Scaling Throughput Processors for Machine Intelligence

ScaledML Stanford 24-Mar-18

GRAPHCORE

simon@graphcore.ai

“MI”

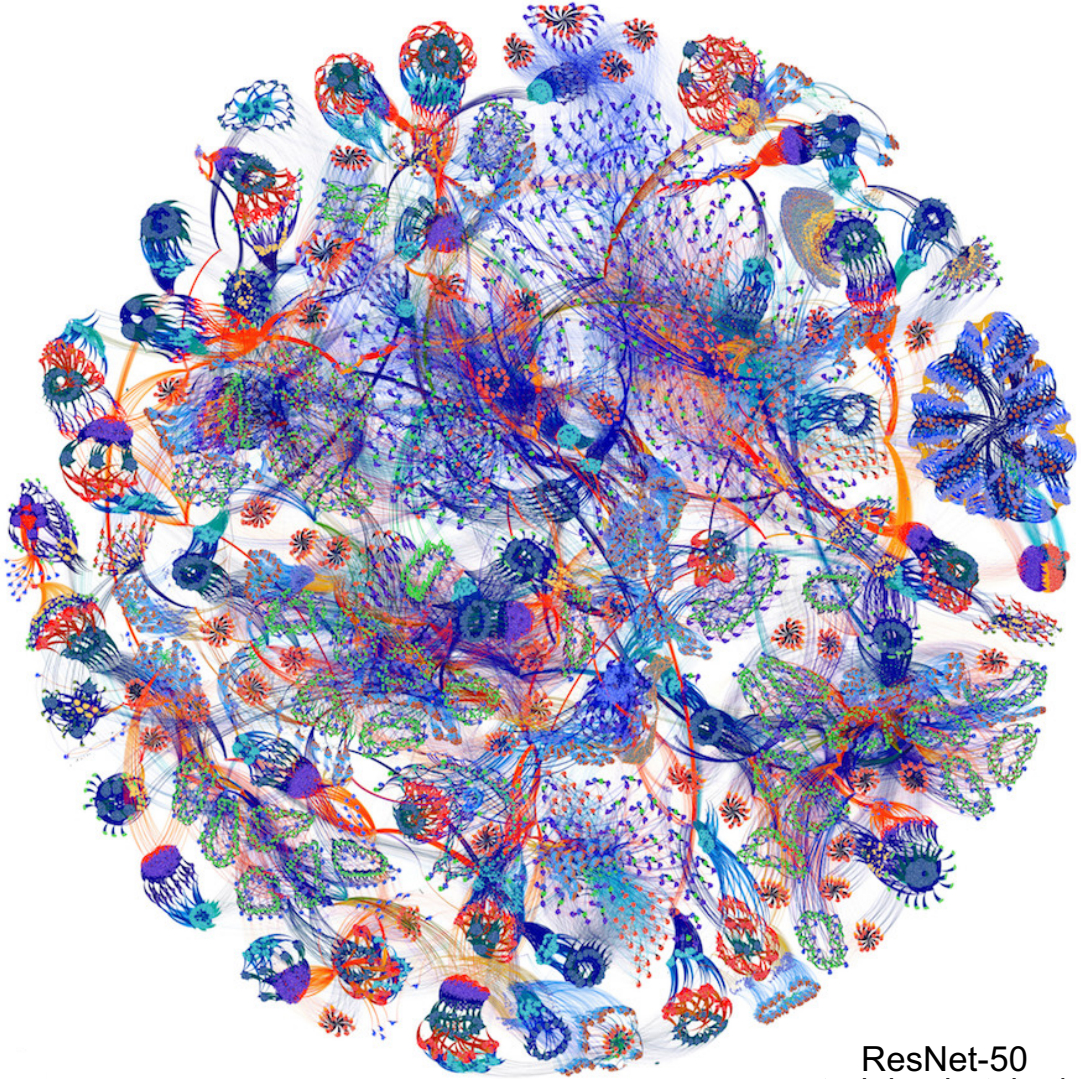
The impact on humanity of harnessing machine intelligence will be greater than the impact of harnessing machine power.

This is the strongest driver to re-invent computing since the invention of computing. Programming & software & hardware will all change.

There are 3 parts (so far) to “intelligence compute":

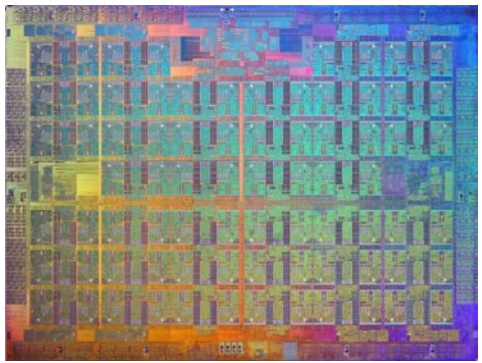
- simulating environments
- exploring model structures
- optimizing candidate models

The fundamental and challenging new data-type is the graph, not really the tensor calculations at its vertices.



ResNet-50
training batch=4

Processor are built for specific workloads



CPU

Scalar

Designed for office apps

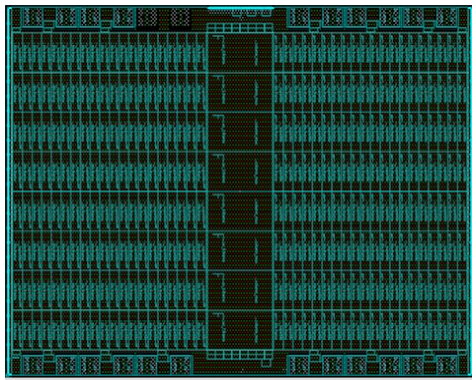
Evolved for web servers

GPU

Vector

Designed for graphics

Evolved for linear algebra



IPU

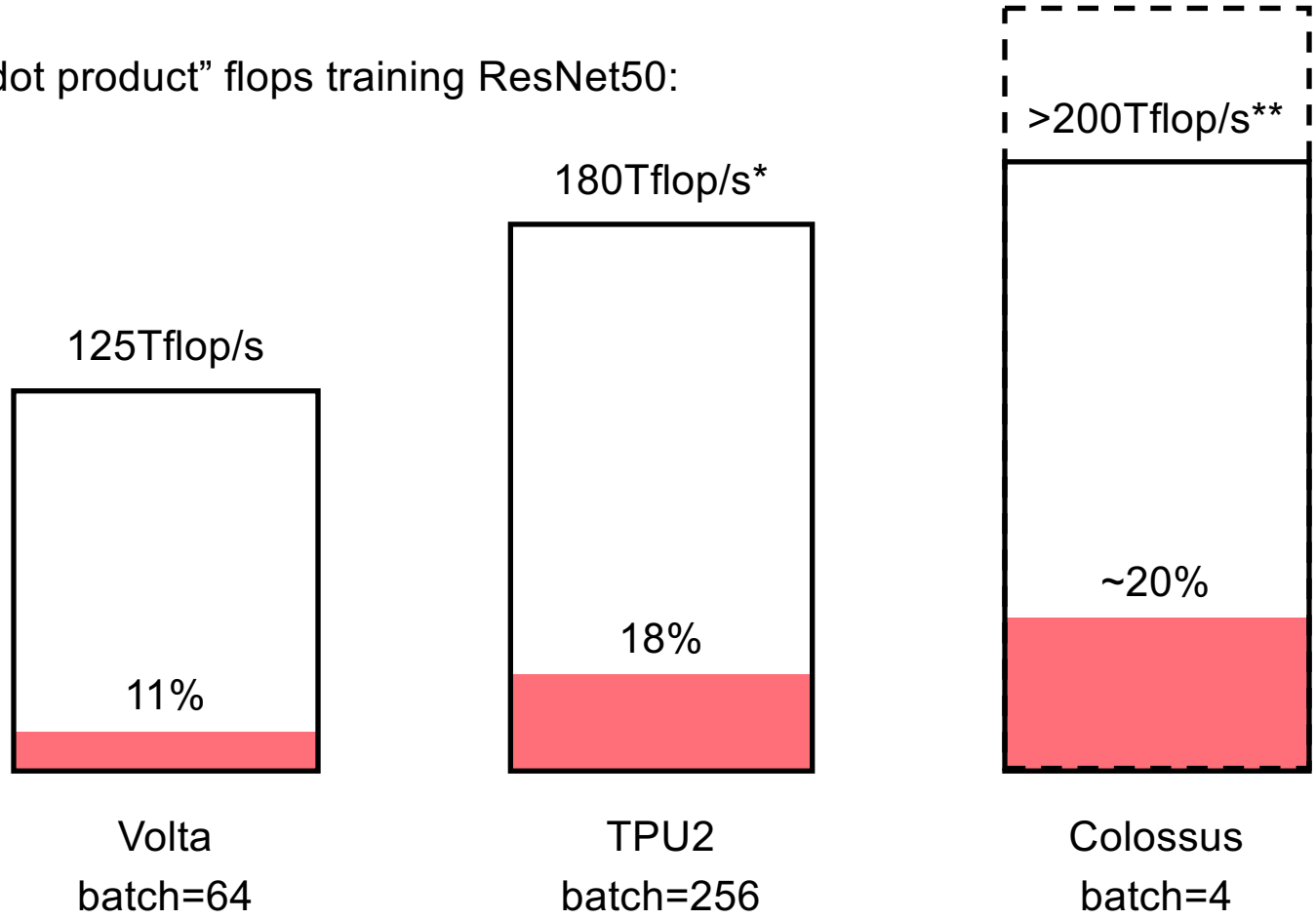
Graph

Designed for intelligence

We've all tried to squeeze the pips out of tensor algebra

Headline vs delivered "dot product" flops training ResNet50:

Amdahl's Law is tough!



*TPU2 card certainly burns more power than V100 and Colossus.

**Colossus peak tba.

Our comprehension and mechanization of intelligence is nascent

- A new class of workload – approximate computing on probability distributions learned from data.
- SOTA models and algorithms change every year.
- Huge compute demand for model optimization, more for exploration.

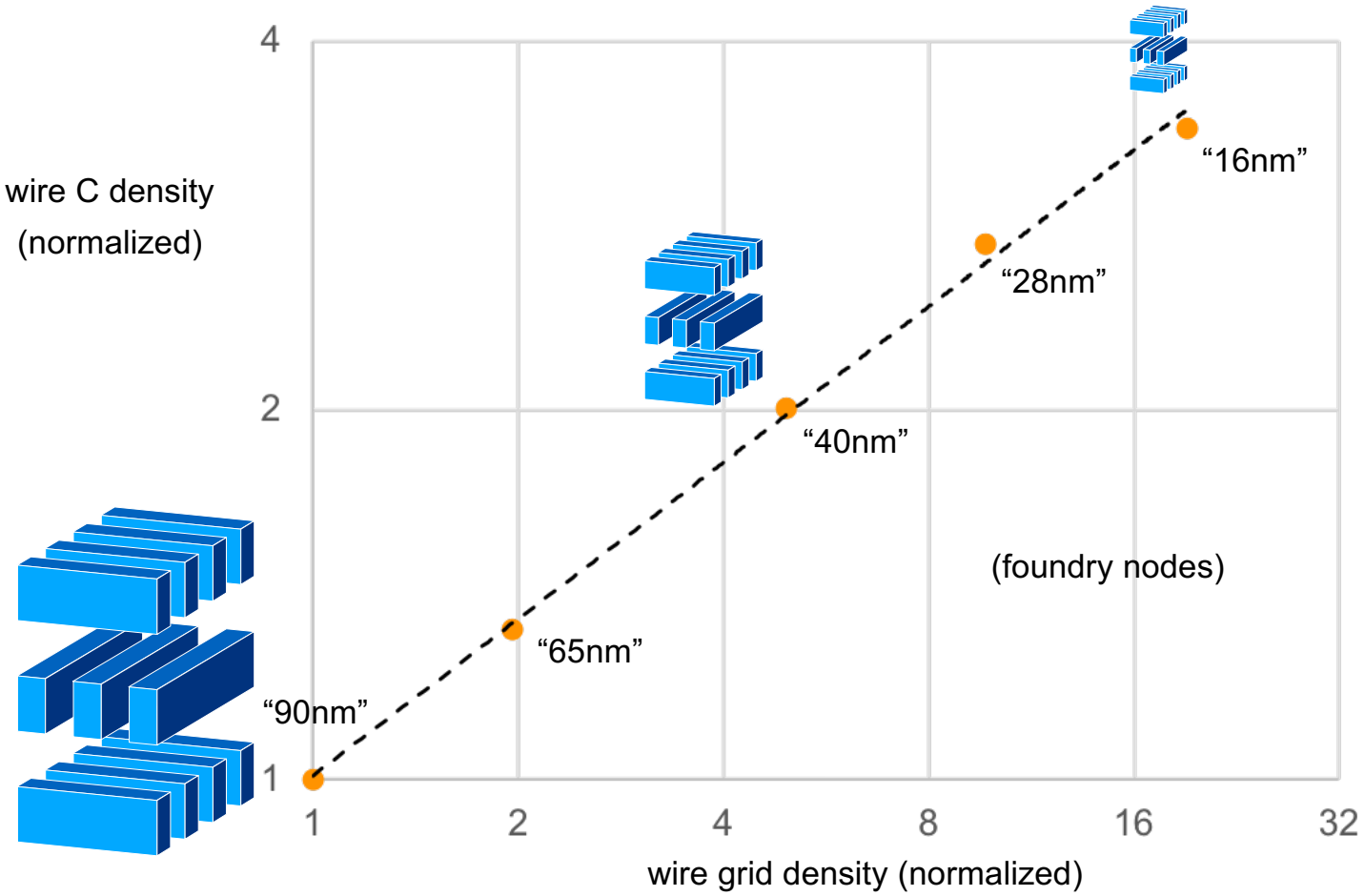
Until we know more, we need computing machines which...

- exploit massive parallelism;
- are broadly agnostic to model structure
  ...but can assume sparsity, stochasticity, and physical invariances in space & time;
- have a simple programming abstraction;
- are efficient for exploring, training and deployed inference.

Say we want 1000x throughput performance in the next decade. How much will come “for free” from silicon scaling?

The era of “amazing” silicon scaling ended around 2005 (90nm). All throughput processors are now power limited.

Wire C density as a proxy for total logic C density:



Each process node:

~2x T density

~√2x C density

$P \sim CV^2f$

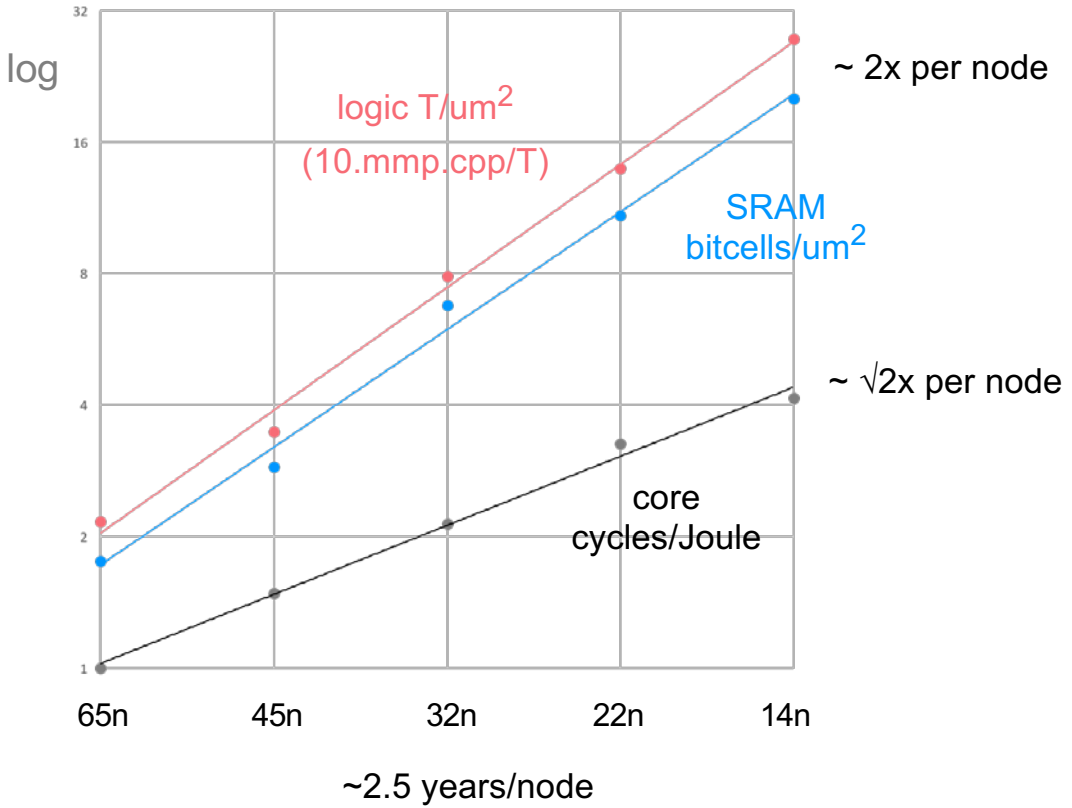
Best case: reduce V by ~20% per node to deliver 2x compute at constant power – this is tough.

Xeon servers are not delivering that best case:

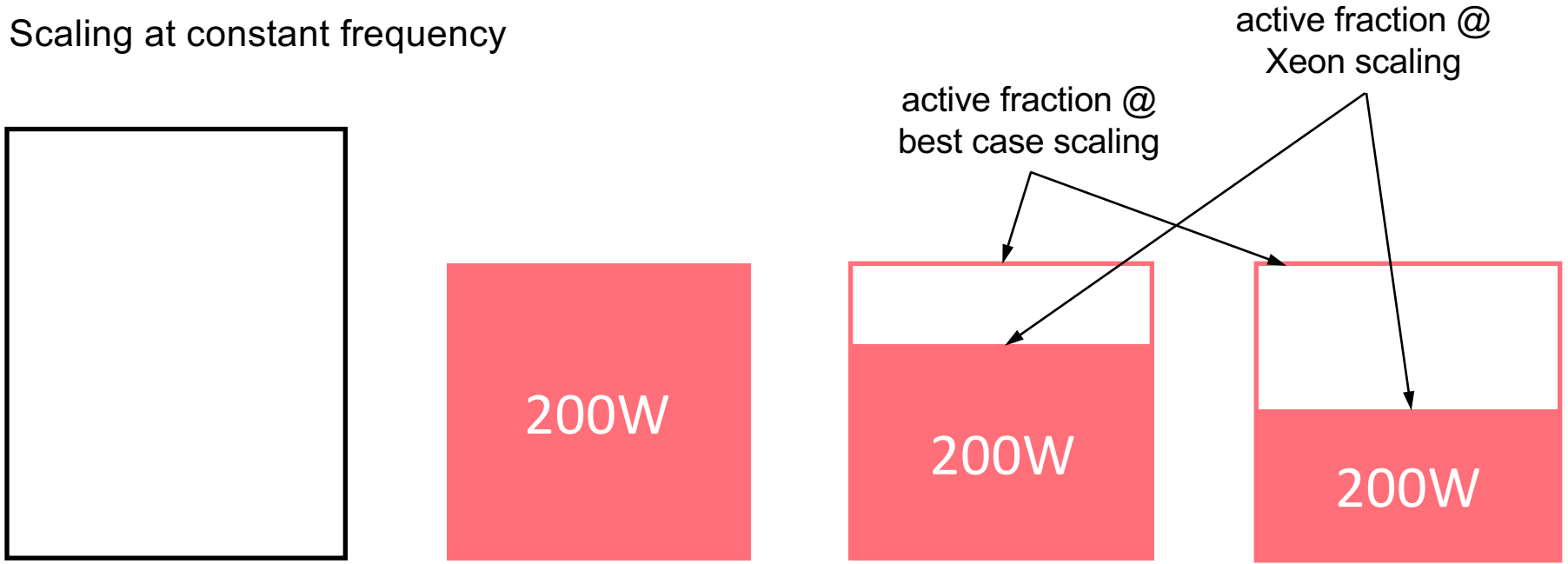
- transistors / chip ~30% / year
- throughput / Watt ~15% / year

Big Xeon exemplars (plotting geomean for each node):

65n	Woodcrest X5160 2c Conroe X3085 2c
45n	Lynnfield X3470 4c Nehalem X7550 8c
32n	Westmere E7-8870 10c SandyBridge E5-2690 8c
22n	IvyBridge E7-2890v2 15c Haswell E7-8890v3 18c
14n	Broadwell E7-8894v4 24c Skylake 8180P 28c



Scaling at constant frequency



largest manufacturable die ~825mm^2

node N
T transistors

node N+1
2T transistors

node N+2
4T transistors

- Silicon process scaling will probably yield 3-10x throughput in the next decade.
- If we want another 100x it must come from connecting chips together, and from new architecture.
- Not enough will come from new hardware architecture without software and algorithm co-innovation.

Architecture drivers

Silicon efficiency is now the full (productive) use of available power

- Memory on chip consumes little power, is more useful than logic which can't be powered!
- Keep data local, ie. distribute the memory.
- Serialise communication and compute.
- Cluster multiple chips to emulate a bigger chip with more power.

There is a poor performance return on single processor complexity

(Pollack's Rule: performance ~ $\sqrt{}$ transistor count)

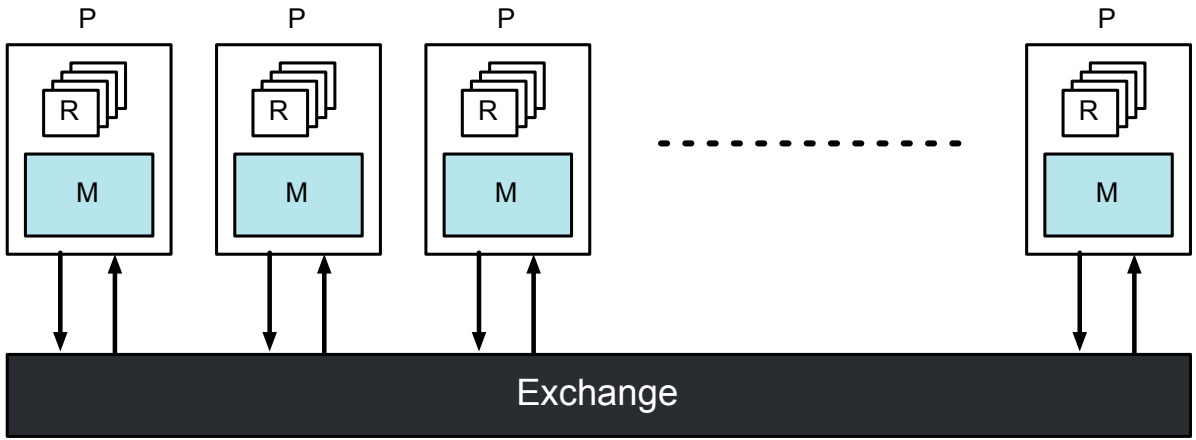
- To maximize throughput performance, use many simple processors.

Parallel programs are hard to get right, hard to map to massively parallel machines

- BSP is a simple abstraction, guaranteed free of concurrency hazards.
- Compile communication patterns, like you compile functions.

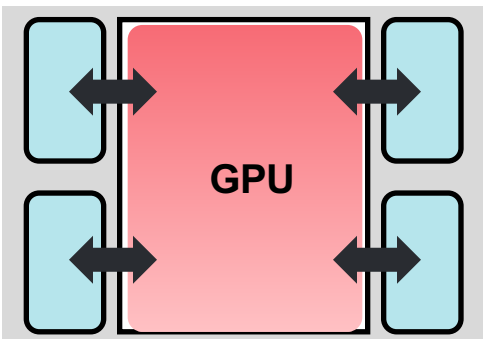
Pure distributed machine with compiled communication

- Static partitioning of work and memory
- Threads hide only local latencies (arithmetic, memory, branch)
- Deterministic communication over a stateless “exchange”



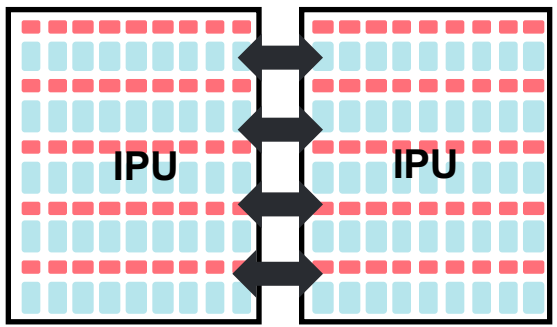
Two different plans

GPU + DRAMs
on interposer



16GB @ 900GB/s

IPU pair with
distributed SRAM

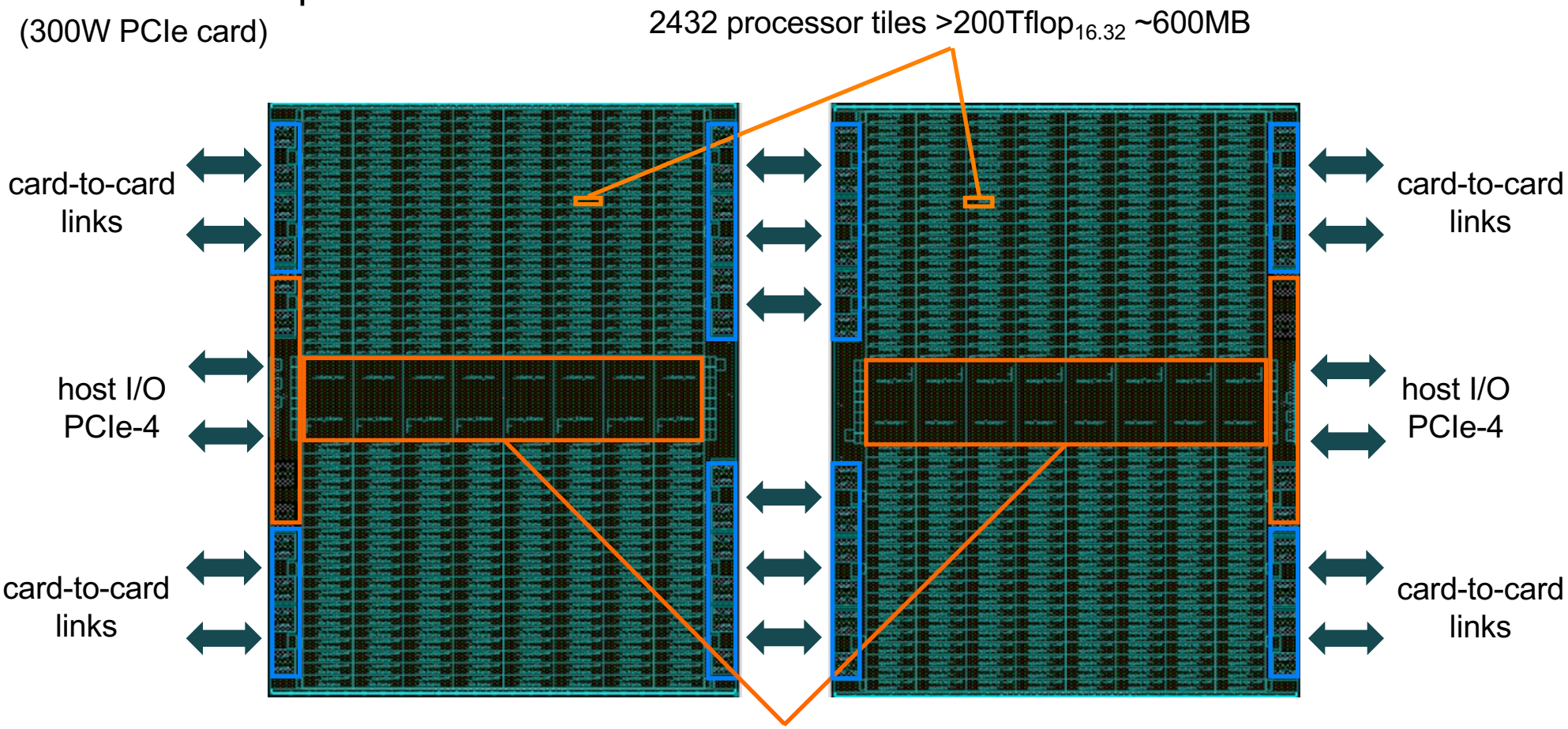


600MB @ 90TB/s, zero latency

Same total power, similar active logic in both cases

“Colossus” IPU pair

(300W PCIe card)

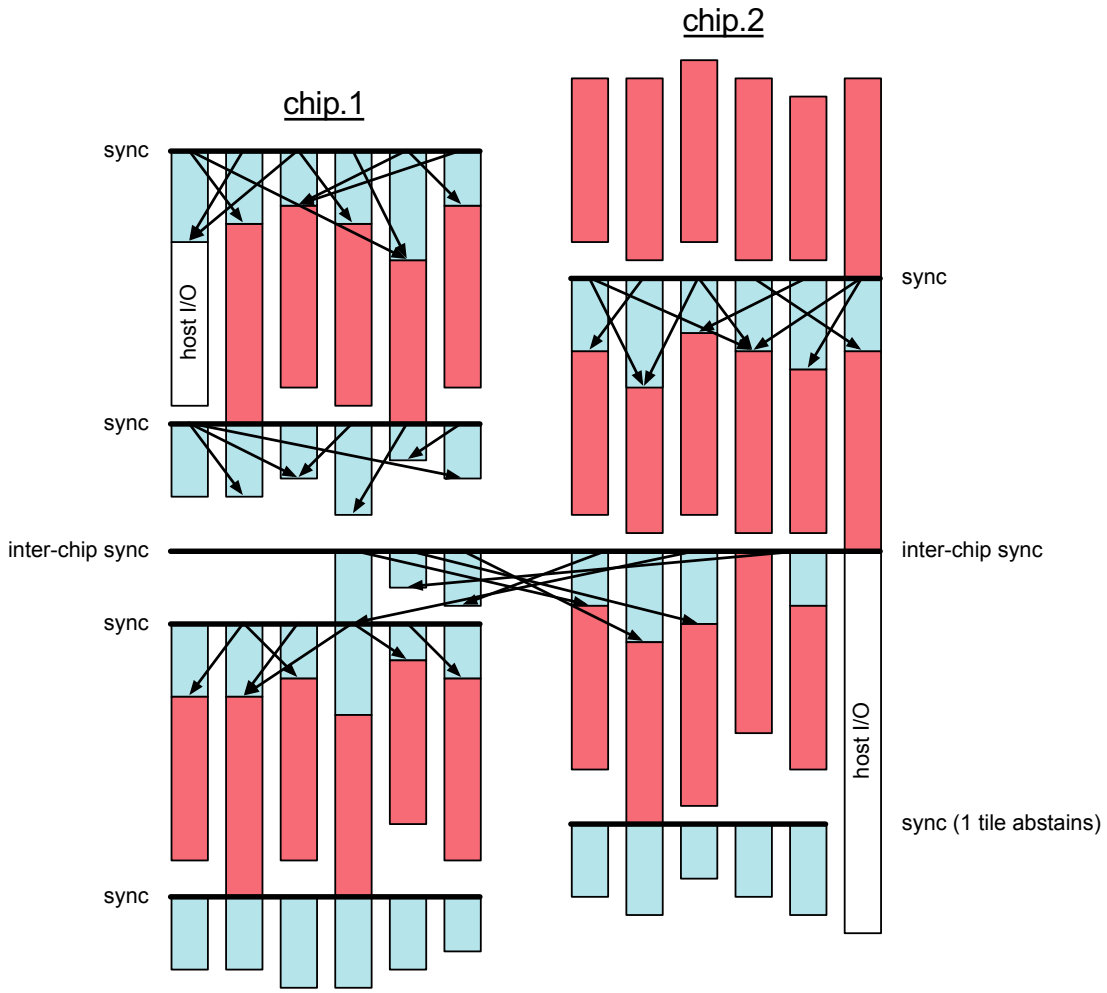


Bulk Synchronous Parallel (BSP)

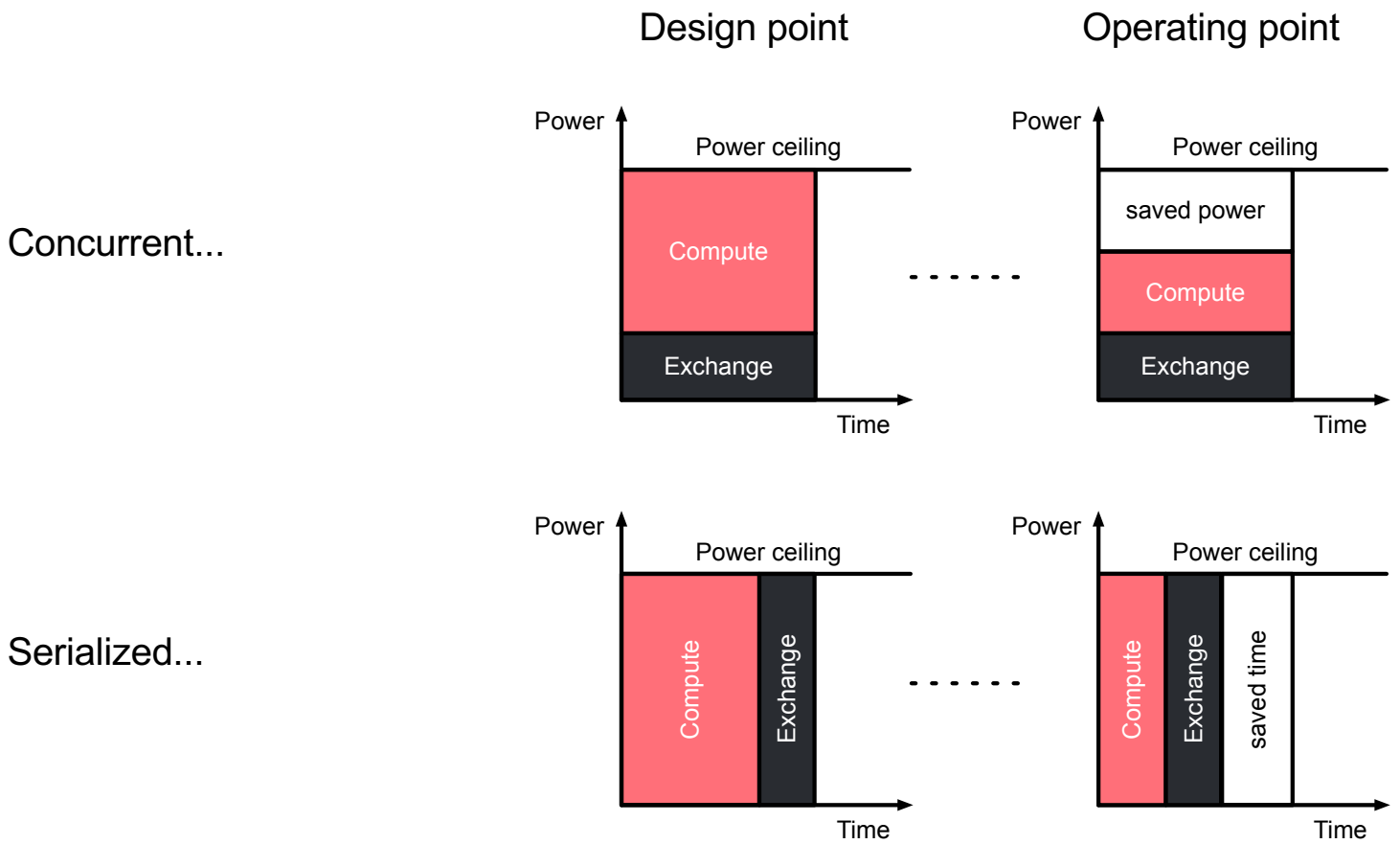
Massively parallel computing
with no concurrency hazards

exchange phase

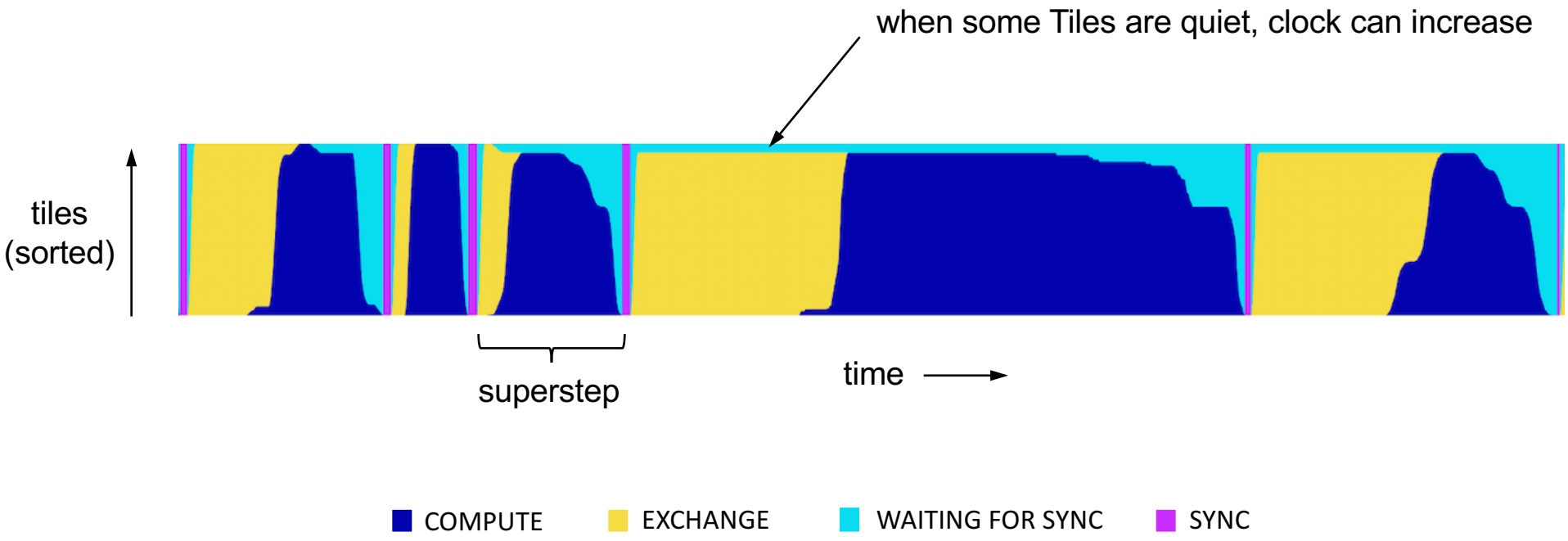
compute phase



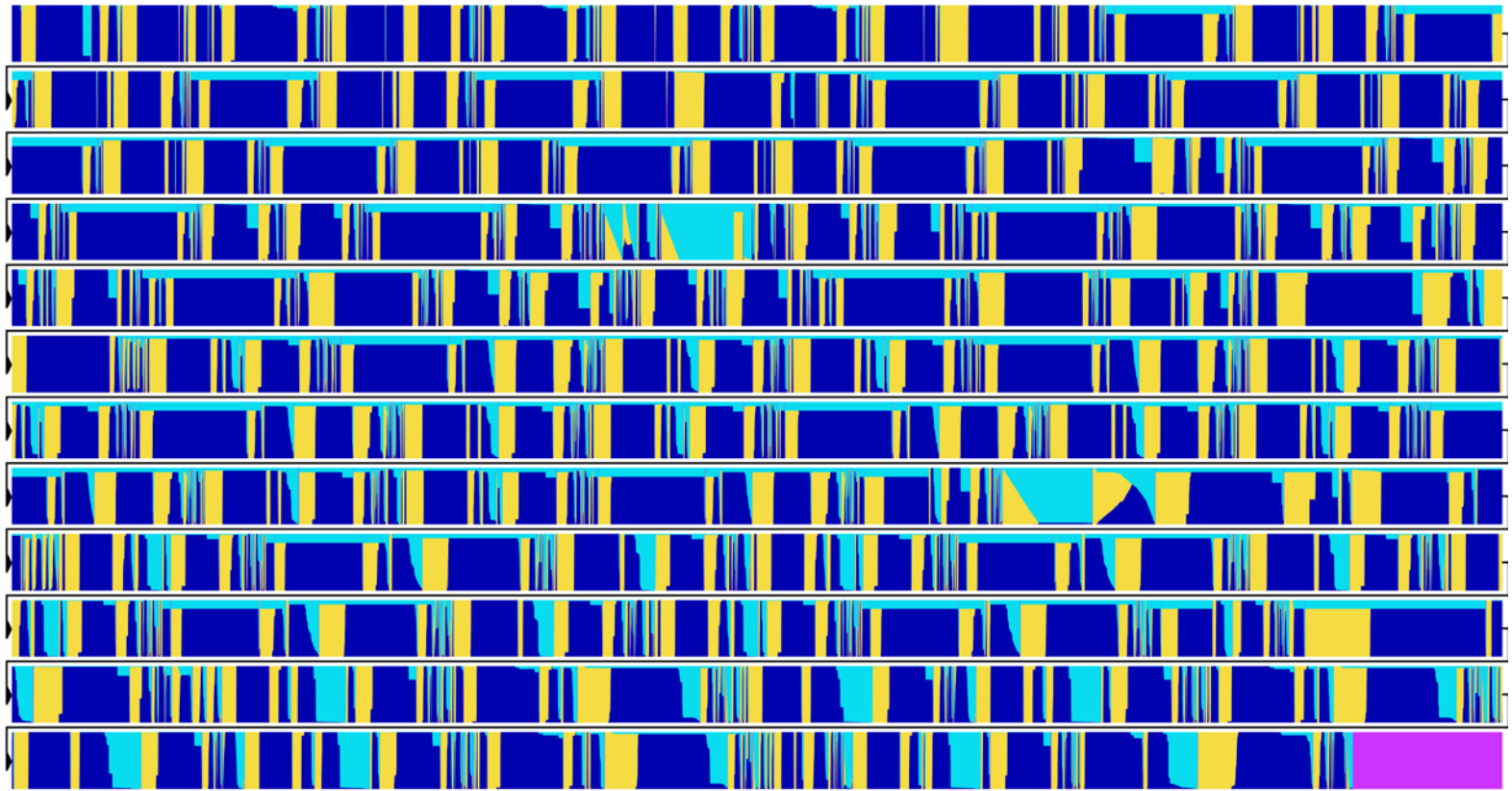
Serializing compute and communication maximizes power-limited throughput



BSP Execution Trace



BSP Trace: ResNet-50 training, batch=4



Colossus integrates enough memory to hold a large model on chip, or on a small cluster of chips. In GPU terms “all kernels are then fused”.

Access to the on-chip model at spectacular bandwidth and zero latency yields spectacular throughput for the many models which are memory-bottlenecked on GPUs.

But what about training CNNs with batch norm using large batches?

Small still batches learn better models, more reliably

Preview of work by Dominic Masters and Carlo Luschi, coming to arXiv soon.

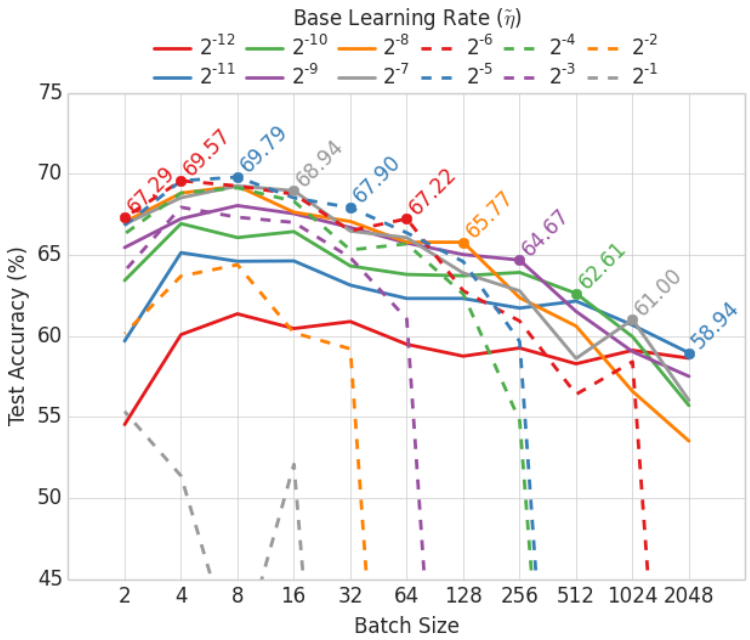
Sweeping batch size m and "base learning rate" $\tilde{\eta} = \eta/m$, where the weight update equation is:

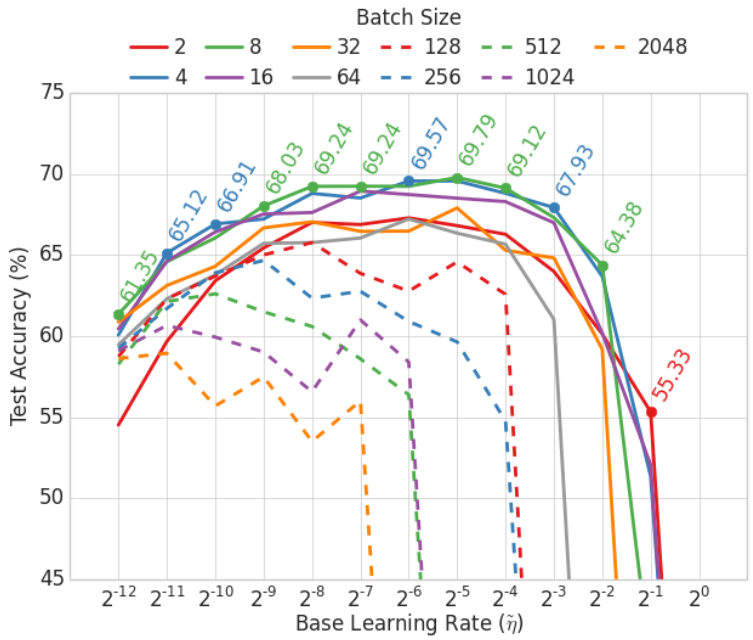
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \tilde{\eta} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}_k)$$

...so all runs do the same amount of computation, larger batches update the gradients less often.

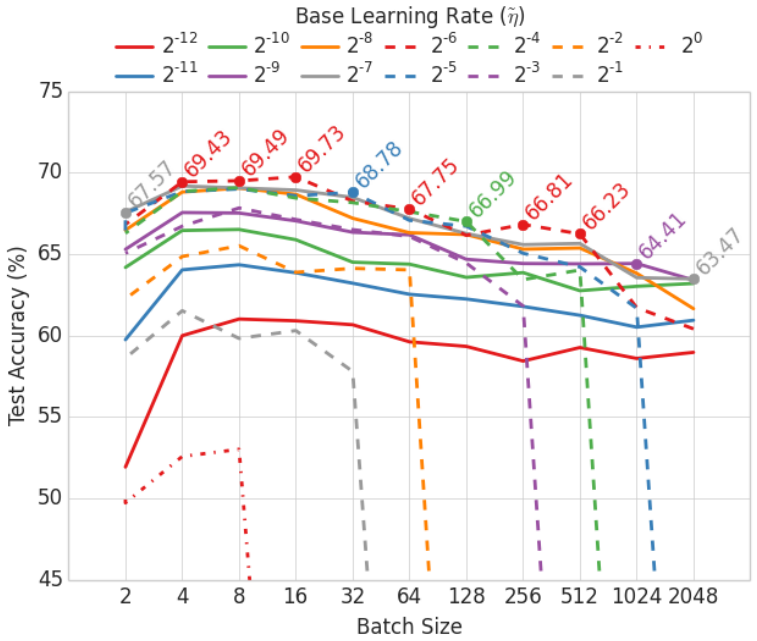
This example is training ResNet-32 on CIFAR100, with batch norm over the full batch. Many other examples in the upcoming paper.

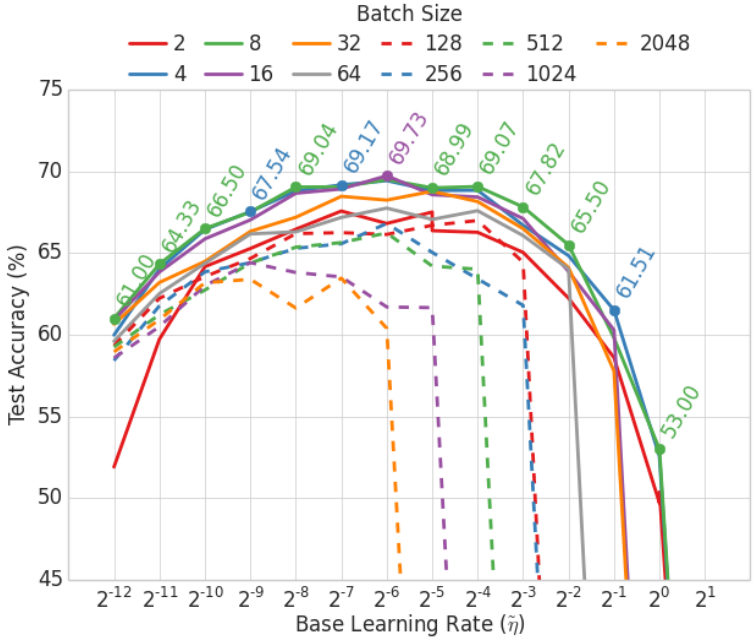
Small batches learn better models, more reliably





For this example, batch size ~8 gives the best models, with the least sensitivity to chosen learning rate.





The warm-up strategy of Goyal *et al* (arXiv:1706.02677) helps larger batches but doesn't level the field.

Best batch sizes 4-16, eg. distribute over 8 chips at m=2 per chip.

SGD with small batches yields better models.

Machines which are efficient for very small (sub-) batches allow parallel learning over more machines.

We also need to embrace other types of memory-efficient algorithm:

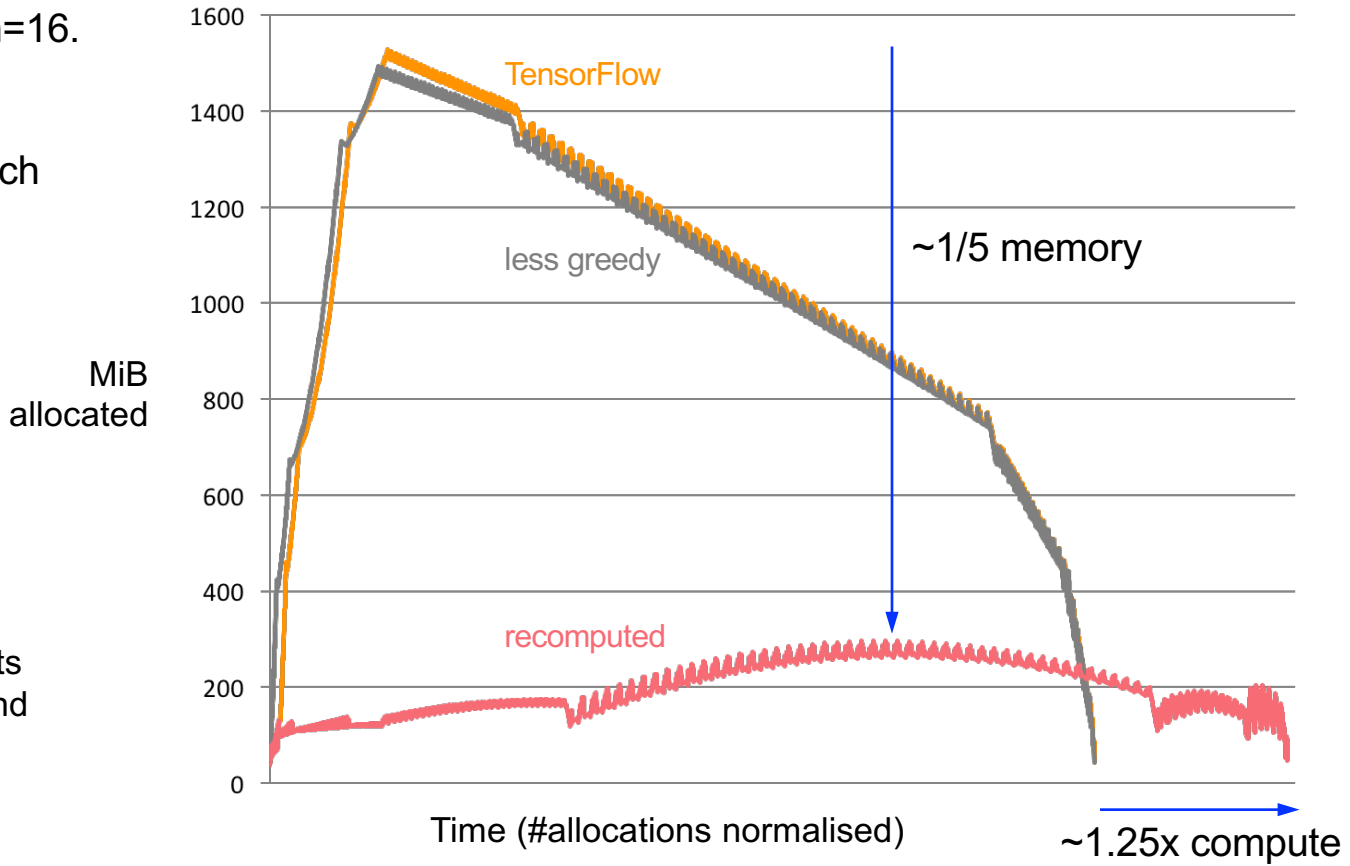
- Trading re-computation for memory.
- Reversible models.

Trading re-computation for memory

DenseNet-201 training, batch=16.

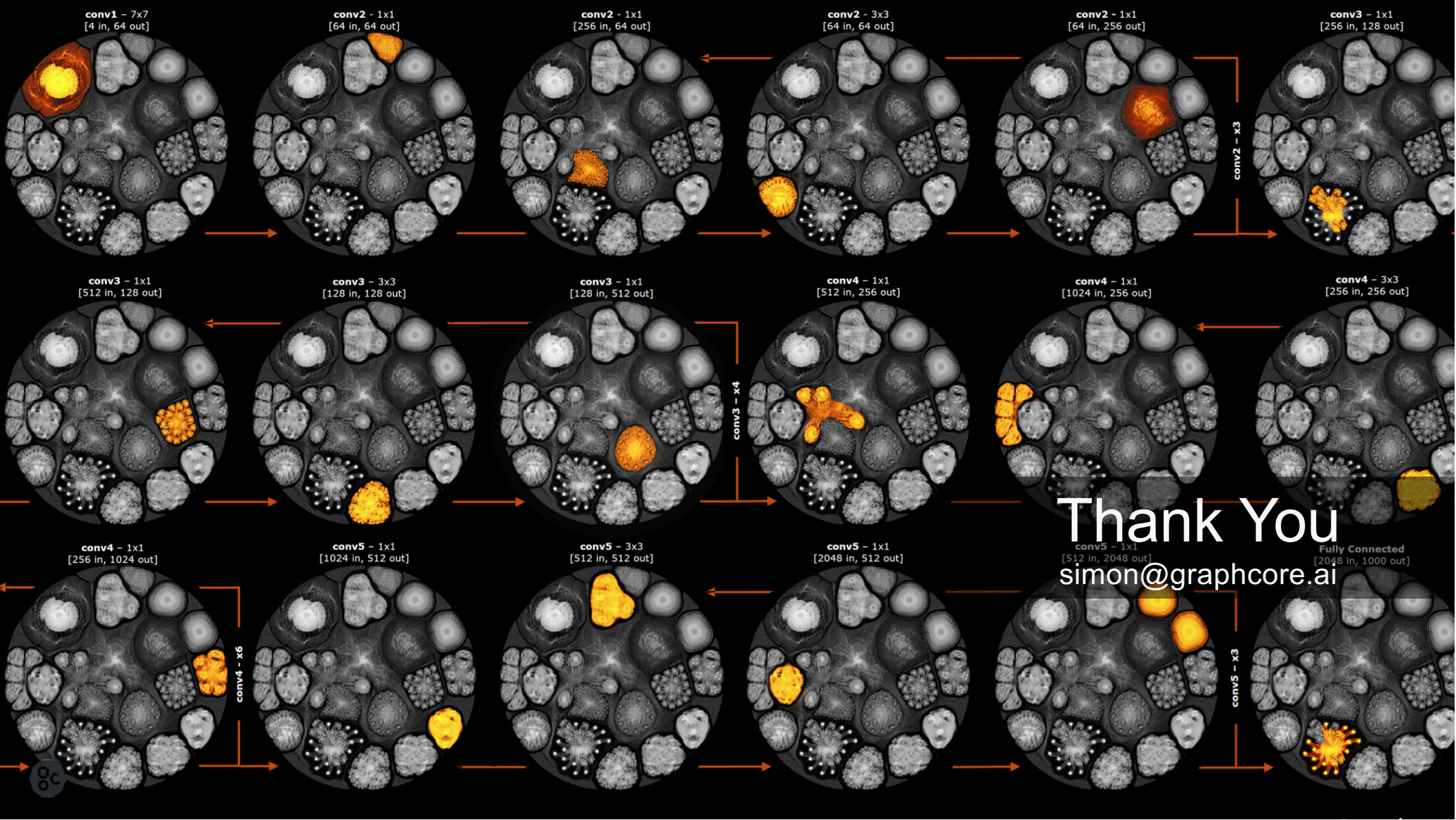
Naive strategy: memorize activations only at input of each residue block, re-compute all others in backward pass.

TF executing on CPU, recording total memory allocated for weights + activations. Float16 weights and activations, single weight copy.



Re-cap

- Throughput processors will be increasingly power limited, so will become mostly memory.
- We can now integrate enough SRAM to hold large models entirely on chip.
- IPUs don’t need large batches for efficiency. Small batches deliver better models, free most of the memory for params, and allow parallel learning over more machines.
- Efficient massively parallel processors need to compile communication patterns as well as compute functions. So program graphs must be pseudo-static.
- BSP is a simple, concurrency-safe, and power-efficient paradigm for massively parallel MI computing.



Thank You

simon@graphcore.ai