



# CLBlast: A Tuned BLAS Library for Faster Deep Learning

Cedric Nugteren

May 11, 2017



<http://github.com/cnugteren/clblast>

<http://cnugteren.github.io/clblast>

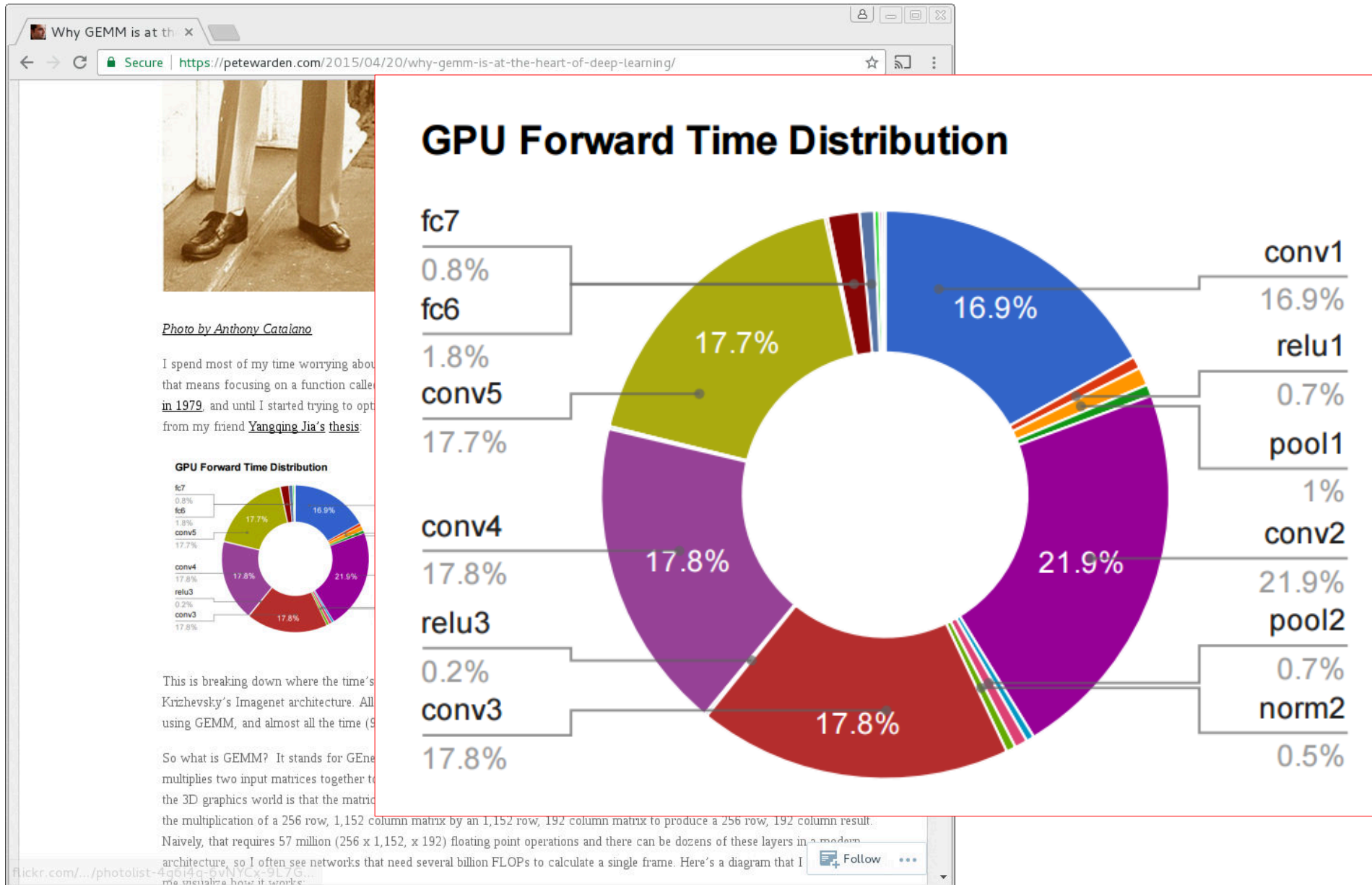
# The Heart of Deep Learning



# GEMM is at the Heart of Deep Learning

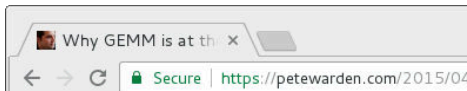


# So where are the Matrix-Multiplications?





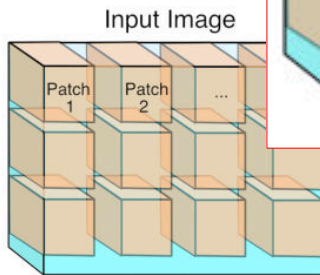
# Convolutions as Matrix Multiplication



## How GEMM works for

This seems like quite a specialized operation but it's not clear how or why we should do it here's how the operation is expressed in

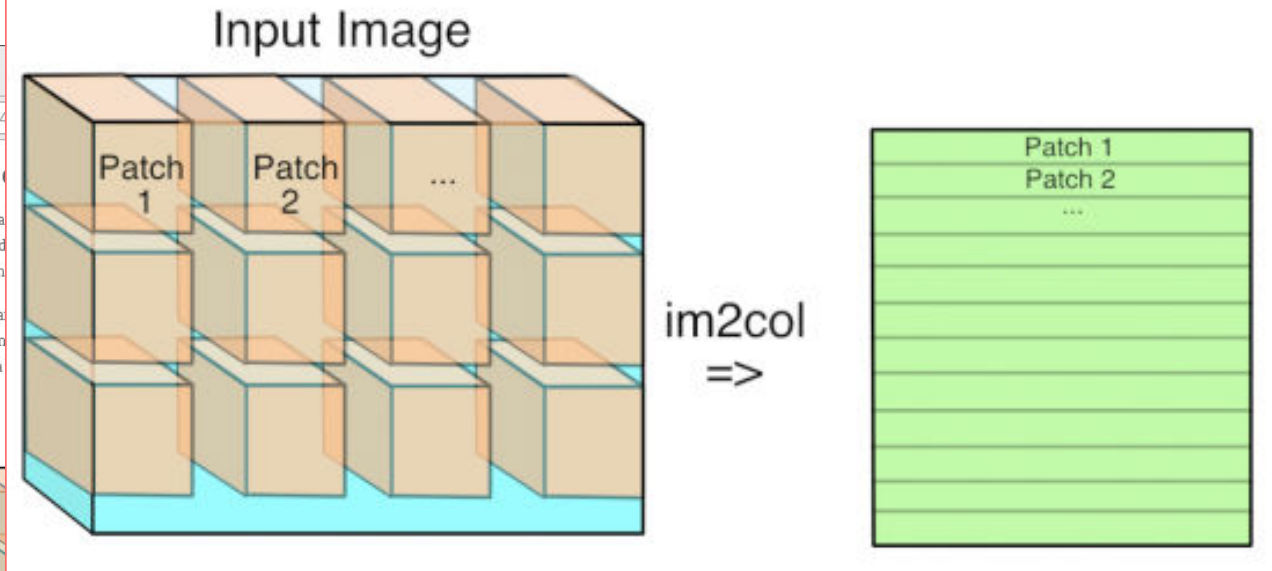
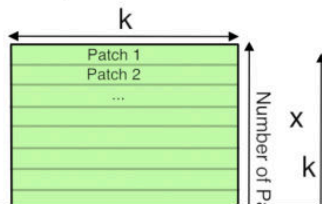
The first step is to turn the input from a 3D volume into a 2D matrix. Each kernel is applied to a little three-dimensional patch and the results are copied out as a single column into a matrix. Here's how I visualize it.



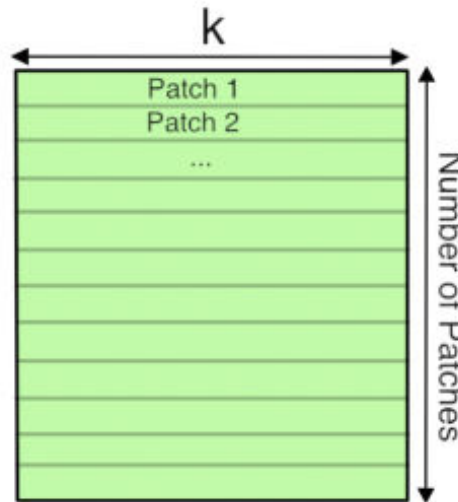
Now if you're an image-processing geek like me, you might think this conversion is inefficient if the stride is less than the kernel size. This is because the patches are duplicated in the matrix, which seems inefficient. Yet

Now you have the input image in matrix form, you need a second matrix for the multiplication. Here's what that looks like.

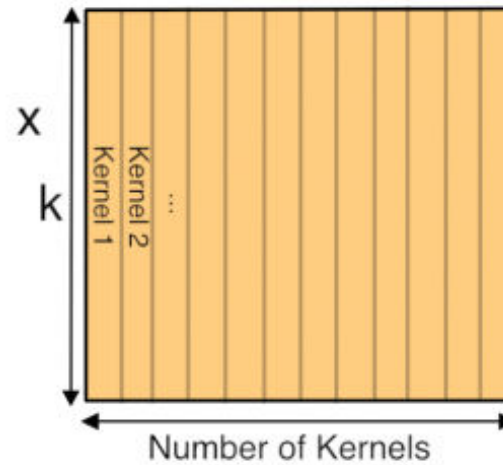
## Input Matrix



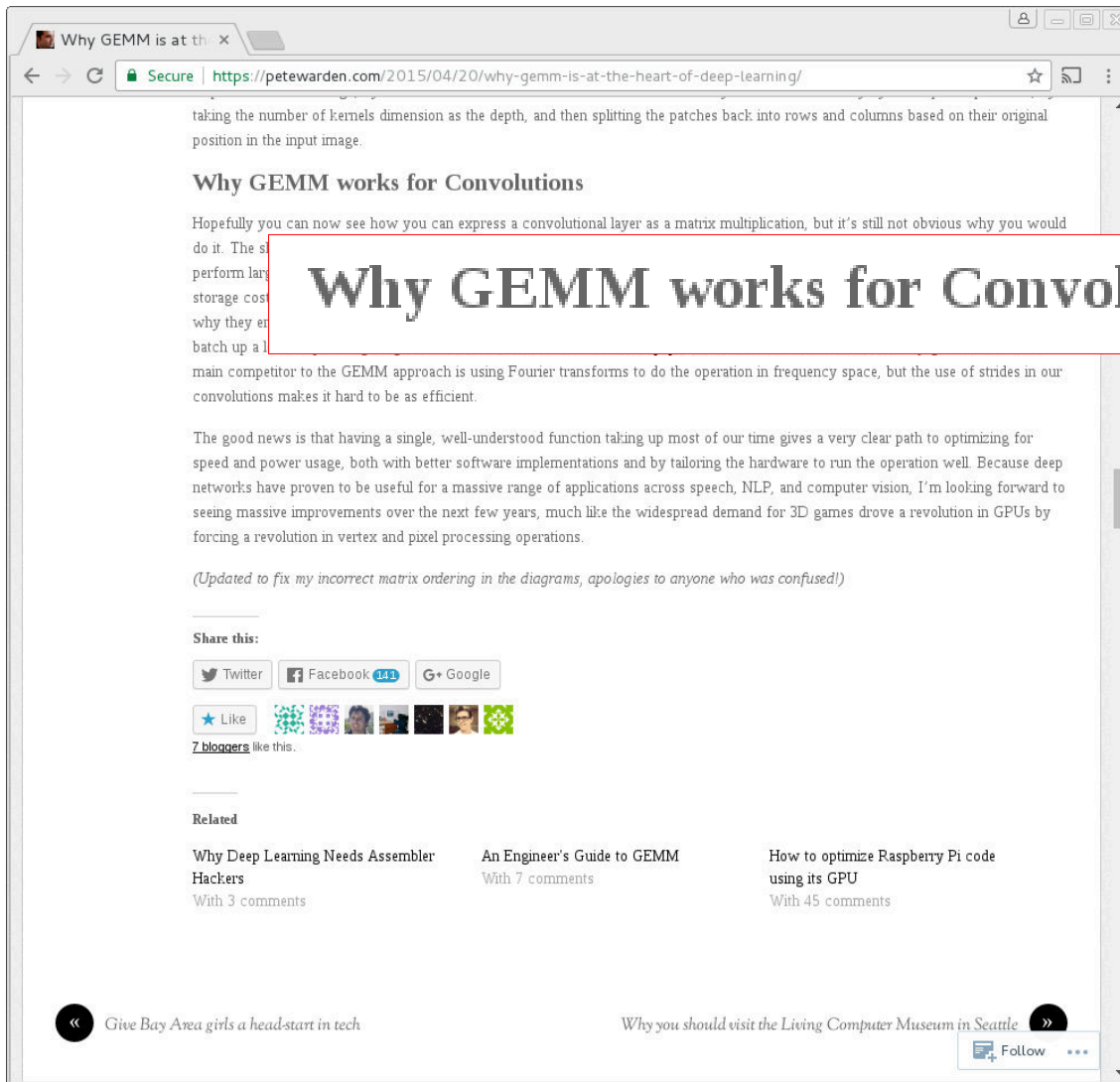
## Input Matrix



## Kernel Matrix



# GEMM is the Heart of Deep Learning



taking the number of kernels dimension as the depth, and then splitting the patches back into rows and columns based on their original position in the input image.

## Why GEMM works for Convolutions

Hopefully you can now see how you can express a convolutional layer as a matrix multiplication, but it's still not obvious why you would do it. The speed performance largely depends on the hardware, but there are some storage costs involved, and that's why they end up batch up a...

### Why GEMM works for Convolutions

main competitor to the GEMM approach is using Fourier transforms to do the operation in frequency space, but the use of strides in our convolutions makes it hard to be as efficient.

The good news is that having a single, well-understood function taking up most of our time gives a very clear path to optimizing for speed and power usage, both with better software implementations and by tailoring the hardware to run the operation well. Because deep networks have proven to be useful for a massive range of applications across speech, NLP, and computer vision, I'm looking forward to seeing massive improvements over the next few years, much like the widespread demand for 3D games drove a revolution in GPUs by forcing a revolution in vertex and pixel processing operations.

*(Updated to fix my incorrect matrix ordering in the diagrams, apologies to anyone who was confused!)*

**Share this:**

Twitter Facebook 141 G+ Google

★ Like

7 bloggers like this.

**Related**

<a href="#">Why Deep Learning Needs Assembler Hackers</a> With 3 comments	<a href="#">An Engineer's Guide to GEMM</a> With 7 comments	<a href="#">How to optimize Raspberry Pi code using its GPU</a> With 45 comments
--	--	---

« Give Bay Area girls a head-start in tech

Why you should visit the Living Computer Museum in Seattle »

Follow

# Does everyone agree?

Why GEMM is at the heart of deep learning

20 responses

**ZYGMUNT** says:  
April 20, 2015 at 12:26 pm  
Correct me if I'm wrong  
 $k \times m * n \times k \neq n \times m$   
You need  
 $n \times k * k \times m = n \times m$

**CPUGUY** says:  
April 21, 2015 at 10:53 am  
Nice paper highlighting the importance of high performance linear algebra

**KART**  
January 5, 2017 at 1:44 am  
What you said is right. You need  $m \times k * k \times n$  to produce  $m \times n$ . The figure also shows the same. Matrix dimensions are always (no. of rows  $\times$  no. of columns). So, as per the first diagram, the matrix dimensions are  $m \times k$  and  $k \times n$ . So, you have what you need to multiply.

**SCOTT GRAY** says:  
April 21, 2015 at 6:55 am  
REPLY  
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.  
So each small matrix multiply is just one position of the filter over the image. The outer dims of this MM are N and K and CRS is reduced. To load in the image data you need to slice the image as you are carrying out the reduction of outer products. This is most easily achieved if N is the contiguous dimension of your image data. This way a single pixel offset applies to a whole row of data and can be efficiently fetched all at once. The best way to calculate that offset is to first build a small lookup table of all the spatial offsets. The channel offset after each lookup. This keeps your lookup table small and easy to fit in fast shared memory.

# Does everyone agree?

The screenshot shows a web browser window with the address bar displaying `https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/`. Below the browser, a comment thread is visible. The top comment is from 'ZYGMUNT' dated April 20, 2015. Below it is a comment from 'CPUGUY' dated April 21, 2015, which is highlighted with a red box. The 'CPUGUY' comment includes a profile picture of Scott Gray, a 'REPLY' button, and several paragraphs of text. The text discusses the efficiency of small gemms on a GPU compared to a large gemm operation, providing links to GitHub benchmarks and an arXiv paper. The phrase 'it's actually much more efficient as many small gemms' is underlined in red in the original image.



# 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 data 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, ...

clMathLibraries / clBLAS

<> Code | Issues 63 | Pull requests 0 | Projects 0 | Wiki


Filters | is:issue is:open | Labels | Miles

63 Open ✓ 93 Closed Author ▾





- Not able to Build clBLAS from Sources  
#315 opened 28 days ago by saviogeorge
- make rebuilds every file even if no source files were changed  
#310 opened on 22 Mar by cirosantilli
- Leaked events in GEMM calls (and probably other functions)  
#304 opened on 22 Feb by Oblomov
- tests go segmentation fault  
#299 opened on 25 Jan by aram4github,
- Error compiling during run-time  
#293 opened on 5 Dec 2016 by am15600
- ixamax errors  
#292 opened on 2 Nov 2016 by mgates3
- test-short fails on Ubuntu with AMD Card



# 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](https://github.com/dividiti/ck-caffe))
  - OpenCL fork of TF ([github.com/hughperkins/tensorflow-cl](https://github.com/hughperkins/tensorflow-cl)) 

# Introducing CLBlast

The screenshot shows the GitHub repository for CLBlast. A red circle highlights the 'Star' button (131) and 'Fork' button (30). A red circle highlights the '722 commits' button, with the word 'activity' written in red next to it. A red circle highlights the '8 contributors' button, with the word 'community' written in red above it. A red circle highlights the CI configuration files: '.appveyor.yml', '.gitignore', and '.travis.yml', with the text 'CI and extensive testing' written in red next to them. A 'BLAST' comic-style graphic is overlaid on the repository header. The repository is in the 'development' branch, which is 143 commits ahead and 1 commit behind master. The latest commit is by CNugteren, dated 9 days ago. The commit history shows various updates to CMake, API, namespace, samples, scripts, and src files, as well as CI configuration updates.

CNugteren / CLBlast

Watch 14 Star 131 Fork 30

Code Issues 11

Tuned OpenCL BLAS

blas opencil blas-libraries cblas matrix-multiplication gemm gpu

722 commits 3 branches 11 releases 8 contributors Apache-2.0

Branch: development New pull request Find file Clone or download

This branch is 143 commits ahead, 1 commit behind master. Pull request Compare

CNugteren committed on GitHub Merge pull request #148 from CNugteren/benchmarking Latest commit c9f39ed 9 days ago

File	Commit Message	Time
cmake	Added proper CMake searching for CUDA and cuBLAS	29 days ago
doc	Added API and test infrastructure for the batched GEMM routine	2 months ago
include	Fixed a namespace clash with CUDA FP16 for the half-datatype	15 days ago
samples	treewide: silence type mismatch warnings in *printf()	3 months ago
scripts	Added an option to the database script to remove tuning results from ...	9 days ago
src	Re-added Titan X (Pascal) tuning results based on more averaging when...	9 days ago
test	Fixed a compiler warning message	9 days ago
.appveyor.yml	Updated AppVeyor script to fix an issue with changes in the latest Ap...	7 months ago
.gitignore	Complete re-write of the database script. Changed Pandas for the much...	8 months ago
.travis.yml	.travis.yml: do not build for osx twice, there's no gcc there	3 months ago

## 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)

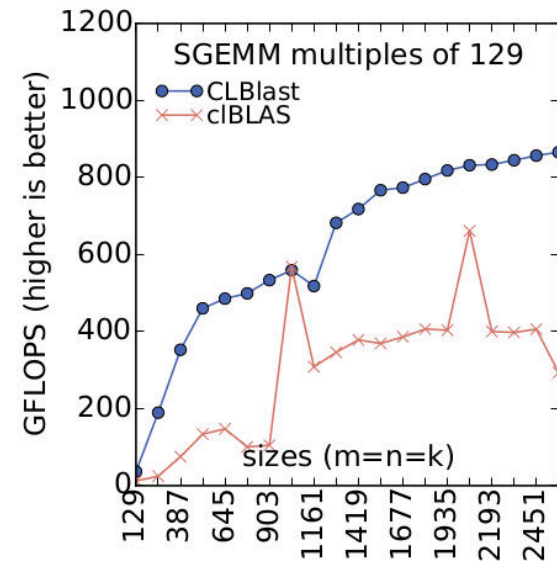
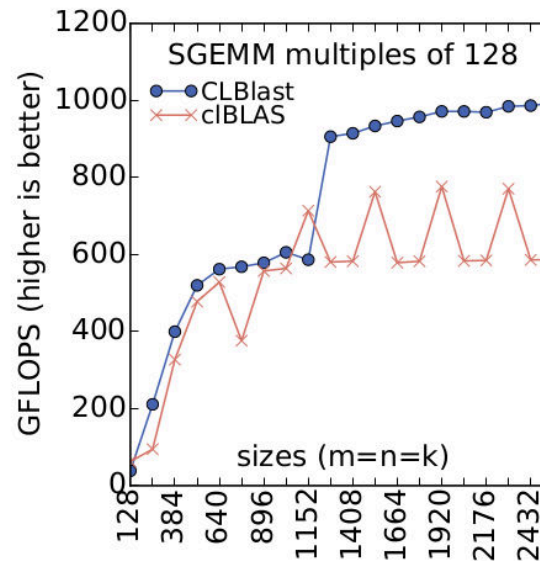
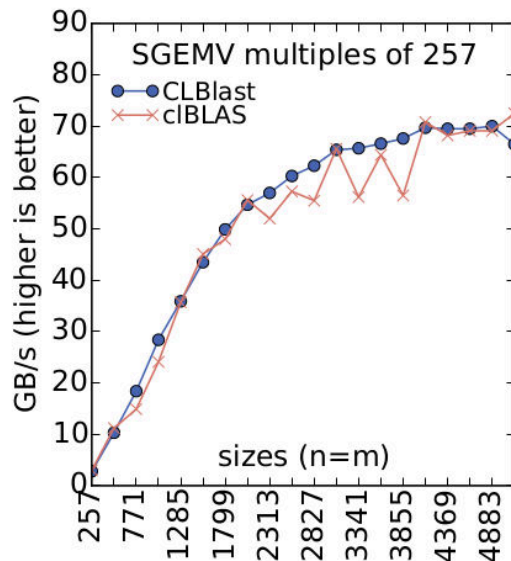
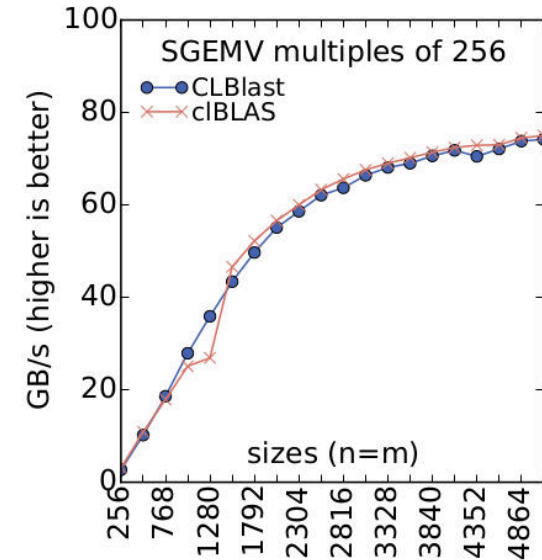
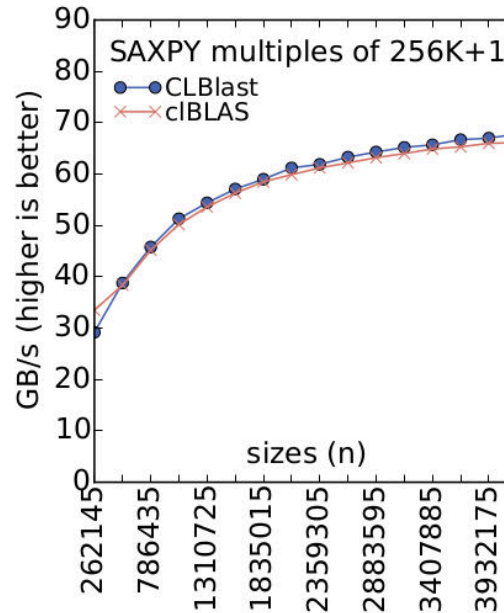
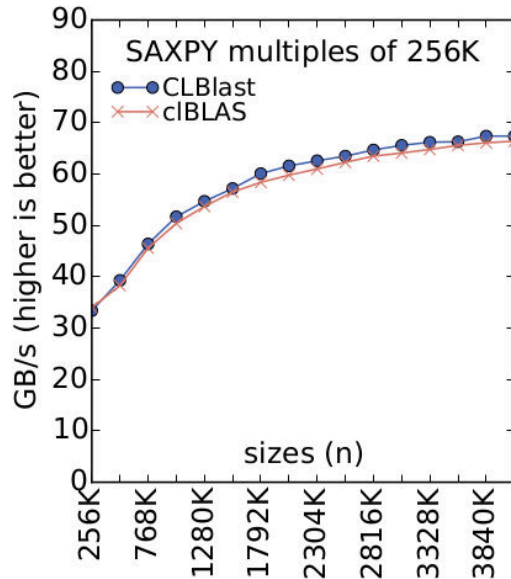
GEMV  
odd  
(in GB/s)

GEMM  
regular  
(in GFLOPS)

GEMM  
odd  
(in GFLOPS)

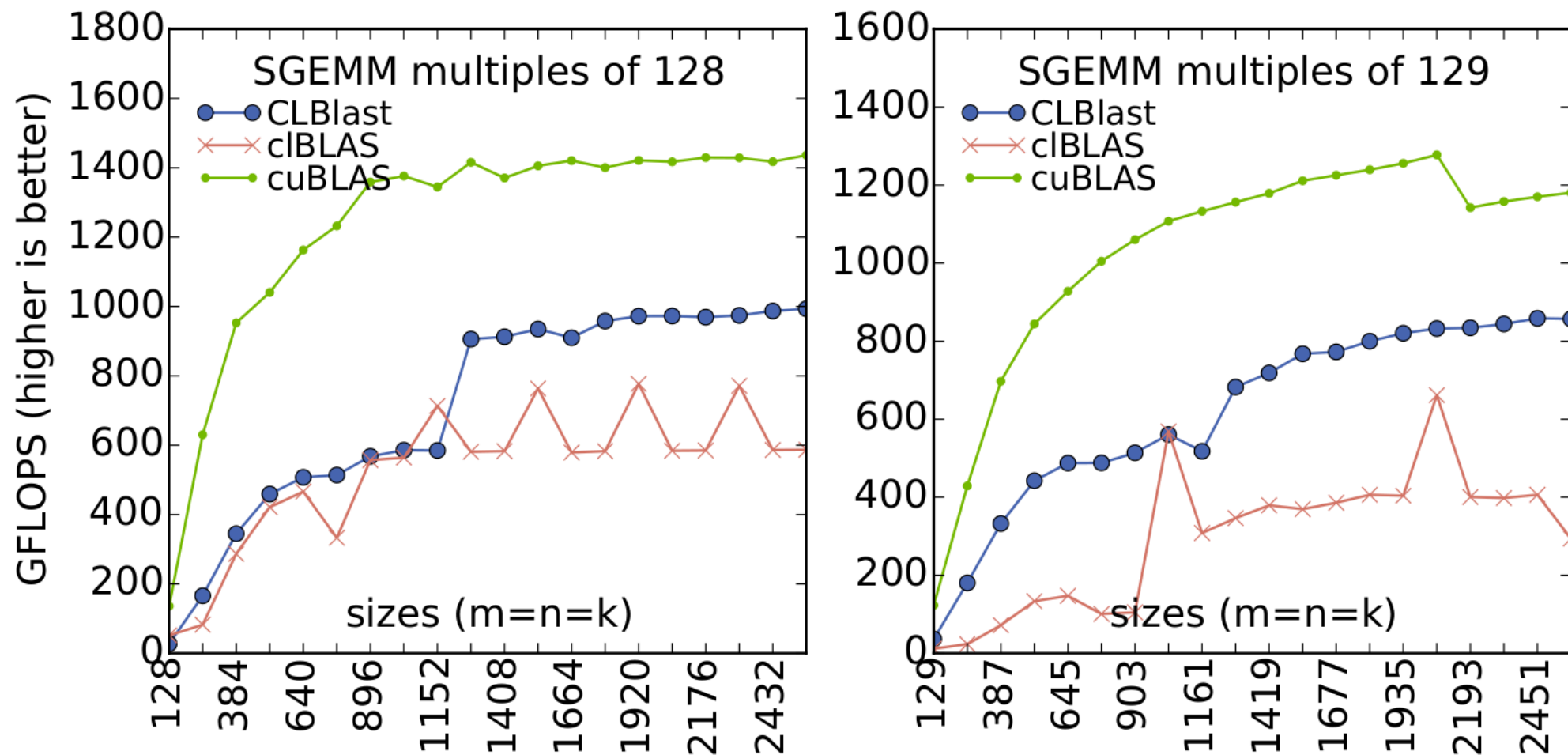
- Higher is better
- More results at <http://cnugteren.github.io/clblast>

# CLBlast on GeForce GTX750Ti



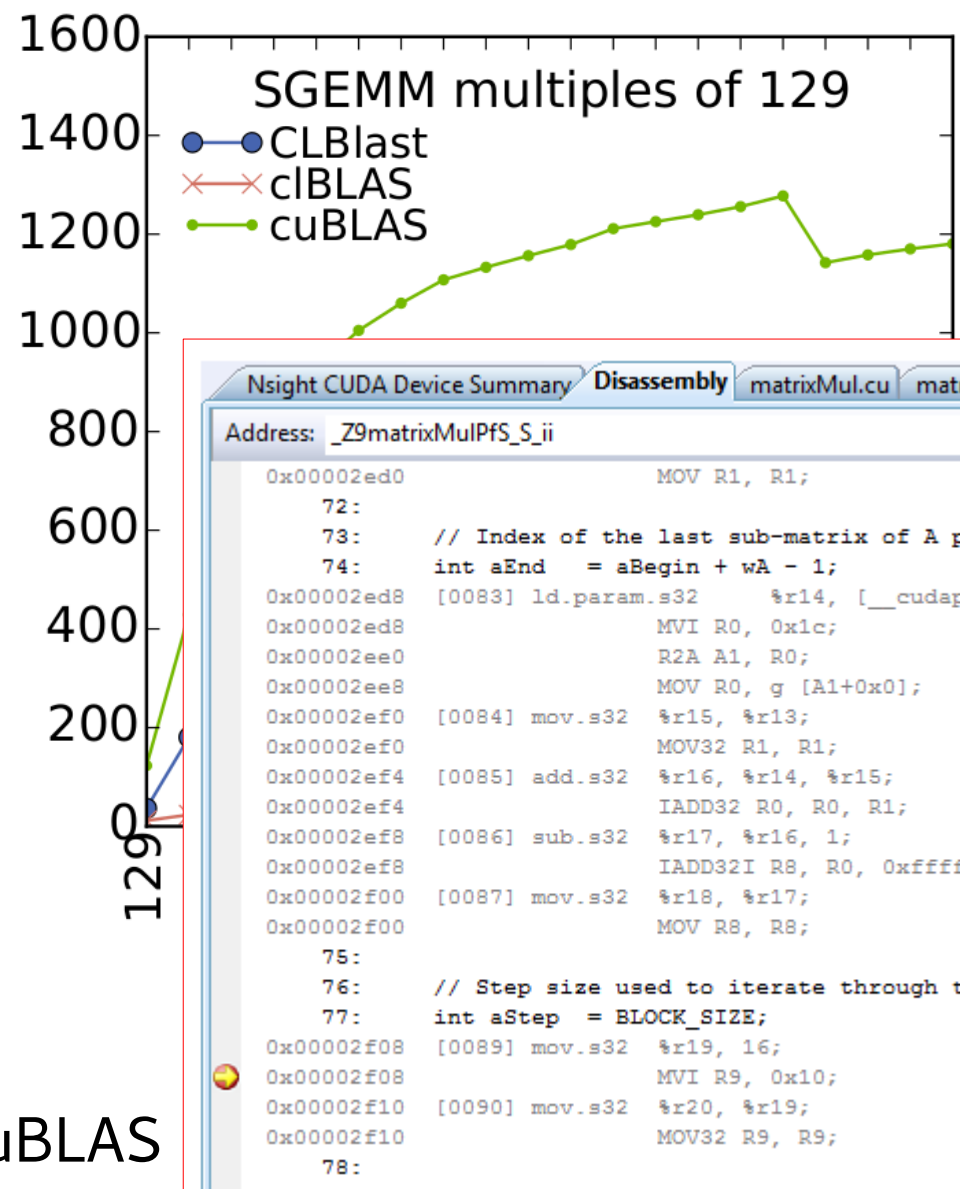
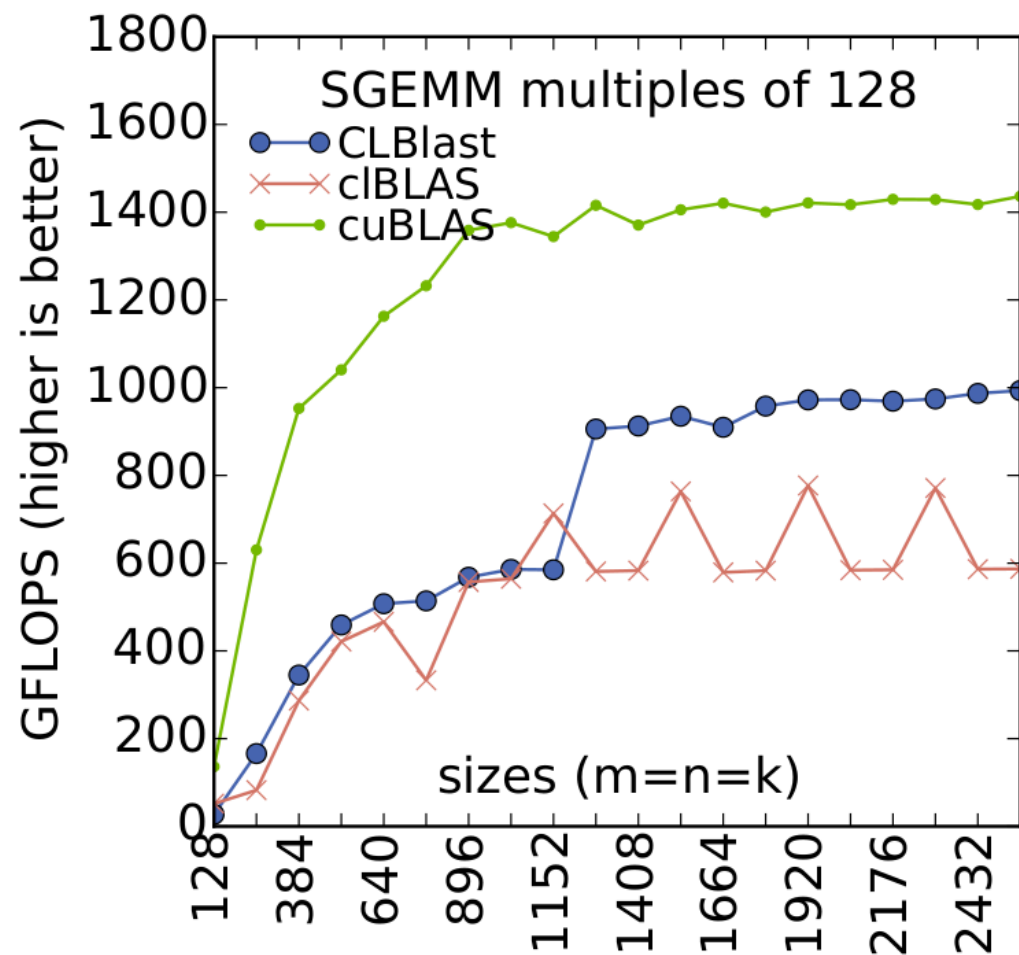
- On-par or better than cBLAS (especially for GEMM)

# CLBlast on GeForce GTX750Ti



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

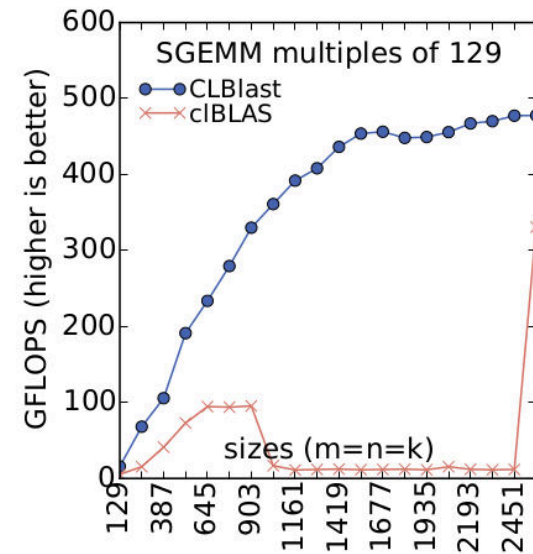
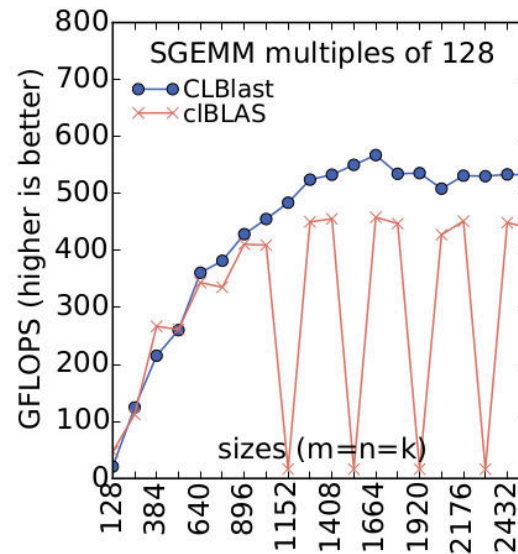
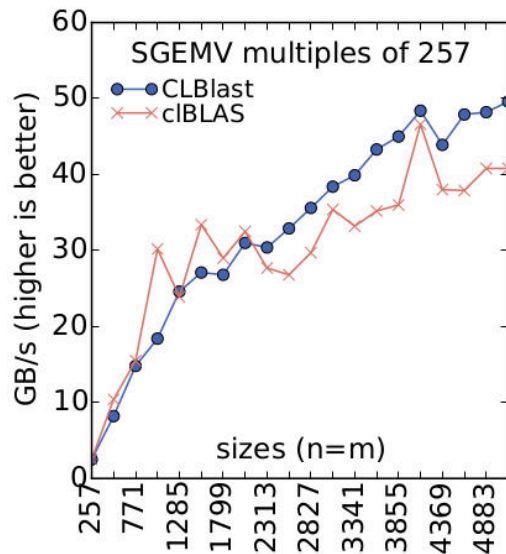
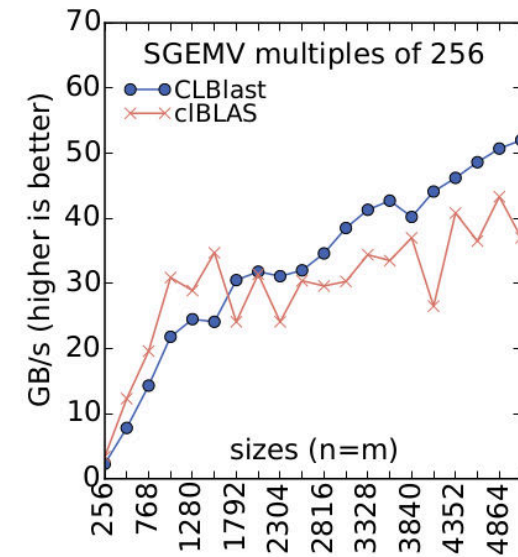
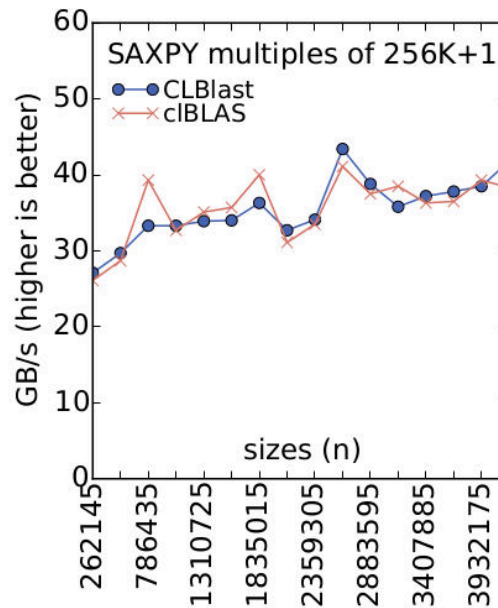
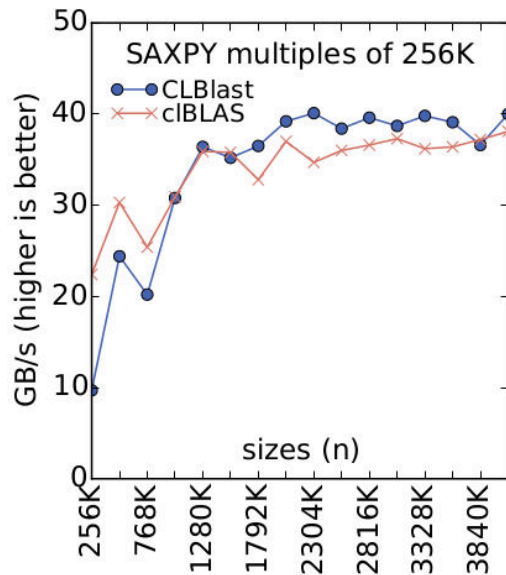
# CLBlast on GeForce GTX750Ti



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

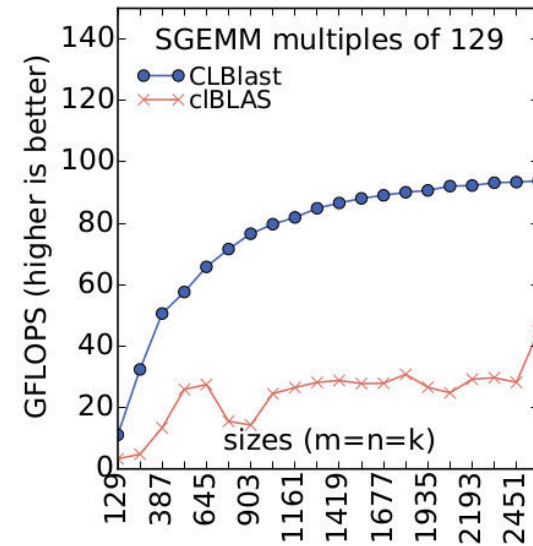
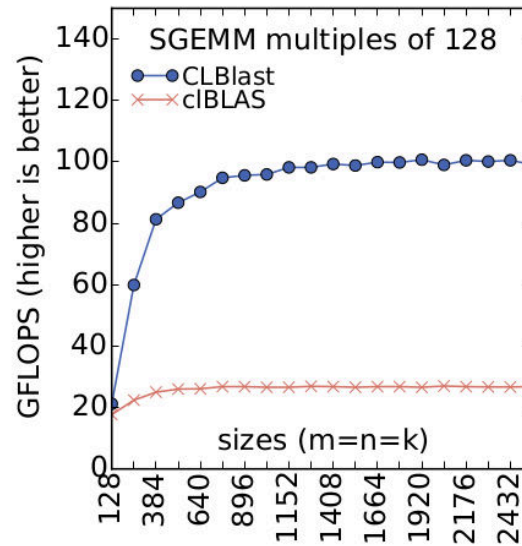
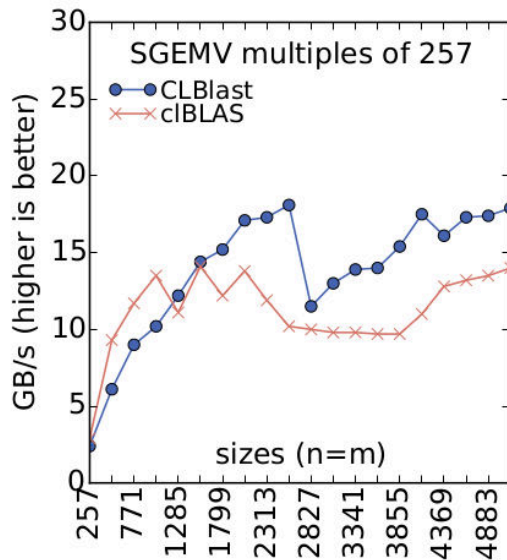
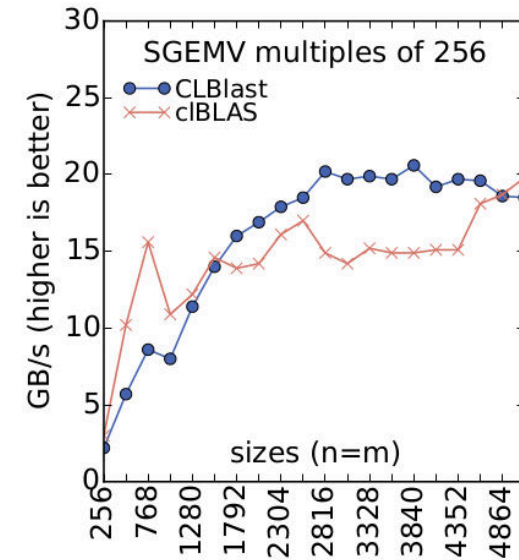
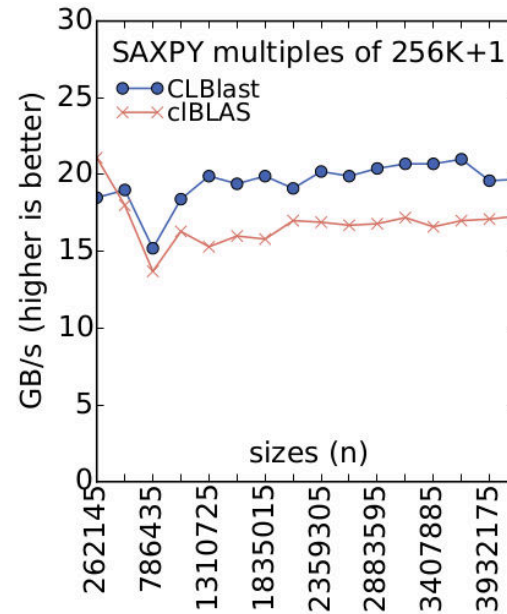
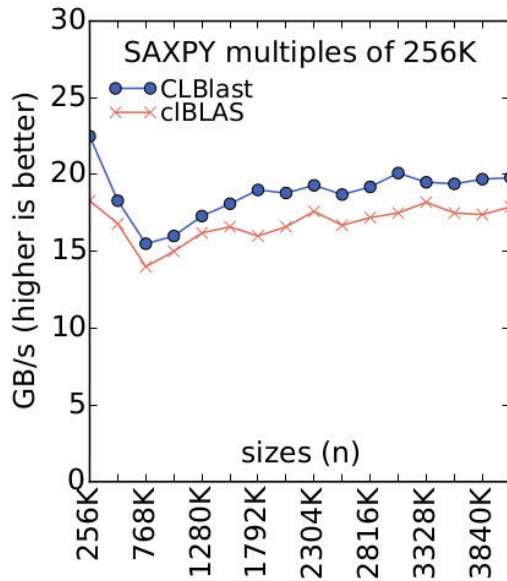


# CLBlast on Radeon M370X



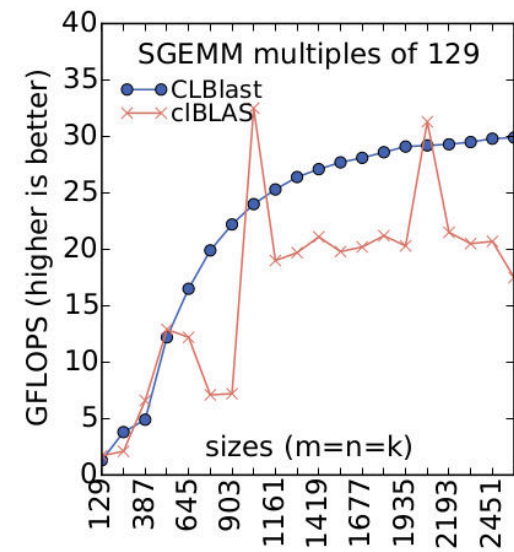
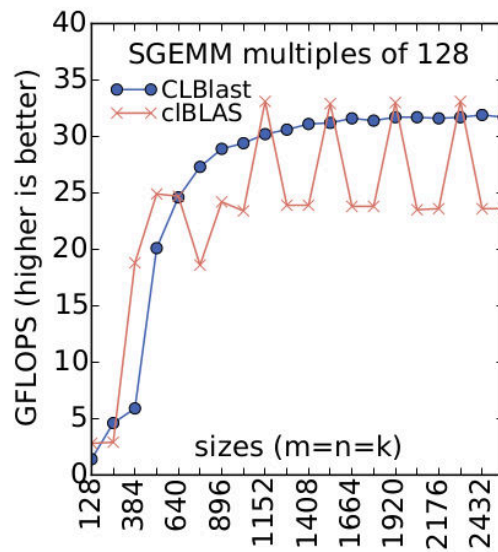
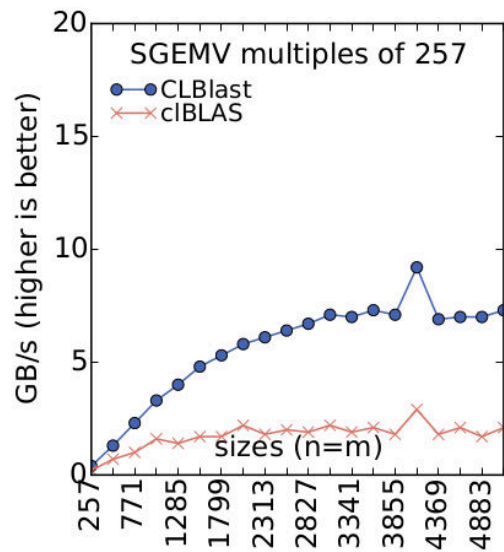
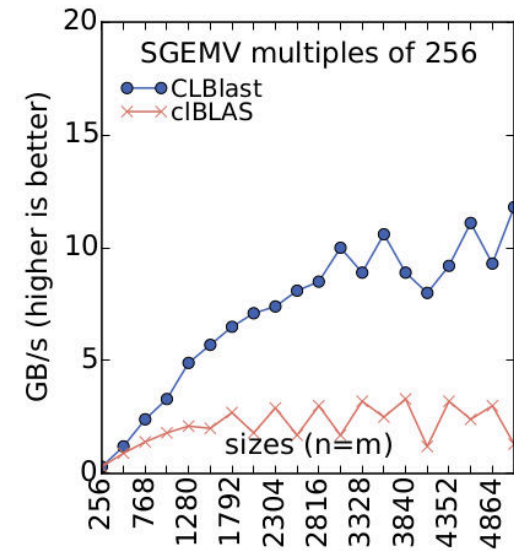
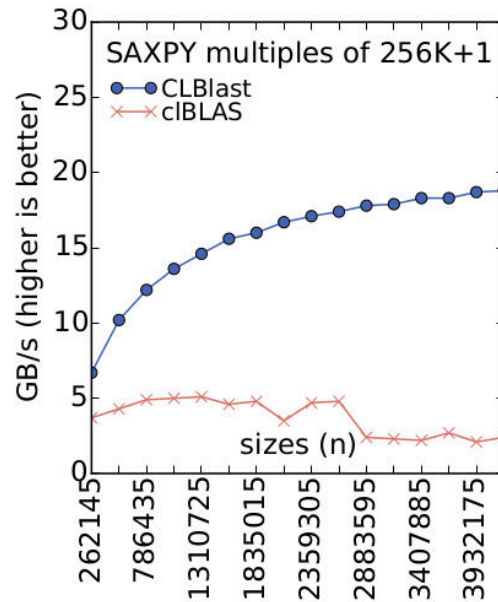
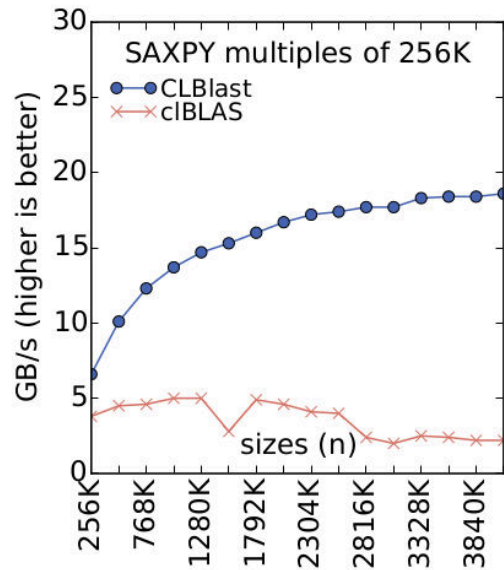
- On-par or better than cBLAS (especially for odd-sized GEMM)

# CLBlast on Skylake ULT GT2



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

# CLBlast on Core i5-6200U



- On-par or better than cBLAS (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

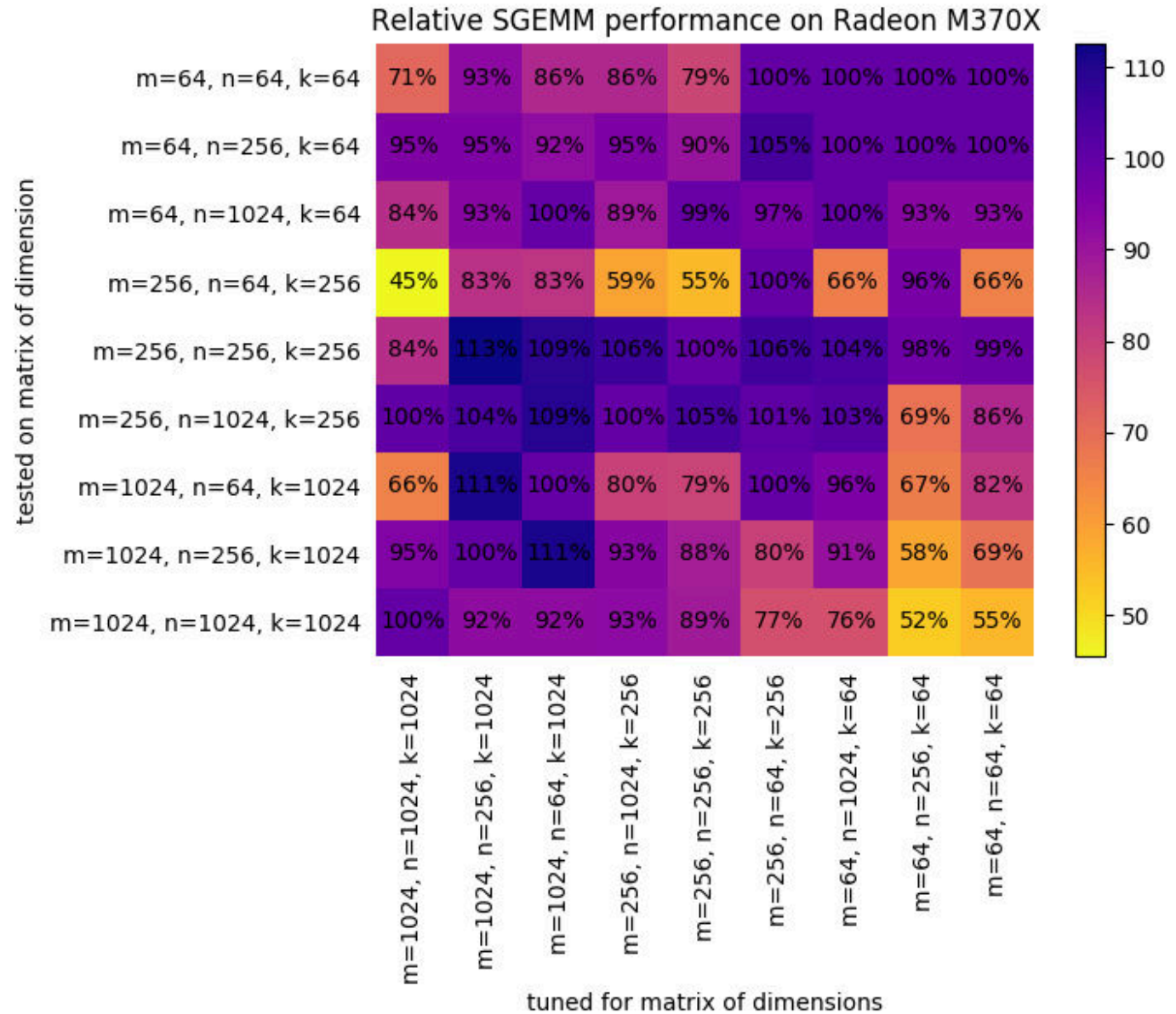
# Tuning Only for a Single Size?

- 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



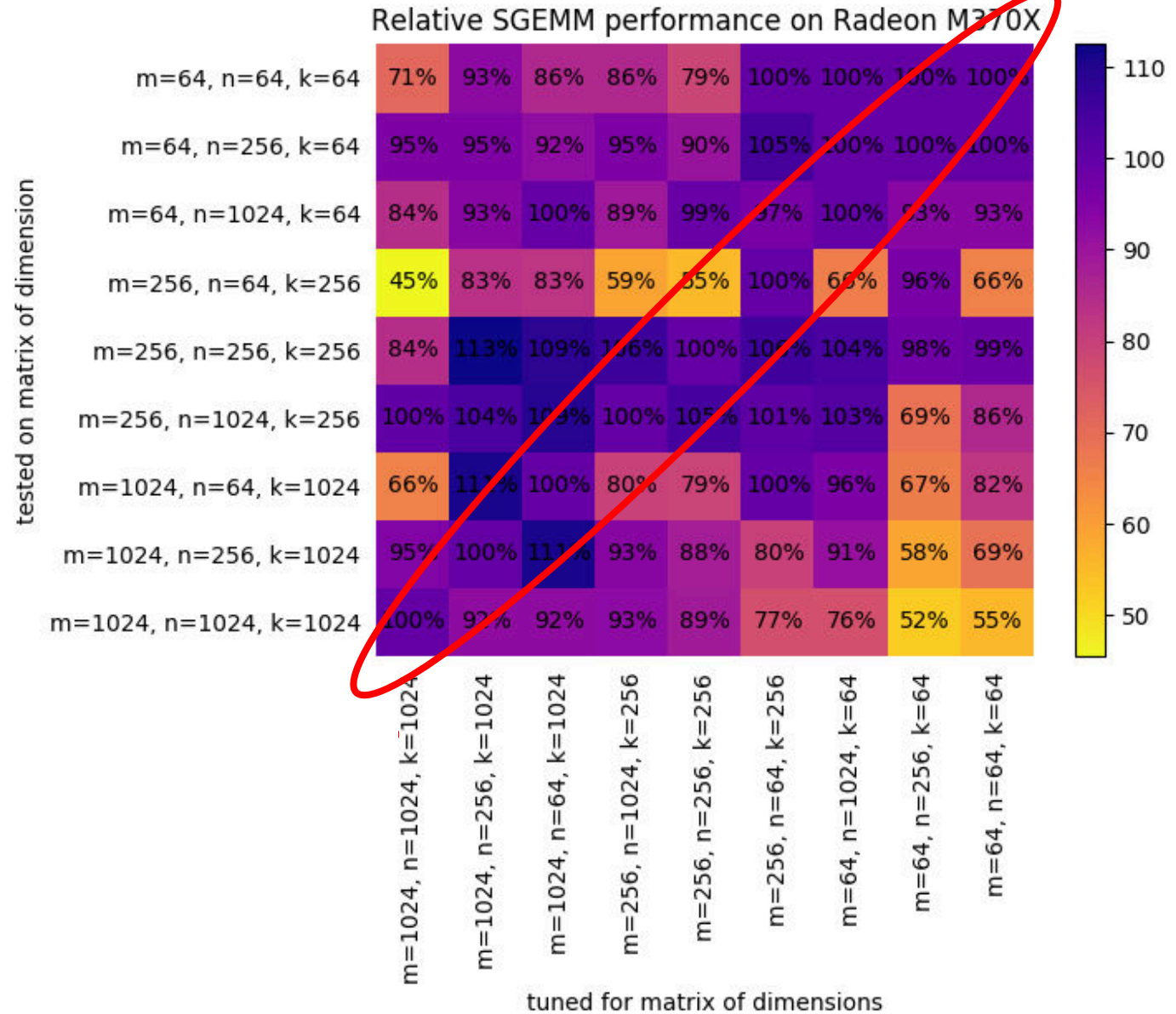
# Problem-Specific Tuning

- SGEMM tuning for Radeon M370X GPU



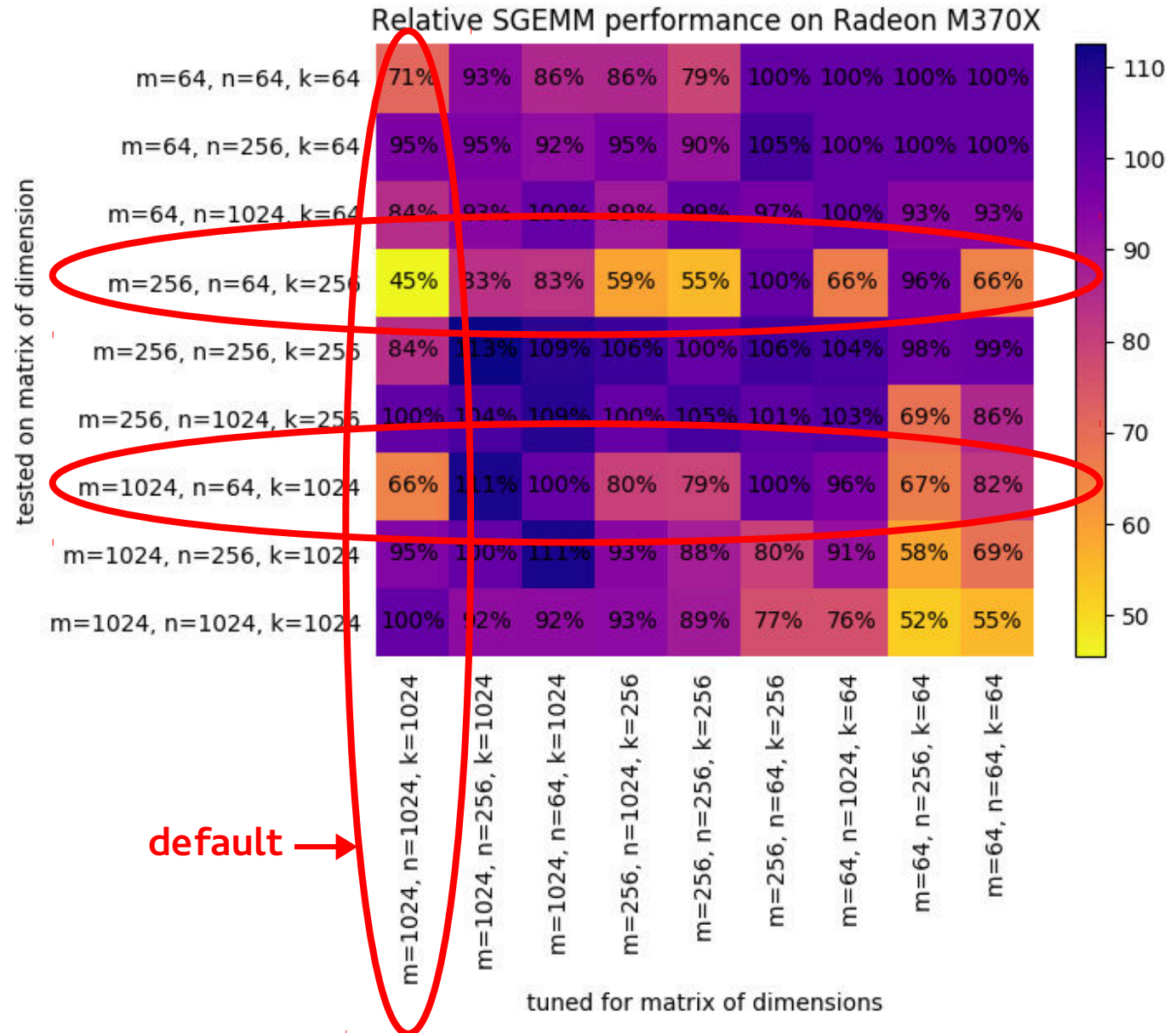
# Problem-Specific Tuning

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



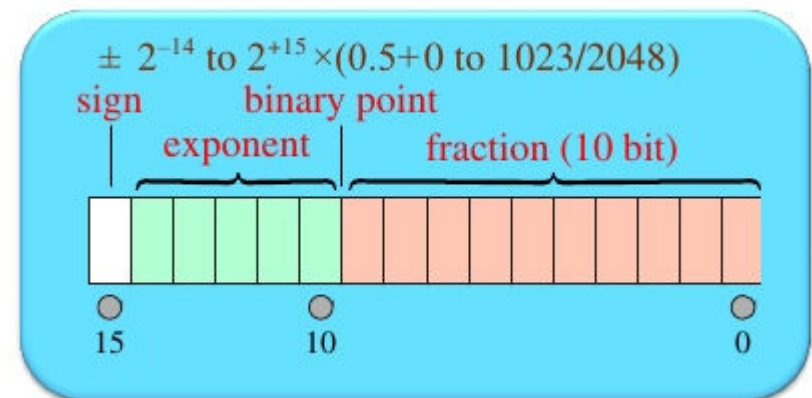
# Problem-Specific Tuning

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



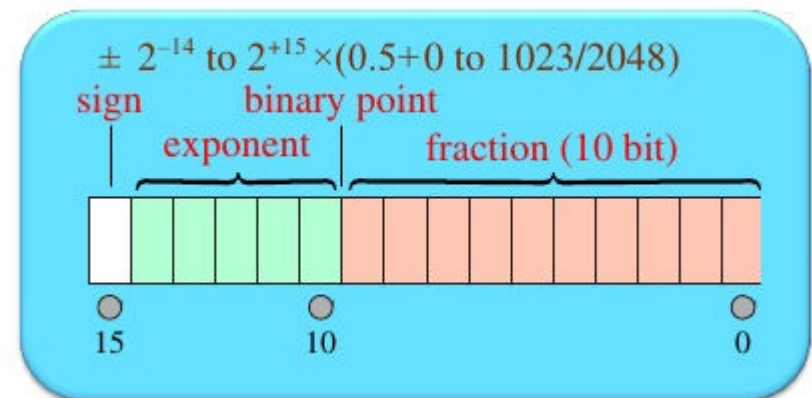
# Half-precision floating-point (FP16)

- 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)



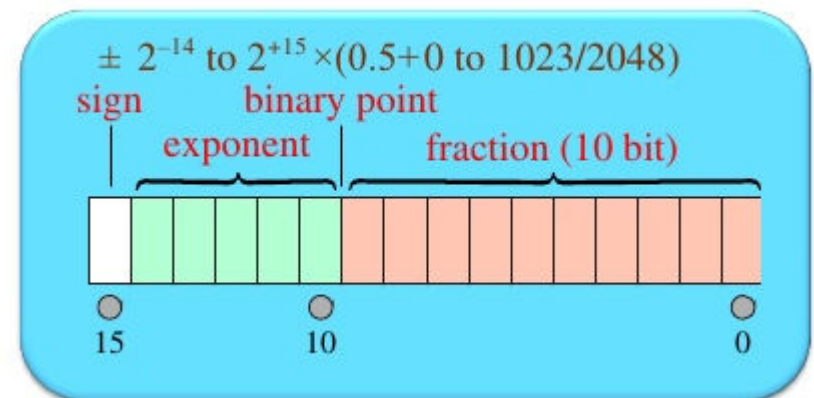
# Half-precision floating-point (FP16)

- 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**



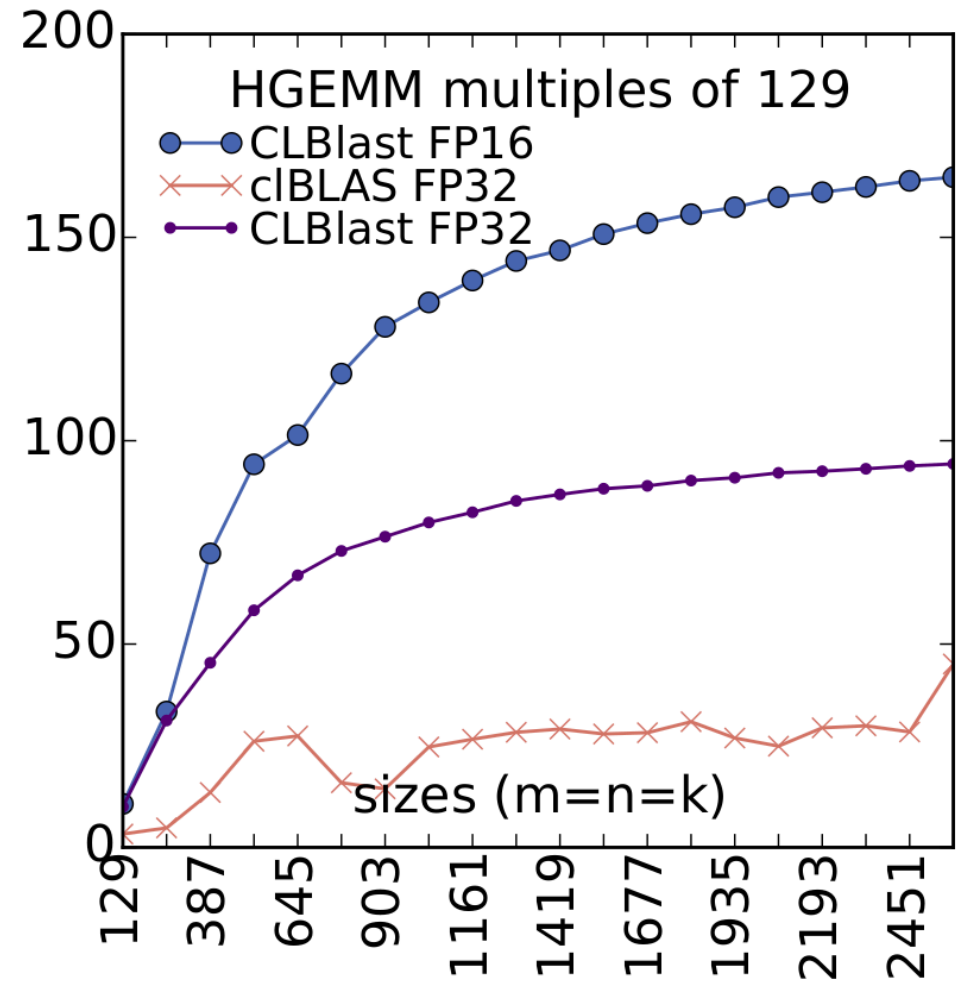
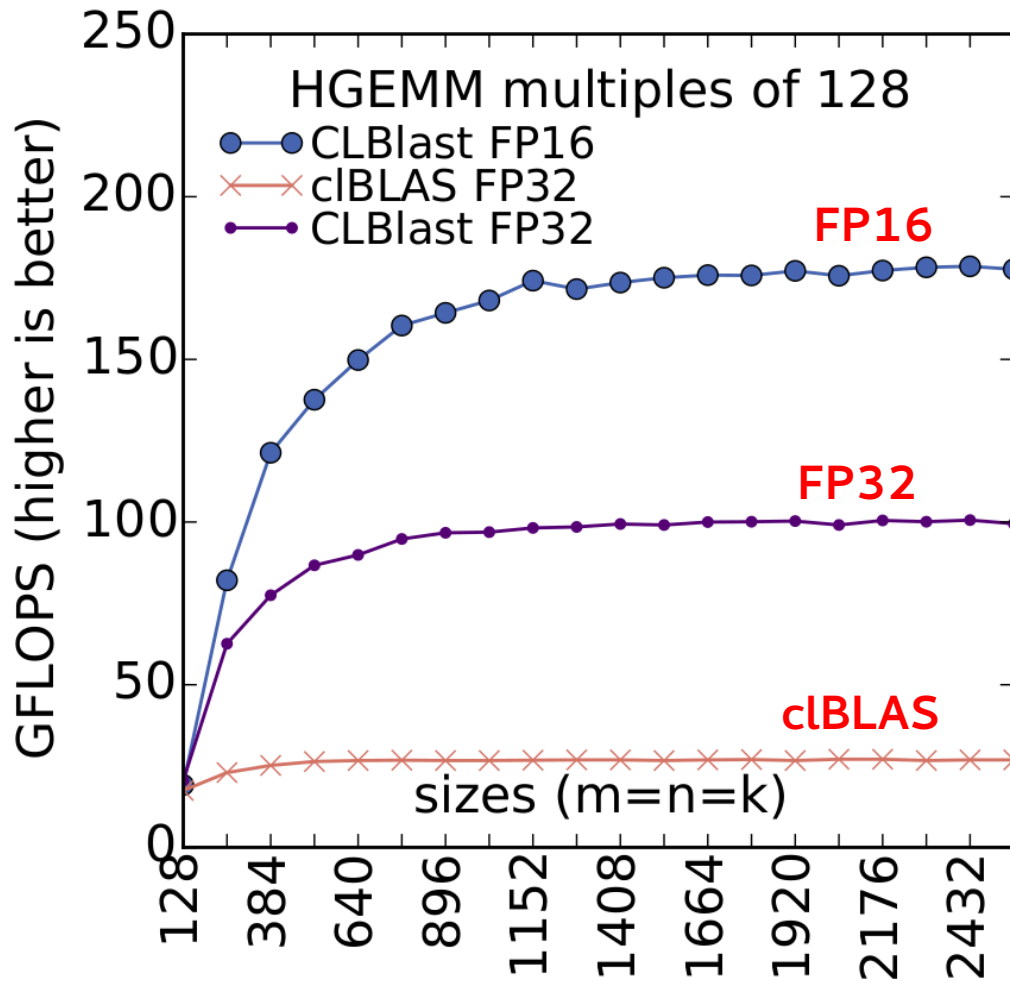
# Half-precision floating-point (FP16)

- 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!**





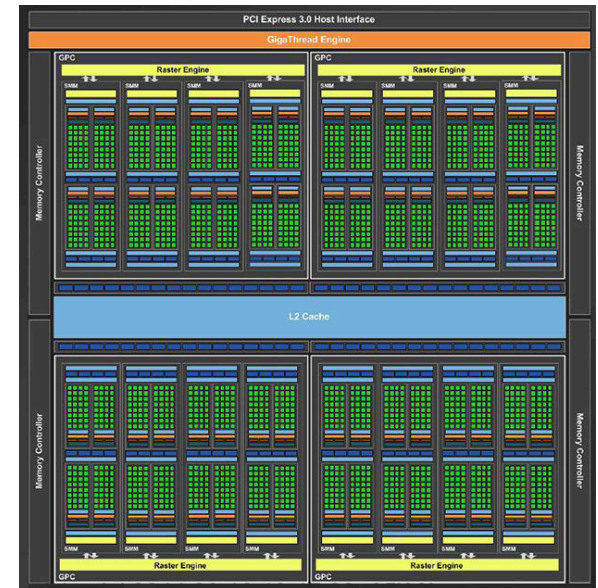
# Half-precision FP16 on Intel Skylake GPU



- FP16 **~1.8x faster** across the board!

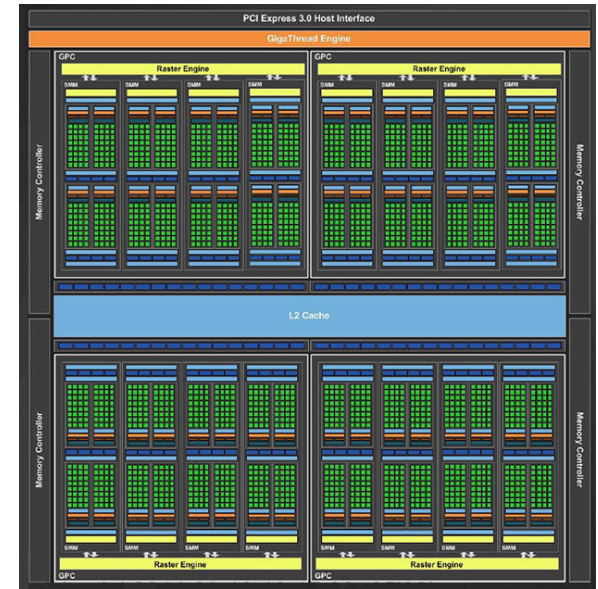
# Batching BLAS routines

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

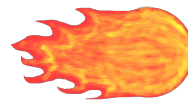


# Batching BLAS routines

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

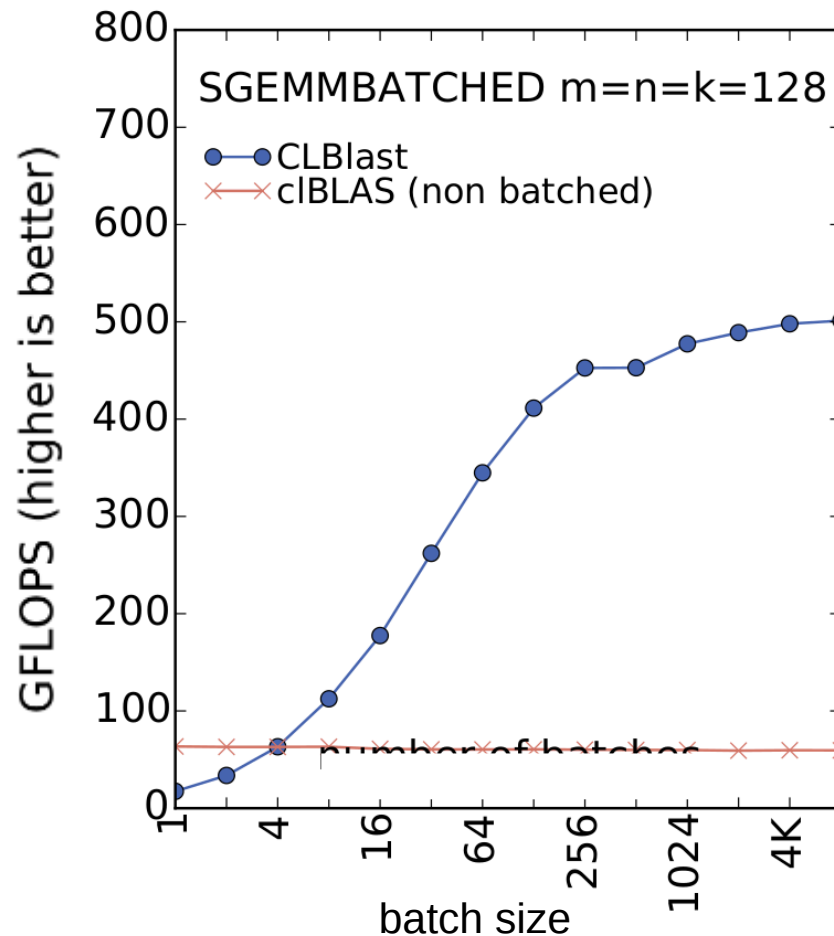


- Let's make it fast again:



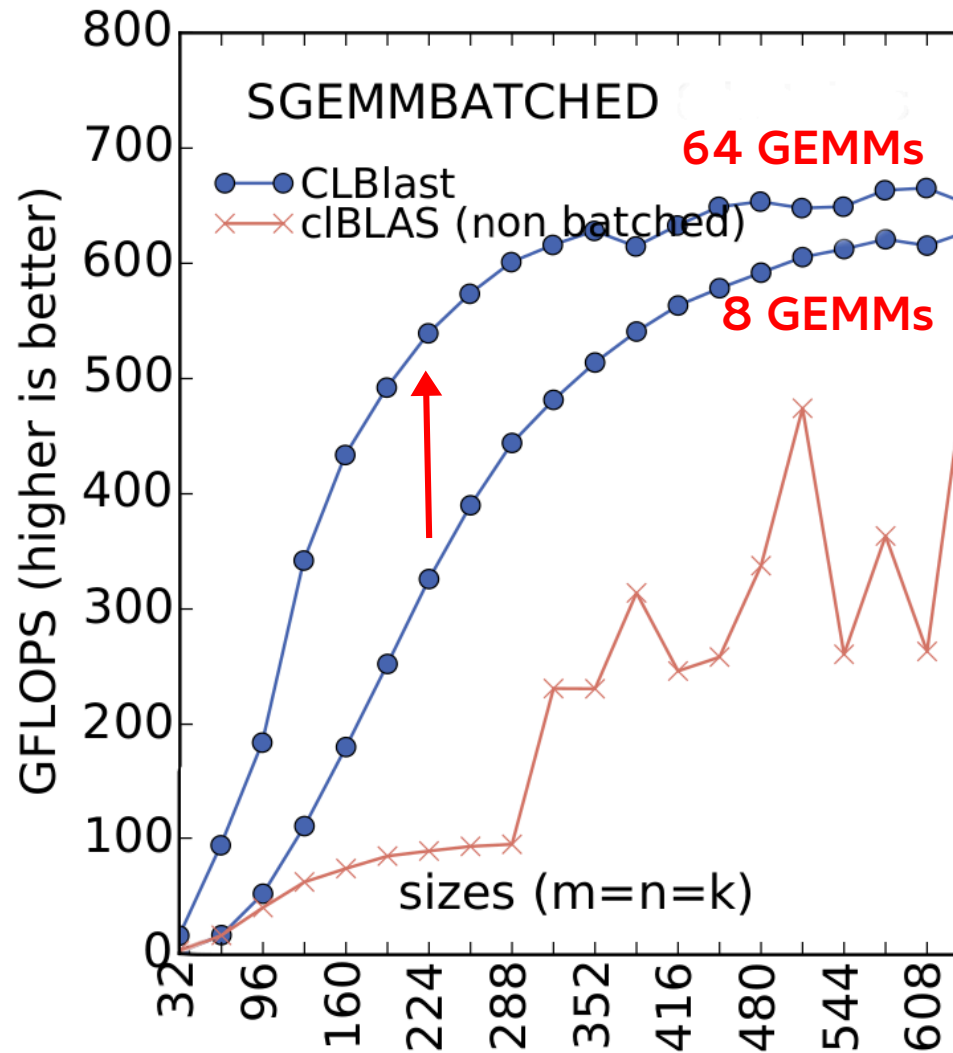
- **Combine multiple small GEMM operations** into a single kernel
- Use offsets to indicate where the next matrices start

# 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



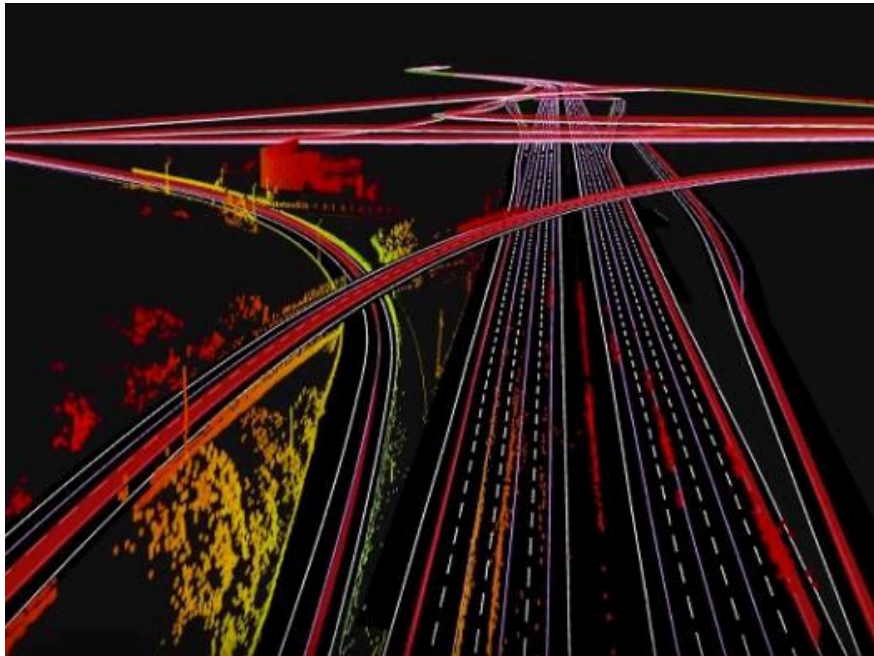
- Significant benefits for larger sizes as well
  - mostly beneficial in the range  $n=64$  till 512

# 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?



- 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



# CLBlast: A Tuned BLAS Library for Faster Deep Learning

Cedric Nugteren

May 11, 2017



<http://github.com/cnugteren/clblast>

<http://cnugteren.github.io/clblast>