TRANSFORMERS: AGE OF PARALLEL MACHINES

(a biased introduction to computer architecture for supercomputing)

Ana Lucia Varbanescu, University of Twente, NL

a.l.varbanescu@utwente.nl



Agenda (ambitious)

Part 1 : The anatomy of supercomputers

Part 2 : What's in a name node?

- Part 3 : Diversity in parallelism
- Part 4 : One more word about performance
- Part 5 : Summary and beyond
 - Famous last words …



"Larry, do you remember where we buried our hidden agenda?"

PART 3: PARALLELISM DIVERSITY

Different parallelism models from hardware to software

First taxonomy: Michael Flynn (1966)



Before 2005: technology push

Moore's Law

 Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.



"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year ... Certainly over the short term this rate can be expected to continue, if not to increase...." Electronics Magazine 1965

Until eathy paradelism?

More transistors = more performance

Thus, every 18 months, we had better and faster

processors.

- Higher clock-speed
- Higher perf/cycle
- Same power



Wait ... why do I care?

- More transistors = ... ?
- = more functionality
 - Think more functional units, more complex units, etc....
- Higher perf/clock (aka, higher ILP) = ... ?
- = more operations per cycle
 - Faster overall applications (when they have different operations...)
- Higher clock frequency = ...?
- = more operations per time unit
 - Faster instructions => faster overall application
- Higher power = ... ?
- = global warming ...
 - Ideally, we want power consumption to be low

Until early 2000s ...

Parallelism = interesting and "quirky", but not main-stream

- Pro: Better performance than frequency scaling would provide.
- Con: Parallelizing code was not always worth the effort
 - Do nothing: the performance will double ~ every 18 months

Around 2005: "hitting the walls"



Single core performance scaling

- The rate of single core performance scaling has significantly decreased (essentially, to 0)
 - Frequency scaling limited by power
 - ILP scaling tapped out
 - Design complexity posing serious limitations
- No more free lunch for software developers!
 - No more dramatic increase of software performance for free.

So what?



Traditionally ... single core CPUs

- More transistors = more functionality
- Improved technology = faster clocks = more speed
- Every 18 months => better and faster processors.

Not anymore!

processors ...



New ways to use transistors

Improve PERFORMANCE by using parallelism on-chip: multi-core (CPUs) and many-core processors (GPUs).



The shift to multi-core



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2017 by K. Rupp

Multi-core CPUs



CPU levels of parallelism

- Instruction-level parallelism (e.g., superscalar processors) (fine)
 - Multiple operations of different kinds per cycle
 - Implemented/supported by the instruction scheduler
 - typically in hardware
- **SIMD** parallelism = data parallelism (fine)
 - Multiple operations *of the same kind* per cycle
 - Run same instruction on vector data
 - Sensitive to divergence
 - Implemented by programmer OR compiler
- Multi-Core parallelism ~ task/data parallelism (coarse)
 - 10s of powerful cores
 - Hardware hyperthreading (2x)
 - Local caches
 - Symmetrical or asymmetrical threading model
 - Implemented by programmer

(1) ILP (Instruction level parallelism)

Multiple instructions issued & executed in the same cycle



*Diagrams adapted from CMU's course "Parallel Computer Architecture and Programming" – http://15418.courses.cs.cmu.edu/spring2016/lectures

Implementing ILP

- Super-scalar processors
 - "dynamic scheduling": instruction reordering and scheduling happens in hardware
 - More complex hardware
 - More area, more power ...
 - Adopted in most high-end CPUs today

VLIW processors

- "static scheduling": instruction reordering and scheduling is done by the compiler
 - Simpler hardware
 - Less area, less power
- Adopted in most GPUs and embedded CPUs

(2) SIMD (single instruction, multiple data)

Same instruction executed on multiple data items



*Diagrams adapted from CMU's course "Parallel Computer Architecture and Programming" –

SIMD programmer intervention

- Auto-vectorization
 - Typically enabled with "-O" compiler flags
- Compiler directives
 - Specifically add directives in the code to force persuade the compiler to vectorize code

C or C++ intrinsics

- Wrappers around ASM instructions
 - Declare vector variables
 - Name instruction
 - Work on variables, not registers
- Assembly instructions
 - Can write assembly to target SIMD

(3) Multi-core parallelism

• Two (or more cores) to execute different streams of instructions.



*Diagrams adapted from CMU's course "Parallel Computer Architecture and Programming" – http://15418.courses.cs.cmu.edu/spring2016/lectures

Multi-core programmer intervention

- Must define concurrent tasks to be executed in parallel
 - Typically called (software) threads
- Threads are executed per core
 - Under the OS scheduling
 - Some control can be exercised with additional programmer intervention



What about the memory?

Three levels of cache: L1 (separate I\$ and D\$, per-core), L2 (per-core), L3 (=LLC, shared)



Putting it all together

A modern CPU has a mix of all these features...



Many-core GPUs

Generic GPU



... or, using our CPU "symbols"

- Instructions operate on 32 pieces of data at a time (called "warps").
 - Warp = thread issuing 32-wide vector instructions
- Up to 48 warps are simultaneously interleaved
- Over 1500 elements can be processed concurrently by a core

• Full board: 15 cores (SMs)!

NVIDIA GTX 480 core

Fetch/ Decode Decode<
Execution contexts (128 KB)
"Shared" memory (16+48 KB)



 = SIMD function unit, control shared across 16 units (1 MUL-ADD per clock)

*Diagrams adapted from CMU's course "Parallel Computer Architecture and Programming" – http://15418.courses.cs.cmu.edu/spring2016/lectures

Inside an NVIDIA GPU architecture



Inside an NVIDIA GPU architecture



Inside a Streaming Multiprocessor

- Different types of cores
 - CUDA Cores (INT/FP32)
 - LD/ST
 - Special function units
- Register file
- Warp scheduler
- Data caches
- Instruction buffers/caches
- Texture units

	SMX															
							Poly	Morph	Engine	2.0						
			Vertex I	Fetch				Tesse	llator			V	liewpor	Transf	orm	
		Attribute Setup Stream Output														
	Instruction Cache															
cor	۷	Varp So	chedule	er	Warp Scheduler			Warp Scheduler			r	Warp Scheduler				
50I	Dispat	ch Unit	Dispat	ch Unit	Dispat	tch Unit	Dispat	ch Unit	Dispate	:h Unit	Dispate	:h Unit	Dispat	ch Unit	Dispate	sh Unit
	Register File (65,536 x 32-bit)															
	* * * * * * * * * * * * *								+ + + + +			T				
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	I D/ST	SELL	Core	Core	Core	Core	Core	Core		SELL
	Core	Core	Core	Core	Core	Core		SELL	Core	Core	Core	Core	Core	Core		SELL
	Core	Coro	Coro	Coro	Coro	Coro	I D/ST	SELL	Core	Coro	Coro	Coro	Core	Coro	LD/ST	9511
	Core	Core	Core	Core	Core	Core	LD/ST	oru	Core	Core	Core	Core	Core	Core	LD/ST	oru
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
	Core	Core	Core	Core	Core	Core	LD/ST	SFU	Core	Core	Core	Core	Core	Core	LD/ST	SFU
								Texture	Cache							
						64	KB Sha	ared Me	mory /	L1 Cad	he					
								Jniform	ı Cache)						
	Te	€X	Te	ex	Т	ex	Т	ex	Те	x	Te	эx	Т	₽X	Te	x
	Те	ex	Те	ex	Т	ex	Т	ex	Te	x	Te	×	Т	ex	Te	x
Maxwell		*****	*****	*****	*****	*****	lote	rconne	ct Netv	vork	*****	*****				

More features ...

- Different types of cores
 - Adding: DP Units (Pascal)
 - Adding: Tensor units (Volta)

SM							Instructi	on Cache								<pascal< th=""></pascal<>
	_	1	nstructi	on Buffe	ır	_					nstructi	on Buffe	H.	_	_	
			Warp Se	cheduler						Warp Scheduler						
	Dispat	ch Unit			Dispat	tch Unit			Dispat	ch Unit			Dispa	tch Unit		Volta
	Register File (32,768 x 32-bit)					Register File (32,768 x 32-bit)										
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	DP Unit	Core	Core	DP Unit	LDIST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	
				_			Texture	L1 Cach	•							
	Te	x			1	ex			т	ex				Tex		
						6	4KB Sha	red Memo	iry :							

						SM								
							L1 Instruc	tion Cache						
			L0 In	struct	ion C	ache			L0 Instruction Cache					
		Warp	Sch	eduler	(32 t	hread/clk)		Warp Scheduler (32 thread/clk)						
		Unit ((32 th	read/clk)		Dispatch Unit (32 thread/clk)								
		Regis	ster F	File (1	6,384	4 x 32-bit)			Regi	ster I	File (16,384	4 x 32-bit)		
	FP64	INT	INT	FP32	FP32			FP64	INT	INT	FP32 FP32			
	FP64	INT	INT	FP32	FP32			FP64	INT	INT	FP32 FP32			
	FP64	INT	INT	FP32	FP32			FP64	INT	INT	FP32 FP32			
	FP64	INT	INT	FP32	FP32	TENSOR	TENSOR	FP64	INT	INT	FP32 FP32	TENSOR	TENSOR	
	FP64	INT	INT	FP32	FP32	CORE	CORE	FP64	INT	INT	FP32 FP32	CORE	CORE	
	FP64	INT	INT	FP32	FP32			FP64	INT	INT	FP32 FP32			
	FP64	INT	INT	FP32	FP32			FP64	INT	INT	FP32 FP32			
	FP64	INT	INT	FP32	FP32			FP64	INT	INT	FP32 FP32			
ascal	LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ LD/ ST ST	SFU	LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ LD/ ST ST	LD/ LD/ ST ST	SFU	
			L0 In	struct	ion C	ache				L0 In	struction C	ache		
	Warp Scheduler (32 thread/clk)							Warp Scheduler (32 thread/clk)						
Volta>		Disp	batch	Unit ((32 th	read/clk)		Dispatch Unit (32 thread/clk)						
		Regis	ster F	File (1	6,384	4 x 32-bit)		Register File (16,384 x 32-bit)						
										0101		+ X 32-bit)		
	FP64	INT	INT	FP32	FP32			FP64	INT	INT	FP32 FP32			
	FP64 FP64	INT I		FP32 FP32	FP32 FP32			FP64 FP64	INT INT	INT INT	FP32 FP32 FP32 FP32			
	FP64 FP64 FP64	INT I INT I	INT INT INT	FP32 FP32 FP32	FP32 FP32 FP32			FP64 FP64 FP64	INT INT INT	INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32			
	FP64 FP64 FP64 FP64	INT I INT I INT I	INT INT INT	FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32	TENSOR	TENSOR	FP64 FP64 FP64 FP64	INT INT INT INT	INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TENSOR	TENSOR	
	FP64 FP64 FP64 FP64 FP64	INT I INT I INT I INT I	INT INT INT INT	FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32	TENSOR CORE	TENSOR	FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT	INT INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TENSOR	TENSOR	
	FP64 FP64 FP64 FP64 FP64 FP64	INT I INT I INT I INT I INT I	INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32	TENSOR CORE	TENSOR CORE	FP64 FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT INT	INT INT INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TENSOR	TENSOR	
	FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT I INT I INT I INT I INT I INT I	INT INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32	TENSOR CORE	TENSOR	FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT INT INT	INT INT INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TENSOR	TENSOR	
	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT I INT I INT I INT I INT I INT I	INT INT INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32 FP32	TENSOR	TENSOR	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT INT INT	INT INT INT INT INT INT INT	FP32 FP32	TENSOR	TENSOR	
	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT	INT INT INT INT INT INT LD/ ST	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TENSOR CORE	TENSOR CORE	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT	INT INT INT INT INT INT INT INT	FP32 FP32	TENSOR CORE	TENSOR CORE	
	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT	INT INT INT INT INT INT LD/ ST	FP32 FP32 FP32 FP32 FP32 FP32 FP32 LD/ ST	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TENSOR CORE	TENSOR CORE SFU	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT INT INT LD/ ST	INT INT INT INT INT INT INT INT LD/ ST	FP32 FP32 FP33 FP32 FP34 FP35 FP35 FP35	TENSOR CORE	TENSOR CORE	

GPU Integration into the host system

- Typically based on a PCI Express bus
- Transfer speed (effectively, CPU-to-GPU):
 16 GT/s per lane x 16 lanes
- Can be NVLink (~10x faster) for specialized motherboards

NVIDIA GPUs (8+ years)

	Fermi	Kepler	Maxwell	Pascal	Volta
GPU	GTX480	GK180	GM200	GP100	GV100
Compute capability (CC)	2.x	3.5	5.2	6.0	7.0
SMs	16	15	24	56	80
TPCs	16	15	24	28	40
FP32 Cores / SM	32	192	128	64	64
FP64 "Cores" / SM	4	64	4	32	32
Clock[MHz]	700	875	1114	1480	1530
Peak FP32 [TFLOPs]	1.35	5.04	6.8	10.6	15.7
Peak FP64 [TFLOPs]	0.168	1.68	.21	5.3	7.8

Other players on the market

• AMD (former ATI)

- Much better performance
- Programmed using OpenCL (standard!)
- Poorer software drivers and infrastructure (so far)
- A lot less libraries and tools
- Much smaller community effort
- arm (formerly ARM \odot)
 - Low-power devices (mobile platforms mostly)
 - Programmed using OpenCL
 - Lower performance than ATI and Intel, by choice
- Intel
 - To support own CPUs with integrated graphics
 - Programmed using OpenCL

orm Intel[®]

GRAPHICS

All GPUs ...

- Have a similar architecture
 - Massively parallel
 - Simple cores
 - Complex memory system
- Are programmed in a similar way
 - Fine-grain (SIMD/SIMT) parallelism
- Programming models ?
 - OpenCL is the de-facto standard for GPU programming
 - Lots of efforts for C++
 - Many other libraries and models on top of CUDA / OpenCL

GPU Levels or Parallelism

• Data parallelism (fine-grain)

- Write 1 thread, instantiate a lot of them
- SIMT (Single Instruction Multiple Thread) execution
 - Many threads execute concurrently
 - Same instruction
 - Different data elements
 - HW automatically handles divergence
 - Not same as SIMD because of multiple register sets, addresses, and flow paths*
- Hardware multithreading
 - HW resource allocation & thread scheduling
 - Excess of threads to hide latency
 - Context switching is (basically) free
- Task parallelism is "emulated" (coarse-grain)
 - Hardware mechanisms exist
 - Specific programming constructs to execute multiple tasks.

Heterogeneous computing

• CPU is always present ...

GPUs vs CPUs

Why so different?

- Different goals produce different designs!
 - CPU must be good at everything
 - GPUs focus on massive parallelism
 - · Less flexible, more specialized
- CPU: minimize latency experienced by 1 thread
 - big on-chip caches
 - sophisticated control logic
- GPU: maximize throughput of all threads
 - # threads in flight limited by resources => lots of resources (registers, etc.)
 - multithreading can hide latency => no big caches
 - share control logic across many threads

CPU vs. GPU

CPU

Low latency, high flexibility. Excellent for irregular codes with limited parallelism.

CPU vs GPU

CPU vs. GPU memory hierarchies

CPU vs. GPU: the movie

- The Mythbusters
 - Jamie Hyneman & Adam Savage
 - Discovery Channel
- Appearance at NVIDIA's NVISION 2008:

https://www.youtube.com/watch?v=-P28LKWTzrl

PART 3: PERFORMANCE

Performance "metrics"

- Clock frequency [GHz] = absolute hardware speed
 - Memories, CPUs, interconnects
- Operational speed [GFLOPs]
 - Operations per second, **single/double/...** precision
- Memory bandwidth [GB/s]
 - Memory operations per second
 - Differs a lot between different memories on chip
- Derived metrics
 - FLOP/Byte, FLOP/Watt

Name	FLOPS
yottaFLOPS	10 ²⁴
zettaFLOPS	10 ²¹
exaFLOPS	10 ¹⁸
petaFLOPS	10 ¹⁵
teraFLOPS	10 ¹²
gigaFLOPS	10 ⁹
megaFLOPS	10 ⁶
kiloFLOPS	10 ³

Theoretical peak performance

Throughput[GFLOP/s] = chips * cores * vectorWidth * FLOPs/cycle * clockFrequency

Bandwidth[GB/s] = memory bus frequency * bits per cycle *
bus width

	Cores	Threads/ALUs	Throughput	Bandwidth
Intel Core i7	4	16	85	25.6
AMD Barcelona	4	8	37	21.4
AMD Istanbul	6	6	62.4	25.6
NVIDIA GTX 580	16	512	1581	192
NVIDIA GTX 680	8	1536	3090	192
AMD HD 6970	384	1536	2703	176
AMD HD 7970	32	2048	3789	264
Intel Xeon Phi 7120	61	240	2417	352

Why should we care?

- Peak performance indicates an absolute bound of the performance that can be achieved on a given machine
 - It is *application independent*
- Such performance is rarely* achievable in practice for real applications.
 - Applications rarely utilize all the machine features.
- The balance of an application must *consistently* match the balance of the machine to get anywhere near the peak...
- ... or else... different bottlenecks!

*Empirical studies show this reads as "almost never" .

https://gitlab.com/astron-misc/benchmark-intrinsics/-/tree/master

Hardware performance

Performance "metrics"

- Clock frequency [GHz] = absolute hardware speed
 - Memories, CPUs, interconnects

Operational speed [GFLOPs]

- Operations per second
- single AND double precision

Memory bandwidth [GB/s]

- Memory operations per second
 - Can differ for read and write operations !
- Differs a lot between different memories on chip
- Power [Watt]
 - The rate of consumption of energy
- Derived metrics
 - FLOP/Byte, FLOP/Watt

1	Name	FLOPS
1	ottaFLOPS	10 ²⁴
2	zettaFLOPS	10 ²¹
	exaFLOPS	10 ¹⁸
1	petaFLOPS	10 ¹⁵
1	teraFLOPS	10 ¹²
9	gigaFLOPS	10 ⁹
	megaFLOPS	10 ⁶
1	kiloFLOPS	10 ³

Theoretical peak performance

Throughput[GFLOP/s] = chips * cores * vectorWidth * FLOPs/cycle * clockFrequency

Bandwidth[GB/s] = memory bus frequency * bits per cycle *
bus width

	Cores	Threads/ALUs	Throughput	Bandwidth
Intel Core i7	4	16	85	25.6
AMD Barcelona	4	8	37	21.4
AMD Istanbul	6	6	62.4	25.6
NVIDIA GTX 580	16	512	1581	192
NVIDIA GTX 680	8	1536	3090	192
AMD HD 6970	384	1536	2703	176
AMD HD 7970	32	2048	3789	264
Intel Xeon Phi 7120	61	240	2417	352

Why should we care?

- Peak performance indicates an absolute bound of the performance that can be achieved on a given machine
 - It is *application independent*
- Such performance is rarely* achievable in practice for real applications.
 - Applications rarely utilize all the machine features.
- The balance of an application must *consistently* match the balance of the machine to get anywhere near the peak...
- ... or else... different bottlenecks!

*Empirical studies show this reads as "almost never".

https://gitlab.com/astron-misc/benchmark-intrinsics/-/tree/master

Measuring hardware performance

- Microbenchmarking*
 - Evaluates hardware features in isolation
 - Goal: find out the true limits of the hardware components
 - Platform-specific results
 - Compared with the theoretical peak, per platform.
- Benchmarking
 - Evaluates the FULL platform
 - Application-specific performance
 - Top500 computation capability
 - Graph500 graph processing capability
 - Green500 energy consumption

Compares platforms

* Henry Wong, Misel-Myrto Papadopoulou, Maryam Sadooghi-Alvandi, Andreas Moshovos. "Demystifying GPU

TOP5(00) JUNE 2024

1 custom-built machine

1 Exascale machine

3 energy-efficient machines: AMD, Intel+NVIDIA, IBM

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	Eagle - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.8	22,786
2	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	Eagle - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.2	29,899
5	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC	2,752,704	379.70	531.51	7,107
	Finland		Gan [.]	$20 - 30^{\circ}$	2/2

Top500 June 2024

Lots of accelerators.

Many many many cores.

High **peak** performance.

Large **gap** from peak to "real" performance.

Let's talk about energy => check out Green500

See more at: https://www.top500.org/

PART 5: SUMMARY AND BEYOND

Where to?

Today's computing machines

- Parallel at different levels
 - Multi- or many-cores
 - Core-level parallelism
 - CPU-accelerator(s) parallelism
 - None-level parallelism
- Different performance and power "profiles" => different energy consumption => different energy efficiency envelops
- Hardware-level performance
 - FLOPs (or INTOPs) for computation
 - GB/s for memory bandwidth
 - FLOPS/Watt for energy efficiency
- Benchmarking machine performance
 - Micro-benchmarking vs. benchmarking
 - Diverse metrics => "performance counters"

Programming and programmability

- Diversity in hardware from nodes to system is challenging for programmers
- A multitude of programming models with different trade-offs between productivity/programmability and performance exist and emerge
- There exists important limitations in suppoting all models on all systems effectively and efficiently
 - ... meaning choices (and therefore criteria for such choices) are needed
- System-level knowledge is difficult to acquire and use as programmer
 - But we should not stop trying.
- Tools for programming, debugging, modelling, analysis, benchmarking exist
 - But they could use further improvements.

Performance/Efficiency will depend on ...

Developers and users

Improve the energy efficiency of their own codes, making use of algorithmic, programming, and hardware tools

Design and implement applications able to adapt to the available system resources

System integrators

Offer the right mix of resources for the application developers and system operators.

Include efficient hardware to enable different application mixes.

System operators

Ensure efficient scheduling of workloads on system resources.

Harvest energy where resources/systems are massively underutilized.

Dolas, Sagar et al. "Making Scientific Research on Dutch National Supercomputer Energy Efficient." *ERCIM News* 131, Oct. 2022 <u>https://ercim-news.ercim.eu/en131/special/making-scientific-research-on-dutch-national-supercomputer-energy-efficient</u>

Looking forward

- Larger and more complex systems
 - With a stronger focus on energy efficiency
- More heterogeneous systems
 - At node, blade, and rack level
 - At memory-system level
- More programming models, runtimes, and schedulers
 - Seamless integration possible, but unlikely
- Gradual performance shift from pure HPC towards efficiency and sustainability
 - Metrics and tools needed

From: <u>https://image.shutterstock.com/image-vector/man-between-arrows-choice-past-260nw-1390413521.jpg</u> Adapted from: Jim Naylor at <u>https://www.cartoonstock.com/cartoon?searchID=CS273198</u>

