Graph Algorithms on Emerging Tile-Centric Accelerators

Johannes Langguth

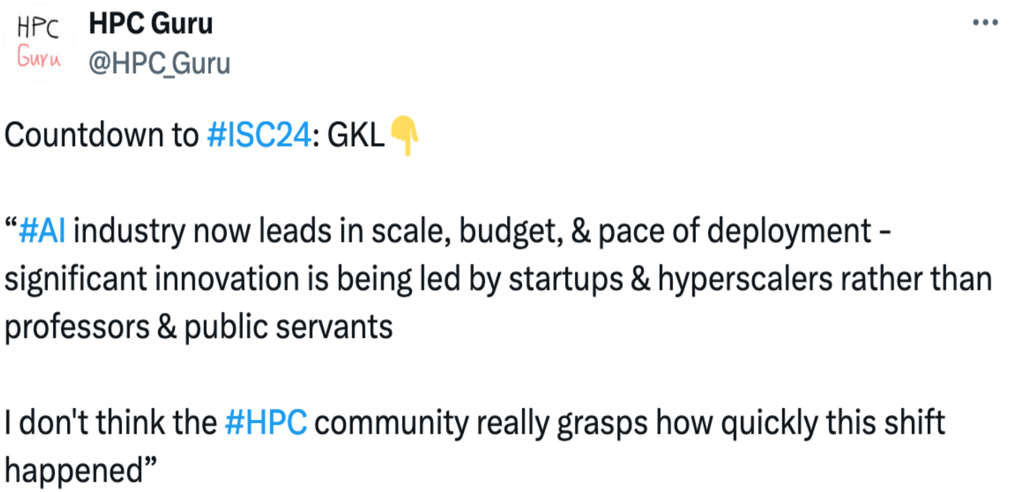
Simula Research Laboratory, Norway

University of Bergen, Norway

Dresden, August 22, 2024

Current Trend in Computer Architecture: AI Hardware

The golden age of computer architecture is here

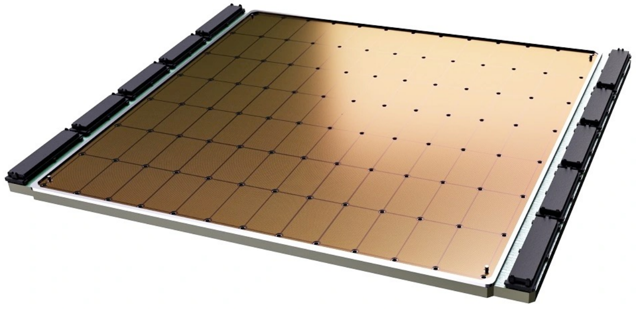


HPC Guru
@HPC_Guru

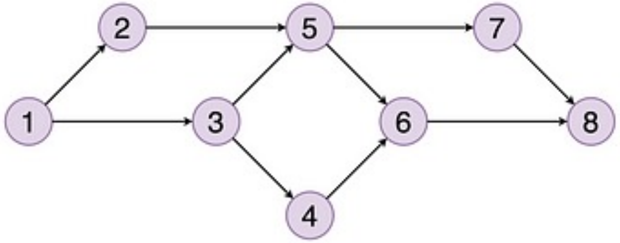
Countdown to #ISC24: GKL👇

“#AI industry now leads in scale, budget, & pace of deployment - significant innovation is being led by startups & hyperscalers rather than professors & public servants

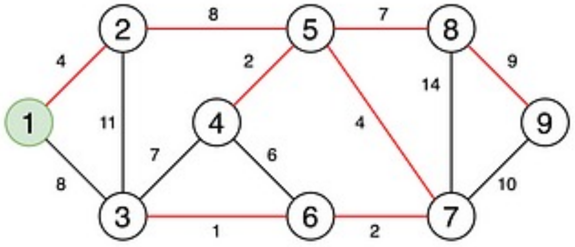
I don't think the #HPC community really grasps how quickly this shift happened”

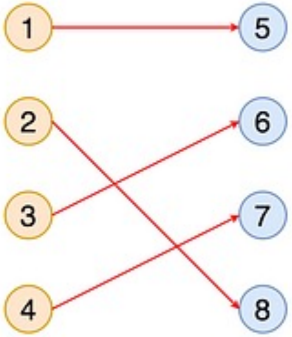


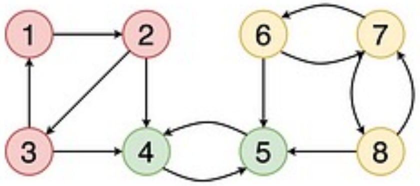
But We Like…

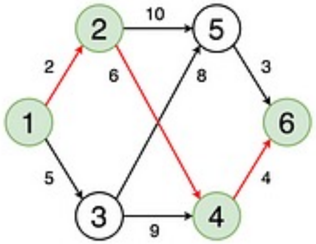


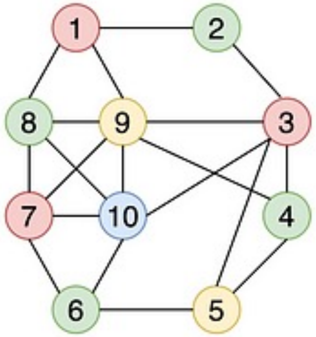
Graph Algorithms







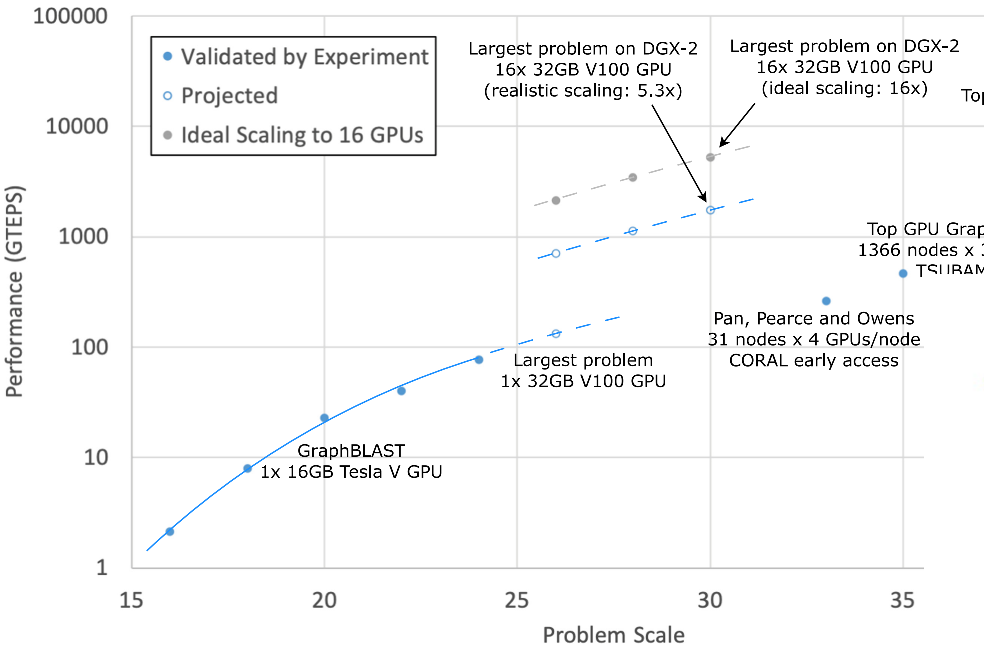




So why should I care?

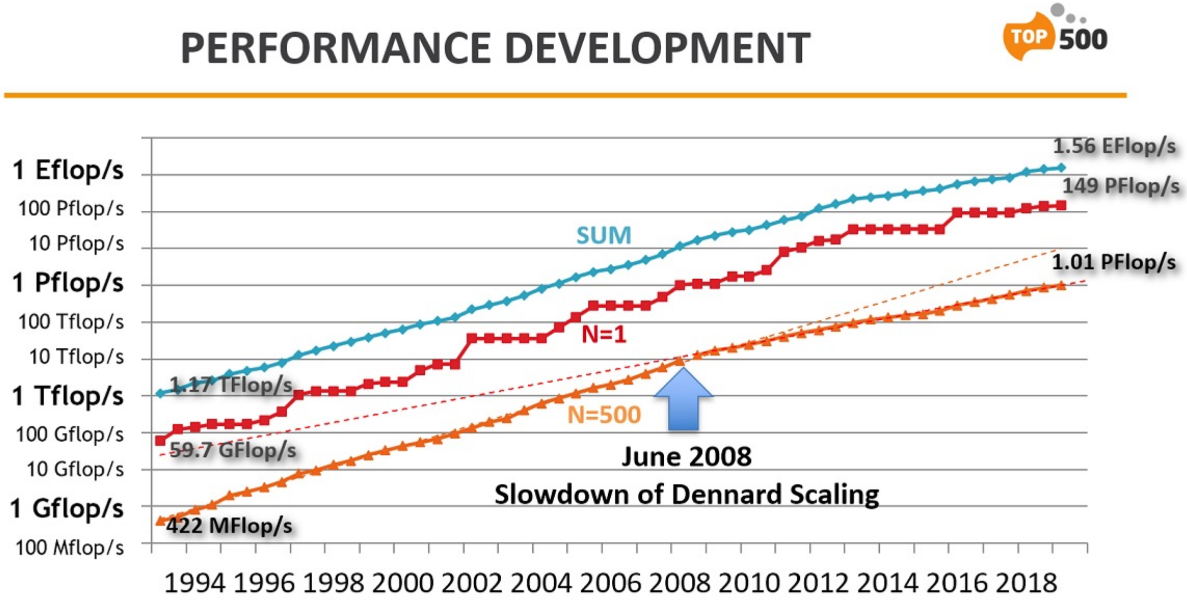
(J.L. in 2017)

Graph Algorithms Are Hard to Scale (and not very efficient)



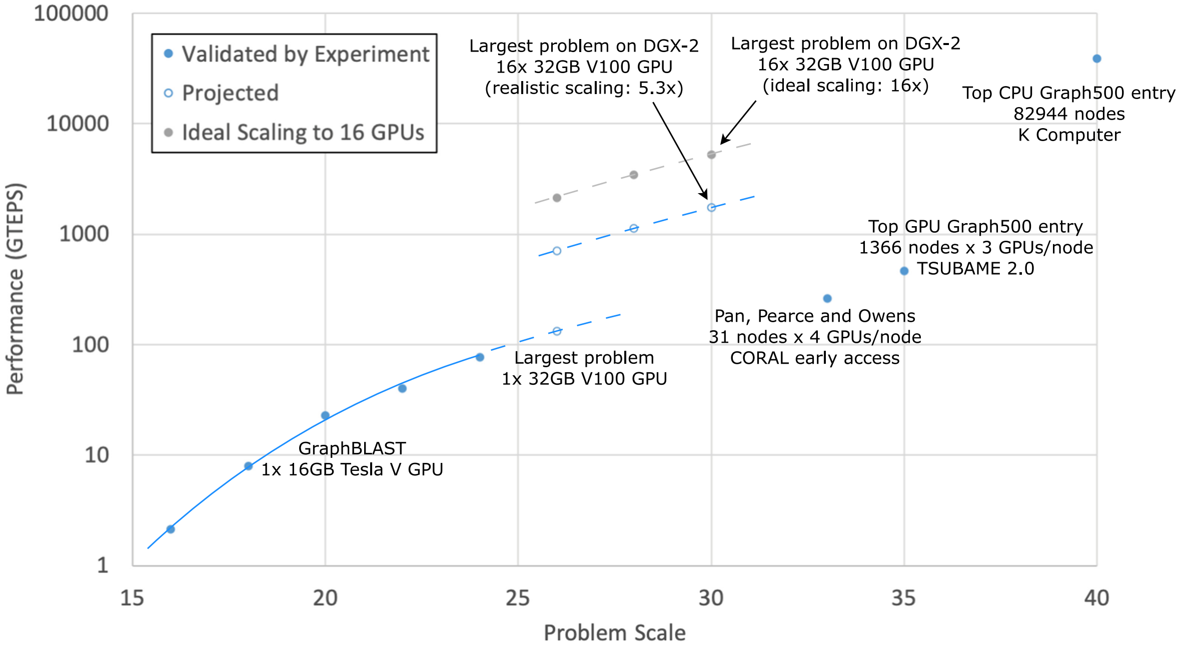
Graph500 BFS scaling

Top500 HPL scaling



Carl Yang, Aydın Buluç, John D. Owens: GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU

Graph Algorithms Are Hard to Scale (and not very efficient)



Maybe CPU/GPU is not the ideal machine for graph problems…

- Lots of transistor budget for extra FLOPs, which we don't need
- GPU has memory bandwidth, but at the cost of high latency

Time to look at different architectures!

Carl Yang, Aydın Buluç, John D. Owens: GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU

The Graphcore GC200 Intelligence Processing Unit

- 1472 cores per chip
- 6 Threads per core
- 6 cycles latency
- 624 KB SRAM/core

Relevant IPU features:

● Matrix units for AI acceleration	Booring
● MIMD rather than SIMD	Better than GPUs

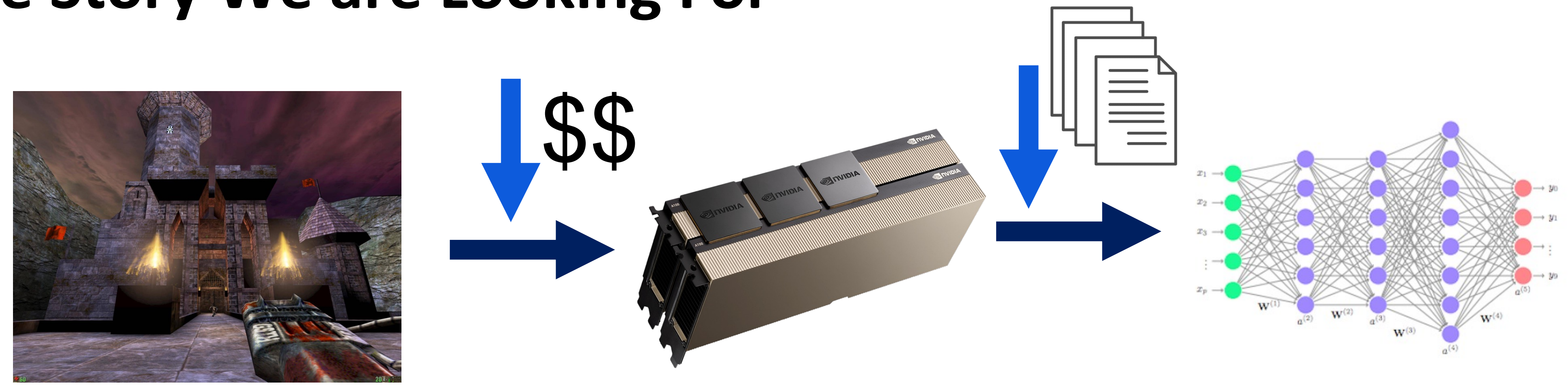
The Graphcore GC200 Intelligence Processing Unit

- 1472 cores per chip
- 6 Threads per core
- 6 cycles latency
- 624 KB SRAM/core

Relevant IPU features:

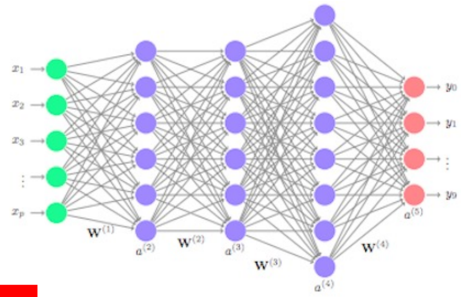
Matrix units for AI acceleration	Booring
MIMD rather than SIMD	Better than GPUs
High on-chip memory bandwidth	Important
Low memory latency	Yes please! (J.L. in 2018/19)
Has very slow DRAM	This will be a problem

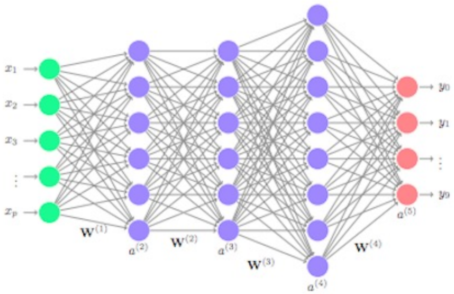
The Story We are Looking For



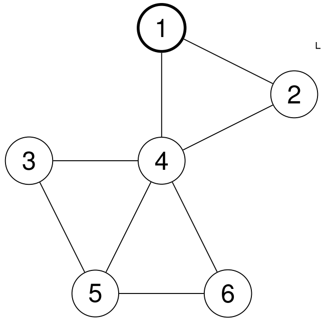
The Story We are Looking For

$$





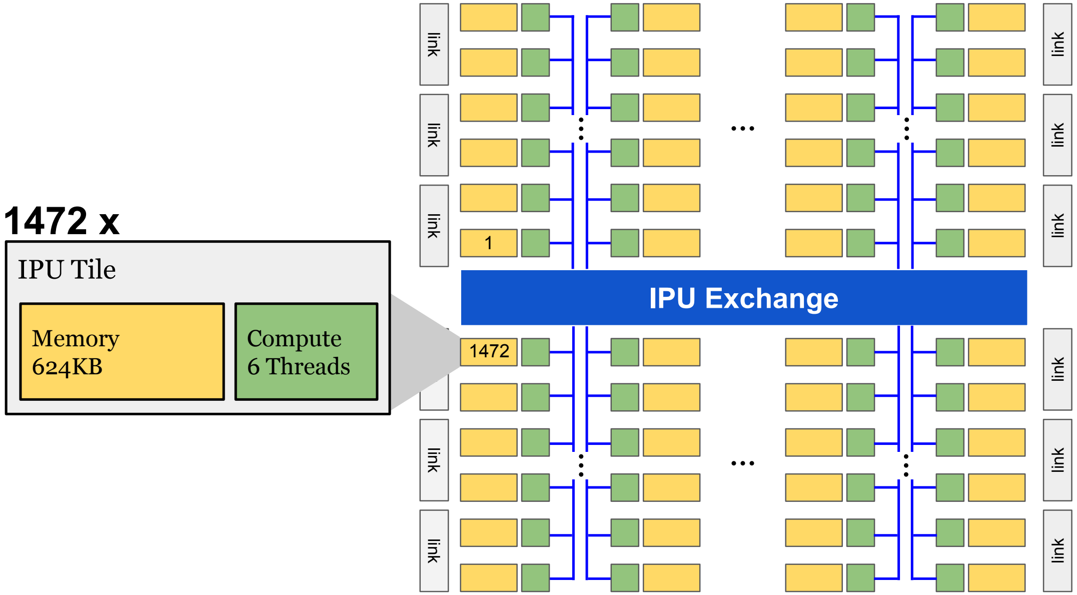
$$



“Let’s just give it a try”
(J.L. in 2020)

We are here now

The GC200 IPU: Shared and Distributed Memory on a Chip



1472 x

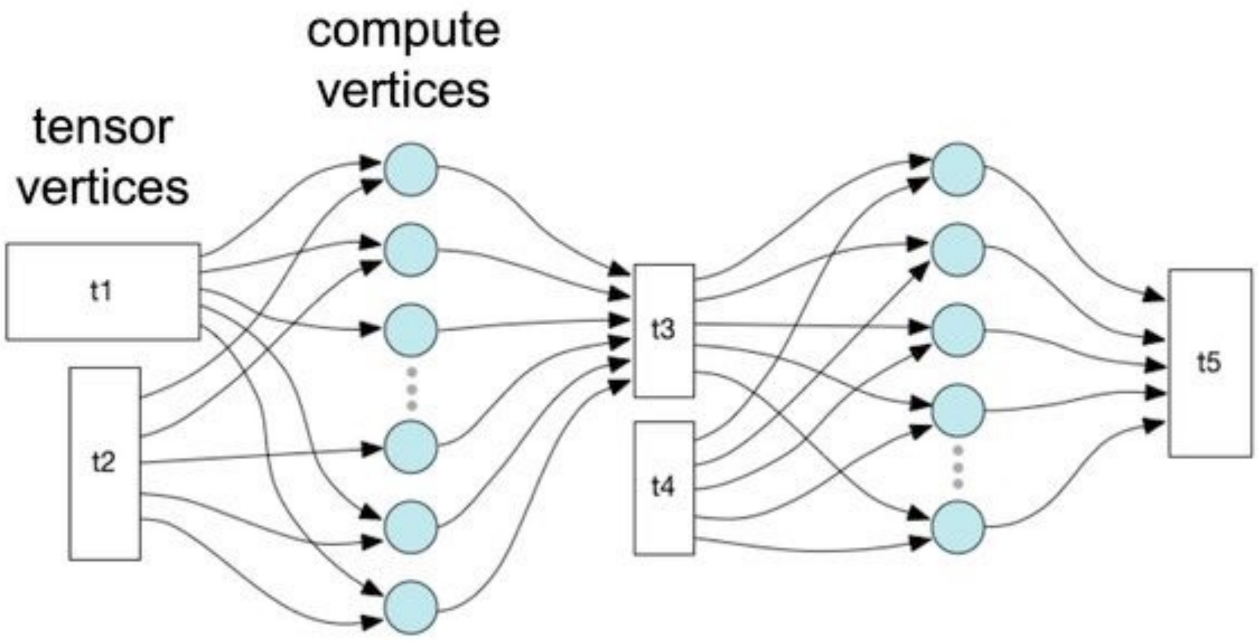
IPU Tile

Memory 624KB

Compute 6 Threads

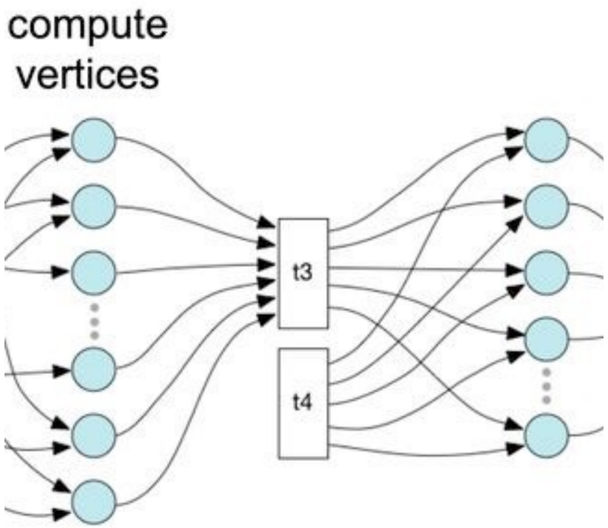
- 624 KB SRAM plus core form a tile
- 6 Threads per core
- 1472 tiles per chip
- Need to exploit very wide parallelism

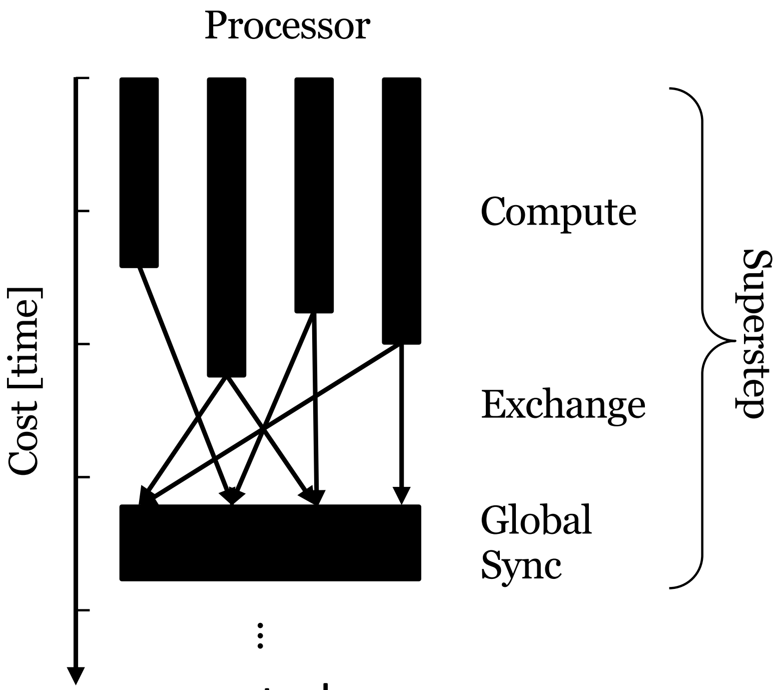
Dataflow-Based Programming Abstraction



- Data is arranged in immutable tensors
- Code is organized in codelets (compute vertices)
- Bipartite graph of data dependency
- Independent compute vertices are scheduled concurrently

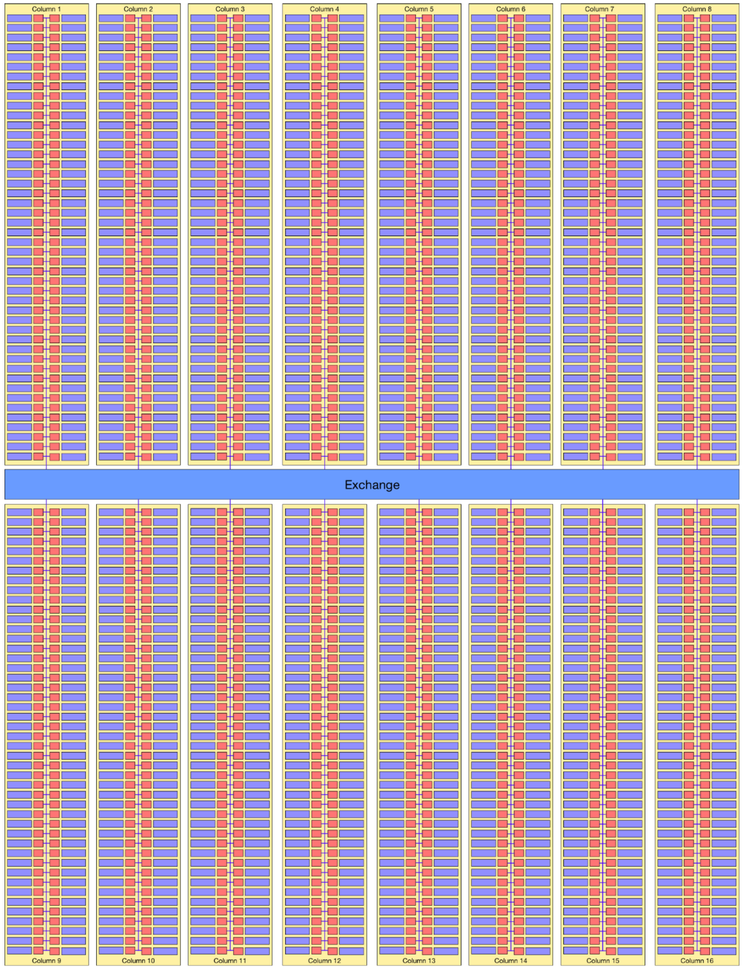
Bulk-Synchronous Parallel (BSP) Communication on the IPU





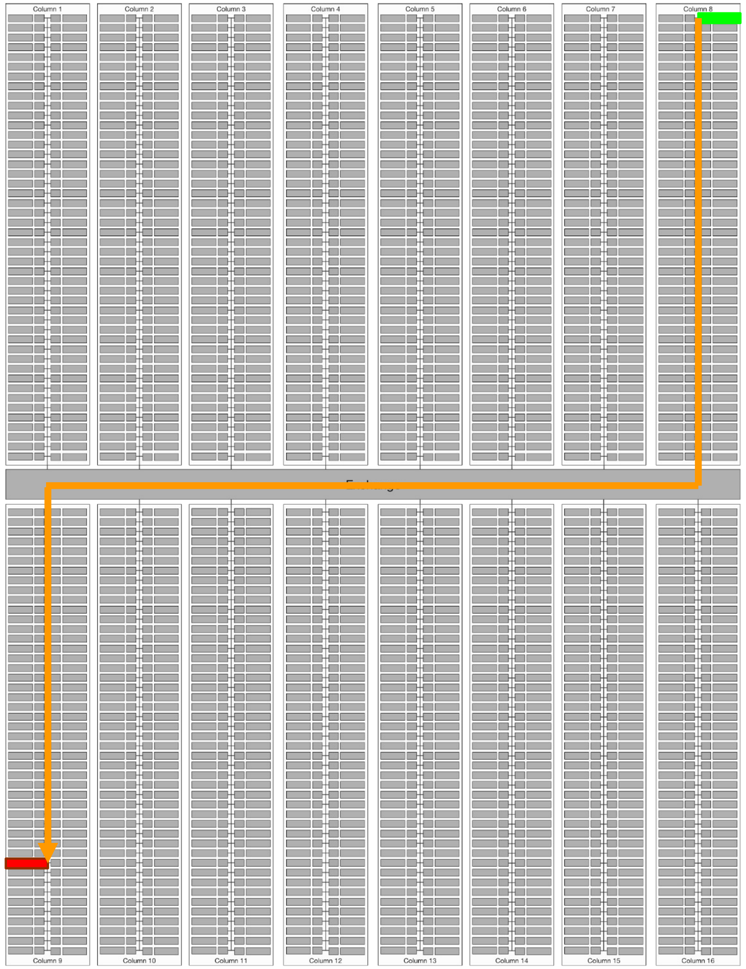
- Data exchange between concurrent phases
- Communication planned at compile time
- No communication/computation overlap
- No need for buffers

The GC200 IPU has 1472 individual cores and 8832 threads - spread into 4 islands and 16 columns



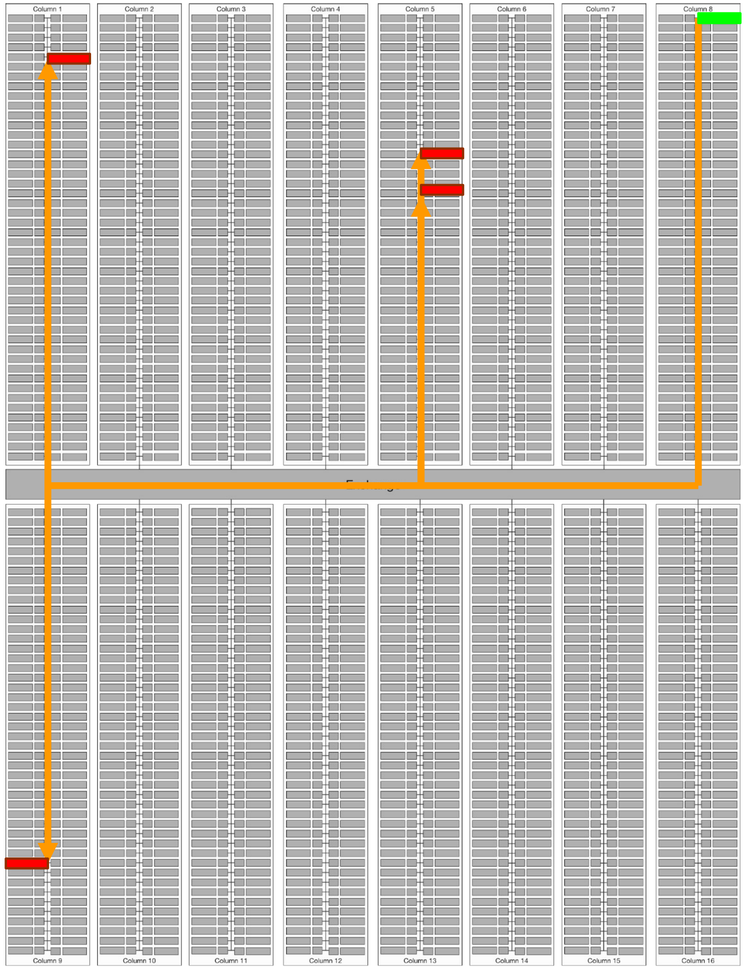
- 1472 cores per chip
- 6 Threads per core
- 6 cycles latency
- 624 KB SRAM/core

Unrestricted point-to-point communication between tiles



More complex communication patterns with broadcasts are possible

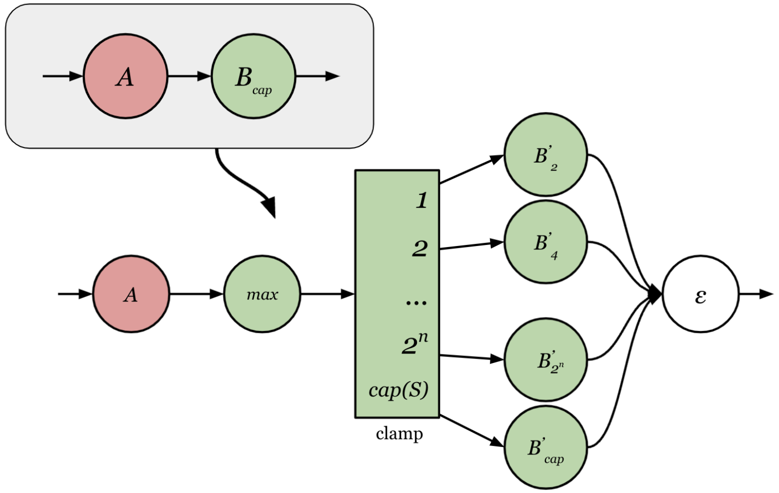
- Per tile BW: ~ 5 GB/s
- Aggregate BW: ~ 8TB/s
- No overlap
- Preplanned communication, messy to get around that.



Irregular Communication on the IPU

- Poplar framework offers PGAS-style data exchange
  PGAS = Partitioned Global Address Space
  (programmer sees shared memory, system handles data exchange)

- Communication can be optimized by controlling data placement
- Communication codes exist at compile time, but we can choose
  between calling different codes at runtime



Reduction in irregular computation

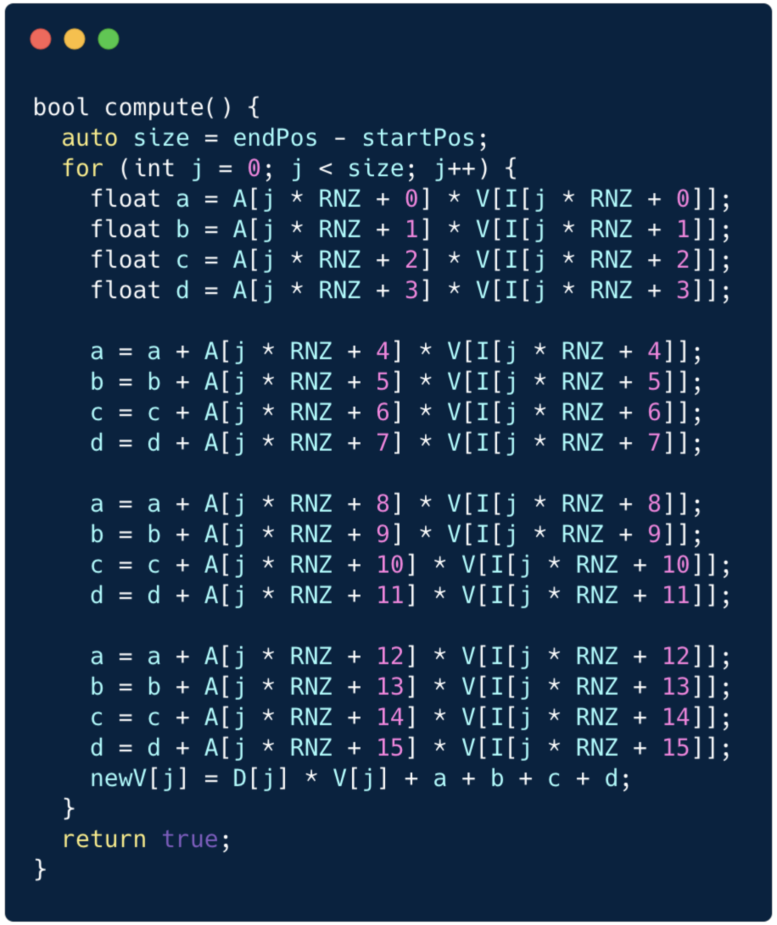
The IPU is a Barrel Processor

- Symmetric multithreading (a.k.a. hyperthreading)
- 2/4/8 threads per core
- Scheduling order variable
- Allows latency hiding

- Temporal multithreading
- 6 threads per core
- Scheduling order fixed

- Individual thread does not experience latency but…
- To use all retirement slots, we need to keep all threads busy
- Easiest to do if all 6 threads work on independent subproblems, but…
- There is usually too little memory for that

Individual IPU Code Optimization is Easy

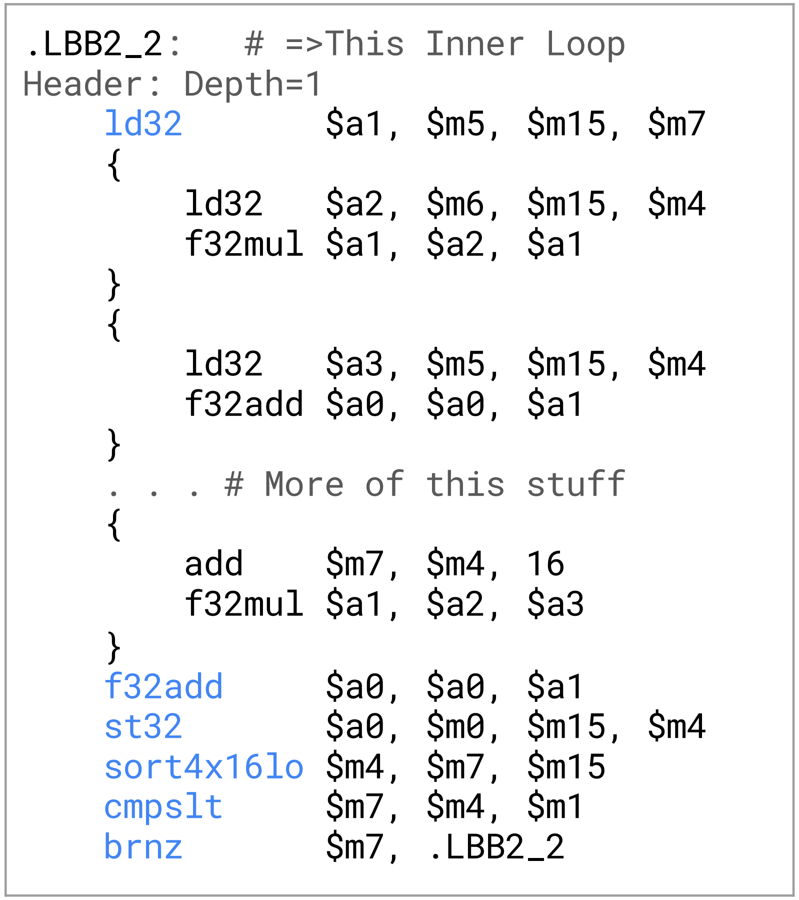


```
bool compute() {
  auto size = endPos - startPos;
  for (int j = 0; j < size; j++) {
    float a = A[j * RNZ + 0] * V[I[j * RNZ + 0]];
    float b = A[j * RNZ + 1] * V[I[j * RNZ + 1]];
    float c = A[j * RNZ + 2] * V[I[j * RNZ + 2]];
    float d = A[j * RNZ + 3] * V[I[j * RNZ + 3]];

    a = a + A[j * RNZ + 4] * V[I[j * RNZ + 4]];
    b = b + A[j * RNZ + 5] * V[I[j * RNZ + 5]];
    c = c + A[j * RNZ + 6] * V[I[j * RNZ + 6]];
    d = d + A[j * RNZ + 7] * V[I[j * RNZ + 7]];

    a = a + A[j * RNZ + 8] * V[I[j * RNZ + 8]];
    b = b + A[j * RNZ + 9] * V[I[j * RNZ + 9]];
    c = c + A[j * RNZ + 10] * V[I[j * RNZ + 10]];
    d = d + A[j * RNZ + 11] * V[I[j * RNZ + 11]];

    a = a + A[j * RNZ + 12] * V[I[j * RNZ + 12]];
    b = b + A[j * RNZ + 13] * V[I[j * RNZ + 13]];
    c = c + A[j * RNZ + 14] * V[I[j * RNZ + 14]];
    d = d + A[j * RNZ + 15] * V[I[j * RNZ + 15]];
    newV[j] = D[j] * V[j] + a + b + c + d;
  }
  return true;
}
```



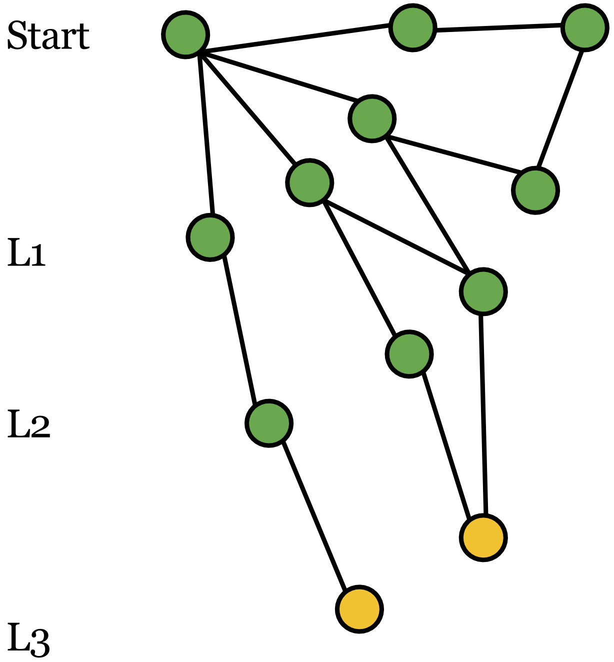
```
.LBB2_2:   # =>This Inner Loop
Header: Depth=1
    ld32          $a1, $m5, $m15, $m7
    {
        ld32    $a2, $m6, $m15, $m4
        f32mul  $a1, $a2, $a1
    }
    {
        ld32    $a3, $m5, $m15, $m4
        f32add  $a0, $a0, $a1
    }
    . . . # More of this stuff
    {
        add     $m7, $m4, 16
        f32mul  $a1, $a2, $a3
    }
    f32add        $a0, $a0, $a1
    st32          $a0, $m0, $m15, $m4
    sort4x16lo    $m4, $m7, $m15
    cmpslt        $m7, $m4, $m1
    brnz          $m7, .LBB2_2
```

Computation optimization is straightforward:

minimize number of instructions

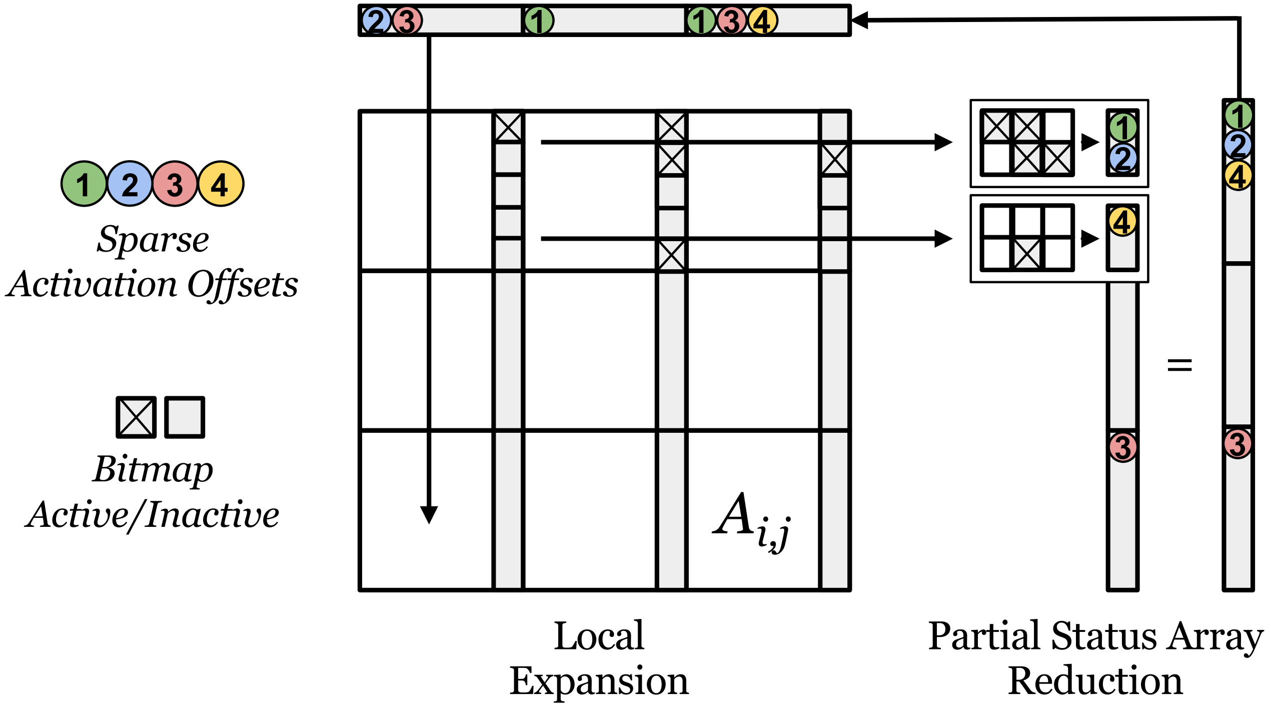
The Hello World of Graph Algorithms: BFS

```
Let Q be the frontier
Let Q’ be the next frontier
Let G(V,E) be the Graph
Let B be visited nodes
```



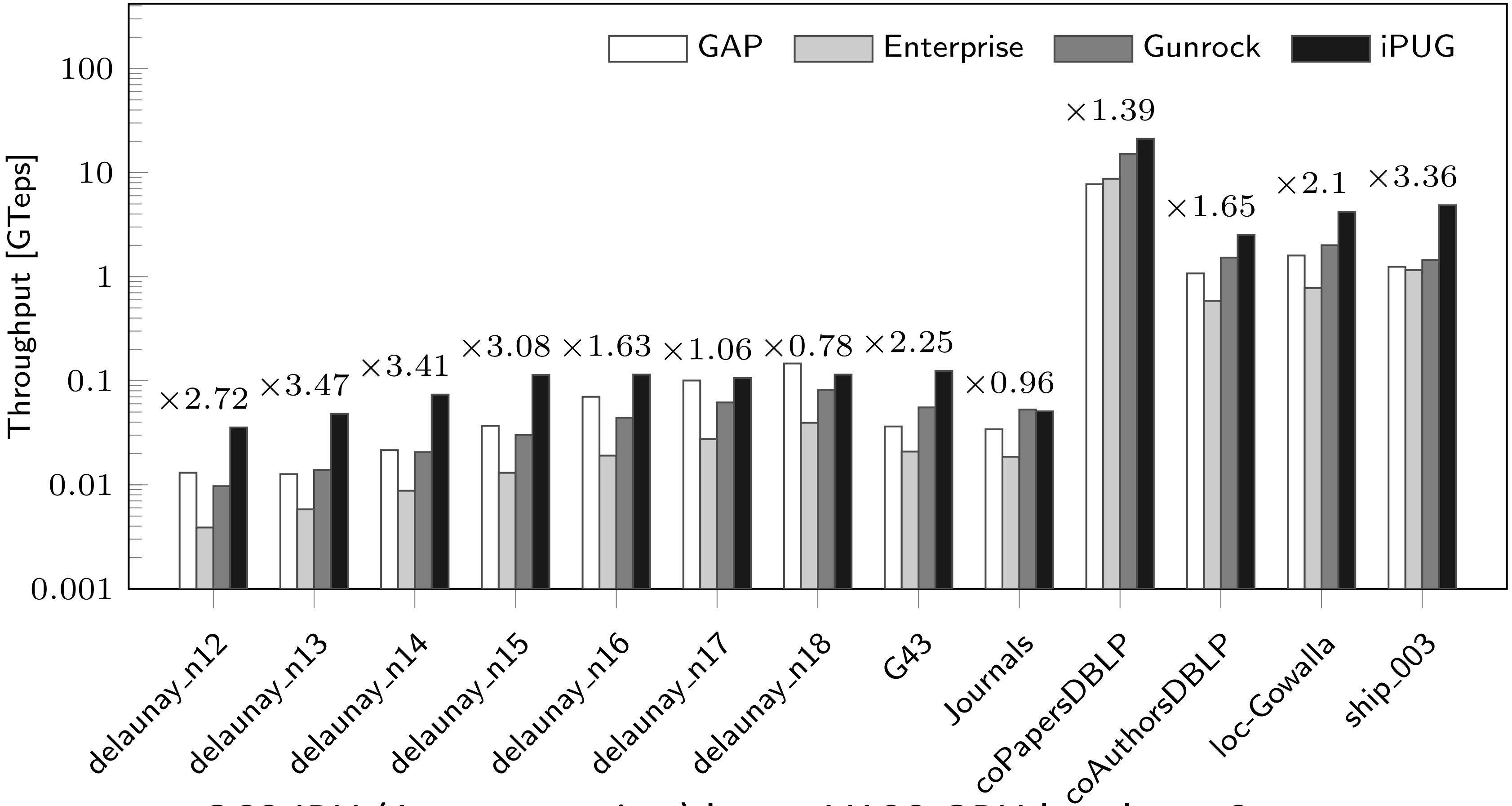
- Basic measure of graph processing performance
- Single O(n+m) execution, no time for expensive partitioning etc…
- Need 2D block partitioning for power-law graphs

Mapping to the IPU



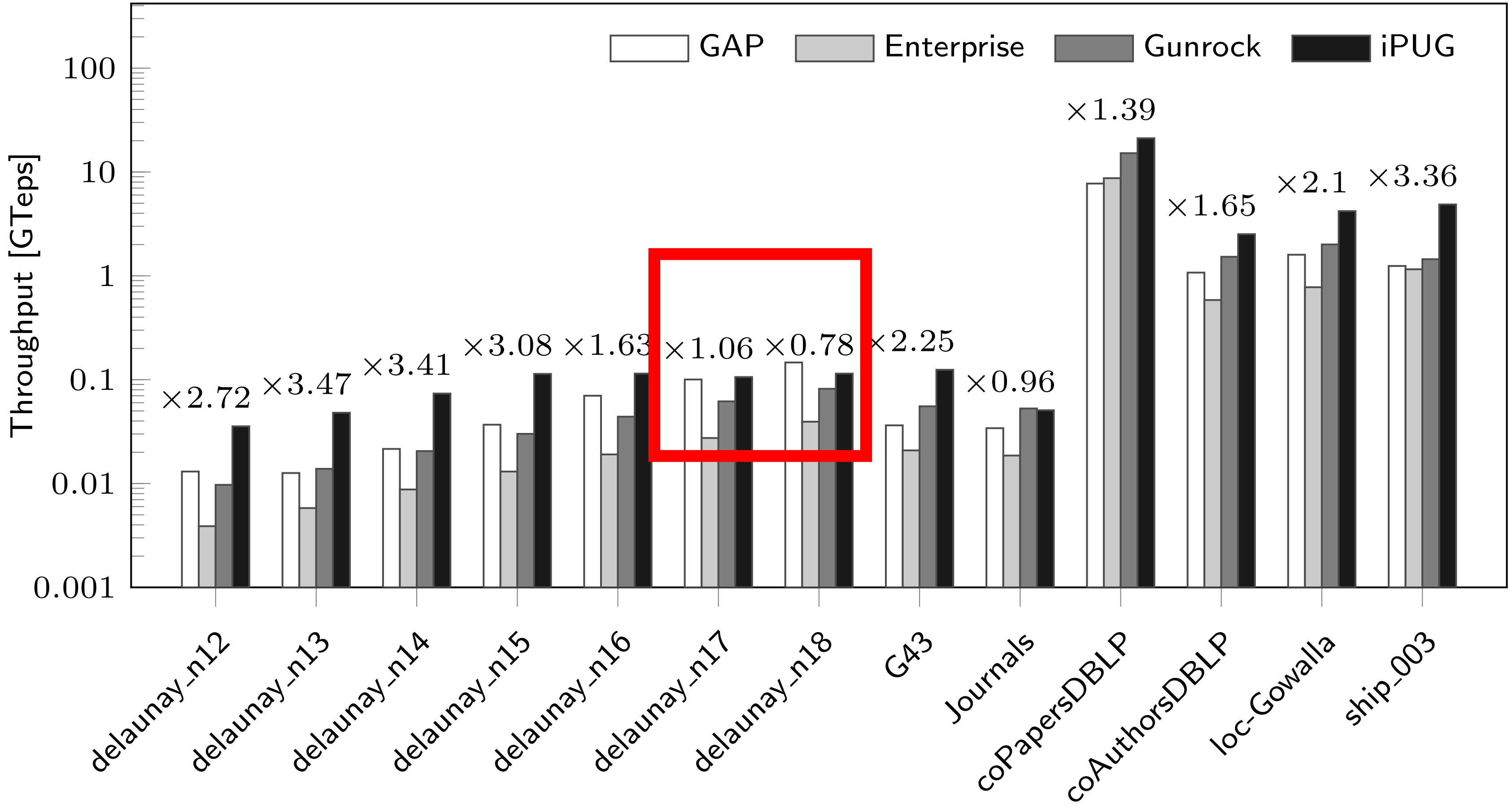
Burchard et al.: iPUG: Accelerating Breadth-First Graph Traversals Using Manycore Graphcore IPUs, 2021

Single Device BFS – IPU vs GPU vs CPU



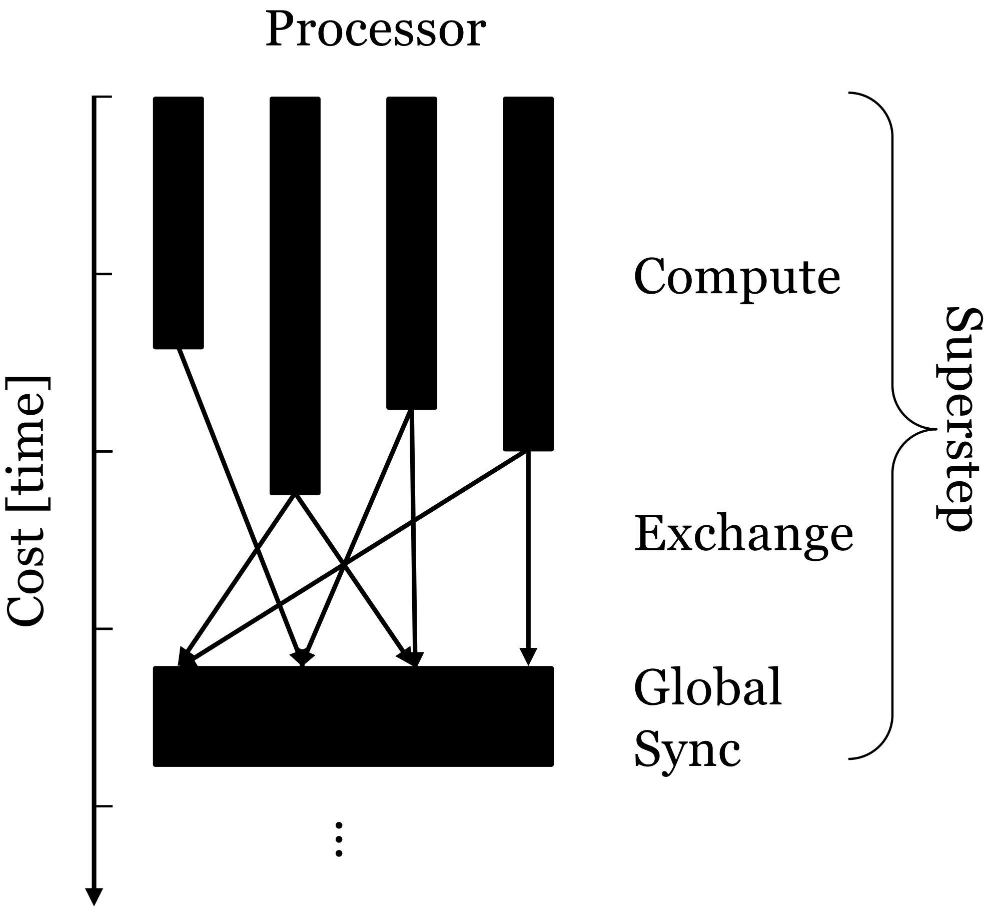
- GC2 IPU (1st generation) beats V100 GPU by about 2x
- CG200 vs A100 is very comparable
- Very good performance/watt

IPU vs CPU



- Needs a lot of parallelism
- GPU is the *real* competitor

Why not more GTEPS?

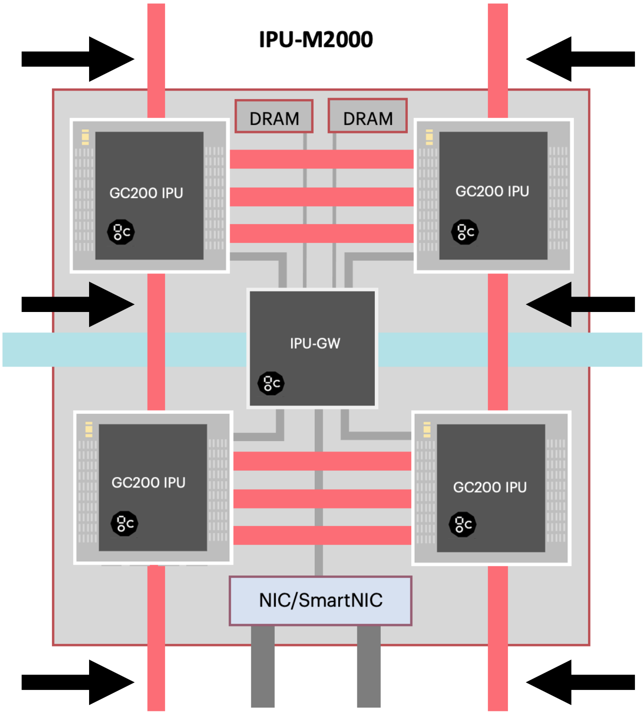


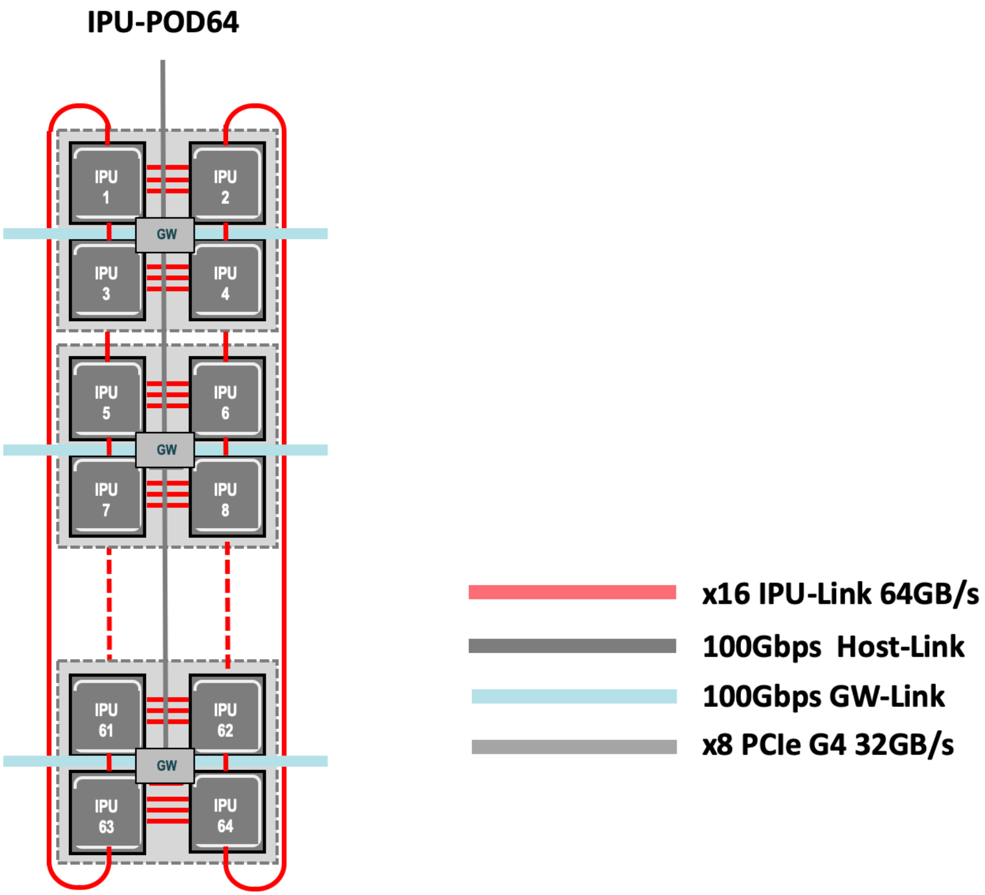
- Global synchronization
- No automatic load balancing at all. Full imbalance penalty (as normal for 1472x distributed memory)
- Dynamic load balancing makes little sense for BFS
- Far better results on dense graphs

- Cannot beat CPU on high-diameter graphs (little parallelism)
- Still needs fewer active threads than GPU:

  8.8K vs 6.6K - 212.9K

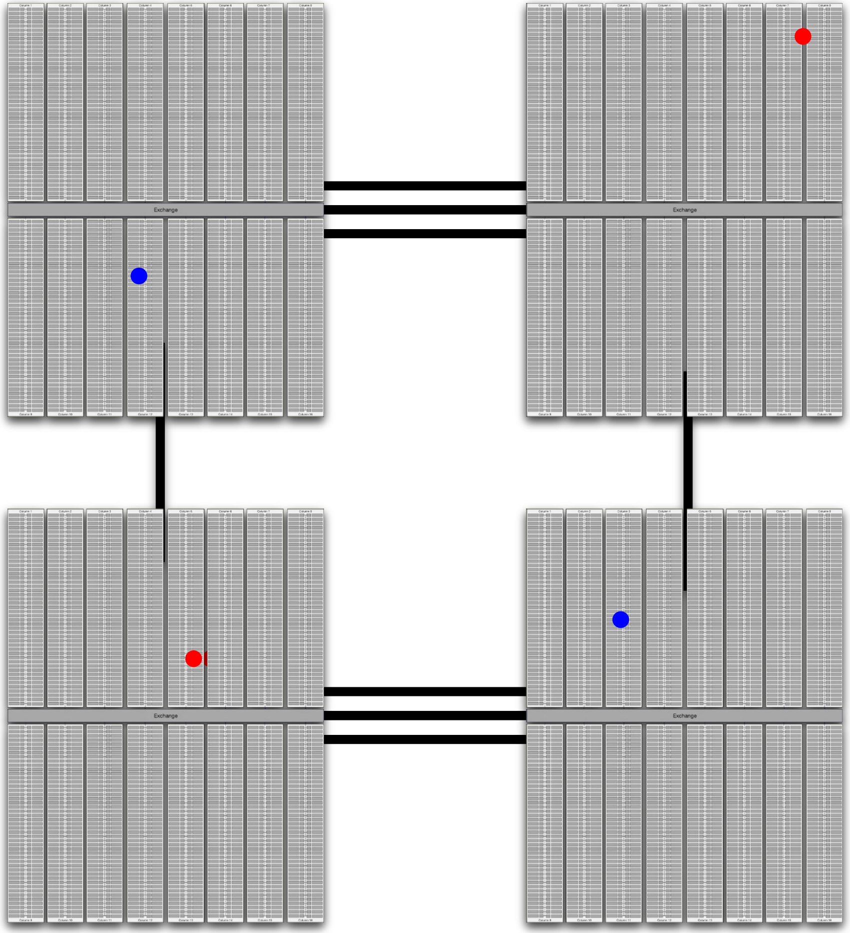
Scaling Out: Multi-IPU BFS



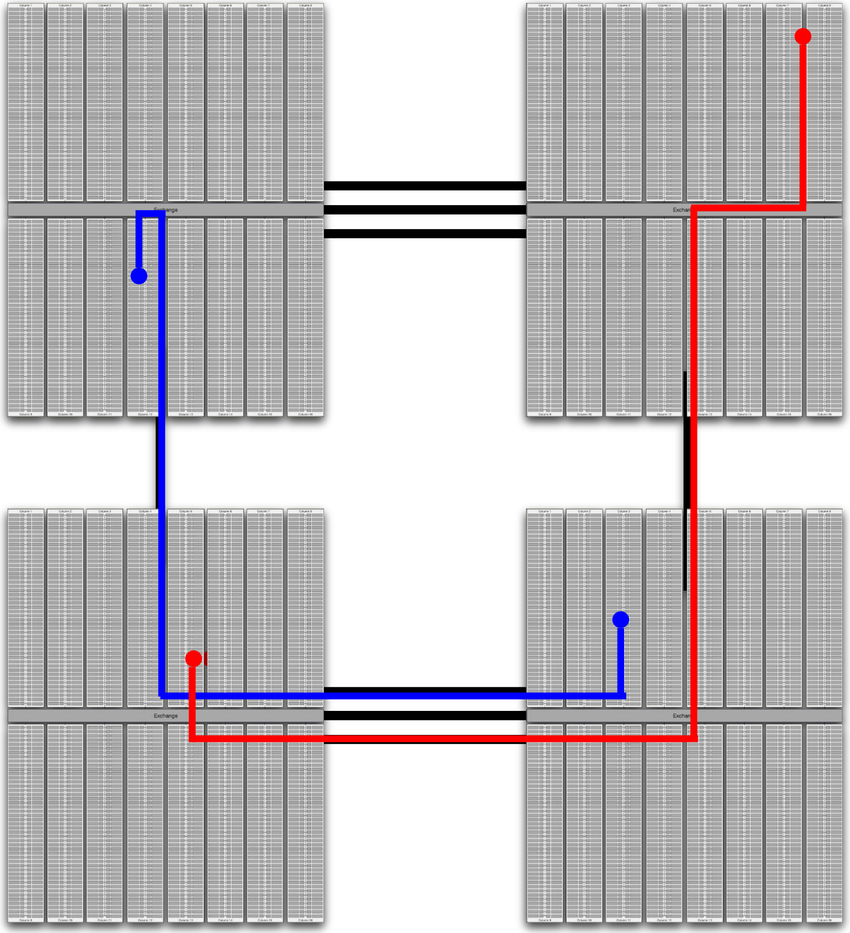


- Computation can scale to multiple IPUs
- BFS is extremely communication-heavy
- Large-scale problems are mostly network dependent

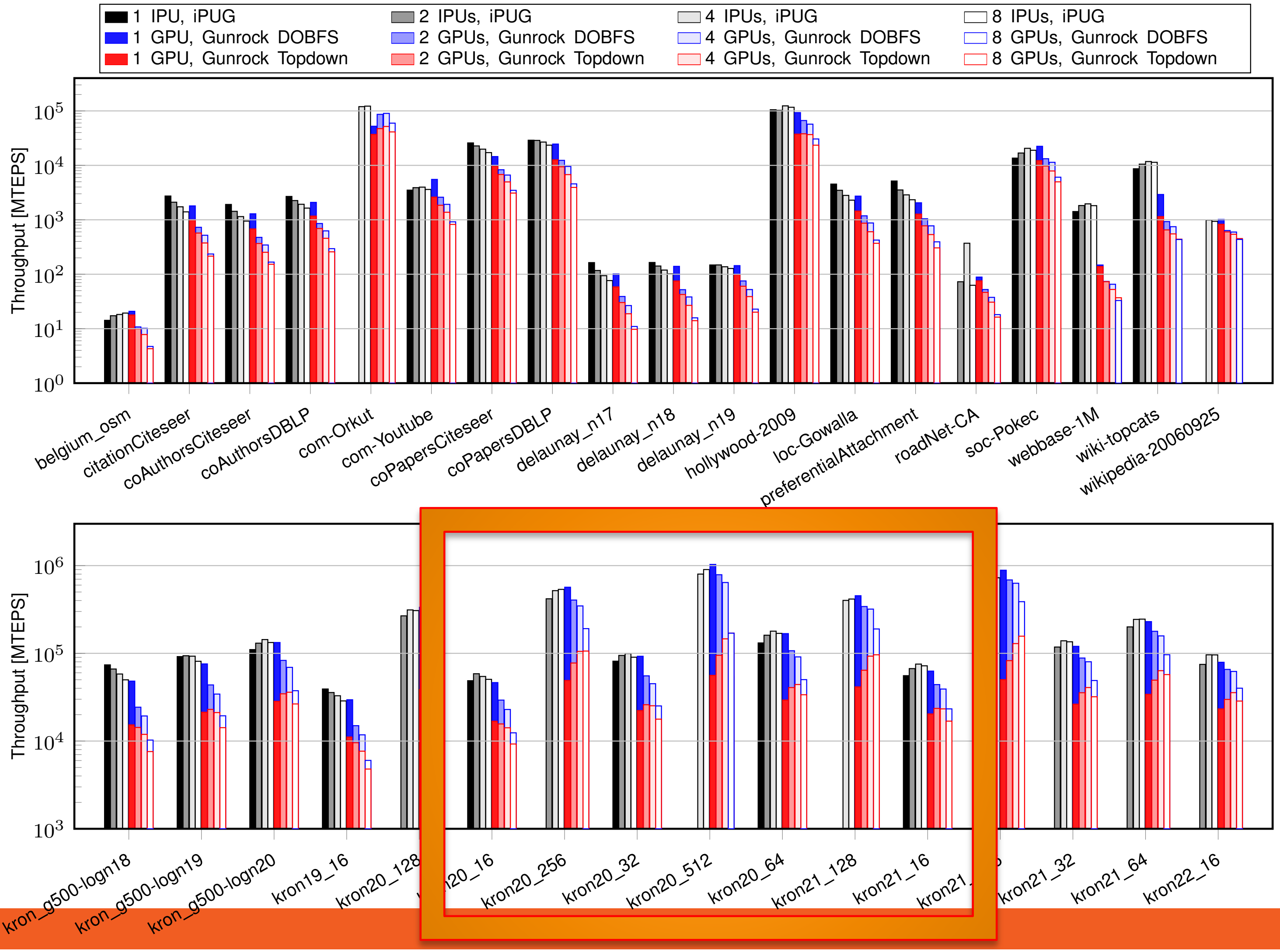
Using multiple IPUs, we are using the same mapping techniques as with the single device



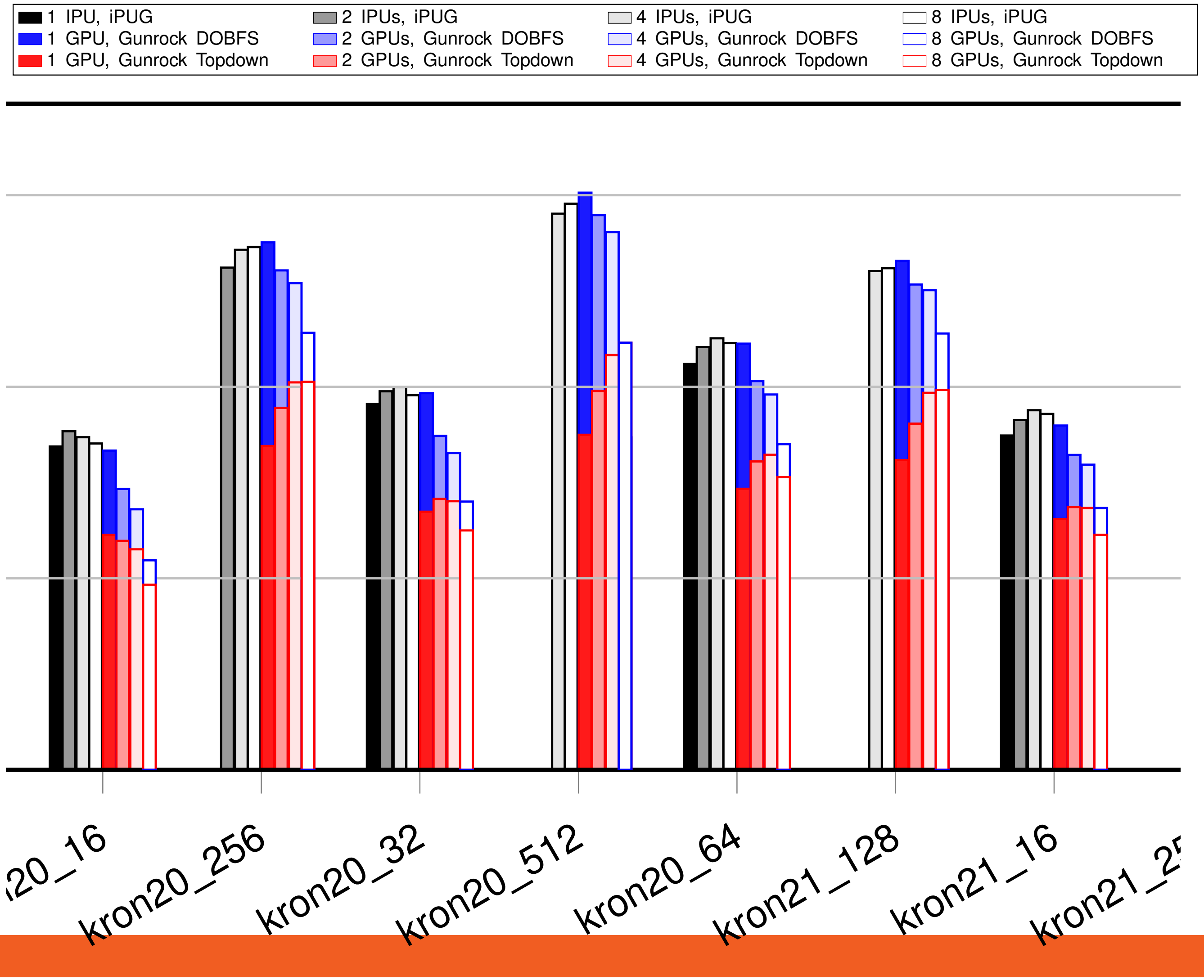
Using multiple IPUs, we are using the same mapping techniques as with the single device



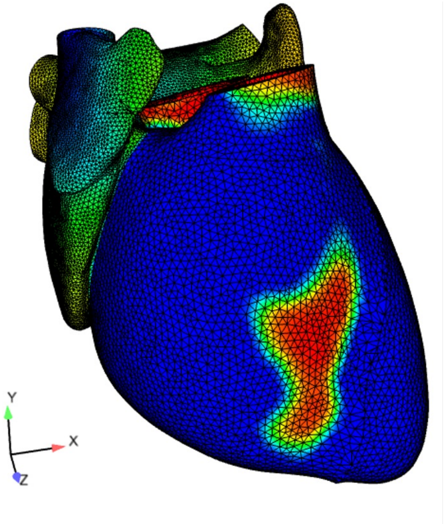
Multiple Device BFS – IPU vs GPU



Multiple Device BFS – IPU vs GPU



Monodomain simulation in cardiac electrophysiology using Lynx



- Tetrahedral mesh for finite volume simulation
- ODE reaction model (ten Tusher)
- PDE diffusion model (SpMV)
- *Large* number of identical time steps

  Allows for lots of optimization techniques including load balancing and partitioning

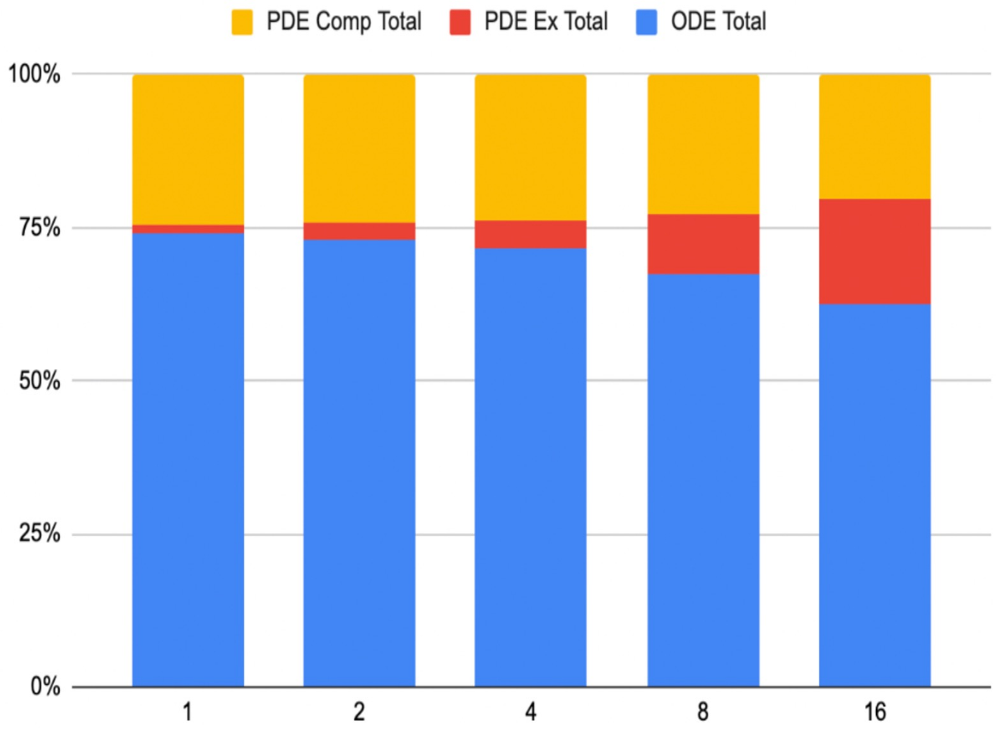
Repeated SpMV: Minimize Communication via Partitioning

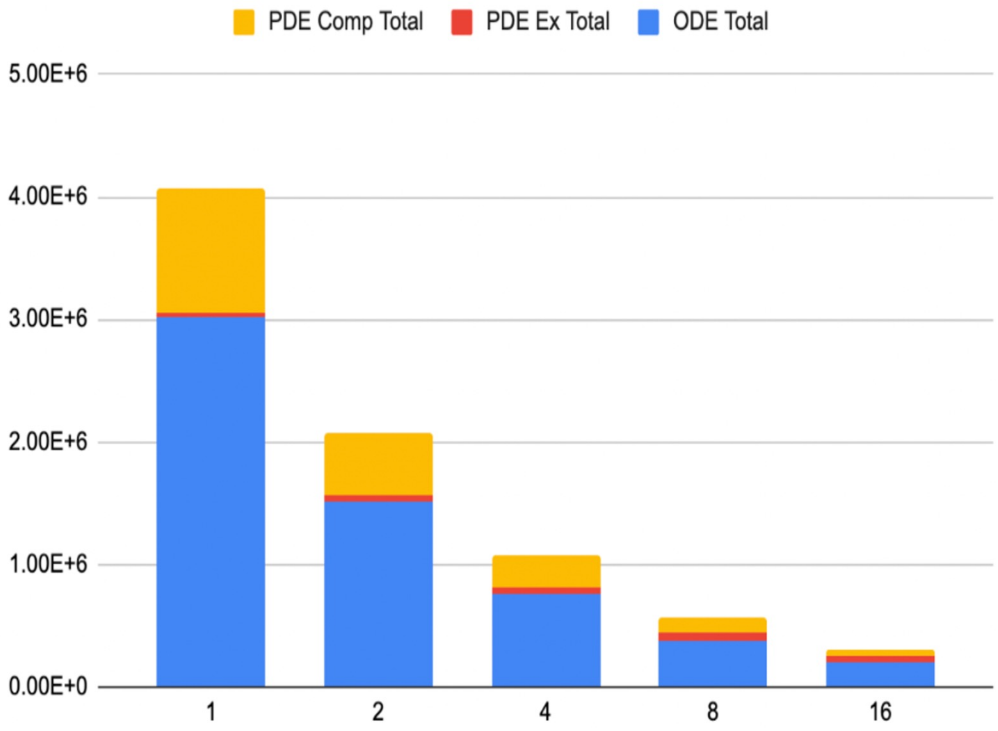
vs

	2x GC200 IPU FP32	V100 GPU FP32	V100 GPU FP64
PDE	0.21	1.208	1.836
ODE	2.52	2.056	2.822
Sum	2.75	3.264	4.658
PDE multistep 128:1	29.4	156.68	239.666

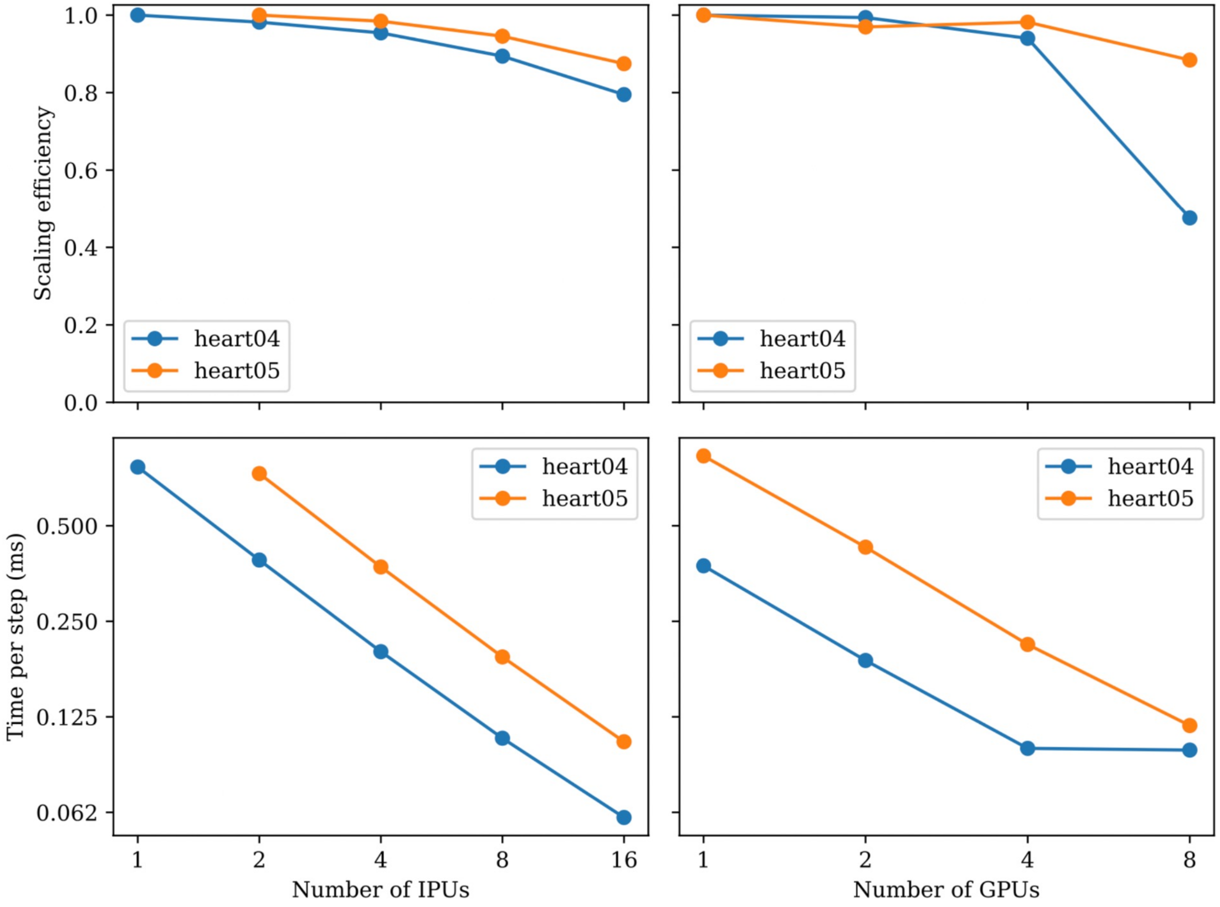
- GPU has more FLOPS for ODE (code does not use IPU matrix units)
- FP32 is a severe limitation for scientific computing
- PDE (SpMV) is much faster on IPU. Dominates multisteping.

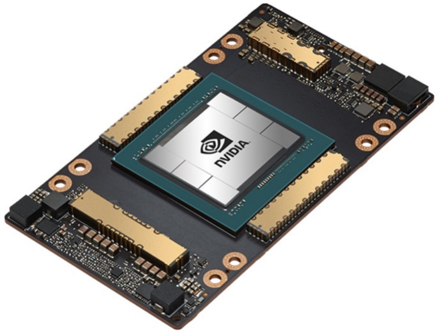
We are increasingly spending more time on exchanges with tiles receiving values for up to 80% of their cells





Currently, the IPU is compute bound; the ODE step is much slower than on the A100





IPU Partitioning Problem

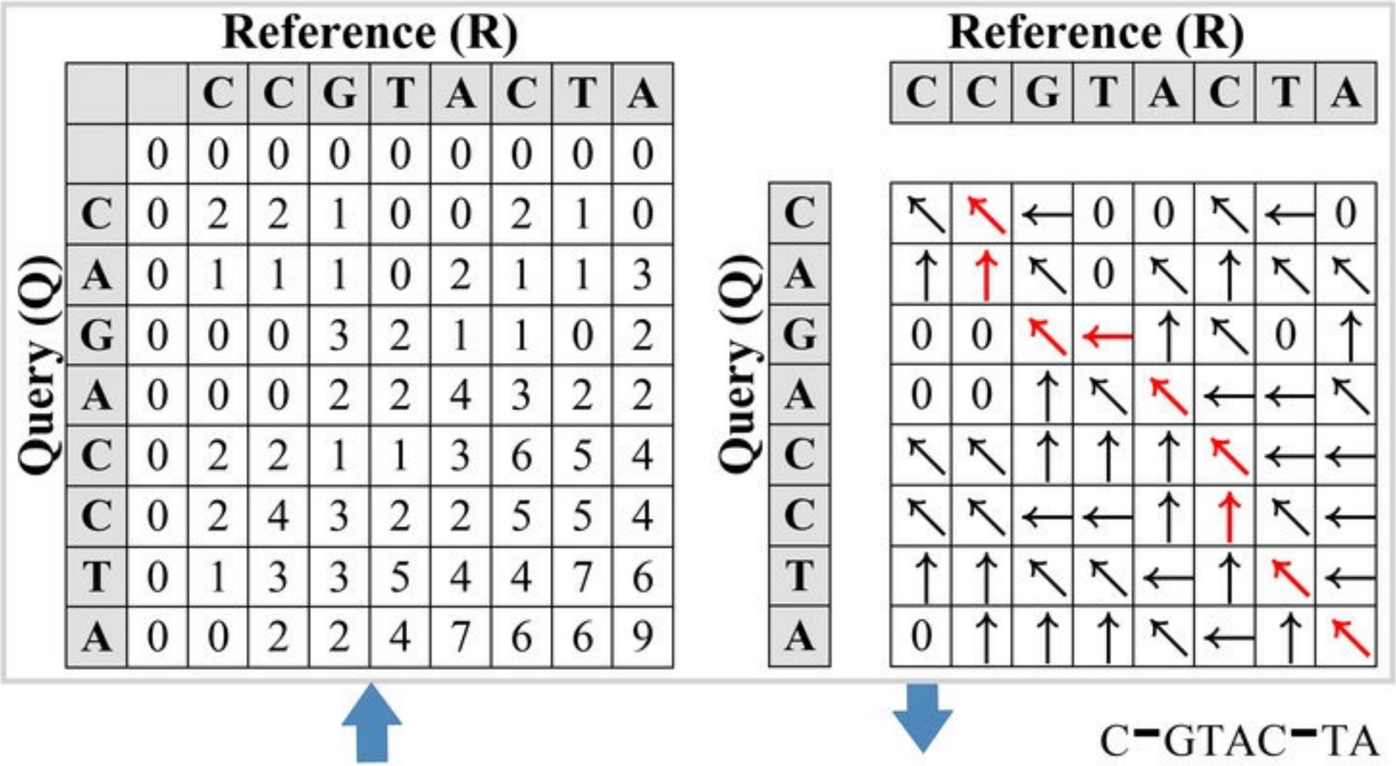
Small scale distributed memory creates special partitioning problem:

1. Minimize cutsize (as usual)
2. Create large number of parts (1472) efficiently
3. Balance total part size (vertices plus ghost cells)
4. Partition hierarchically for multi-IPU
5. Optimize mapping along 1D ladder topology (hard)

Large benefit from load balanced and communication optimized data distribution

Very fast for e.g. Page-Rank

Sequence Alignment



Reference (R)

Query (Q)

		C	C	G	T	A	C	T	A
	0	0	0	0	0	0	0	0	0
C	0	2	2	1	0	0	2	1	0
A	0	1	1	1	0	2	1	1	3
G	0	0	0	3	2	1	1	0	2
A	0	0	0	2	2	4	3	2	2
C	0	2	2	1	1	3	6	5	4
C	0	2	4	3	2	2	5	5	4
T	0	1	3	3	5	4	4	7	6
A	0	0	2	2	4	7	6	6	9

Reference (R)

Query (Q)

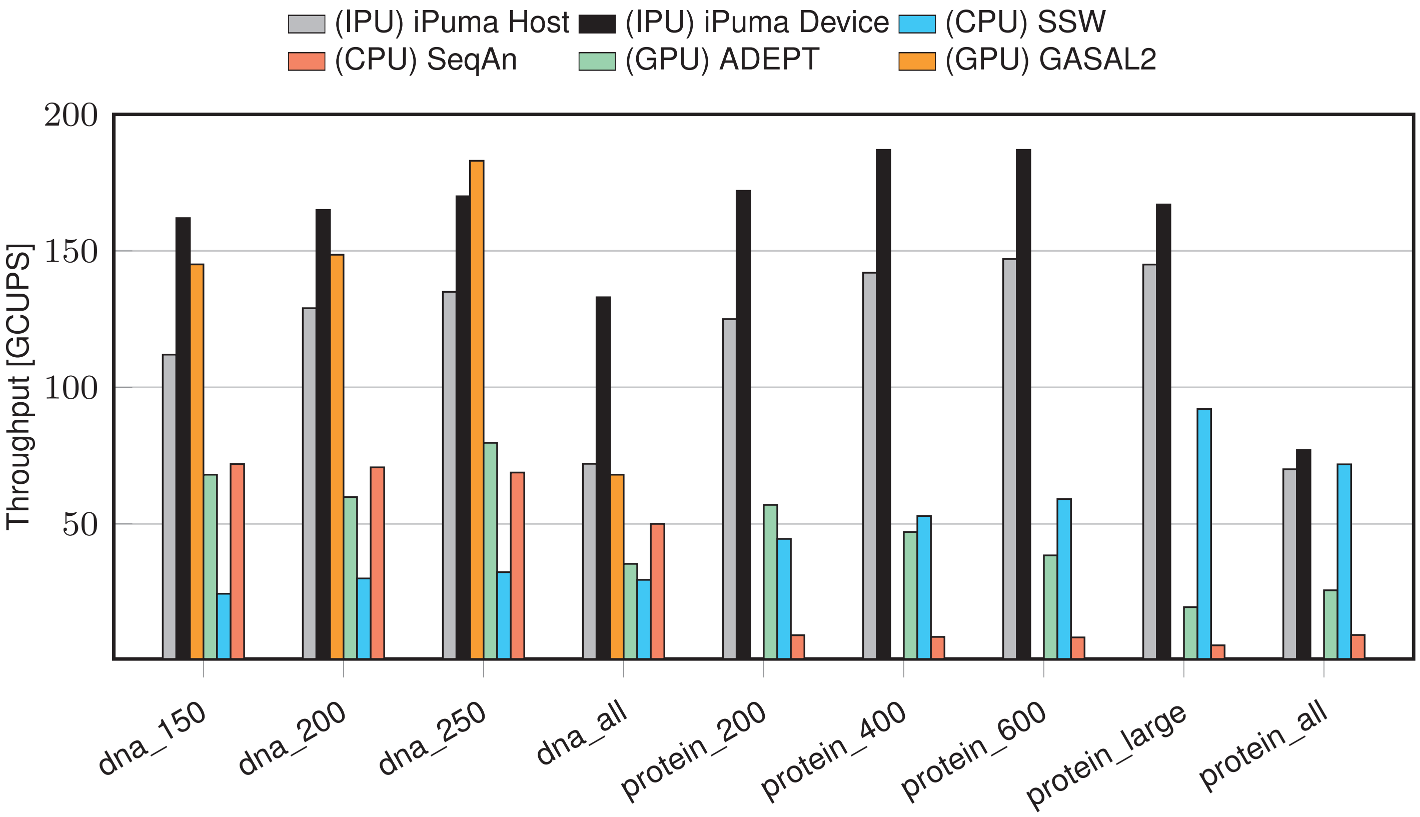
	C	C	G	T	A	C	T	A
C	↖	↖	←	0	0	↖	←	0
A	↑	↑	↖	0	↖	↑	↖	↖
G	0	0	↖	←	↑	↖	0	↑
A	0	0	↑	↖	↖	←	←	↖
C	↖	↖	↑	↑	↑	↖	←	←
C	↖	↖	←	←	↑	↑	↖	←
T	↑	↑	↖	↖	←	↑	↖	←
A	0	↑	↑	↑	↖	←	↑	↖

Reference (R): CCGTACTA
Query (Q): CAGACCTA

Alignment:
Score = 9

```
C-GTAC-TA
|  | || ||
CAG-ACCTA
```

Smith Waterman Algorithm



- MIMD IPU better at dealing with irregularity than SIMD GPU
- Host connection becomes a bottleneck

Other Applications

- SpMV: about 200 GFLOP/s
- Coloring (backtracking) with dynamic load balancing: 2x faster than CPU
- X-Drop sequence alignment: 4-10x vs A100
- BERT ~ 4x faster inference than A100 (4x slower on train)
- Hungarian algorithm (Matching): 4-10x vs A100

Conclusions

- GC 200 IPU often allows 2-10x speedups over A100
- Both devices are very comparable (7nm, about 60 billion transistors)
- IPU implementations only work for specific sizes (small problems)
- Need lots of IPUs for larger problems

- First IPU implementations compete with very mature GPU codes

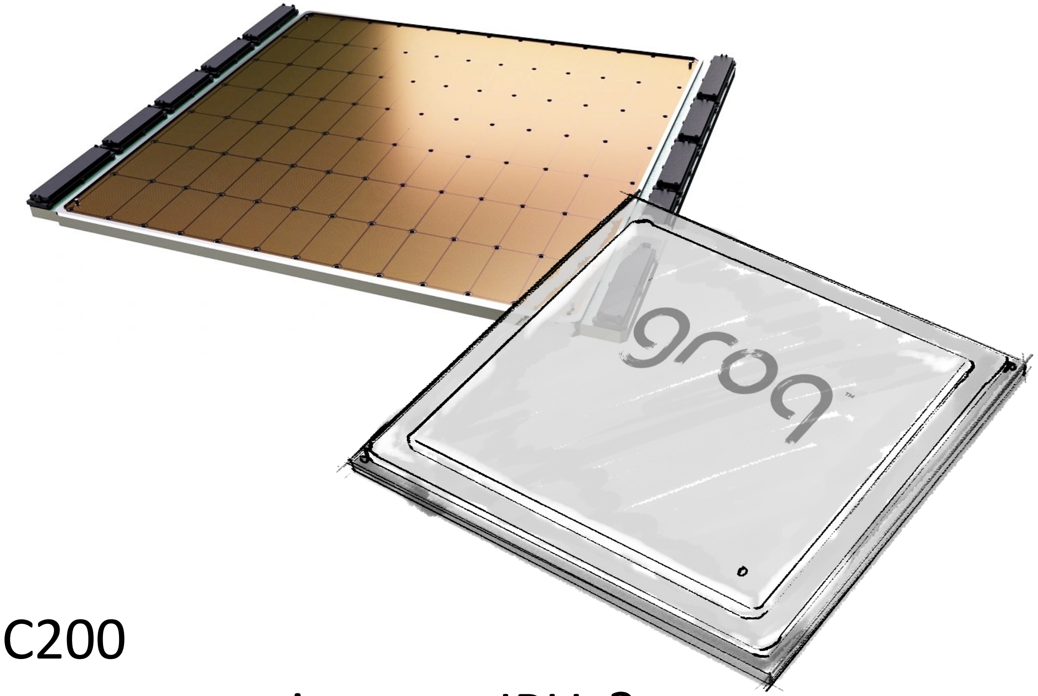
And now for the bad News

- Bow IPU from 2022 is just upclocked GC200
- Not clear what the future of the IPU is like – this is a problem

And now for the bad News

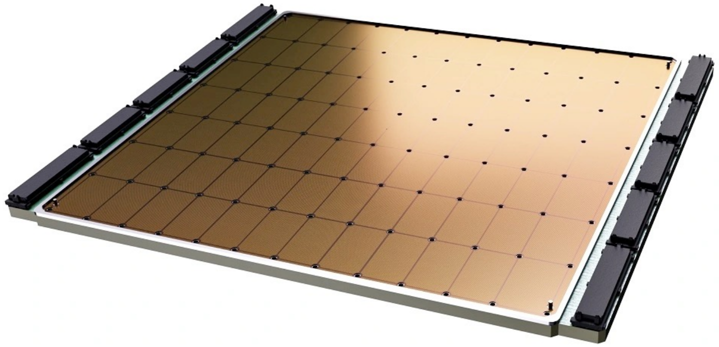
- Bow IPU from 2022 is just upclocked GC200
- Softbank recently bought Graphcore – are we getting new IPUs?

And now for the bad News



- Bow IPU from 2022 is just upclocked GC200
- Softbank recently bought Graphcore – are we getting new IPUs?
- However: there are many competitors with similar chips
- Next step: apply lessons learned to other *tile-centric* devices
- Goal: develop a theory of *tile-centric* graph algorithms

Potential for Processing Large Graphs: Cerebras WSE-2 Wafer Scale Engine

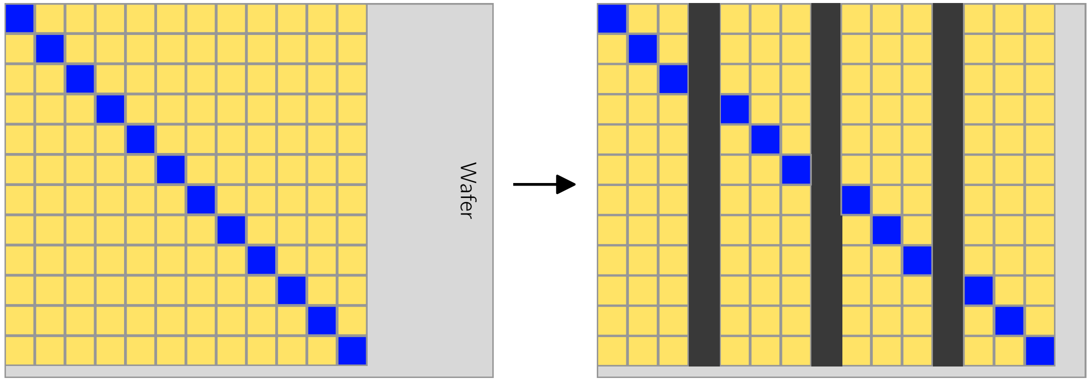


- Similar design to IPU
- Tiles with compute & SRAM memory
- 850 × 1000 cores/tiles
- Roughly equivalent to 64 CPU/GPU/IPU
- Fast communication among neighboring processors (2D mesh)

➢ Miniature distributed memory no longer works here.
Need to embed problem into 2D space.

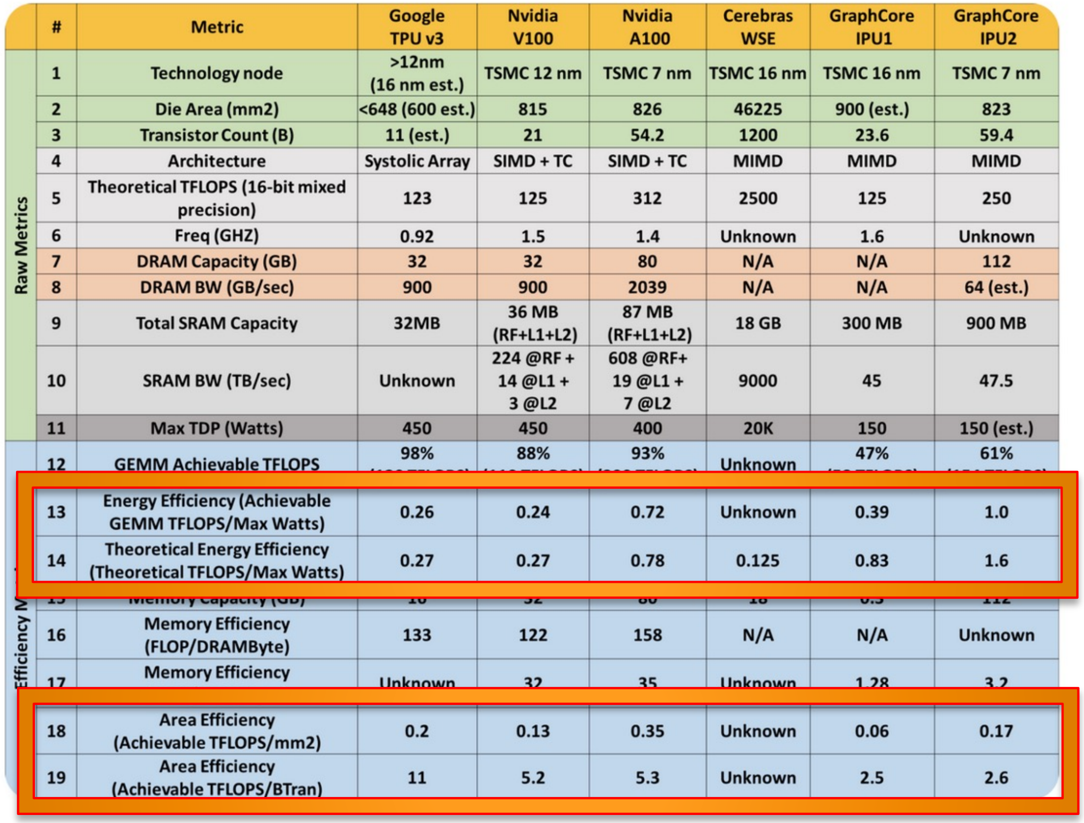
BFS on the Cerebras WSE-2 Wafer Scale Engine

- Working prototype
- Still based on 2D adjacency matrix decomposition
- Neighbor too neighbor communication causes imbalance
- Exploit rectangular shape of WSE to add functional units (filters)
- Use additional tiles to increase available memory along diagonal



Fair Comparisons Between Different Processors are Not Easy

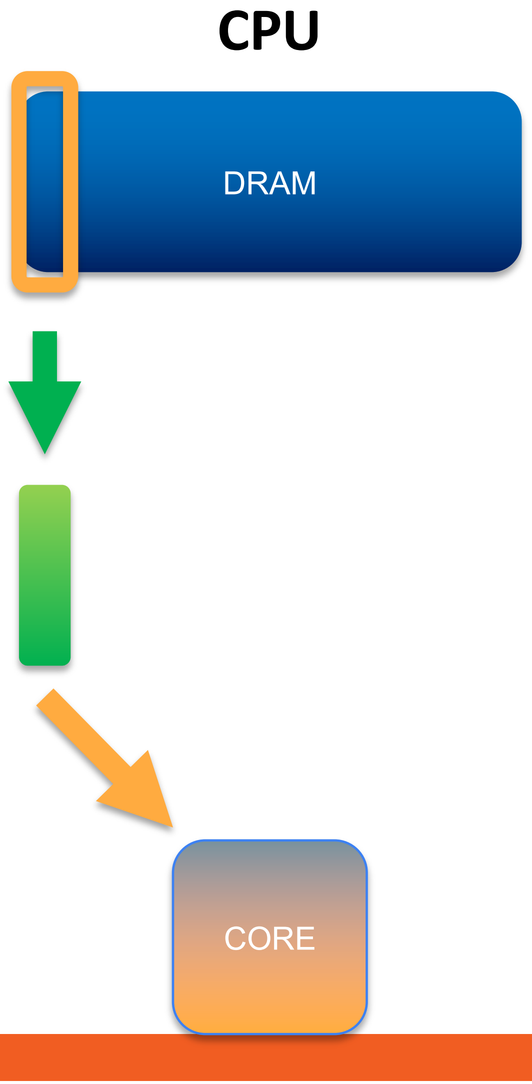
MLPerf efficiency metrics



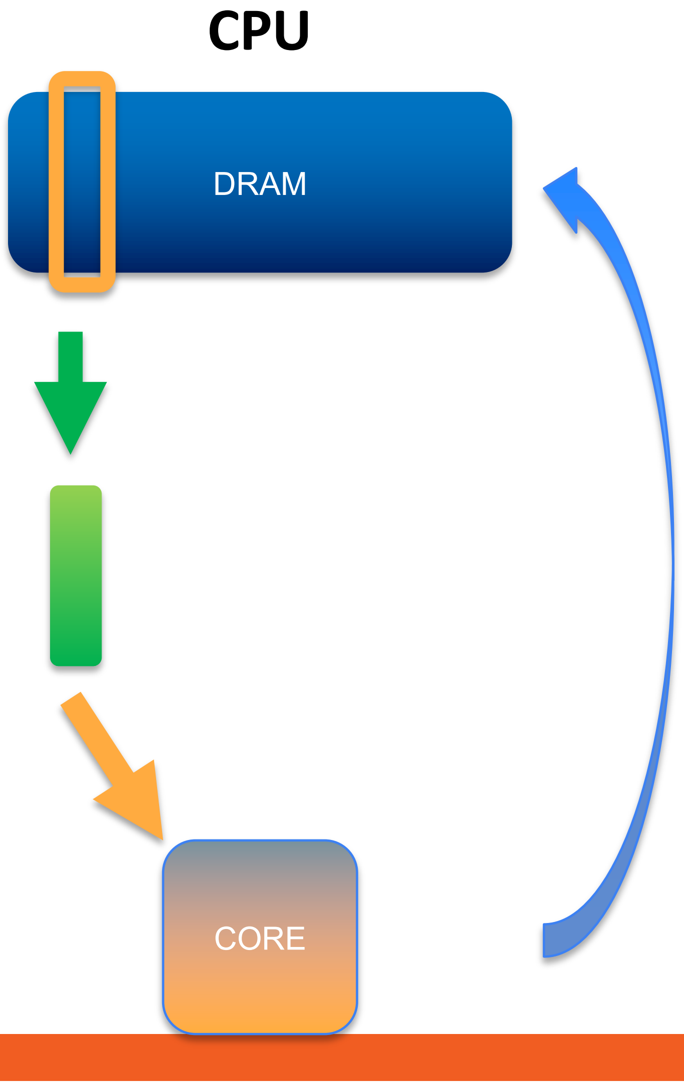
	#	Metric	Google TPU v3	Nvidia V100	Nvidia A100	Cerebras WSE	GraphCore IPU1	GraphCore IPU2
Raw Metrics	1	Technology node	>12nm (16 nm est.)	TSMC 12 nm	TSMC 7 nm	TSMC 16 nm	TSMC 16 nm	TSMC 7 nm
	2	Die Area (mm2)	<648 (600 est.)	815	826	46225	900 (est.)	823
	3	Transistor Count (B)	11 (est.)	21	54.2	1200	23.6	59.4
	4	Architecture	Systolic Array	SIMD + TC	SIMD + TC	MIMD	MIMD	MIMD
	5	Theoretical TFLOPS (16-bit mixed precision)	123	125	312	2500	125	250
	6	Freq (GHZ)	0.92	1.5	1.4	Unknown	1.6	Unknown
	7	DRAM Capacity (GB)	32	32	80	N/A	N/A	112
	8	DRAM BW (GB/sec)	900	900	2039	N/A	N/A	64 (est.)
	9	Total SRAM Capacity	32MB	36 MB (RF+L1+L2)	87 MB (RF+L1+L2)	18 GB	300 MB	900 MB
	10	SRAM BW (TB/sec)	Unknown	224 @RF + 14 @L1 + 3 @L2	608 @RF+ 19 @L1 + 7 @L2	9000	45	47.5
	11	Max TDP (Watts)	450	450	400	20K	150	150 (est.)
Efficiency M	12	GEMM Achievable TFLOPS	98%	88%	93%	Unknown	47%	61%
	13	Energy Efficiency (Achievable GEMM TFLOPS/Max Watts)	0.26	0.24	0.72	Unknown	0.39	1.0
	14	Theoretical Energy Efficiency (Theoretical TFLOPS/Max Watts)	0.27	0.27	0.78	0.125	0.83	1.6
	15	Memory Capacity (GB)	16	32	80	18	0.3	112
	16	Memory Efficiency (FLOP/DRAMByte)	133	122	158	N/A	N/A	Unknown
	17	Memory Efficiency	Unknown	32	35	Unknown	1.28	3.2
	18	Area Efficiency (Achievable TFLOPS/mm2)	0.2	0.13	0.35	Unknown	0.06	0.17
	19	Area Efficiency (Achievable TFLOPS/BTran)	11	5.2	5.3	Unknown	2.5	2.6

Tim Rogers and Mahmoud Khairy, An Academic's Attempt to Clear the Fog of the Machine Learning Accelerator War, ACM SIGARCH blog

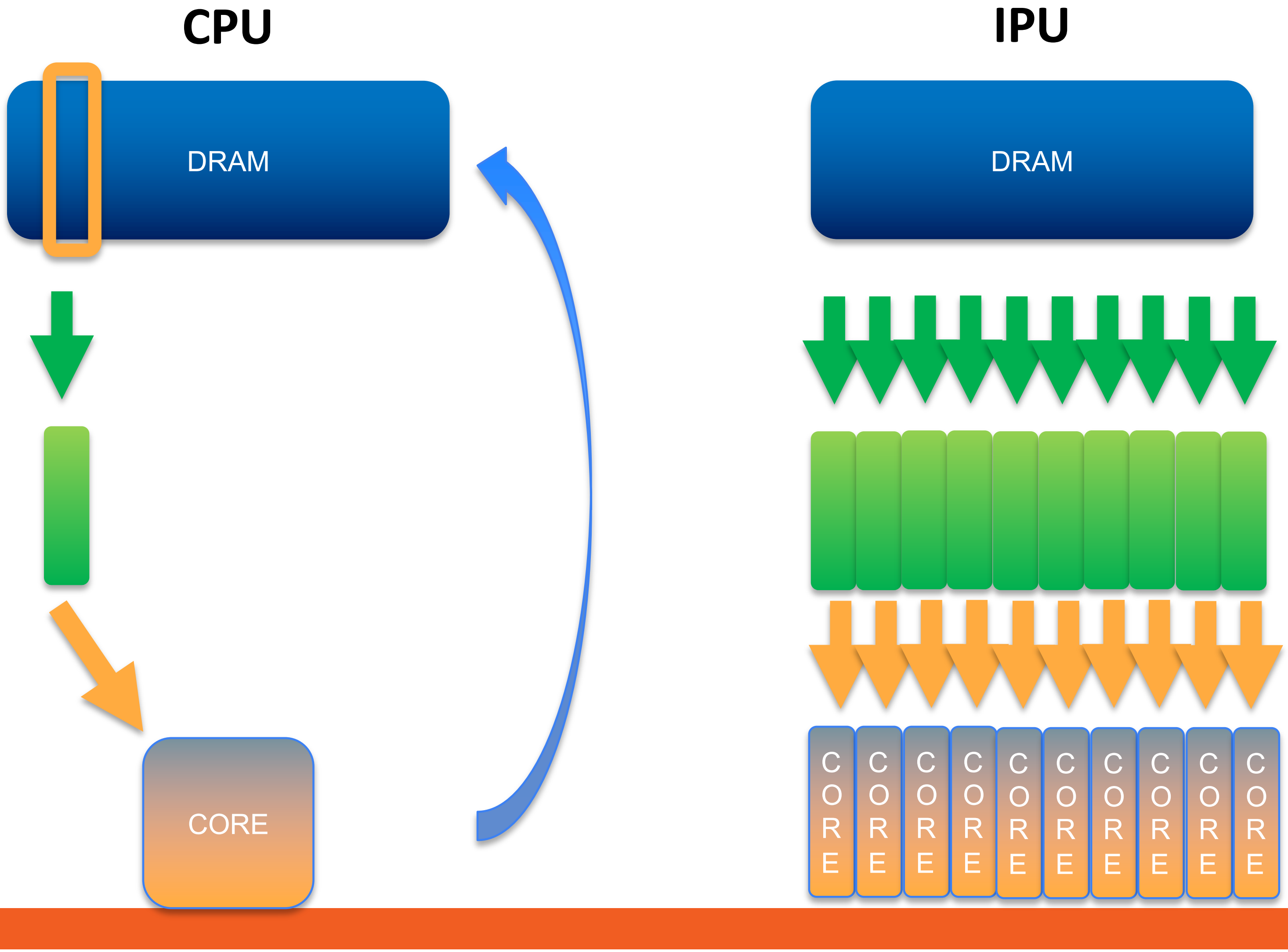
Next Step: Towards a Theory of Tile Centric Computing



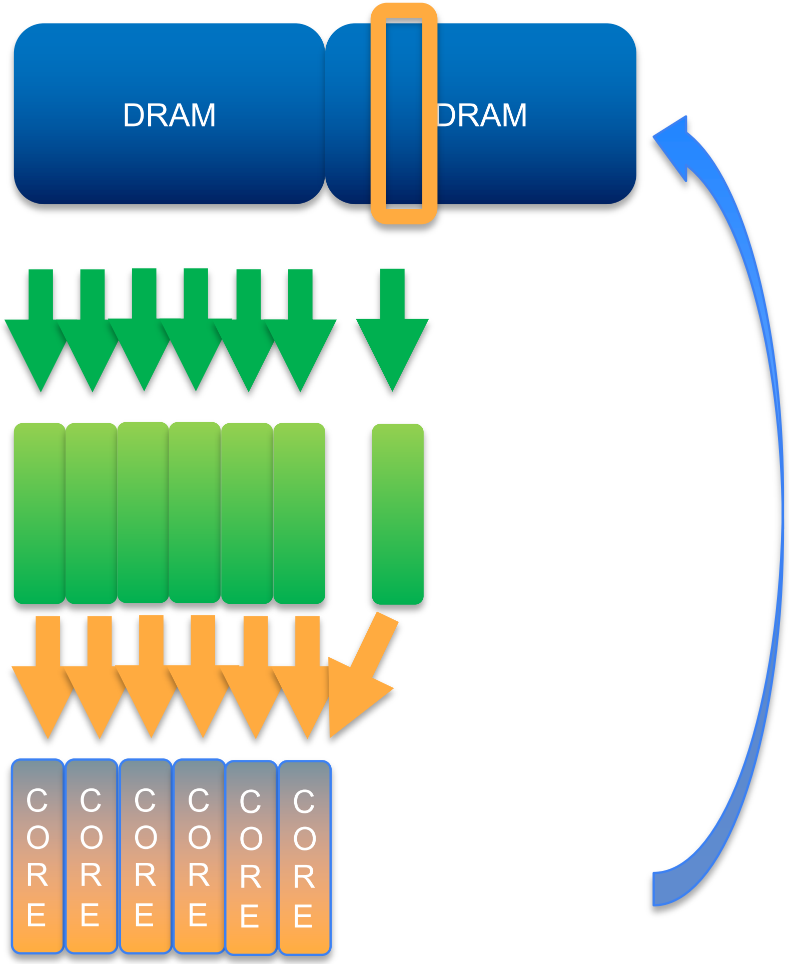
Traditional CPU: SRAM Computation Spreads over Time



ML Accelerators: SRAM Computation Spreads over Cores



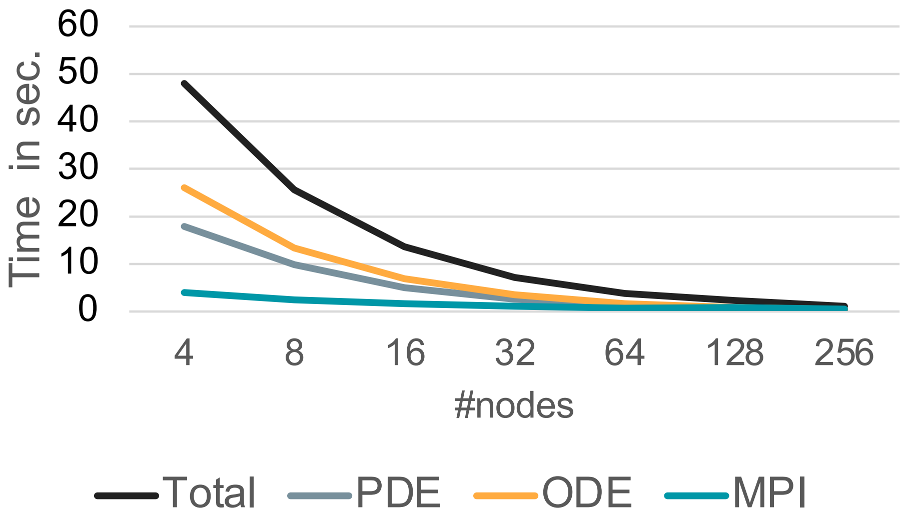
One Ideal Case: Data Streaming



- Best case: small persistent working set, large streamable data, lots of operations on data
- Typical example: NN with weights and training data
- Sequence alignment has this structure
- Not the case for most graph/ matrix algorithms

Ideal Case: Use lots of SRAM-based processors to exploit superlinear scaling

- Needs enough processors/SRAM to fit entire graph
- Needs enough prarallelism to run on lots of tiles (or use non-tile based architecture)
- Needs fast and ideally flexible network



Cardiac simulation on Oakbridge-CX (Cascade lake)

Speedups for 64x nodes:

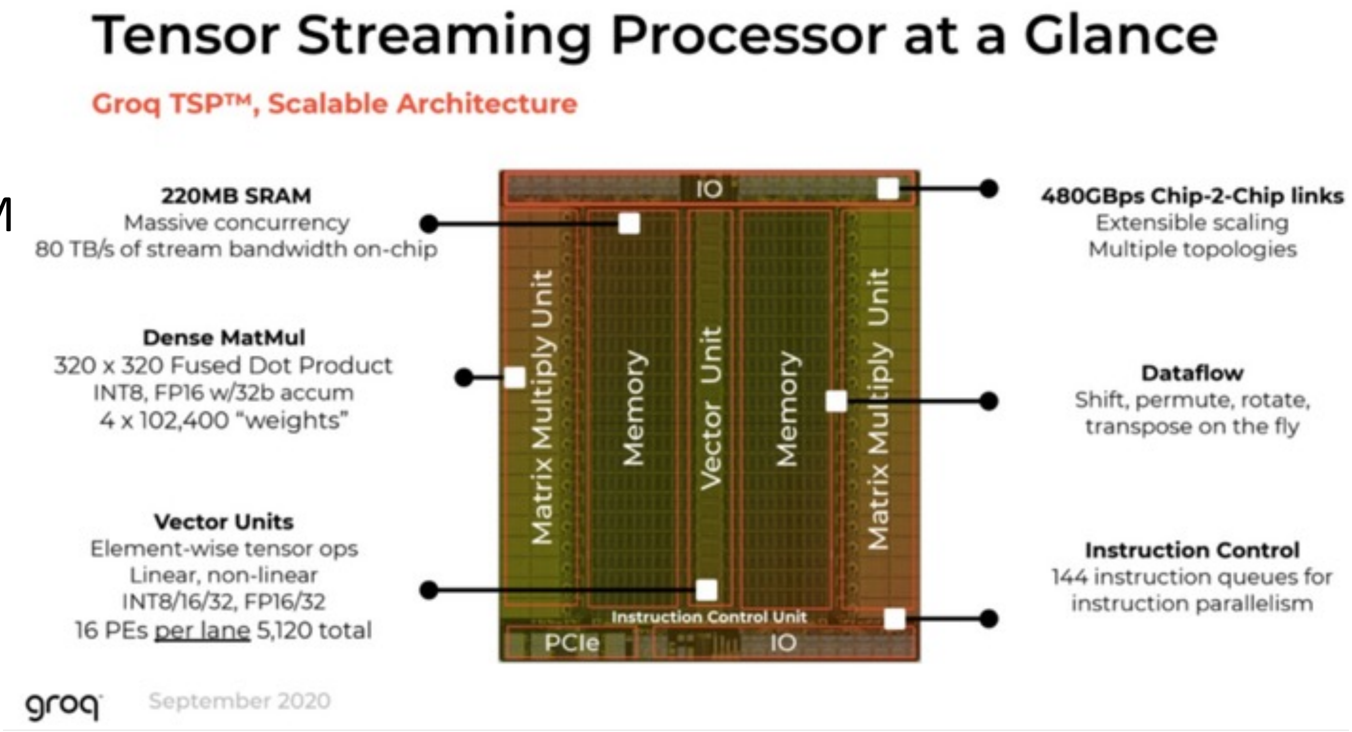
- PDE (SpMV) 128x
- ODE 60x
- MPI 8x
- Total 43x

Ideal Case: Use lots of SRAM-based processors to exploit superlinear scaling

- Should work well for high time, low space complexity algotithms (e.g. exact weighted matching)
- Unfortunately many such algorithms are not parallel
- Brute force approaches work, but this may not help time to solution
- Kernelizations are VERY attractive

Application of the Idea: LLM Inference with Groq

- Groq TSP is another SRAM-based processor
- Uses 572 TSPs to store entire LLM in SRAM
- Low latency
- Fastest system for LLM inference (Llama-3)
- Streaming architecture, not ideal for most graph algorithms



Tensor Streaming Processor at a Glance

Groq TSP™, Scalable Architecture

220MB SRAM
Massive concurrency
80 TB/s of stream bandwidth on-chip

Dense MatMul
320 x 320 Fused Dot Product
INT8, FP16 w/32b accum
4 x 102,400 "weights"

Vector Units
Element-wise tensor ops
Linear, non-linear
INT8/16/32, FP16/32
16 PEs per lane 5,120 total

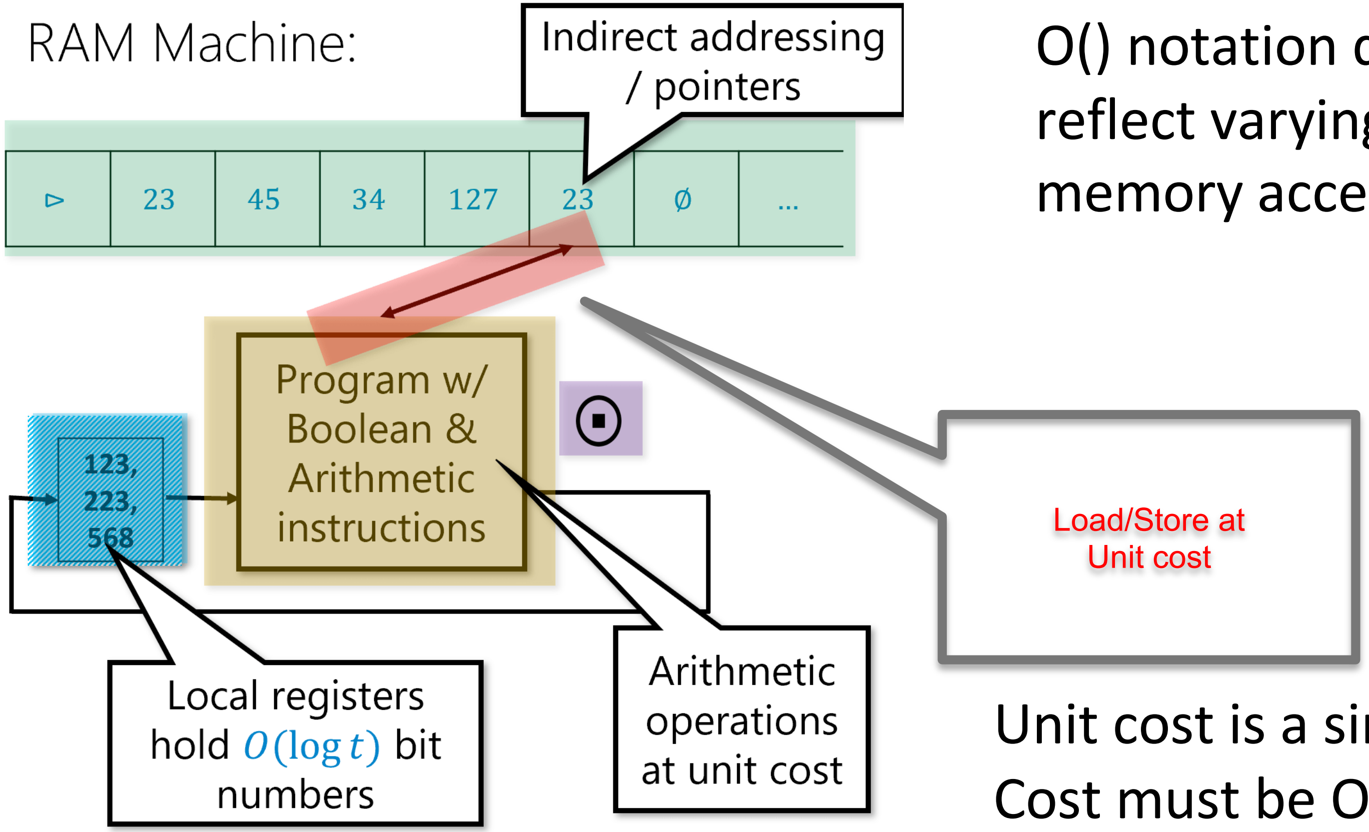
480GBps Chip-2-Chip links
Extensible scaling
Multiple topologies

Dataflow
Shift, permute, rotate, transpose on the fly

Instruction Control
144 instruction queues for instruction parallelism

groq September 2020

Basic Model of Computation: RAM Machine


RAM Machine:
Indirect addressing / pointers
▻
23
45
34
127
23
∅
...
123,
223,
568
Program w/ Boolean & Arithmetic instructions
⊡
Load/Store at Unit cost
Local registers hold $O(\log t)$ bit numbers
Arithmetic operations at unit cost

O() notation does not reflect varying cost of memory accesses.

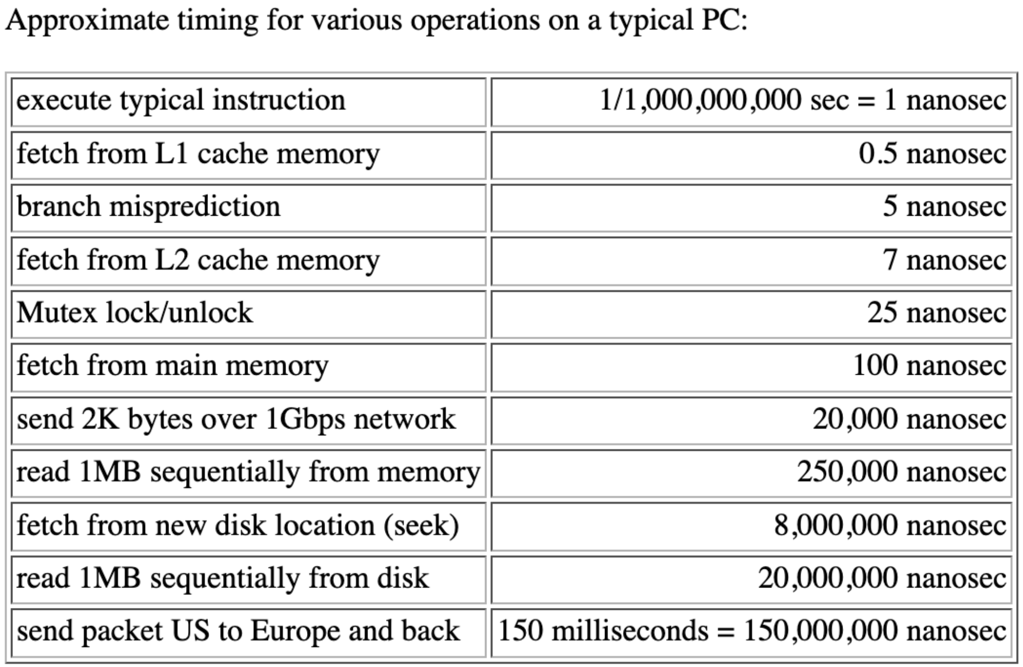
Unit cost is a simplification

Cost must be O($n^{1/3}$)

Note that Turing machines do not make this simplification.

Boaz Barak: Introduction to Theoretical Computer Science

Data Movement Cost depends on Data Size



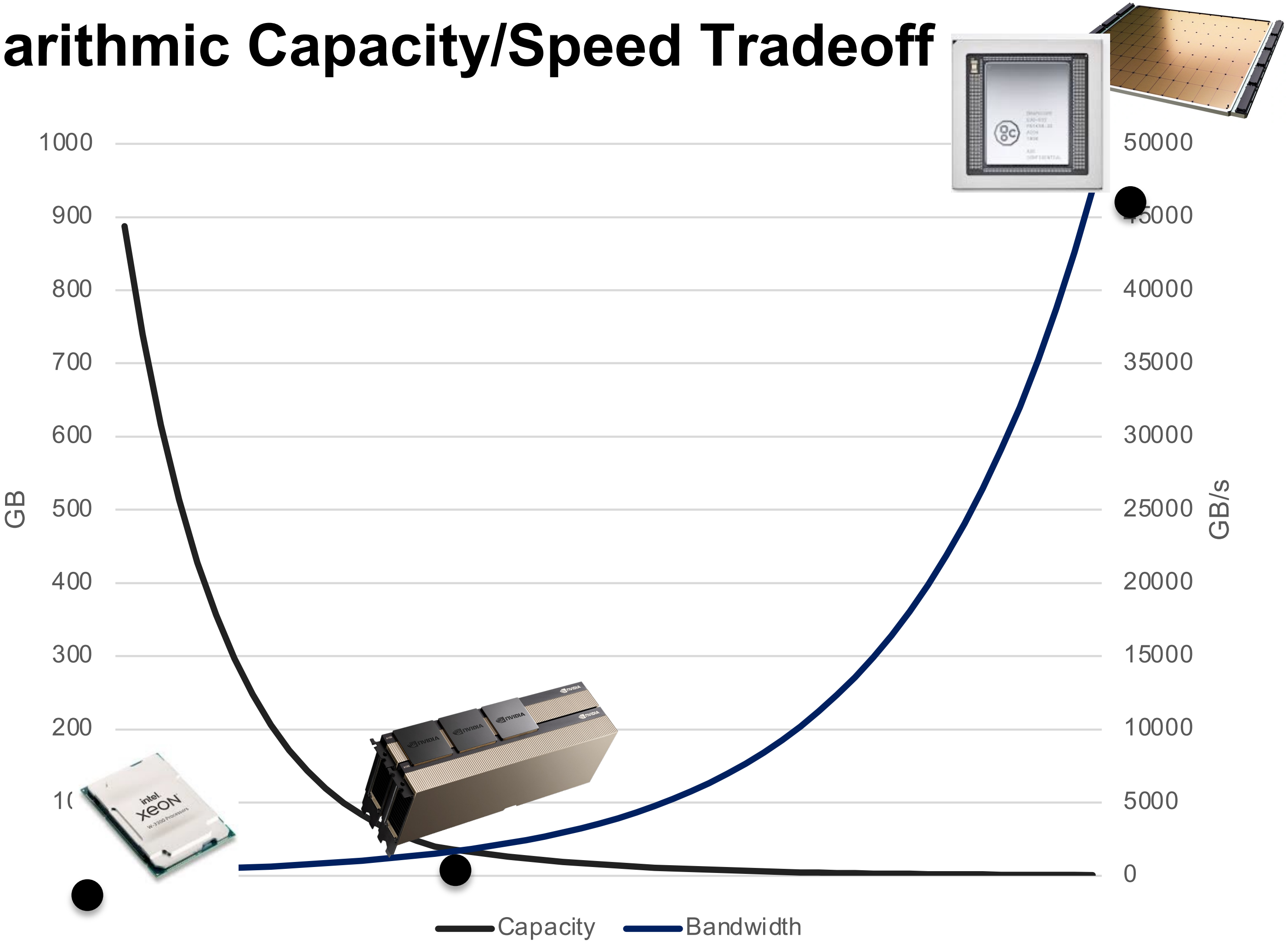
Approximate timing for various operations on a typical PC:

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

- All efficient memory technologies lie on a pareto curve of the ratio between bandwith/latency and size/cost
- Technologies not on the curve are not efficient
- This is a law

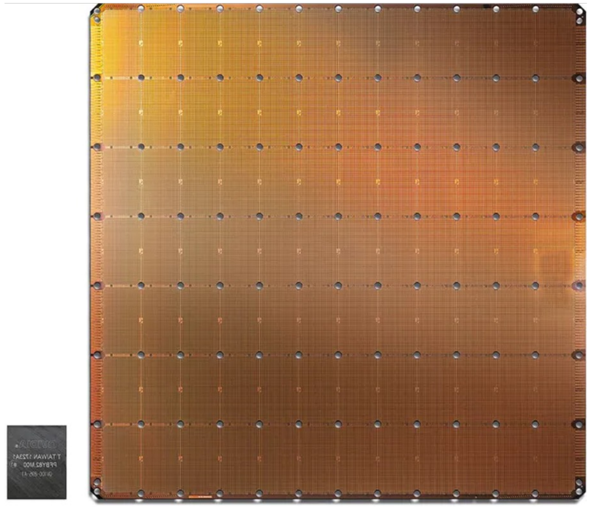
Latency Numbers Every Programmer Should Know by Peter Norvig

Logarithmic Capacity/Speed Tradeoff



Space efficient algorithms are inherently faster (if you can select optimal hardware)

How do these lessons apply to other ML Accelerators ?



Graphcore GC 200 IPU

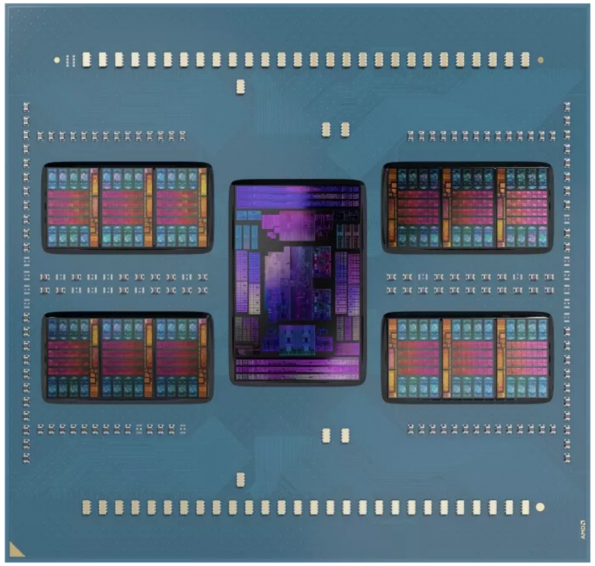
- Great SpMV
- Slow ODE
- Good intra-device comm
- Inter-device a challenge
- MPI-inspired strategy works

Cerebras WSE-2

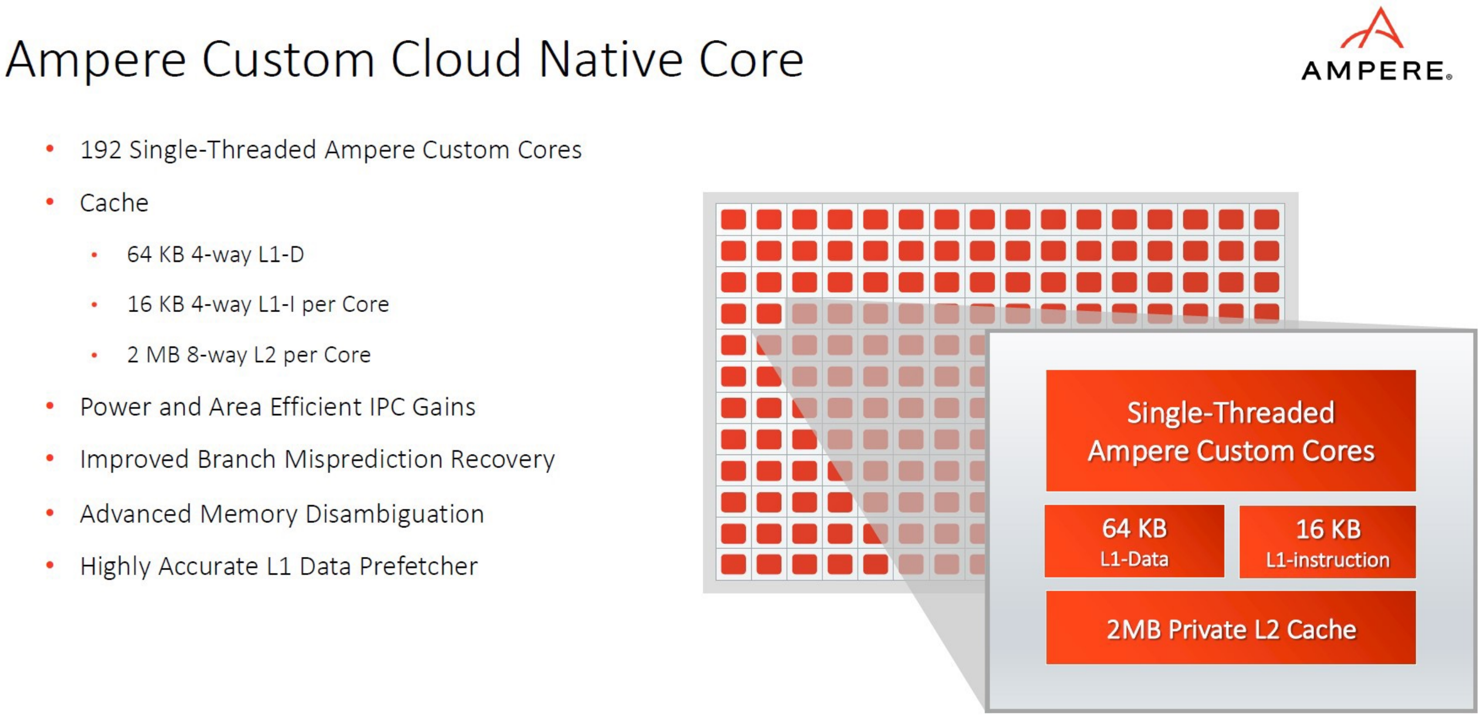
- Great SpMV
- Pretty good ODE
- Intra-device comm a challenge
- Inter-device not better
- Needs completely new algorithms

Tiles are coming, One Way or Another

With 135 million transistors, you can get a lot of cache…
But the internal structure becomes more and more tile based



AMD 9684X CPU has 1.1 GB of L3 cache, but a lot of NUMA effects on chip



Ampere Custom Cloud Native Core

AMPERE

- 192 Single-Threaded Ampere Custom Cores
- Cache
  - 64 KB 4-way L1-D
  - 16 KB 4-way L1-I per Core
  - 2 MB 8-way L2 per Core
- Power and Area Efficient IPC Gains
- Improved Branch Misprediction Recovery
- Advanced Memory Disambiguation
- Highly Accurate L1 Data Prefetcher

The Tile-based Graph Accelerator

What we would like...

- Large SRAM
- Some FLOPs
- No matrix or vector units
- Fine grained communication
- No global synchronization
- Nonlocal topology

- And a high-level description of algorithms....