Learning Systems 2018:
Lecture 7 – TensorFlow Internals and Parallelism

Crescat scientia; vita excolatur

Ian Foster and Rick Stevens
Argonne National Laboratory
The University of Chicago

A TensorFlow program

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))                    # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1)) # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")                        # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)              # Relu(Wx+b)
C = [...]                                           # Cost computed as a function
                                                    # of Relu


s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...        # Create 100-d vector for input
  result = s.run(C, feed_dict={x: input})           # Fetch cost, feeding x=input
  print step, result
```

https://arxiv.org/pdf/1603.04467.pdf

Its dataflow graph

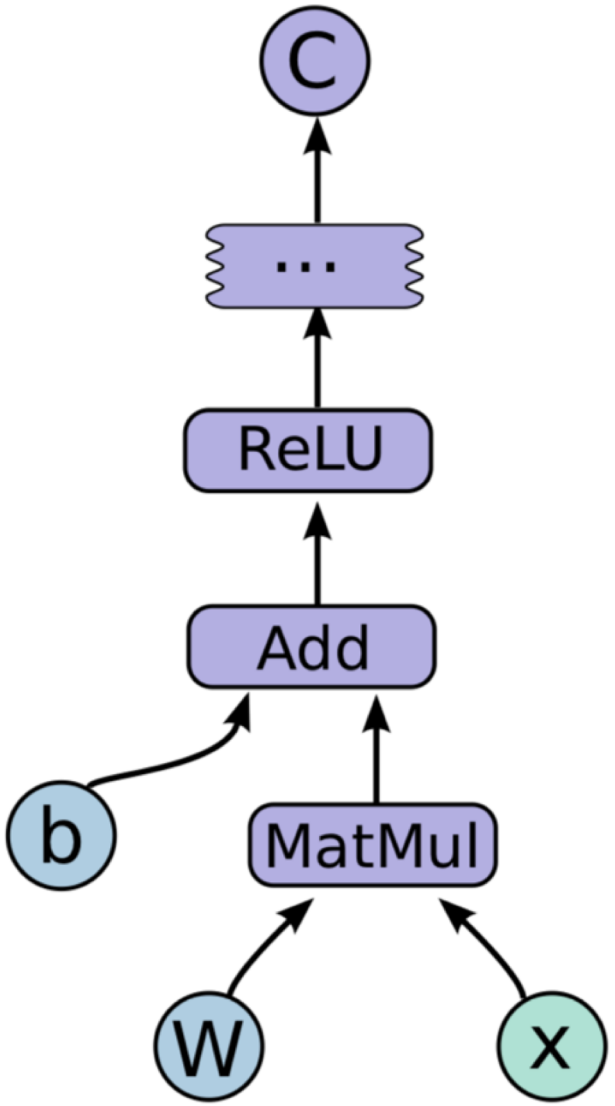
```
import tensorflow as tf
```

Define graph

```
b = tf.Variable(tf.zeros([100]))
W = tf.Variable(tf.random_uniform([784,100],-1,1))
x = tf.placeholder(name="x")
relu = tf.nn.relu(tf.matmul(W, x) + b)
C = [...]
```

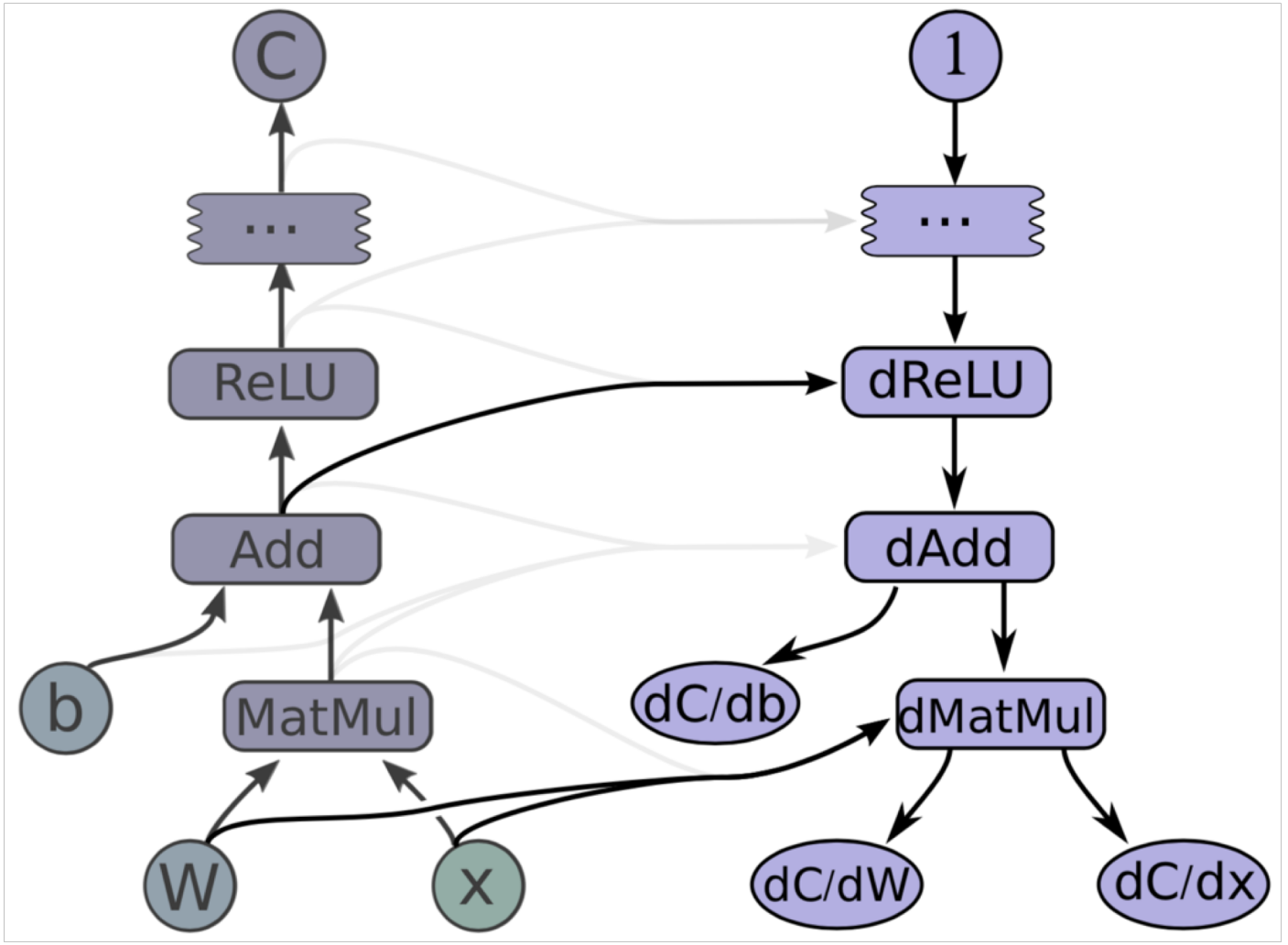
```
s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...
  result = s.run(C, feed_dict={x: input})
  print step, result
```

Deploy graph



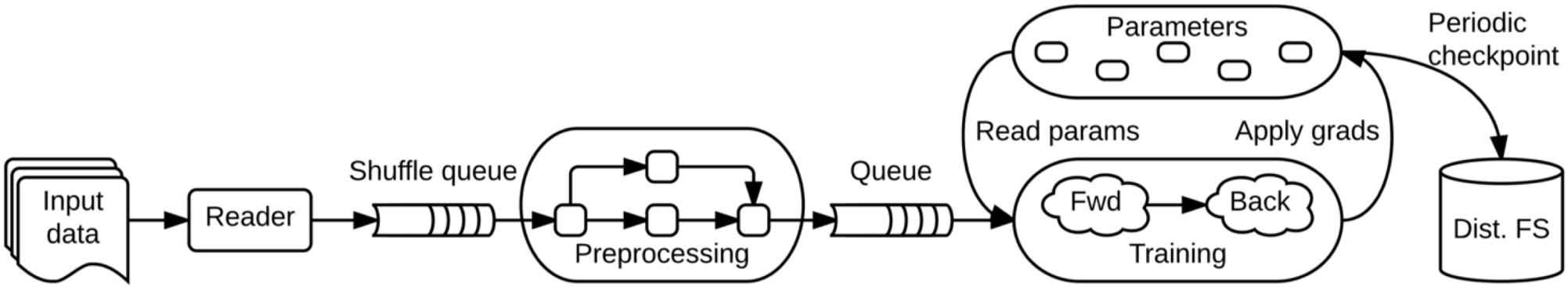
https://arxiv.org/pdf/1603.04467.pdf

And gradients



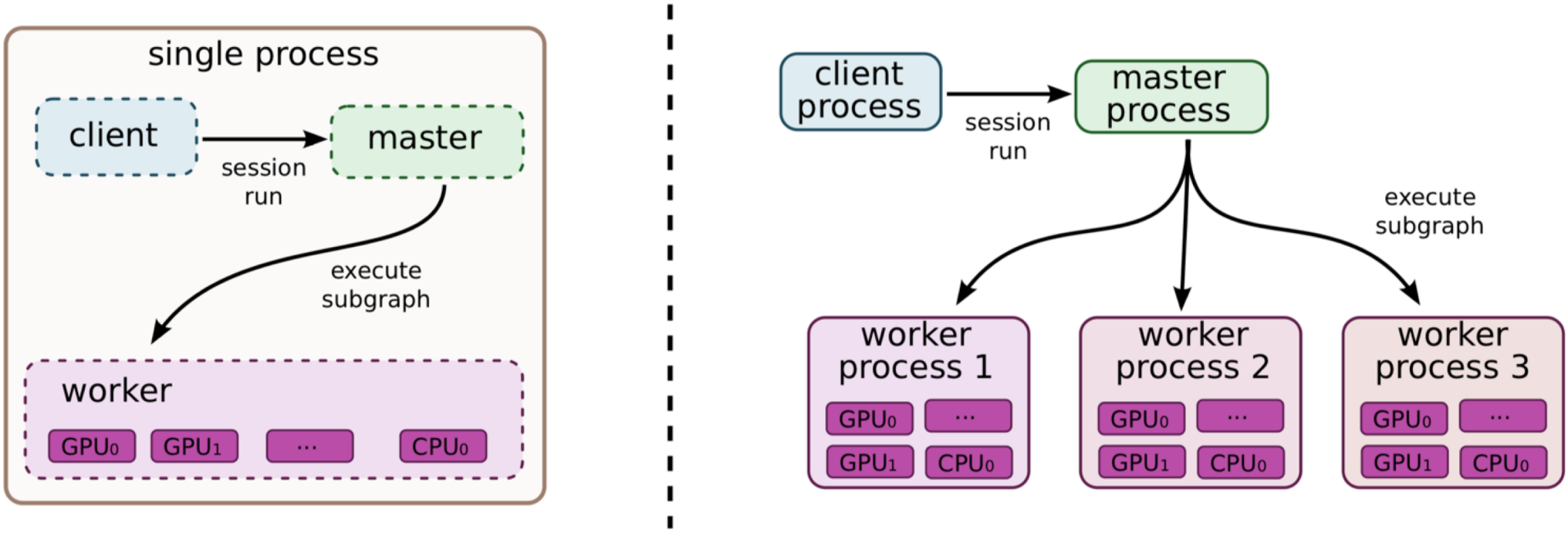
[db,dW,dx] = tf.gradients(C, [b,W,x])

https://arxiv.org/pdf/1603.04467.pdf



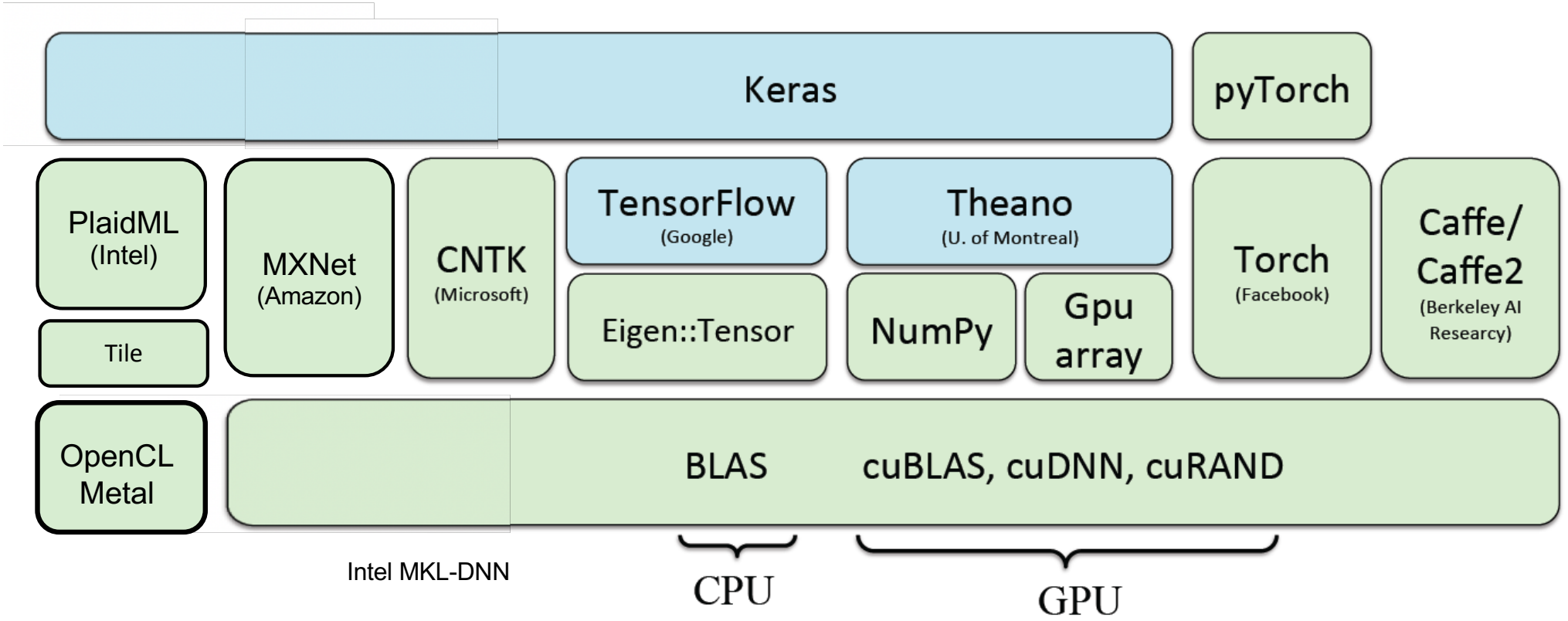
https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

Single and distributed machine structures



https://arxiv.org/pdf/1603.04467.pdf

Deep learning software stacks



BLAS: Basic Linear Algebra Subprograms

Level 1 BLAS

	dim	scalar	vector	vector	scalars		5-element array		prefixes
SUBROUTINE xROTG (					A, B, C, S)			Generate plane rotation	S, D
SUBROUTINE xROTMG(					D1, D2, A, B,		PARAM)	Generate modified plane rotation	S, D
SUBROUTINE xROT (	N,		X, INCX,	Y, INCY,		C, S)		Apply plane rotation	S, D
SUBROUTINE xROTM (	N,		X, INCX,	Y, INCY,			PARAM)	Apply modified plane rotation	S, D
SUBROUTINE xSWAP (	N,		X, INCX,	Y, INCY)				$x \leftrightarrow y$	S, D, C, Z
SUBROUTINE xSCAL (	N,	ALPHA,	X, INCX)					$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
SUBROUTINE xCOPY (	N,		X, INCX,	Y, INCY)				$y \leftarrow x$	S, D, C, Z
SUBROUTINE xAXPY (	N,	ALPHA,	X, INCX,	Y, INCY)				$y \leftarrow \alpha x + y$	S, D, C, Z
FUNCTION xDOT (	N,		X, INCX,	Y, INCY)				$dot \leftarrow x^T y$	S, D, DS
FUNCTION xDOTU (	N,		X, INCX,	Y, INCY)				$dot \leftarrow x^T y$	C, Z
FUNCTION xDOTC (	N,		X, INCX,	Y, INCY)				$dot \leftarrow x^H y$	C, Z
FUNCTION xxDOT (	N,		X, INCX,	Y, INCY)				$dot \leftarrow \alpha + x^T y$	SDS
FUNCTION xNRM2 (	N,		X, INCX)					$nrm2 \leftarrow \|x\|_2$	S, D, SC, DZ
FUNCTION xASUM (	N,		X, INCX)					$asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$	S, D, SC, DZ
FUNCTION IxAMAX(	N,		X, INCX)					$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

	options			dim	b-width	scalar	matrix	vector	scalar	vector		
xGEMV (		TRANS,		M, N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xGBMV (		TRANS,		M, N,	KL, KU,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xHEMV (	UPLO,			N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHBMV (	UPLO,			N,	K,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHPMV (	UPLO,			N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xSYMV (	UPLO,			N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSBMV (	UPLO,			N,	K,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSPMV (	UPLO,			N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xTRMV (	UPLO,	TRANS,	DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTBMV (	UPLO,	TRANS,	DIAG,	N,	K,		A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTPMV (	UPLO,	TRANS,	DIAG,	N,			AP,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTRSV (	UPLO,	TRANS,	DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTBSV (	UPLO,	TRANS,	DIAG,	N,	K,		A, LDA,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTPSV (	UPLO,	TRANS,	DIAG,	N,			AP,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z

	options		dim	scalar	vector	vector	matrix		
xGER (			M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
xGERU (			M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
xGERC (			M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
xHER (	UPLO,		N,	ALPHA,	X, INCX,		A, LDA)	$A \leftarrow \alpha xx^H + A$	C, Z
xHPR (	UPLO,		N,	ALPHA,	X, INCX,		AP)	$A \leftarrow \alpha xx^H + A$	C, Z
xHER2 (	UPLO,		N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xHPR2 (	UPLO,		N,	ALPHA,	X, INCX,	Y, INCY,	AP)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xSYR (	UPLO,		N,	ALPHA,	X, INCX,		A, LDA)	$A \leftarrow \alpha xx^T + A$	S, D
xSPR (	UPLO,		N,	ALPHA,	X, INCX,		AP)	$A \leftarrow \alpha xx^T + A$	S, D
xSYR2 (	UPLO,		N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D
xSPR2 (	UPLO,		N,	ALPHA,	X, INCX,	Y, INCY,	AP)	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

Level 3 BLAS

	options			dim	scalar	matrix	matrix	scalar	matrix		
xGEMM (			TRANSA, TRANSB,	M, N, K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha op(A)op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
xSYMM (	SIDE,	UPLO,		M, N,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$	S, D, C, Z
xHEMM (	SIDE,	UPLO,		M, N,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C, Z
xSYRK (		UPLO,	TRANS,	N, K,	ALPHA,	A, LDA,		BETA,	C, LDC)	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S, D, C, Z
xHERK (		UPLO,	TRANS,	N, K,	ALPHA,	A, LDA,		BETA,	C, LDC)	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C, Z
xSYR2K(		UPLO,	TRANS,	N, K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB^T + \bar{\alpha} BA^T + \beta C, C \leftarrow \alpha A^T B + \bar{\alpha} B^T A + \beta C, C - n \times n$	S, D, C, Z
xHER2K(		UPLO,	TRANS,	N, K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$	C, Z
xTRMM (	SIDE,	UPLO,	TRANSA, DIAG,	M, N,	ALPHA,	A, LDA,	B, LDB)			$B \leftarrow \alpha op(A)B, B \leftarrow \alpha Bop(A), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
xTRSM (	SIDE,	UPLO,	TRANSA, DIAG,	M, N,	ALPHA,	A, LDA,	B, LDB)			$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha Bop(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

http://www.netlib.org/blas/

Level 1 BLAS

	dim	scalar	vector	vector	scalars				5-element array	
SUBROUTINE xROTG (						A,	B,	C, S)		Generate plane rotation
SUBROUTINE xROTMG(					D1, D2,	A,	B,		PARAM)	Generate modified plane rotation
SUBROUTINE xROT (	N,		X,	INCX, Y, INCY,				C, S)		Apply plane rotation
SUBROUTINE xROTM (	N,		X,	INCX, Y, INCY,					PARAM)	Apply modified plane rotation
SUBROUTINE xSWAP (	N,		X,	INCX, Y, INCY)						$x \leftrightarrow y$
SUBROUTINE xSCAL (	N,	ALPHA,	X,	INCX)						$x \leftarrow \alpha x$
SUBROUTINE xCOPY (	N,		X,	INCX, Y, INCY)						$y \leftarrow x$
SUBROUTINE xAXPY (	N,	ALPHA,	X,	INCX, Y, INCY)						$y \leftarrow \alpha x + y$
FUNCTION xDOT (	N,		X,	INCX, Y, INCY)						$dot \leftarrow x^T y$
FUNCTION xDOTU (	N,		X,	INCX, Y, INCY)						$dot \leftarrow x^T y$
FUNCTION xDOTC (	N,		X,	INCX, Y, INCY)						$dot \leftarrow x^H y$
FUNCTION xxDOT (	N,		X,	INCX, Y, INCY)						$dot \leftarrow \alpha + x^T y$
FUNCTION xNRM2 (	N,		X,	INCX)						$nrm2 \leftarrow \|x\|_2$
FUNCTION xASUM (	N,		X,	INCX)						$asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$
FUNCTION IxAMAX(	N,		X,	INCX)						$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$

Level 2 BLAS

	options			dim	b-width	scalar	matrix	vector		scalar	vector	
xGEMV (		TRANS,		M, N,		ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$
xGBMV (		TRANS,		M, N,	KL, KU,	ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$
xHEMV (	UPLO,			N,		ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$
xHBMV (	UPLO,			N,	K,	ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$
xHPMV (	UPLO,			N,		ALPHA,	AP,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$
xSYMV (	UPLO,			N,		ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$
xSBMV (	UPLO,			N,	K,	ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$
xSPMV (	UPLO,			N,		ALPHA,	AP,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$
xTRMV (	UPLO,	TRANS,	DIAG,	N,			A, LDA,	X, INCX)				$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$
xTBMV (	UPLO,	TRANS,	DIAG,	N,	K,		A, LDA,	X, INCX)				$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$
xTPMV (	UPLO,	TRANS,	DIAG,	N,			AP,	X, INCX)				$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$
xTRSV (	UPLO,	TRANS,	DIAG,	N,			A, LDA,	X, INCX)				$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$
xTBSV (	UPLO,	TRANS,	DIAG,	N,	K,		A, LDA,	X, INCX)				$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$
xTPSV (	UPLO,	TRANS,	DIAG,	N,			AP,	X, INCX)				$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$

	options			dim	scalar	vector		vector	matrix	
xGER (				M, N,	ALPHA,	X, INCX,		Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$

http://www.netlib.org/blas/

xGEMM:

C := α * A * B + β * C

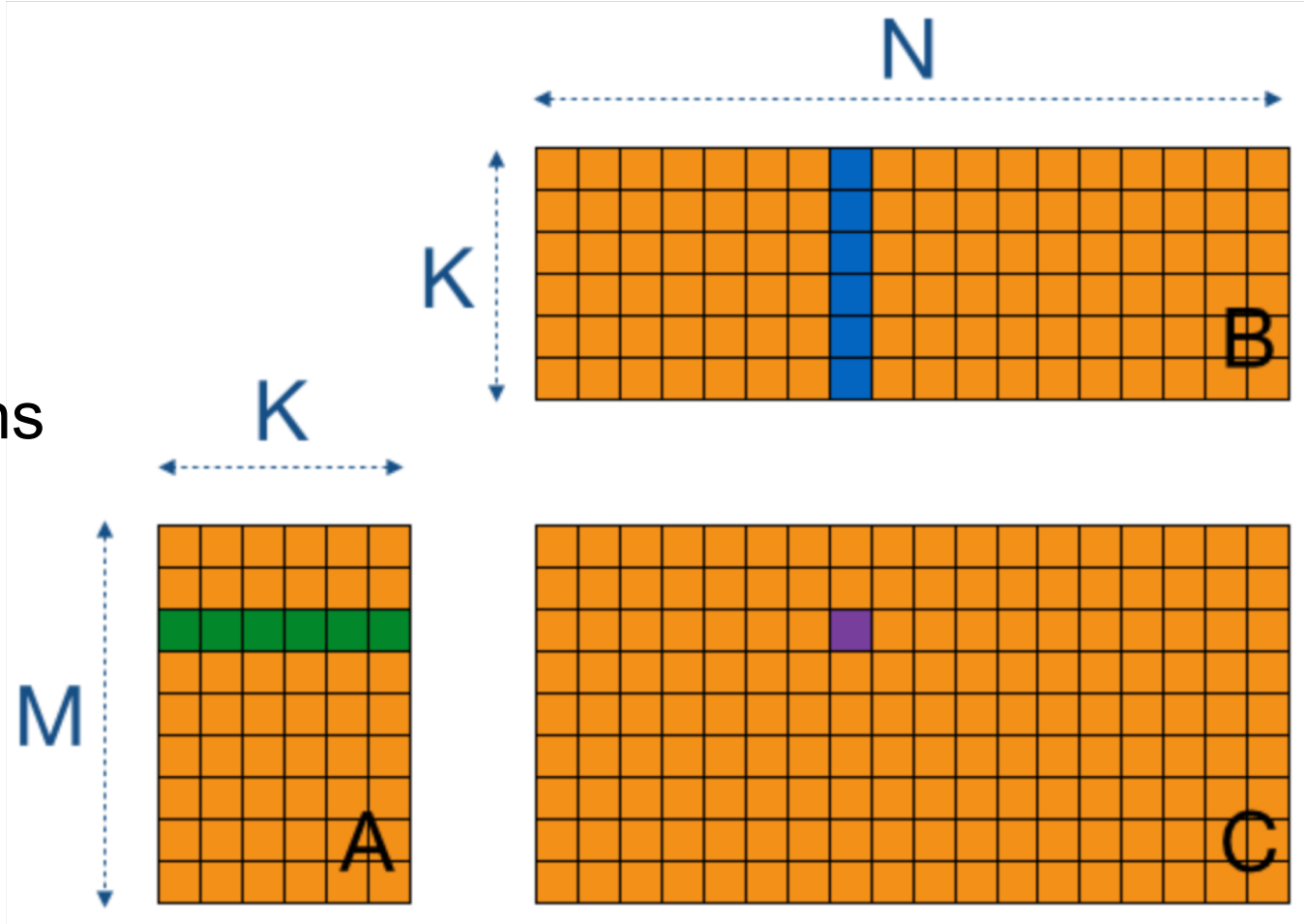
If α = 1 and β = 0:

C := A * B

In total:

M*N*K multiplications

M*N*K additions



Matrix multiplication: Simple

```
1. for (int m=0; m<M; m++) {
2.   for (int n=0; n<N; n++) {
3.     float acc = 0.0f;
4.     for (int k=0; k<K; k++) {
5.       acc += A[k*M + m] * B[n*K + k];
6.     }
7.     C[n*M + m] = acc;
8.   }
9. }
```

Out of curiosity I decided to benchmark my own matrix multiplication function versus the BLAS implementation... I was to say the least surprised at the result:

> Custom Implementation, 10 trials of 1000x1000 matrix multiplication:
>
> ```
> Took: 15.76542 seconds.
> ```
>
> BLAS Implementation, 10 trials of 1000x1000 matrix multiplication:
>
> ```
> Took: 1.32432 seconds.
> ```

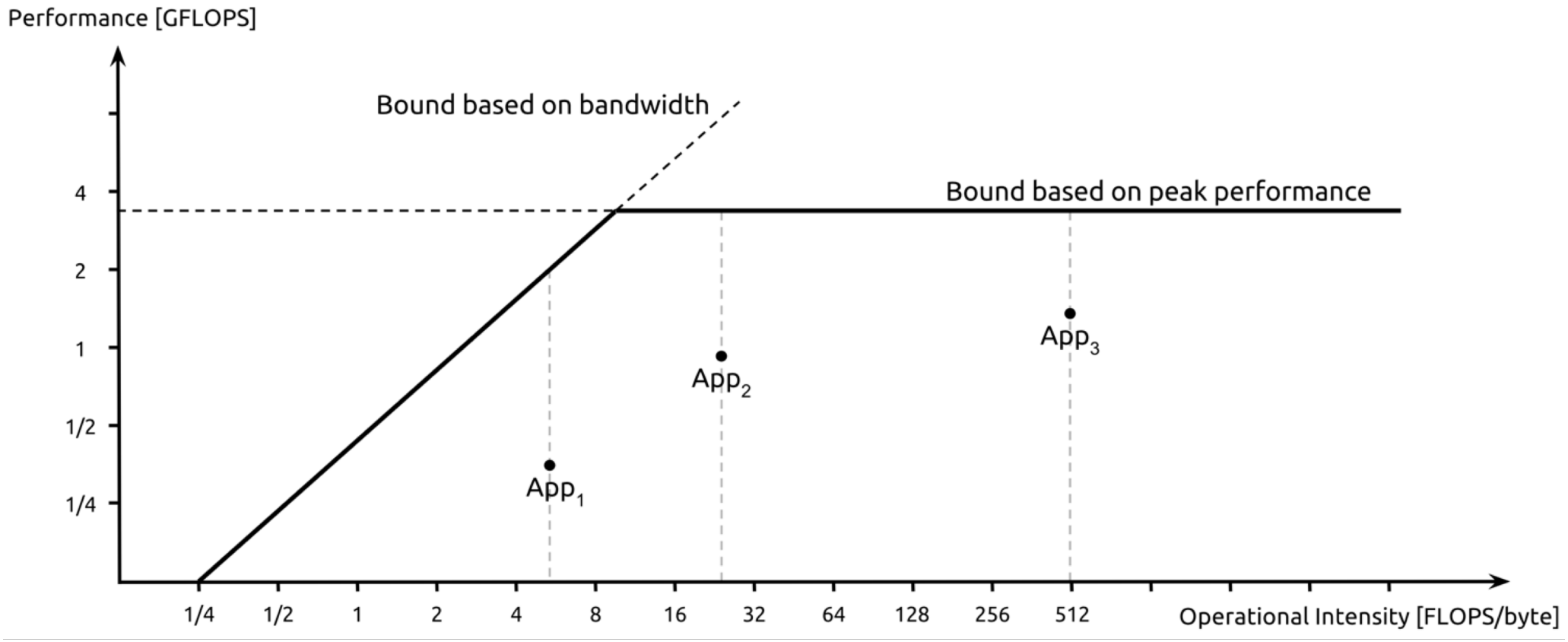
This is using single precision floating point numbers.

My Implementation:

```
template<class ValT>
void mmult(const ValT* A, int ADim1, int ADim2, const ValT* B, int BDim1, int BDim2,
{
    if ( ADim2!=BDim1 )
        throw std::runtime_error("Error sizes off");

    memset((void*)C,0,sizeof(ValT)*ADim1*BDim2);
    int cc2,cc1,cr1;
    for ( cc2=0 ; cc2<BDim2 ; ++cc2 )
        for ( cc1=0 ; cc1<ADim2 ; ++cc1 )
            for ( cr1=0 ; cr1<ADim1 ; ++cr1 )
                C[cc2*ADim2+cr1] += A[cc1*ADim1+cr1]*B[cc2*BDim1+cc1];
}
```

Roofline performance model



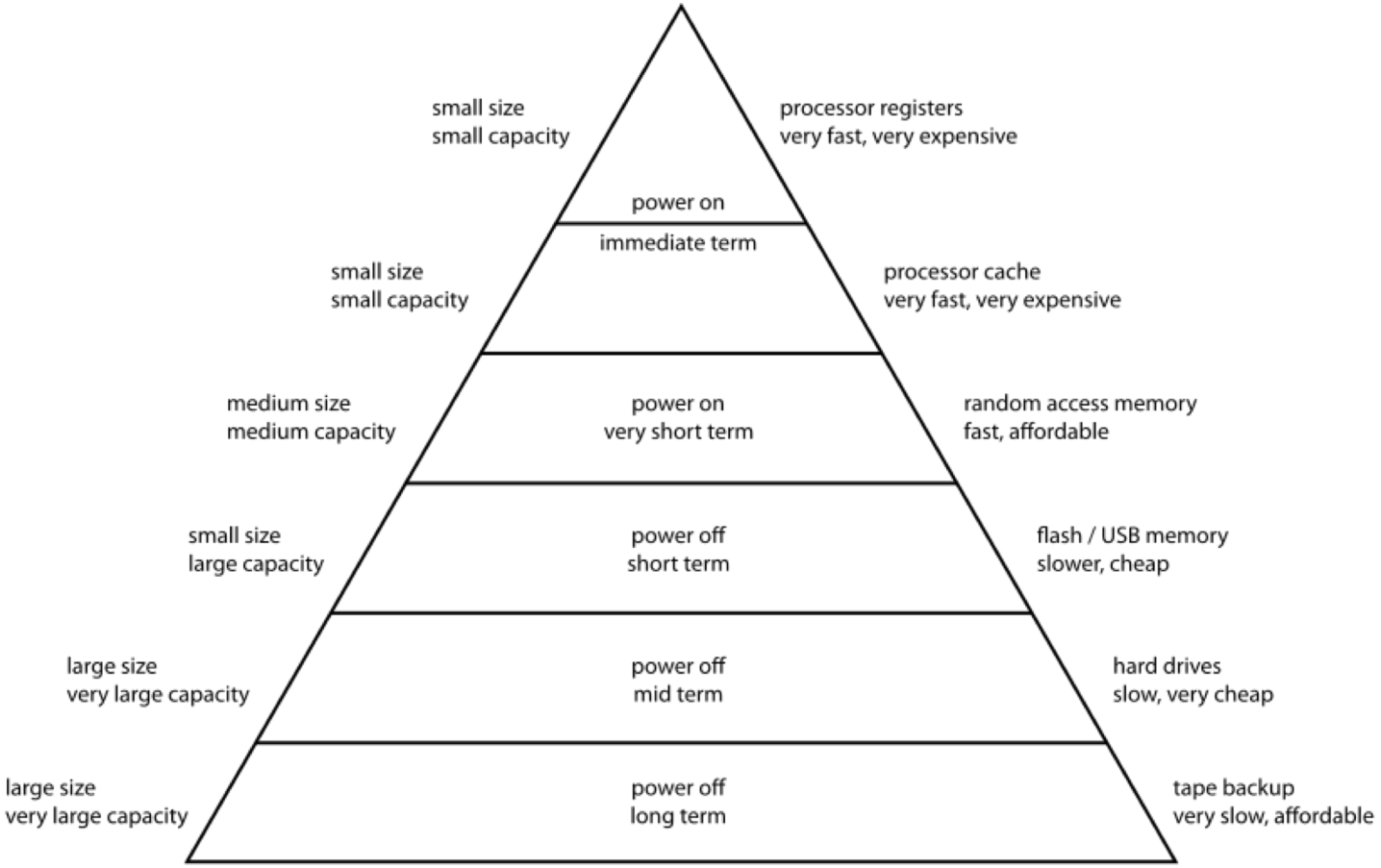
https://en.wikipedia.org/wiki/Roofline_model

Matrix multiplication: Simple and slow

```
1. for (int m=0; m<M; m++) {
2.   for (int n=0; n<N; n++) {
3.     float acc = 0.0f;
4.     for (int k=0; k<K; k++) {
5.       acc += A[k*M + m] * B[n*K + k];
6.     }
7.     C[n*M + m] = acc;
8.   }
9. }
```

M*N*K*2 loads and M*N stores
to perform M*N*K multiplications and M*N*K additions
=> 0.5 ops per load, if multiplication & addition are merged

Computer Memory Hierarchy

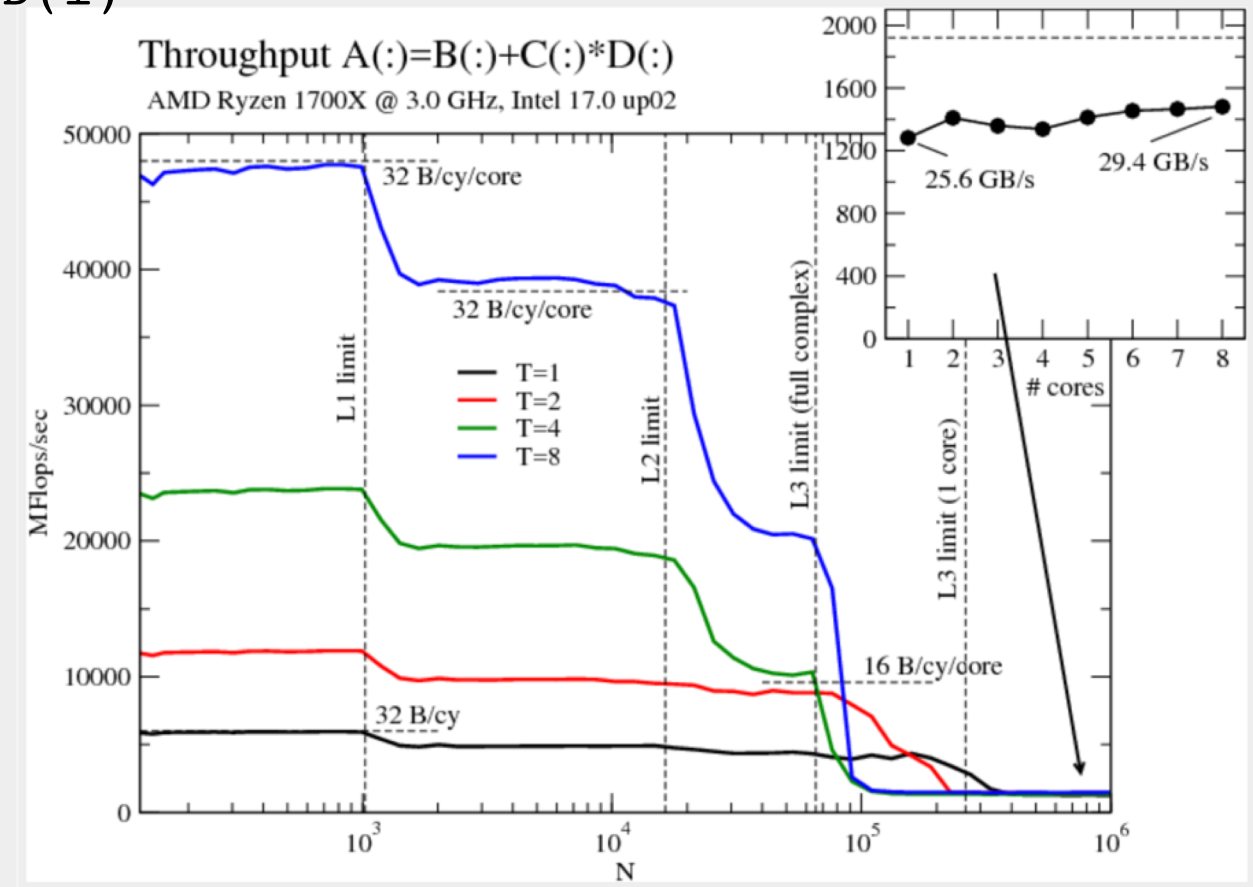


Wikipedia

```
do r=1,NITER
  do i=1,N
    A(i) = B(i) + C(i) * D(i)
  enddo
enddo
```

Cache Organization

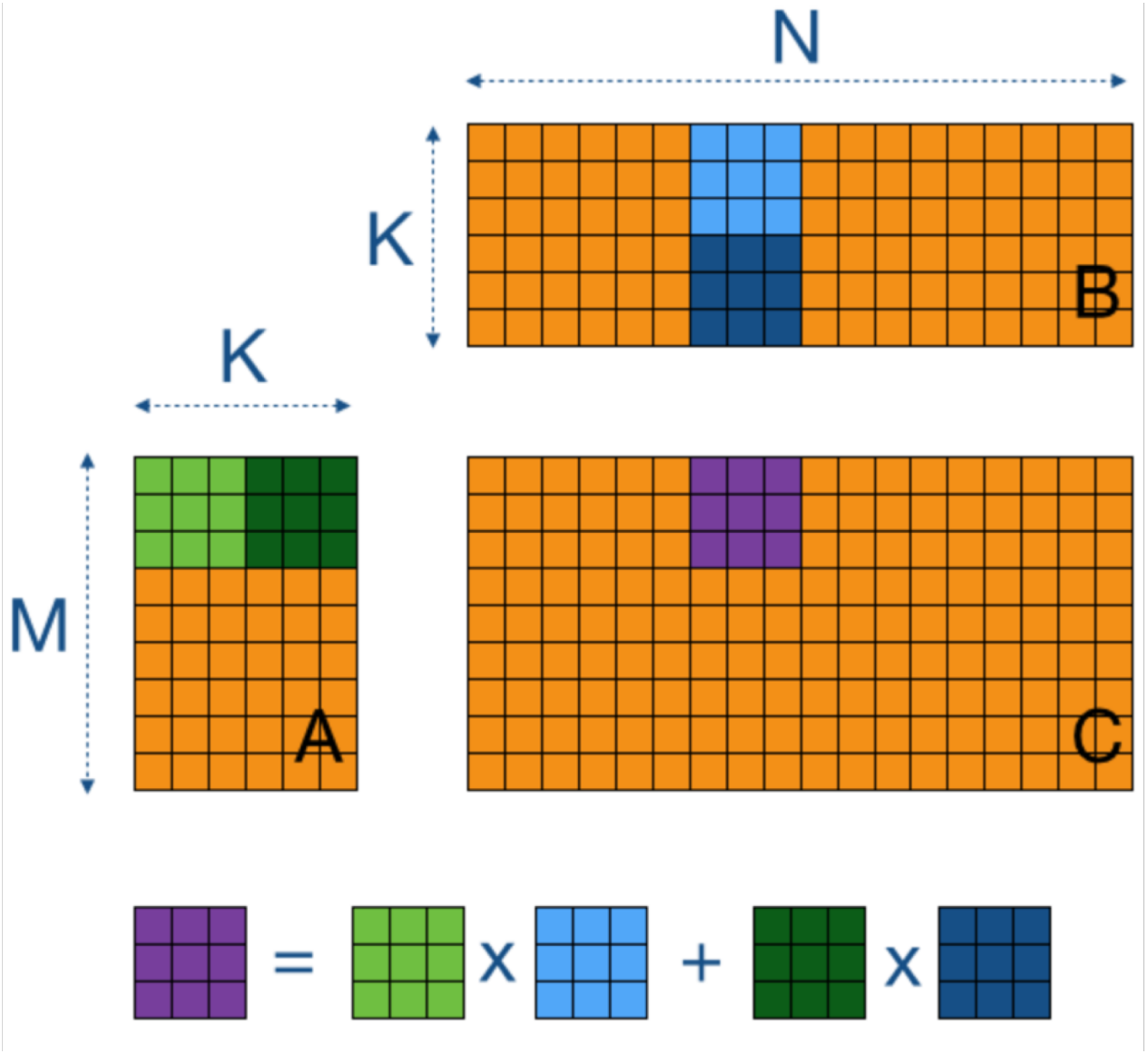
L1$	768 KiB	**L1I$**	512 KiB 8x64 KiB
		L1D$	256 KiB 8x32 KiB
L2$	4 MiB		8x512 KiB
L3$	16 MiB		2x8 MiB



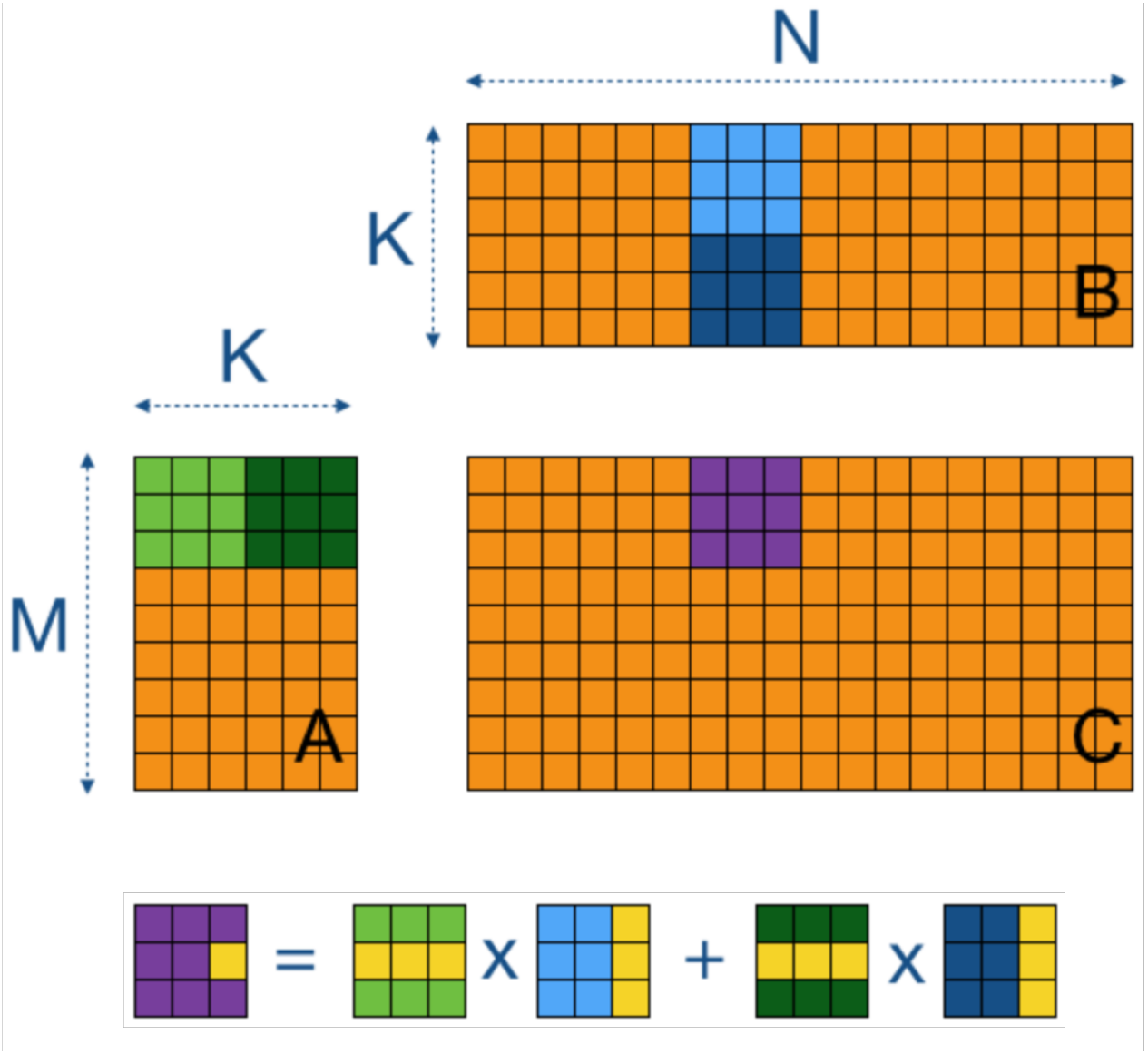
Throughput-mode vector triad on the Ryzen chip vs. array length with 1, 2, 4, and 8 cores. Inset: Performance vs. number of cores for an in-memory dataset. The bandwidth numbers assume a code balance of 20 B/flop in all hierarchy levels except L1.

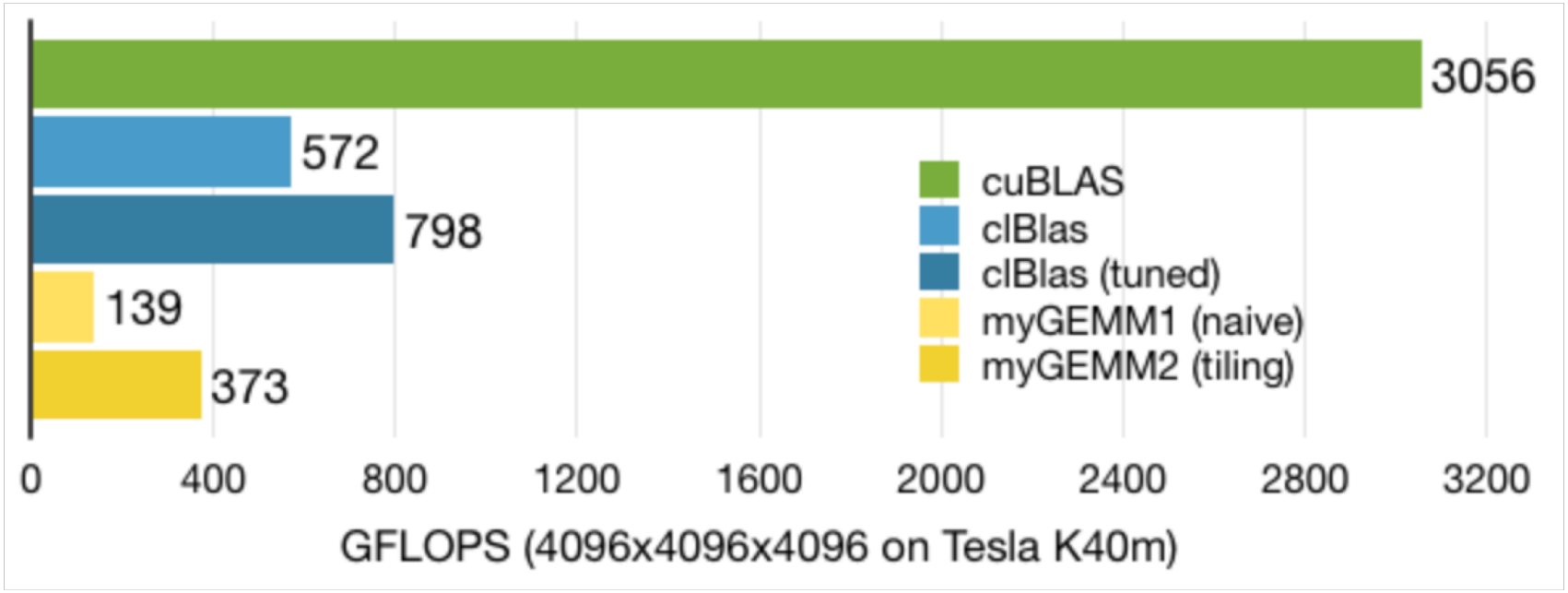
https://blogs.fau.de/hager/archives/7810

Tiling



Tiling




3056
572
798
139
373
cuBLAS
clBlas
clBlas (tuned)
myGEMM1 (naive)
myGEMM2 (tiling)
0
400
800
1200
1600
2000
2400
2800
3200
GFLOPS (4096x4096x4096 on Tesla K40m)

Writing the Fastest Code, by Hand, for Fun: A Human Computer Keeps Speeding Up Chips

By **JOHN MARKOFF** NOV. 28, 2005

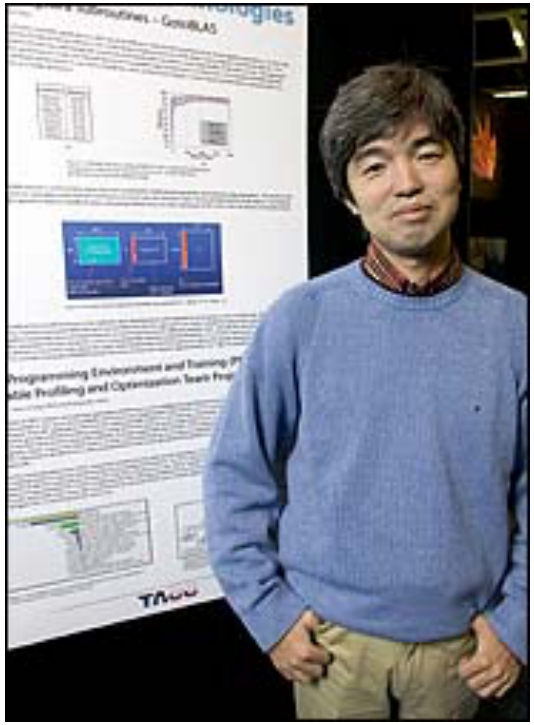
SEATTLE - There was a time long ago when the word "computer" was a job description referring to the humans who performed the tedious mathematical calculations for huge military and engineering projects.

It is in the same sense that Kazushige Goto's business card says simply "high performance computing."

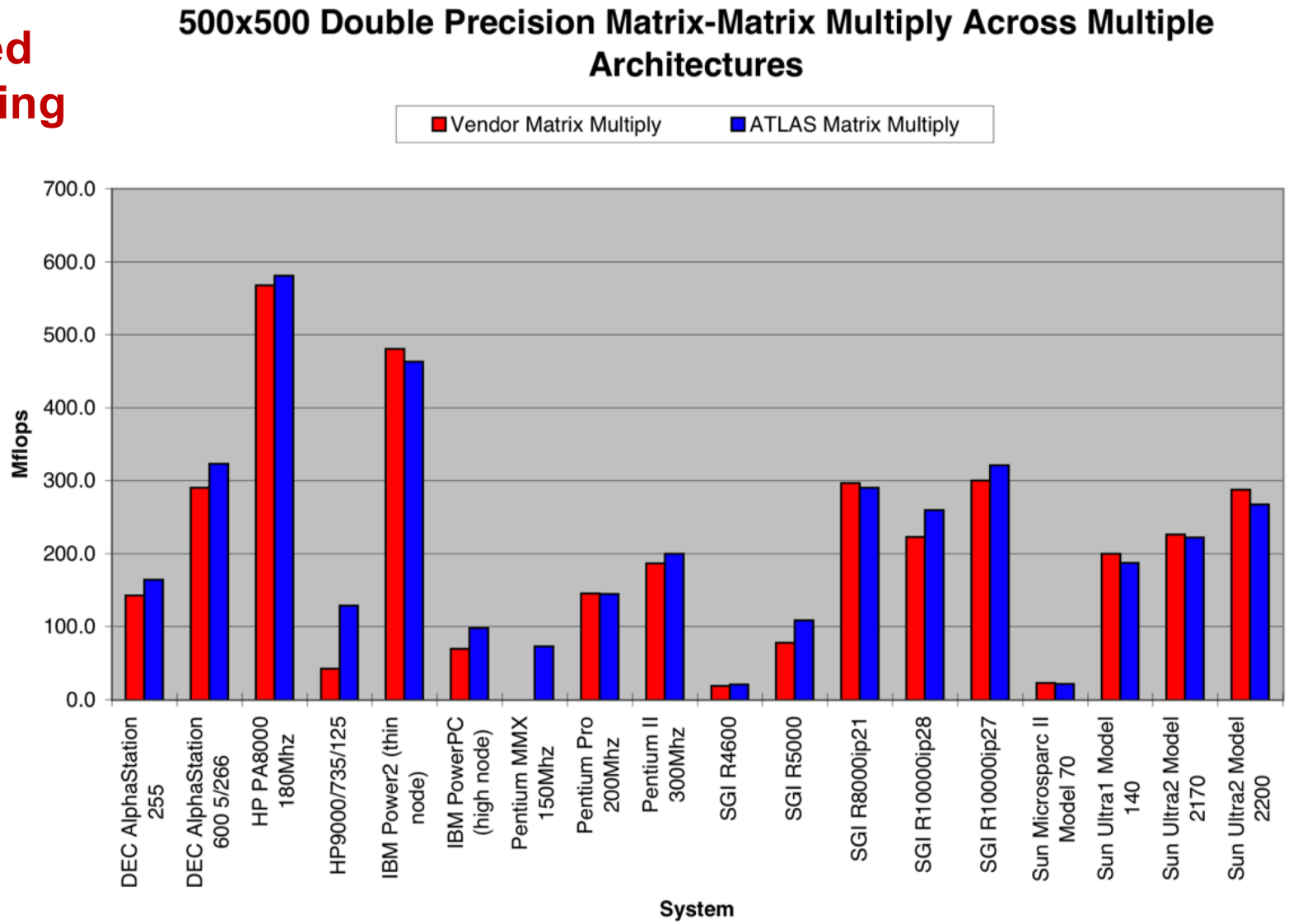
Mr. Goto, who is 37, might even be called the John Henry of the information age.

But instead of competing against a steam drill, Mr. Goto, a research associate at the Texas Advanced Computing Center at the University of Texas at Austin, has bested the work of a powerful automated system and entire teams of software developers in producing programs that run the world's fastest supercomputers.

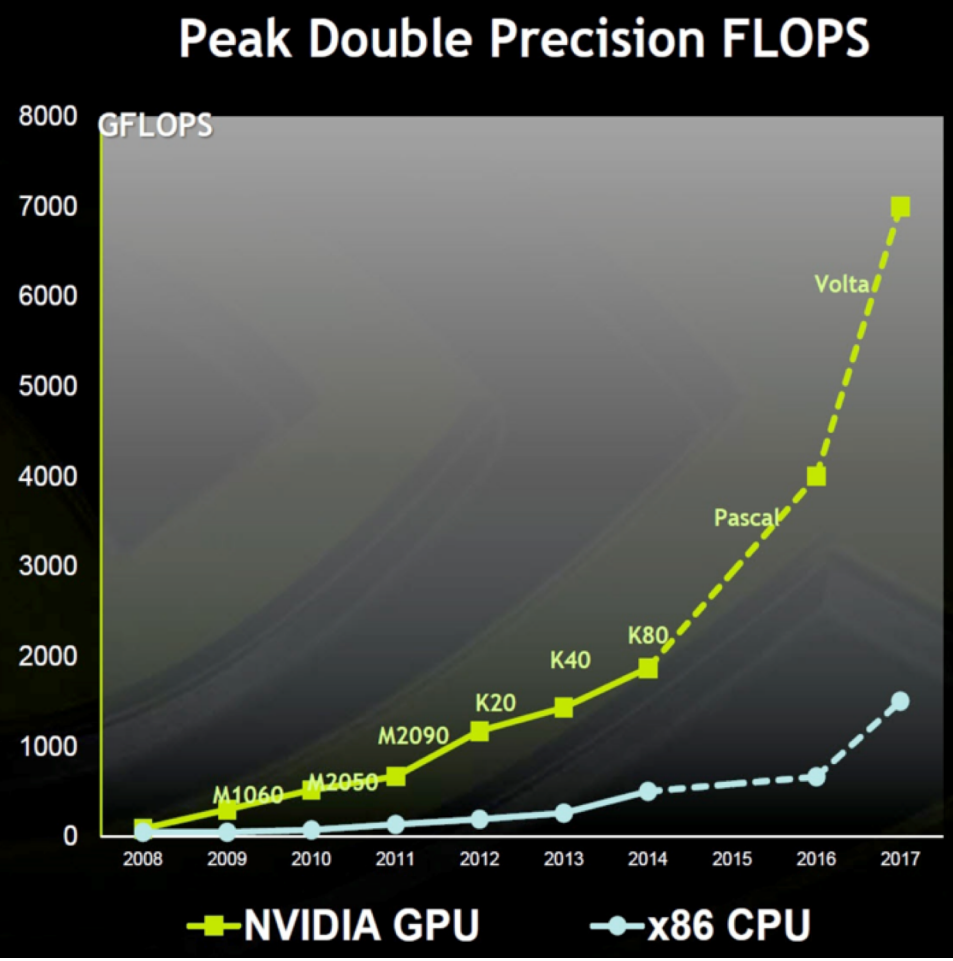
He has done it alone at his keyboard the old-fashioned way -- by writing code that reorders, one at a time, the instructions given to microprocessor chips.

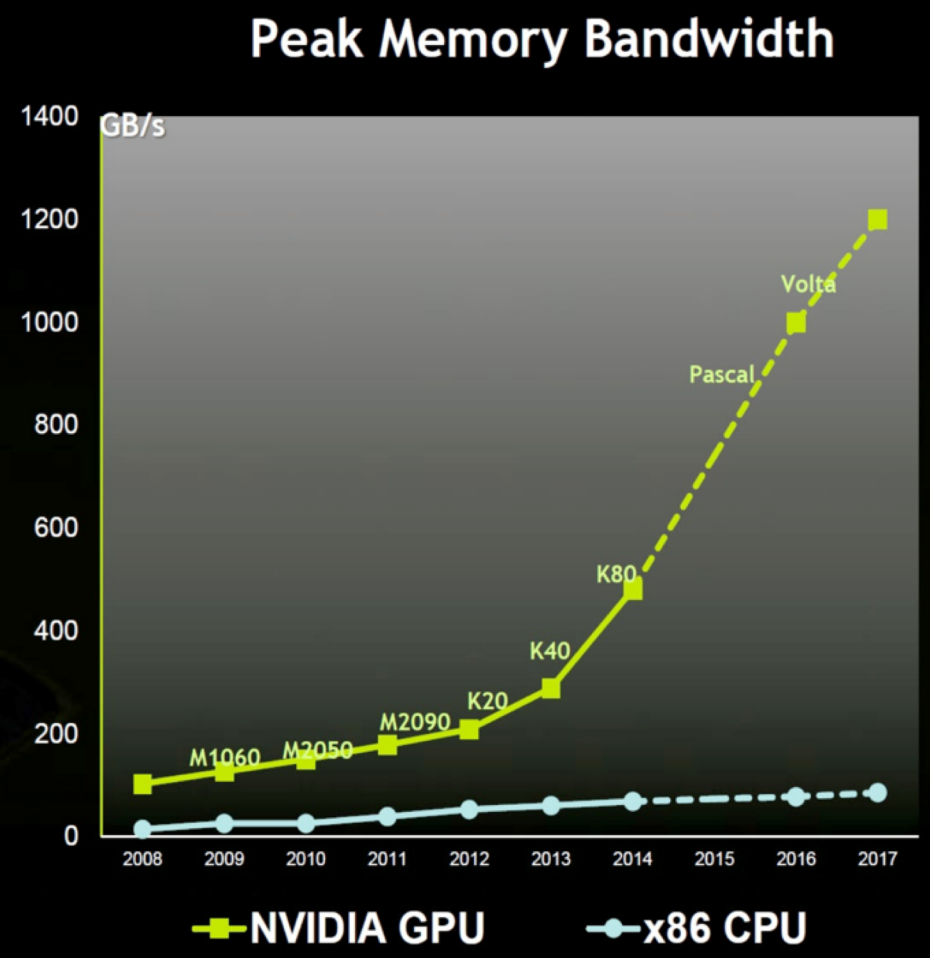


ATLAS automated performance tuning

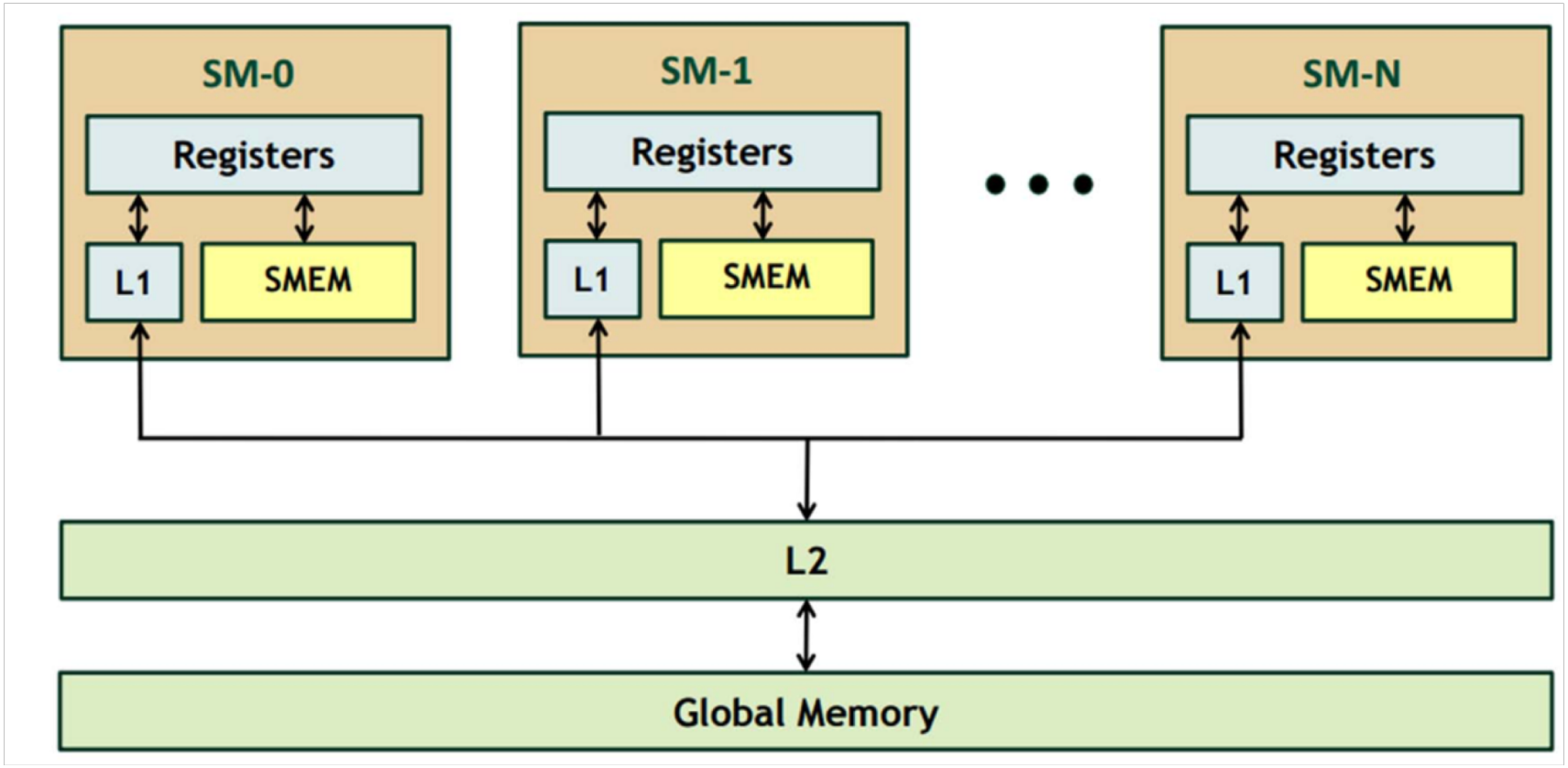


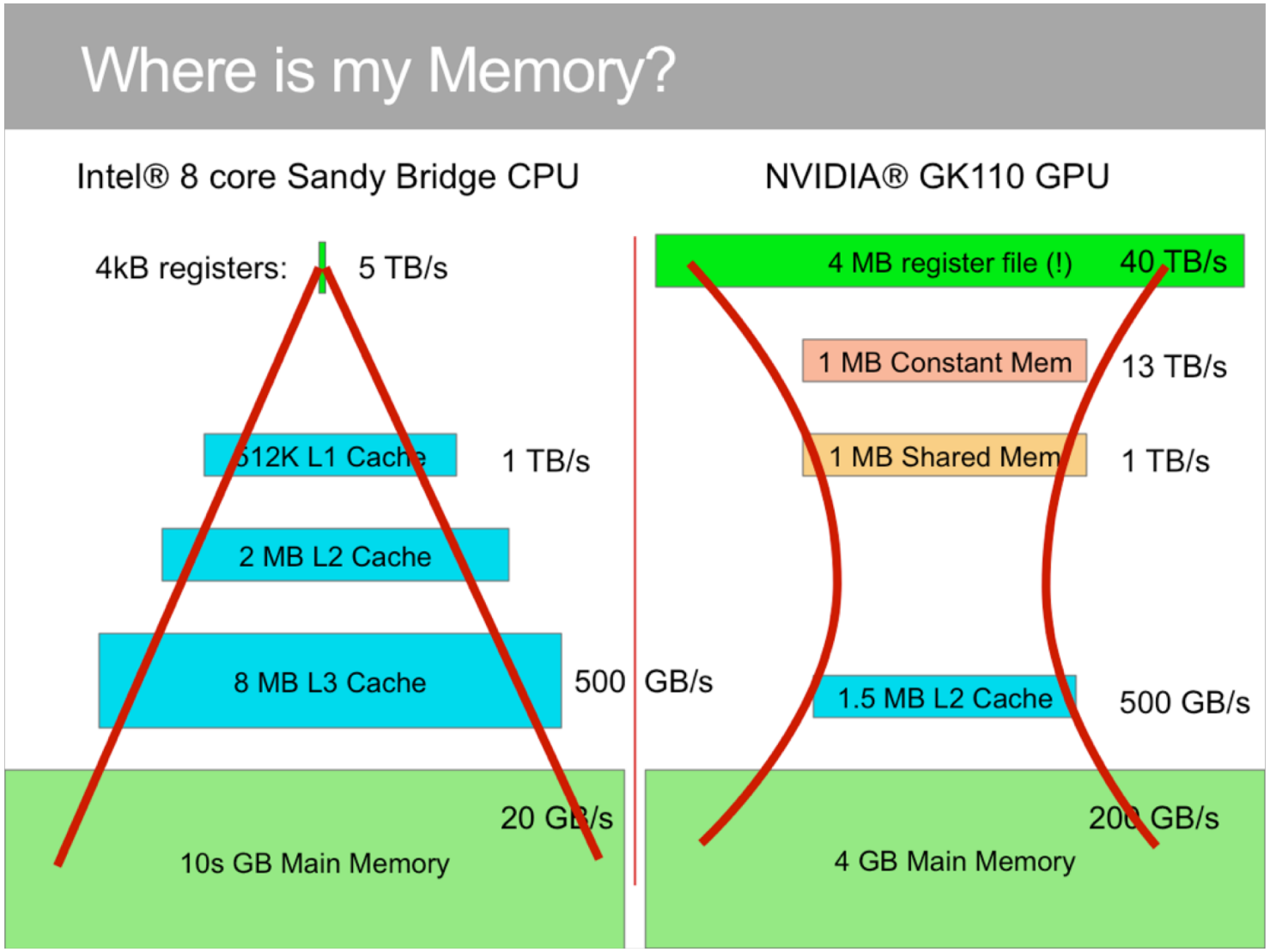
GPU Motivation (I): Performance Trends





GPU memory hierarchy: NVIDIA Fermi

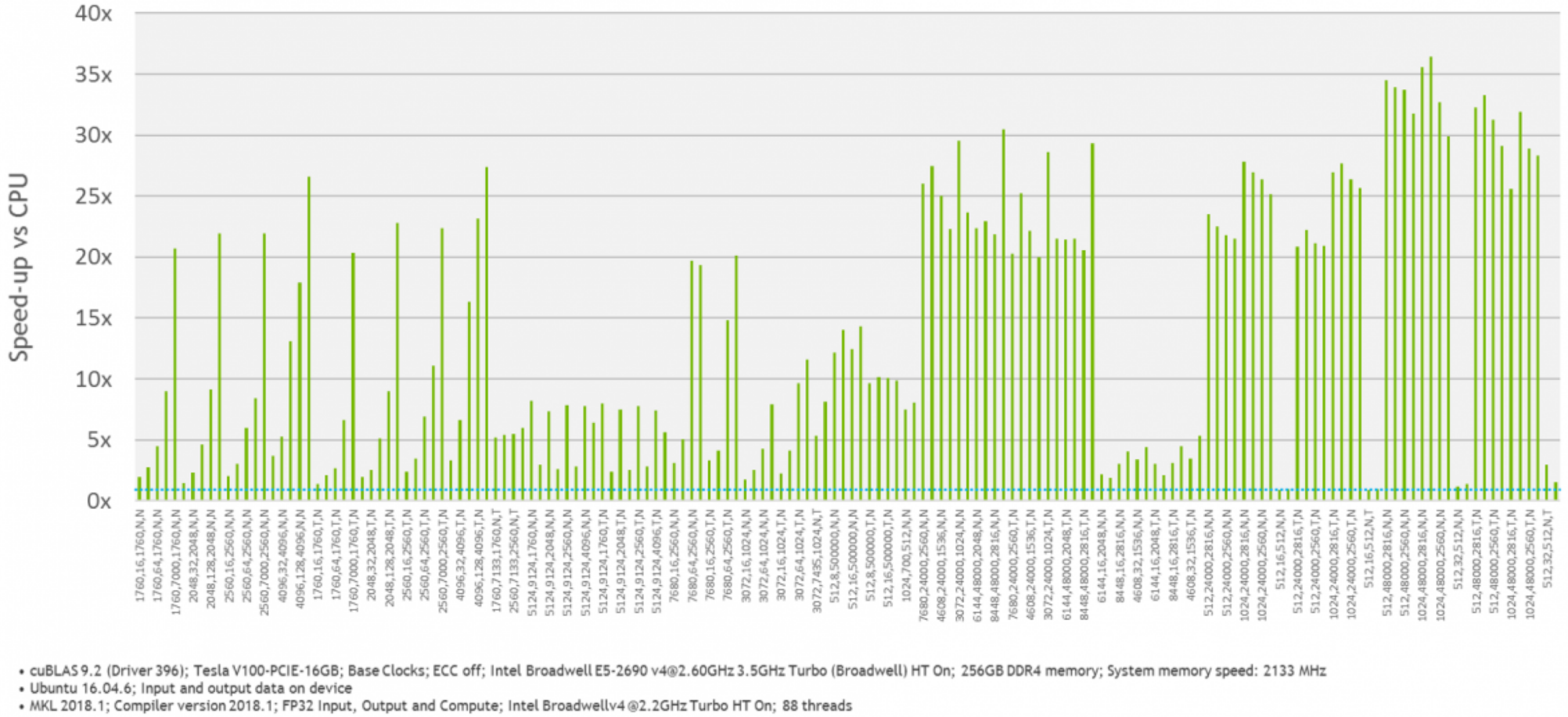




https://devblogs.nvidia.com/bidmach-machine-learning-limit-gpus/

cuBLAS 9.2: UP TO 35x FASTER DEEPBENCH SGEMM THAN CPU

C := alpha * A * B + beta * C



- cuBLAS 9.2 (Driver 396); Tesla V100-PCIE-16GB; Base Clocks; ECC off; Intel Broadwell E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On; 256GB DDR4 memory; System memory speed: 2133 MHz
- Ubuntu 16.04.6; Input and output data on device
- MKL 2018.1; Compiler version 2018.1; FP32 Input, Output and Compute; Intel Broadwellv4 @2.2GHz Turbo HT On; 88 threads

https://developer.nvidia.com/cublas

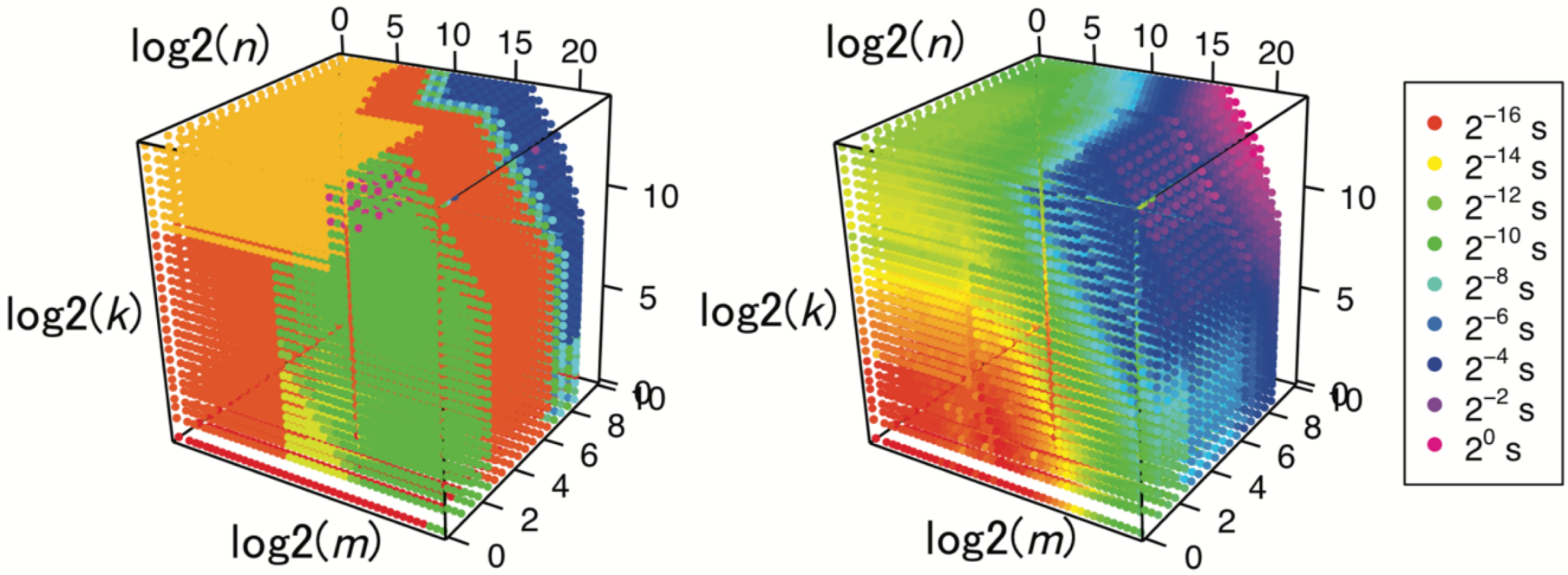
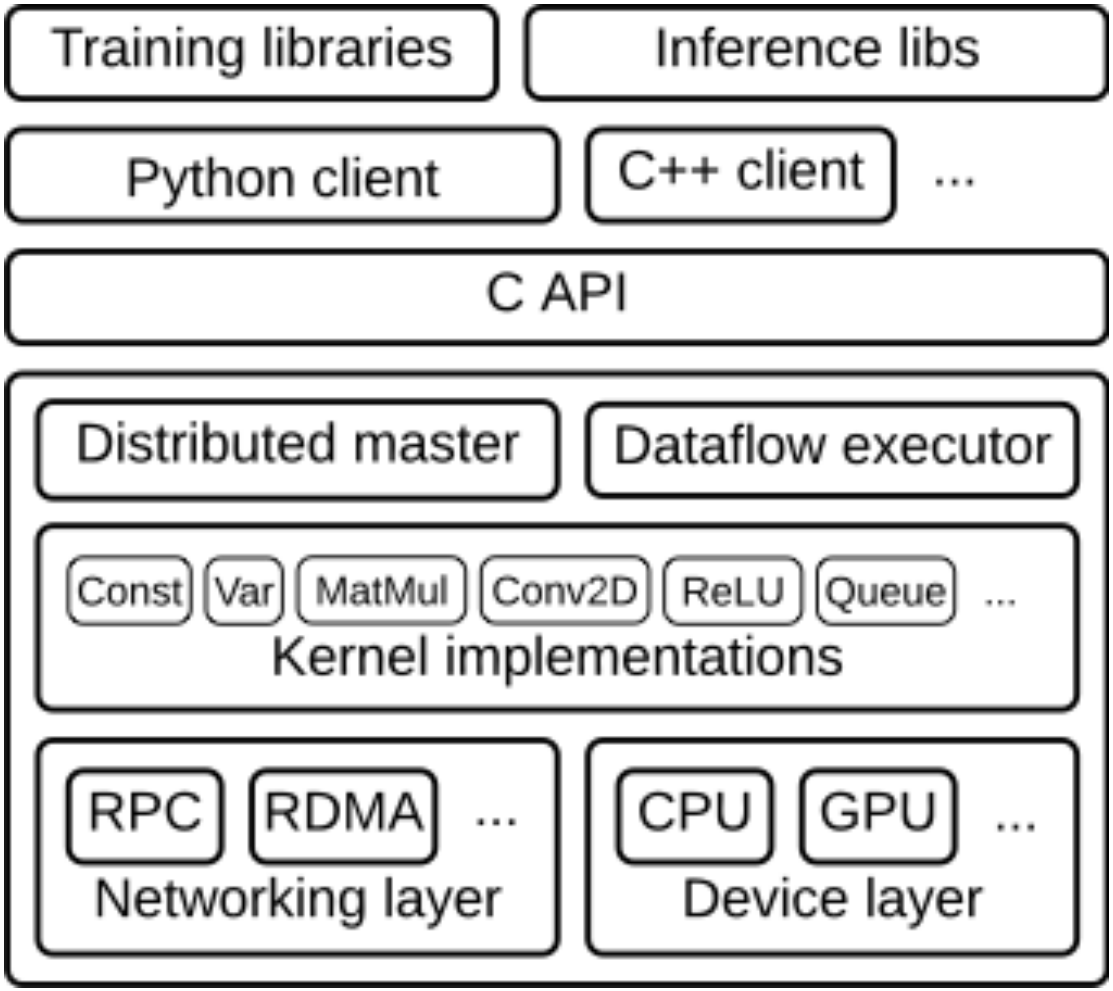


Fig. 4. Combination of kernels (left) and their computation time (right) of non-transposed *cublasSgemm*: Where $m \times k$ is the size of matrix A and $k \times n$ is that of B, where $C = AB$. There are 15 combinations of kernels in our experiment. The computation time shows average time of 5 times execution on NVIDIA Tesla K80.

https://ieeexplore.ieee.org/document/7840590

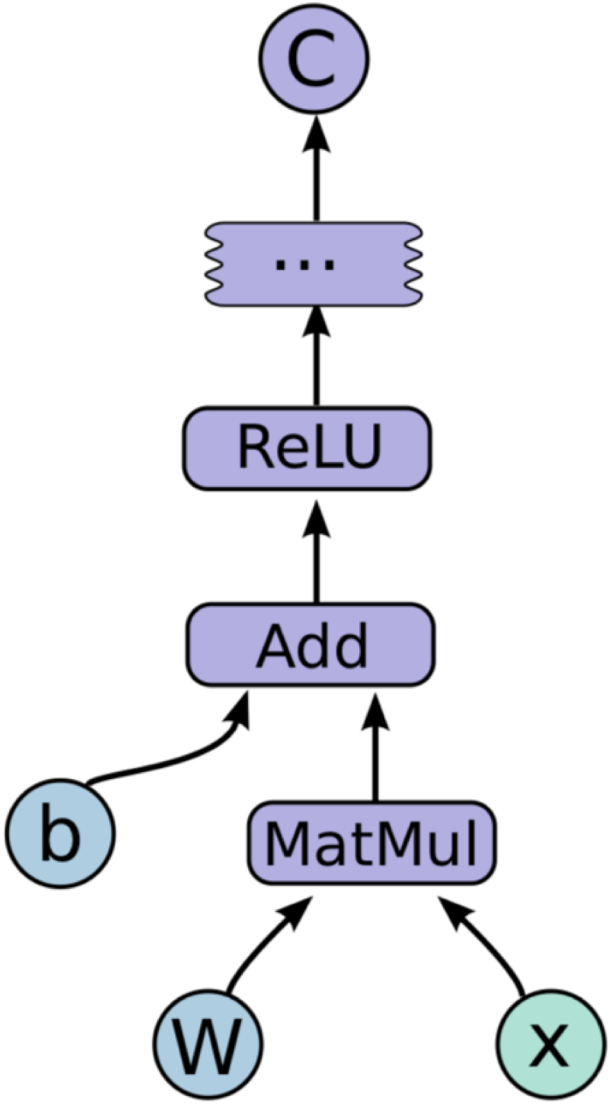
Tensorflow architecture



https://www.tensorflow.org/extend/architecture

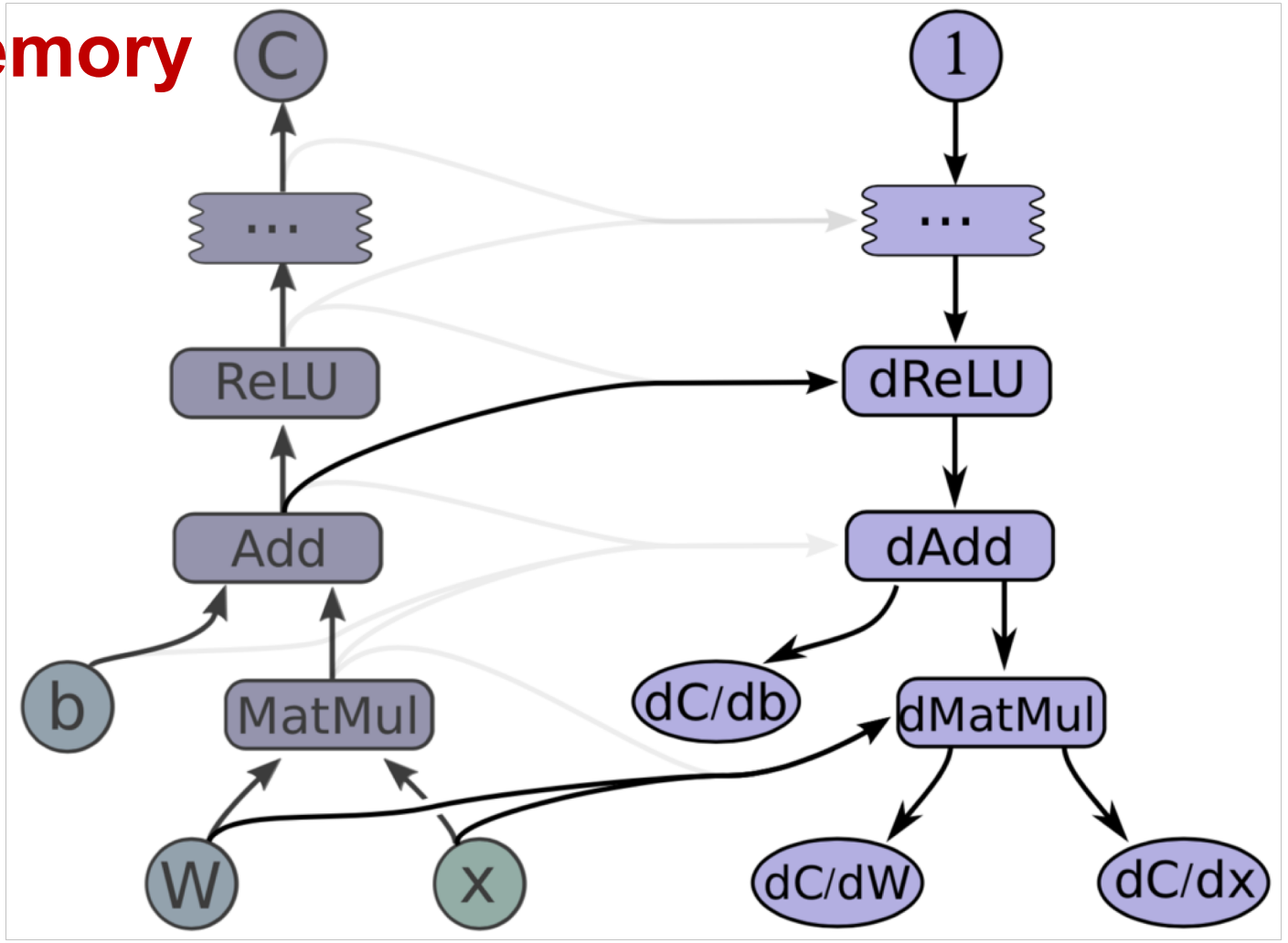
TensorFlow execution

- Nodes become executable when all required inputs are available
- Order not otherwise defined



https://arxiv.org/pdf/1603.04467.pdf

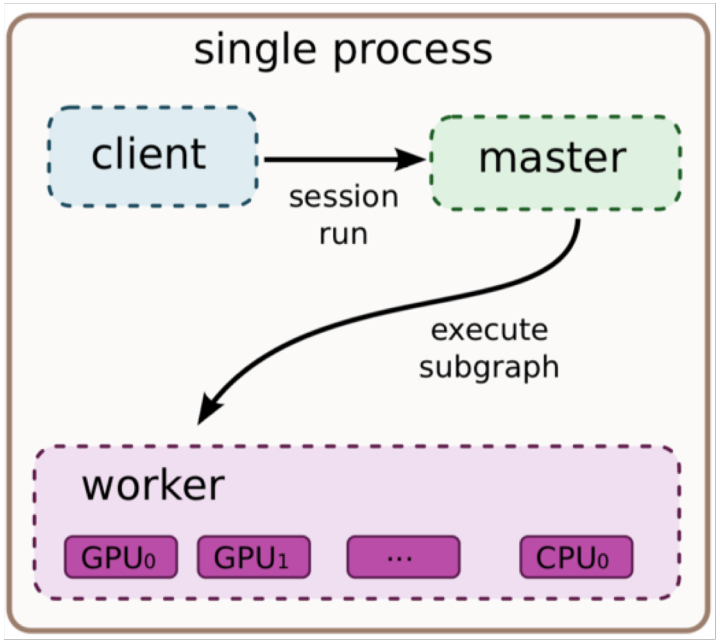
Potential for memory management challenges



[db,dW,dx] = tf.gradients(C, [b,W,x])

https://arxiv.org/pdf/1603.04467.pdf

Multiple devices



- TensorFlow places data and computation on devices based on a model of data movement and computation costs
- Users can provide guidance

Devices are named: e.g., /job:localhost/device:cpu:0

https://arxiv.org/pdf/1603.04467.pdf

Determining placement: MacBook

```
import tensorflow as tf
# Create a graph
a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Create a session with log_device_placement set to True
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op
print(sess.run(c))
```

```
MatMul: (MatMul): /job:localhost/replica:0/task:0/device:CPU:0
a: (Const): /job:localhost/replica:0/task:0/device:CPU:0
b: (Const): /job:localhost/replica:0/task:0/device:CPU:0
[[22. 28.]
 [49. 64.]]
```

On a system with a GPU

```
Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla K40c, pci bus
id: 0000:05:00.0
b: /job:localhost/replica:0/task:0/device:GPU:0
a: /job:localhost/replica:0/task:0/device:GPU:0
MatMul: /job:localhost/replica:0/task:0/device:GPU:0
[[ 22.  28.]
 [ 49.  64.]]
```

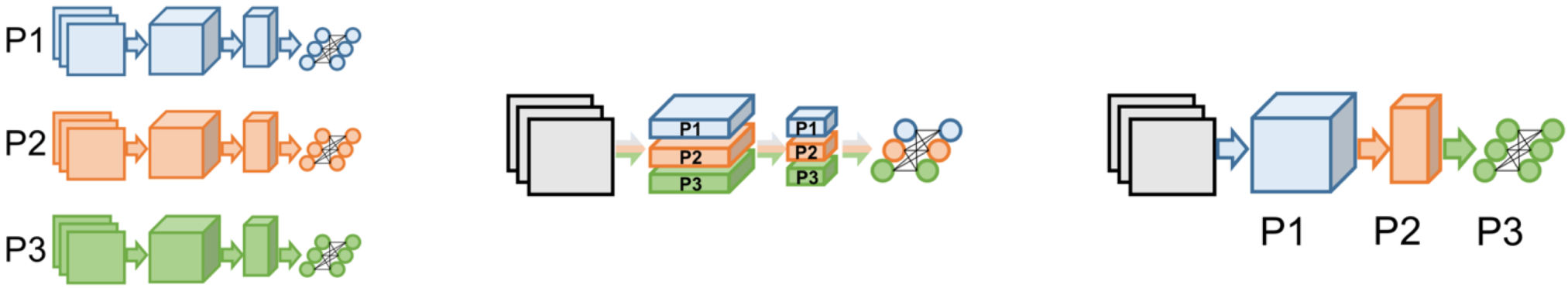
Manual placement

```
# Create a graph
with tf.device('/cpu:0'):
  a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
  b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Create a session with log_device_placement set to True
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Run the op
print(sess.run(c))

Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla
K40c, pci bus
id: 0000:05:00.0
b: /job:localhost/replica:0/task:0/cpu:0
a: /job:localhost/replica:0/task:0/cpu:0
MatMul: /job:localhost/replica:0/task:0/device:GPU:0
[[ 22.  28.]
 [ 49.  64.]]
```

Manual placement on two GPUs

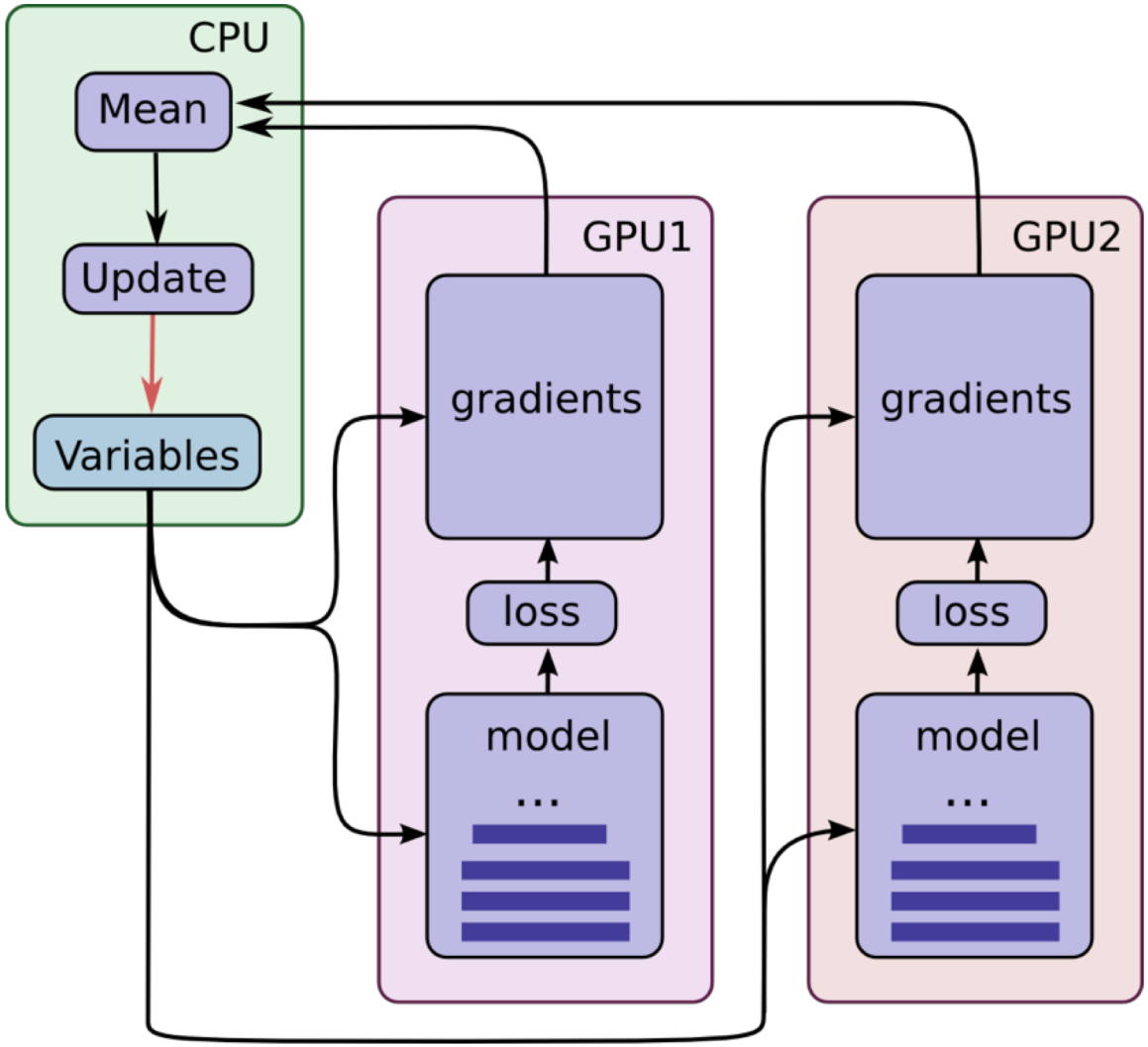
```
# Create a graph
c = []
for d in ['/device:GPU:2', '/device:GPU:3']:
  with tf.device(d):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
    c.append(tf.matmul(a, b))
with tf.device('/cpu:0'):
  sum = tf.add_n(c)
# Create a session with log_device_placement set to True
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Run the op
print(sess.run(sum))
```



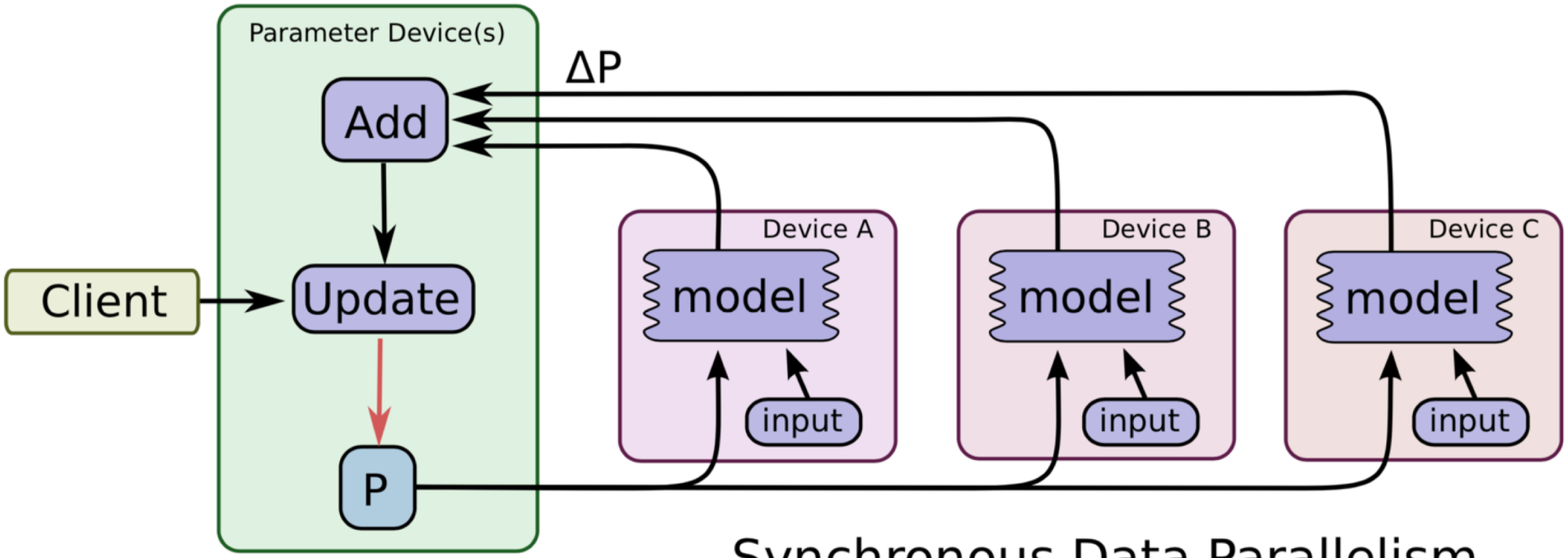
(a) Data Parallelism (b) Model Parallelism (c) Layer Pipelining

Fig. 14. Neural Network Parallelism Schemes

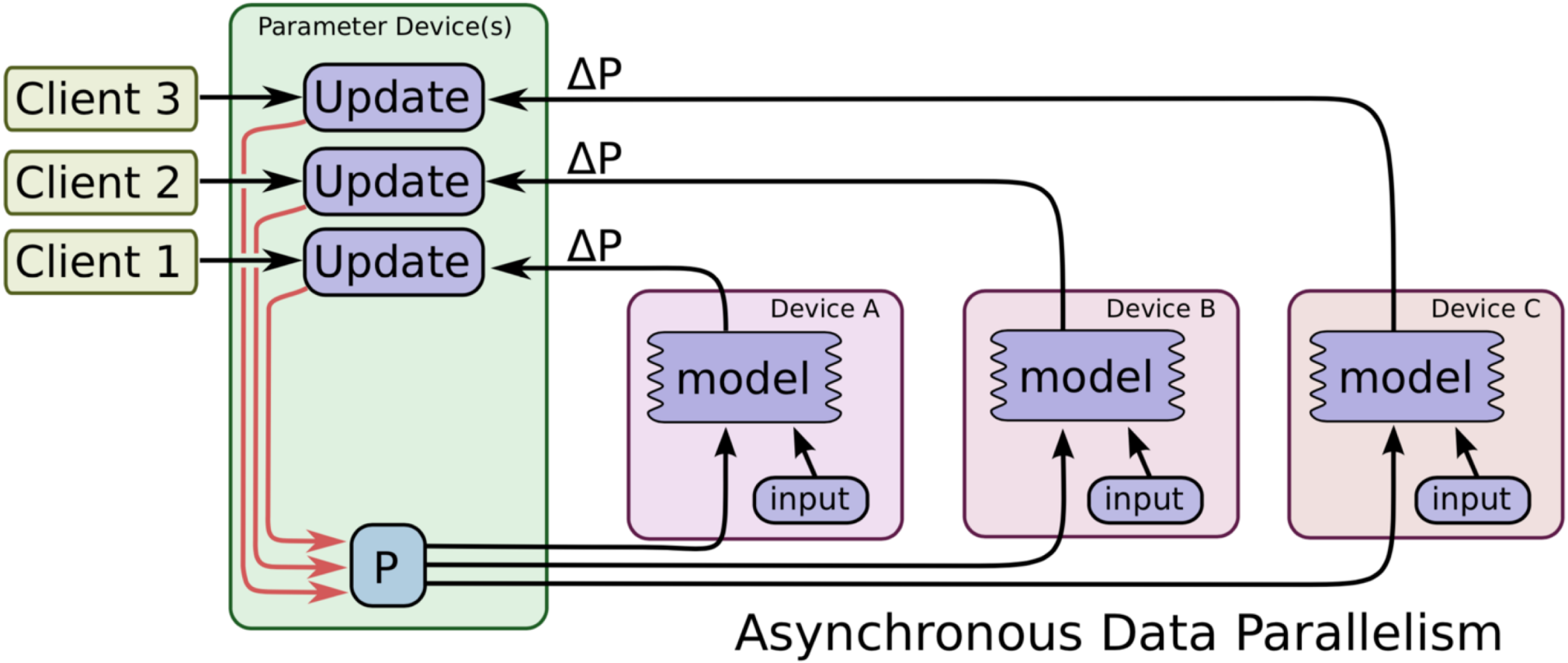
https://arxiv.org/pdf/1802.09941.pdf



https://arxiv.org/pdf/1603.04467.pdf



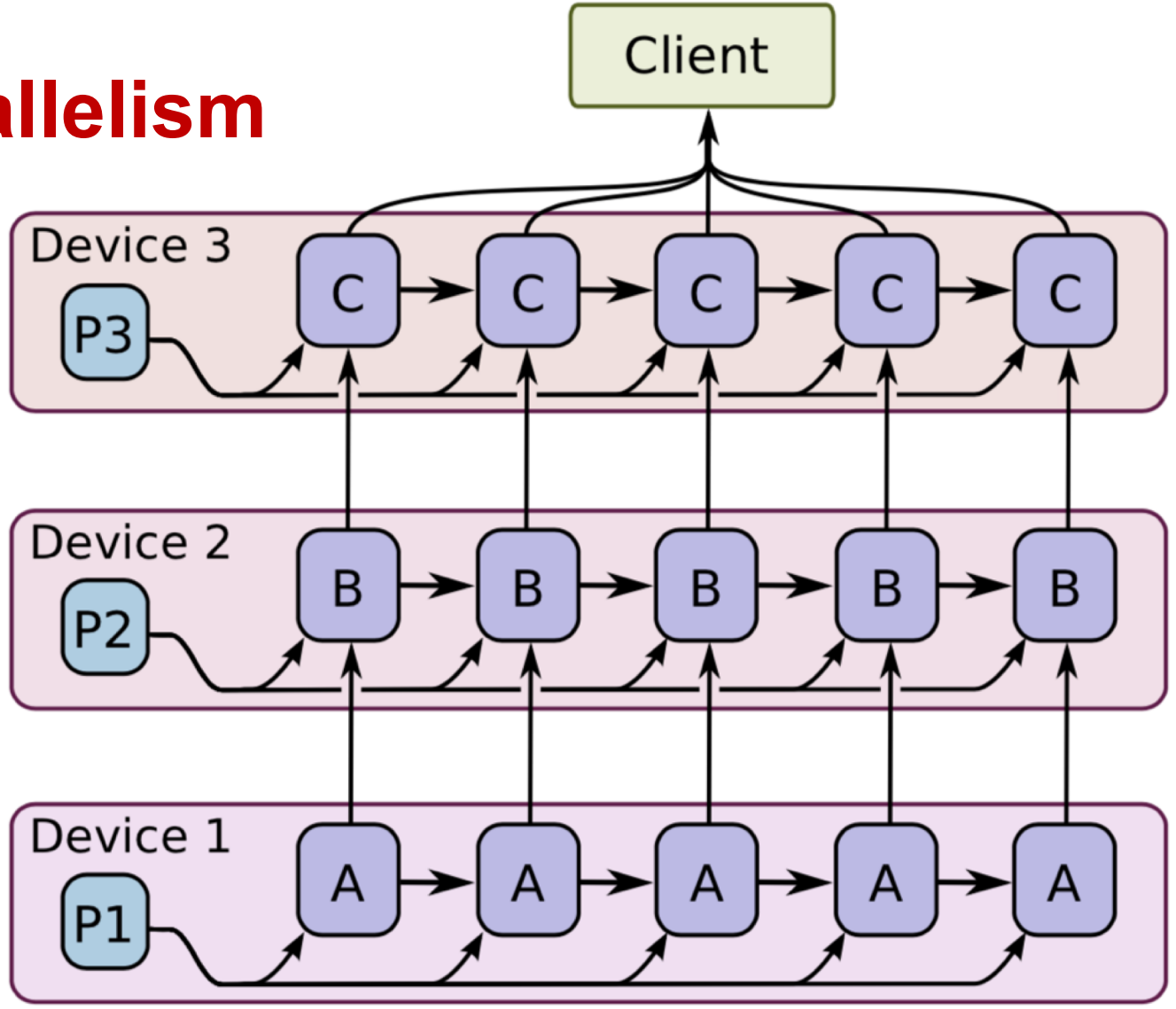
Synchronous Data Parallelism



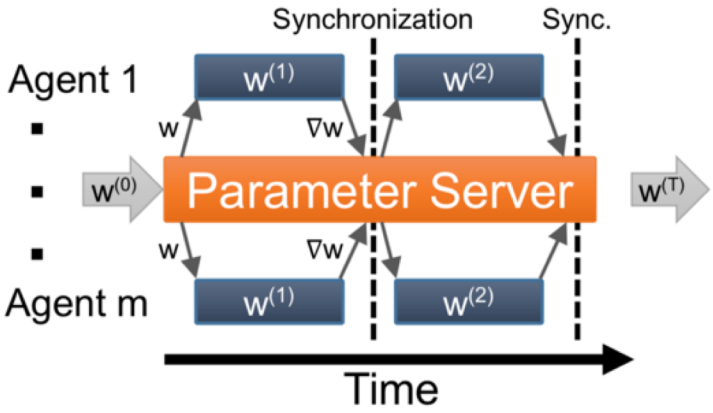
https://arxiv.org/pdf/1603.04467.pdf

Model parallelism

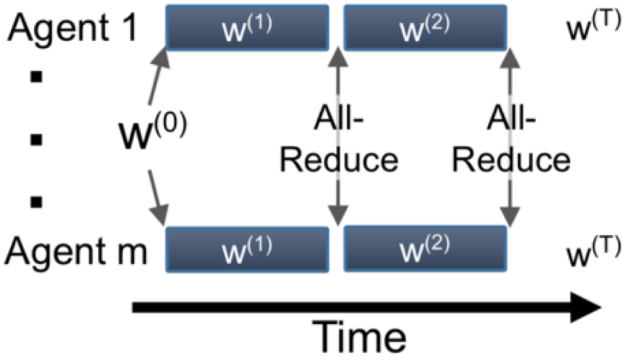
A recurrent, deep LSTM model parallelized across three devices



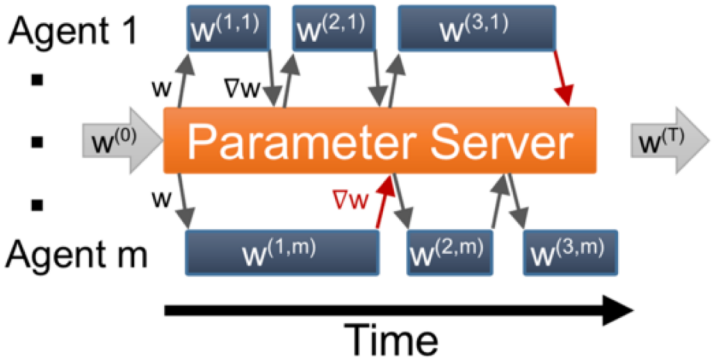
https://arxiv.org/pdf/1603.04467.pdf



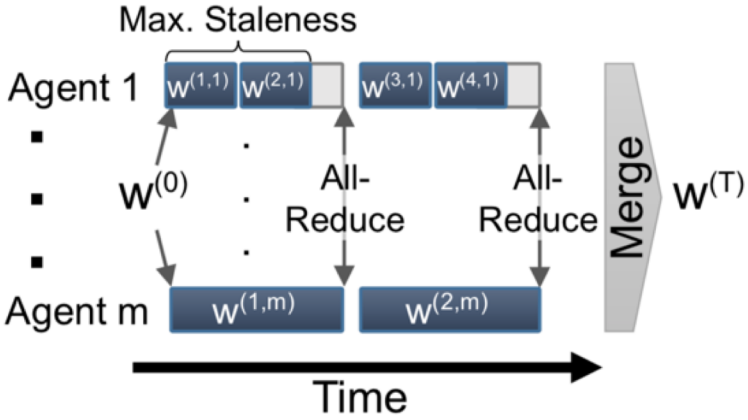
(a) Synchronous, Parameter Server



(b) Synchronous, Decentralized



(c) Asynchronous, Parameter Server



(d) Stale-Synchronous, Decentralized

Fig. 20. Training Distribution in Deep Learning (Model Consistency, Centralization)

https://arxiv.org/pdf/1802.09941.pdf

Table 1: Open-source Frameworks

Platform	Tensorflow	CNTK	Deeplearning4j	MXNet	H2O	Caffe	Theano	Torch
Release Date	2016	2016	2015	2015	2014	2014	2010	2011 (deep learning)
Core Language	C++	C++	C++	C++	Java	C++	C++	C
API	C++, Python	NDL	Java, Scala	C++, Python, R, Scala, Matlab, Javascript, Go, Julia	Java, R, Python, Scala, Javascript, web-UI	Python, Matlab	Python	Lua
Synchronization Model	Sync or async	Sync	Sync	Sync or async	Async	Sync	Async	Sync
Communication Model	Parameter server	MPI	Iterative MapReduce	Parameter server	Distributed fork-join	N/A	N/A	N/A
Multi-GPU	✓	✓	✓	✓	✓	✓	✓	✓
Multi-node	✓	✓	✓	✓	✓	✗	✗	✗
Data Parallelism	✓	✓	✓	✓	✓	✓	✓	✓
Model Parallelism	✓	N/A	✗	✓	✗	✗	✓	✓
Deep Learning Models	DBN, CNN, RNN	DBN, CNN, RNN	DBN, CNN, RNN	DBN, CNN, RNN	DBN	DBN, CNN, RNN	DBN, CNN, RNN	DBN, CNN, RNN
Programming Paradigm	Imperative	Imperative	Declarative	Both	Declarative	Declarative	Imperative	Imperative
Fault Tolerance	Checkpoint-and-recovery	Checkpoint-and-resume	Checkpoint-and-resume	Checkpoint-and-resume	N/A	N/A	Checkpoint-and-resume	Checkpoint-and-resume
Visualization	Graph (interactive), training monitoring	Graph (static)	Training monitoring	None	None	Summary Statistics	Graph (static)	Plots

Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

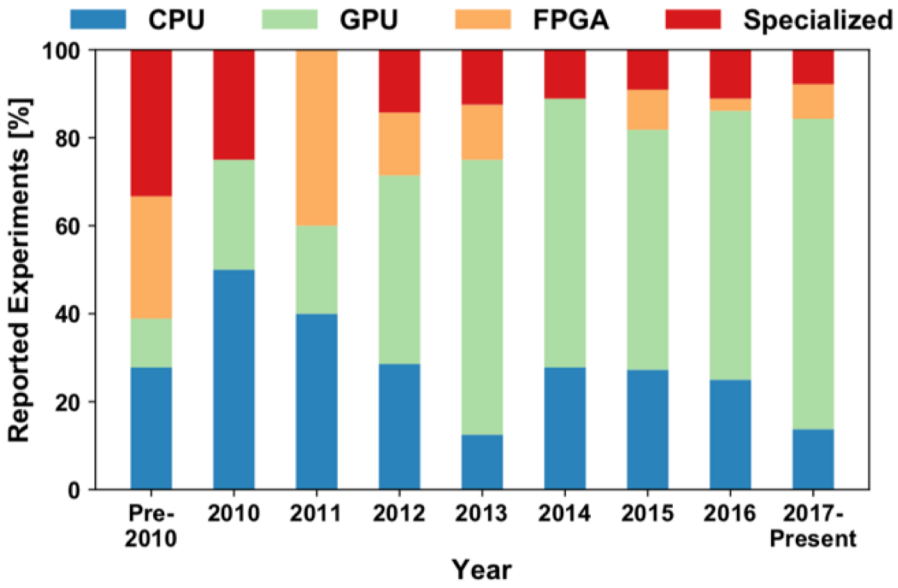
TAL BEN-NUN and TORSTEN HOEFLER, ETH Zurich, Switzerland

Deep Neural Networks (DNNs) are becoming an important tool in modern computing applications. Accelerating their training is a major challenge and techniques range from distributed algorithms to low-level circuit design. In this survey, we describe the problem from a theoretical perspective, followed by approaches for its parallelization. We present trends in DNN architectures and the resulting implications on parallelization strategies. We then review and model the different types of concurrency in DNNs: from the single operator, through parallelism in network inference and training, to distributed deep learning. We discuss asynchronous stochastic optimization, distributed system architectures, communication schemes, and neural architecture search. Based on those approaches, we extrapolate potential directions for parallelism in deep learning.

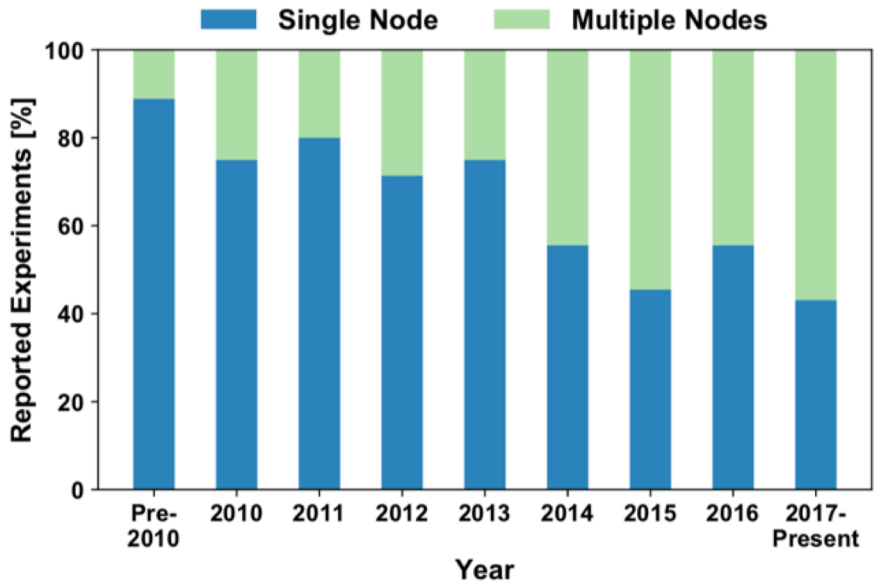
CCS Concepts: • **General and reference** → *Surveys and overviews*; • **Computing methodologies** → **Neural networks**; **Parallel computing methodologies**; **Distributed computing methodologies**;

Additional Key Words and Phrases: Deep Learning, Distributed Computing, Parallel Algorithms

ACM Reference Format:
Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. 47 pages.



(a) Hardware Architectures

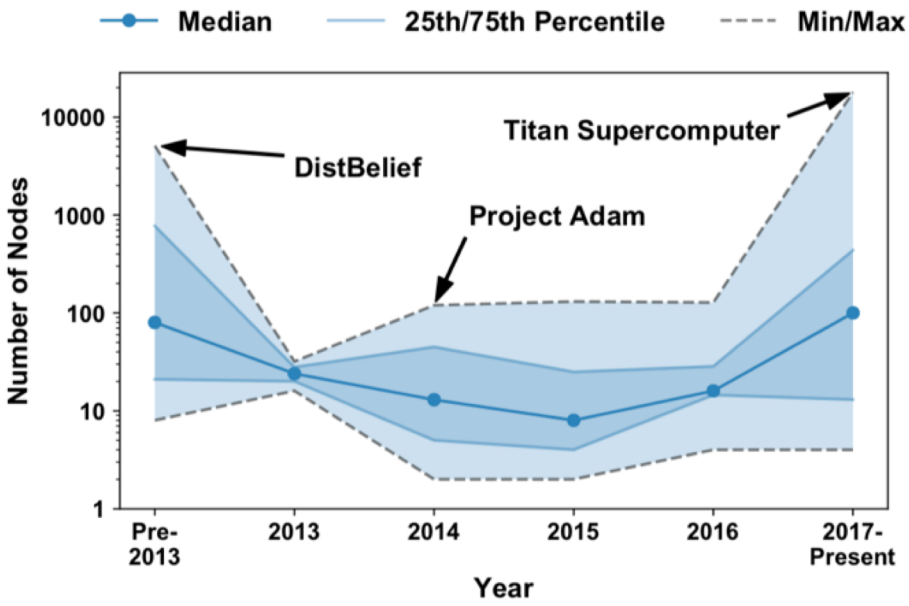


(b) Training with Single vs. Multiple Nodes

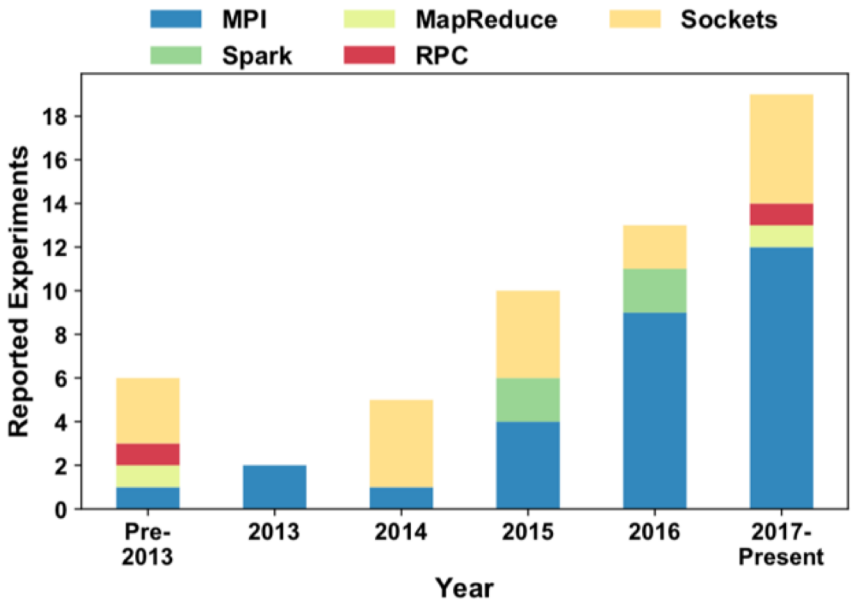
Fig. 3. Parallel Architectures in Deep Learning

(based on a review of 240 papers from the literature)

https://arxiv.org/pdf/1802.09941.pdf



(a) Node Count



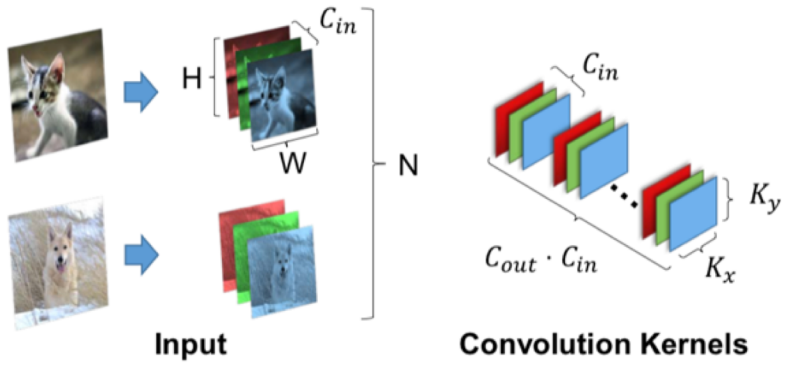
(b) Communication Layer

Fig. 4. Characteristics of Deep Learning Clusters

https://arxiv.org/pdf/1802.09941.pdf

Name	Description
N	Minibatch size
C	Number of channels, features, or neurons
H	Image Height
W	Image Width
K_x	Convolution kernel width
K_y	Convolution kernel height

(a) Data Dimensions



(b) Convolution Dimensions

Fig. 8. Summary of Data Dimensions in Operators

https://arxiv.org/pdf/1802.09941.pdf

Table 4. Asymptotic Work-Depth Characteristics of DNN Operators

Operator Type	**Eval.**	**Work (W)**	**Depth (D)**
Activation	y	$O(NCHW)$	$O(1)$
	∇w	$O(NCHW)$	$O(1)$
	∇x	$O(NCHW)$	$O(1)$
Fully Connected	y	$O(C_{out} \cdot C_{in} \cdot N)$	$O(\log C_{in})$
	∇w	$O(C_{in} \cdot N \cdot C_{out})$	$O(\log N)$
	∇x	$O(C_{in} \cdot C_{out} \cdot N)$	$O(\log C_{out})$
Convolution (Direct)	y	$O(N \cdot C_{out} \cdot C_{in} \cdot H' \cdot W' \cdot K_x \cdot K_y)$	$O(\log K_x + \log K_y + \log C_{in})$
	∇w	$O(N \cdot C_{out} \cdot C_{in} \cdot H' \cdot W' \cdot K_x \cdot K_y)$	$O(\log K_x + \log K_y + \log C_{in})$
	∇x	$O(N \cdot C_{out} \cdot C_{in} \cdot H \cdot W \cdot K_x \cdot K_y)$	$O(\log K_x + \log K_y + \log C_{in})$
Pooling	y	$O(NCHW)$	$O(\log K_x + \log K_y)$
	∇w	–	–
	∇x	$O(NCHW)$	$O(1)$
Batch Normalization	y	$O(NCHW)$	$O(\log N)$
	∇w	$O(NCHW)$	$O(\log N)$
	∇x	$O(NCHW)$	$O(\log N)$