TOMTOM

CLBlast: A Tuned BLAS Library
for Faster Deep Learning

Cedric Nugteren

May 11, 2017

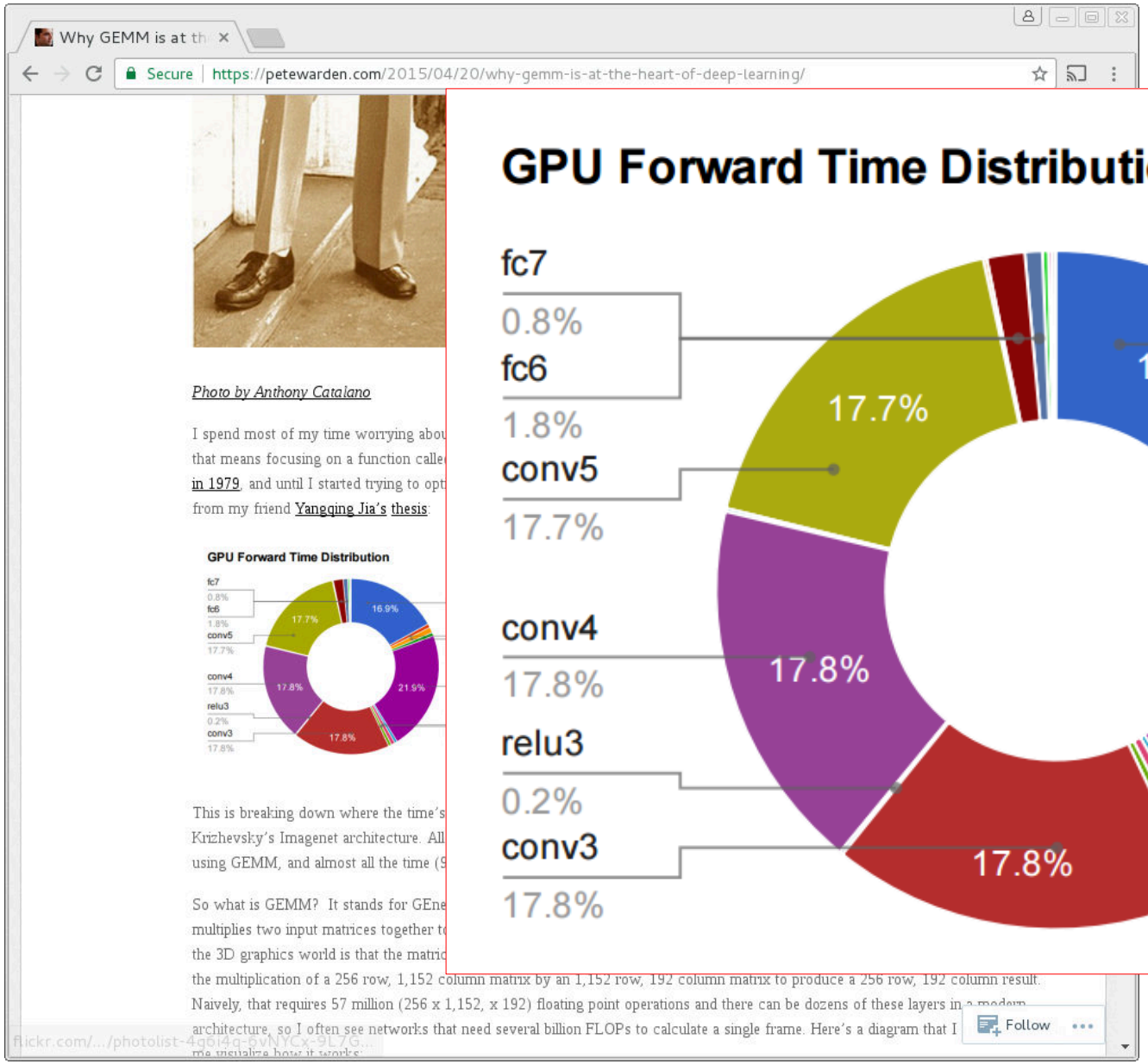
http://github.com/cnugteren/clblast

http://cnugteren.github.io/clblast

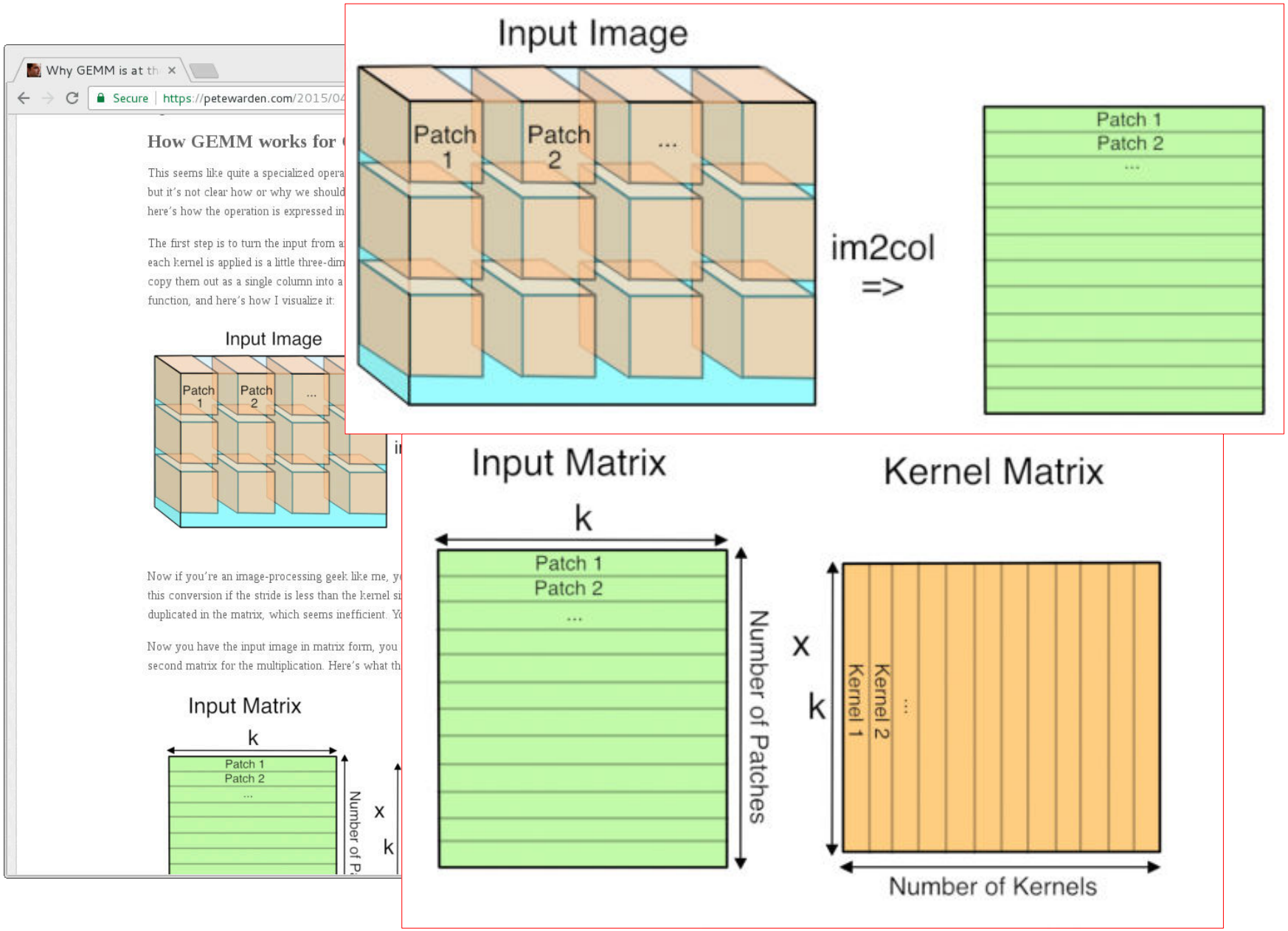
The Heart of Deep Learning

GEMM is at the Heart of Deep Learning

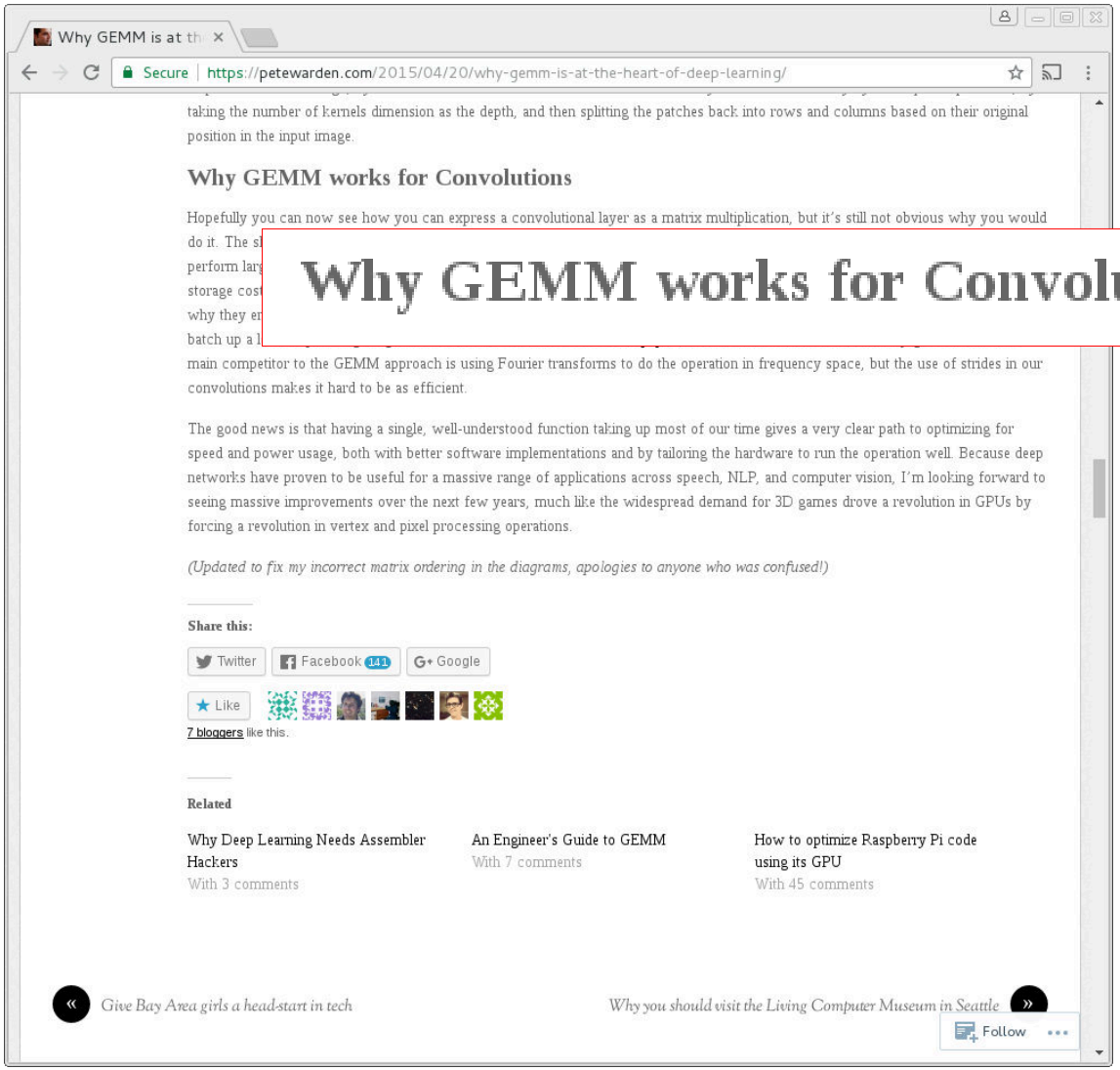
So where are the Matrix-Multiplications?



Convolutions as Matrix Multiplication

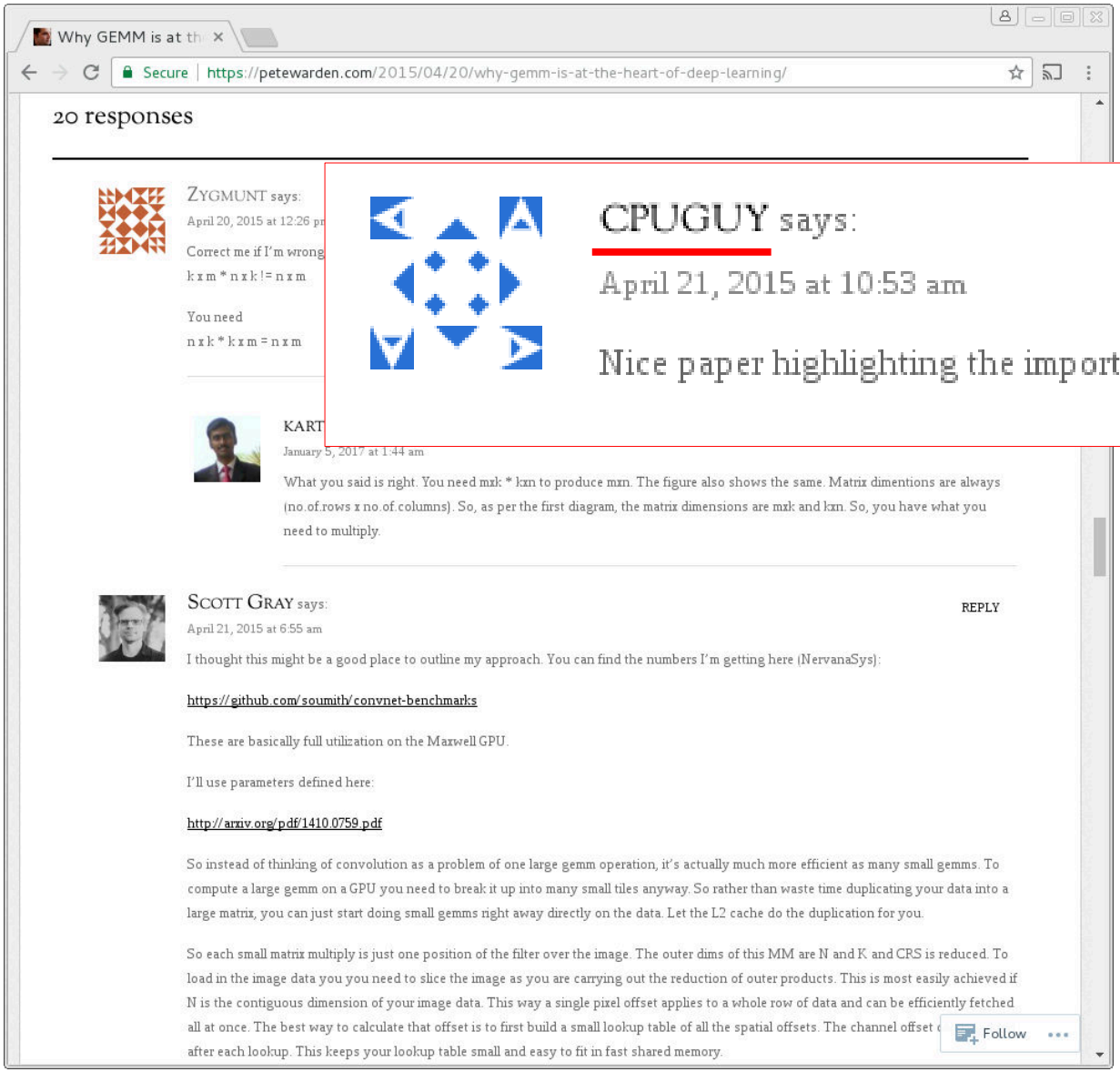


GEMM is the Heart of Deep Learning



Why GEMM works for Convolutions

Does everyone agree?



Does everyone agree?

Why GEMM is at th

Secure | https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

20 responses

ZYGMUNT says:
April 20, 2015 at 12:26 pr
Correct me if I'm wrong

CPUGUY says:

SCOTT GRAY says: REPLY

April 21, 2015 at 6:55 am

I thought this might be a good place to outline my approach. You can find the numbers I'm getting here (NervanaSys):

https://github.com/soumith/convnet-benchmarks

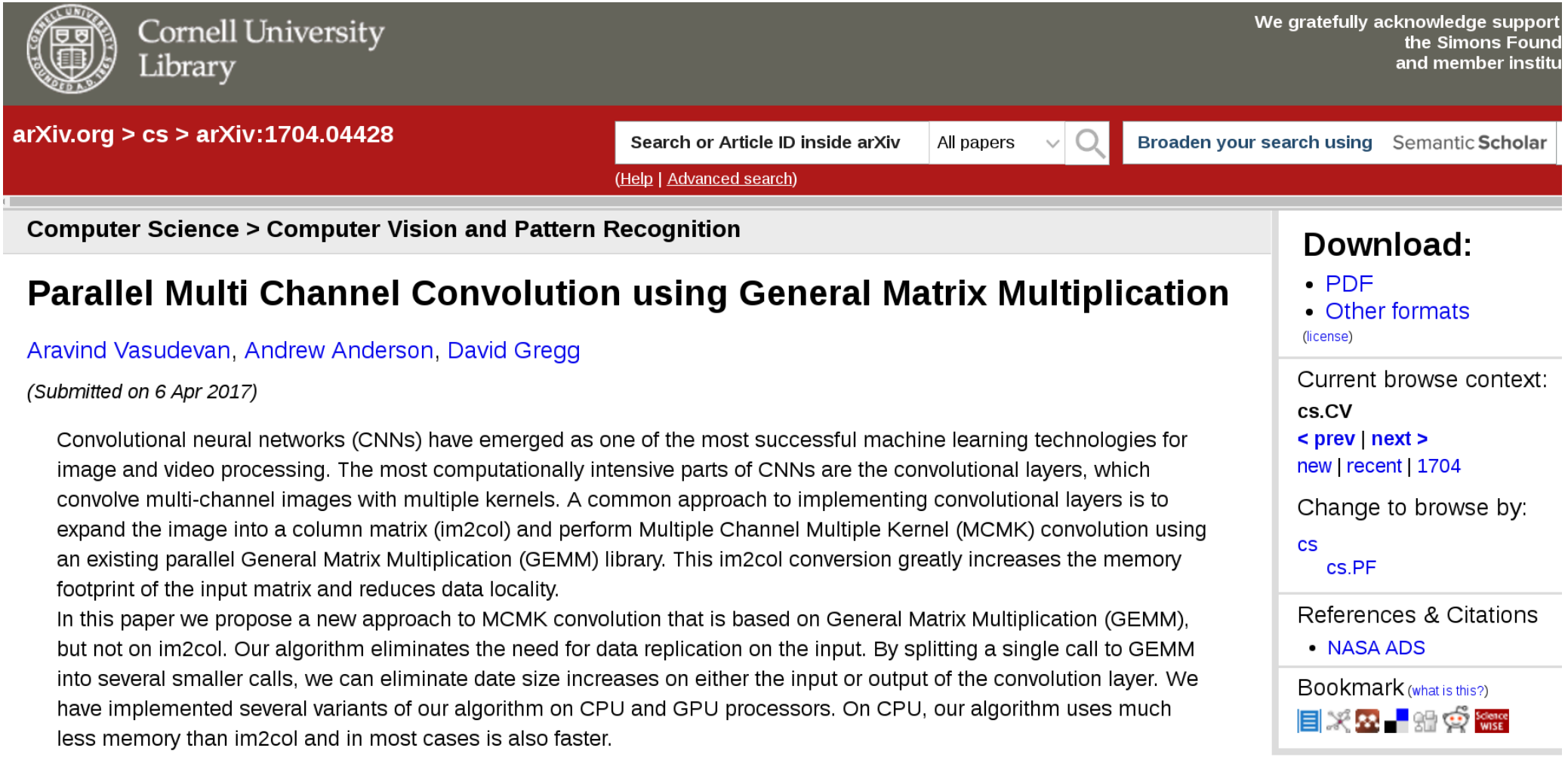
These are basically full utilization on the Maxwell GPU.

I'll use parameters defined here:

http://arxiv.org/pdf/1410.0759.pdf

So instead of thinking of convolution as a problem of one large gemm operation, it's actually much more efficient as many small gemms. To compute a large gemm on a GPU you need to break it up into many small tiles anyway. So rather than waste time duplicating your data into a large matrix, you can just start doing small gemms right away directly on the data. Let the L2 cache do the duplication for you.

Still true in 2017!



Cornell University Library

We gratefully acknowledge support the Simons Found and member institu

arXiv.org > cs > arXiv:1704.04428

Search or Article ID inside arXiv | All papers | Broaden your search using Semantic Scholar

(Help | Advanced search)

Computer Science > Computer Vision and Pattern Recognition

Parallel Multi Channel Convolution using General Matrix Multiplication

Aravind Vasudevan, Andrew Anderson, David Gregg

(Submitted on 6 Apr 2017)

Convolutional neural networks (CNNs) have emerged as one of the most successful machine learning technologies for image and video processing. The most computationally intensive parts of CNNs are the convolutional layers, which convolve multi-channel images with multiple kernels. A common approach to implementing convolutional layers is to expand the image into a column matrix (im2col) and perform Multiple Channel Multiple Kernel (MCMK) convolution using an existing parallel General Matrix Multiplication (GEMM) library. This im2col conversion greatly increases the memory footprint of the input matrix and reduces data locality.
In this paper we propose a new approach to MCMK convolution that is based on General Matrix Multiplication (GEMM), but not on im2col. Our algorithm eliminates the need for data replication on the input. By splitting a single call to GEMM into several smaller calls, we can eliminate date size increases on either the input or output of the convolution layer. We have implemented several variants of our algorithm on CPU and GPU processors. On CPU, our algorithm uses much less memory than im2col and in most cases is also faster.

Download:
- PDF
- Other formats

(license)

Current browse context:
cs.CV
< prev | next >
new | recent | 1704

Change to browse by:
cs
cs.PF

References & Citations
- NASA ADS

Bookmark (what is this?)

But why a new BLAS Library?

- NVIDIA's cuBLAS is great, or is it?

But why a new BLAS Library?

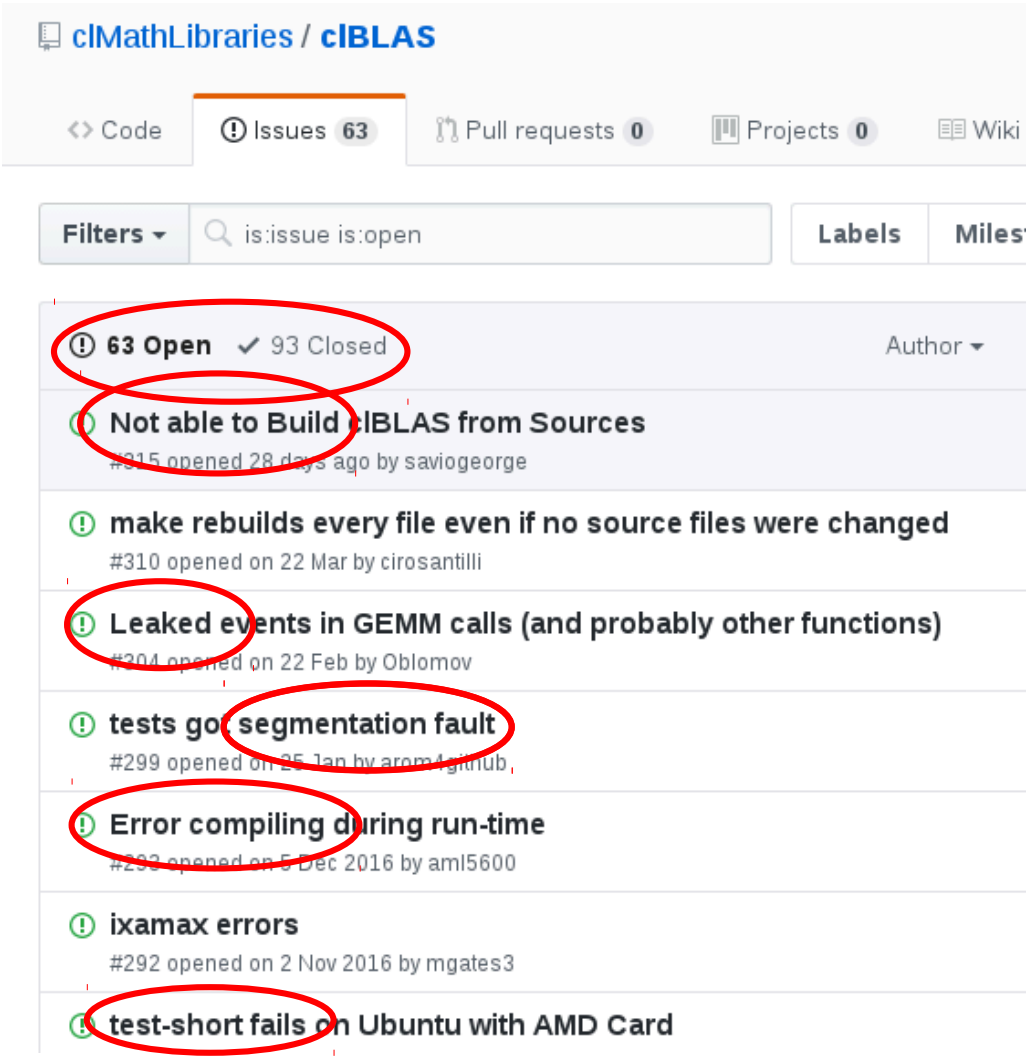
- NVIDIA's cuBLAS is great, or is it?

But why a new BLAS Library?

- NVIDIA's cuBLAS is great, or is it?
  - Not portable, not customisable, not open-source, ...

But why a new BLAS Library?

- NVIDIA's cuBLAS is great, or is it?
  - Not portable, not customisable, not open-source, ...
- Is AMD's clBLAS great?
  - Not performance portable, not well engineered, lack of new features, ...



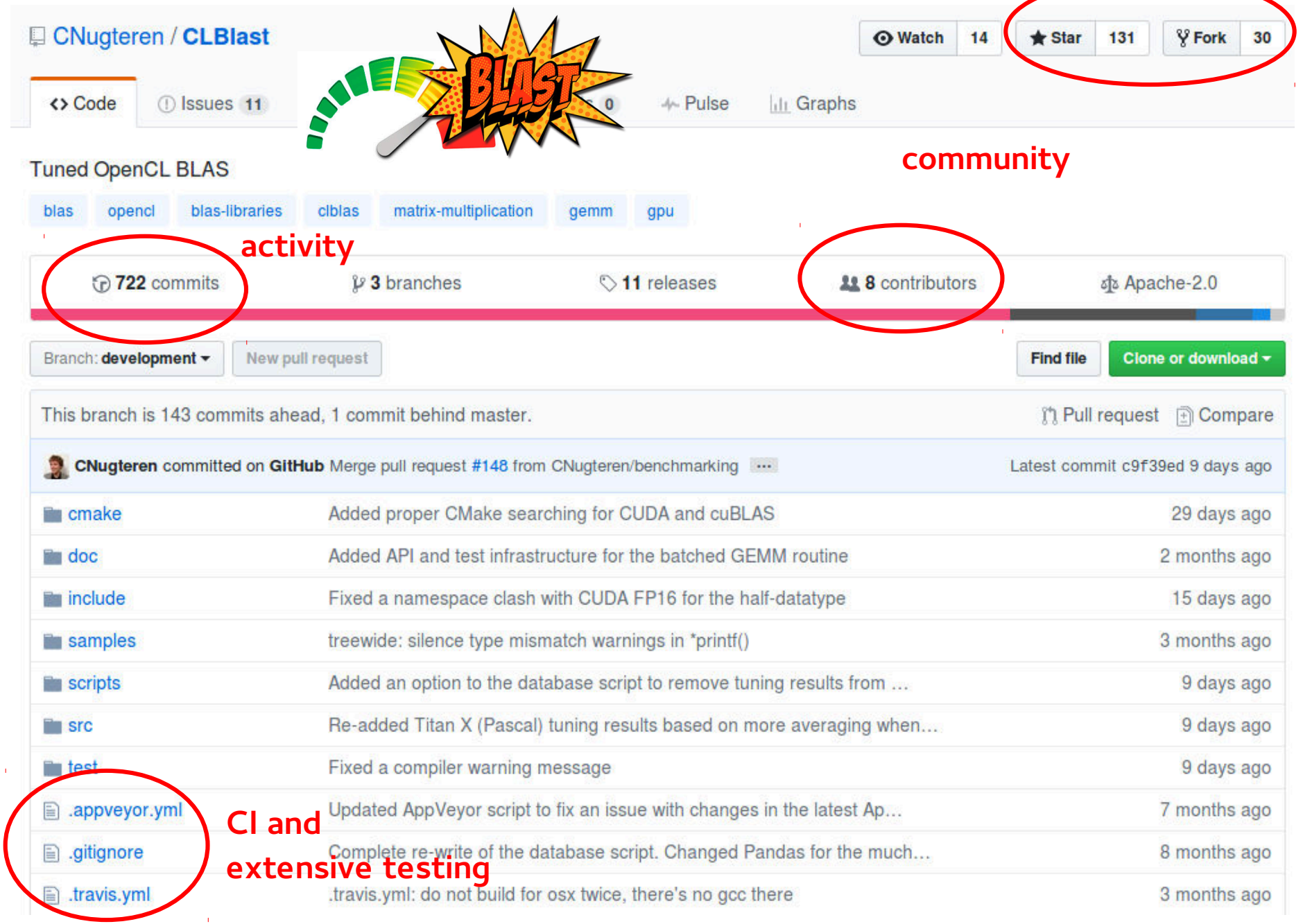
Introducing CLBlast

- CLBlast: Modern C++11 OpenCL BLAS library
- Implements all BLAS routines for all precisions (S, D, C, Z)
- Accelerates all kinds of applications:
  - Fluid dynamics, quantum chemistry, linear algebra, etc.
  - Today's focus: deep learning

Introducing CLBlast

- CLBlast: Modern C++11 OpenCL BLAS library
- Implements all BLAS routines for all precisions (S, D, C, Z)
- Accelerates all kinds of applications:
  - Fluid dynamics, quantum chemistry, linear algebra, etc.
  - Today's focus: deep learning
- Already integrated into various projects:
  - JOCLBlast (Java bindings)
  - ArrayFire (GPU accelerated library and applications)
  - OpenCL fork of Caffe (github.com/dividiti/ck-caffe)
  - OpenCL fork of TF (github.com/hughperkins/tensorflow-cl)

Introducing CLBlast



But... is it fast?

- All kernels are generic and tunable thanks to integration of the CLTune auto-tuner (presented at last year's GTC)

But… is it fast?

- All kernels are generic and tunable thanks to integration of the CLTune auto-tuner (presented at last year's GTC)

```
#define WGS 64 // The local work-group size
#define WPT 1  // The amount of work-per-thread
#define VW 1   // Vector width of vectors X and Y

typedef float dtype; // Example data-type
#if VW == 1
  typedef float dtypeV;
#elif VW == 2
  typedef float2 dtypeV;
#endif
```

```
__kernel __attribute__((reqd_work_group_size(WGS)))
void Xaxpy(const int n, const dtype alpha,
           const __global dtypeV* restrict xgm,
           __global dtypeV* ygm) {
  #pragma unroll
  for (int w=0; w<WPT; ++w) {
    int i = w*get_global_size(0)+get_global_id(0);
    ygm[i] = ygm[i] + alpha * xgm[i];
  }
}
```

But… is it fast?

- All kernels are generic and tunable thanks to integration of the CLTune auto-tuner (presented at last year's GTC)

```
#define WGS 64 // The local work-group size
#define WPT 1  // The amount of work-per-thread
#define VW 1   // Vector width of vectors X and Y

typedef float dtype; // Example data-type
#if VW == 1
  typedef float dtypeV;
#elif VW == 2
  typedef float2 dtypeV;
#endif
```

```
__kernel __attribute__((reqd_work_group_size(WGS)))
void Xaxpy(const int n, const dtype alpha,
           const __global dtypeV* restrict xgm,
           __global dtypeV* ygm) {
  #pragma unroll
  for (int w=0; w<WPT; ++w) {
    int i = w*get_global_size(0)+get_global_id(0);
    ygm[i] = ygm[i] + alpha * xgm[i];
  }
}
```

- Tuned out-of-the-box for 40 common devices
  - For new devices: run the auto-tuner when installing CLBlast

CLBlast Benchmark Results

AXPY
regular
(in GB/s)

AXPY
odd
(in GB/s)

GEMV
regular
(in GB/s)

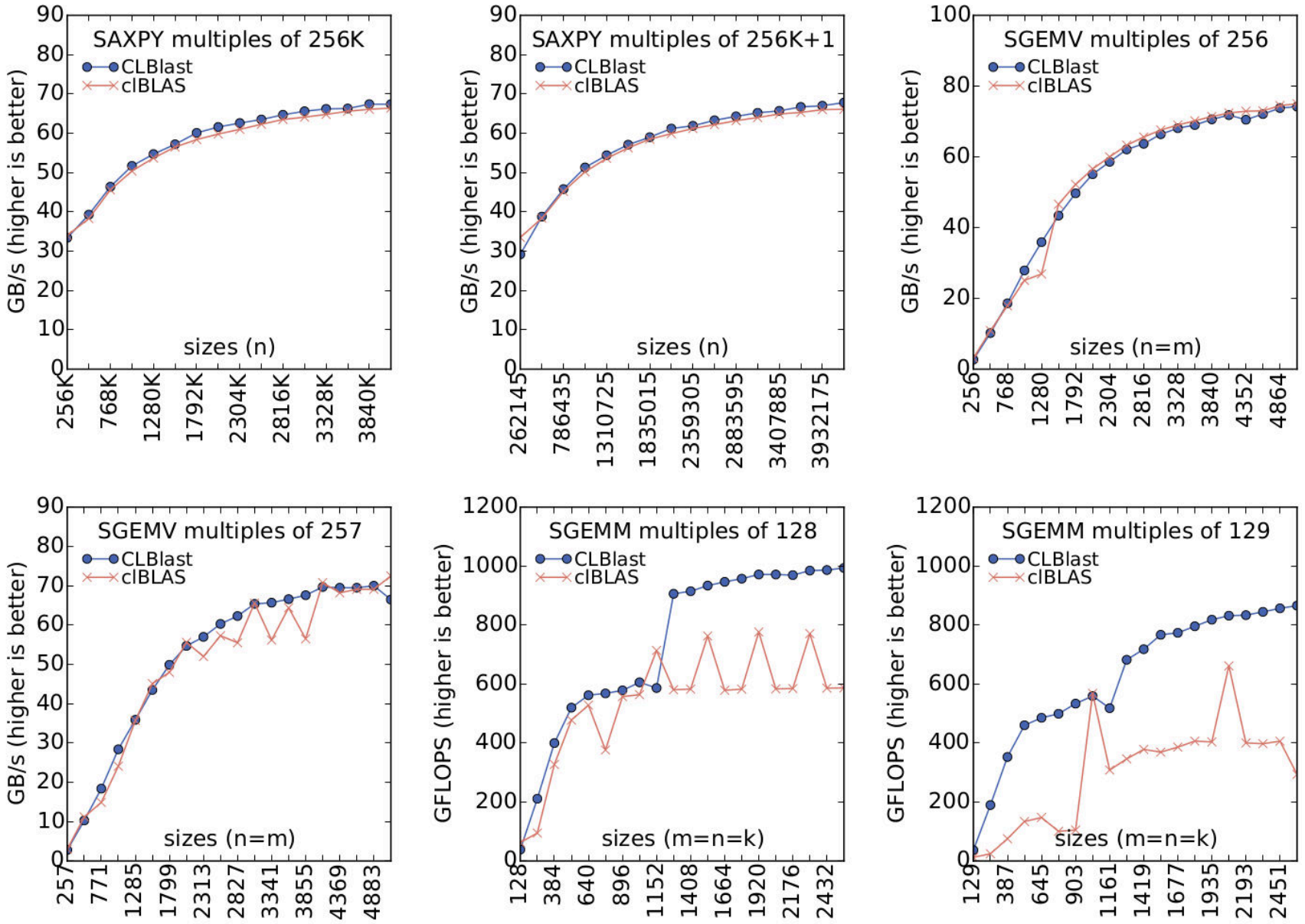
GEMV
odd
(in GB/s)

GEMM
regular
(in GFLOPS)

GEMM
odd
(in GFLOPS)

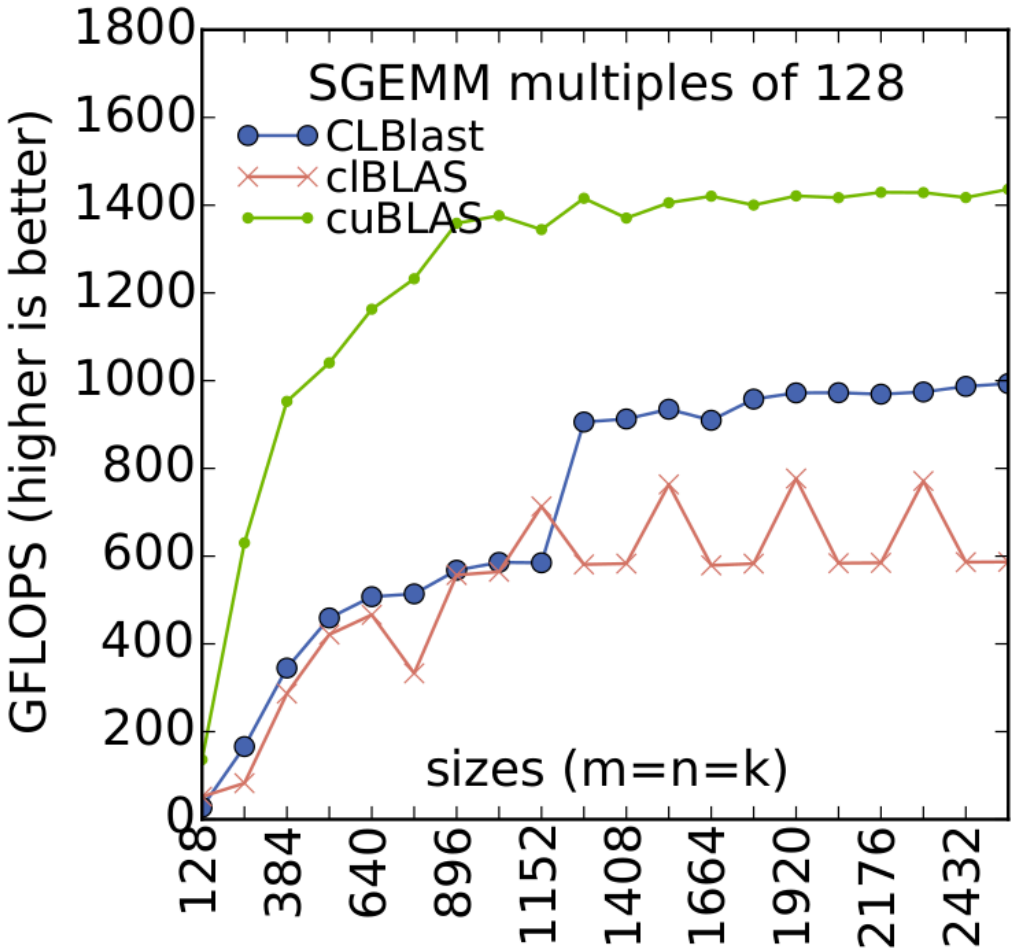
- Higher is better
- More results at http://cnugteren.github.io/clblast

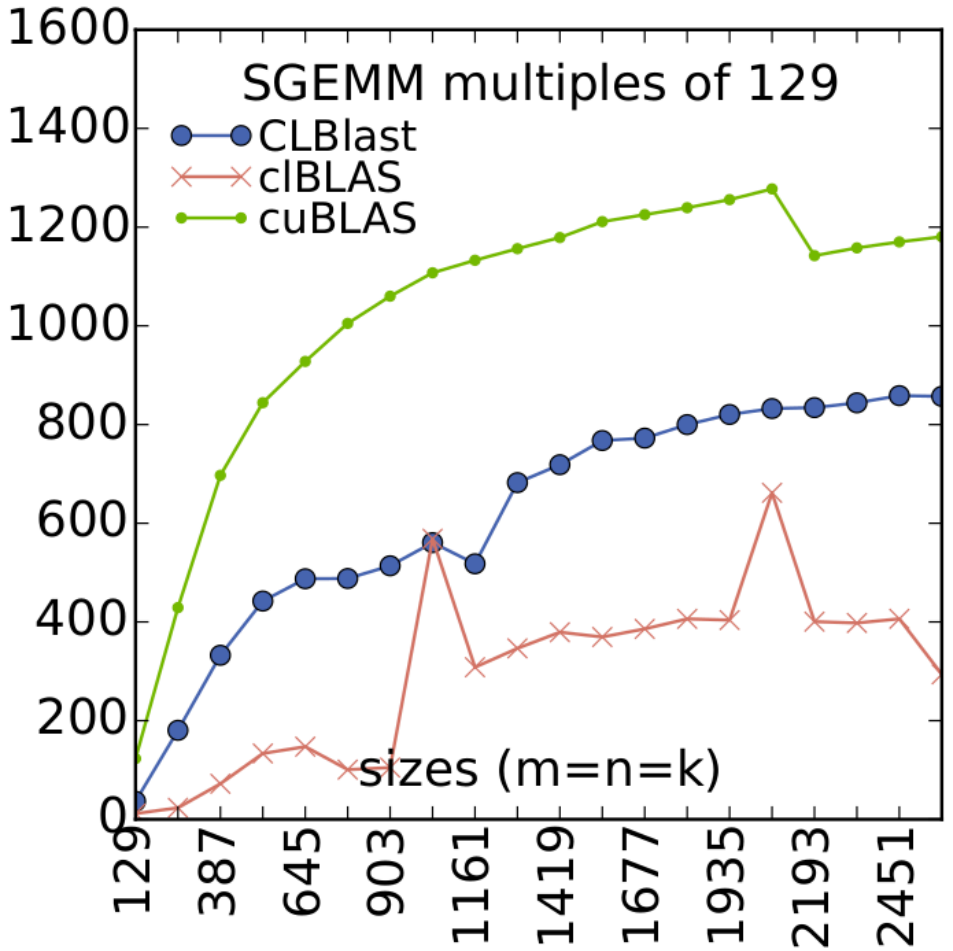
CLBlast on GeForce GTX750Ti



- On-par or better than clBLAS (especially for GEMM)

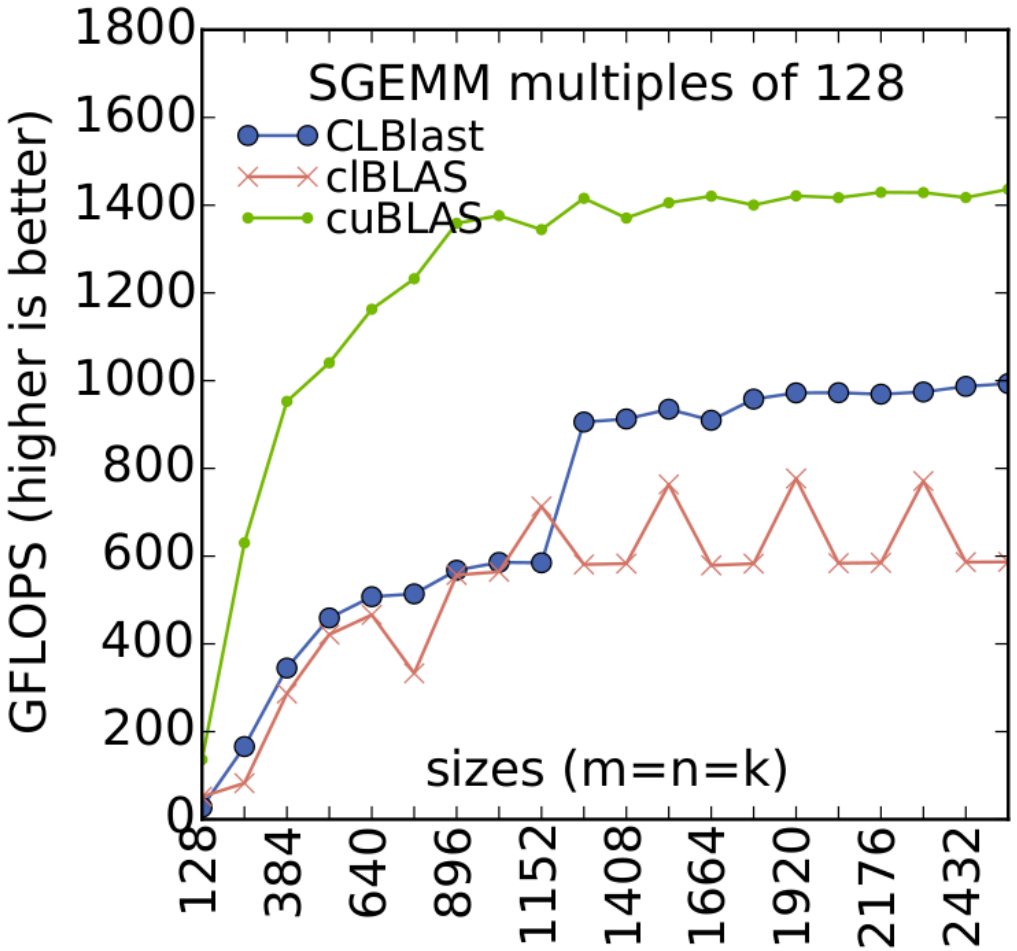
CLBlast on GeForce GTX750Ti

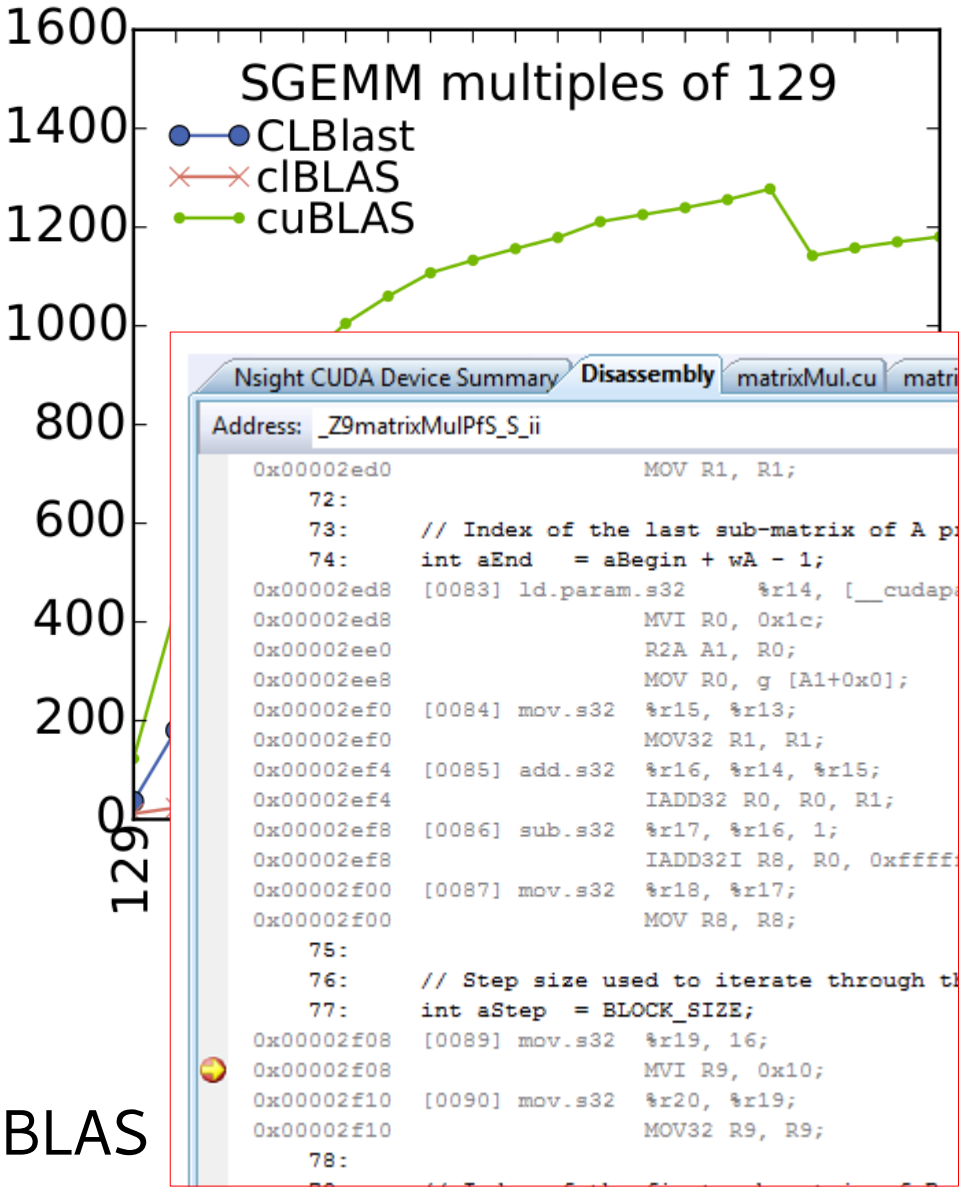




- ...but not as fast as NVIDIA's cuBLAS

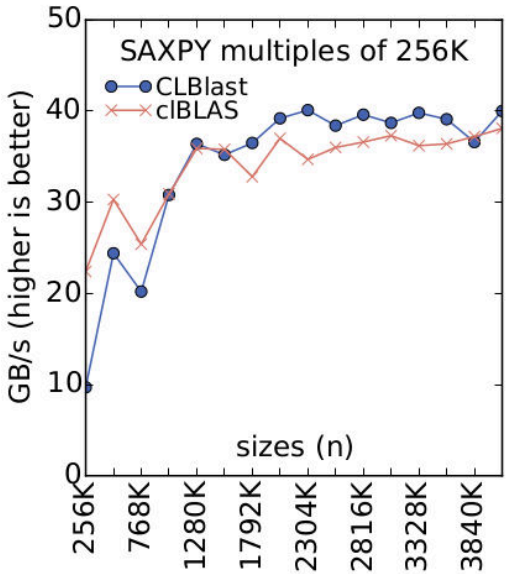
CLBlast on GeForce GTX750Ti

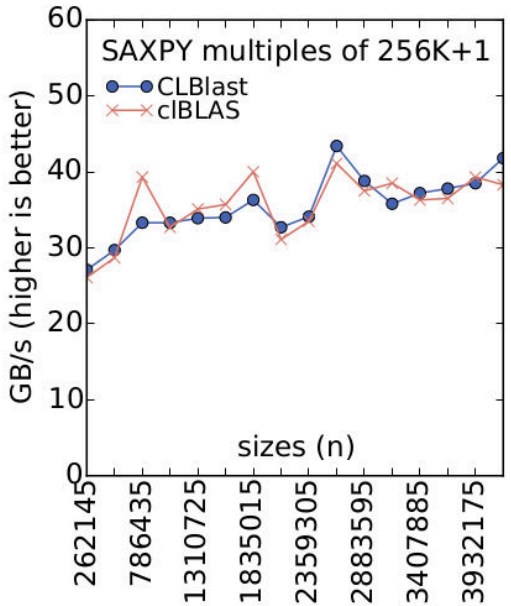


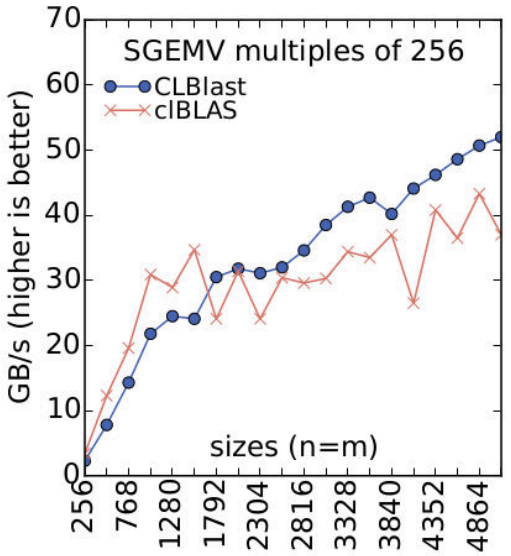


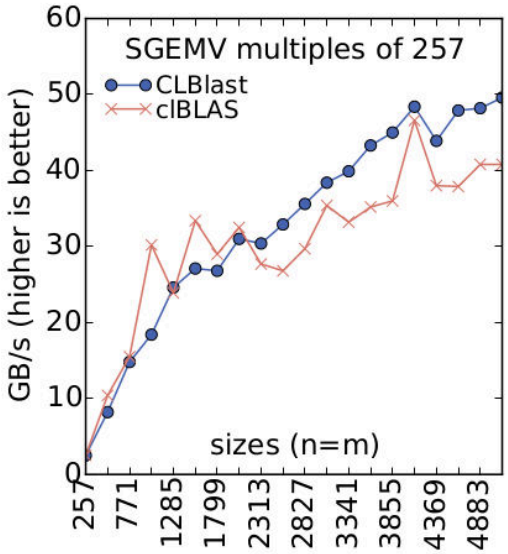
- ...but not as fast as NVIDIA's cuBLAS

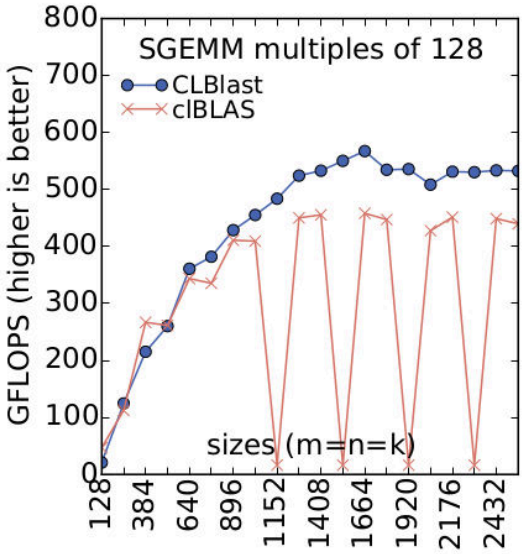
CLBlast on Radeon M370X

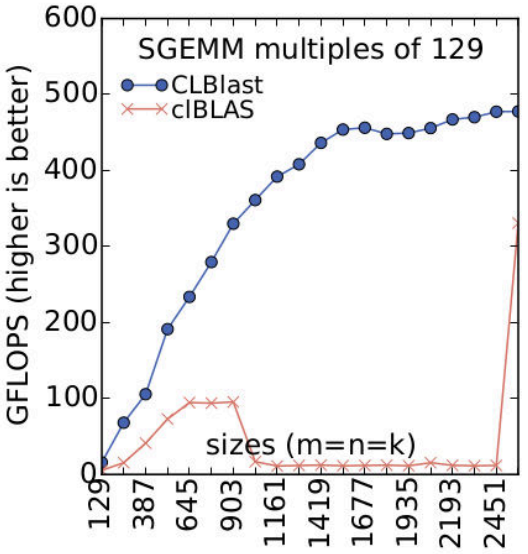






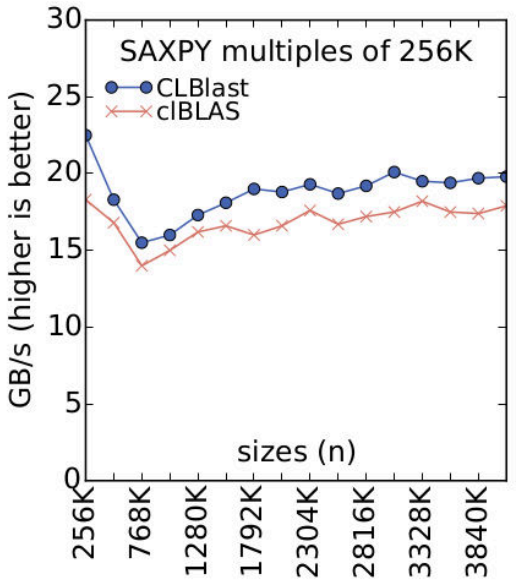


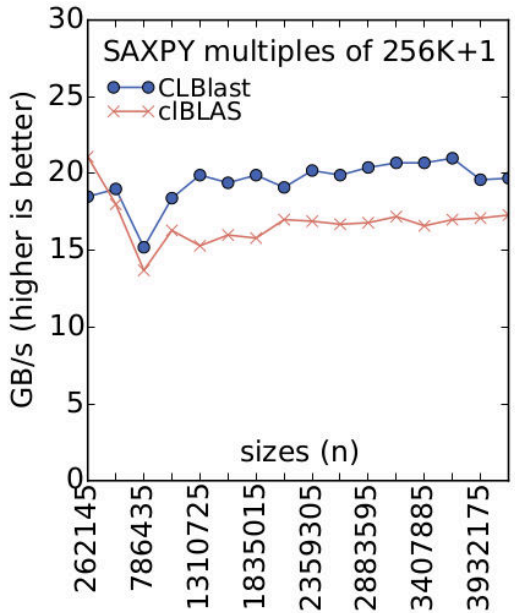


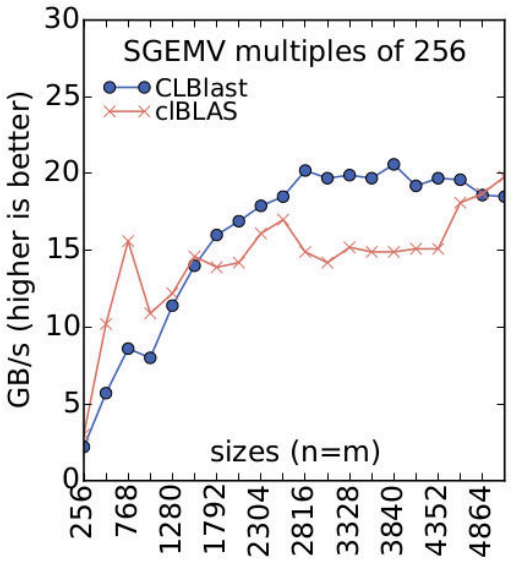


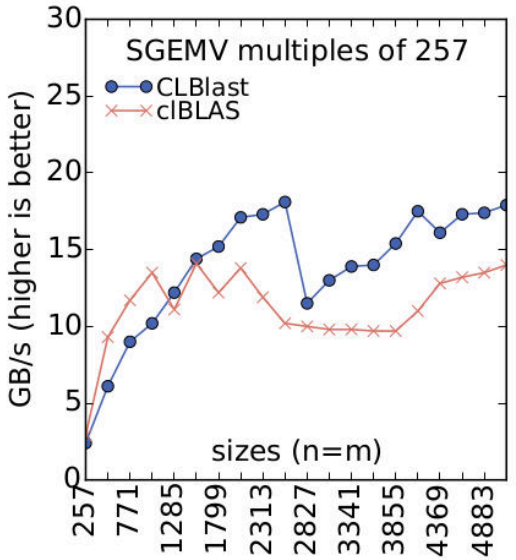
- On-par or better than clBLAS (especially for odd-sized GEMM)

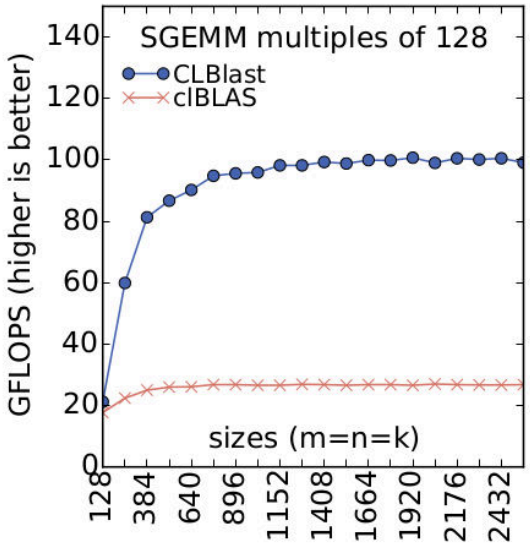
CLBlast on Skylake ULT GT2

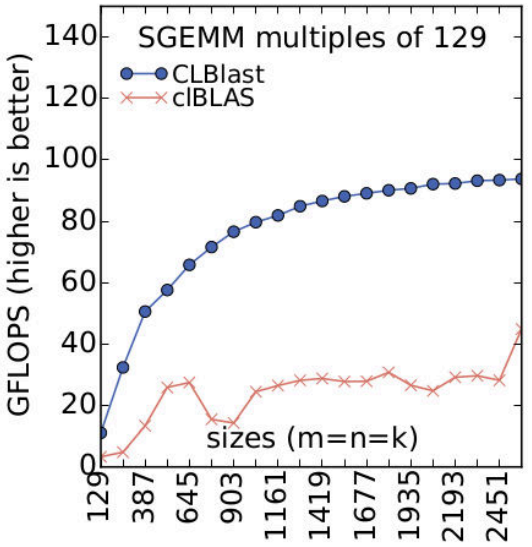






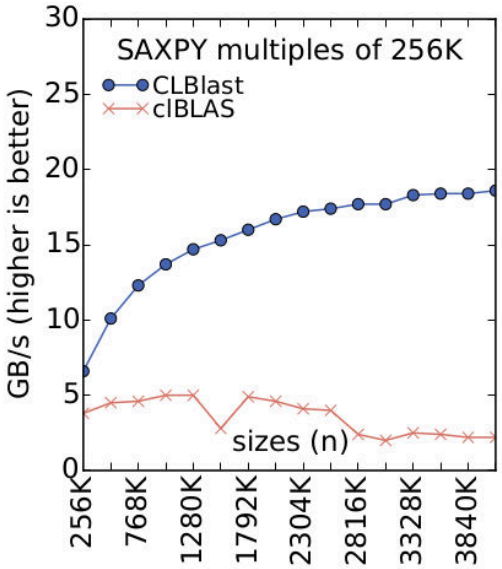


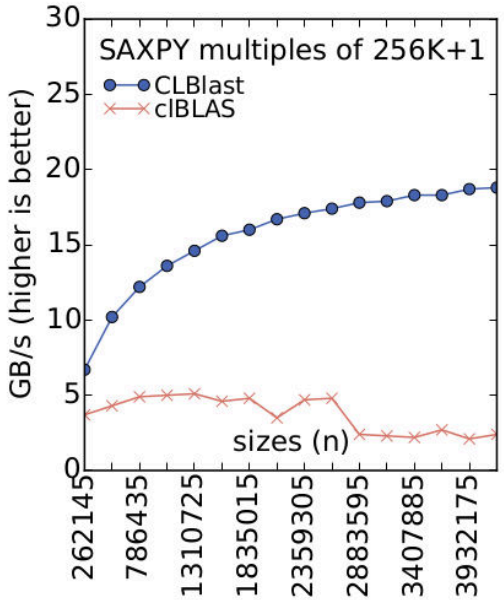


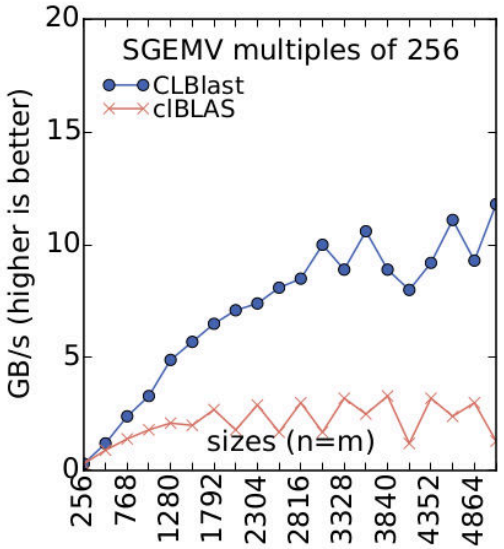


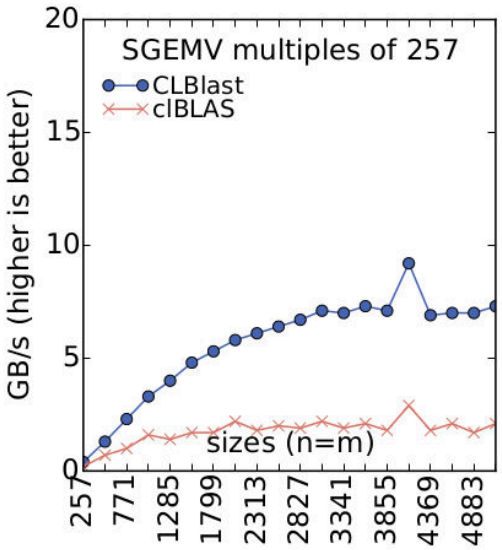
- On-par or better than clBLAS (especially for GEMM)

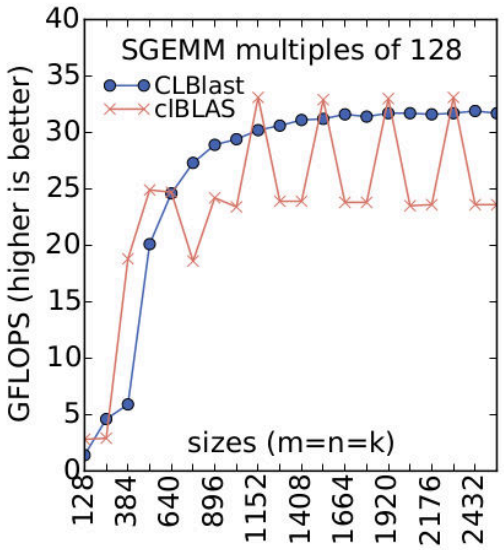
CLBlast on Core i5-6200U

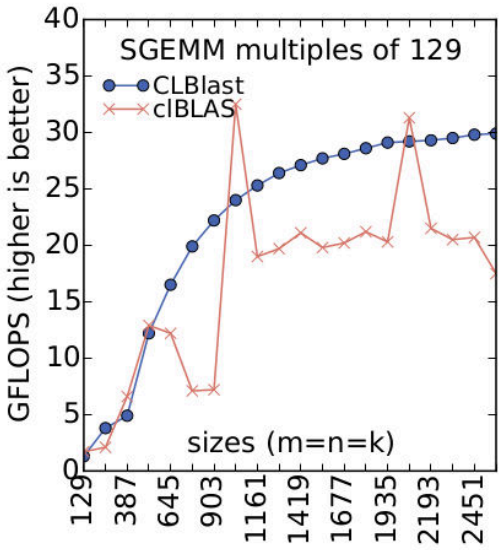












- On-par or better than clBLAS (especially for AXPY & GEMV)

CLBlast for Deep Learning

- What can we do for the deep-learning community?
  - Problem-specific tuning
  - Half-precision floating-point (FP16)
  - Batched routines

Tuning Only for a Single Size?

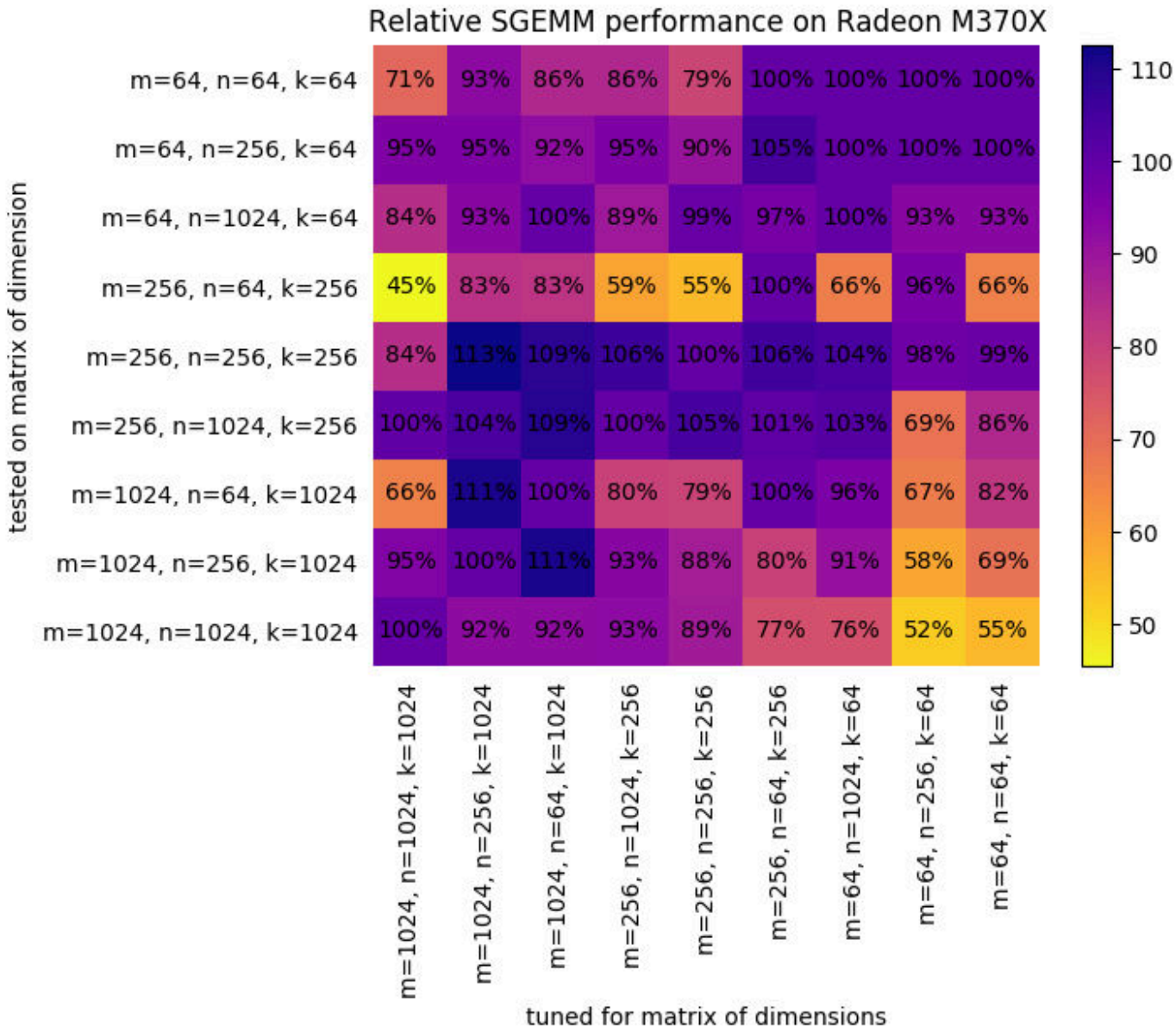
- Default GEMM tuning:
  - 1024x1024 matrices
- Deep-learning:
  - Various but fixed matrix sizes (dependent on network layout)
  - Typically smaller and/or rectangular matrices

Tuning Only for a Single Size?

- Default GEMM tuning:
  - 1024x1024 matrices
- Deep-learning:
  - Various but fixed matrix sizes (dependent on network layout)
  - Typically smaller and/or rectangular matrices
- Potential for optimal performance in CLBlast:
  - Tuning for a custom size possible
  - C++ API to change parameters at run-time

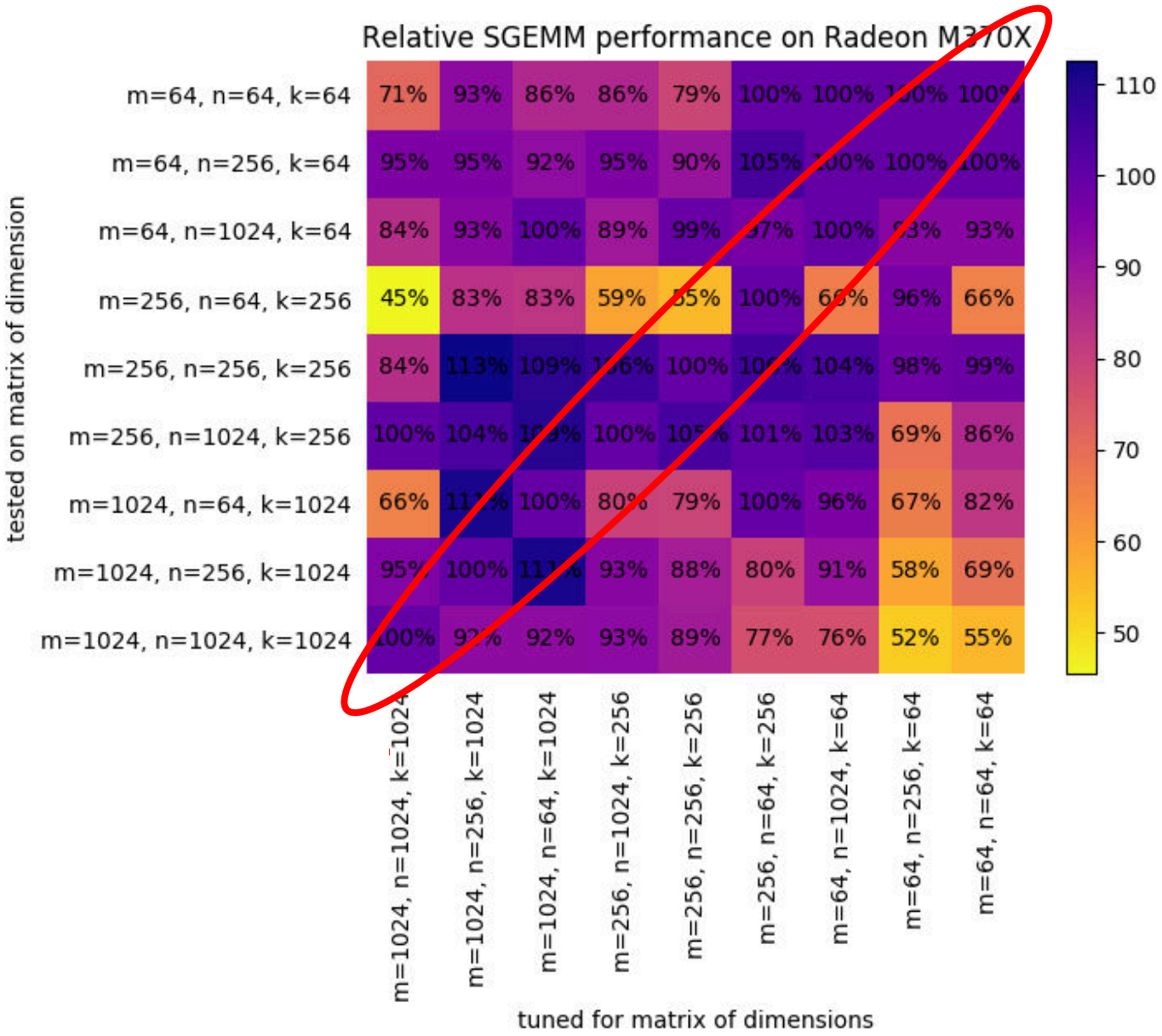
Problem-Specific Tuning

- SGEMM tuning for Radeon M370X GPU



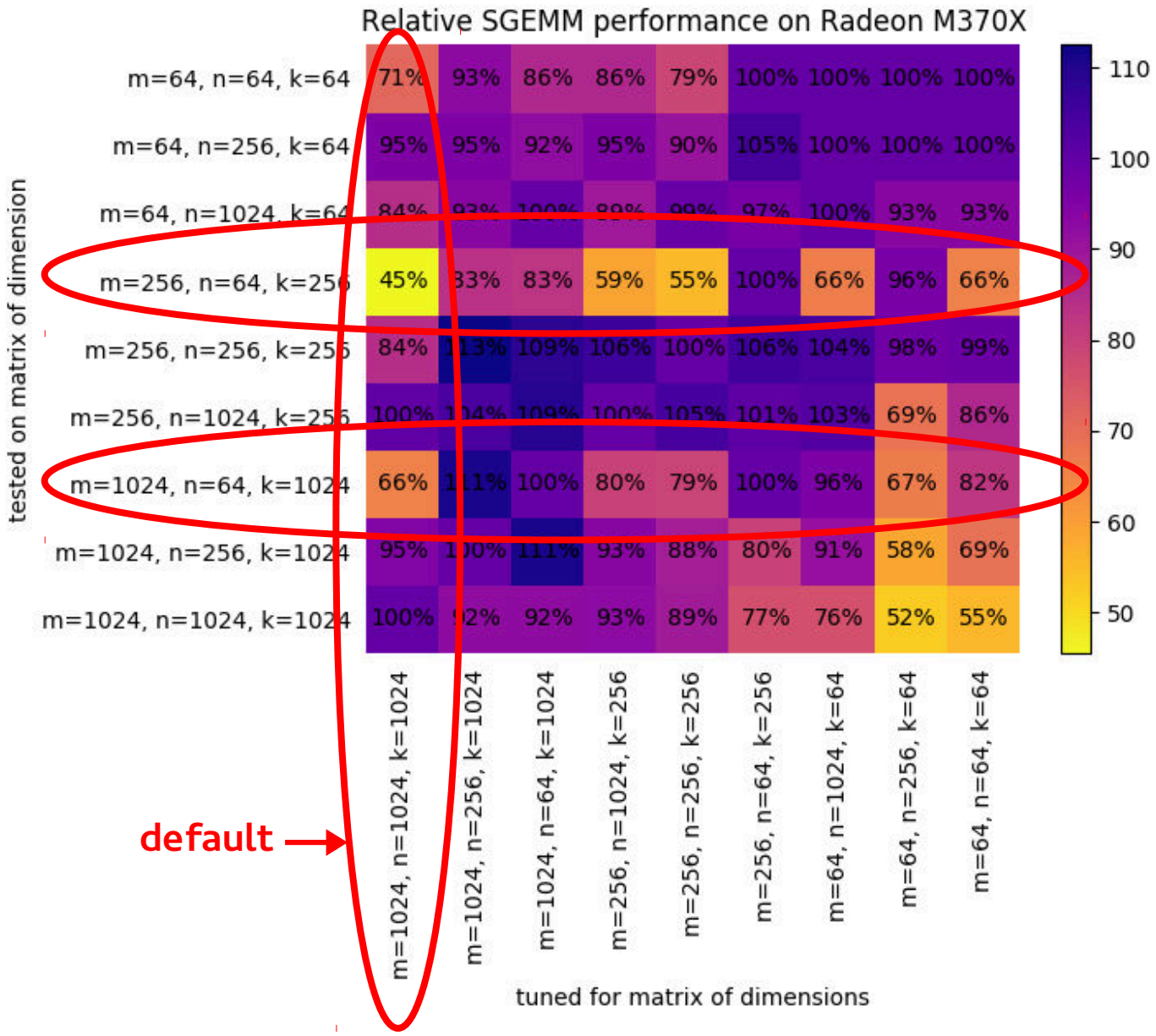
Problem-Specific Tuning

- SGEMM tuning for Radeon M370X GPU
- Best on the diagonal
- >100% due to random tuning



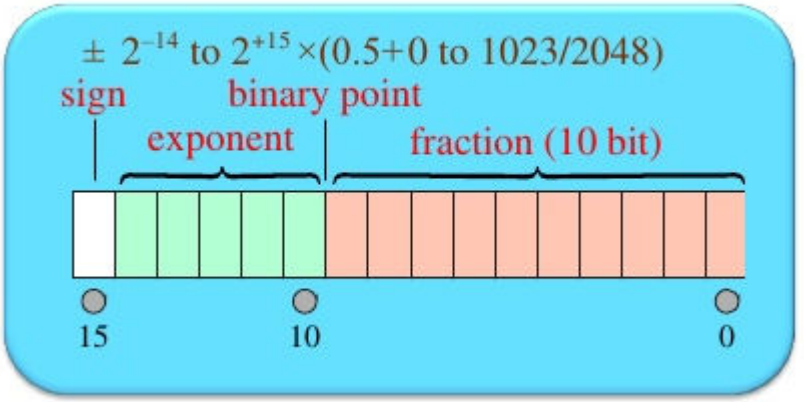
Problem-Specific Tuning

- SGEMM tuning for Radeon M370X GPU
- Best on the diagonal
- >100% due to random tuning
- Gain of ~2x for some cases



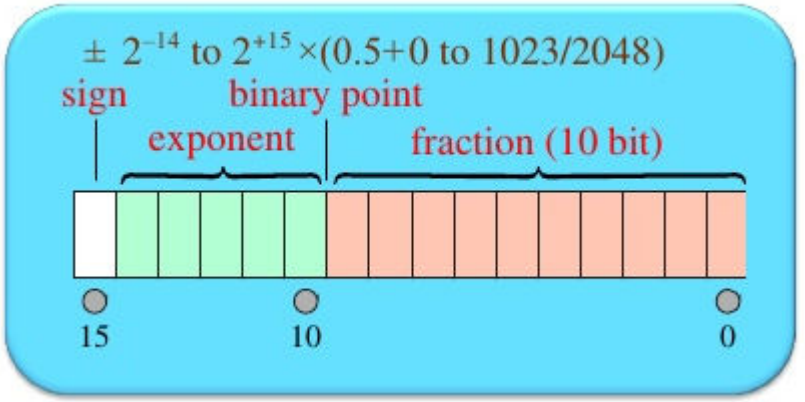
Half-precision floating-point (FP16)

- Double-precision (FP64) not needed for deep-learning
- Even FP32 is too much → introducing half-precision FP16
- Implemented in low-power devices (ARM Mali, Intel GPUs) and deep-learning specific GPUs (P100)



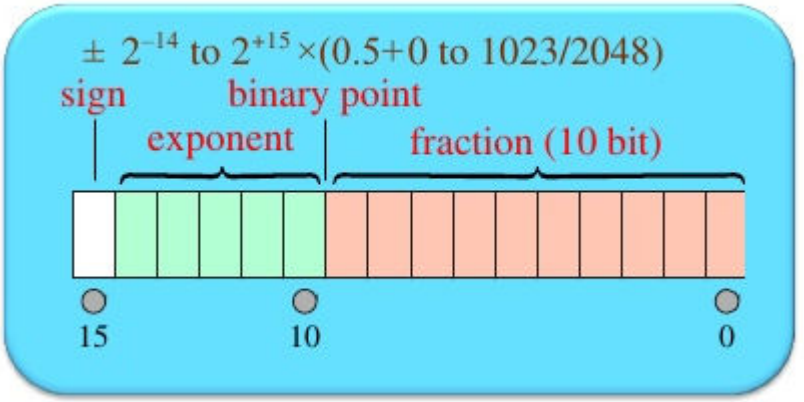
Half-precision floating-point (FP16)

- Double-precision (FP64) not needed for deep-learning
- Even FP32 is too much → introducing half-precision FP16
- Implemented in low-power devices (ARM Mali, Intel GPUs) and deep-learning specific GPUs (P100)
- Potential for 2x savings in:
  bandwidth, storage, compute, energy

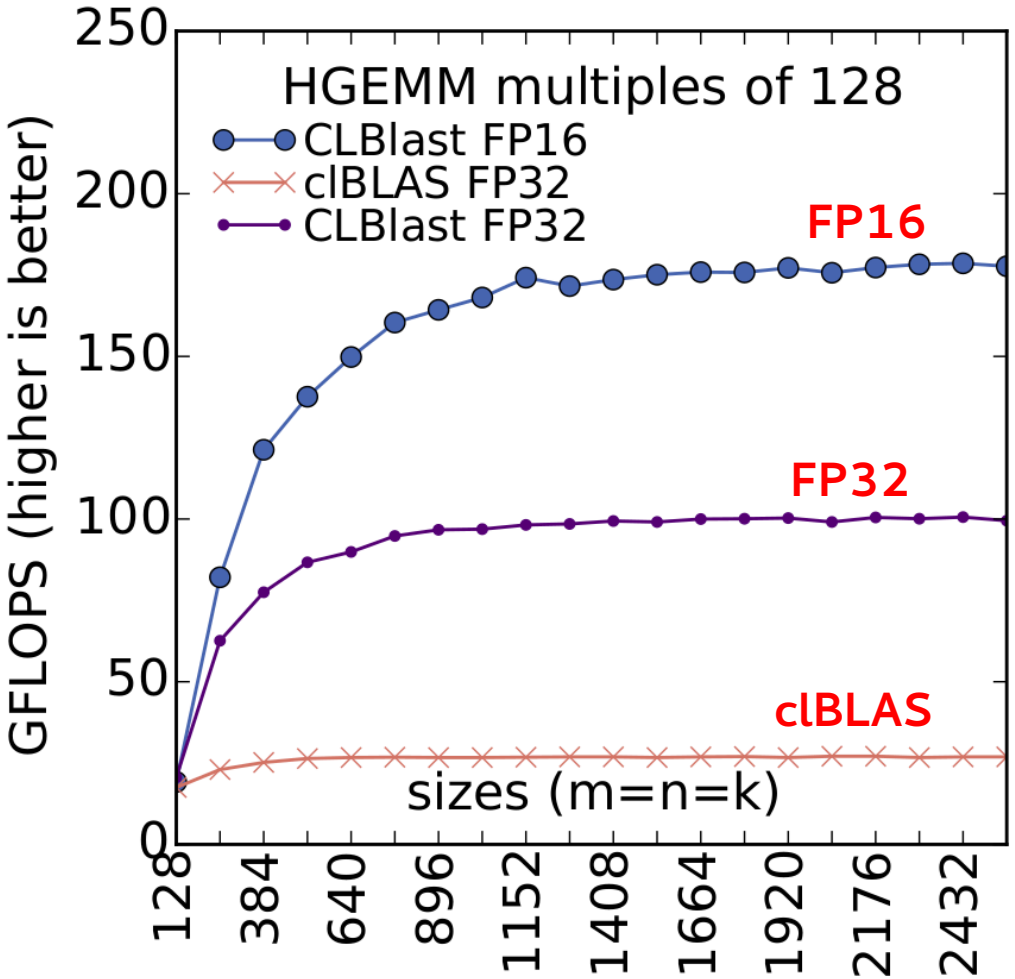


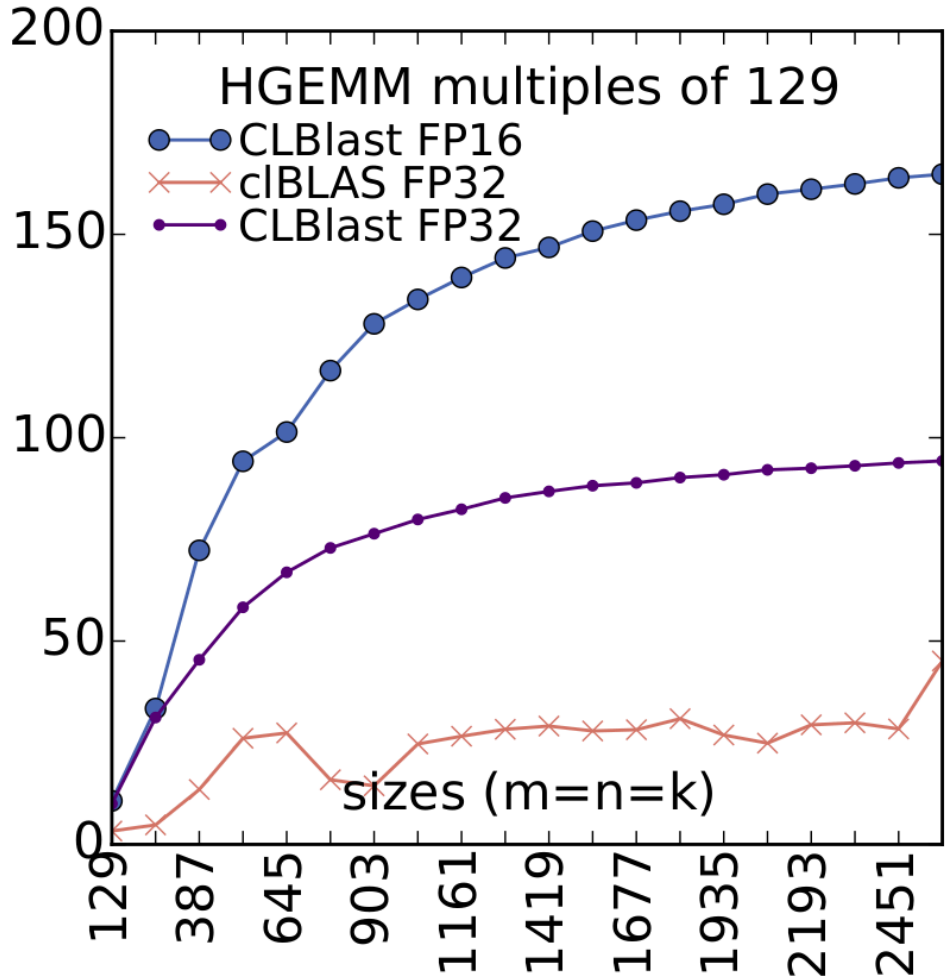
Half-precision floating-point (FP16)

- Double-precision (FP64) not needed for deep-learning
- Even FP32 is too much → introducing half-precision FP16
- Implemented in low-power devices (ARM Mali, Intel GPUs) and deep-learning specific GPUs (P100)
- Potential for 2x savings in:
  bandwidth, storage, compute, energy
- Current FP16 support for GPUs:
  - cuBLAS: HGEMM only
  - clBLAS: no FP16 at all
  - CLBlast: all routines!



Half-precision FP16 on Intel Skylake GPU

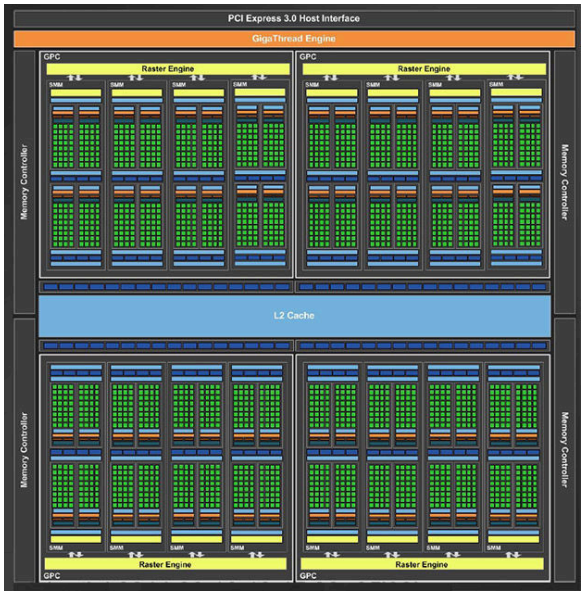




- FP16 ~1.8x faster across the board!

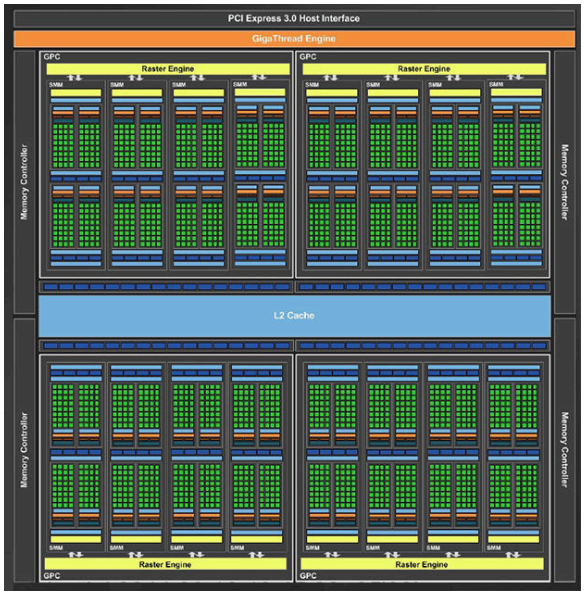
Batching BLAS routines

- Small-sized GEMM is super slow
  - Not enough work-groups
  - Not enough threads



Batching BLAS routines

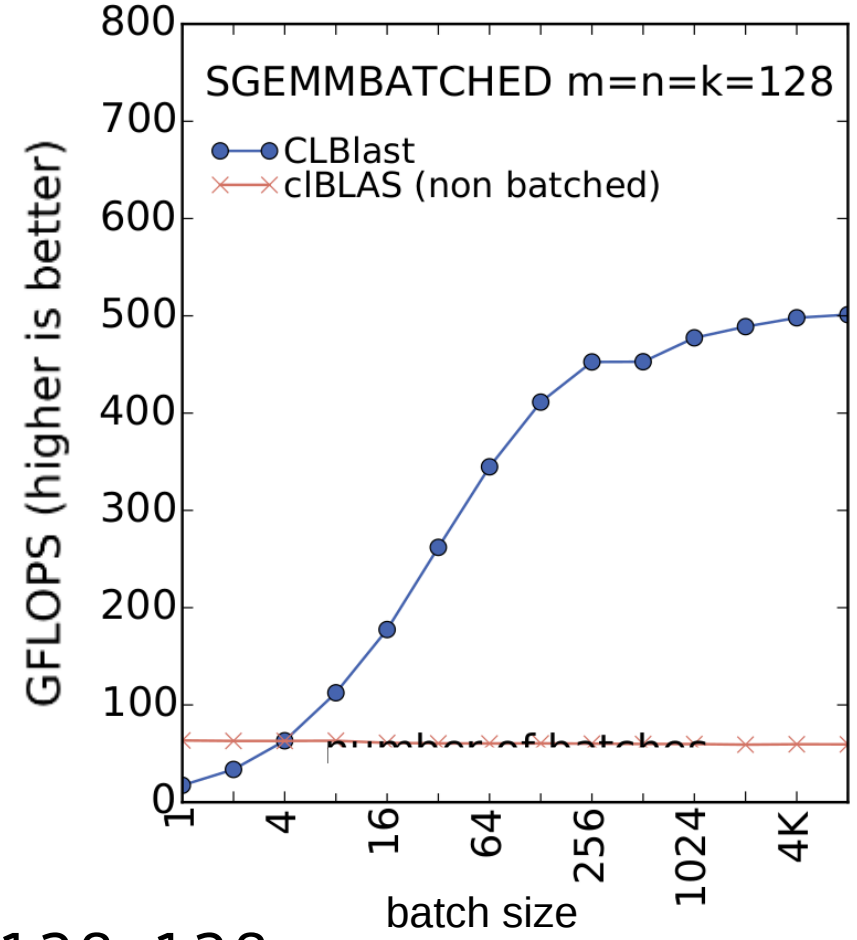
- Small-sized GEMM is super slow
  - Not enough work-groups
  - Not enough threads



- Let's make it fast again:

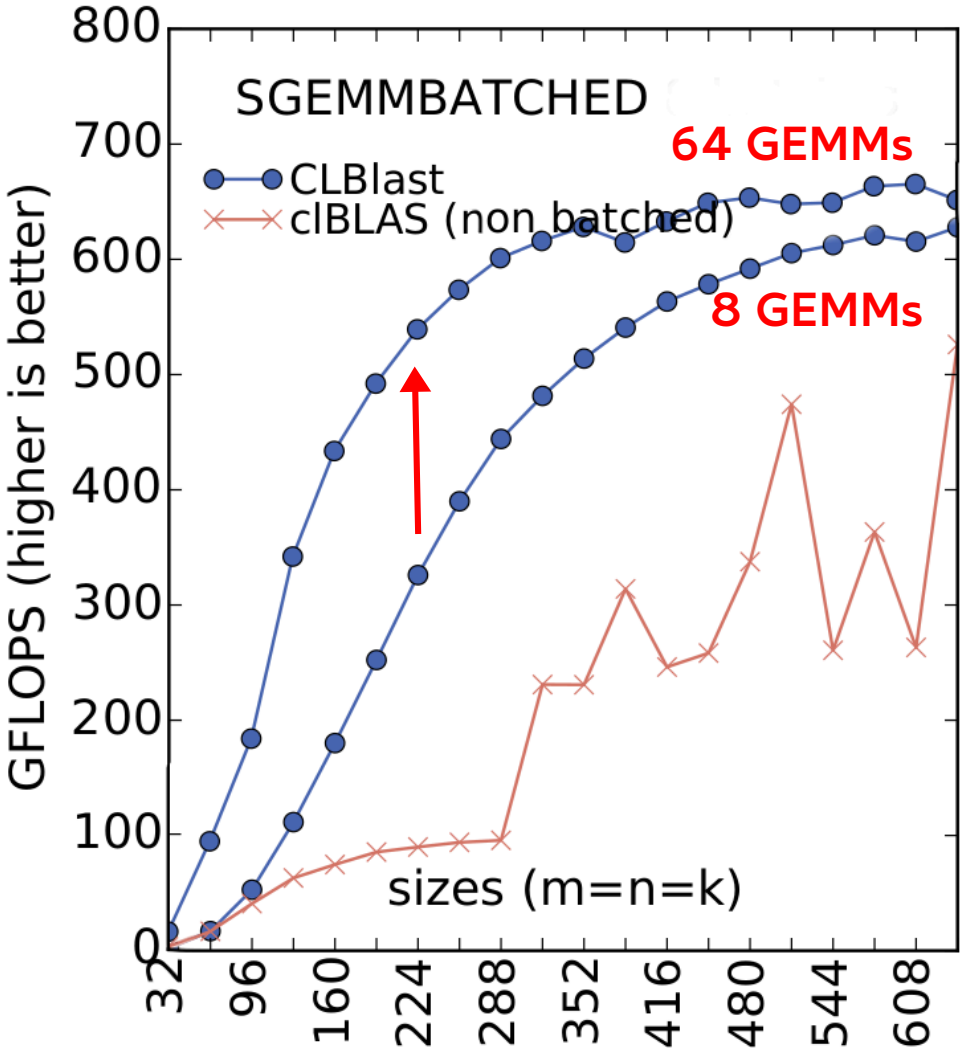
  - Combine multiple small GEMM operations into a single kernel
  - Use offsets to indicate where the next matrices start

Batched GEMM on GeForce GTX 750Ti



- SGEMM 128x128x128:
  - Regular: ~40 GFLOPS
  - Batched: ~10 GFLOPS (1 GEMM) up to ~500 GFLOPS (8K)!

Batched GEMM on GeForce GTX 750Ti



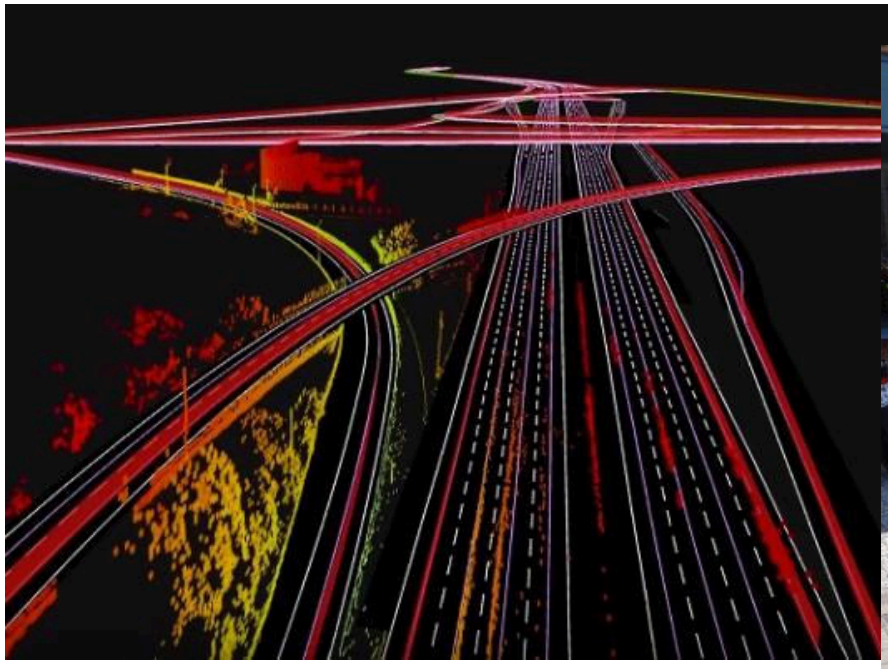
- Significant benefits for larger sizes as well
  - mostly beneficial in the range n=64 till 512

What's next?

- More features for deep learning:
  - 'im2col'
  - Winograd? FFT?
- Input-based auto-tuning using learned models
  - Similar to S7150: The ISAAC library
- Integration into OpenCL deep-learning projects
  - TensorFlow SYCL? LibDNN?
- Suggestions?

Why is BLAS Important for TomTom ?

- HDMap making → Deep-learning
- Deep-learning → Fast BLAS libraries



- More info: S7809 - A Multi-Source, Multi-Sensor Approach to HDMap Creation
  - Willem Strijbosch - Head of Autonomous Driving, TomTom
  - Today at 10:30 AM in room 210D

Conclusion

- Introducing CLBlast: a modern C++11 OpenCL BLAS library
- Performance portable thanks to generic kernels and auto-tuning
- Especially targeted at accelerating deep-learning:
  - Problem-size specific tuning:
    - Up to 2x in an example experiment
  - Half-precision FP16 support:
    - Up to 2x benefit in speed and memory savings
  - Batched GEMM routine:
    - Order of magnitude benefit depending on the use-case

TOMTOM

CLBlast: A Tuned BLAS Library
for Faster Deep Learning

Cedric Nugteren

May 11, 2017

http://github.com/cnugteren/clblast

http://cnugteren.github.io/clblast