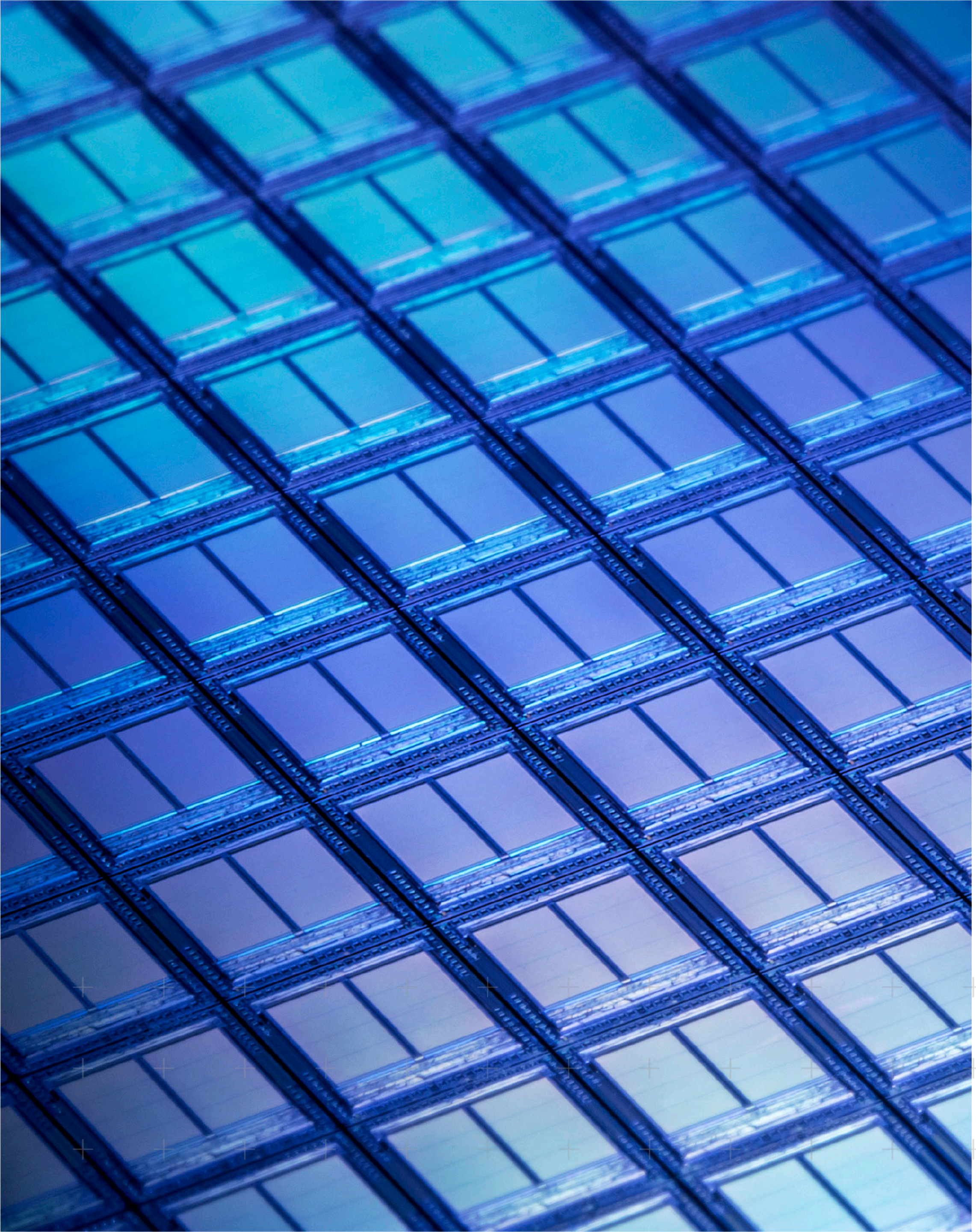


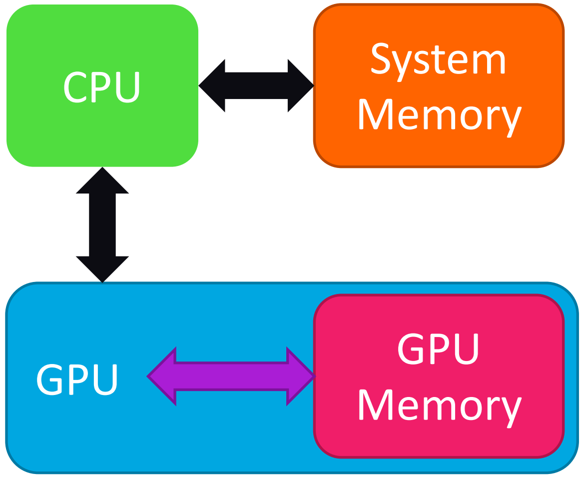
Running AI Inference On CPUs

Pradeep Ramachandran
Director and Head of Research,
Advanced Computing Labs,
KLA, India

1st October, 2021

Why is Running AI Inference on CPUs Important?

- Because it is there!
  - Every system typically has a powerful CPU to host system & legacy applications
- Accelerators may not be available in the system used for inference
  - Accelerators (even GPUs) are still the “new kid in the block”; not all systems have them
  - System to be used in deployment may be unknown in certain use-cases; edge devices, cloud, etc.
- Data for inference already in CPU-accessible memory; no need for data movement!



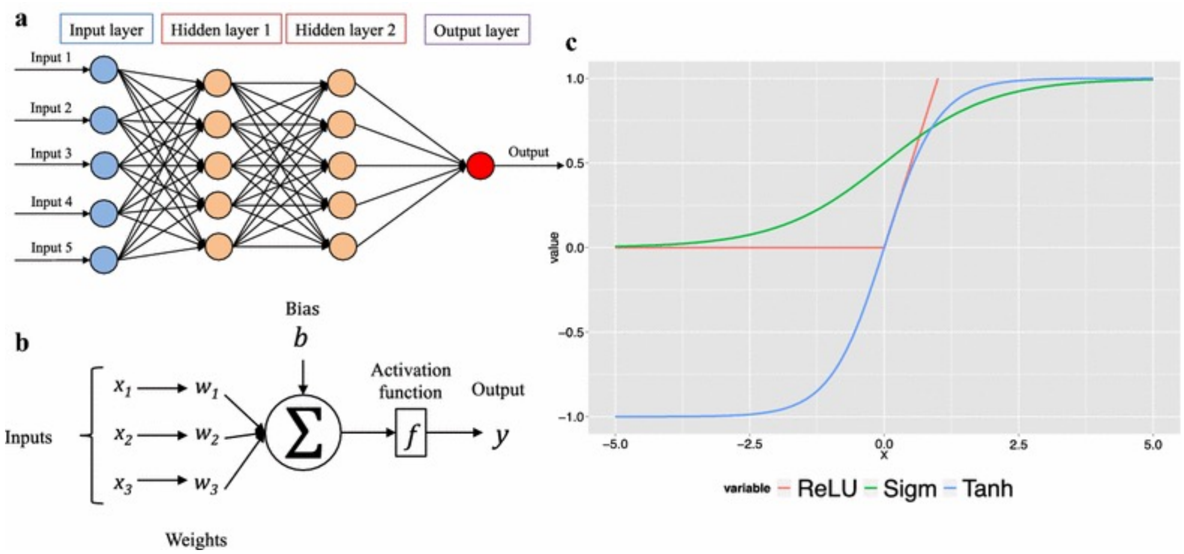
Data Source	Connection	BW (GBps) [1][2]	
		Theoretical	Practical
System Memory	DDR4-3200 (8 ch)	204.8	171.5
GPU memory	HBM2	1555.0	1250.0
CPU ⇔ GPU connect	Gen4 PCIe x16	32.0	26.2
CPU ⇔ GPU connect	NVLink (used by SXM2)	300.0	240.0

Big gap!

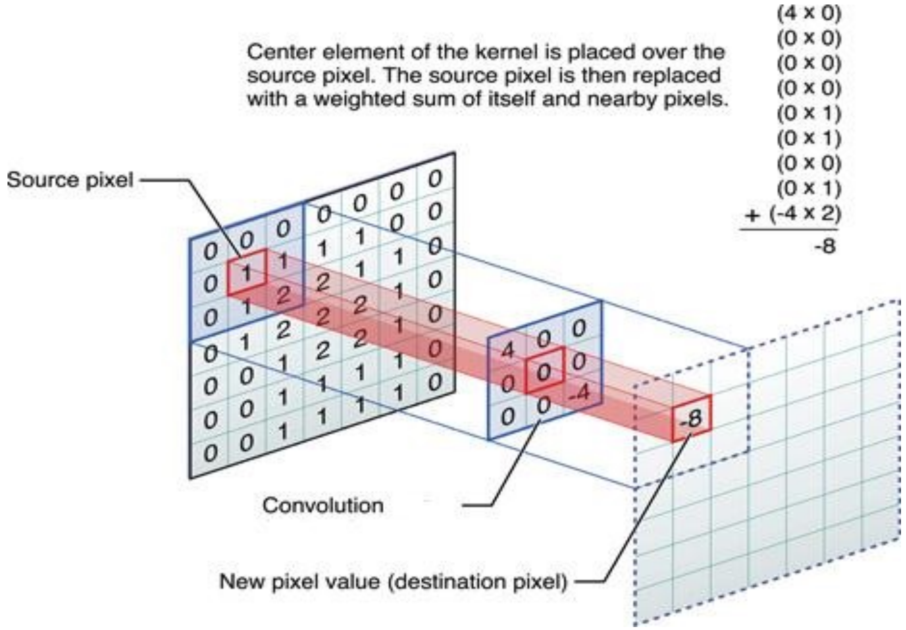
[1] https://www.microway.com/knowledge-center-articles/performance-characteristics-of-common-transports-buses/
[2] https://www.nextplatform.com/micro-site-content/achieving-maximum-compute-throughput-pcie-vs-sxm2/

Compute & Data Patterns of AI Workloads

Typical Operations in Dense Layers [3]



Typical Operation in Convolution Layers [4]



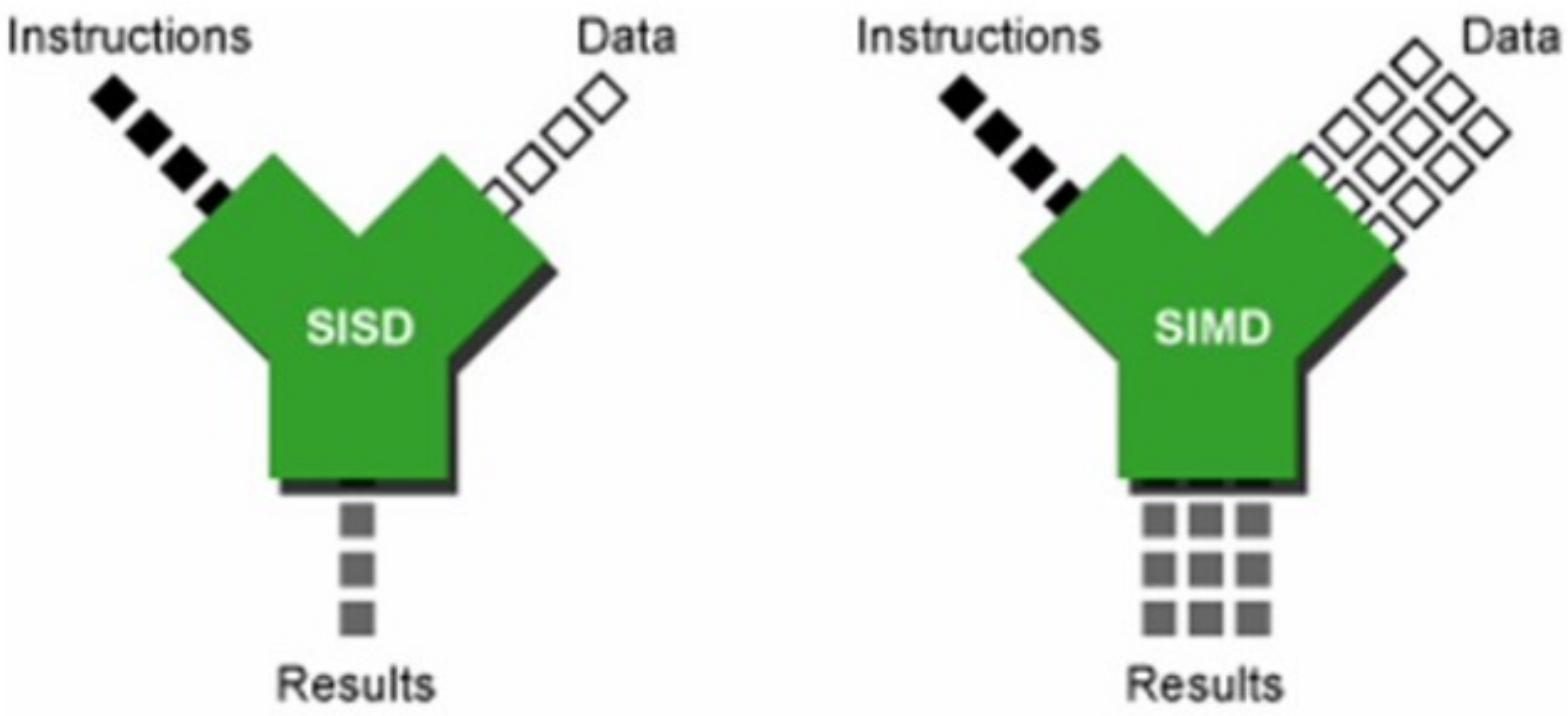
- Observations about compute patterns
  - Multiplication followed-by addition (or reduction) is common
  - Most operations are repeated across several data elements with limited scalar operations in between
- Observations about data patterns
  - Outputs of one layer serve as inputs to the next; but inputs aren't reused
  - Only weights are reused across instances; no other information in shared (during inference)

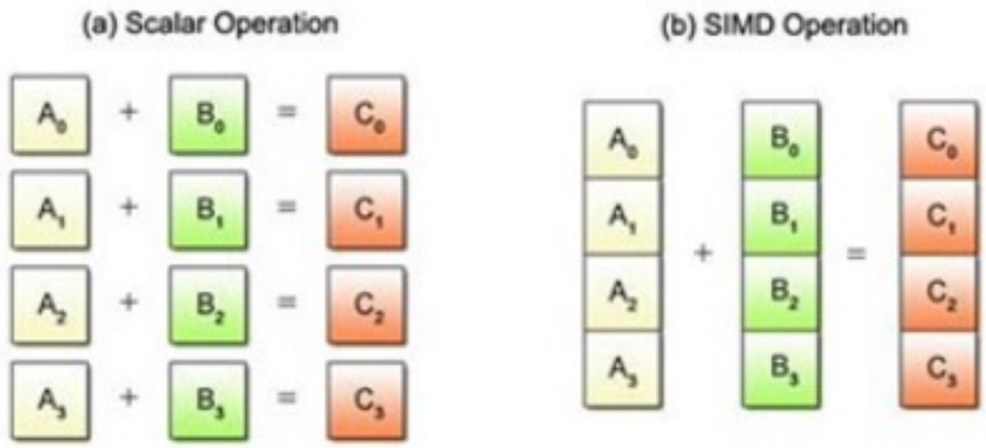
[3] https://jcheminf.biomedcentral.com
[4] https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

CPU Support to Exploit Compute Patterns

Workload Pattern	CPU Support	Benefits
Multiplication + addition is common	ISA natively supports fused multiply-add (FMA)	Reduced cycle count for mult + add; limited front-end bottlenecks
Most operations are repeated across several data elements	Width of Vector units (leveraged by multimedia workloads) increased	Significantly increased throughput for AI loads
Networks use limited scalar operations amidst parallel operations	Thread ganging support	Leverage single-thread performance of CPU cores for improved inference throughput

- Rest of the talk will focus on these three elements
  - Techniques that exploit the workloads' data patterns is beyond the scope of this talk

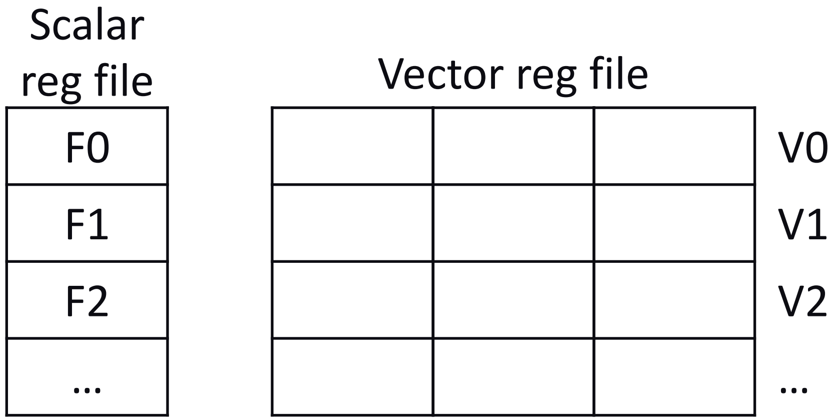




Inference with CPU Vector Units

Vector Architectures – The Basics

- Basic idea:
  - Read sets of data elements into “vector registers”
  - Operate on those registers
  - Disperse the results back into memory
- Registers are controlled by compiler
  - Used to hide memory latency
  - Leverage memory bandwidth



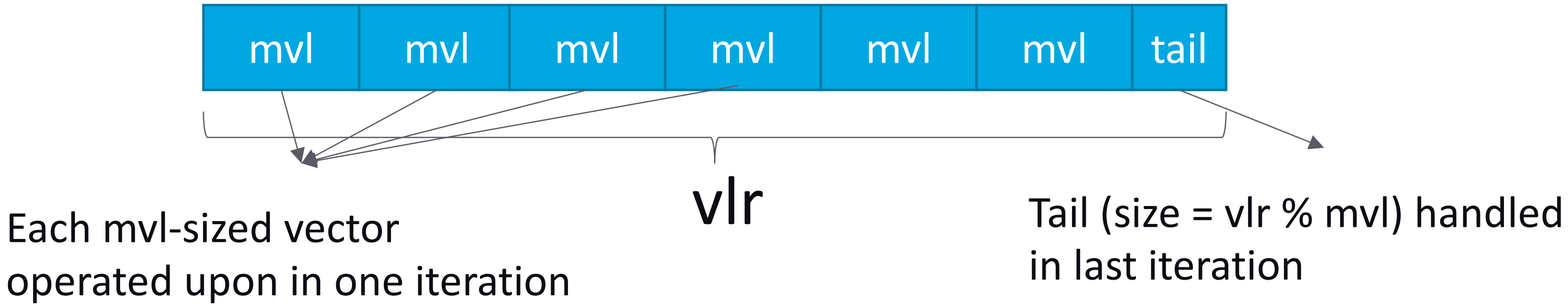
- Example: DAXPY (Ry = a * Rx + Ry) takes 6 instructions with vector; 100s with scalar [5]

```
L.D      F0,a       ; load scalar a
LV       V1,Rx      ; load vector X
MULVS.D  V2,V1,F0   ; vector-scalar multiply
LV       V3,Ry      ; load vector Y
ADDVV    V4,V2,V3   ; vector-vector add
SV       Ry,V4      ; store the result vector
```


[5] J. Hennessey, and D. Patterson, “Computer Architecture, a Quantiative Approach”, Sixth Edition, 2017

Vector Architecture – The Basics, Continued…

- Operating on long- and short-vectors supported via dedicated registers
  - Variable length vectors supported with knowledge of max (mvl), and current vector length (vlr)
    - Use strip-mining to break vector into mvl-sized vectors that can be operated on in parallel



  - Vector mask register supports "disabling" some elements of vector during operation
- Main challenge – real-world apps didn't have as much vector parallelism!
  - Cray-1 implemented a vector-style architecture but moved away from subsequent generations

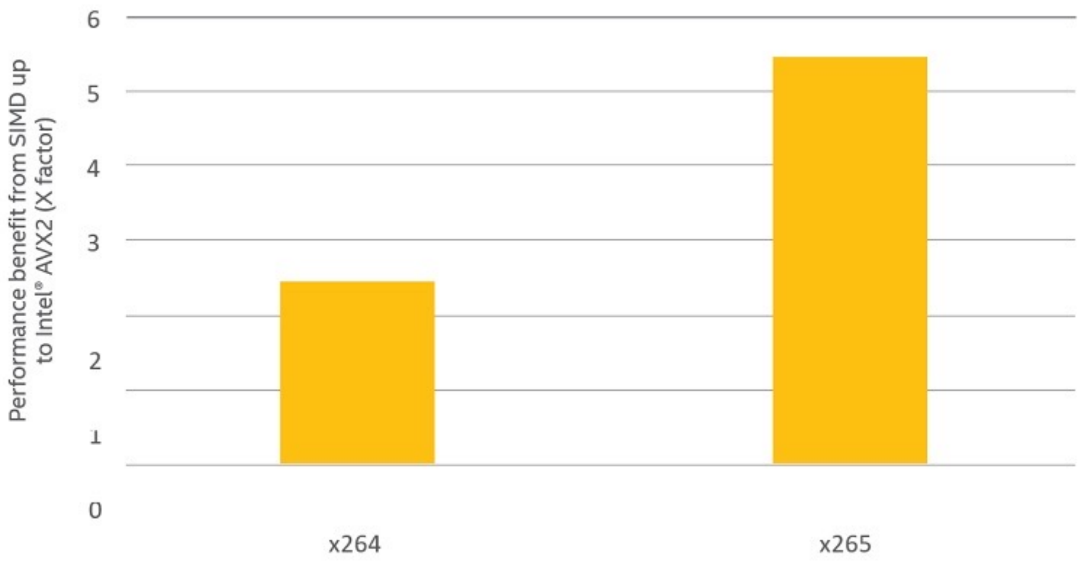
Vector Architecture – The Practical SIMD Implementation

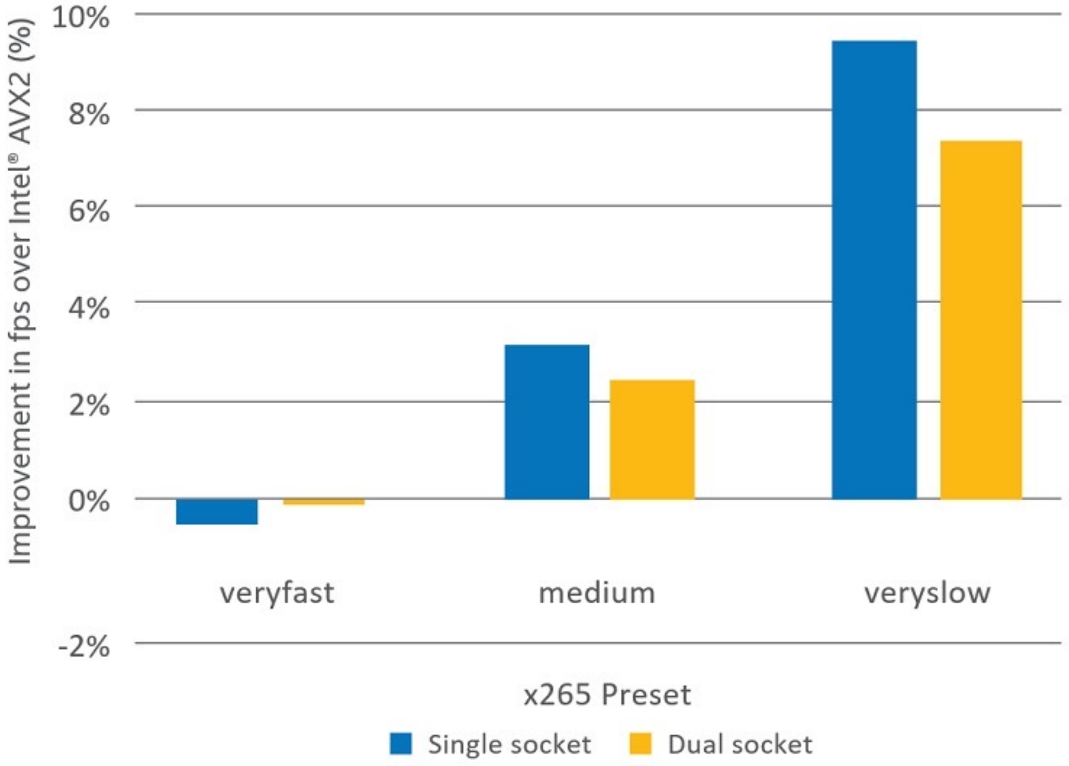
- Two key observations led to practical adaptations of vector architectures
  - #1: Media workloads were the main “data parallel applications” until the last decade
  - #2: Media workloads operated on data that was 8b or 16; 64b was too long!
    - # bits per pixel dictated by accuracy of camera sensors; even today it isn’t common to go to 16
    - Providing vector registers of mvl * 64b wasn’t that useful
- CPUs provide vector units that operate on “configurable” register files
  - Dedicated int and fp vector register file provided; can be configured for various vector lengths
    - Int file = n x 8b, or n/2 x 16b, or n/4 x 32b, or n/8 * 64b regs; fp file = 32b (single) or 64b (double) regs
  - Dedicated instructions in ISA to operate on vectors of different widths
    - E.g., `vaddpd` in AVX adds two double precision fp regs, while `vaddps` adds two single precision fp regs [6]
  - Implementations called SIMD (Single Instruction Multiple Data) vector units
    - Examples – MMX, SSE, AVX, AVX2, AVX512 in x86 processors, Neon in ARM, Altivec in POWER, etc.

[6] x86 Assembly Language Reference Manual - https://docs.oracle.com/cd/E37838_01/html/E61064/gntbd.html#scrolltoc

Leveraging Vector Units: Case Study with Video Encoders

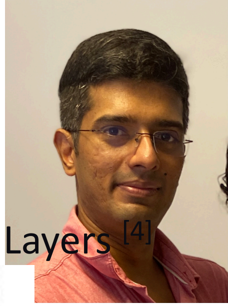
- Video encoders are highly data-parallel & leverage vector units for SIMD parallelism
  - Pixels represented as 8bit or 16bit numbers; integer and fp operations (like SAD, MSE) typical
- Study: How x264, and x265 (open source video codecs) gain from SIMD parallelism [7]



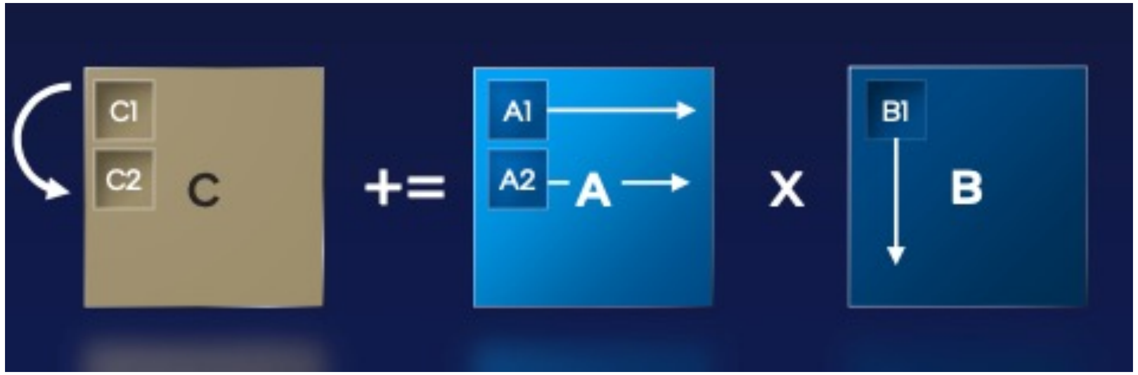


[7] https://software.intel.com/content/www/us/en/develop/articles/accelerating-x265-with-intel-advanced-vector-extensions-512-intel-avx-512.html

Expanded Vector Architecture for AI Inference

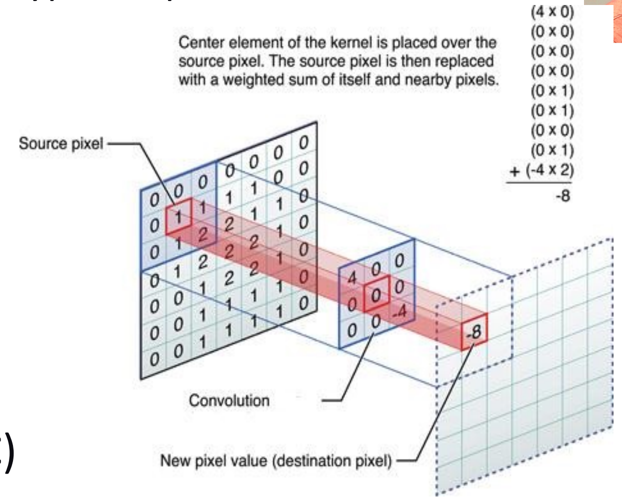


- AI workloads use convolutions that are matrix operations
  - Dense layers tend to be more vector-like, but may also leverage matrix
- Modern CPUs adding matrix-units on top of vector-units
  - Intel released Advanced Matrix Extension (AMX) focused on AI loads [8]
    - Register file is a 2D “tile”; 8 regs * 1Kb per register
    - TMUL instruction operates on tiles performing multiply and accumulate (MAC)



  - Throughput increases from 256 → 2048 int8 ops / cycle / core!

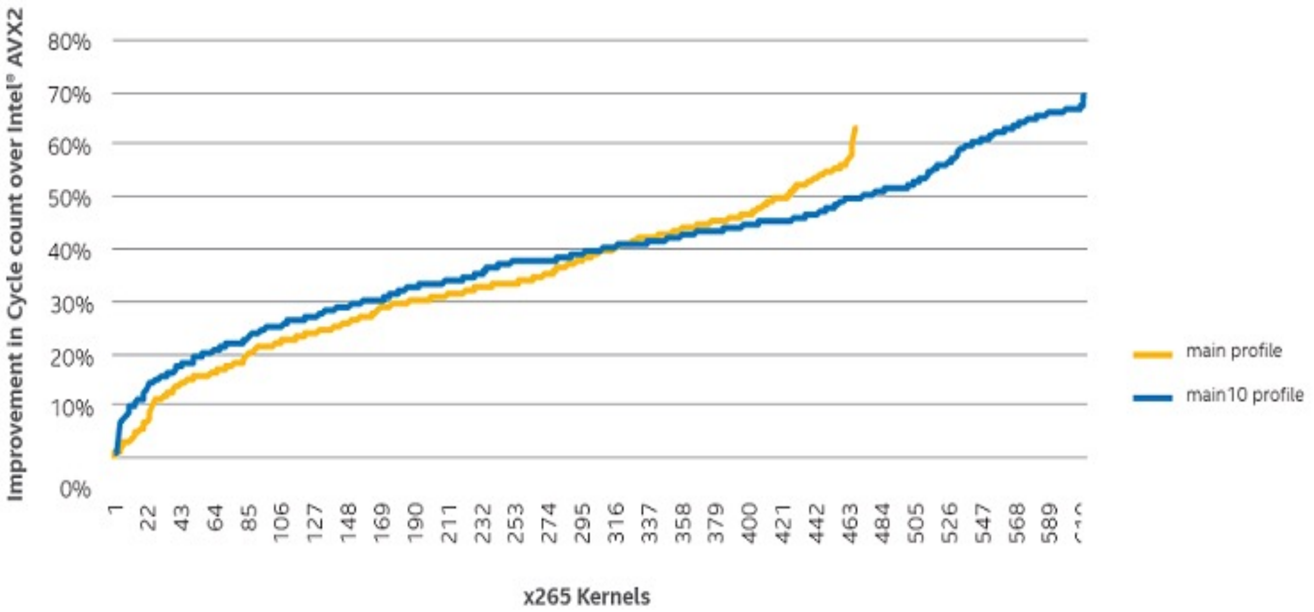
Typical Operation in Convolution Layers [4]



[8] https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf

Vectors: The Devil is In the Detail!

- Vector (and matrix) implementations giveth throughput but…
  - … they taketh clock-speed away! [9]
  - … and automatic generation isn't practical to-date
- Implementation can be long and arduous
  - Every new generation typically requires a rewrite of kernels
  - And enough kernels need rewrite to overcome freq hits [7]



10m Cannon Lake Core i3-8121U	*AnandTech*
2 / 4	Cores / Threads
15 W	Rated TDP
2.2 GHz	Base Frequency
3.2 GHz	Single Core Turbo
3.1 GHz	Dual Core Turbo
2.2 GHz	AVX2 Frequency
1.8 GHz	AVX512 Frequency

[9] https://www.anandtech.com/show/13405/intel-10nm-cannon-lake-and-core-i3-8121u-deep-dive-review/5

Vector Implementations – Recent Trends!

- SIMD implementations a huge success, but has significantly complicated the ISA [10]
  - IA-32 instruction set has grown from 80 instructions in 1978 to ~1400 instructions largely due to SIMD!
    - Ensuring backwards compatibility of older instructions with wider new gen one of the key contributors
  - Complex ISA complicates decode, and make job of user (application writer / compiler) harder
- RISC-V recently proposed going back to a vector-like ISA that uses vlr, and mlr
  - Registers 64b wide; operating them as 32b / 16b / 8b, and non-mlr-conformat vectors handled in HW
  - Significantly simplifies ISA, and book-keeping overhead when running in a loop

ISA	*MIPS-32 MSA*	*IA-32 AVX2*	*RV32V*
Instructions (static)	22	29	13
Instructions per Main Loop	7	6	10
Bookkeeping Instructions	15	23	3
Results per Main Loop	2	4	64
Instructions (dynamic n=1000)	3511	1517	163

[10] https://www.sigarch.org/simd-instructions-considered-harmful/

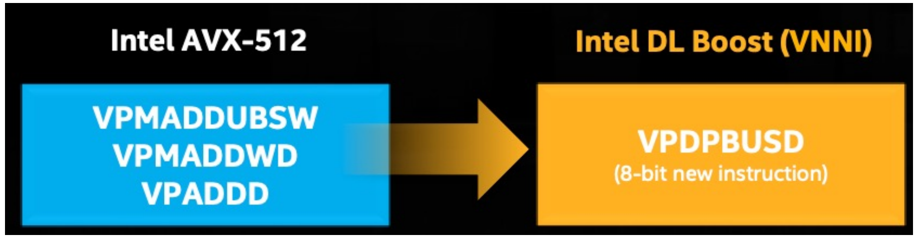
Other CPU Advancements for Inference

Extensions to ISA that Aid Inference

- Most CPUs support Fused Multiply Add (FMA) instructions at low latency
  - Initially introduced for media workloads; extensions to 128b and 256b SIMD instructions [11]

Opcode	Operation	Opcode	Operation
VFMADD	$result = +a \cdot b + c$	VFMADDSUB	$result = a \cdot b + c$ for $i = 1, 3, \ldots$
VFNMADD	$result = -a \cdot b + c$		$result = a \cdot b - c$ for $i = 0, 2, \ldots$
VFMSUB	$result = +a \cdot b - c$	VFMSUBADD	$result = a \cdot b - c$ for $i = 1, 3, \ldots$
VFNMSUB	$result = -a \cdot b - c$		$result = a \cdot b + c$ for $i = 0, 2, \ldots$

- Intel VNNI added specific AI-focused instructions that fuse AVX512 instructions [12]

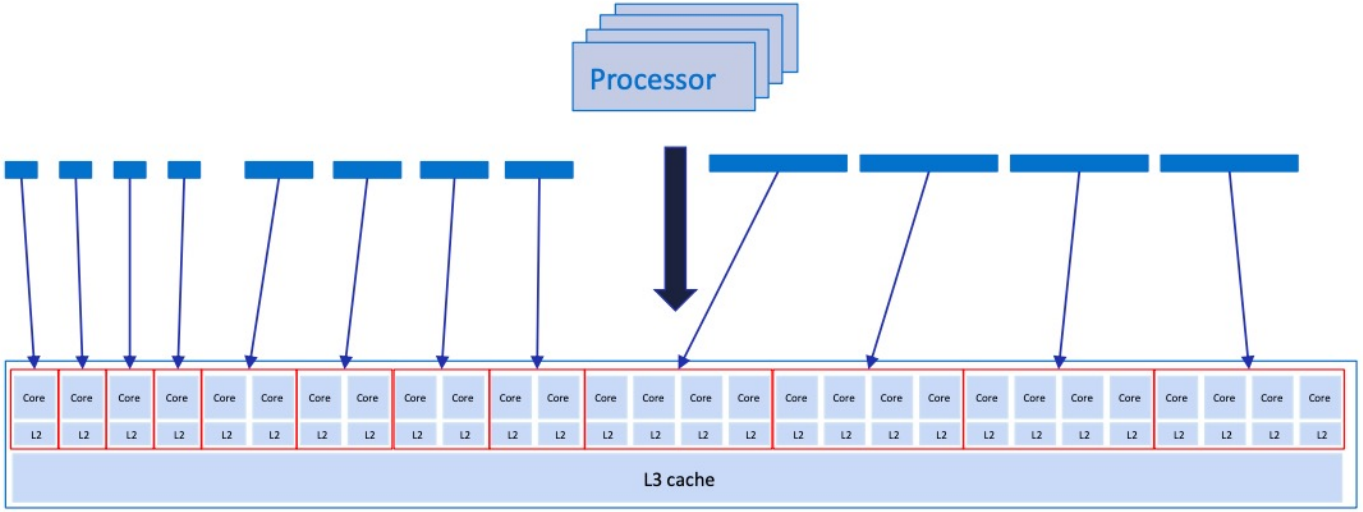


- Leveraged by compilers during back-end code-gen to optimize inference latency
  - Hand-coded intrinsics, or assembly functions can also use these instructions

[11] https://en.wikipedia.org/wiki/FMA_instruction_set
[12] https://www.intel.com/content/dam/www/public/us/en/documents/product-overviews/dl-boost-product-overview.pdf

Running Multiple Inference Sessions on CPU

- CPU cores may be “grouped” to improve utilization for multi-session inference [13]



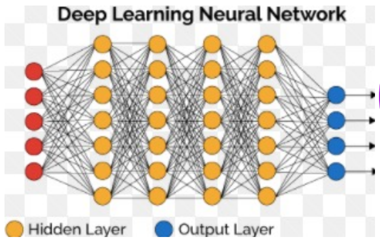
- Multi-session inference offers several advantages
  - Reduced synchronization overhead
  - Improved performance if work is grouped such that activations fit inside L2
- Similar to the idea of Multi-Process Service (MPS) with CUDA for NVIDIA GPUs [14]

[13] Guokai Ma, “Tutorial: Deep Learning Inference Optimizations for CPU ”, Hot Chips 2021
[14] https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

The CPU is Not New, Einstein!

- Techniques used to improve inference performance in accelerators also apply here
- Models that operate with reduced precisions at inference deployed on CPUs
  - CPUs can accelerate BF16 (16-bit block-float), int8, and lower precision inference using vector units
  - Throughput typically doubles with half the precision
    - Int8 throughput = 2X FP16 = 2X FP32 = 2X FP64 (typically)
  - Impact to model accuracy handled with techniques like calibration, centering, etc.
- Fuse kernels to limit moving data between caches and system memory (DRAM)
  - Cache ⬄ DRAM is arguably faster than GPU ⬄ DRAM as there is no PCIe
  - Nevertheless, fusing enables reusing buffers in caches
  - Great candidate for auto-generation; compiler community is very excited!
- And other learnings can also be applied…

Why GPU based accelerated computing is winning …..



Deep Learning Networks

5-15X

GPU

Newer CPU instructions (like AMX) may narrow CPU⇔GPU gap

GPU

Traditional Image Processing

3-6X

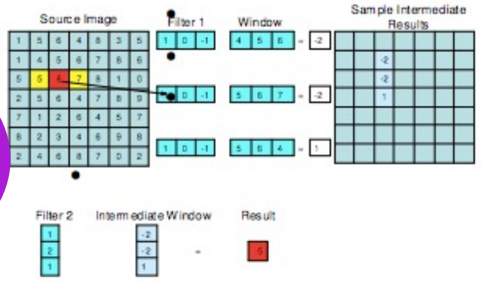
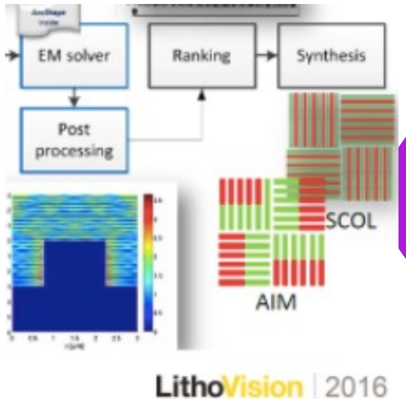


Figure 3. Separable filter convolution.

Traditional CPU Algos

GPU

GPU

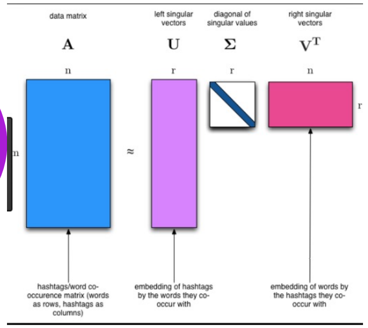


EM Solvers, 3D Tomography

2-4X

PCA, SVD, 2D-Histograms Small Matrix Solvers

2-4X



Two Incumbent Revolutions for AI Inference

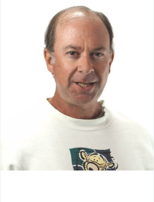
- Bringing CPUs and accelerators closer with better interconnect - the "HW Revolution"
  - CPUs and accelerators (GPUs ++) will continue to co-exist with specific advantages
  - Data movement is becoming THE bottleneck; problem exacerbated with bigger models!
  - Future bright for higher BW CPU⇔GPU links (Gen5 PCIe, CXL, NVLink, etc.), cache-coherent accelerators

- Better auto-generated HPC code – the "SW revolution"
  - Current models leverage hand-tuned kernels to get practical performance
    - Hand-written in (intrinsics for CPU, or CUDA, SYCL for GPUs) expressed as custom-ops in TensorFlow (or similar)
  - Cannot scale as every new HW requires new kernels making it hard to even evaluate them!
  - Auto-generating performant code will enable quick real evaluation making systems more nimble
    - Enables quicker adoption of multiple accelerators; lots of work in this area (MLIR, C++, TVM, XLA)

Conclusion

- CPU can be considered as the first option to use for AI inference
  - Accelerators, while performant, aren't default in HPC systems today
- Many key innovations in the CPU space target faster AI inference
  - Improved vector units that treat matrices as first-order-citizens
  - Extensions to ISA to support AI operations like multiply + add, applying activation function, etc.
- But sustained high-performance AI inference needs two revolutions!
  - HW revolution – Design better interconnect to bring CPU and accelerators closer
  - SW revolution – Auto-generate high-performance kernels; avoid hand-tuned low-level kernels

Recap Of The Workshop

Session	Speaker	Comments
Modern AI in manufacturing	Kris Bhaskar 	Overview of the problem space
Challenges in Adopting ML in manufacturing	Jacob George 	Use-cases, algorithms, and challenges
AI Models in the Fab	Steve Esbenshade 	Examples of how our images and data flows in our tools
Minimizing copy overhead while sharing GPUs on a single box	Mark Ruolo 	Data movement challenges, GPU memory / compute bandwidth imbalance, etc.
AI inference on CPUs	Pradeep 	Discusses leveraging vector arch, and other recent developments in CPU to aid inference

Conclusion (For the Workshop)

- Semiconductor manufacturing revolutionized by AI & HPC technologies
  - Semiconductors are now a critical part of the global economy
  - Inspection and metrology require cutting-edge AI & HPC technologies to keep Moore's law alive!

- KLA is leveraging several solutions in this space in its products
  - eSL10™ is the industry-first manufacturing tool that leverages integrated Artificial Intelligence with SMARTs™ deep learning algorithms
- Exciting time to be an engineer at the intersection of AI, HPC, and manufacturing!
  - Exciting time to be an engineer at KLA

Thank You

- We’re actively looking for collaboration with academic partners in this space
  - Write to me at Pradeep.Ramachandran@klatencor.com
- We’re hiring interns, and full-time engineers in this space (AI, HPC, Software)
  - KLA India –https://www.kla.com/careers/locations/india
  - KLA world-wide - https://www.kla.com/careers