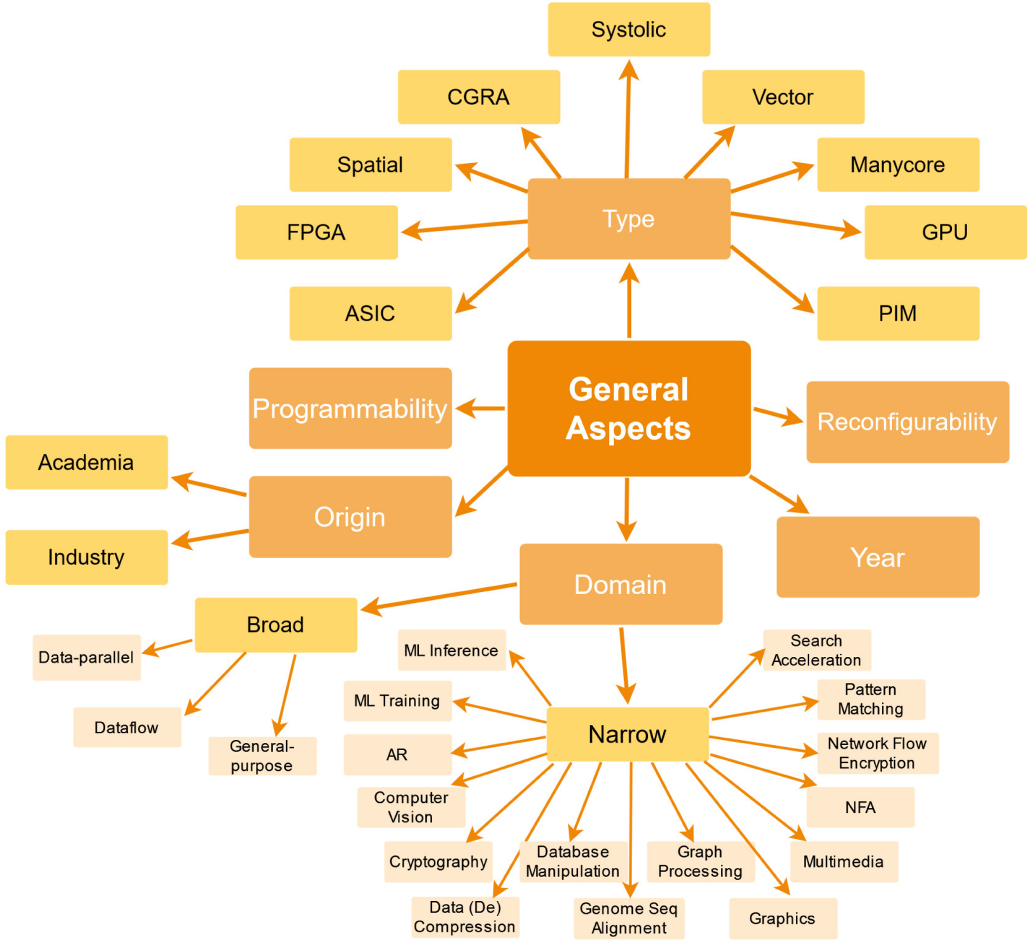
Domain specific architectures

- Many aspects are considered in DSA design



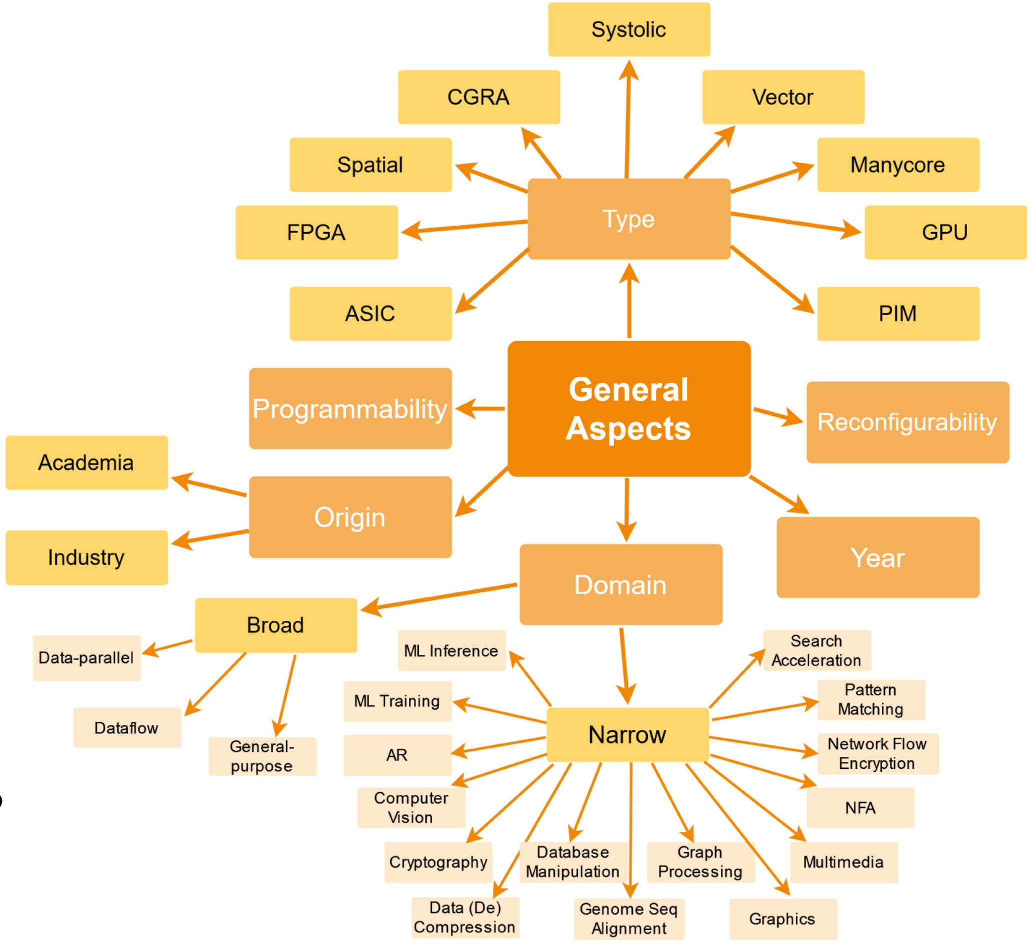
Domain specific architectures

- ❑ Many aspects are considered in DSA design
- ❑ We wouldn't cover all of them in this lecture but will touch specific instances

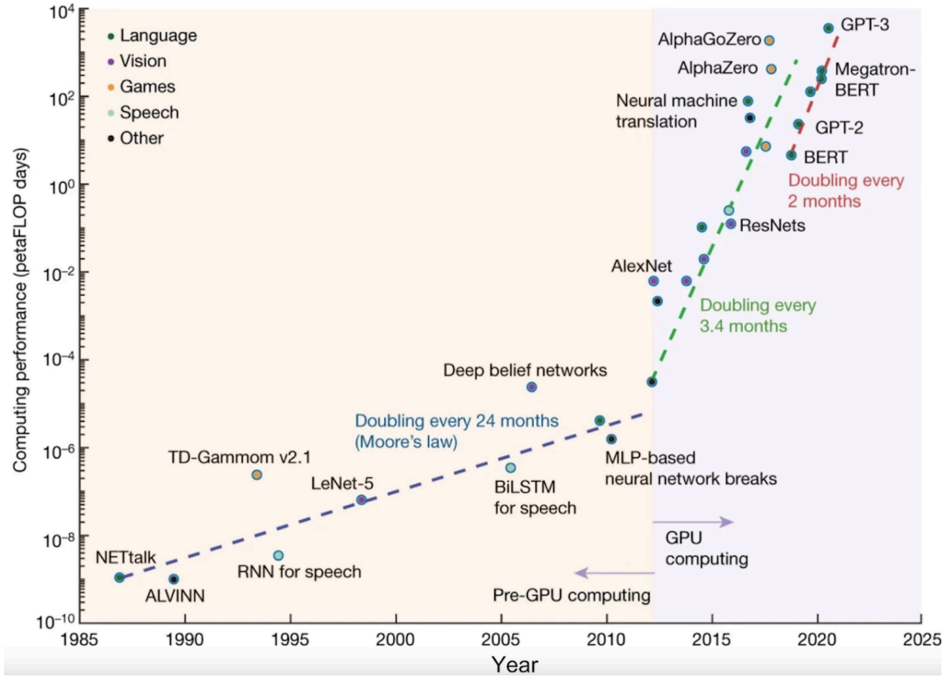


Domain specific architectures

- ❑ Many aspects are considered in DSA design
- ❑ We wouldn’t cover all of them in this lecture but will touch specific instances
- ❑ Details can be found in this overview paper: *Peccerillo et al., “A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives”, JSA 2022*

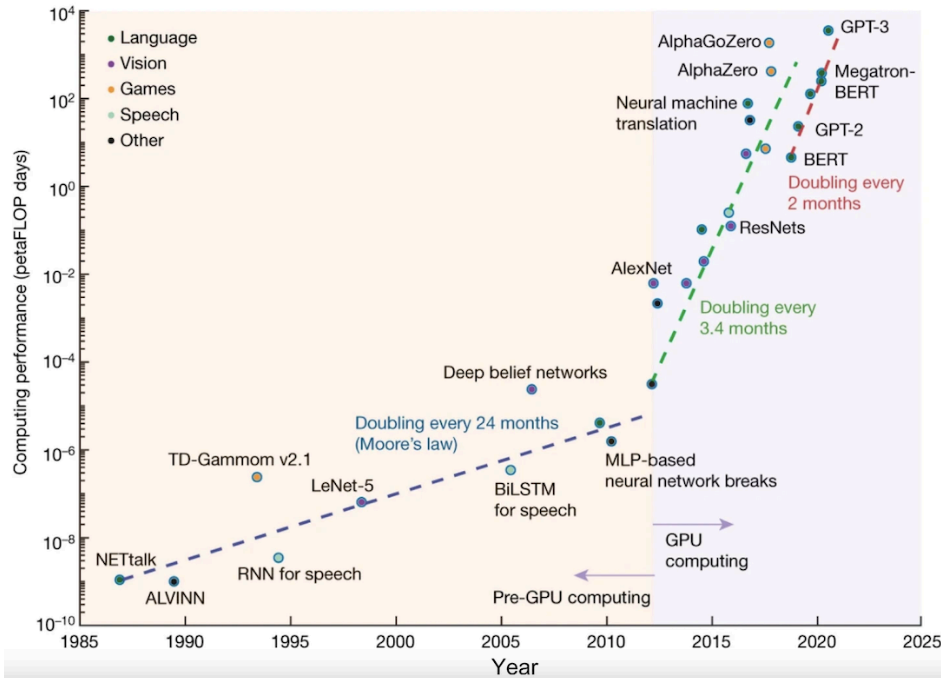


Why are DSAs needed?



Aguirre et al, Nature Communications, 2024

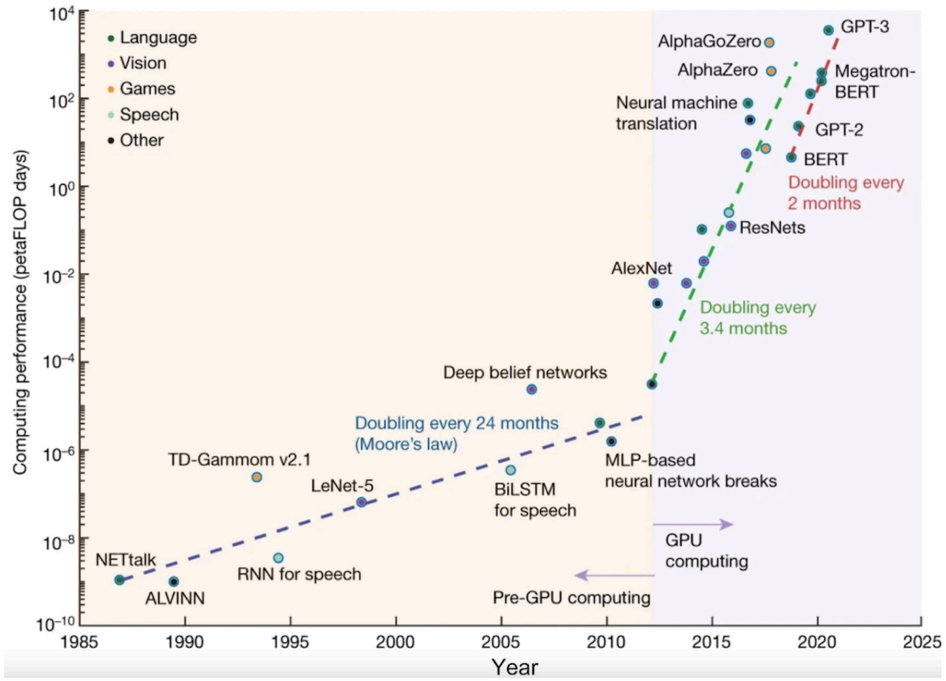
Why are DSAs needed?



Aguirre et al, Nature Communications, 2024

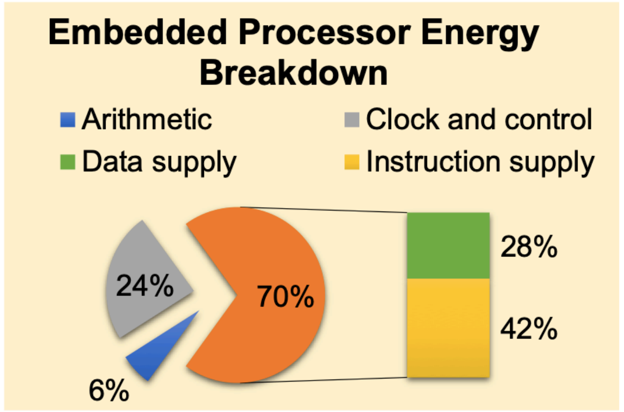
Traditional general-purpose (GP) computing can not fulfill the compute-demands of these applications due to their inefficiencies

Why are DSAs needed?



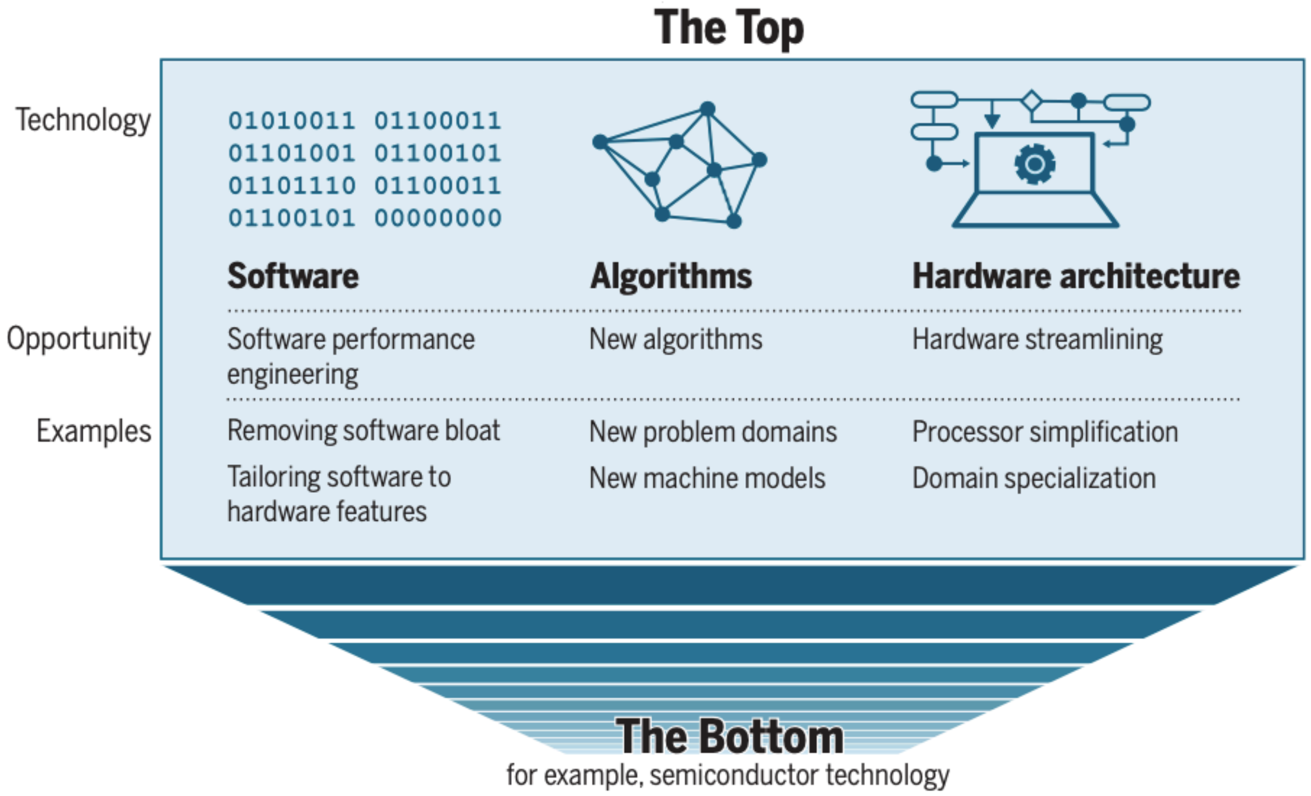
Aguirre et al, Nature Communications, 2024

Traditional general-purpose (GP) computing can not fulfill the compute-demands of these applications due to their inefficiencies



Dally et al, Efficient embedded computing, IEEE'08

Source of inefficiencies in GP computing



The Top

	Software	Algorithms	Hardware architecture
Technology	01010011 01100011 01101001 01100101 01101110 01100011 01100101 00000000		
Opportunity	Software performance engineering	New algorithms	Hardware streamlining
Examples	Removing software bloat Tailoring software to hardware features	New problem domains New machine models	Processor simplification Domain specialization

The Bottom
for example, semiconductor technology

Leiserson, C. et al. There's plenty of room at the top. Science, June 2020

Source of inefficiencies in GP computing

- Productivity oriented languages are slow

Source of inefficiencies in GP computing (the software side)

- Productivity oriented languages are slow

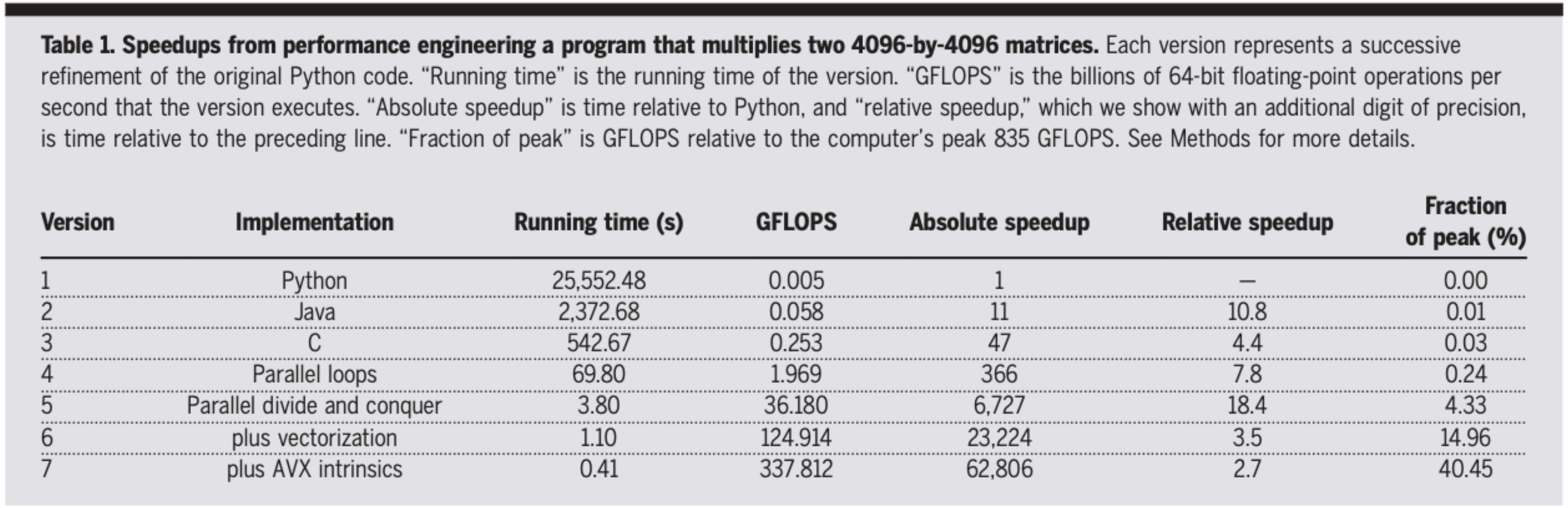


Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that the version executes. "Absolute speedup" is time relative to Python, and "relative speedup," which we show with an additional digit of precision, is time relative to the preceding line. "Fraction of peak" is GFLOPS relative to the computer's peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	–	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

Leiserson, C. et al. There's plenty of room at the top. Science, June 2020

Source of inefficiencies in GP computing (the software side)

- ❑ Productivity oriented languages are slow

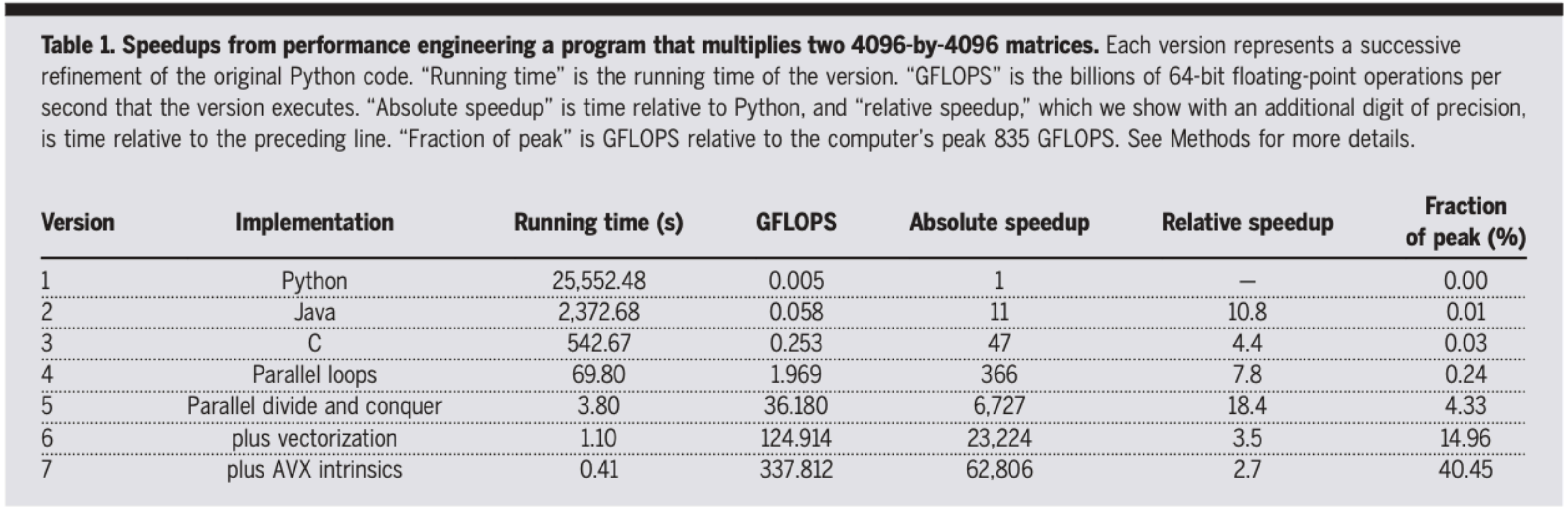


Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that the version executes. "Absolute speedup" is time relative to Python, and "relative speedup," which we show with an additional digit of precision, is time relative to the preceding line. "Fraction of peak" is GFLOPS relative to the computer's peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	–	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

- ❑ There is a huge between productivity and efficiency

Leiserson, C. et al. There's plenty of room at the top. Science, June 2020

Source of inefficiencies in GP computing (the software side)

- Productivity oriented languages are slow

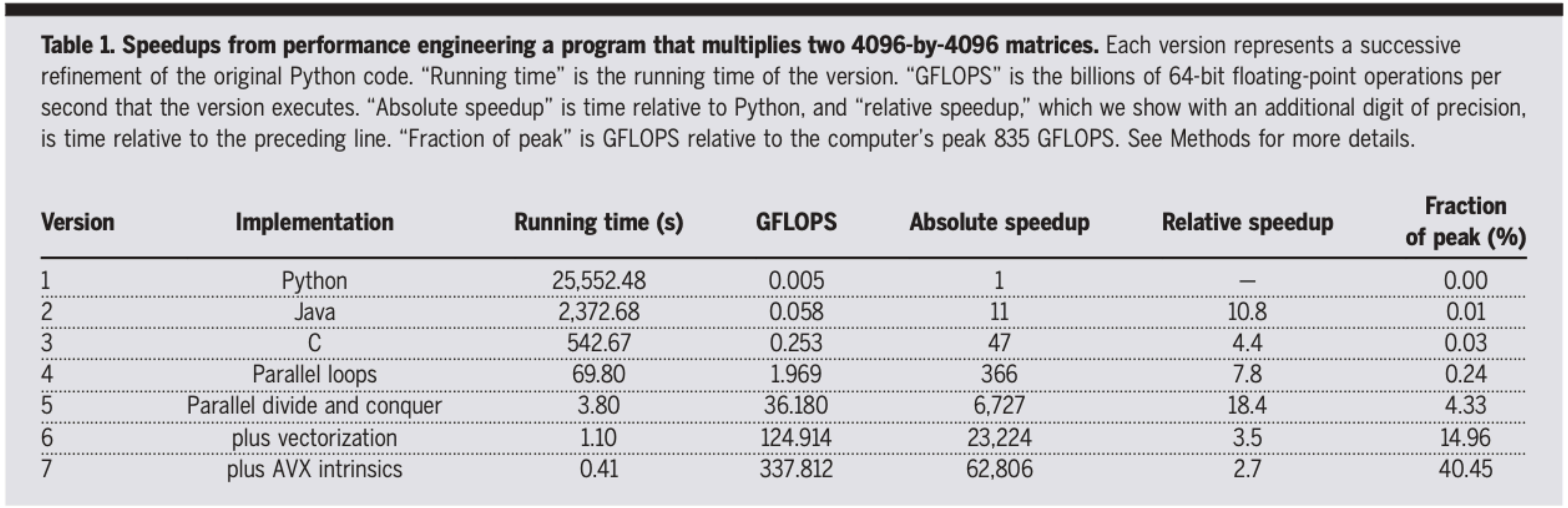


Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that the version executes. "Absolute speedup" is time relative to Python, and "relative speedup," which we show with an additional digit of precision, is time relative to the preceding line. "Fraction of peak" is GFLOPS relative to the computer's peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	–	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

- There is a huge between productivity and efficiency
- Domain specific languages (Matlab, Tensorflow etc.) are proposed to bridge this gap

Leiserson, C. et al. There's plenty of room at the top. Science, June 2020

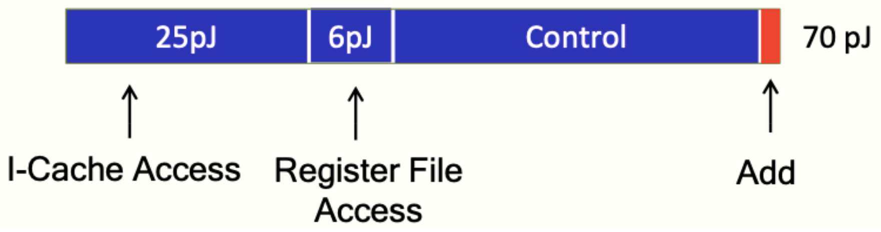
Source of inefficiencies in GP computing (the hardware side)

Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3.1pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1.1pJ
32 bit	3.7pJ

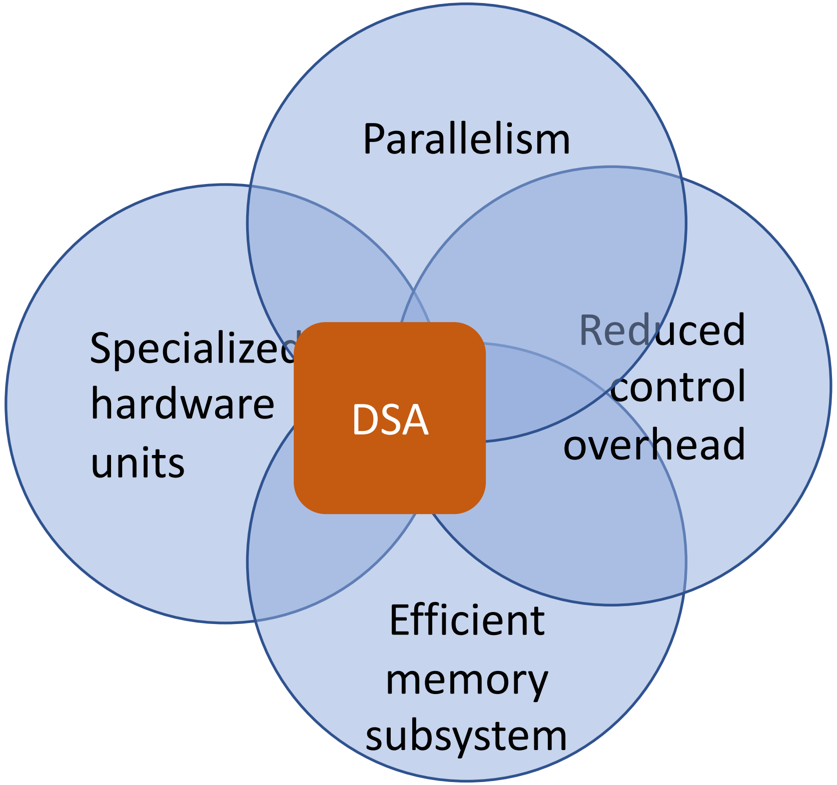
Memory	
Cache	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ

Instruction Energy Breakdown



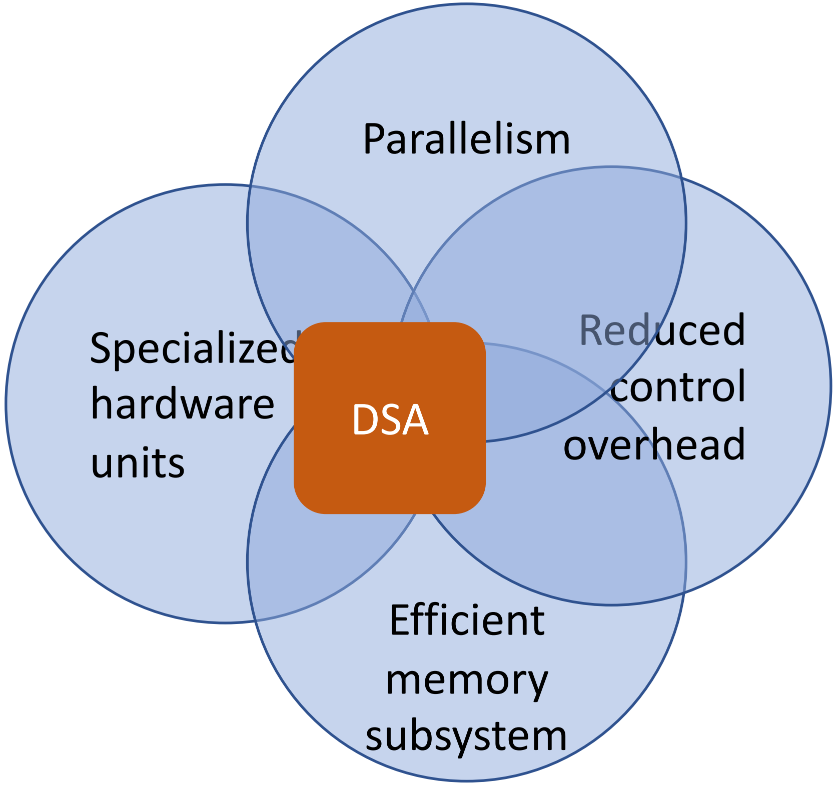
M. Horowitz, "Computing's energy problem (and what we can do about it)," ISSCC 2014

Domain specific accelerators

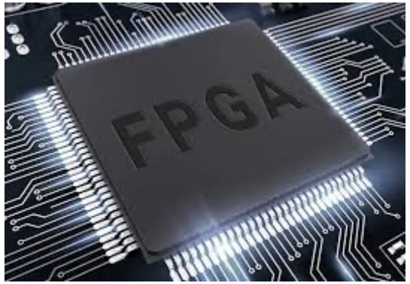


Domain specific accelerators

- Can be for any domain
- Typical/common domains are:
  - Machine Learning
  - Graphics processing
  - Simulation
  - Bioinformatics
  - Image Processing
  - Etc.



The landscape of conventional computing technologies

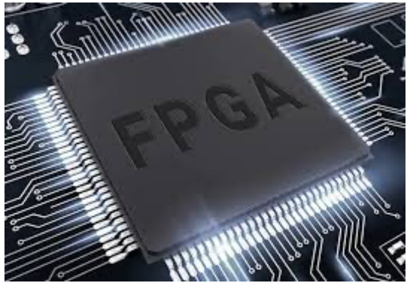


CPU GPU FPGA ASIC

The landscape of conventional computing technologies

CPU

GPU



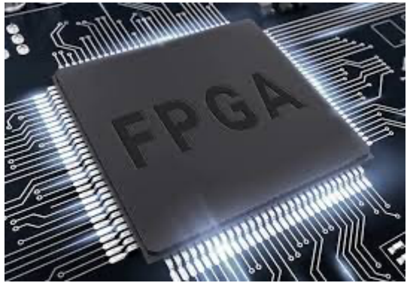
FPGA

ASIC

Efficiency and cost per unit

The landscape of conventional computing technologies

Flexibility and ease-of-use



CPU GPU FPGA ASIC

Efficiency and cost per unit

The landscape of conventional computing technologies

- ASICs: Most efficient but high cost and worst reprogrammability

The landscape of conventional computing technologies

- ASICs: Most efficient but high cost and worst reprogrammability
- FPGAs: 10-100x less efficient compared to ASIC but
  - Can be reconfigured
  - Enables custom accelerator design

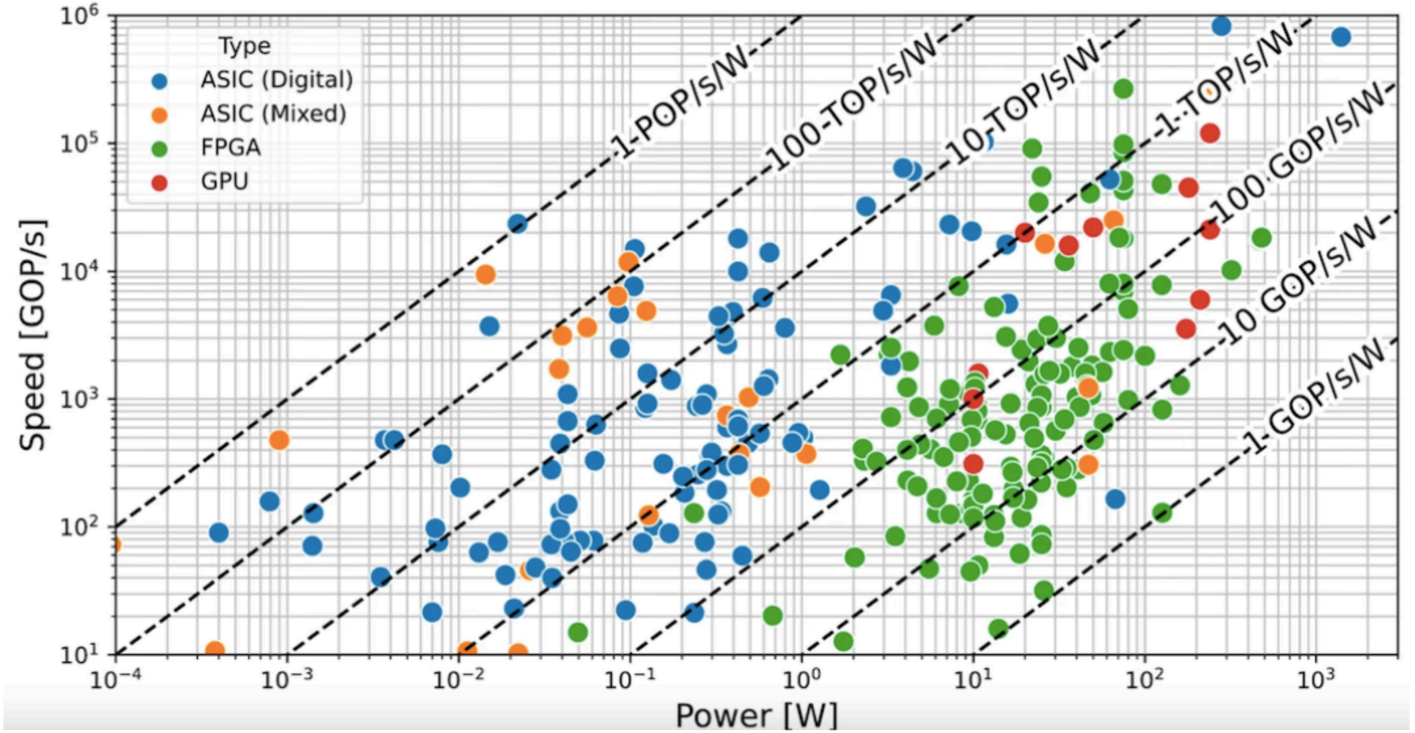
The landscape of conventional computing technologies

- ASICs: Most efficient but high cost and worst reprogrammability

- FPGAs: 10-100x less efficient compared to ASIC but
  - Can be reconfigured
  - Enables custom accelerator design

- GPUs: General-purpose accelerators
  - Can be 10-100x less efficient compared to FPGAs
  - BUT for specific applications, the performance can be as good as ASICs
  - Still follows the Von-Neumann model of computation

Performance/efficiency comparison



Aguirre et al, Nature Communications, 2024

To continue…

Thank you!

asif.ali@uetpeshawar.edu.pk