If It's Not Deterministic, It's Crap: Deterministic Machine Learning and Molecular Dynamics

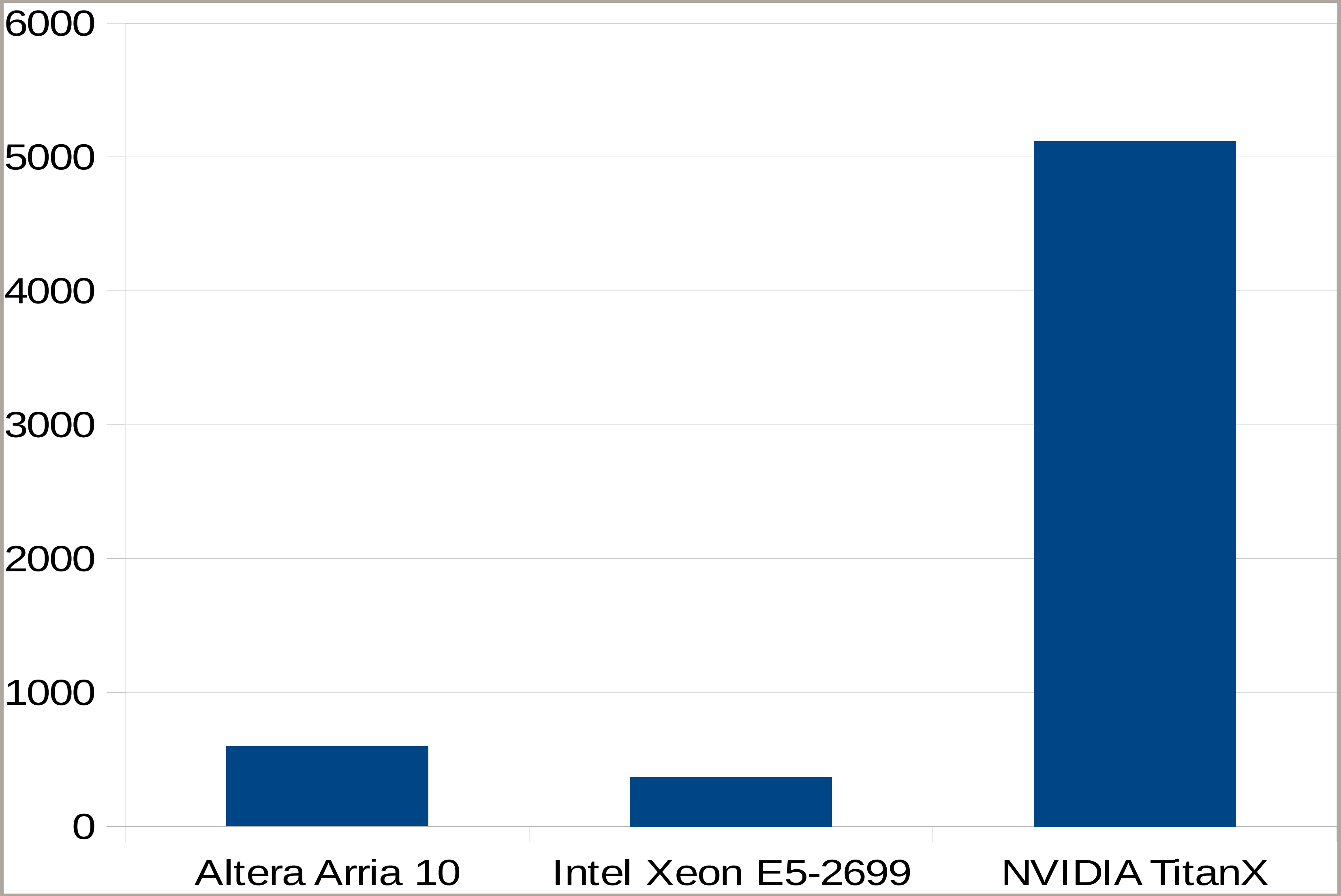
Spoilers

- GPUs/FPGAs/CPUs/ASICs ad nauseum
- AMBER Molecular Dynamics
- Determinism Matters
- Multi-GPU Servers
- Neural Networks
- Deterministic Model Parallelism
- DGX1: $129K of Aspirational Computing for the 1%

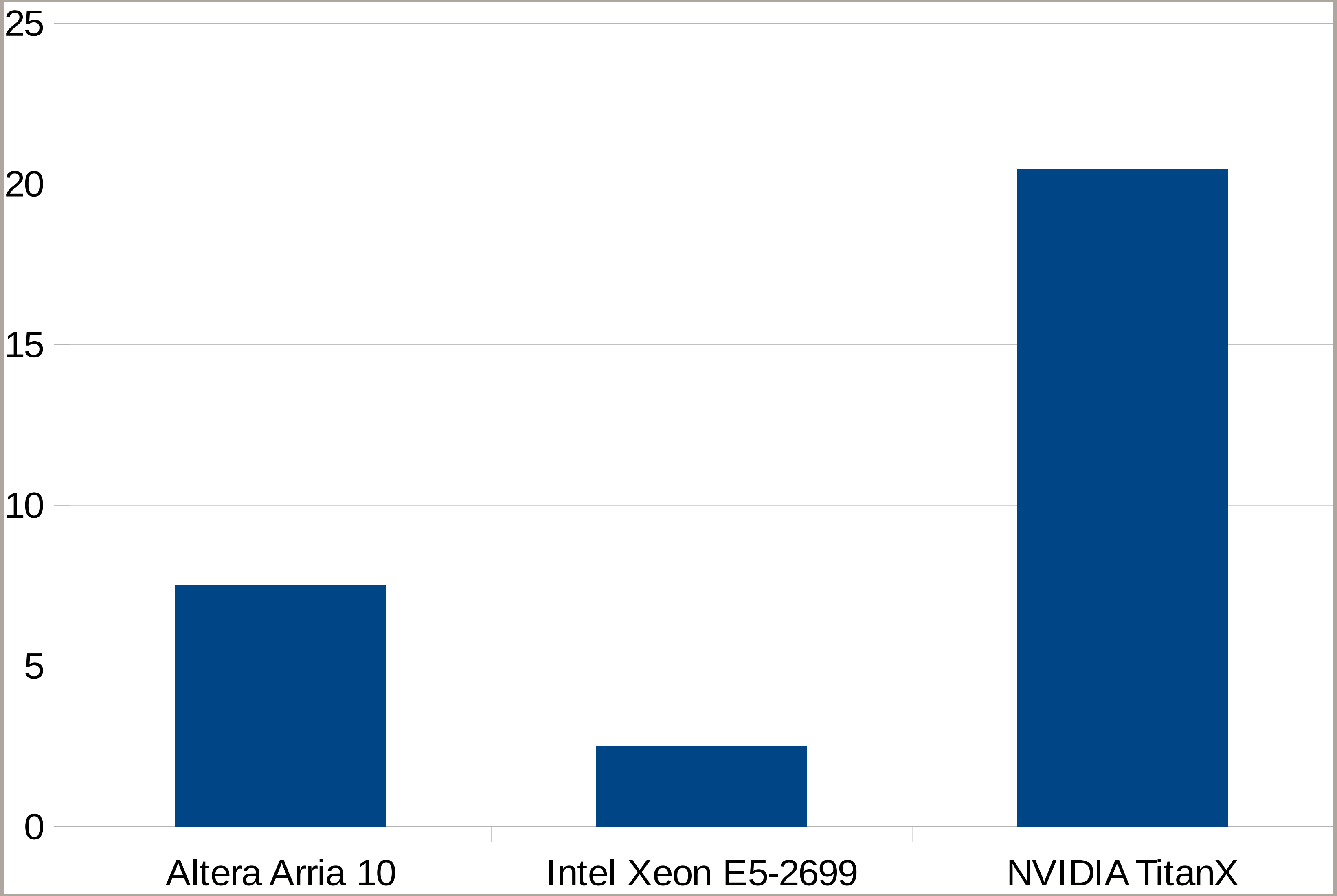
2016 TLDR: It's (still) the GPUs, Stupid

- Despite new hardware from Altera, IBM and Intel, not much has changed
- Intel/Altera training performance sucks
- Intel/Altera prediction performance also sucks (just not quite as much)

AlexNet Images/s

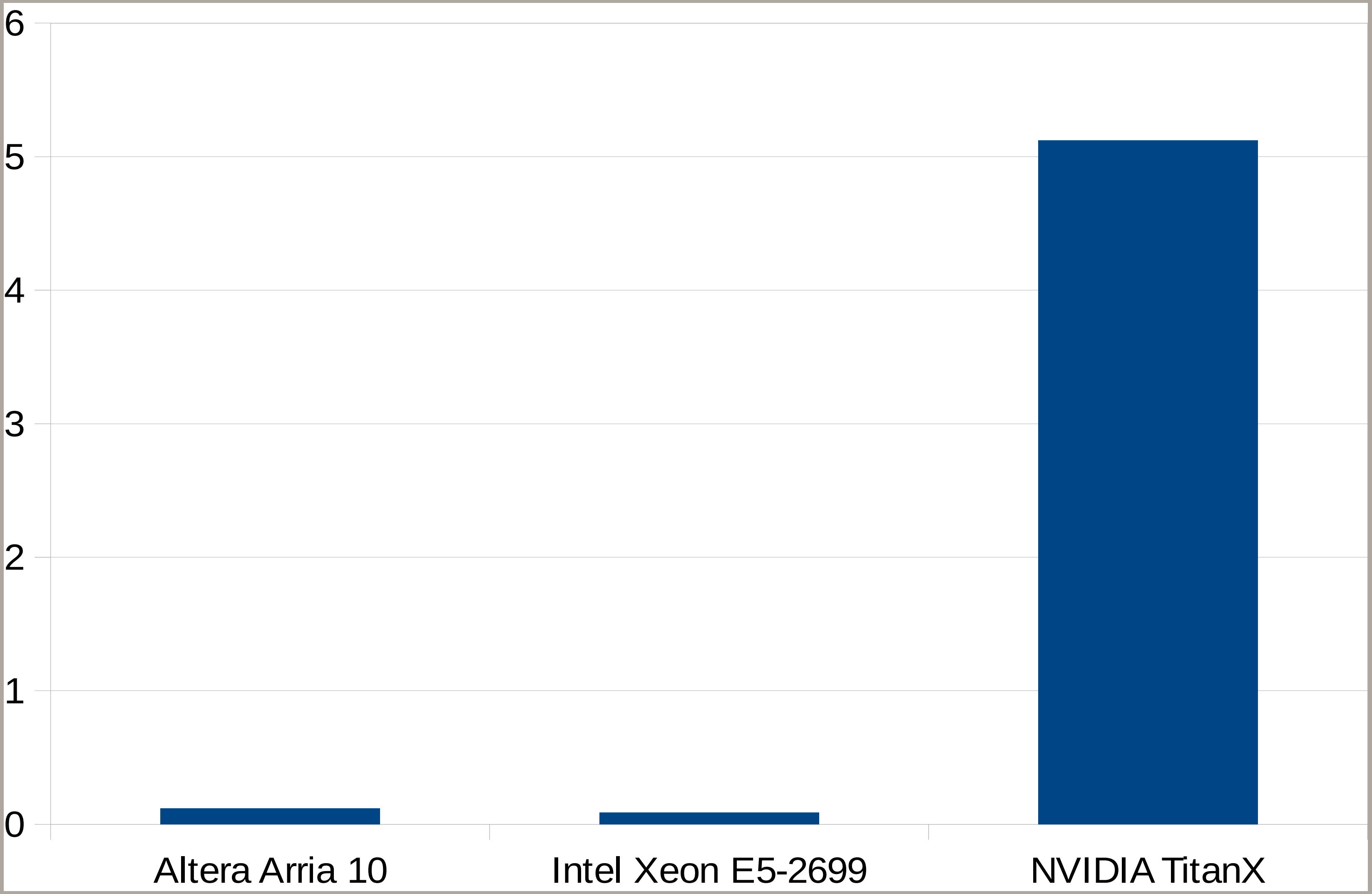


AlexNet Images/Joule*



*Kudos to Ross Walker

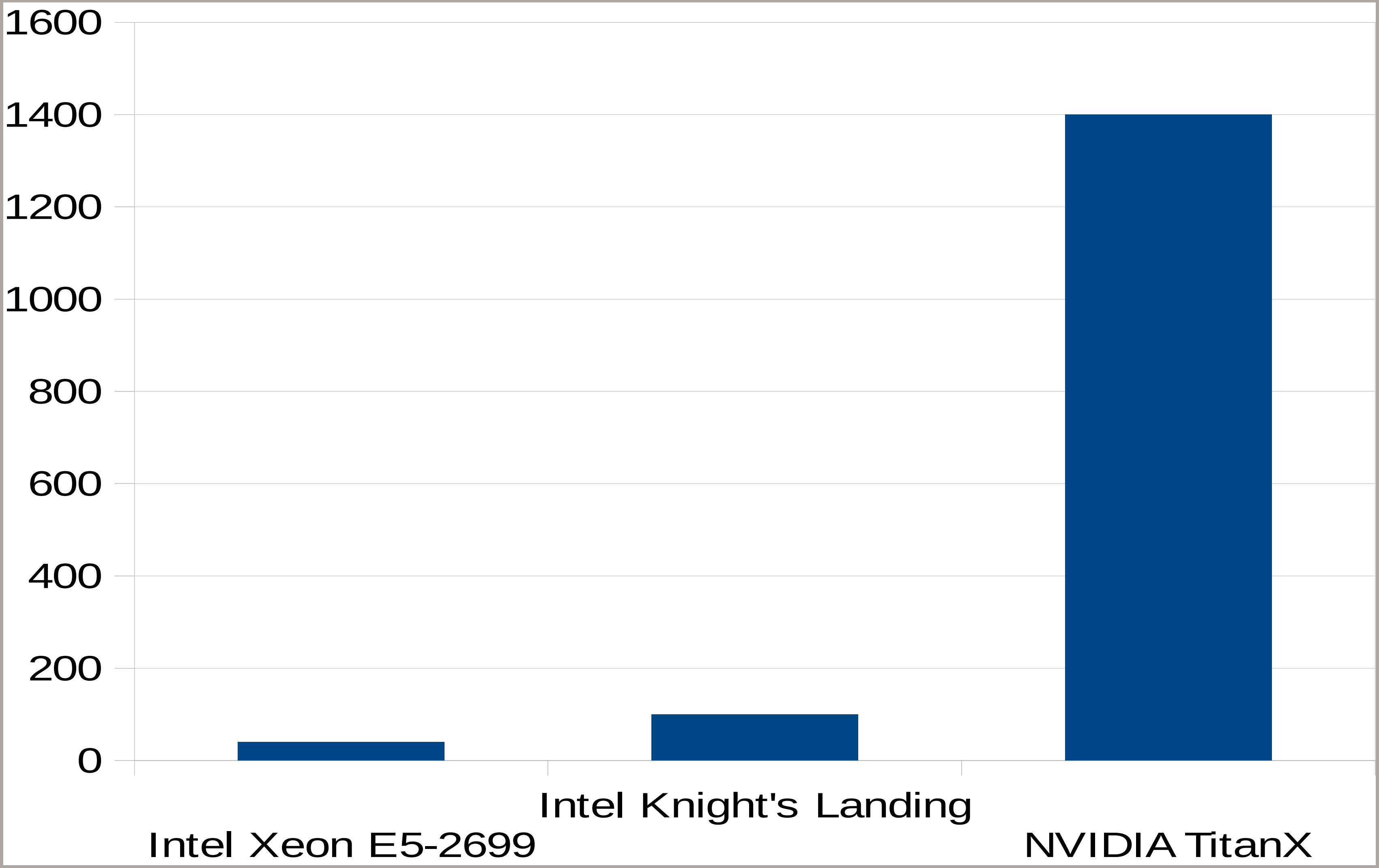
AlexNet Images/s/$



What About Knight's Landing?

- Knight's Landing training performance projected from a HotChips talk (because Intel hates giving out real numbers unless they have to)...
- This is not good news for them, CPU training performance is awful...

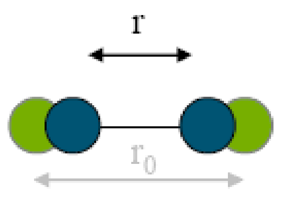
Projected KNL Training Performance

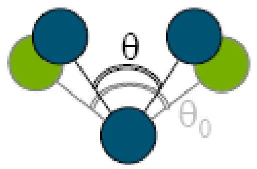


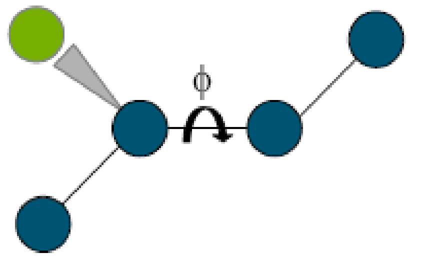
Xeon Phi: A Trail of Tears

- KNL is ~6 TFLOPs, the HW can do a lot better
- But the engineers have been ordered to rely 100% on compiler improvements to implement “recompile and run”
- This is a fool's errand (IMO of course!)
- Nervana, NVIDIA and others have no such constraints
- Recompile and run is a no-win scenario
- Make OpenCL work across CPUs/Xeon Phi/FPGAs
- CUDA/OpenCL subsumes SIMD, multithreading, and multi-core

AMBER Molecular Dynamics







$$E_{stretching} = \sum_{1,2\,pairs} K_r (r - r_0)^2$$

$$E_{bending} = \sum_{1,2\,pairs} K_\theta (\theta - \theta_0)^2$$

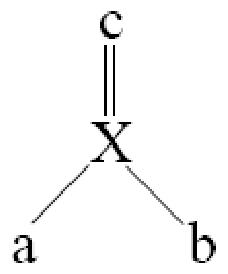
$$E_{torsion} = \sum_{1,4\,pairs} K_\phi (1 - \cos(n\phi - \delta))$$

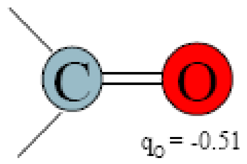
$$\boxed{E = E_{stretch} + E_{bend} + E_{torsion} + E_{impropers} + E_{electrostatic} + E_{vanderWaals}}$$

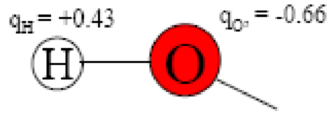
$$E_{impropers} = \sum_{impr.} K_\varphi (\varphi - \varphi_0)^2$$

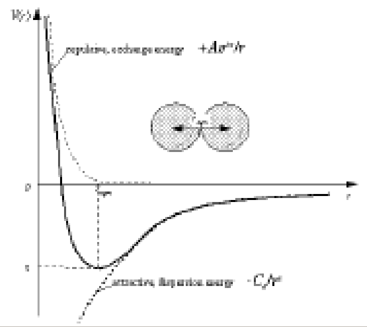
$$E_{electrostatic} = \sum_{\substack{\text{nonbonded}\\ \text{i-k pairs}}} q_i \cdot q_k / D.r_{ik}$$

$$E_{van-der-Waals} = \sum_{\substack{nonbonded\\ pairs}} \left(\frac{\boxed{A_{ik}}}{r_{ik}^{12}} - \frac{\boxed{C_{ik}}}{r_{ik}^{6}} \right)$$









AMBER on GPUs

(or how to play a 30,720 string guitar)

On a CPU, the dominant performance spike is:

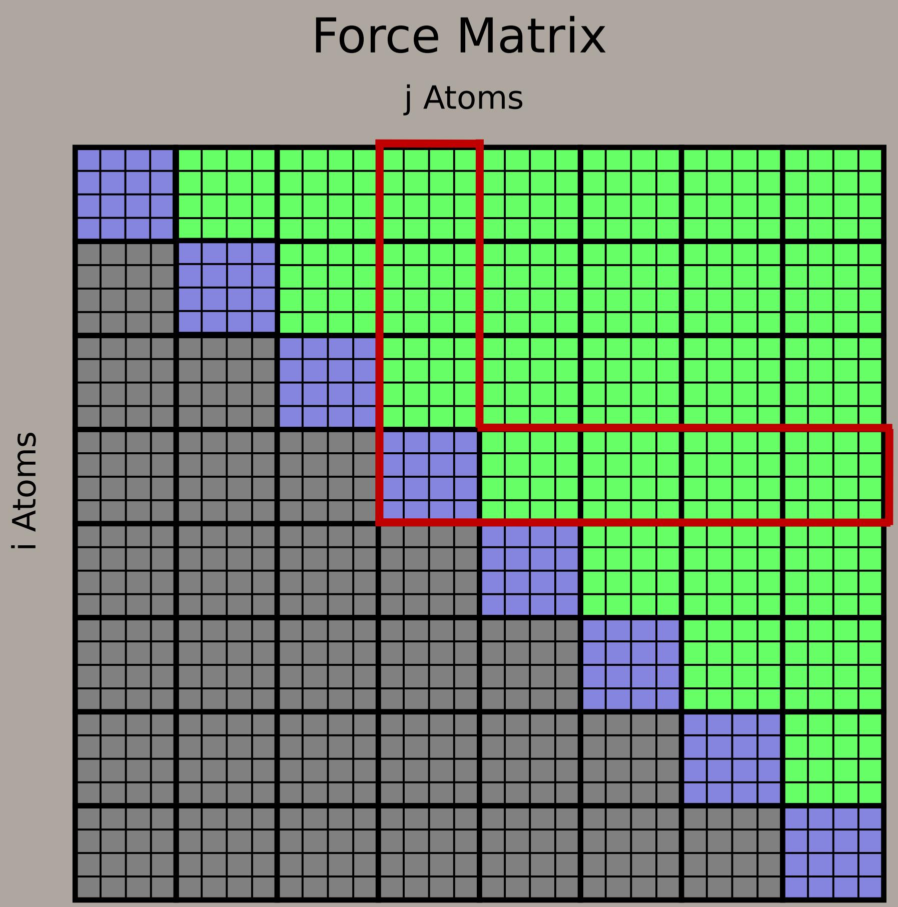
```
for ($i$ =0; $i < N$; $i$++)
    for ($j = i + 1$; $j < N$; $j$++)
        Calculate $f_{ij}$, $f_{ji}$;
```

$O(N^2)$ Calculation

If we naively ported this to a GPU, it would die the death of a thousand race conditions and memory overwrites

Solution: Reinvent mapreduce

Mapreduced Molecular Dynamics



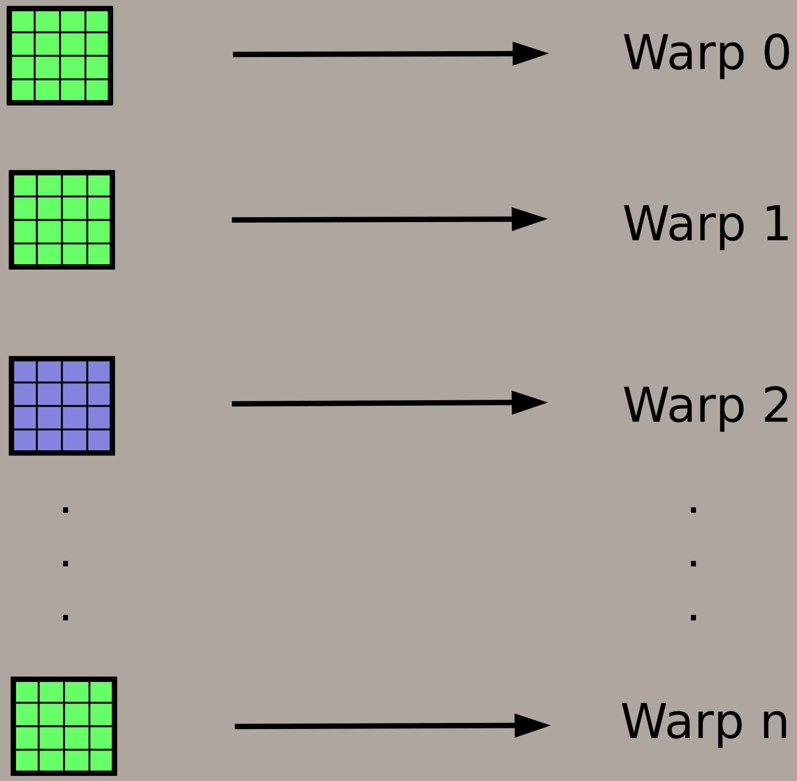
Subdivide force matrix into 3 classes of independent tiles

🟩 Off-diagonal

🟦 On-diagonal

⬛ Redundant

“Map” each nonredundant tile to a warp™



Slow down, what’s a warp?

The smallest unit of execution in a GPU similar to an AVX unit in a CPU

Up through GM2xx, it’s groups of 32 consecutive threads within the same core that execute in lockstep

GPU cores each run 8-64 warps at once on 4-6 vector units

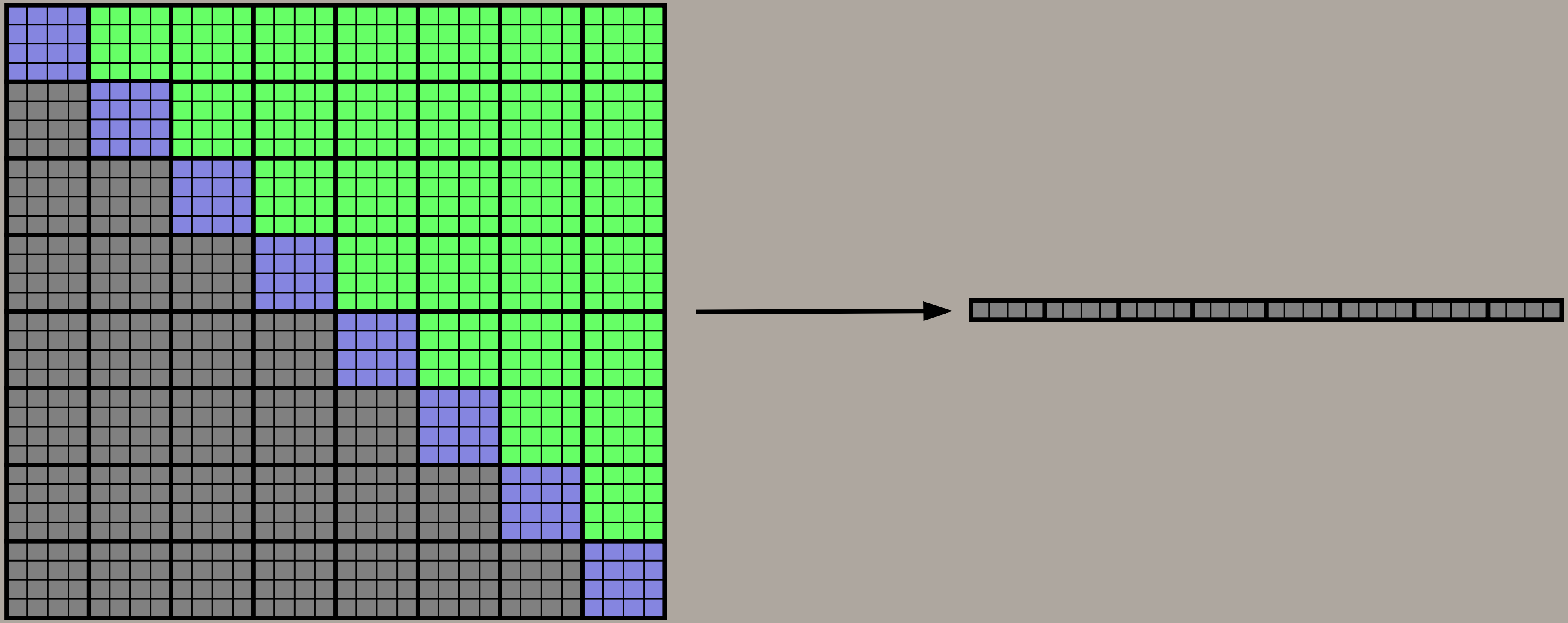
May change in the future

Implements “lock-free computing”

What’s So Special About Warps?

__shfl:	Exchanges data between warp threads
__ballot:	Each bit gives state of a predicate for each warp thread
__all:	True if predicate is true across all warp threads
_any:	True if predicate is true on any warp thread

What About The Reduce Part?



We've “mapped” the force matrix, now we have to “reduce” it to a force vector

Two ways to Reduce

- Execute n separate n-way sums in parallel
- Simple algorithm but it requires $O(N^2)$ memory

- Use Atomic Operations
- No extra memory needed, but floating-point atomic operations are not deterministic

Floating Point Math isn't Associative

A + B == B + A   (Commutative)

A + B + C?   (Associative)

!= B + C + A

!= A + C + B

!= C + B + A

So what? Big deal... Why should we care?

Can you spot the broken GPU/Race Condition/Driver Bug/Thermal Issue/Software Bug?

GPU #1	GPU #2
ETot = -288,718.2326	ETot = -288,718.2326
ETot = -288,718,2325	Etot = -288,718,2326

Let’s make it easier…

GPU #1	GPU #2
ETot = -288,718.2326	ETot = -288,718.2326
ETot = -288,718,2325	Etot = -288,718,2326

Non-Deterministic Accumulation

GPU #1	GPU #2
ETot = -288,456.6774	ETot = -288,458.5931
ETot = -288,453.8133	Etot = -288,454.1539

GeForce GPUs are not QAed for HPC, only gaming…

Dynamic Range and Molecular Dynamics

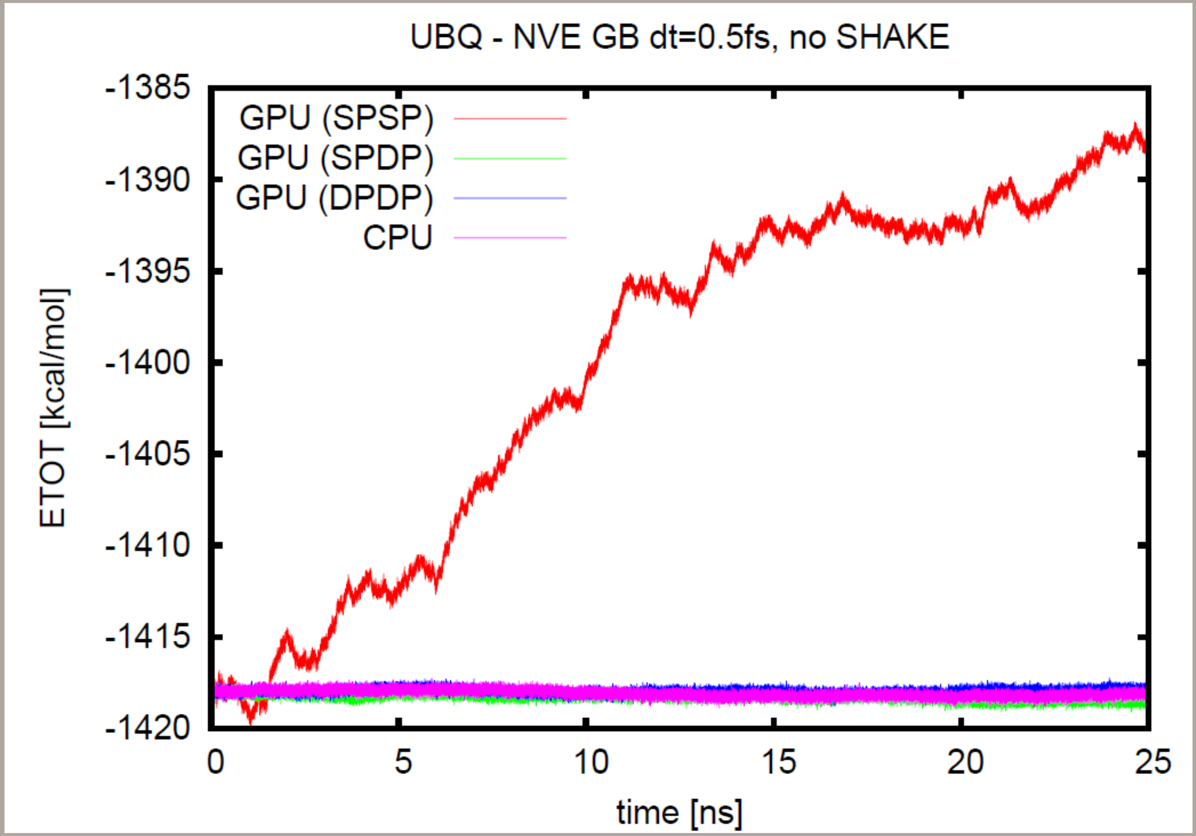
32-bit floating point has approximately 7 significant figures

```
  1.4567020
+0.3046714
---------------
  1.7613730
 -1.4567020
--------------
  0.3046710
Lost a sig fig
```

```
  1456702.0000000
+            0.3046714
-------------------------
  1456702.0000000
 -1456702.0000000
-------------------------
                0.0000000
Lost everything.
```

When it happens: PBC, SHAKE, and Force Accumulation in MD, backpropagation and recurrence in Neural Networks, esp. with FP16 gradients

Dynamic Range Matters



Deterministic Stable MD (using single-precision)

Acceptable force error is ~10^{-5} (as determined by D.E. Shaw)

Single-precision error is ~10^{-7}

So calculate forces in single precision, but accumulate in extended precision

Before Kepler GPUs, we used double-precision and reduction buffers

GK104 (GTX 6xx made it necessary to switch to 64-bit fixed point atomic Adds for accumulation because FP64 perf was reduced to 1/24 FP32

64-bit fixed point deterministic accumulation

Each iteration of the main kernel in PMEMD uses 9 double-precision operations

Fermi double-precision was ¼ to 1/10th of single-precision

GTX6xx double-precision is 1/24th single precision!

So accumulate forces in 64-bit fixed point

Fixed point forces are *perfectly* conserved

3 double-precision operations per iteration

Integer extended math (add with carry) is 32-bit!

Along Came GM2xx

- On GM2xx, double-precision (llrintf) was further reduced to 1/32 that of single-precision whilst nearly doubling attainable single-precision performance (GM200 versus GK110, GM204 versus GK104)
- Initially GM204 is slightly better than GTX 780, GM200 ~20% better than GK110
- Fortunately, we had a solution waiting in the wings that we developed for GK1xx

Use 2 x FP32 (~48-bit FP)

Extended-Precision Floating-Point Numbers for GPU Computation - Andrew Thall, Alma College

http://andrewthall.org/papers/df64_qf128.pdf

High-Performance Quasi Double-Precison Method Using Single-Precision Hardware for Molecular Dynamics on GPUs – Tetsuo Narumi *et al.*

HPC Asia and APAN 2009

Knuth & Dekker Summation

Represent ~FP48 as 2 floats

```
struct Accumulator {
    float hs;
    float  ls;
    Accumulator() : hs(0.0f), ls(0.0f) {}
};
```

Accumulation

```
void add_forces(Accumulator& a, float ys)
{
    float hs, ls, ws;

    // Knuth and Dekker addition
    hs              = a.hs + ys;
    ws              = hs - a.hs;
    a.hs            = ls;
    a.ls            = ys - ws;
}
```

Conversion to 64-bit int

```
long long int upcast_forces(Accumulator& a)
{
    long long int l      = llrintf(a.hs * FORCESCALEF) +
                llrintf(a.ls * FORCESCALEF);
    return l;
}
```

NVIDIA fixes the problem

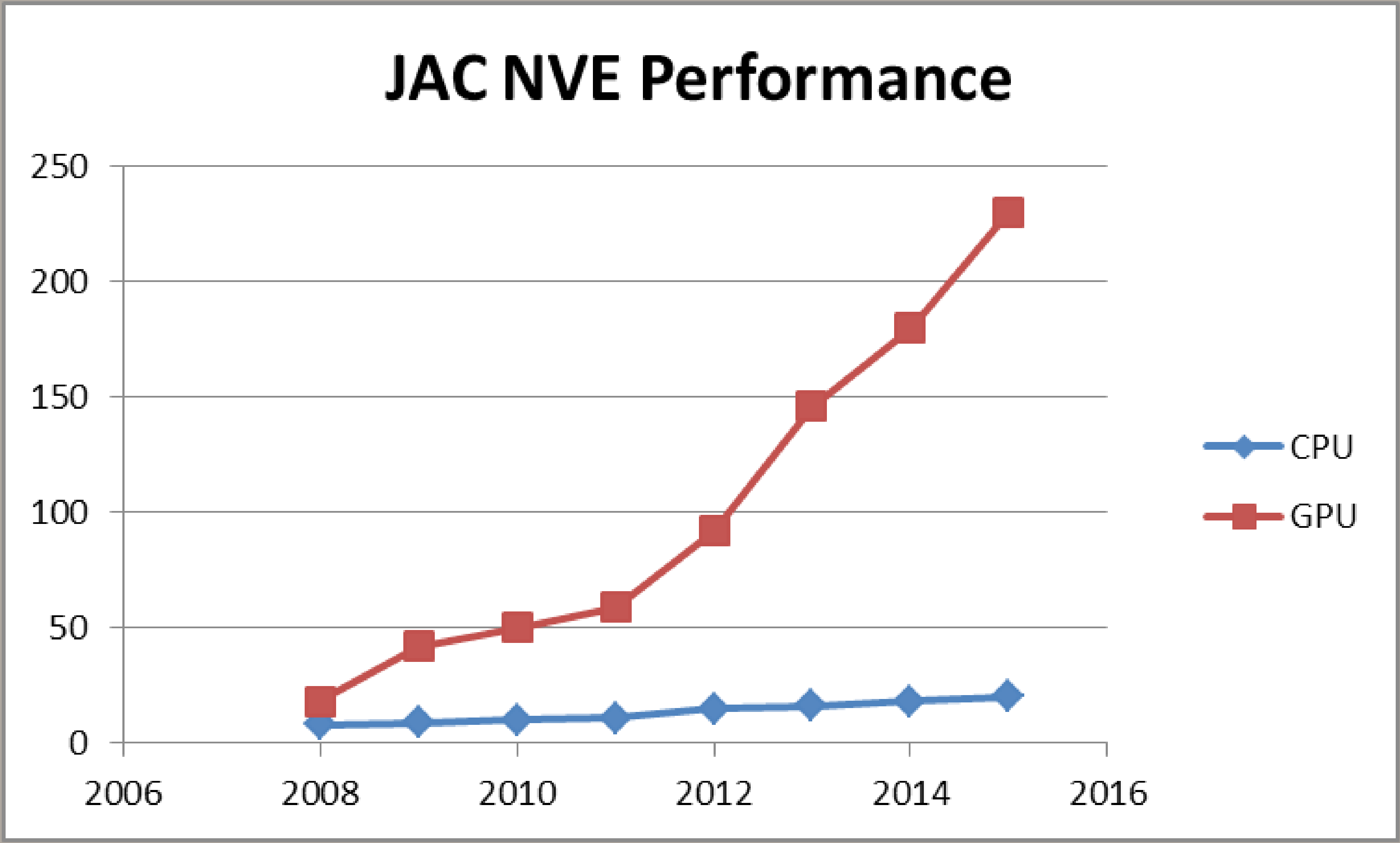
```
long long fast_llrintf(float x) {

    float z = x * (float)0x1.00000p-32;
    int hi = __float2int_rz( z );

    float delta = x - ((float)0x1.00000p32*((float)hi));
    int test = (__float_as_uint(delta) > 0xbf000000);
    int lo = __float2uint_rn(fabsf(delta));
    lo = (test) ? -lo: lo;
    hi -= test;
    long long res =
      __double_as_longlong(__hiloint2double(hi,lo));
    return res;

}
```

AMBER Performance



Summary

- Refactoring Molecular Dynamics into a mapreduce-like task decomposition has allowed performance to scale proportionally to GPU performance
- Refactoring for the next GPU generation is a 1-2 week task based on 7 years and 4 GPU generations
- Much less work than SSE/SSE2/SSE3/SSE4/AVX/AVX2/AVX512 hand-coded intrinsics (IMO of course)

More AMBER?

Speed Without Compromise: Precision and Methodology/Innovation in the AMBER GPU MD Software

Ross Walker, April 7, 10:30 AM right here

CPUs are looking more and more like GPUs

- CPU clocks haven't gone up in significantly in a decade
- Broadwell will have up to 22 physical cores and dual 8-way AVX2 units
- TitanX has 24 cores and 4 32-way vector units
- Later Skylake chips will have Dual AVX 512 units
- GPU-friendly algorithms are AVX-friendly algorithms

Neural Networks*

$$X_{L+1} = X_L \;\; * \;\; W_{L \to L+1}$$

$$\delta_L \;\; = \delta_{L+1} * W_{L \to L+1}$$

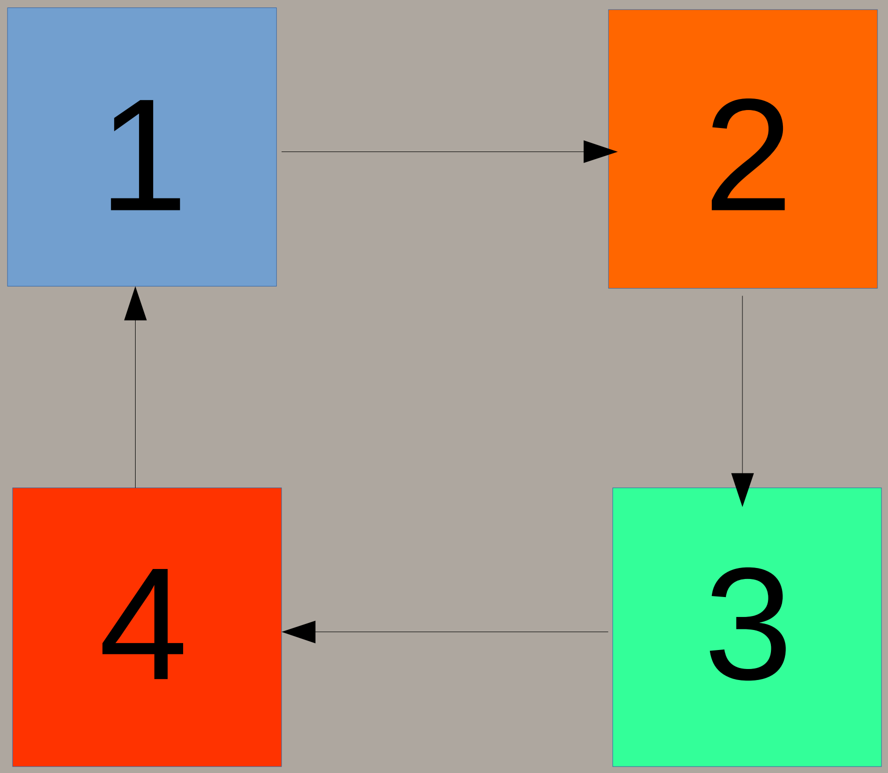
$$\Delta W = X^T_L \;\; * \;\; \delta_{L+1}$$

*The definitive answer to whether you should take Calculus, Statistics and Linear Algebra in college

Model Parallel Training

“My belief is that we’re not going to get human-level abilities until we have systems that have the same number of parameters in them as the brain.” - Geoffrey Hinton

P2P Scatter/Gather Ops 2016*



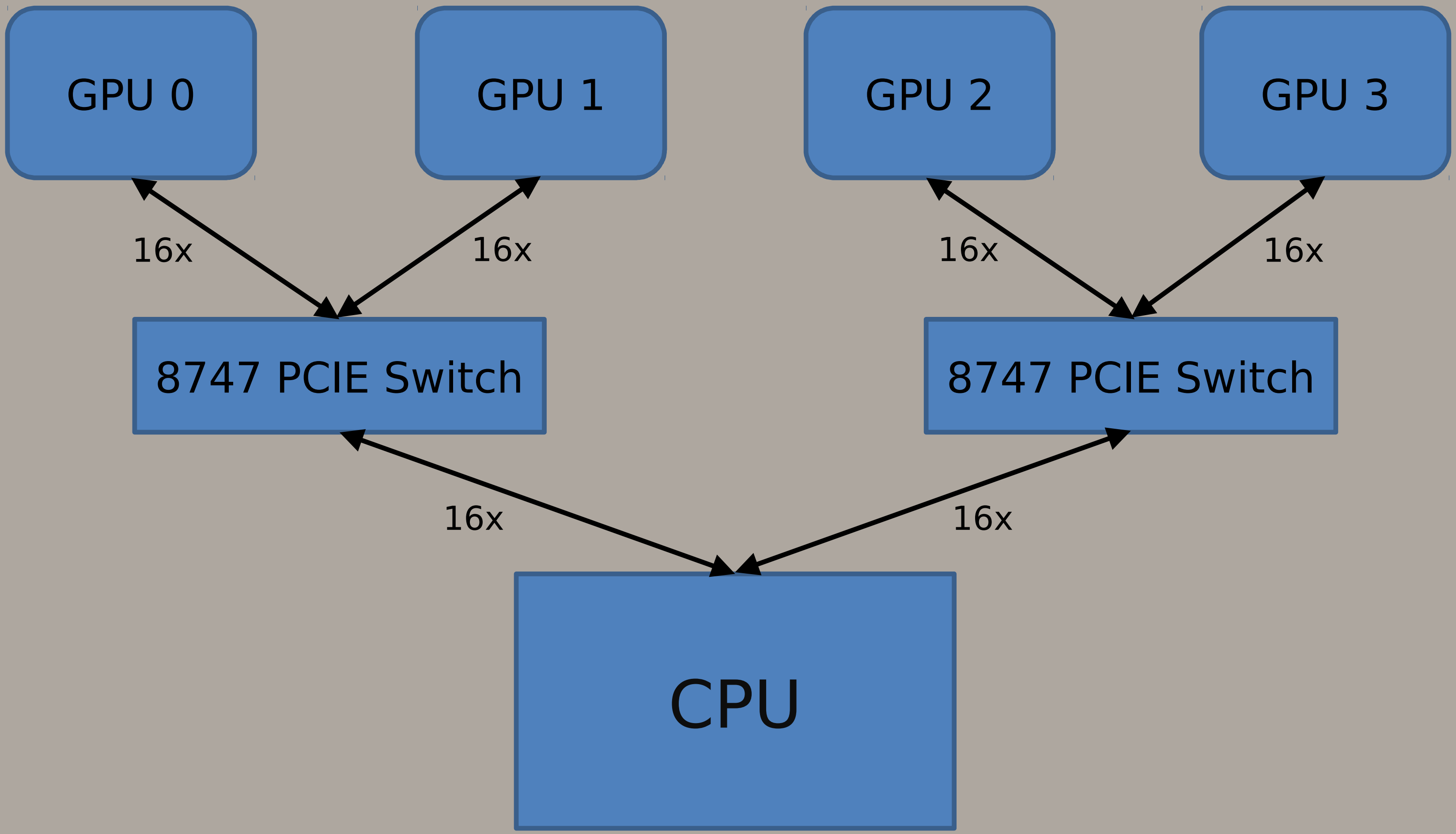
*As seen (but implemented inefficiently) in the NVIDIA NCCL library

P2P Ring Ops Performance*

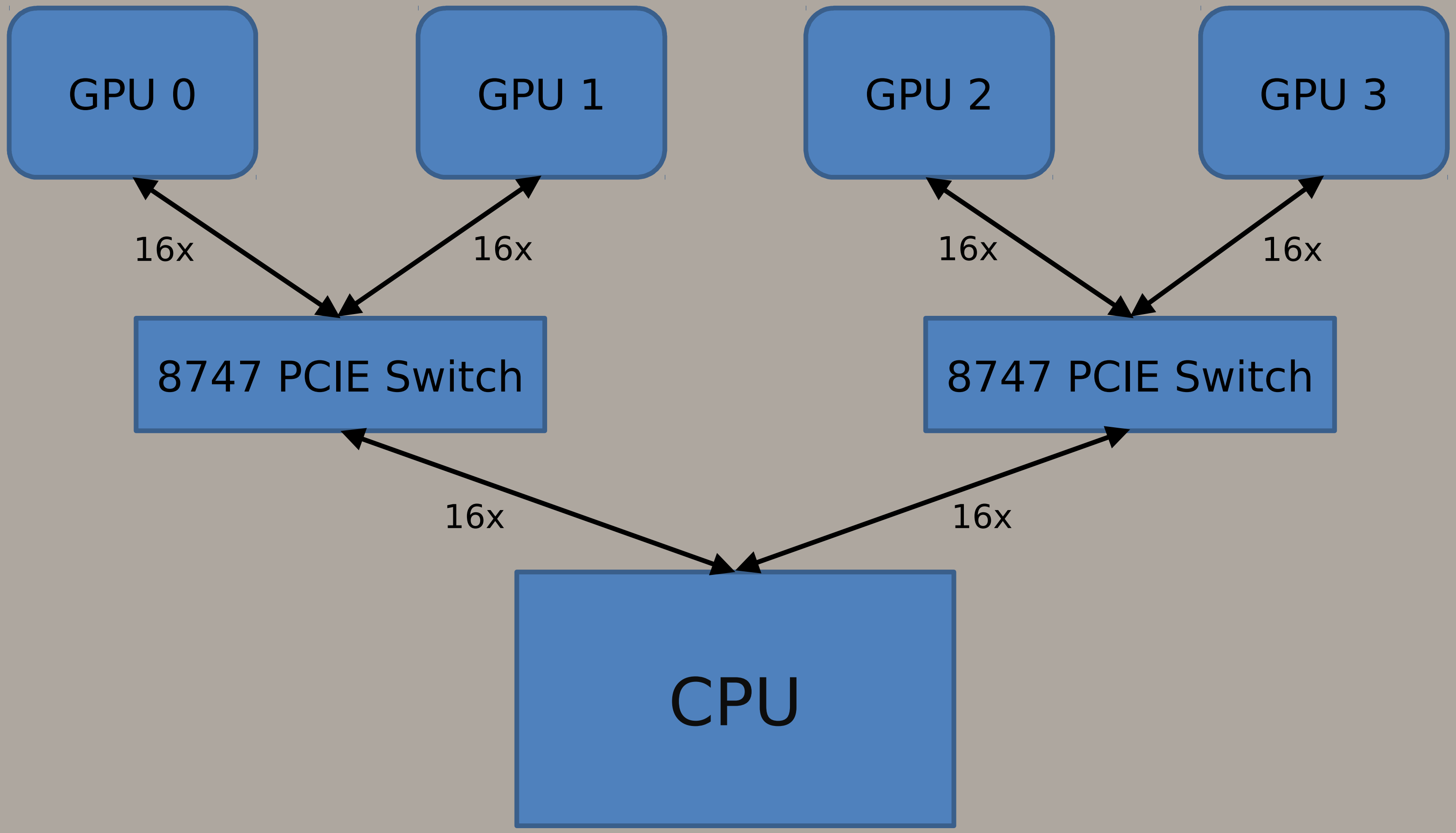
- AllReduce: 2 * D * (N – 1) / N
- Scatter/Gather/AllGather: D * (N - 1) / N
- Reduce: D * (N – 1) / N

*NVLINK makes everything better, but we'll get to that...

The AMBERnator (2013)

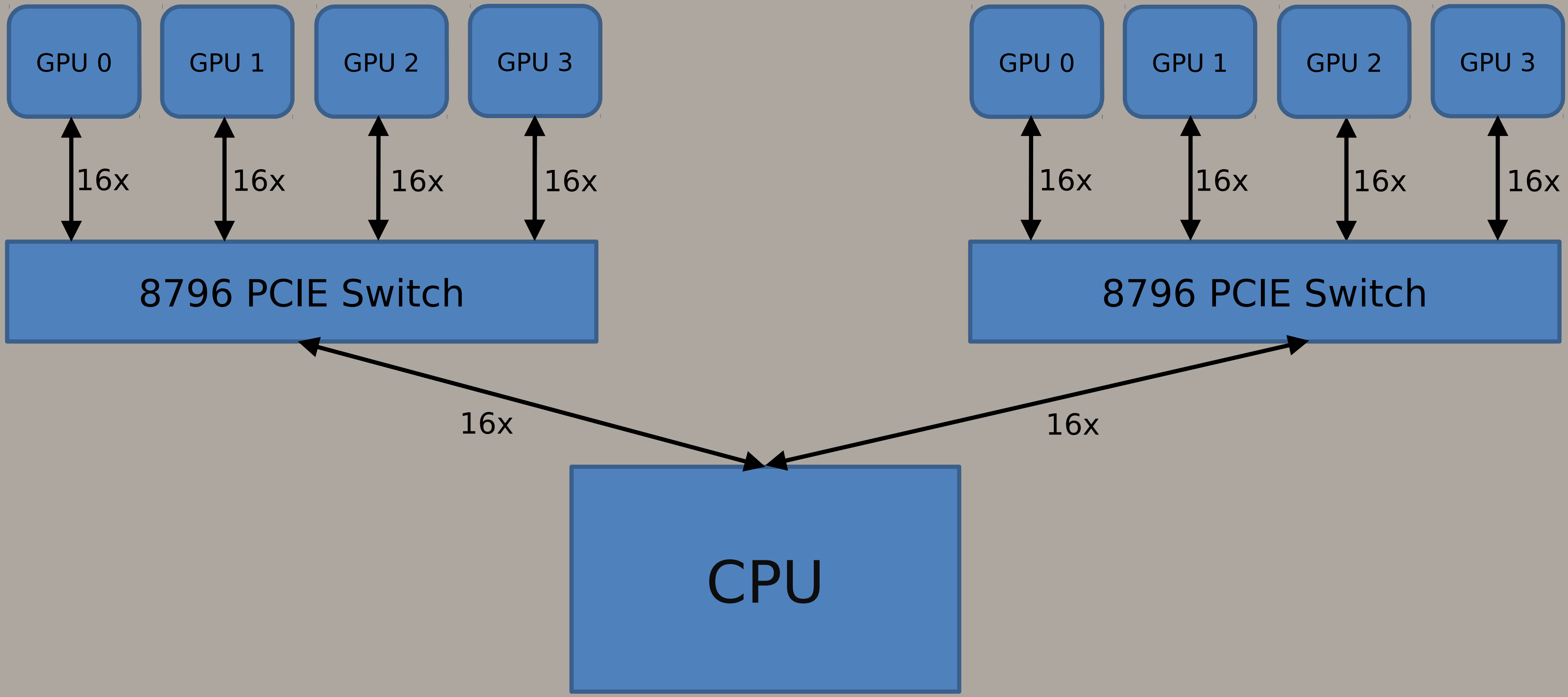


Digits Dev Box (2015)*



*Maybe you can tell me the difference?

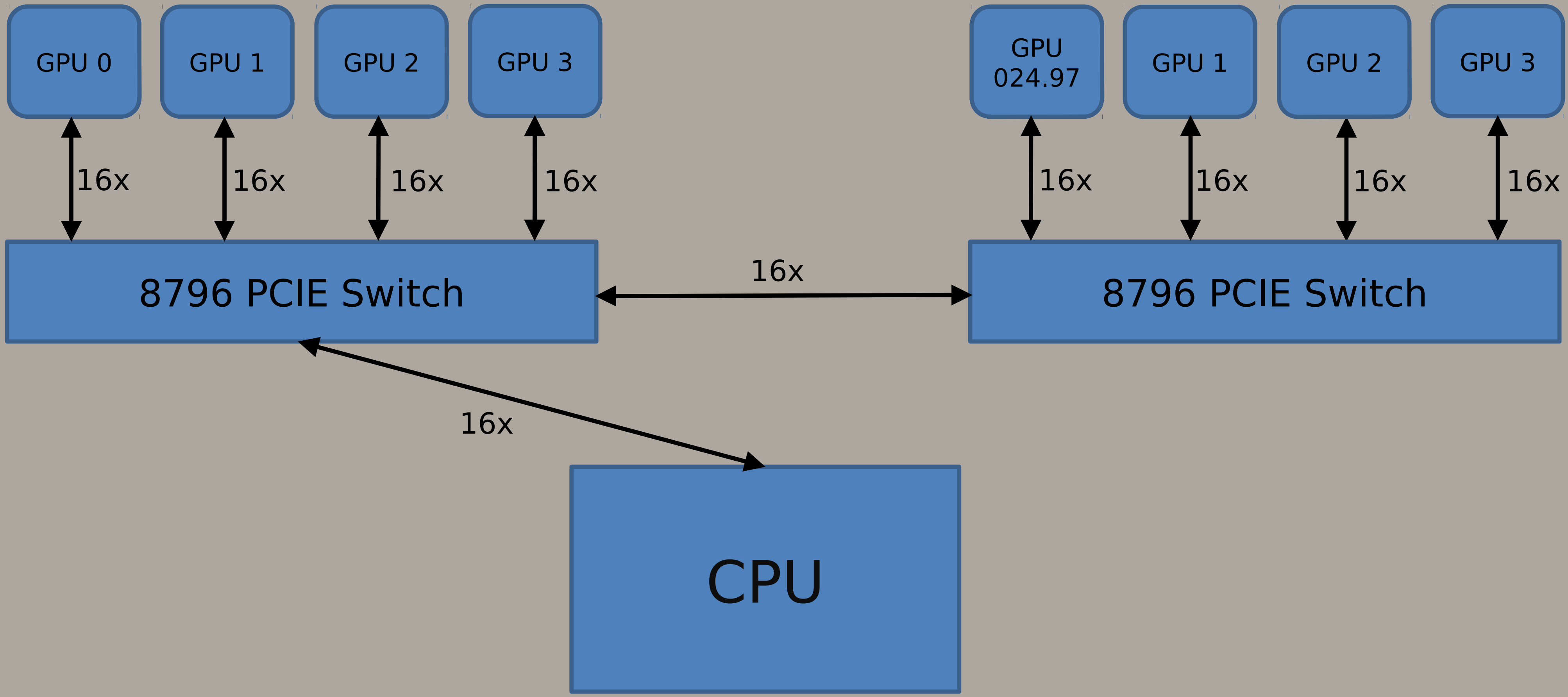
Inefficient (2016)



Intel hates P2P Bandwidth

	0	1	2	3	4	5	6	7
0	NA	25.03	25.02	25.01	15.97	15.97	14.73	15.97
1	25.03	NA	25.04	25.02	15.96	15.97	14.73	15.97
2	25.02	25.04	NA	25.02	15.97	15.96	14.73	15.96
3	25.02	25.03	25.02	NA	14.69	14.69	14.7	14.69
4	15.98	15.98	15.99	14.73	NA	25.02	25.04	25.03
5	15.98	15.98	15.98	14.73	25.03	NA	25.02	25.03
6	14.69	14.7	14.69	14.7	25.03	25.02	NA	25.03
7	15.98	15.97	15.98	14.73	25.04	25.04	25.03	NA

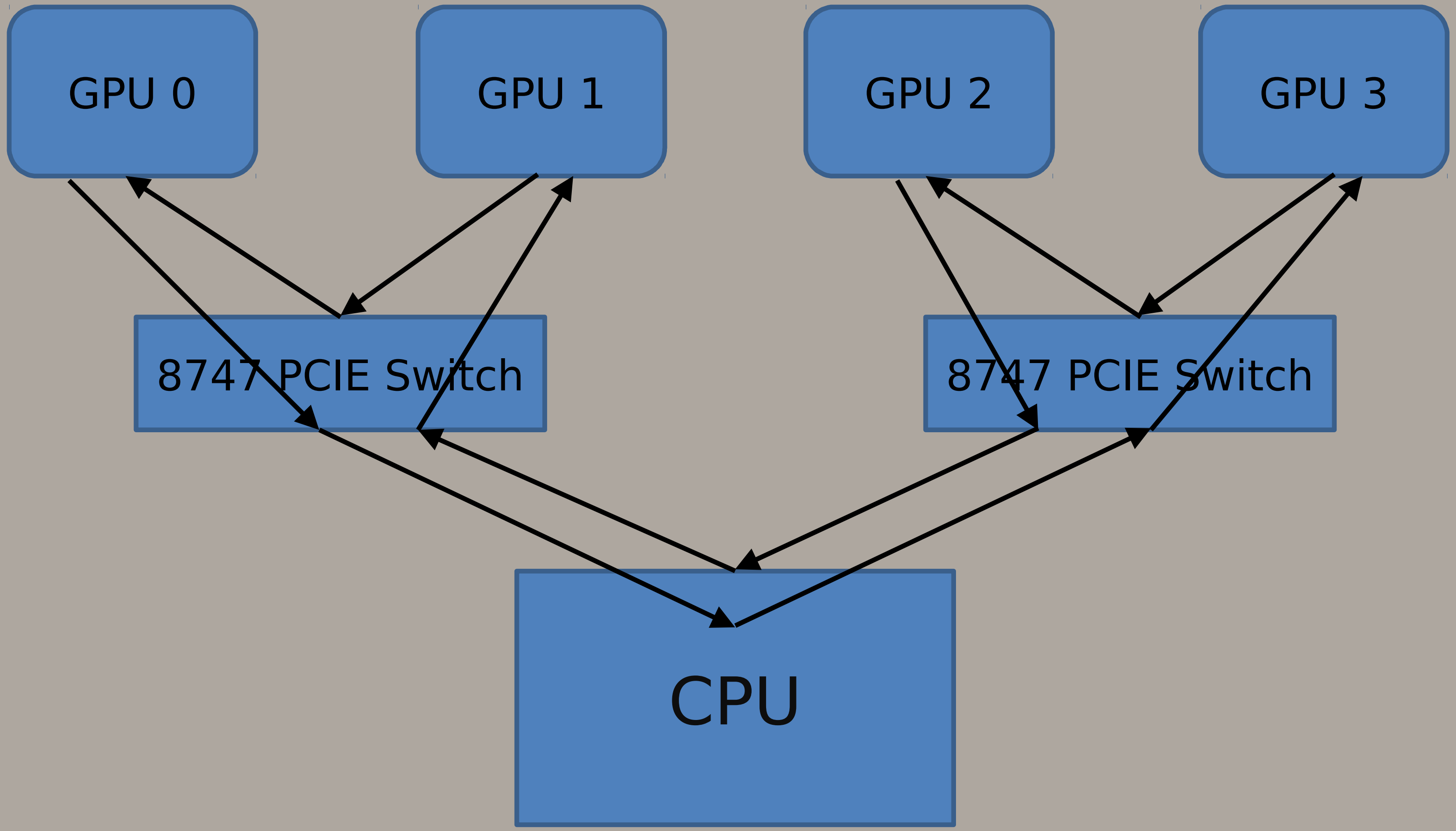
Big Sur (Efficient, 2016)



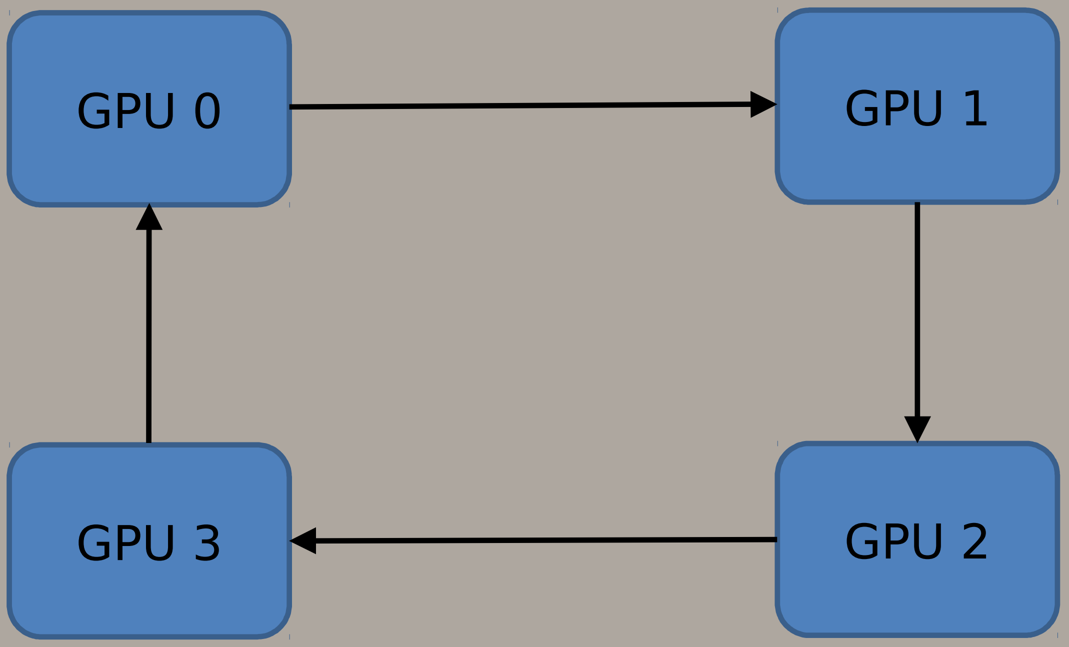
PLX **loves** P2P Bandwidth

	0	1	2	3	4	5	6	7
0	NA	24.97	24.96	24.95	24.95	24.95	24.96	24.95
1	24.97	NA	24.97	24.96	24.96	24.95	24.95	24.96
2	24.97	24.95	NA	24.95	24.96	24.96	24.95	24.95
3	24.95	24.95	24.95	NA	24.94	24.96	24.96	24.96
4	24.95	24.95	24.95	24.95	NA	24.94	24.95	24.94
5	24.95	24.95	24.94	24.94	24.95	NA	24.94	24.95
6	24.95	24.95	24.95	24.94	24.94	24.94	NA	24.95
7	24.94	24.94	24.95	24.94	24.95	24.95	24.96	NA

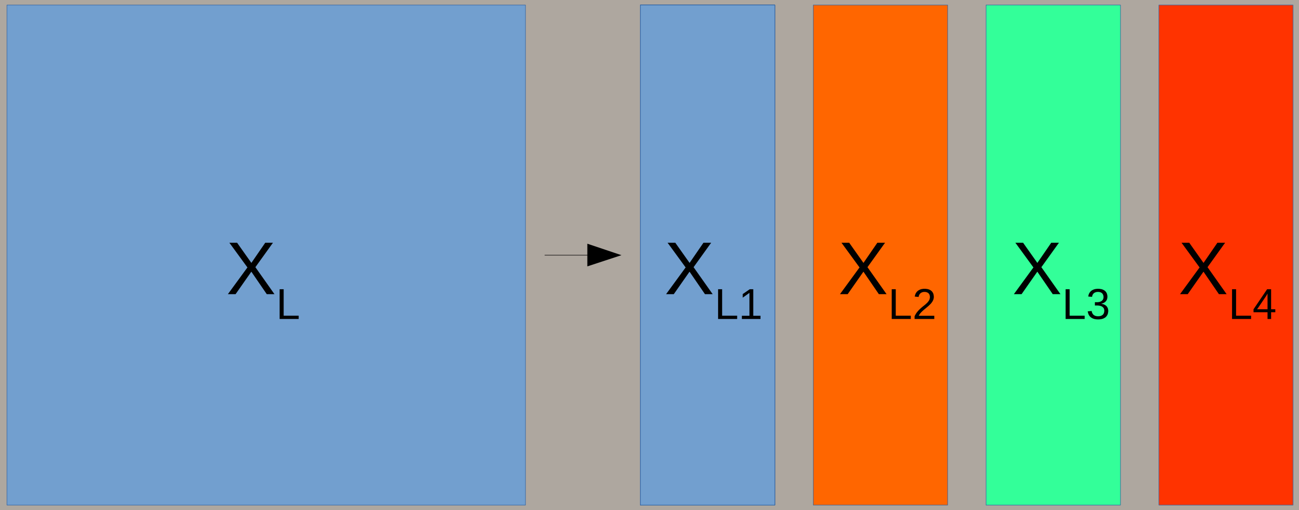
P2P Ring Implementation



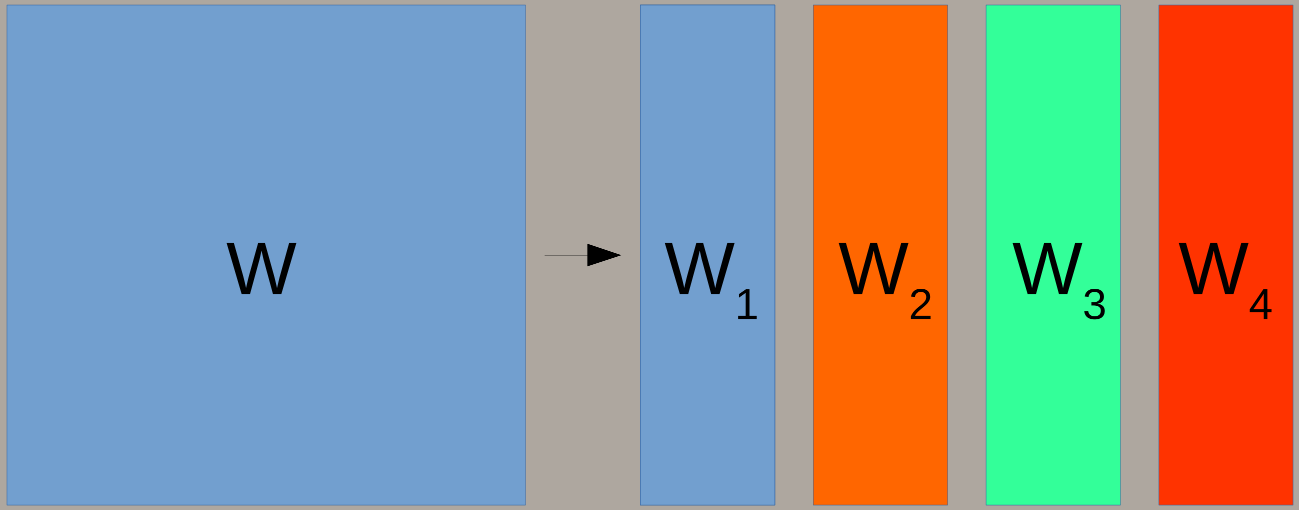
P2P Ring Simplified



Model Parallel Data

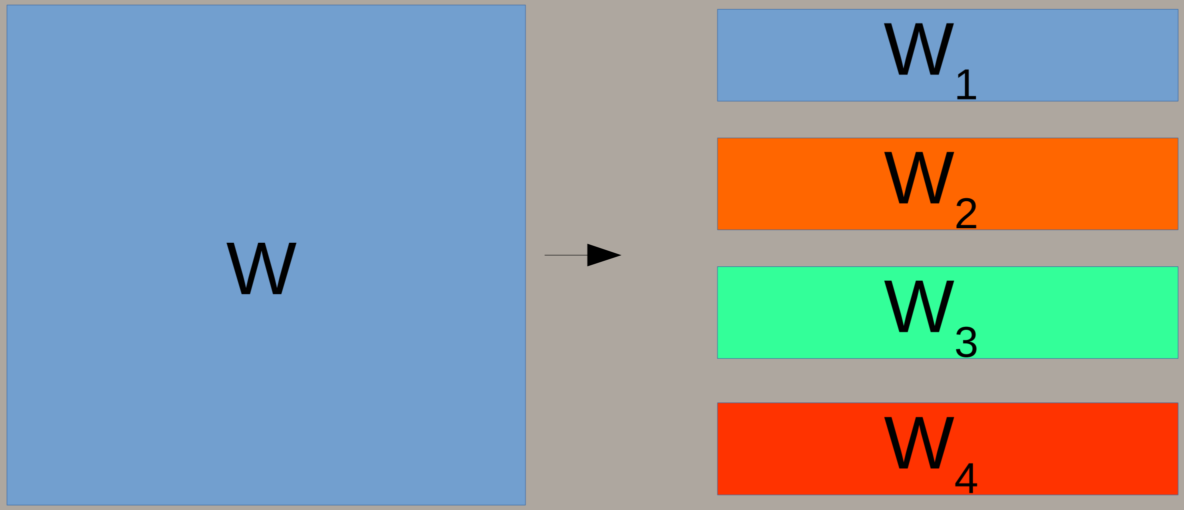


Model Parallel Weights



*Have you spotted the kryptonite yet?

Other Model Parallel Weights*

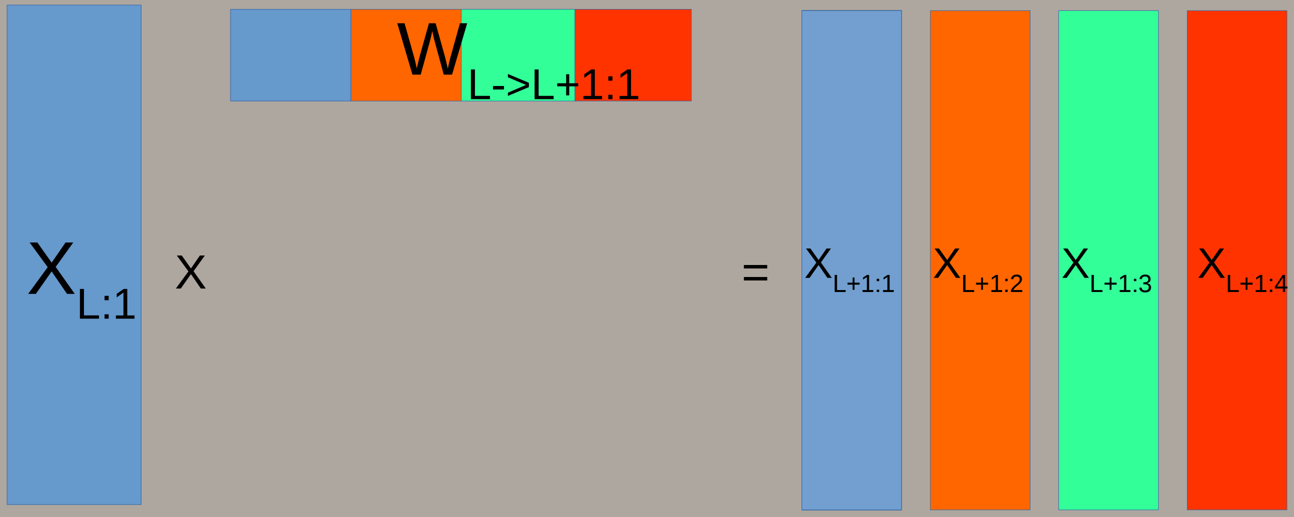


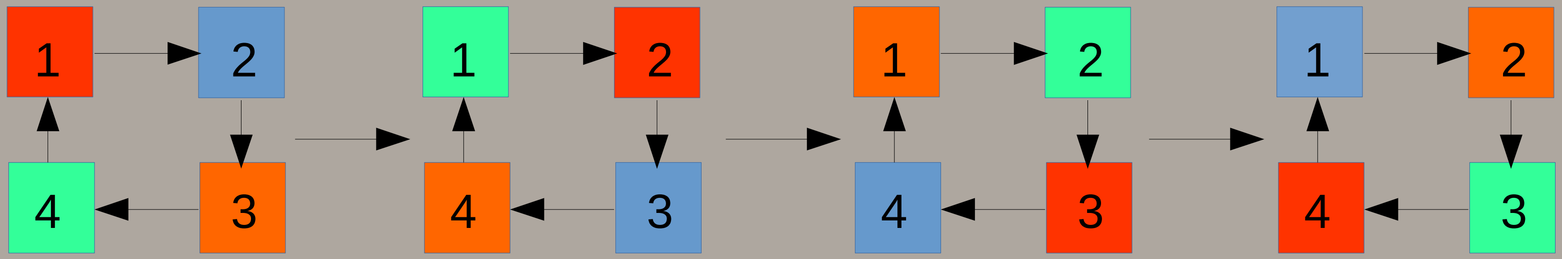
*Weight Subdivision Style Matters!

Other Other Model Parallel

- Layer by layer subdivision of network
- Sir not appearing in this talk
- Supported by Tensorflow

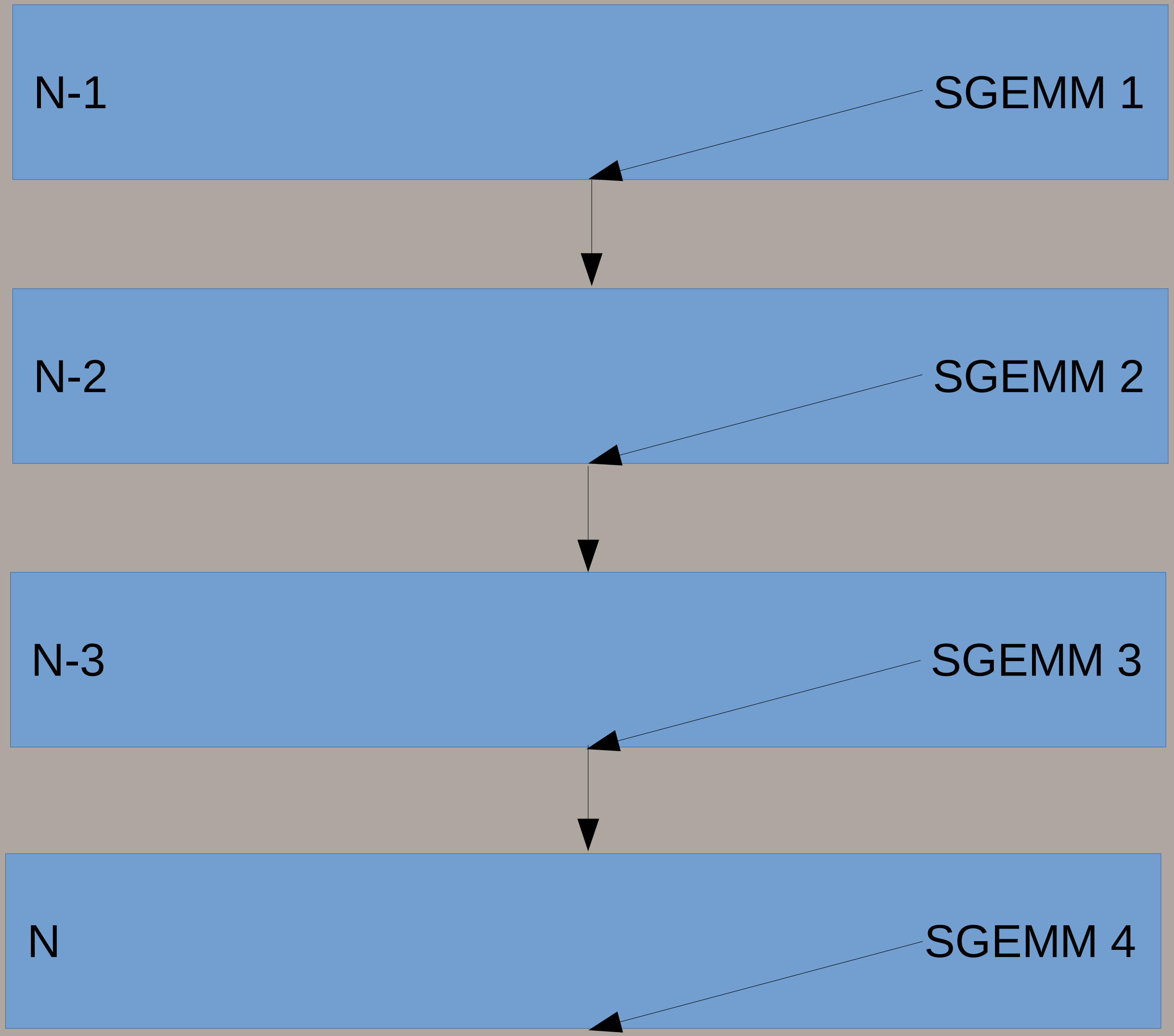
One Weird(er) Trick*



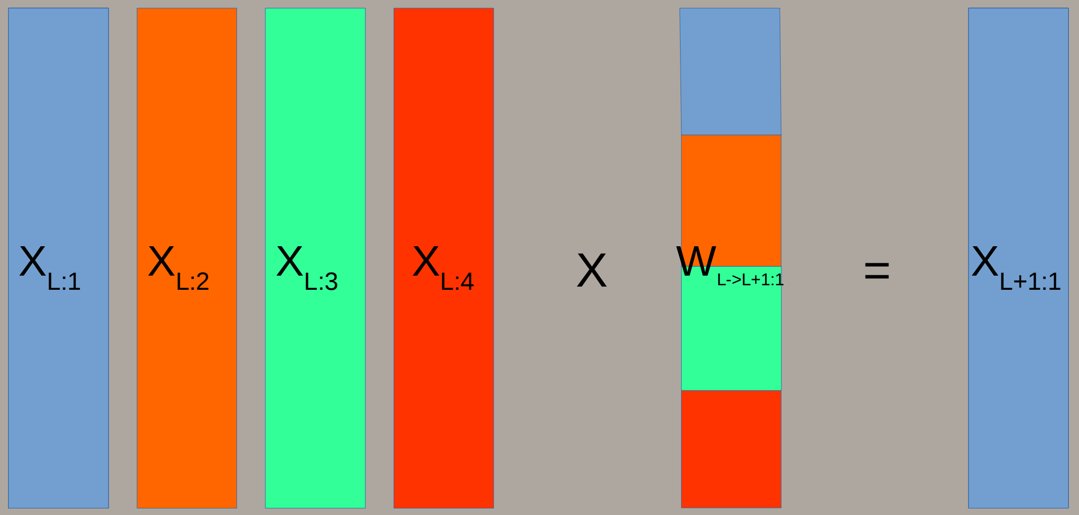


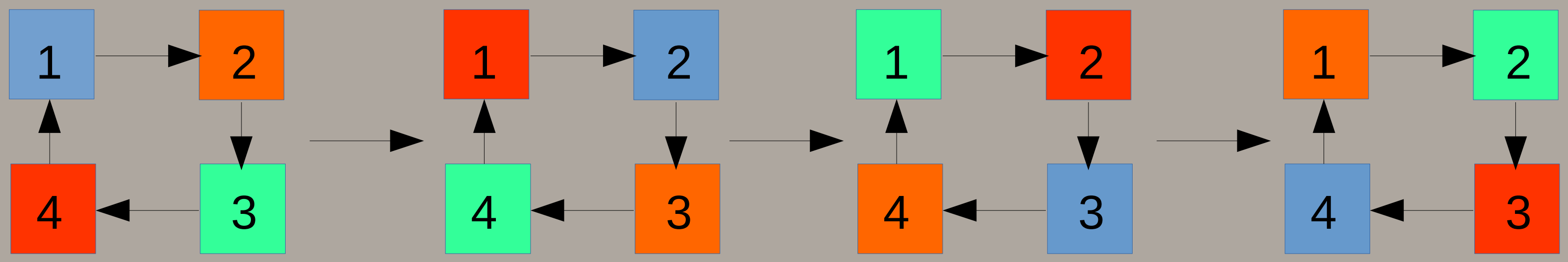
* Perform N SGEMM operations and reduce the outputs over N-1 communication steps if the model outputs are smaller than the model inputs

Reduction Flowchart



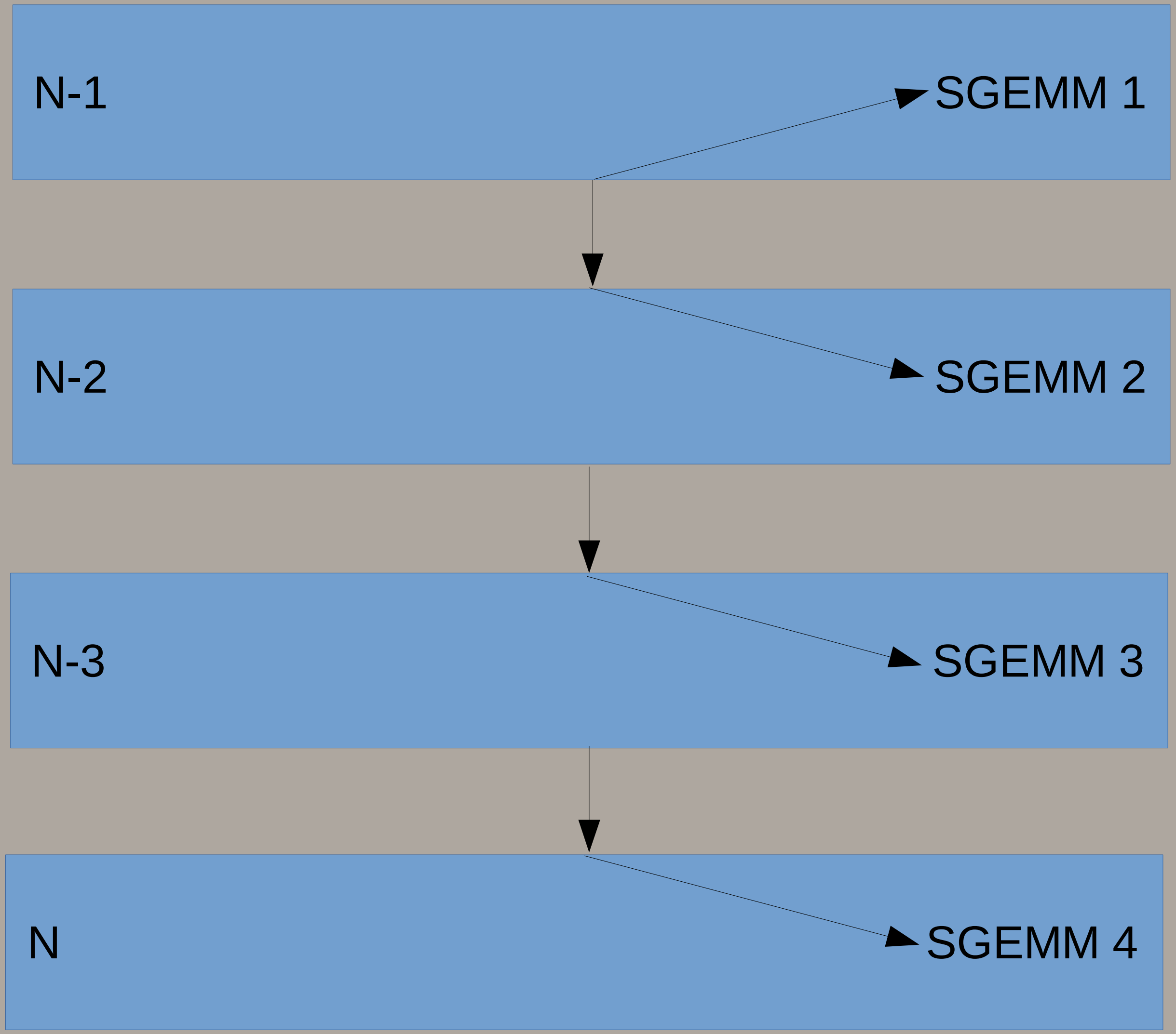
And the other Weirder Trick*





*Scatter the inputs over N-1 communication steps and SGEMMs if the model inputs are smaller than the model outputs

Gather Flowchart



Overlappping Computation/Communication

- TitanX is ~6.6 TFLOPS
- PCIE BW is ~12.5 GB/s if you don't buy crap HW
- 6.6 TFLOPS / 3.125 Gfloats ~= 2000 FLOPS
- So if you have ~1000 FMADs per output per GPU, you can run at SOL*

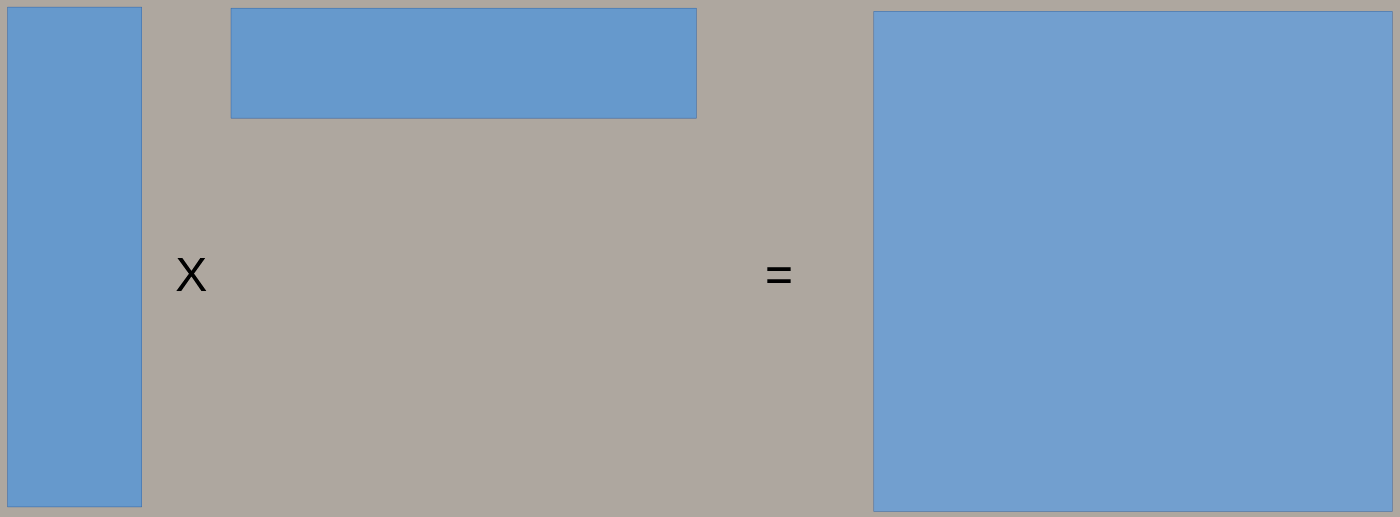
*This requires efficient SGEMM, cuBLAS is fraught...

TitanX is ~6.6 TFLOPS, cuBLAS is whatever it feels like...

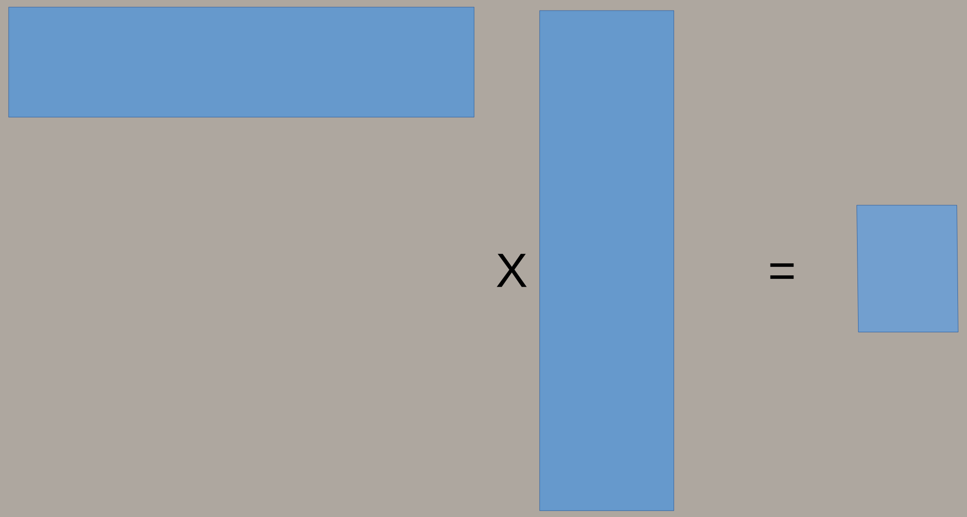
- For small batch sizes, cuBlas is anywhere from 1/10th to 1/6th of SOL
- cuBLAS **hates hates hates** narrow matrices
- cuBLAS is obsessed with multiples of 128
- Scott Gray's BLAS kernels have no such psychological issues

https://github.com/NervanaSystems/neon

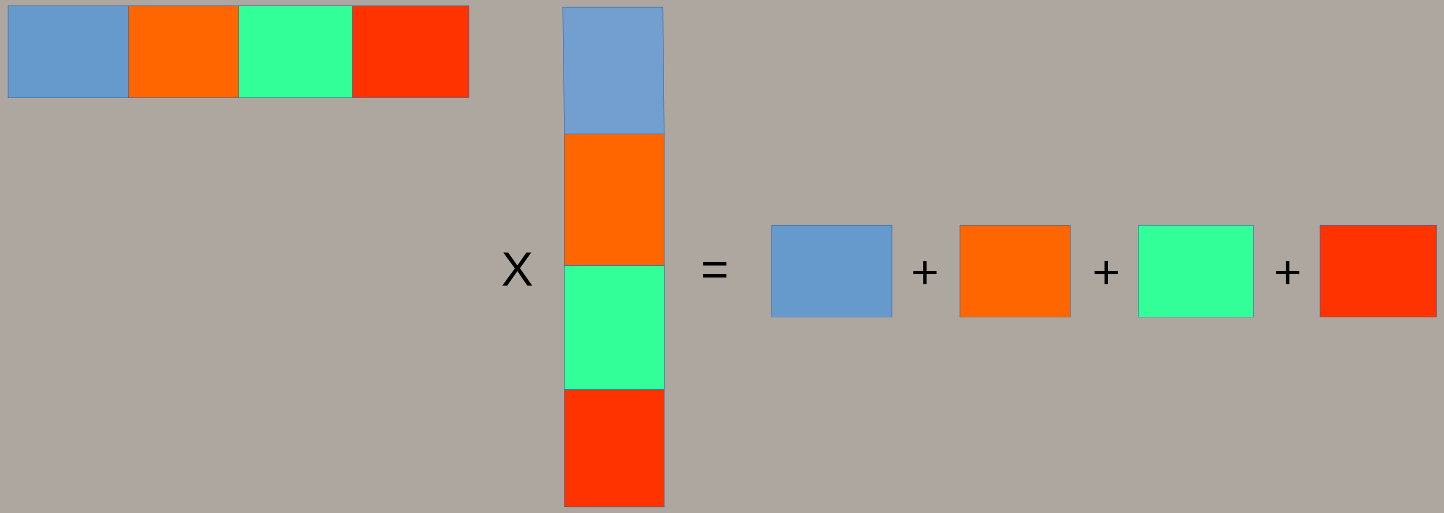
Fast Case (many outputs)



Slow Case (few outputs)



Solution: Subdivide each SGEMM within each GPU*



*Independently discovered by Scott Gray

Do you need a DGX1?

- 85 TFLOPS FP32 (10.6 TFLOPS per GPU) no FP16 for now
- 20 GB/s channels connected in a cube (N == 8)

Reduction:	2 * D / N	vs ~1.6 *D * (N -1) / N
Gather:	2 * D / N	vs ~1.6 D * (N -1) / N
AllReduce:	0.5 * D	vs ~3.2 * D * (N – 1) / N

Significant reduction in communication costs, but is AlexNet communication-limited?

Are you data-parallel?

- AlexNet has ~61M parameters
- We'll assume a batch size of 128 and Soumith Chintala's training perf numbers for TitanX scaled up by ~1.6 to arrive at 2,884 images/s FP32
- 16 images at 2,884 images/s is ~5.5 ms
- AllReducing 61M (244 MB) parameters at 20 GB/s is ~6 ms (buried 5.5 ms of backprop for overlapping copy and compute) for a final result of 0.4 ms or nearly free.
- Using P2P, this would take ~34 ms, $129K is a bargain!

Alex Krizhevsky to the Rescue! (or are you model-parallel?)

- AlexNet has ~61M parameters. ~4.3M of which are convolutional (data-parallel) and ~56.7M of which are fully-connected (model-parallel)
- Fully connected layers at a batch size of 128 is ~1.7M neurons
- P2P allReduce of 4.3M parameters takes ~2.4 ms
- P2P gather/reduction of 1.7M neurons is ~0.5 ms
- 2.9 ms is << 5.5 ms so once again it's free(tm)
- It's also faster than NVLINK data-parallel…
- NVLINK model-parallel would of course win...

TLDR: Go Model Parallel or Go Home...

Summary

- GPUs still rule HPC/Machine Learning
- Rethink algorithms into parallel-friendly implementations instead of waiting for compilers to do this for you (because they won't)
- Who needs DGX1 if we utilize model parallelism?

Acknowledgments (Amazon)

Matias Benitez

Kiuk Chung

Leo Dirac

Rejith Joseph George

Mitchell Goodman

Sebastian Gunningham

Shruti Kamath

Oleg Rybakov

Srikanth Thirumulai

Jane You

Acknowledgments (AMBER)

David Case

Romelia Salomon-Ferrer

Ben Madej

Perri Needham

Levi Pierce

Adrian Roitberg

Jason Swails

Ross Walker

Acknowledgments (NVIDIA)

Jonathan Bentz

Mark Berger

Jerry Chen

Kate Clark

Simon Layton

Duncan Poole

Sarah Tariq