

EFFICIENT AI COMPUTE PLATFORM

REAL TIME CONTROL FOR ADAPTIVE OPTICS (RTC4AO)
CONFERENCE / WORKSHOP
6 - 8 NOVEMBER 2023
ESO GARCHING



GRAPHCORE

Gautier Soubrane
Business Development Director



Neural network visualization from [POPLAR™](#)

GRAPHCORE



Graphcore is a UK-based global chip design company established in 2016.

Graphcore has designed the Intelligence Processing Unit (IPU) from the ground up considering the requirements of Machine Intelligence workloads.

Find out more at www.graphcore.ai



GRAPHCORE'S MISSION

Deliver the world's best processors

and software tools, to support the next

breakthroughs in machine intelligence,

that will expand human potential.



The Hardware Lottery

Sara Hooker

Google Research, Brain Team

shooker@google.com

Abstract

Hardware, systems and algorithms research communities have historically had different incentive structures and fluctuating motivation to engage with each other explicitly. This historical treatment is odd given that hardware and software have frequently determined which research ideas succeed (and fail). This essay introduces the term hardware lottery to describe when a research idea wins because it is suited to the available software and hardware and *not* because the idea is superior to alternative research directions. Examples from early computer science history illustrate how hardware lotteries can delay research progress by casting successful ideas as failures. These lessons are particularly salient given the advent of domain specialized hardware which make it increasingly costly to stray off of the beaten path of research ideas. This essay posits that the gains from progress in computing are likely to become even more uneven, with certain research directions moving into the fast-lane while progress on others is further obstructed.

HARDWARE CAN BE AN ENABLING OR A LIMITING FACTOR IN THE SUCCESS OF IDEAS



THE INTELLIGENCE PROCESSING UNIT (IPU)

WHAT MAKES IT DIFFERENT?

CPU

GPU

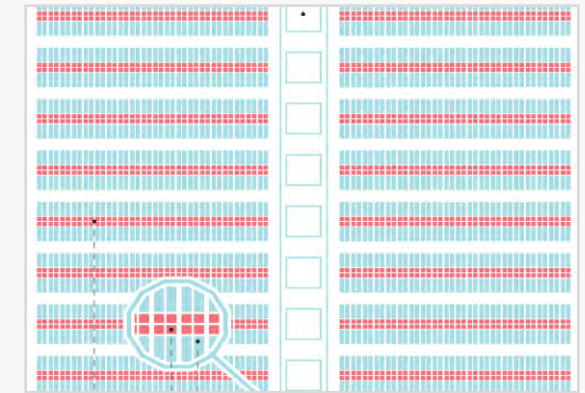
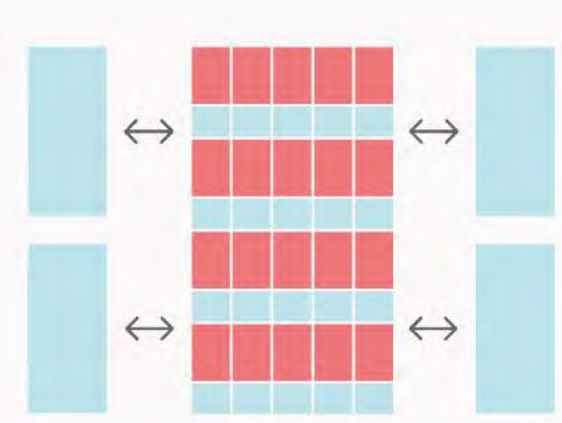
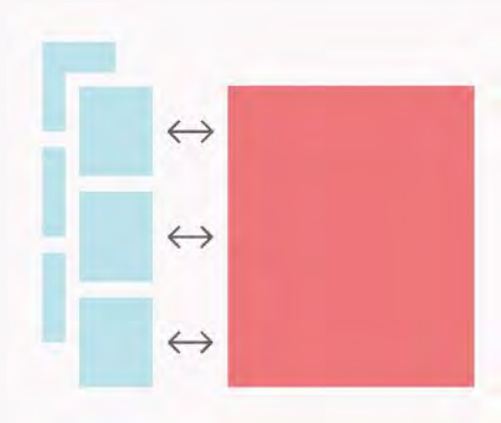
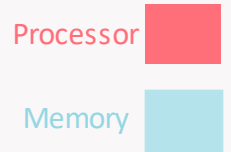
IPU

Parallelism

Designed for scalar processing

SIMD/SIMT architecture.
Designed for large blocks of dense contiguous data

Massively parallel MIMD architecture.
High performance/efficiency
for future ML trends



Memory Bandwidth

Off-chip memory

Model and Data spread across off-chip and small on-chip cache and shared memory

Main Model & Data in tightly coupled large locally distributed SRAM

(2TB/s for A100 HBM)

(~65 TB/s for Bow IPU)



INTRODUCING THE BOW IPU

WORLD'S FIRST 3D WAFER-ON-WAFER PROCESSOR



3D silicon wafer stacked processor

350 TeraFLOPS AI compute

Optimized silicon power delivery

0.9 GigaByte In-Processor-Memory @ **65TB/s**

1,472 independent processor cores

8,832 independent parallel programs

10x IPU-Links™ delivering 320GB/s

IPU vs GPU MICROARCHITECTURE

Feature	Bow IPU	A100 GPU
FP16/ FP32 Flops	350 / 87 TFlops	312 / 156 TFlops
Independent Threads (IIS: Independent Instruction Stream)	1472 (tiles) x 6 (threads) = 8832 IIS	108 (SMs) x 64 (warps) = 6912 IIS
SRAM per core	624KiB (per tile)	192KiB (up to 164KiB shared) (per SM)
“Contention free” single-cycle-access memory (SRAM/registers) per IIS	624/6 = 104 KiB	192/64 + 256/64 = 7 KiB
SIMD vector width per IIS (Lower is better, easier to keep filled)	2 (float32) 4 (float16)	32 (float32) 64 (float16)
Integer	Non-vectorized (but can be dual issued with vector float ops).	Vectorized with above width (but no dual issue).
HW RNG	Yes, based on xoroshiro (enables HW stochastic rounding)	No (but maybe not a bottleneck in most applications?)

IPU unique MICROARCHITECTURE



Steering Customized AI Architectures for HPC Scientific Applications

Hatem Ltaief¹, Yuxi Hong¹, Adel Dabah¹, Rabab Alomairy¹, Sameh Abdulah¹,
Chris Goreczny², Pawel Gepner³, Matteo Ravasi¹, Damien Gratadour⁴, and
David Keyes¹

¹Extreme Computing Research Center
Division of Computer, Electrical, and Mathematical Sciences and Engineering
King Abdullah University of Science and Technology
Thuwal, Jeddah 23955 Saudi Arabia

Hatem.Ltaief, Yuxi.Hong, Adel.Dabah.1, Rabab.Alomairy, Sameh.Abdulah,
Matteo.Ravasi, David.Keyes@kaust.edu.sa

²Graphcore, Poland
chrisgo@graphcore.ai

³Warsaw University of Technology, Poland
pawel.gepner@pw.edu.pl

⁴Paris Observatory, France
damien.gratadour@obspm.fr

Abstract. AI hardware technologies have revolutionized computational science. While they have been mostly used to accelerate deep learning training and inference models for machine learning, HPC scientific applications do not seem to directly benefit from these specific hardware features unless AI-based components are introduced into their simulation workflows, for instance, as a replacement of their numerical solvers. This paper proposes to take another direction in an attempt to democratize customized AI architectures for HPC scientific computing. The main idea consists in demonstrating how legacy applications can leverage these AI engines after a necessary algorithmic redesign. It is critical that the resulting software implementations map onto the underlying memory-austere hardware architectures to extract the expected performance. To facilitate this process, we promote the matricization technique for restructuring codes (1) by exploiting data sparsity via algebraic compression and (2) by expressing the critical computational phases in terms of tile low-rank matrix-vector multiplications (TLR-MVM) and batch matrix-matrix multiplications (batch GEMM). Algebraic compression enables to reduce memory footprint and to fit into small local cache/memory, while batch execution ensures high occupancy. We highlight how we can steer the Graphcore AI-focused Wafer-on-Wafer Intelligence Processing Units (IPUs) to deliver high performance for both operations. We conduct a performance benchmarking campaign of these two matrix operations that account for most of the elapsed times of four real applications in computational astronomy, seismic imaging, wireless communications, and climate/weather predictions. We report bandwidth and execution rates with speedup factors up to 150X/14X/25X/40X, respectively, on IPUs compared to other systems.

Intelligence Processing Unit (IPU)

Competitor to NVIDIA A100/H100

- **Similar FLOPs, but lots more SRAM/thread**

Better perf/\$ for most ML workloads

- **IPUs don't rely on expensive HBM logic and sub-7nm manufacturing processes.**

Much better for “interesting” workloads

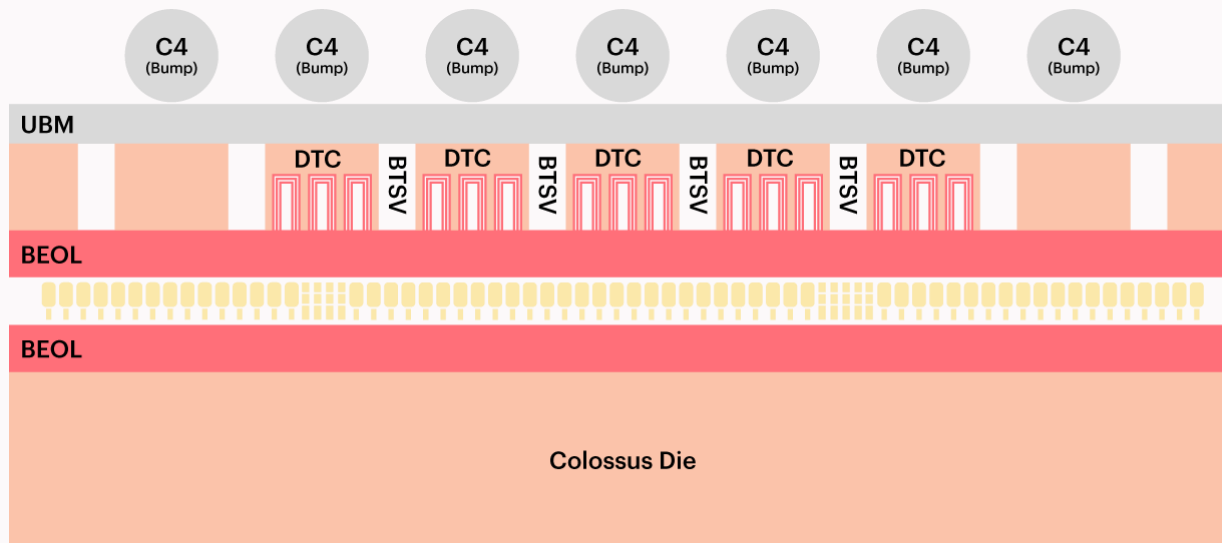
- **Graph Neural Networks**
- **Sparse models**
- **Hybrid ML/HPC workloads**

Available in cloud or on prem

- **6000 in data centers worldwide**
- **Try for free on DigitalOcean Paperspace**
- **Program in PyTorch, TF, JAX, or low-level C++**



BOW IPU: 3D WAFER-ON-WAFER PROCESSOR

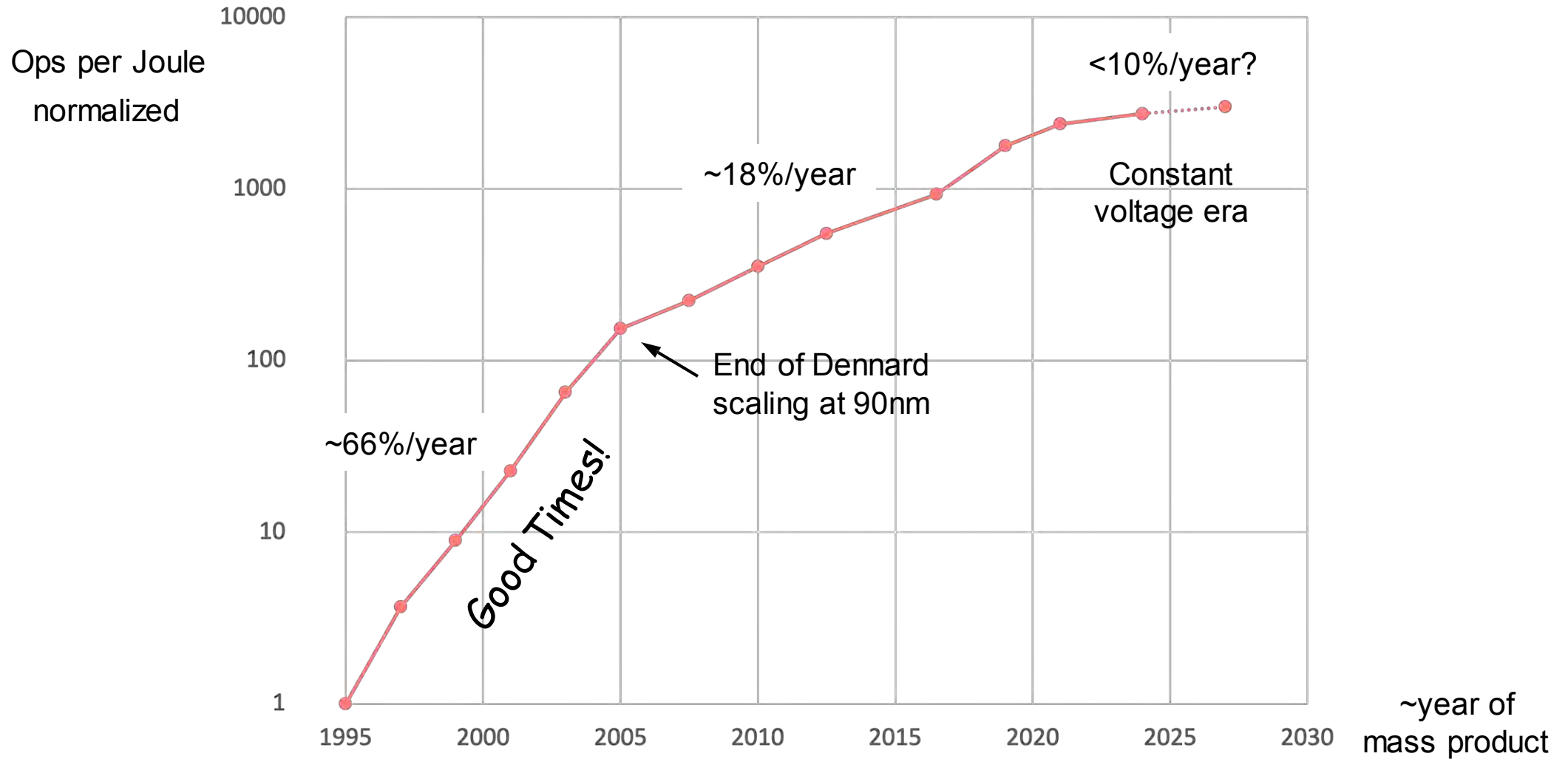


Advanced silicon wafer stacking technology co-developed between Graphcore and TSMC

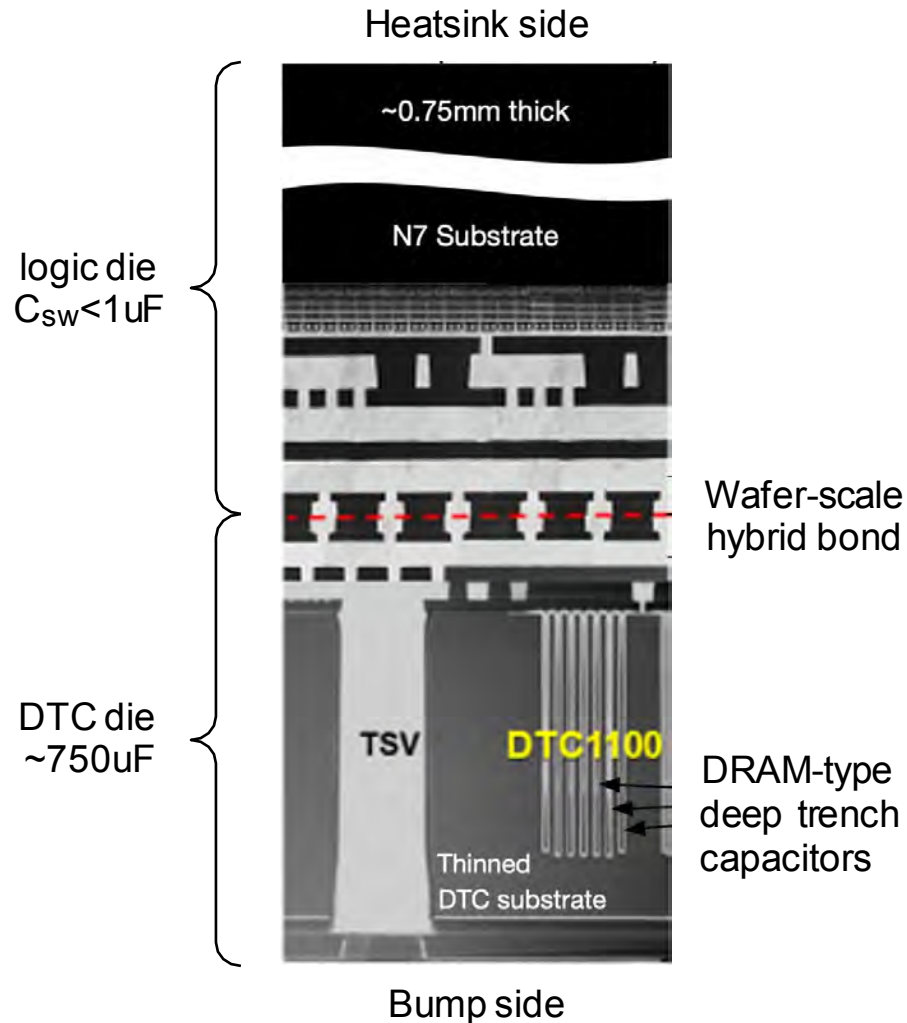
World's first commercial deployment using TSMC SoIC-WoW™ technology in Bow IPU

Enabling technology for closely coupled power delivery die to maximize application performance

Performance per Watt

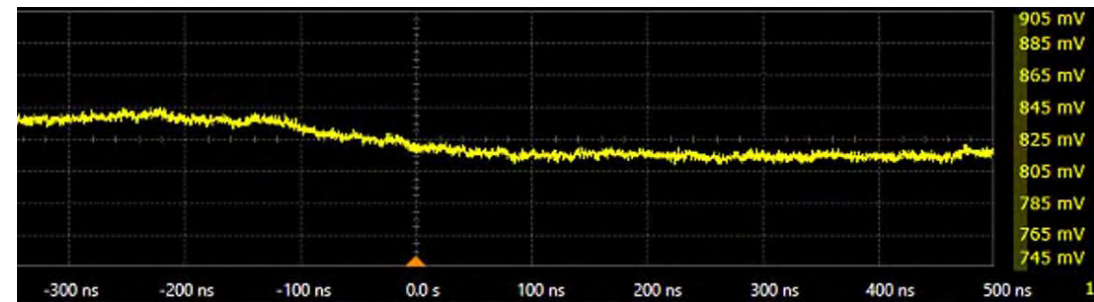
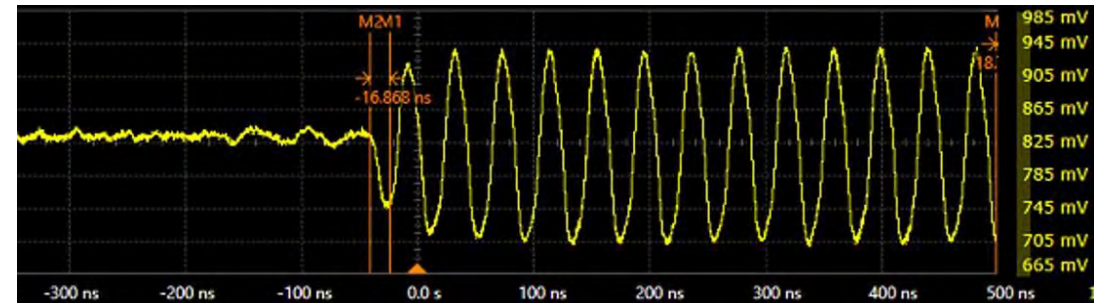


TAMING VDD



Graphcore Colossus Mk2w:

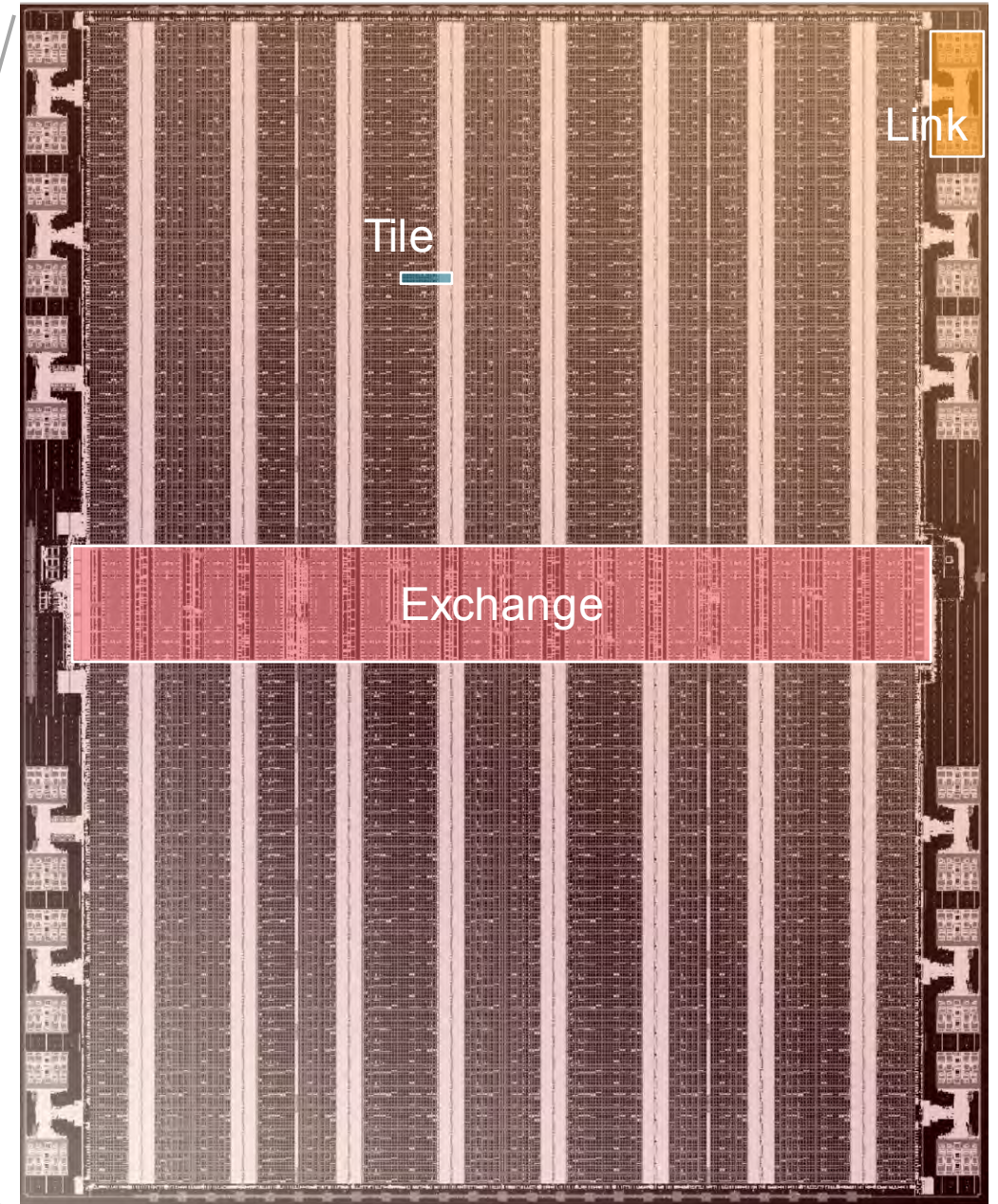
- First Wafer-on-Wafer (WoW) 3D logic chip
- 1.4x speed at same energy/op



Vdd at die without/with DTC; 25MHz 50/50 min/max activity virus

GRAPHCORE COLOSSUS MK2 IPU

- Full-reticle N7, 14 metals, 59bn transistors
- 1472 processors (MIMD)
- 350Tflop/s fp16, 87Tflop/s fp32
- 897MiB distributed SRAM @ 65TB/s
- 11TB/s crossbar inter-tile interconnect
- 10x 16GB/s inter-chip links, 2x 16GB/s PCIe



BOW IPU

Deep Trench Capacitor

Efficient power delivery
Enables increase in operational performance

Wafer-On-Wafer

Advanced silicon 3D stacking technology
Closely coupled power delivery die
Higher operating frequency and enhanced overall performance

IPU-Tiles™

1472 independent IPU-Tiles™ each with an IPU-Core™ and In-Processor-Memory™

IPU-Core™

1472 independent IPU-Core™
8832 independent program threads executing in parallel

In-Processor-Memory™

900MB In-Processor-Memory™ per IPU
65.4TB/s memory bandwidth per IPU

Solder Bumps

IPU-Links™

10x IPU-Links,
320GB/s chip to chip bandwidth

IPU-Exchange™

11 TB/s all to all IPU-Exchange™
Non-blocking, any communication pattern

PCIe

PCI Gen4 x16
64 GB/s bidirectional bandwidth to host



BOW-2000 IPU MACHINE

1U blade form factor delivering 1.4 PetaFLOPS AI Compute

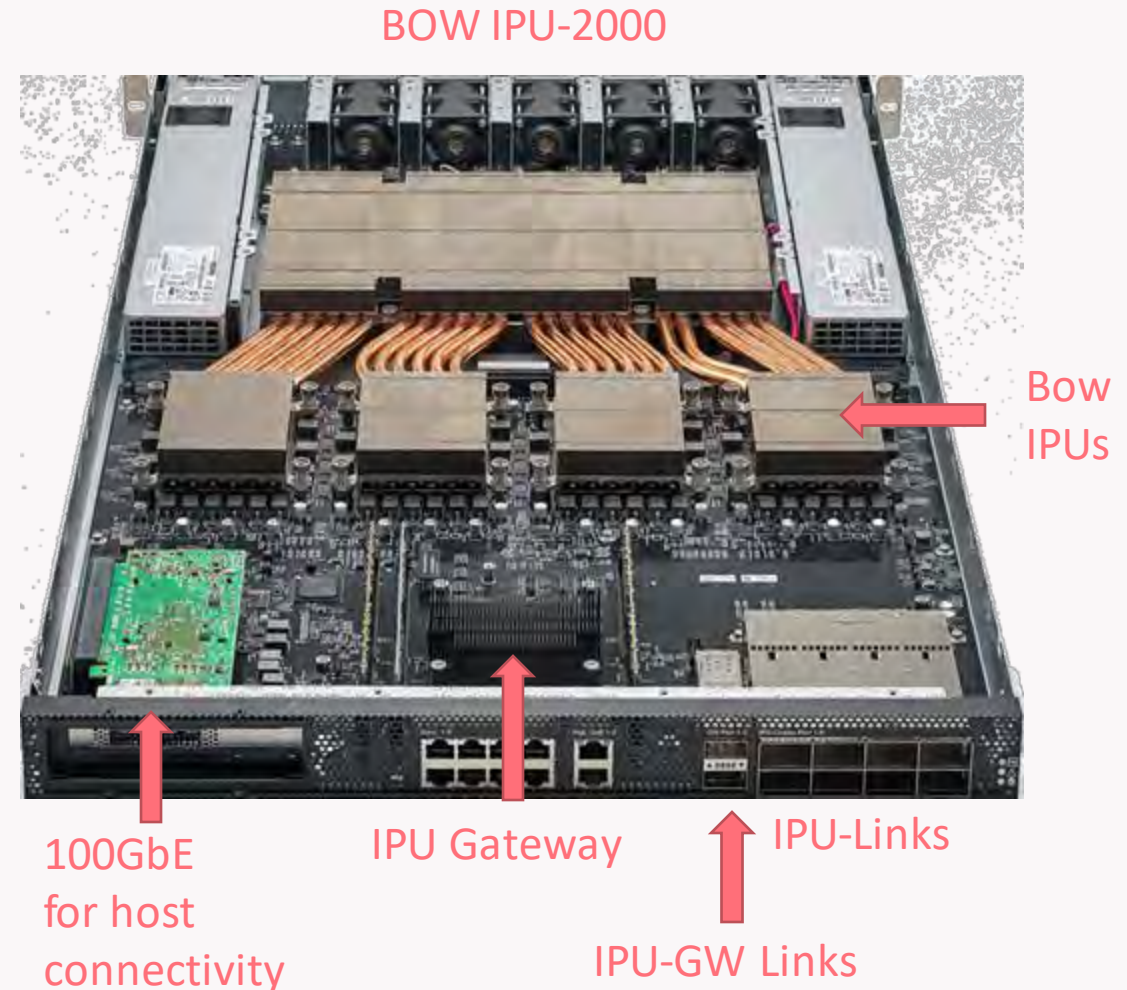
Disaggregated AI/ML accelerator platform

Excellent performance & TCO leveraging
Processor memory & IPU-Exchange

IPU-Links scale to Bow Pod64

Expansion to Bow Pod256 and beyond
with IPU-GW Links

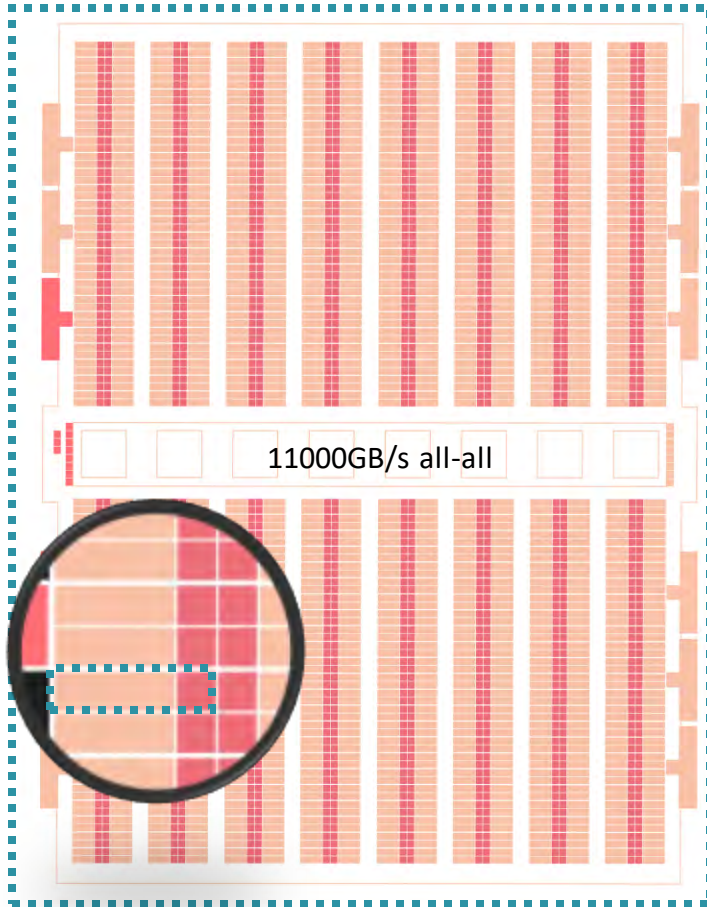
In-



HOW IPU WORKS

CHIP-LEVEL

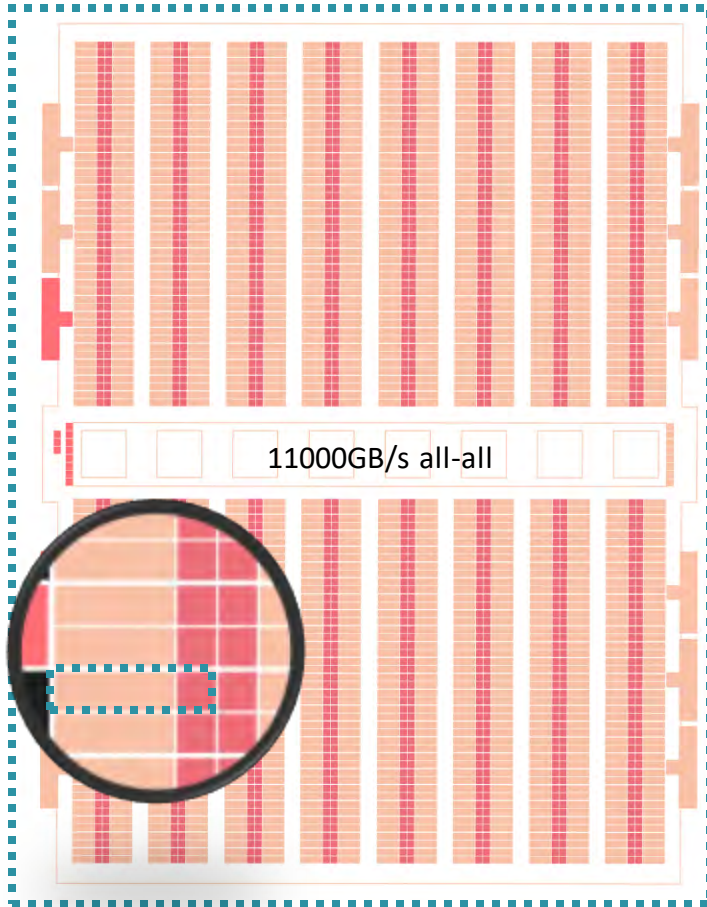
BOW IPU
350TFLOPS(F16)
900MB SRAM
1472 IPU-Tiles
8832 independent
instruction streams



IPU-Tile
236GFLOPS(F16)
640KB SRAM
6 HW Threads 1.83GHz
128 F16Ops/Cycle

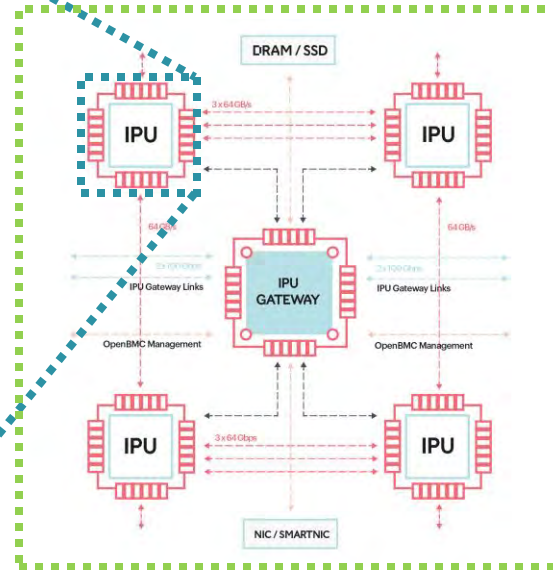
HOW IPU WORKS POD-LEVEL

BOW IPU
 350TFLOPS(F16)
 900MB SRAM
 1472 IPU-Tiles
 8832 independent
 instruction streams



IPU-Tile
 236GFLOPS(F16)
 640KB SRAM
 6 HW Threads 1.83GHz
 128 F16Ops/Cycle

BOW-2000 IPU-Machine
 4x IPU
 256 GB DRAM

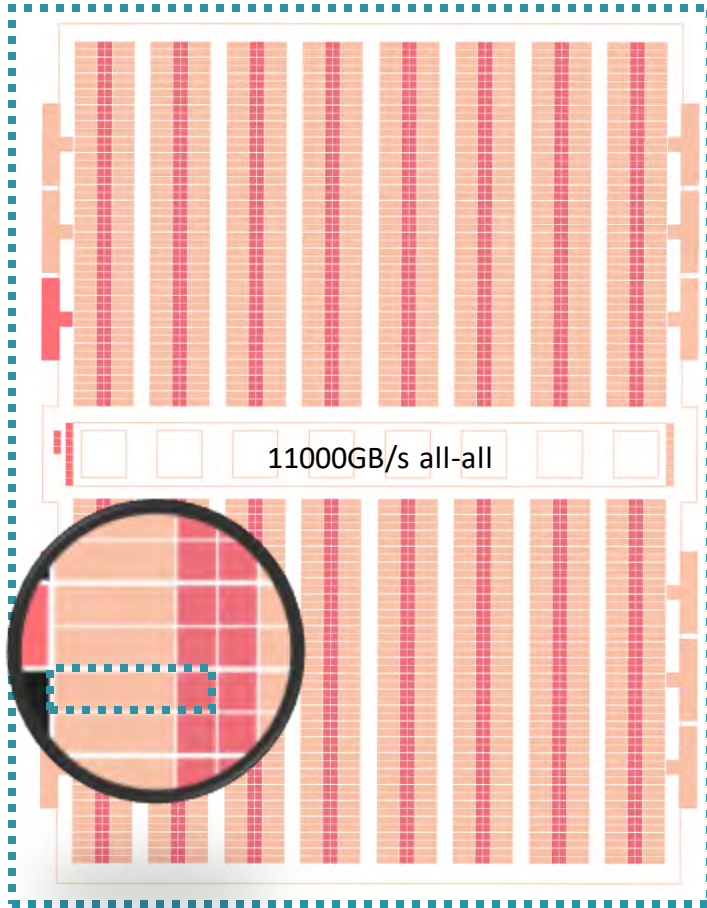


320 GB/s chip-to-chip bandwidth

HOW IPU WORKS

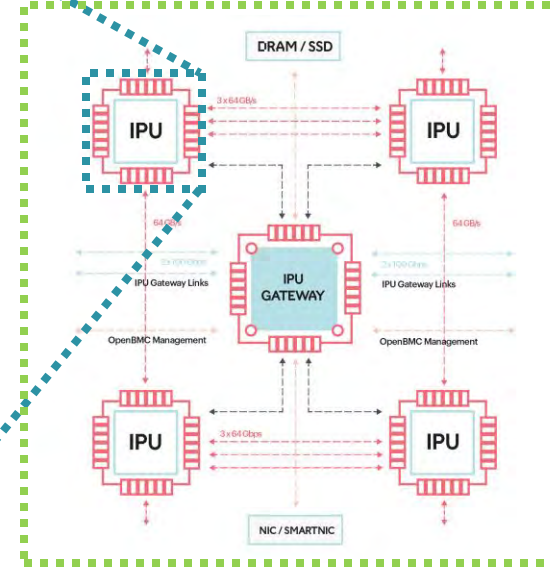
SYSTEM-LEVEL

BOW IPU
 350TFLOPS(F16)
 900MB SRAM
 1472 IPU-Tiles
 8832 independent instruction streams

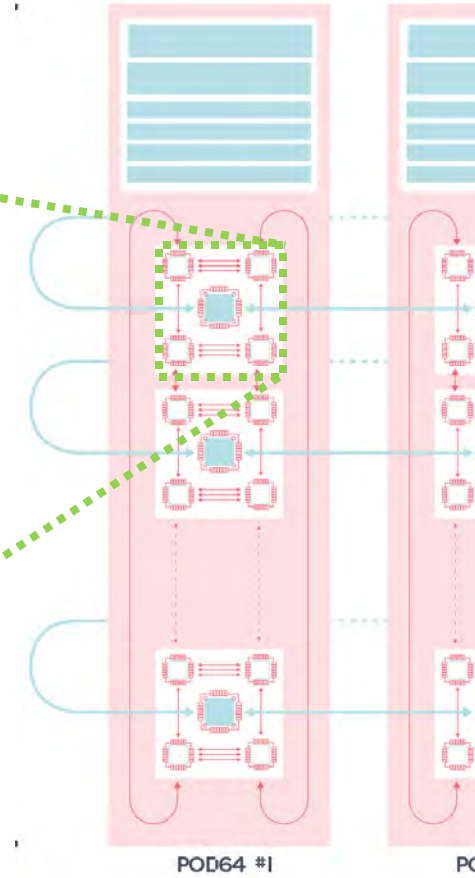


IPU-Tile
 236GFLOPS(F16)
 640KB SRAM
 6 HW Threads 1.83GHz
 128 F16Ops/Cycle

BOW-2000 IPU-Machine
 4x IPU
 256 GB DRAM



320 GB/s chip-to-chip bandwidth



HOW IPU WORKS

SCALING THE SYSTEM



POD₁₆



POD₆₄



POD₂₅₆

COMPUTE:	5.6 PetaFLOPS	22.4 PetaFLOPS	89.6 PetaFLOPS
ON-CHIP CACHE:	14.7 GB	58.9 GB	235.5 GB
OFF-CHIP MEMORY:	256 GB	1024 GB	4096 GB
CPU SERVERS:	1	1-4	4-16

PROGRAMMING THE IPU

PyTorch/JAX Model Code
~100 lines

```
# Start with token embeddings
embeddings = cfg.lambda_e * params.embeddings[x, :] # L x Dm

# Add (learned) positional encodings
embeddings += cfg.lambda_pe * params.positional_encodings[:, :]

# Apply the transformer layers
for layer in params.layers:

    # Layer-normalize embeddings
    t1 = vmap(standardize)(embeddings)
    t1 = elementwise_linear(layer.norm_self_attn, t1) # L x Dm

    # Multi-head self-attention
    for head in layer.heads:

        # Project into this head's query/key space
        query = linear(head.query, t1) # L x Dk
        key = linear(head.key, t1) # L x Dk

        # Compute L x L attention matrix
        score = query @ key.T + mask # L x L
        attn = jax.nn.softmax(cfg.tau * score, axis=1) # L x L

        value = linear(head.value, t1) # L x Dm
        self_attn = attn @ value # L x Dm

    # Add this head's contribution into embeddings
    embeddings += self_attn # L x Dm

    # Layer-normalize embeddings
    t2 = vmap(standardize)(embeddings)
    t2 = elementwise_linear(layer.norm_ff, t2) # L x Dm

    # Feedforward fully connected
    t2 = linear(layer.ffn1, t2) # L x Dff
    t2 = jax.nn.relu(t2)
    t2 = linear(layer.ffn2, t2) # L x Dm

    # Add this layer's contribution into embeddings
    embeddings += t2

# Layer-normalize embeddings
embeddings = vmap(standardize)(embeddings)
embeddings = elementwise_linear(params.pre_output_norm, embeddings)

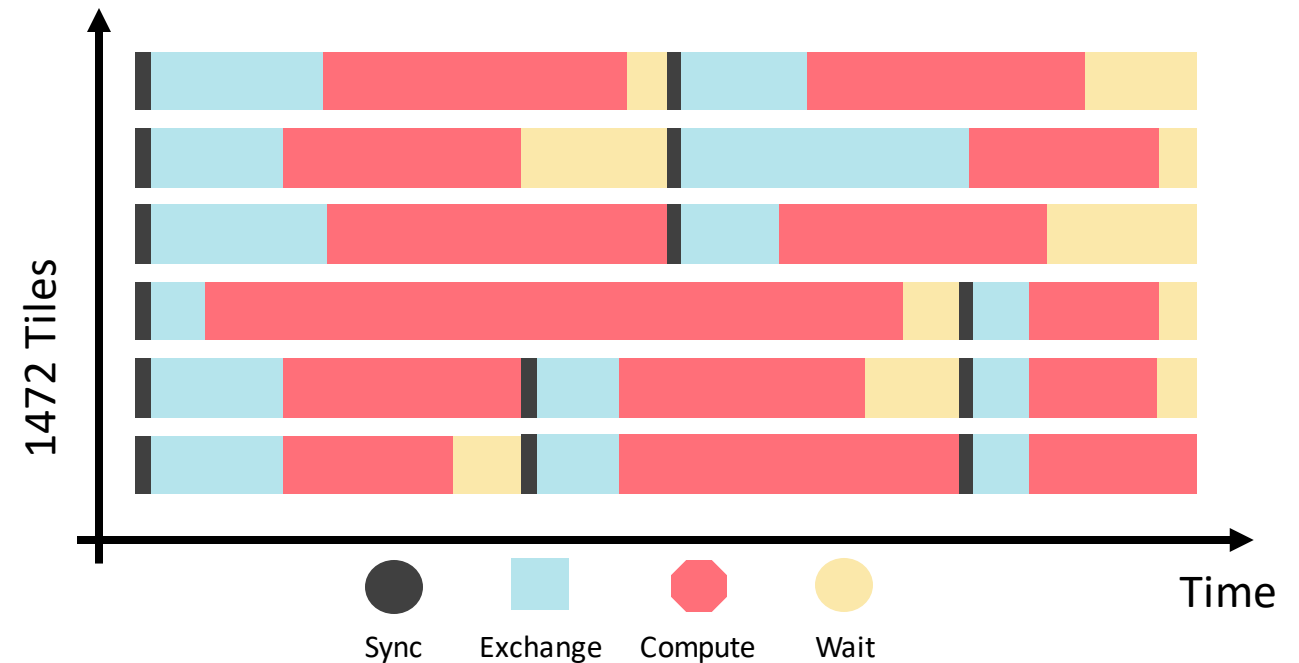
# And linearly project to output dimension
return linear(params.output, embeddings) # L x n_vocab
```

XLVA/MLIR Compute Graph
~ 1K nodes, O(GB) tensors

```
module_0035 jit_layer7.before_optimizations.tst v poplar/xla/module_0035 jit_layer7.before_optimizations.tst
144 Arg_0.907 = %21[] parameter(0)
145 Arg_1.908 = %22[] parameter(1)
146 ROOT add.909 = %21[] add(Arg_0.907, Arg_1.908), metadata=(op_name="jit(layer)/jit(main)", ...)
...
region_76.915 {
147 Arg_0.916 = %21[] parameter(0)
148 Arg_1.917 = %22[] parameter(1)
149 ROOT add.920 = %21[] add(Arg_0.916, Arg_1.917), metadata=(op_name="jit(layer)/jit(main)", ...)
...
norm_21.921 {
149 Arg_0.922 = %21[256,256][1,0] parameter(0)
150 multiply.924 = %21[256,256][1,0] multiply(Arg_0.922, Arg_0.922), metadata=(op_name="jit(layer)/jit(main)", ...)
151 constant.923 = %21[] constant(0)
152 reduce.925 = %21[] reduce(multiply.924, constant.923), dimensions=(0,1), to_apply=region_76.915,
153 ROOT sqrt.926 = %21[] sqrt(reduce.925), metadata=(op_name="jit(layer)/jit(main)", ...)
...
region_76.932 {
154 Arg_0.933 = %21[] parameter(0)
155 Arg_1.934 = %22[] parameter(1)
156 ROOT maximum.935 = %21[] maximum(Arg_0.933, Arg_1.934), metadata=(op_name="jit(layer)/jit(main)", ...)
...
region_76.943 {
157 Arg_0.944 = %21[] parameter(0)
158 Arg_1.945 = %22[] parameter(1)
159 ROOT add.946 = %21[] add(Arg_0.944, Arg_1.945), metadata=(op_name="jit(layer)/jit(main)", ...)
...
region_77.954 {
160 Arg_0.955 = %21[] parameter(0)
161 Arg_1.956 = %22[] parameter(1)
162 ROOT add.957 = %21[] add(Arg_0.955, Arg_1.956), metadata=(op_name="jit(layer)/jit(main)", ...)
...
norm_24.958 {
163 Arg_0.959 = %21[256,256][1,0] parameter(0)
164 multiply.960 = %21[256,256][1,0] multiply(Arg_0.959, Arg_0.959), metadata=(op_name="jit(layer)/jit(main)", ...)
165 constant.961 = %21[] constant(0)
166 ROOT reduce.962 = %21[] reduce(multiply.961, constant.961), dimensions=(0,1), to_apply=region_77.954,
167 ROOT sqrt.963 = %21[] sqrt(reduce.962), metadata=(op_name="jit(layer)/jit(main)", ...)
...
ENTRY main.967 {
168 Arg_0.1 = %21[128,128][1,0] parameter(0)
169 transpose.5 = %21[128,128][0,1] transpose(Arg_0.1), dimensions=(1,0), metadata=(op_name="jit(layer)/jit(main)", ...)
170 dot.6 = %21[128,128][1,0] dot(Arg_0.1, transpose.5), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
171 constant.4 = %21[] constant(-inf)
172 reduce.11 = %21[128,0] reduce(dot.6, constant.4), dimensions=(1), to_apply=region_76.915,
173 reshape.12 = %21[128,1][1,0] reshape(reduce.11), metadata=(op_name="jit(layer)/jit(main)", ...)
174 broadcast.13 = %21[128,1][1,0] broadcast(reshape.12), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
175 reshape.14 = %21[128][1,0] reshape(broadcast.13), metadata=(op_name="jit(layer)/jit(main)", ...)
176 broadcast.15 = %21[128,128][1,0] broadcast(reshape.14), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
177 subtract.16 = %21[128,128][1,0] subtract(dot.6, broadcast.15), metadata=(op_name="jit(layer)/jit(main)", ...)
178 exponential.17 = %21[128,128][1,0] exponential(subtract.16), metadata=(op_name="jit(layer)/jit(main)", ...)
179 constant.3 = %21[] constant(0)
180 reduce.22 = %21[128][0] reduce(exponential.17, constant.3), dimensions=(1), to_apply=region_76.915,
181 reshape.23 = %21[128,1][1,0] reshape(reduce.22), metadata=(op_name="jit(layer)/jit(main)", ...)
182 broadcast.24 = %21[128][1,0] broadcast(reshape.23), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
183 reshape.25 = %21[128][0] reshape(broadcast.24), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
184 broadcast.26 = %21[128,128][1,0] broadcast(reshape.25), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
185 divide.27 = %21[128,128][1,0] divide(exponential.17, broadcast.26), metadata=(op_name="jit(layer)/jit(main)", ...)
186 Arg_1.2 = %21[128,256][1,0] parameter(1)
187 dot.28 = %21[128,256][1,0] dot(divide.27, Arg_1.2), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
188 call.39 = %21[] call(dot.28), to_apply=norm.24.958
189 broadcast.40 = %21[128,256][1,0] broadcast(call.39), dimensions=(1), metadata=(op_name="jit(layer)/jit(main)", ...)
190 divide.41 = %21[128,256][1,0] divide(dot.28, broadcast.40), metadata=(op_name="jit(layer)/jit(main)", ...)
191 transpose.42 = %21[256,128][0,1] transpose(divide.41), dimensions=(1,0), metadata=(op_name="jit(layer)/jit(main)", ...)
192 dot.43 = %21[128,128][1,0] dot(divide.41, transpose.42), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
193 reduce.44 = %21[128][0] reduce(dot.43, constant.4), dimensions=(1), to_apply=region_76.915,
194 reshape.49 = %21[128,1][1,0] reshape(reduce.44), metadata=(op_name="jit(layer)/jit(main)", ...)
195 broadcast.50 = %21[128,1][1,0] broadcast(reshape.49), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
196 reshape.51 = %21[128][1,0] reshape(broadcast.50), metadata=(op_name="jit(layer)/jit(main)", ...)
197 broadcast.52 = %21[128,128][1,0] broadcast(reshape.51), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
198 subtract.53 = %21[128,128][1,0] subtract(dot.43, broadcast.52), metadata=(op_name="jit(layer)/jit(main)", ...)
199 exponential.54 = %21[128][0] exponential(subtract.53), metadata=(op_name="jit(layer)/jit(main)", ...)
200 reduce.59 = %21[128][0] reduce(exponential.54, constant.1), dimensions=(1), to_apply=region_76.915,
201 reshape.68 = %21[128,1][1,0] reshape(reduce.59), metadata=(op_name="jit(layer)/jit(main)", ...)
202 broadcast.61 = %21[128,1][1,0] broadcast(reshape.68), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
203 broadcast.63 = %21[128,128][1,0] broadcast(reshape.62), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
204 divide.64 = %21[128,128][1,0] divide(exponential.54, broadcast.63), metadata=(op_name="jit(layer)/jit(main)", ...)
205 dot.65 = %21[128,256][1,0] dot(divide.64, Arg_1.2), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
206 call.76 = %21[] call(dot.65), to_apply=norm.24.958
207 broadcast.77 = %21[128,256][1,0] broadcast(call.76), dimensions=(1), metadata=(op_name="jit(layer)/jit(main)", ...)
208 divide.78 = %21[128,256][1,0] divide(dot.65, broadcast.77), metadata=(op_name="jit(layer)/jit(main)", ...)
209 transpose.79 = %21[256,128][0,1] transpose(divide.78), dimensions=(1,0), metadata=(op_name="jit(layer)/jit(main)", ...)
210 dot.80 = %21[128,128][1,0] dot(divide.78, transpose.79), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
211 reduce.83 = %21[128][0] reduce(dot.80, constant.4), dimensions=(1), to_apply=region_76.915,
212 reshape.85 = %21[128][1,0] reshape(reduce.83), metadata=(op_name="jit(layer)/jit(main)", ...)
213 broadcast.87 = %21[128,1][1,0] broadcast(reshape.85), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
214 reshape.88 = %21[128][0] reshape(broadcast.87), metadata=(op_name="jit(layer)/jit(main)", ...)
215 broadcast.89 = %21[128,128][1,0] broadcast(reshape.88), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
216 subtract.90 = %21[128,128][1,0] subtract(dot.80, broadcast.89), metadata=(op_name="jit(layer)/jit(main)", ...)
217 exponential.91 = %21[128,128][1,0] exponential(subtract.90), metadata=(op_name="jit(layer)/jit(main)", ...)
218 reduce.96 = %21[128][0] reduce(exponential.91, constant.1), dimensions=(1), to_apply=region_76.915,
219 reshape.97 = %21[128,1][1,0] reshape(reduce.96), metadata=(op_name="jit(layer)/jit(main)", ...)
220 broadcast.98 = %21[128,1][1,0] broadcast(reshape.97), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
221 divide.99 = %21[128,1][1,0] divide(reshape.97, broadcast.98), metadata=(op_name="jit(layer)/jit(main)", ...)
222 broadcast.100 = %21[128,128][1,0] broadcast(divide.99), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
223 dot.101 = %21[128,128][1,0] dot(divide.99, broadcast.100), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
224 constant.102 = %21[] constant(-inf)
225 reduce.103 = %21[128,0] reduce(dot.101, constant.102), dimensions=(1), to_apply=region_76.915,
226 reshape.104 = %21[128,1][1,0] reshape(reduce.103), metadata=(op_name="jit(layer)/jit(main)", ...)
227 broadcast.105 = %21[128,1][1,0] broadcast(reshape.104), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
228 reshape.106 = %21[128][1,0] reshape(broadcast.105), metadata=(op_name="jit(layer)/jit(main)", ...)
229 broadcast.107 = %21[128,128][1,0] broadcast(reshape.106), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
230 subtract.108 = %21[128,128][1,0] subtract(dot.101, broadcast.107), metadata=(op_name="jit(layer)/jit(main)", ...)
231 exponential.109 = %21[128,128][1,0] exponential(subtract.108), metadata=(op_name="jit(layer)/jit(main)", ...)
232 reduce.110 = %21[128][0] reduce(exponential.109, constant.1), dimensions=(1), to_apply=region_76.915,
233 reshape.111 = %21[128,1][1,0] reshape(reduce.110), metadata=(op_name="jit(layer)/jit(main)", ...)
234 broadcast.112 = %21[128,1][1,0] broadcast(reshape.111), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
235 divide.113 = %21[128,1][1,0] divide(reshape.111, broadcast.112), metadata=(op_name="jit(layer)/jit(main)", ...)
236 broadcast.114 = %21[128,128][1,0] broadcast(divide.113), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
237 dot.115 = %21[128,128][1,0] dot(divide.113, broadcast.114), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
238 constant.116 = %21[] constant(-inf)
239 reduce.117 = %21[128,0] reduce(dot.115, constant.116), dimensions=(1), to_apply=region_76.915,
240 reshape.118 = %21[128,1][1,0] reshape(reduce.117), metadata=(op_name="jit(layer)/jit(main)", ...)
241 broadcast.119 = %21[128,1][1,0] broadcast(reshape.118), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
242 reshape.120 = %21[128][1,0] reshape(broadcast.119), metadata=(op_name="jit(layer)/jit(main)", ...)
243 broadcast.121 = %21[128,128][1,0] broadcast(reshape.120), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
244 subtract.122 = %21[128,128][1,0] subtract(dot.115, broadcast.121), metadata=(op_name="jit(layer)/jit(main)", ...)
245 exponential.123 = %21[128,128][1,0] exponential(subtract.122), metadata=(op_name="jit(layer)/jit(main)", ...)
246 reduce.124 = %21[128][0] reduce(exponential.123, constant.1), dimensions=(1), to_apply=region_76.915,
247 reshape.125 = %21[128,1][1,0] reshape(reduce.124), metadata=(op_name="jit(layer)/jit(main)", ...)
248 broadcast.126 = %21[128,1][1,0] broadcast(reshape.125), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
249 divide.127 = %21[128,1][1,0] divide(reshape.125, broadcast.126), metadata=(op_name="jit(layer)/jit(main)", ...)
250 broadcast.128 = %21[128,128][1,0] broadcast(divide.127), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
251 dot.129 = %21[128,128][1,0] dot(divide.127, broadcast.128), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
252 constant.130 = %21[] constant(-inf)
253 reduce.131 = %21[128,0] reduce(dot.129, constant.130), dimensions=(1), to_apply=region_76.915,
254 reshape.132 = %21[128,1][1,0] reshape(reduce.131), metadata=(op_name="jit(layer)/jit(main)", ...)
255 broadcast.133 = %21[128,1][1,0] broadcast(reshape.132), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
256 reshape.134 = %21[128][1,0] reshape(broadcast.133), metadata=(op_name="jit(layer)/jit(main)", ...)
257 broadcast.135 = %21[128,128][1,0] broadcast(reshape.134), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
258 subtract.136 = %21[128,128][1,0] subtract(dot.129, broadcast.135), metadata=(op_name="jit(layer)/jit(main)", ...)
259 exponential.137 = %21[128,128][1,0] exponential(subtract.136), metadata=(op_name="jit(layer)/jit(main)", ...)
260 constant.138 = %21[] constant(0)
261 reduce.139 = %21[128][0] reduce(exponential.137, constant.138), dimensions=(1), to_apply=region_76.915,
262 reshape.140 = %21[128,1][1,0] reshape(reduce.139), metadata=(op_name="jit(layer)/jit(main)", ...)
263 broadcast.141 = %21[128][1,0] broadcast(reshape.140), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
264 reshape.142 = %21[128][0] reshape(broadcast.141), metadata=(op_name="jit(layer)/jit(main)", ...)
265 broadcast.143 = %21[128,128][1,0] broadcast(reshape.142), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
266 divide.144 = %21[128,128][1,0] divide(exponential.137, broadcast.143), metadata=(op_name="jit(layer)/jit(main)", ...)
267 Arg_1.3 = %21[128,256][1,0] parameter(1)
268 dot.145 = %21[128,256][1,0] dot(divide.144, Arg_1.3), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
269 call.146 = %21[] call(dot.145), to_apply=norm.24.958
270 broadcast.147 = %21[128,256][1,0] broadcast(call.146), dimensions=(1), metadata=(op_name="jit(layer)/jit(main)", ...)
271 divide.148 = %21[128,256][1,0] divide(dot.145, broadcast.147), metadata=(op_name="jit(layer)/jit(main)", ...)
272 transpose.149 = %21[256,128][0,1] transpose(divide.148), dimensions=(1,0), metadata=(op_name="jit(layer)/jit(main)", ...)
273 dot.150 = %21[128,128][1,0] dot(divide.148, transpose.149), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
274 reduce.151 = %21[128][0] reduce(dot.150, constant.4), dimensions=(1), to_apply=region_76.915,
275 reshape.152 = %21[128,1][1,0] reshape(reduce.151), metadata=(op_name="jit(layer)/jit(main)", ...)
276 broadcast.153 = %21[128,1][1,0] broadcast(reshape.152), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
277 reshape.154 = %21[128][1,0] reshape(broadcast.153), metadata=(op_name="jit(layer)/jit(main)", ...)
278 broadcast.155 = %21[128,128][1,0] broadcast(reshape.154), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
279 subtract.156 = %21[128,128][1,0] subtract(dot.150, broadcast.155), metadata=(op_name="jit(layer)/jit(main)", ...)
280 exponential.157 = %21[128,128][1,0] exponential(subtract.156), metadata=(op_name="jit(layer)/jit(main)", ...)
281 reduce.158 = %21[128][0] reduce(exponential.157, constant.1), dimensions=(1), to_apply=region_76.915,
282 reshape.159 = %21[128,1][1,0] reshape(reduce.158), metadata=(op_name="jit(layer)/jit(main)", ...)
283 broadcast.160 = %21[128,1][1,0] broadcast(reshape.159), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
284 divide.161 = %21[128,1][1,0] divide(reshape.159, broadcast.160), metadata=(op_name="jit(layer)/jit(main)", ...)
285 broadcast.162 = %21[128,128][1,0] broadcast(divide.161), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
286 dot.163 = %21[128,128][1,0] dot(divide.161, broadcast.162), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
287 constant.164 = %21[] constant(-inf)
288 reduce.165 = %21[128,0] reduce(dot.163, constant.164), dimensions=(1), to_apply=region_76.915,
289 reshape.166 = %21[128,1][1,0] reshape(reduce.165), metadata=(op_name="jit(layer)/jit(main)", ...)
290 broadcast.167 = %21[128,1][1,0] broadcast(reshape.166), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
291 divide.168 = %21[128,1][1,0] divide(reshape.166, broadcast.167), metadata=(op_name="jit(layer)/jit(main)", ...)
292 broadcast.169 = %21[128,128][1,0] broadcast(divide.168), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
293 dot.170 = %21[128,128][1,0] dot(divide.168, broadcast.169), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
294 constant.171 = %21[] constant(-inf)
295 reduce.171 = %21[128,0] reduce(dot.170, constant.171), dimensions=(1), to_apply=region_76.915,
296 reshape.172 = %21[128,1][1,0] reshape(reduce.171), metadata=(op_name="jit(layer)/jit(main)", ...)
297 broadcast.173 = %21[128,1][1,0] broadcast(reshape.172), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
298 divide.174 = %21[128,1][1,0] divide(reshape.172, broadcast.173), metadata=(op_name="jit(layer)/jit(main)", ...)
299 broadcast.175 = %21[128,128][1,0] broadcast(divide.174), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
300 dot.176 = %21[128,128][1,0] dot(divide.174, broadcast.175), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
301 constant.177 = %21[] constant(-inf)
302 reduce.177 = %21[128,0] reduce(dot.176, constant.177), dimensions=(1), to_apply=region_76.915,
303 reshape.178 = %21[128,1][1,0] reshape(reduce.177), metadata=(op_name="jit(layer)/jit(main)", ...)
304 broadcast.179 = %21[128,1][1,0] broadcast(reshape.178), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
305 divide.180 = %21[128,1][1,0] divide(reshape.178, broadcast.179), metadata=(op_name="jit(layer)/jit(main)", ...)
306 broadcast.181 = %21[128,128][1,0] broadcast(divide.180), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
307 dot.182 = %21[128,128][1,0] dot(divide.180, broadcast.181), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
308 constant.183 = %21[] constant(-inf)
309 reduce.183 = %21[128,0] reduce(dot.182, constant.183), dimensions=(1), to_apply=region_76.915,
310 reshape.184 = %21[128,1][1,0] reshape(reduce.183), metadata=(op_name="jit(layer)/jit(main)", ...)
311 broadcast.185 = %21[128,1][1,0] broadcast(reshape.184), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
312 divide.186 = %21[128,1][1,0] divide(reshape.184, broadcast.185), metadata=(op_name="jit(layer)/jit(main)", ...)
313 broadcast.187 = %21[128,128][1,0] broadcast(divide.186), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
314 dot.188 = %21[128,128][1,0] dot(divide.186, broadcast.187), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
315 constant.189 = %21[] constant(-inf)
316 reduce.189 = %21[128,0] reduce(dot.188, constant.189), dimensions=(1), to_apply=region_76.915,
317 reshape.190 = %21[128,1][1,0] reshape(reduce.189), metadata=(op_name="jit(layer)/jit(main)", ...)
318 broadcast.191 = %21[128,1][1,0] broadcast(reshape.190), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
319 divide.192 = %21[128,1][1,0] divide(reshape.190, broadcast.191), metadata=(op_name="jit(layer)/jit(main)", ...)
320 broadcast.193 = %21[128,128][1,0] broadcast(divide.192), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
321 dot.194 = %21[128,128][1,0] dot(divide.192, broadcast.193), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
322 constant.195 = %21[] constant(-inf)
323 reduce.195 = %21[128,0] reduce(dot.194, constant.195), dimensions=(1), to_apply=region_76.915,
324 reshape.196 = %21[128,1][1,0] reshape(reduce.195), metadata=(op_name="jit(layer)/jit(main)", ...)
325 broadcast.197 = %21[128,1][1,0] broadcast(reshape.196), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
326 divide.198 = %21[128,1][1,0] divide(reshape.196, broadcast.197), metadata=(op_name="jit(layer)/jit(main)", ...)
327 broadcast.199 = %21[128,128][1,0] broadcast(divide.198), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
328 dot.200 = %21[128,128][1,0] dot(divide.198, broadcast.199), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
329 constant.201 = %21[] constant(-inf)
330 reduce.200 = %21[128,0] reduce(dot.200, constant.201), dimensions=(1), to_apply=region_76.915,
331 reshape.201 = %21[128,1][1,0] reshape(reduce.200), metadata=(op_name="jit(layer)/jit(main)", ...)
332 broadcast.202 = %21[128,1][1,0] broadcast(reshape.201), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
333 divide.203 = %21[128,1][1,0] divide(reshape.201, broadcast.202), metadata=(op_name="jit(layer)/jit(main)", ...)
334 broadcast.204 = %21[128,128][1,0] broadcast(divide.203), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
335 dot.205 = %21[128,128][1,0] dot(divide.203, broadcast.204), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
336 constant.206 = %21[] constant(-inf)
337 reduce.206 = %21[128,0] reduce(dot.205, constant.206), dimensions=(1), to_apply=region_76.915,
338 reshape.207 = %21[128,1][1,0] reshape(reduce.206), metadata=(op_name="jit(layer)/jit(main)", ...)
339 broadcast.208 = %21[128,1][1,0] broadcast(reshape.207), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
340 divide.209 = %21[128,1][1,0] divide(reshape.207, broadcast.208), metadata=(op_name="jit(layer)/jit(main)", ...)
341 broadcast.210 = %21[128,128][1,0] broadcast(divide.209), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
342 dot.211 = %21[128,128][1,0] dot(divide.209, broadcast.210), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
343 constant.212 = %21[] constant(-inf)
344 reduce.212 = %21[128,0] reduce(dot.211, constant.212), dimensions=(1), to_apply=region_76.915,
345 reshape.213 = %21[128,1][1,0] reshape(reduce.212), metadata=(op_name="jit(layer)/jit(main)", ...)
346 broadcast.214 = %21[128,1][1,0] broadcast(reshape.213), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
347 divide.215 = %21[128,1][1,0] divide(reshape.213, broadcast.214), metadata=(op_name="jit(layer)/jit(main)", ...)
348 broadcast.216 = %21[128,128][1,0] broadcast(divide.215), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
349 dot.217 = %21[128,128][1,0] dot(divide.215, broadcast.216), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
350 constant.218 = %21[] constant(-inf)
351 reduce.218 = %21[128,0] reduce(dot.217, constant.218), dimensions=(1), to_apply=region_76.915,
352 reshape.219 = %21[128,1][1,0] reshape(reduce.218), metadata=(op_name="jit(layer)/jit(main)", ...)
353 broadcast.220 = %21[128,1][1,0] broadcast(reshape.219), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
354 divide.221 = %21[128,1][1,0] divide(reshape.219, broadcast.220), metadata=(op_name="jit(layer)/jit(main)", ...)
355 broadcast.222 = %21[128,128][1,0] broadcast(divide.221), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
356 dot.223 = %21[128,128][1,0] dot(divide.221, broadcast.222), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
357 constant.224 = %21[] constant(-inf)
358 reduce.224 = %21[128,0] reduce(dot.223, constant.224), dimensions=(1), to_apply=region_76.915,
359 reshape.225 = %21[128,1][1,0] reshape(reduce.224), metadata=(op_name="jit(layer)/jit(main)", ...)
360 broadcast.226 = %21[128,1][1,0] broadcast(reshape.225), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
361 divide.227 = %21[128,1][1,0] divide(reshape.225, broadcast.226), metadata=(op_name="jit(layer)/jit(main)", ...)
362 broadcast.228 = %21[128,128][1,0] broadcast(divide.227), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
363 dot.229 = %21[128,128][1,0] dot(divide.227, broadcast.228), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
364 constant.230 = %21[] constant(-inf)
365 reduce.229 = %21[128,0] reduce(dot.229, constant.230), dimensions=(1), to_apply=region_76.915,
366 reshape.230 = %21[128,1][1,0] reshape(reduce.229), metadata=(op_name="jit(layer)/jit(main)", ...)
367 broadcast.231 = %21[128,1][1,0] broadcast(reshape.230), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
368 divide.232 = %21[128,1][1,0] divide(reshape.230, broadcast.231), metadata=(op_name="jit(layer)/jit(main)", ...)
369 broadcast.233 = %21[128,128][1,0] broadcast(divide.232), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
370 dot.234 = %21[128,128][1,0] dot(divide.232, broadcast.233), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
371 constant.235 = %21[] constant(-inf)
372 reduce.235 = %21[128,0] reduce(dot.234, constant.235), dimensions=(1), to_apply=region_76.915,
373 reshape.236 = %21[128,1][1,0] reshape(reduce.235), metadata=(op_name="jit(layer)/jit(main)", ...)
374 broadcast.237 = %21[128,1][1,0] broadcast(reshape.236), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
375 divide.238 = %21[128,1][1,0] divide(reshape.236, broadcast.237), metadata=(op_name="jit(layer)/jit(main)", ...)
376 broadcast.239 = %21[128,128][1,0] broadcast(divide.238), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
377 dot.240 = %21[128,128][1,0] dot(divide.238, broadcast.239), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
378 constant.241 = %21[] constant(-inf)
379 reduce.240 = %21[128,0] reduce(dot.240, constant.241), dimensions=(1), to_apply=region_76.915,
380 reshape.241 = %21[128,1][1,0] reshape(reduce.240), metadata=(op_name="jit(layer)/jit(main)", ...)
381 broadcast.242 = %21[128,1][1,0] broadcast(reshape.241), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
382 divide.243 = %21[128,1][1,0] divide(reshape.241, broadcast.242), metadata=(op_name="jit(layer)/jit(main)", ...)
383 broadcast.244 = %21[128,128][1,0] broadcast(divide.243), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
384 dot.245 = %21[128,128][1,0] dot(divide.243, broadcast.244), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
385 constant.246 = %21[] constant(-inf)
386 reduce.245 = %21[128,0] reduce(dot.245, constant.246), dimensions=(1), to_apply=region_76.915,
387 reshape.246 = %21[128,1][1,0] reshape(reduce.245), metadata=(op_name="jit(layer)/jit(main)", ...)
388 broadcast.247 = %21[128,1][1,0] broadcast(reshape.246), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
389 divide.248 = %21[128,1][1,0] divide(reshape.246, broadcast.247), metadata=(op_name="jit(layer)/jit(main)", ...)
390 broadcast.249 = %21[128,128][1,0] broadcast(divide.248), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
391 dot.250 = %21[128,128][1,0] dot(divide.248, broadcast.249), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
392 constant.251 = %21[] constant(-inf)
393 reduce.250 = %21[128,0] reduce(dot.250, constant.251), dimensions=(1), to_apply=region_76.915,
394 reshape.251 = %21[128,1][1,0] reshape(reduce.250), metadata=(op_name="jit(layer)/jit(main)", ...)
395 broadcast.252 = %21[128,1][1,0] broadcast(reshape.251), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
396 divide.253 = %21[128,1][1,0] divide(reshape.251, broadcast.252), metadata=(op_name="jit(layer)/jit(main)", ...)
397 broadcast.254 = %21[128,128][1,0] broadcast(divide.253), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
398 dot.255 = %21[128,128][1,0] dot(divide.253, broadcast.254), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
399 constant.256 = %21[] constant(-inf)
400 reduce.255 = %21[128,0] reduce(dot.255, constant.256), dimensions=(1), to_apply=region_76.915,
401 reshape.256 = %21[128,1][1,0] reshape(reduce.255), metadata=(op_name="jit(layer)/jit(main)", ...)
402 broadcast.257 = %21[128,1][1,0] broadcast(reshape.256), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
403 divide.258 = %21[128,1][1,0] divide(reshape.256, broadcast.257), metadata=(op_name="jit(layer)/jit(main)", ...)
404 broadcast.259 = %21[128,128][1,0] broadcast(divide.258), dimensions=(0,1), metadata=(op_name="jit(layer)/jit(main)", ...)
405 dot.260 = %21[128,128][1,0] dot(divide.258, broadcast.259), lhs_contracting_dims=(1), rhs_contracting_dims=(1),
406 constant.261 = %21[] constant(-inf)
407 reduce.260 = %21[128,0] reduce(dot.260, constant.261), dimensions=(1), to_apply
```

GROUP SYNCHRONOUS PARALLEL (GSP)

- Bulk synchronous parallel (BSP) algorithms proceed in a series of global supersteps, which consist of concurrent **computation**, **communication**, and **barrier synchronization**.
- The BSP model is well-suited for automatic memory management for distributed-memory computing:
 - decomposition of the problem into parallel jobs
 - allows every participating processor to perform local computations asynchronously
 - this strategy can lead to almost perfect load balancing, both of work and communication
- Grouped synchronous parallel means different subsets of tiles can participate in each sync



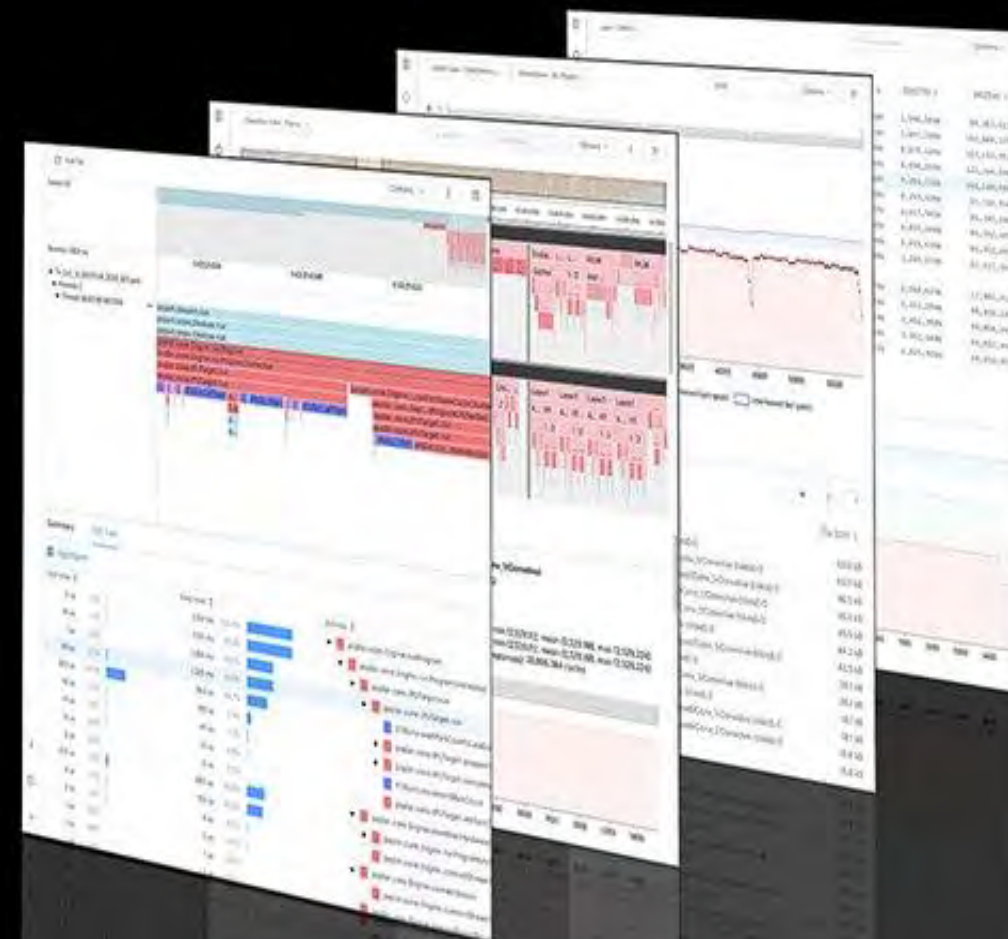
POPVISION™ TOOLS

GRAPH ANALYSER

Useful for analysing and optimising the memory use and execution performance of ML models on the IPU

SYSTEM ANALYSER

Graphical view of the timeline of host-side application execution steps



“Our team was very impressed by the care and effort Graphcore has clearly put into the PopVision graph and system analysers. It’s hard to imagine getting such a helpful and comprehensive profiling of the code elsewhere, so this was really a standout feature in our IPU experience.”

AHEAD OF TIME COMPILATION

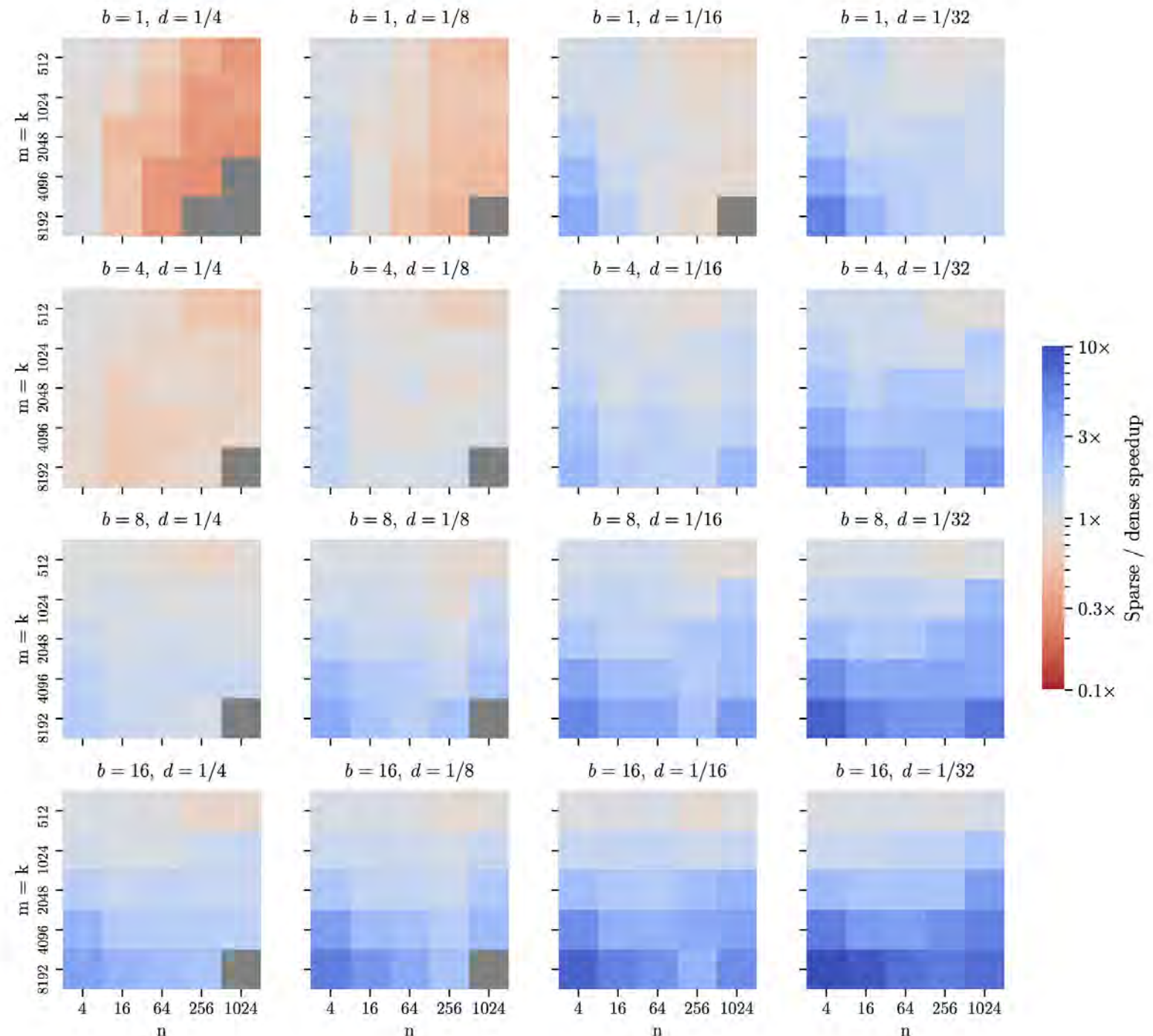
- Makes best use of HW
- Increasingly widely used in ML
 - XLA in TensorFlow and Jax
 - PyTorch 2.0 introduction of `torch.compile()`
- Knowing the whole computational graph allows performing advanced code optimisations
- On the IPUs, tile-level code can be determined at the runtime, based on the input data. However, communication between tiles needs to be compiled ahead of time.

HOW IPU WORKS

SPARSITY

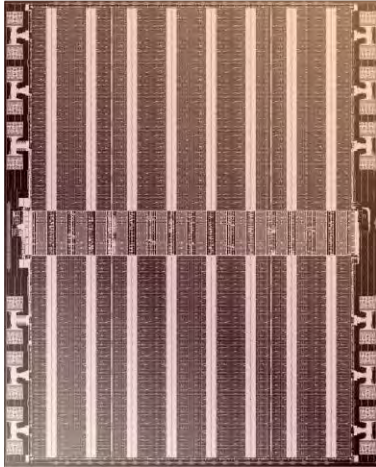
PopSparse API

- provides support for unstructured ($b=1$) and structured ($b>1$) sparsity
- benchmark for sparse-dense matmul either static or dynamic



[2023] Li, Zhiyi, et al. "PopSparse: Accelerated Block Sparse Matrix Multiplication on IPU." <https://arxiv.org/abs/2303.16999>

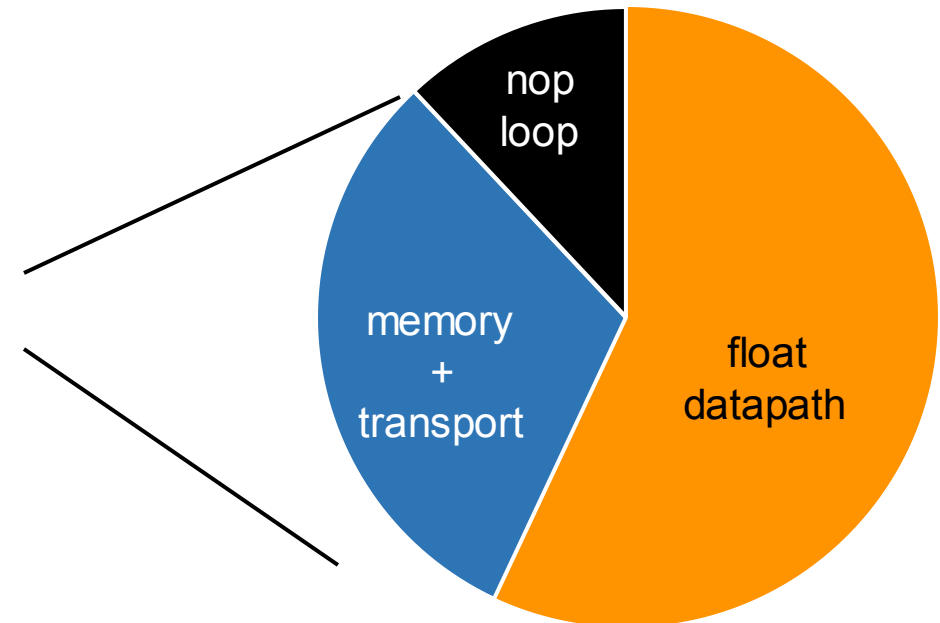
KERNEL POWER



Convolution dynamic power measured at the die level with synthetic data

- Real application data is typically 1/3~1/2 less energetic.
- Power-optimized Mk2 die with wafer-on-wafer DTC decoupler.

Multiply	Accumulate		pJ/flop
	Datapath	Memory	
f16	f32	f16	1.0
	f32	f32	1.3
f32	f32	f32	2.5



FLOAT8

IEEE Standards WG P3109

- ~50% of fp16 energy/flop
- Works for training with managed scaling
- More accurate for inference than int8

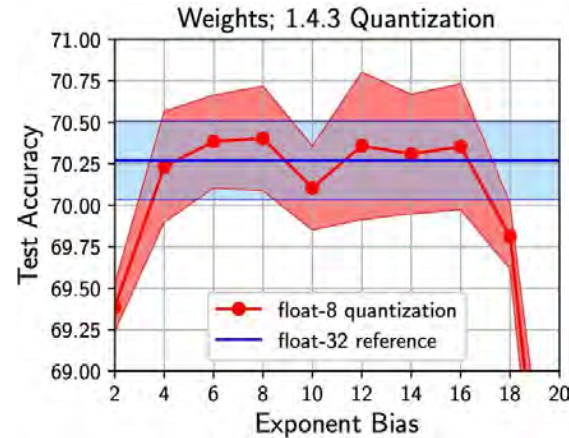
1.4.3 “af8” : 4b exponent, 4b precision

For weights and activations; best accuracy

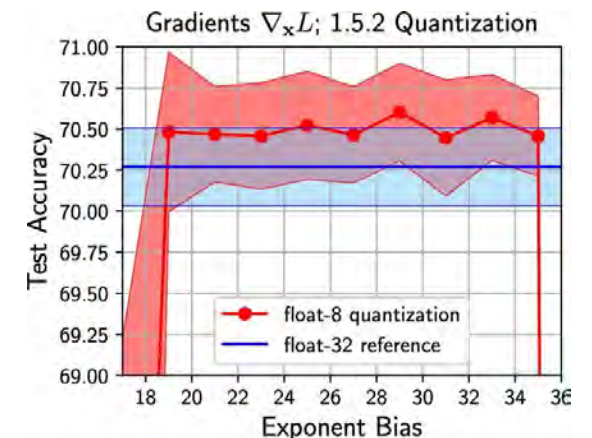
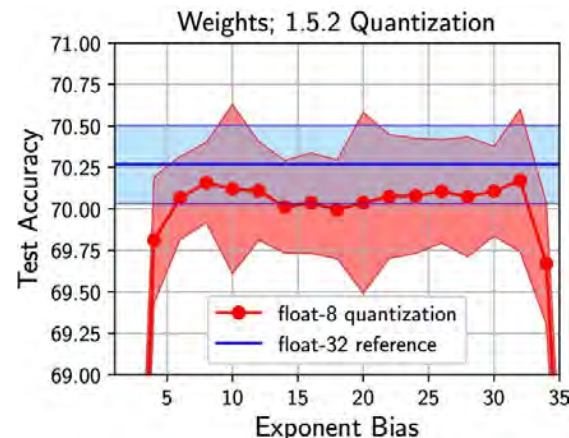
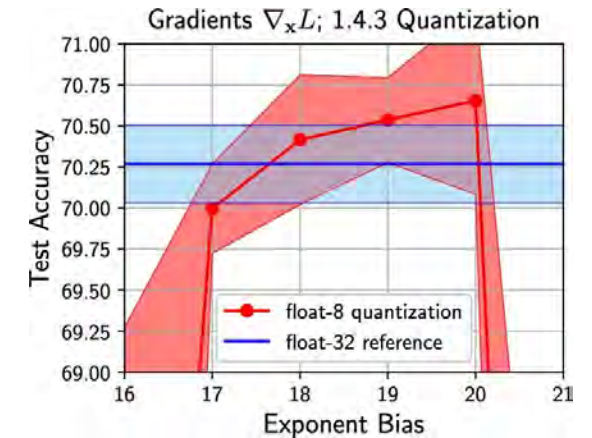
1.5.2 “bf8” : 5b exponent, 3b precision

For gradients; best stability

weights
(activations similar)

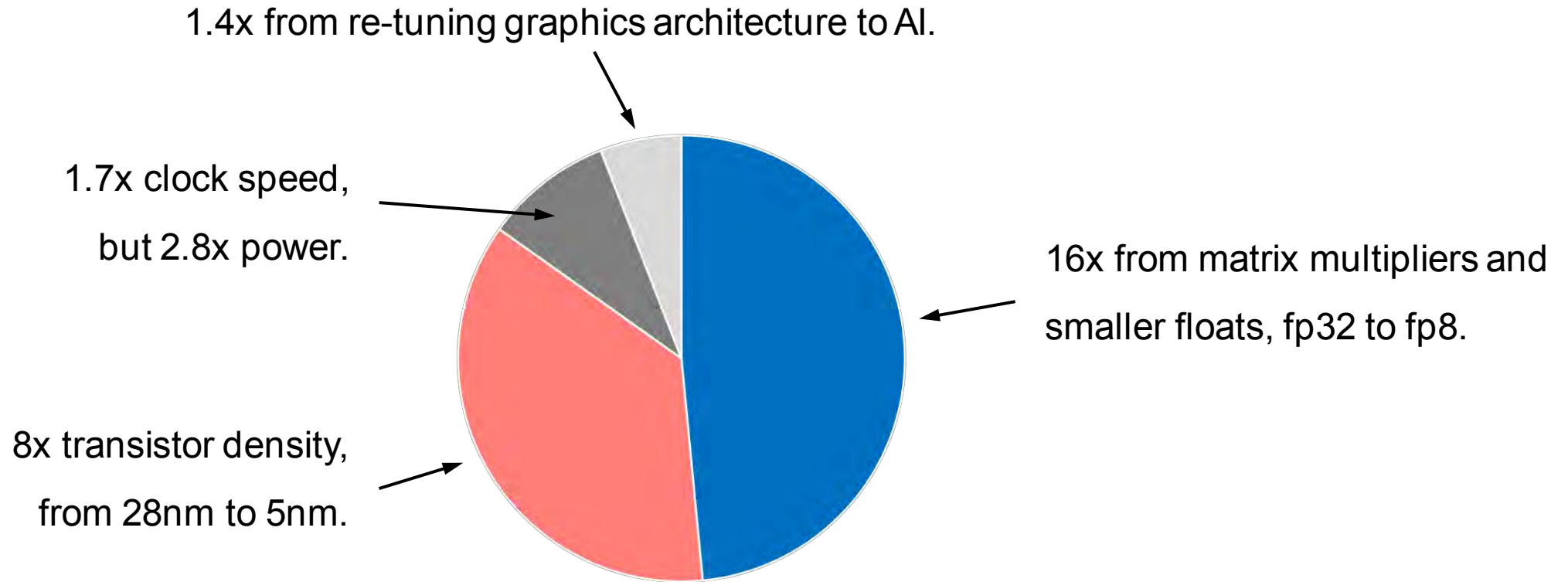


gradA
(gradW similar)



~300X PEAK GPU ARITHMETIC IN THE FIRST AI DECADE

NVIDIA Maxwell 6.6Tflop32/s in 2014 to Hopper 2000Tflop8/s in 2023



THE NEXT DECADE?

More from AI-specific architectures

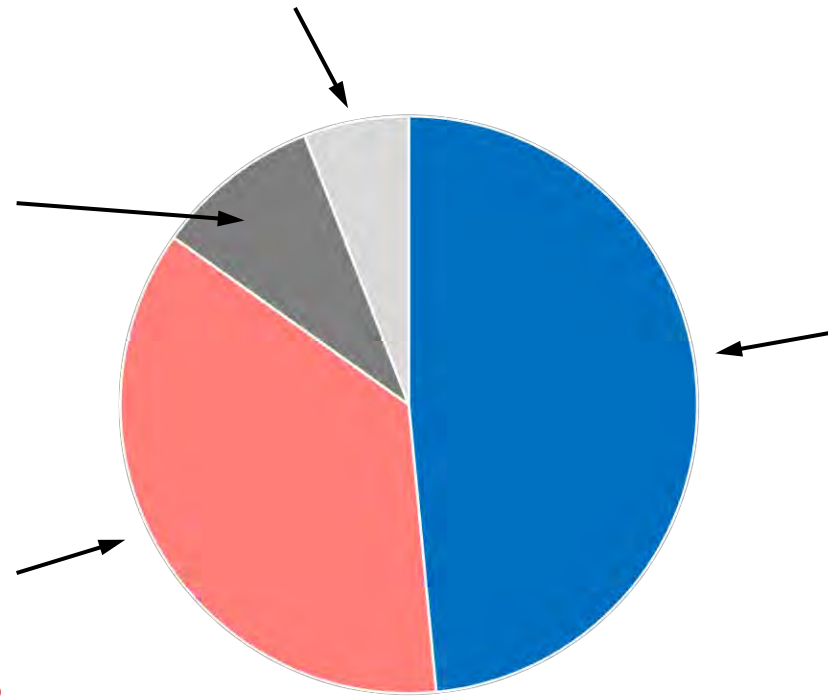
1.4x from re-tuning graphics architecture to AI.

1.7x clock speed,
but 2.8x power.

Another 2x, at 3x power?

8x transistor density,
from 28nm to 5nm.

Another 2-3x?



16x from matrix multipliers and smaller floats, fp32 to fp8.

Done?

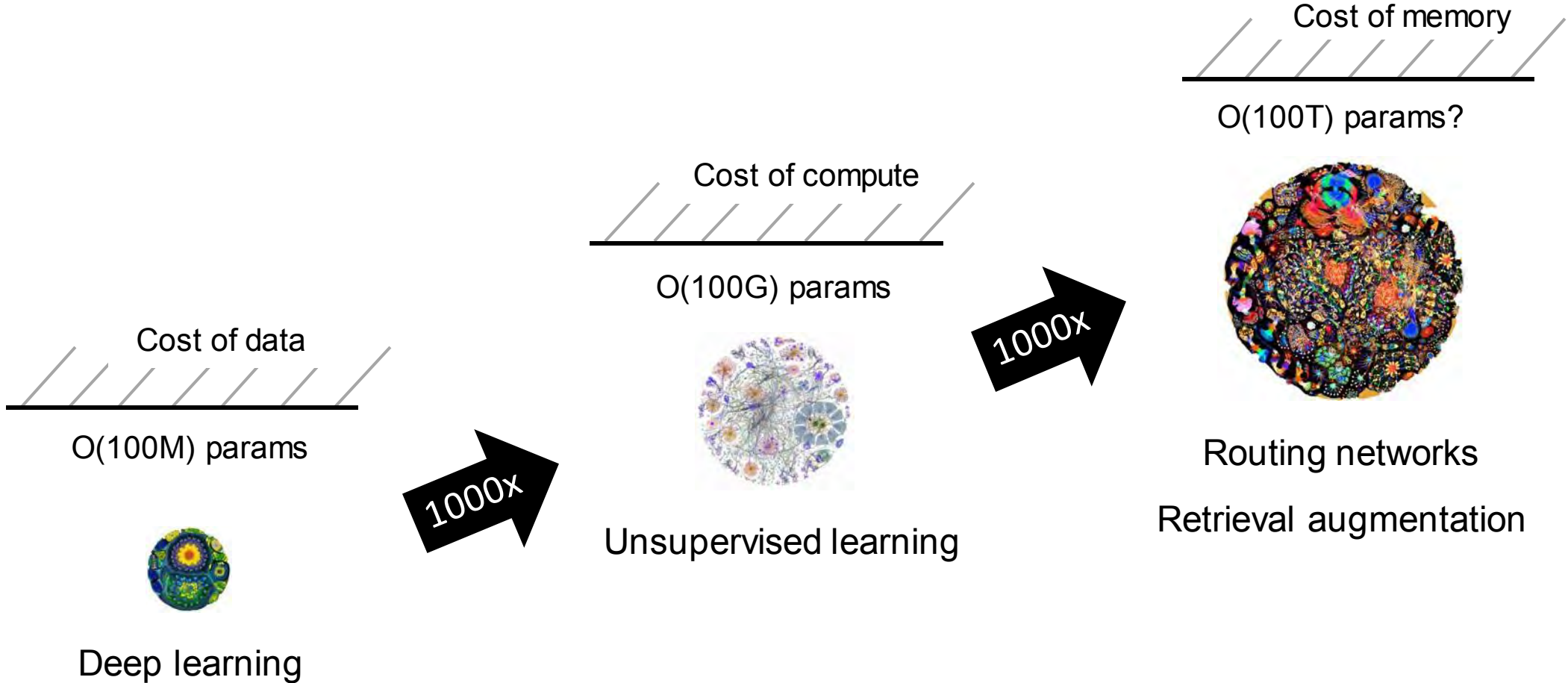
AI ALGORITHMS - STATE OF PLAY

- Dense neural networks are pervasive.
- Useful, efficient models of $O(100\text{m} - 100\text{bn})$ weights.
- Training compute $\sim O(100 \cdot \text{weights}^2)$... Eflop - Yflop
- Inference compute $\sim O(100 \cdot \text{weights})$... Gflops - Tflops
- All signs point to bigger models being more capable

ALGORITHM IMPERATIVE: USE FEWER FLOPS, MORE INFORMATION

- Routing *aka* Conditional Sparsity ... like a brain
- Retrieval Augmentation ... like a human + www

MODEL SCALE BREAKTHROUGHS



AI ACCELERATORS WILL NEED MORE (CHEAP, FAST) MEMORY

SoTA per reticle-sized logic die:

	GB	GB/s	pJ/B	normalized \$cost/B
SRAM over 50% die	1	>> 50,000	<< 1	1
6x HBM3-4800 on silicon substrate	96	3000	40	4
16x LPDDR5-6400 on organic substrate	512	800	50	1
12x DDR5-5600 on 128GB DIMMs	1536	500	300	1.25

...all +0.5pJ/B/mm on die (max 30pJ/B for half-perimeter)

AI ACCELERATORS WILL NEED MORE (CHEAP, FAST) MEMORY

SoTA per reticle-sized logic die:

	GB	GB/s	pJ/B	normalized \$cost/B
SRAM over 50% die	1	>> 50,000	<< 1	1
6x HBM3-4800 on silicon substrate	96	3000	40	4
16x LPDDR5-6400 on organic substrate	512	800	50	1
12x DDR5-5600 on 128GB DIMMs	1536	500	300	1.25

...all +0.5pJ/B/mm on die (max 30pJ/B for half-perimeter)

ROADMAP TO ULTRA-INTELLIGENCE AI

Human brain has around 100 billion neurons

With 100Tn+ synapses, equivalent to parameters in an AI model

Current largest AI models are around 1Tn parameters

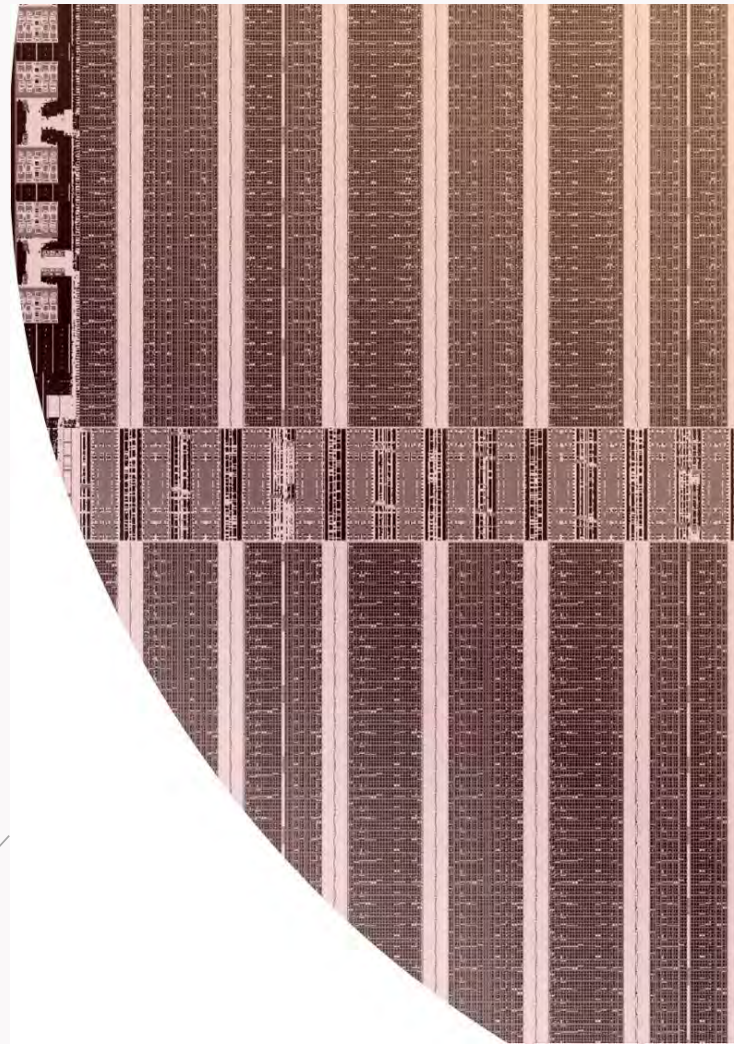
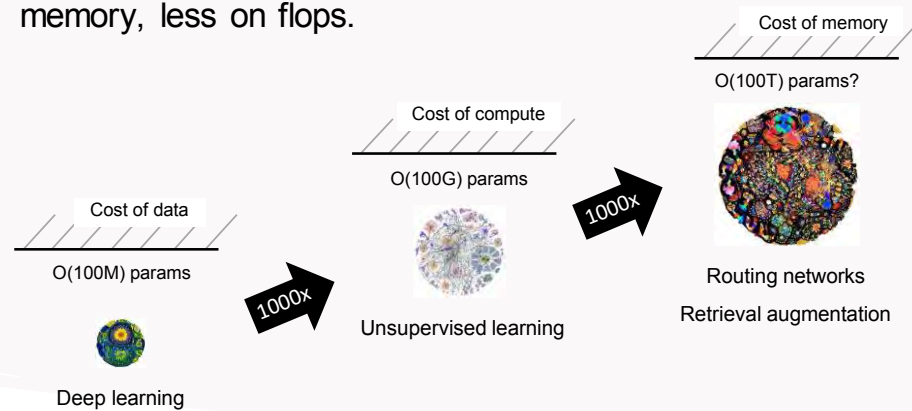
Graphcore is developing an Ultra-Intelligence Machine that will surpass the parametric capacity of the brain



GRAPHCORE

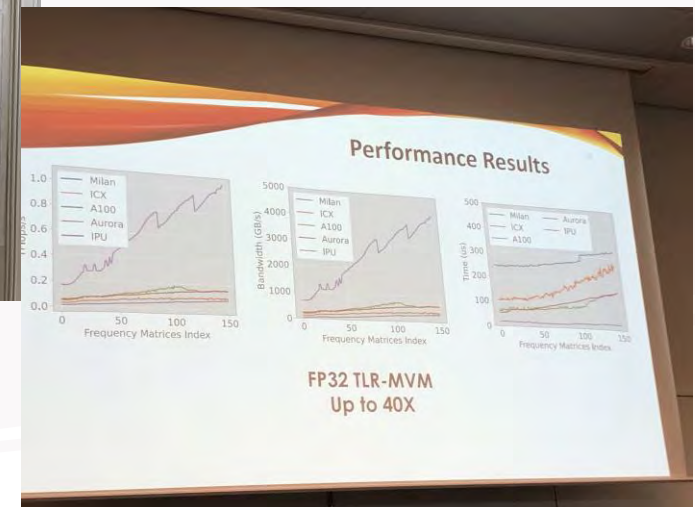
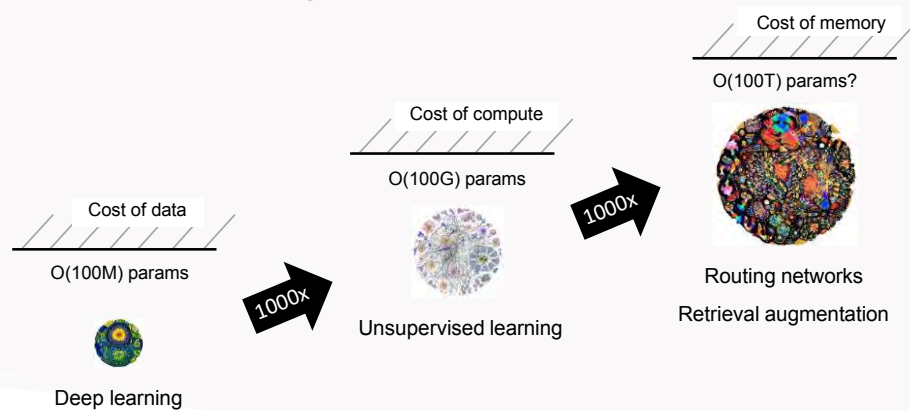
REMEMBER THIS:

- Silicon is approaching “constant energy per op”.
- Information capacity ultimately determines the potency of an AI, given sufficient training.
- AI computing will be limited by power and memory.
- “Human scale” AI is feasible.
- AI algorithm innovation needs to focus more on memory, less on flops.



FEW TAKEAWAYS:

- The AI and HPC research community, OBSPM and KAUST obtained between x20 and x150 perf improvement
- Memory needs have to be harnessed in order to regain acquisition and operating cost control is key
- IPU could well be the new golden age of compute and this is designed in Europe !



APPLICATIONS

Very AI Large-Models:

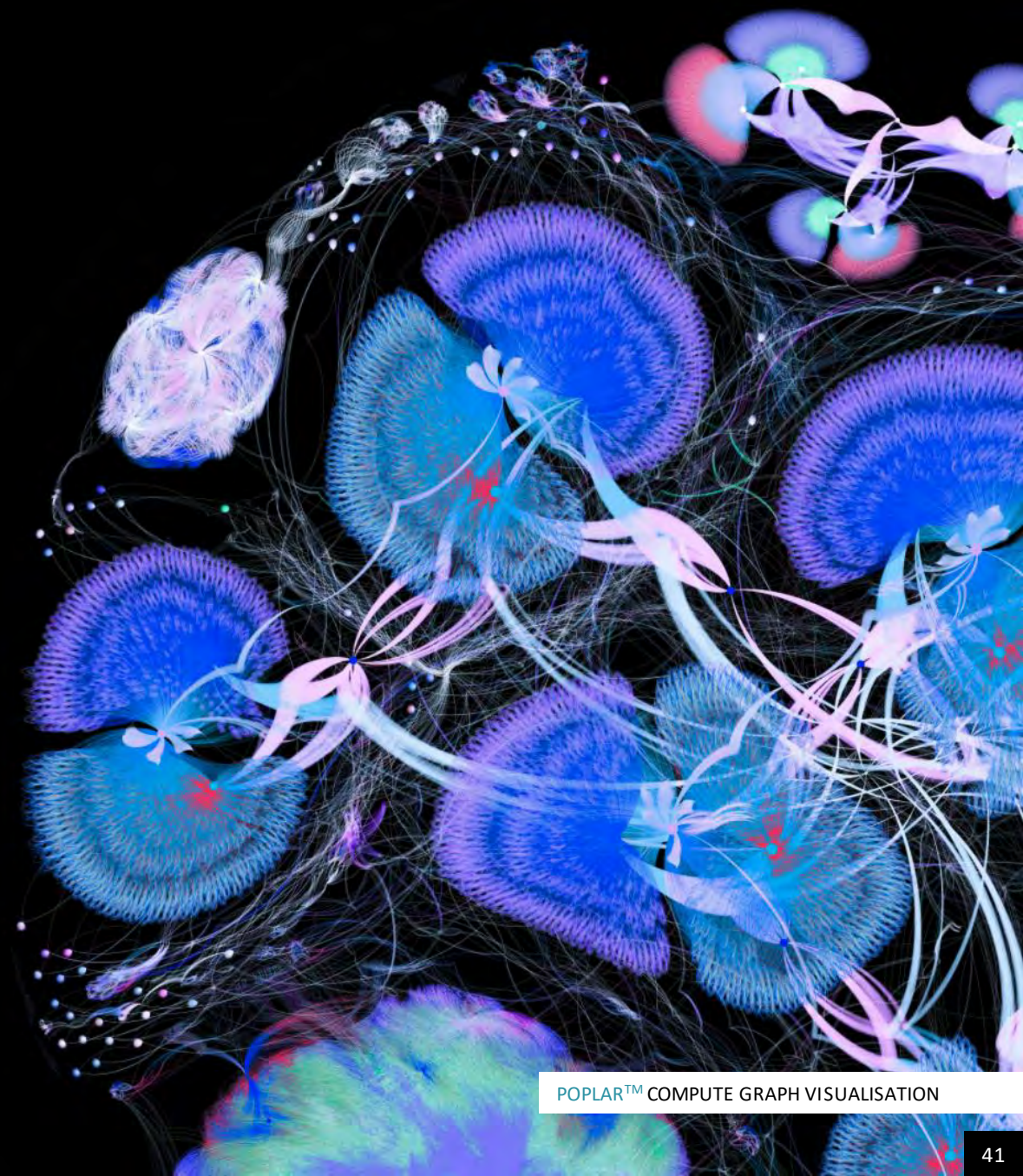
- Multi-trillion parameter model training and inference
- Next-generation, large, conditional and sparse models

AI in Science and Industry

- Healthcare: Genomics | Proteomics | Analysis
- AI-HPC: Simulation | Modeling
- Autonomous systems
- Materials Science | Manufacturing
- Environment: Weather prediction | Smart city

AI in Business

- Language understanding | Process automation | Bots
- Advanced big-data graph analytics and graph databases
- Next generation Recommenders





ACCESS IPU_s IN THE CLOUD

GRAPHCORE



IPU CLOUD SERVICE



On-Demand
IPU Cloud Infrastructure

Scalable, Flexible, Convenient

Europe focus initially

GET UP & RUNNING ON IPUS RIGHT AWAY

AI COMPUTE ON DEMAND

IMPROVE INNOVATION WITH FASTER & LOWER COST
TO TRAIN FOR:
- NLP, CV & GRAPH ML MODELS

EXTENSIVE RANGE OF ML MODELS READY TO RUN
ON IPU

IPU-POD AND BOW POD PLATFORMS AVAILABLE
FROM POD4 THROUGH TO LARGER SYSTEMS

PROVEN IPU ADVANTAGE

IN CASE STUDIES ACROSS MANY INDUSTRIES



DRUG DISCOVERY
DISCRIMINATIVE NLP

[blog](#)



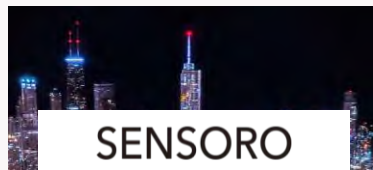
FINANCE
OPTION PRICING

[blog](#) [paper](#) [code](#)



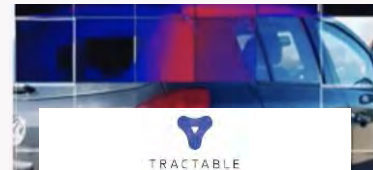
AI SAAS
TEXT ANALYTICS

[blog](#)



SMART CITY
OBJECT DETECTION

[blog](#)



FINANCE
IMAGE CLASSIFICATION

[blog](#)



RESEARCH
AI + HPC

[blog](#)



RESEARCH
WEATHER FORECASTING

[blog](#)



DEFENSE
DECISION MAKING

[blog](#)



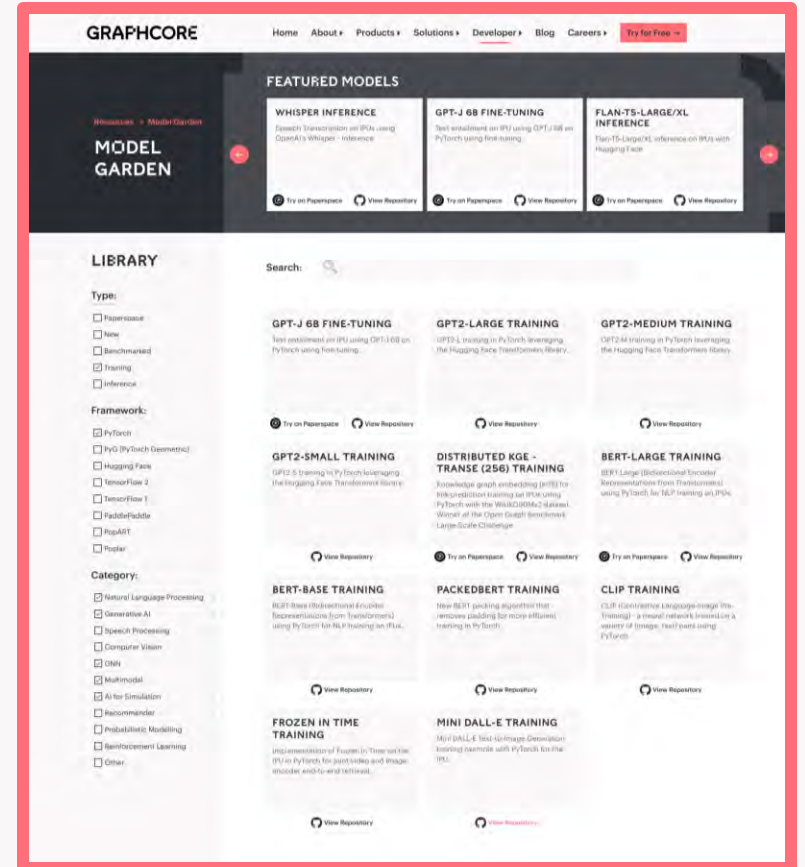
SOCIAL MEDIA
DYNAMIC GRAPHS

[blog](#) [code](#)



RESEARCH
COMPUTATIONAL CHEMISTRY

[blog](#)



graphcore.ai/resources/model-garden



THANK YOU

Contact us

Gautier Soubrane

gautiers@Graphcore.ai





RECENT ANNOUNCEMENTS & CUSTOMER CASE STUDIES

GRAPHCORE



PROVEN IPU ADVANTAGE IN LIFE SCIENCES

1st place
OGB-LSC
quantum property prediction

MOLECULAR MACHINE LEARNING

GPS++
model

SchNet
model

PopTorch
Geometric

paper

blog



GPS++: AN OPTIMISED HYBRID MPNN/TRANSFORMER FOR MOLECULAR PROPERTY PREDICTION

FIRST PLACE SOLUTION TO THE 2022 OPEN GRAPH BENCHMARK - LARGE SCALE CHALLENGE

Dominiac Masters¹ Graphcore, UK, dmasters@gphcore.ai
Joseph Dean¹ Graphcore, London, UK, josephd@gphcore.ai
Kerstin Klaser¹ Graphcore, London, UK, kerstin.k@gphcore.ai
Zhiyi Li¹ Graphcore, Cambridge, UK, zhiyi@gphcore.ai
Madresh-Mander¹ Graphcore, Bristol, UK, smandesh@gphcore.ai
Adam Sanders¹ Graphcore, Bristol, UK, adam@gphcore.ai
Hatem Helal¹ Graphcore, Cambridge, UK, hatemh@gphcore.ai
Demir Bekar¹ Graphcore, Bristol, UK, demirb@gphcore.ai
Felipe Rampoch¹ Universitat de Montreal, rampoch@umontreal.ca
Dominique Beaulieu¹ Valence Discovery, Mtl, Universitat de Montreal, dominique@valencediscovery.ca

ABSTRACT

al report presents GPS++, the first-place solution to the Open Graph Benchmark Large-Scale (OGB-LSC 2022) for the PCQM4Mv2 molecular property prediction task. Our implementation leverages several key principles from the prior literature. At its core GPS++ method implements a hybrid MPNN/Transformer model that incorporates 3D atom positions and an auxiliary denoising technique of GPS++ is demonstrated by achieving 0.0719 mean absolute error on the 1-cent-cha1Lenge PCQM4Mv2 split. Thanks to Graphcore IPU acceleration, GPS++ up architectures (16 layers), training at 3 minutes per epoch, and large ensemble (12 models) the final predictions in 1 hour 52 minutes, well under the 4-hour inference limit. Our implementation is publicly available at: <https://github.com/graphcore/ogb-lsc>.

property prediction, Graph learning, Hybrid MPNN/Transformer, OGB-LSC PCQM4Mv2

development of machine learning on graph-structured data, Open Graph Benchmark (OGB) [Hu et al., 2022] has introduced a variety of graph learning tasks in total, ranging from graph-level prediction, to link-level prediction tasks. Each of these categories has its own challenges, particularly when scaling to larger sets of graphs or to graphs with a considerably larger number of nodes. The task of distal scaling therefore has a major impact on the machine learning method development, necessitating highly impactful applications, OGB Large Scale Challenge (LSC) [Hu et al., 2022] and for the first time task at KDD 2021. In this technical report we present our GPS++ submission as second installment of the challenge, for the graph-level prediction task PCQM4Mv2.

ari [Hu et al., 2021] is specifically aimed at aiding the development of machine learning methods for prediction. The task presented in this report is to predict the HOM0-LUMO energy gap of a molecule, a task calculated using Density Functional Theory (DFT) [Kohn and Sham, 1965]. DFT is the de facto standard for accurately predicting quantum phenomena across a range of molecular systems. Unfortunately, it is extremely computationally expensive, precluding the efficient exploration of chemical space. In this context the motivation for replacing it with fast and accurate machine learning models is clear: does aim to accelerate the development of new methods for DFT it also serves as a proxy for other

COMPUTATIONAL CHEMISTRY

PySCF_{GPU}
library

paper

PySCF_{GPU}: Repurposing Density Functional Theory to Suit Deep Learning

Alexander Mathiasen¹ Hatem Helal¹ Kerstin Klaser¹ Paul Balanca¹ Josef Dean¹ Carlo Lucchi¹ Dominique Beaulieu¹ Andrew Fitzgibbon¹ Dominiac Masters¹

Abstract

Density Functional Theory (DFT) accurately predicts the properties of molecules given their atom types and positions, and often serves as ground truth for molecular property prediction tasks. Neural Networks (NN) are popular tools for such tasks and are trained on DFT datasets, with the aim to approximate DFT as a fraction of the computational cost. Research in other areas of machine learning has shown that generalisation performance of NN tends to improve with increased dataset size; however, the computational cost of DFT limits the size of DFT datasets. We present PySCF_{GPU}, a DFT library that allows us to iterate on both dataset generation and NN training. We create QM10X, a dataset with 10⁶ conformers, in 13 hours, on which we subsequently train SchNet in 12 hours. We show that the predictions of SchNet improve solely by increasing training data without incorporating better inductive biases.

1. Introduction

Density Functional Theory (DFT) is a widely used scientific tool that predicts molecular properties based on each atom's type and position [Kohn & Sham, 1965; Kohn, 1999; Pople, 1999]. It is the means of choice for molecular property prediction as its accuracy is close to gold standard experimental measurement. However, DFT is computationally expensive and recently, neural networks (NN) have become a popular tool to approximate DFT at a much lower cost with comparable accuracy [Gilmer et al., 2017].

A common strategy to improve NN model accuracy is to incorporate inductive biases, e.g. positional encodings or auxiliary losses. However, research in other areas of machine learning like vision [Zhai et al. (2022)] and large language models [Gou et al., 2022] has shown that the inductive biases from chemical space [Dobson, 2004]. In Table 1, we compare the choices made by the authors of QM9, ANI and PCQ. While trade-offs in DFT have been researched for decades, it remains unclear how these distinct decisions impact subsequent deep learning models. Therefore, we trained SchNet [Schütt et al., 2018]) on multiple datasets that were created based on different sets of choices.

models [Kupin et al. (2020)] has shown that more data is crucial in order to unlock the full potential of NN training. Datasets in the molecular domain like QM9 [Ramakrishnan et al., 2014], ANI [Smith et al., 2017] and PCQ [Nakata & Shimazaki, 2017] contain 100k to 20M molecules, which is considered small in the context of machine learning datasets. This lack of data presents a bottleneck in molecular machine learning and prevents foundation models from better understanding chemical space. Therefore, we developed a library that allows us to decrease the time it takes to perform DFT computations to generate significantly larger datasets.

The datasets QM9, ANI, and PCQ contain computed properties using the DFT libraries Gaussian09 and GAMESS [Frisch et al., 2009; Barca et al., 2020]. Both excel at a broad spectrum of computational chemistry tasks for systems with hundreds of atoms. However, QM9, ANI and PCQ contain molecules with at most twenty atoms. We introduce PySCF_{GPU}, a DFT library optimised for small sized chemical systems (8-12 heavy atoms) that we used to create QM10X, a dataset containing 100M examples with 10 heavy atoms.

We generated QM10X using Intelligence Processing Units (IPUs) and find that they are inherently well suited for creating DFT datasets in two distinct ways:

1. IPUs have 948MB memory with 12.65TB/s bandwidth, enough to perform small DFT computations without relying on RAM with < 3TB/s bandwidth.
2. IPUs support Multiple Instruction Multiple Data (MIMD) parallelism which simplifies the computationally demanding Electron Repulsion Integrals (ERIs).

Main Contribution. PySCF_{GPU} is tailored to the unique needs of generating molecular datasets. Creating a DFT dataset requires a number of distinct choices. Notable examples include DFT trade-offs (accuracy vs compute cost) and how to sample from chemical space [Dobson, 2004]. In Table 1, we compare the choices made by the authors of QM9, ANI and PCQ. While trade-offs in DFT have been researched for decades, it remains unclear how these distinct decisions impact subsequent deep learning models. Therefore, we trained SchNet [Schütt et al., 2018]) on multiple datasets that were created based on different sets of choices.

BESS: BALANCED ENTITY SAMPLING AND SHARING FOR LARGE-SCALE KNOWLEDGE GRAPH COMPLETION

FIRST PLACE OGB-LSC 2022 (WIKIKG9Mv2 TRACK)

Alberto Cattaneo¹ Graphcore, UK, albertoc@gphcore.ai

Daniel Joshi¹ Graphcore, UK, danielj@gphcore.ai

Harry Mellor¹ Graphcore, UK, harrym@gphcore.ai

Douglas Orr¹ Graphcore, UK, douglaso@gphcore.ai

Jerome Malberti¹ Graphcore, UK, jerome@gphcore.ai

Zhenyong Liu¹ Graphcore, UK, zhenyong@gphcore.ai

Thomas Farrel¹ Graphcore, UK, thomasf@gphcore.ai

Andrew Fitzgibbon¹ Graphcore, UK, andrewf@gphcore.ai

Blazej Banaszewski¹ Graphcore, UK, blazeb@gphcore.ai

Caig G. Cui¹ Graphcore, UK, caigc@gphcore.ai

November 2022

ABSTRACT

We present the award-winning submission to the WikikG9Mv2 track of OGB 2022. The task is link prediction on the large-scale knowledge graph WikikG9 of 90M+ nodes and 600M+ edges. Our solution uses a diverse ensemble of 51 NN Embedding models combining five different scoring functions (TransE, TransH, ComplEx) and two different loss functions (log-sigmoid, sampled softmax cross-entropy). We trained in parallel on a Graphcore Brevi Pro IPU, using BESS (Balanced Entity Sampling and Sharing), a new distribution framework for KGE training and an balanced collective communications between workers. Our final model achieves a 0.992 and a test-challenge MRR of 0.2562, winning the first place in the comp is publicly available at: <https://github.com/graphcore/distributed-kg-2022-ogb-optimization>.

1 Introduction

Knowledge Graphs encode a knowledge base in the form of a heterogeneous directed graph predicate-object triples which are represented as labelled edges (relations) connecting pairs of past decades they have attracted growing interest, finding a wide variety of commercial applications [Borner et al., 2021] to question-answering [Liu et al., 2017] and recommender systems [Kloppsch et al., 2014]. Knowledge Graph Embedding (KGE) models perform reasoning on knowledge graphs by mapping of entities and relations to low-dimensional vector spaces V_e, V_r , respectively, as triples is measured by a scoring function of the head, relation and tail embeddings $f(V_h, V_r, V_t)$ embeddings can then be used to infer missing links in the graph (Knowledge Graph Completion tasks).

*Equal contribution



1st place
OGB-LSC
link prediction

KNOWLEDGE GRAPHS

TGN model

NBNet model

BESS-KGE library

paper

blog

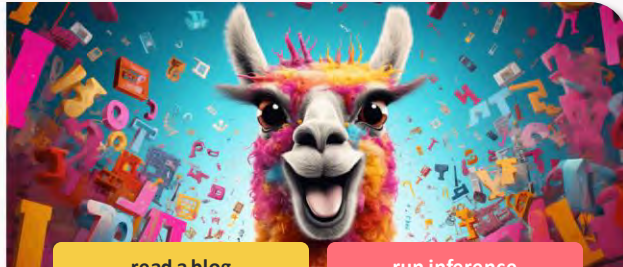
APPLICATIONS

LARGE LANGUAGE MODELS

Aug 01, 2023

LLAMA 2: RUN META'S OPEN SOURCE LARGE LANGUAGE MODEL FOR FREE ON IPUS

Written By:
Tim Santos and Arsalan Uddin



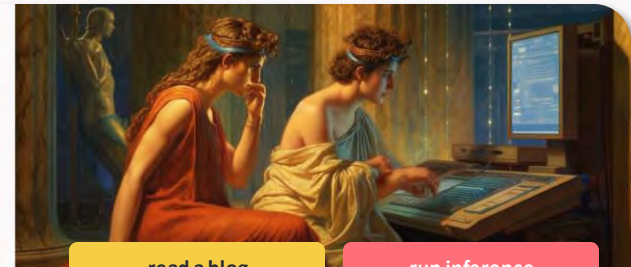
[read a blog](#)

[run inference](#)

May 31, 2023

OPENASSISTANT FINE-TUNED PYTHIA-12B: OPEN-SOURCE CHATGPT ALTERNATIVE

Written By:
Steve Barlow



[read a blog](#)

[run inference](#)

Mar 24, 2023

FINE-TUNE GPT-J: A COST-EFFECTIVE GPT-4 ALTERNATIVE FOR MANY NLP TASKS

Written By:
Sofia Liguori



[read a blog](#)

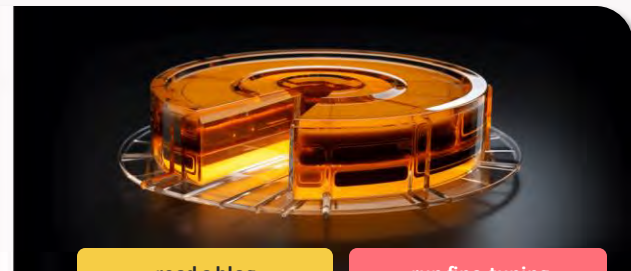
[run text-generation](#)

[run fine-tuning](#)

Jul 27, 2023

FINE-TUNING FLAN-T5 XXL - THE POWERFUL AND EFFICIENT LLM

Written By:
Manuele Sigona

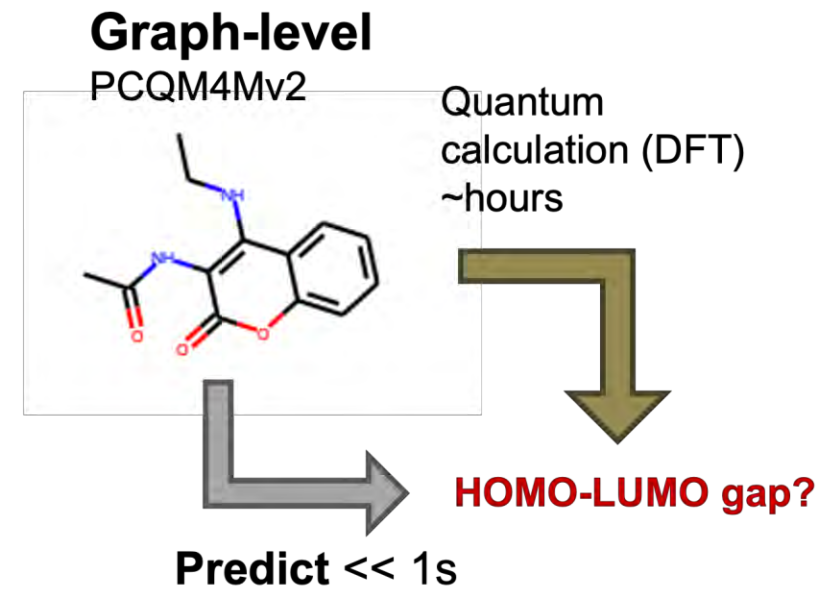


[read a blog](#)

[run fine-tuning](#)

MOLECULAR PROPERTY PREDICTION

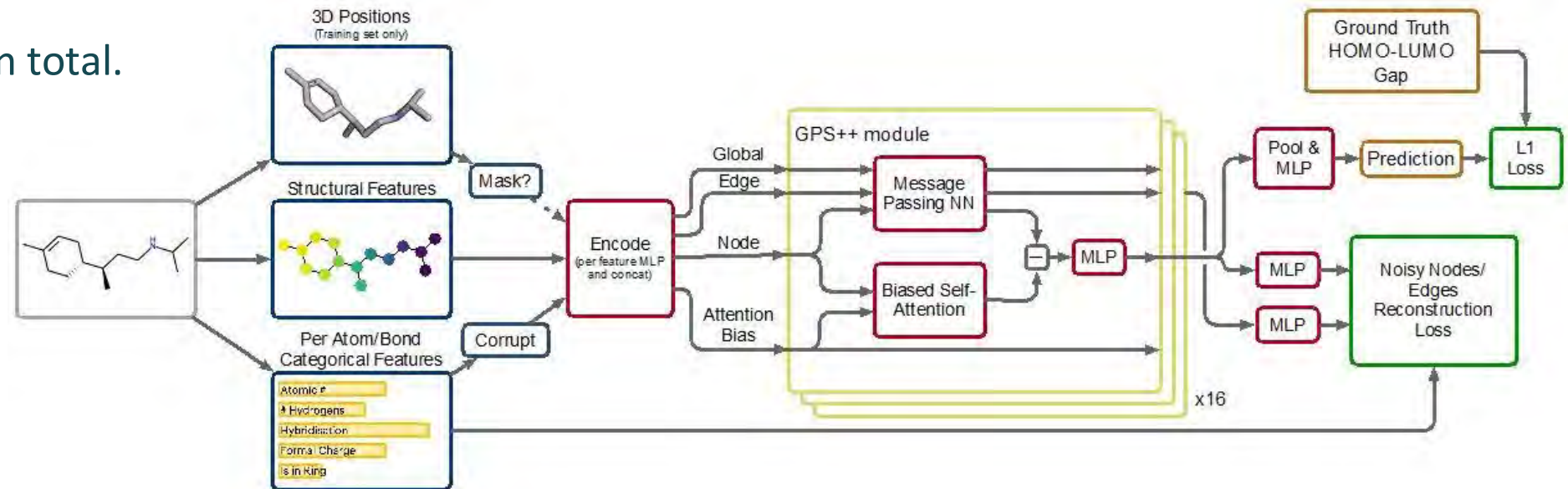
- The PCQM4Mv2 track
 - A quantum chemistry dataset generated by computationally expensive molecular simulation in ~hours/molecule.
 - The task is to predict the HOMO-LUMO gap of a given molecule.
 - The metric used is MAE (Mean Absolute Error).



MOLECULAR PROPERTY PREDICTION

■ GPS++

- An optimised hybrid of MPNN and Transformer for molecular property prediction.
- Builds on top of General, Powerful, Scalable Graph Transformer (GPS) framework (Rampášek et al. 2022).
- 44M parameters in total.



Run on Gradient

Inference GPS++

Run on Gradient

Training GPS++

MOLECULAR PROPERTY PREDICTION



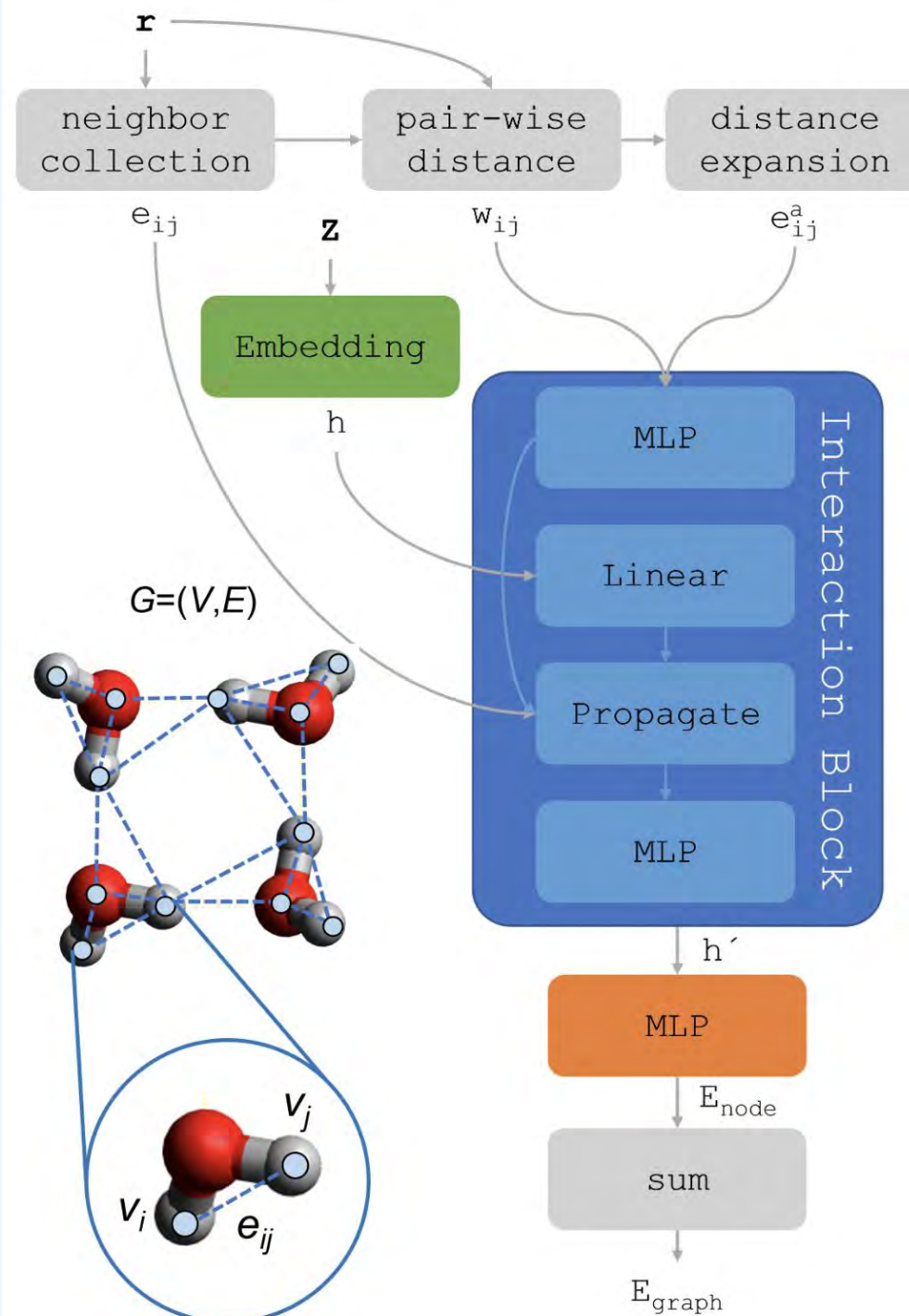
- SchNet – a GNN model for modelling quantum interactions.
- A collaboration with Pacific Northwest National Laboratory (PNNL), Department of Energy in the US, on improving the performance of the model.

Run on Gradient

Molecular property prediction using SchNet



github.com/graphcore/examples/gnn/schnet



ARGONNE ADDS BOW POD SYSTEM TO LEADERSHIP COMPUTING FACILITY (ALCF) AI TESTBED



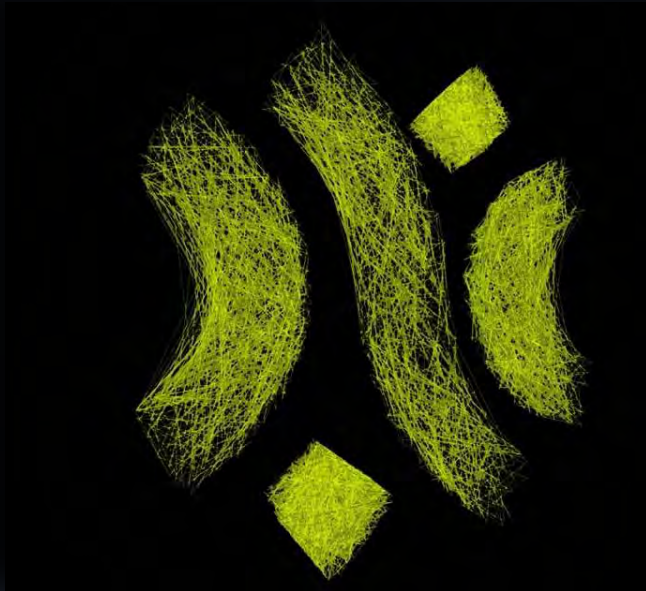
Argonne National Laboratory is adding the latest-generation Bow IPU system to its AI testbed which is enabling pioneering research at the intersection of AI, HPC, and big data, including studies involving climate predictions, drug discovery, and the analysis of large-scale experimental datasets.

“Novel processor architectures like the IPU are facilitating and accelerating new AI techniques and model types. We are excited to work with the research community on ALCF’s latest Bow Pod system to advance AI for science,”

Venkatram Vishwanath, Data Science Team Lead – ALCF

SC22 paper presentation by Argonne covering scientific ML applications (BraggNN & Candle UNO) running on previous generation IPU

ALEPH ALPHA CHATBOT SPARSIFIED & RUNNING ON THE GRAPHCORE IPU





LAWRENCE BERKELEY NATIONAL LABORATORY (LBNL) PAPER PUBLISHED & PRESENTED AT SC22



“Abstract—We compare the ML-training performance of a Graphcore IPU-M2000-based system with Nvidia A100 GPU based system on the Perlmutter HPC machine at NERSC/LBL....”

...

“Finally, we compared the energy consumption and observed that *IPUs used 2.5 to 3 times less energy* while accomplishing the same amount of compute work.”

