COMPUTER SCIENCE

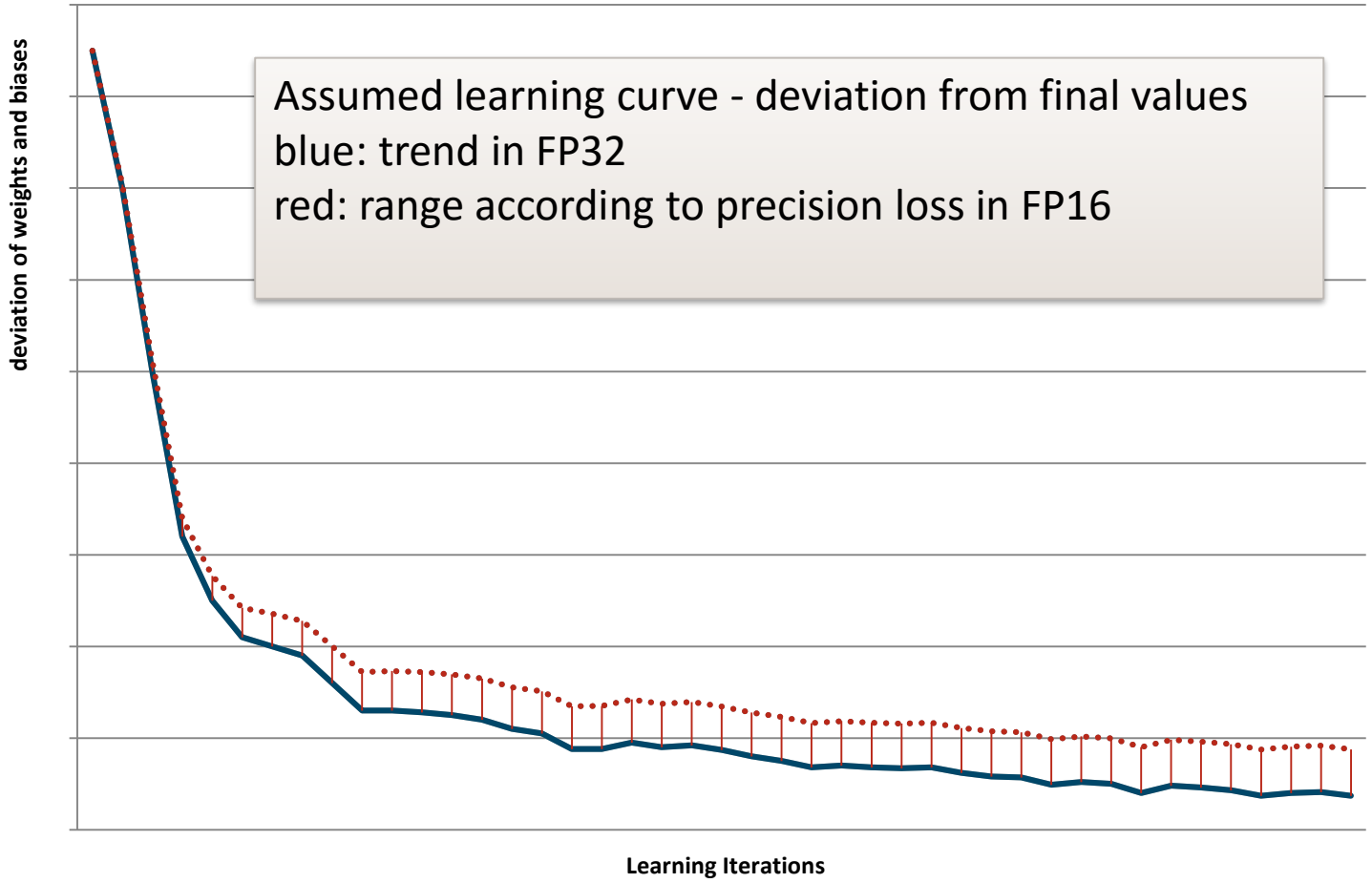
TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Tensor Core

Performance and Precision

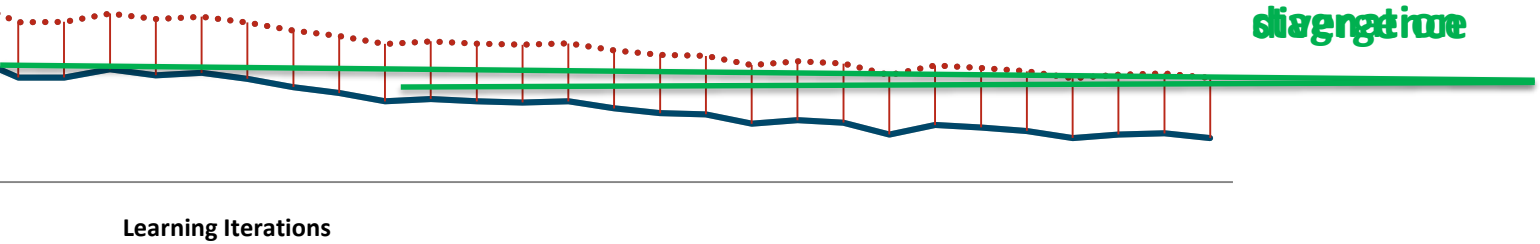
Josef Schüle, University Kaiserslautern, Germany, josef.schuele@rhrk.uni-kl.de

Why attend this Session?



Why attend this Session?

Assumed learning curve - deviation from final values
blue: trend in FP32
red: range according to precision loss in FP16
green: possible behaviours in FP16



Why attend this Session?

Mixed precision

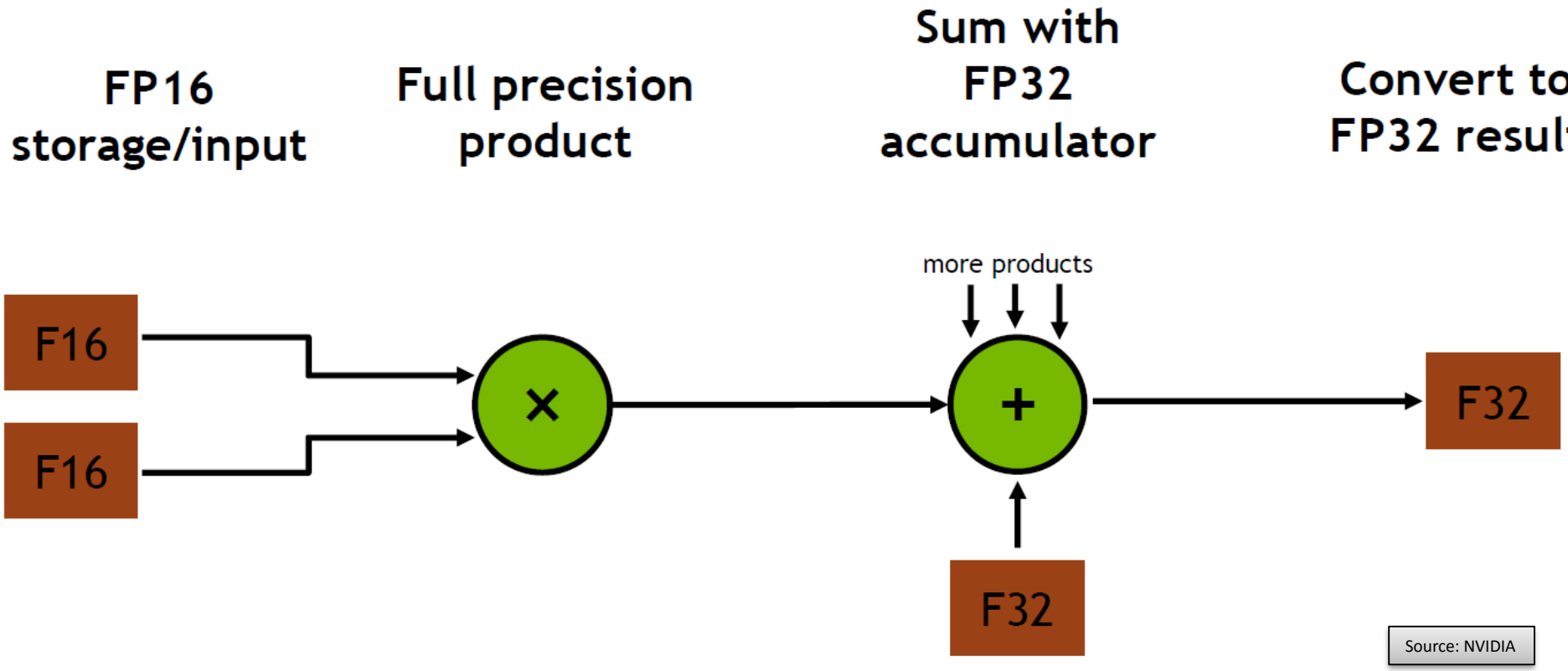
- Each iteration is faster
- Number of iterations is increased

But: Does mixed precision really fasten up the learning?

Outline

- Tensor Cores - Way of Operation and Consequences
- Improving Quality of Tensor Core Usage
- Performance
- Results and Outlook

VOLTA TENSOR OPERATION



How can we use Tensor Cores?

- **Easiest and fastest way - NVIDIAs BLAS library (cublas)**

```
#include "cublas_v2.h"
cublasHandle_t handle=0;
cublasStatus_t cublasStat = cublasCreate(&handle);
cublasStat=cublasSetMathMode(handle,CUBLAS_TENSOR_OP_MATH);
cublasGemmEx(handle,CUBLAS_OP_N,CUBLAS_OP_N,m,n,k,&beta,
            B, CUDA_R_16F, ldb, A, CUDA_R_16F, lda, &alpha,
            C, CUDA_R_32F, ldc, CUDA_R_32F,
            CUBLAS_GEMM_DEFAULT_TENSOR_OP);
```

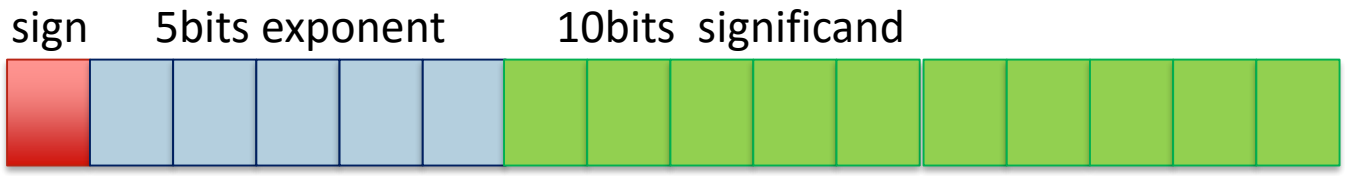
N= 8192 -> 91 Tflops (of 120 Tflops Peak)

Tensor core API

- **Nvidia provides Warp Matrix Multiply Accumulate API**
- **contains very few functionality:**
  - fill_fragment - initialize an accumulator
  - load_matrix_sync - load input data
  - mma_sync - perform the multiplication
  - store_matrix_sync - store result
- **limitations**

  Matrices A, B, C, D may be
  - 8x16 (A), 16x32 (B), 8x32 (C,D)
  - 16x16 (A, B, C, D)
  - 32x16 (A), 16x8 (B), 32x8 (C,D)
  - and - like cublas - it's FORTRAN data layout

What is a FLOAT16?



- maximum absolute value $\pm 65,504$
- machine epsilon 2^{-10} (0.0009765)
- non-uniform precision loss
  - 1,024 representable values for each power-interval i.e.
  - 1,024 representables between 0.0 and 1.0 $([0, 2^0])$
  - 1,024 representables between 1,024 and 2,048
  - 32,768; 32,800; 32,832, … only representables in $[2^{15}, 2^{16}]$

What is a FLOAT32?

sign 8bits exponent 23bits significand

- **maximum absolute value $\pm 10^{38}$**
- **machine epsilon 2^{-23} (10^{-7})**

Conversion of FLOAT32 x to FLOAT16 produces round(x) with

- **abserr(x)= |round(x)-x|**
- **Significands b11..b23 are lost (assuming proper range)**
- **relerr(x)=abserr(x)/|x|=2^{-10}=eps**

Multiply-Accumulate with Float16

x=$(2^{-6}, 2^{-6}, 2^{-6}, 2^{-6})$
y=$(2^{-5}, 2^{-5}, 2^{-5}, 2^{-5})$

Float32

$$s = x^T \cdot y = 4 \cdot \left(2^{-6} \cdot 2^{-5}\right) = 4 \cdot 2^{-11} = 2^{-9}$$
$$= 1.95 \cdot 10^{-3}$$

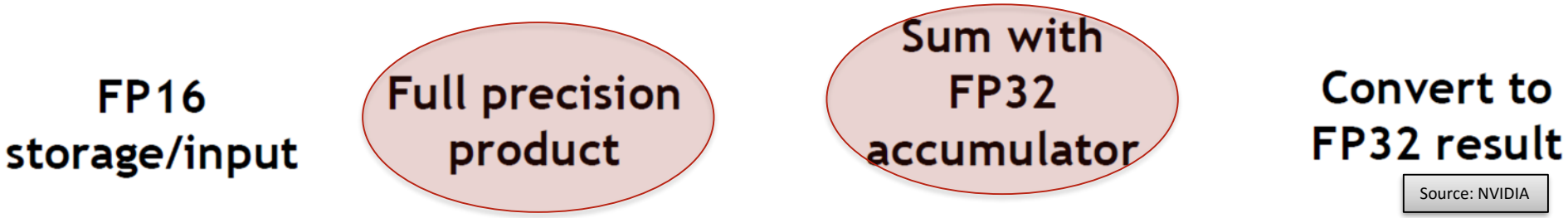
Float16

abserr(x)=abserr(y)= 0. (no initial rounding error)

Conversion of intermediate product 2^{-11} into float16 results in 0.

Final result in float16 is 0., abserr(s)=2^{-9}.

VOLTA TENSOR OPERATION



Additional rounding errors are prevented.

If abserr(__float2half(.)) = 0., it remains 0.

Thus - good and important that FP16 products are accumulated in FP32 precision.

absolute error and matrix values

abserr(x)=eps=2^{-10} for $x \in [0, 2^0]$

abserr(x)=2 eps=2^{-9} for $x \in [2^1, 2^2]$

abserr(x)=1 for $x \in [2^{10}, 2^{11}]$

abserr(x)=32 for $x \in [2^{15}, 2^{16}]$

absolute rounding error increases with value

absolute error and matrix size

x=$(1 - 2^{-11})$

Float32

$$s = x^T \cdot x = 1 - 2^{-10} + 2^{-22}$$

If x is a vector of length *N*, $s = N - N \cdot 2^{-10} + N \cdot 2^{-22}$

Float16

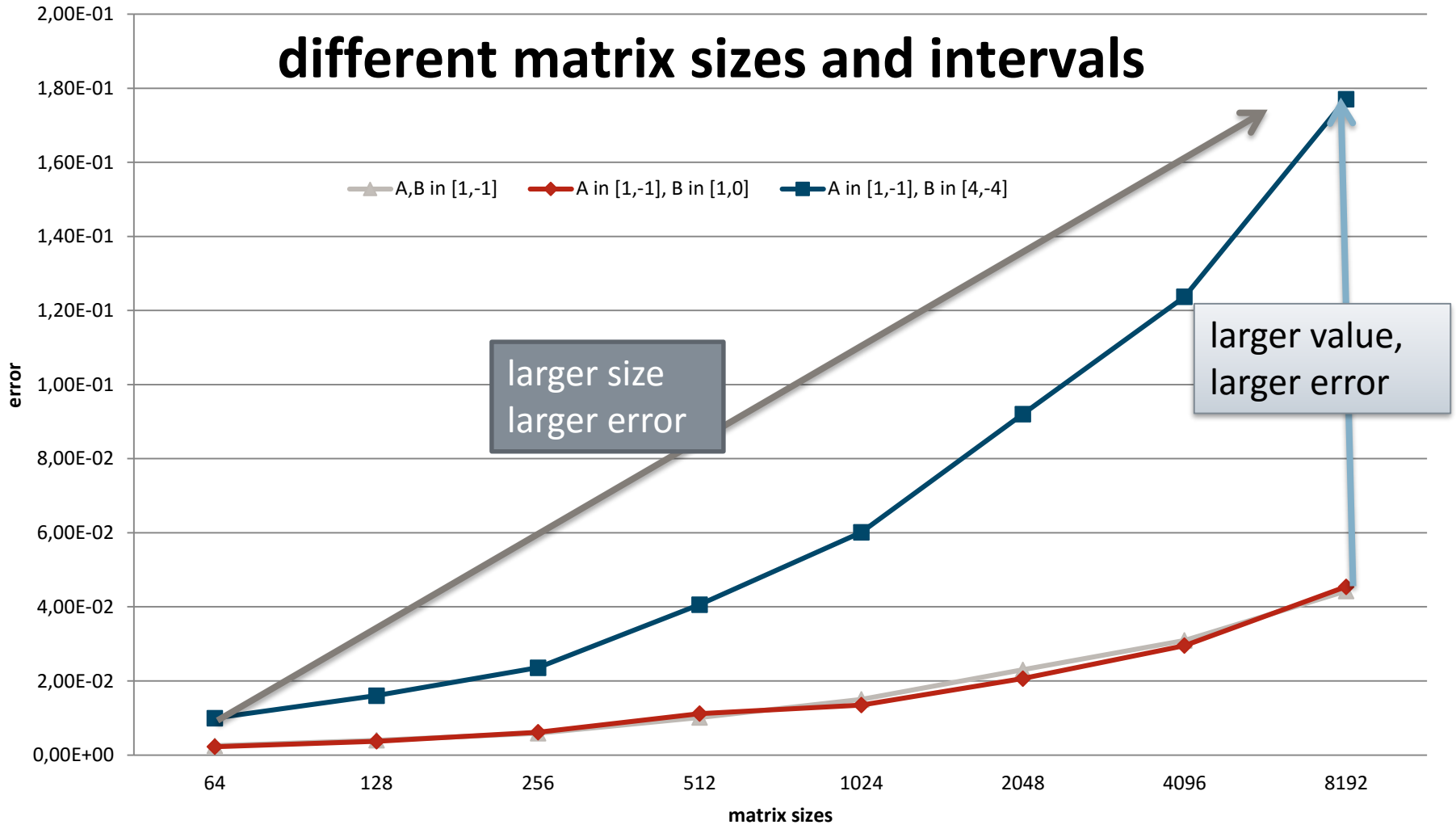
x=1., abserr(x) =2^{-11}

$$s = x^T \cdot x = 1$$

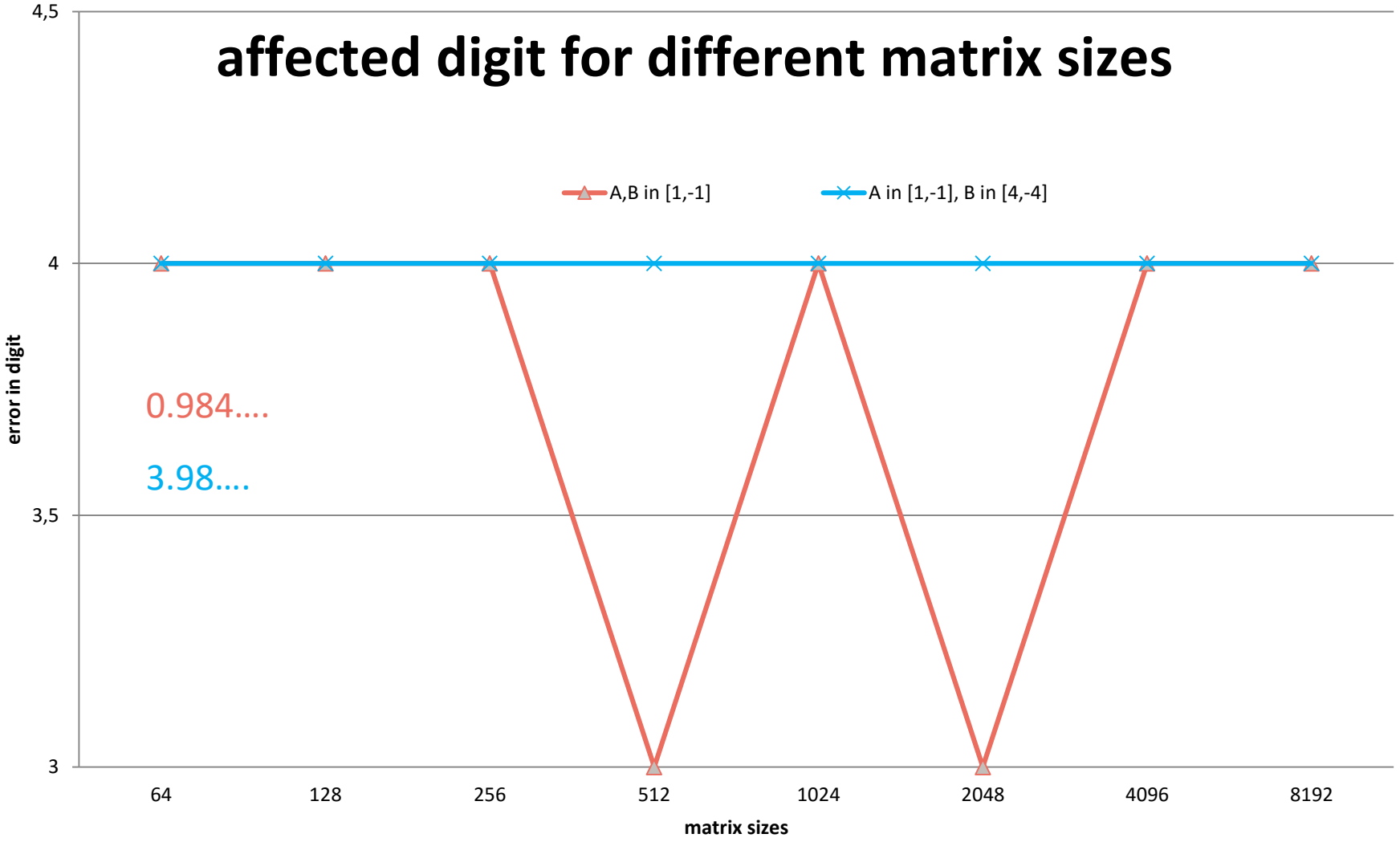
If x is a vector of length N, $s = \mathrm{N}$

Final result in float16 is N, abserr(s)≈ $N \cdot 2^{-10}$.
Rounding errors increase with matrix size.

absolute errors for C=AB



But - it is the 4th digit



range problems

NVIDIA and others recommend scaling to prevent over- or underflow

- reason: small gradient values otherwise will be ignored because they are treated as 0.

Assume all entries of A below threshold T.

Scale A with σ: $\tilde{A} = \sigma A$

D=αAB+βC becomes: $D = \frac{\alpha}{\sigma} \tilde{A}B + \beta C$

Larger value, larger error: $\text{abserr}(\tilde{A}B) \approx \sigma \, \text{abserr}(AB)$

Division by σ: $\text{abserr}(\alpha/\sigma \tilde{A}B) = \alpha/\sigma \, \text{abserr}(\tilde{A}B) \approx \text{abserr}(AB)$

Scaling introduces no additional rounding error to AB, but

scaling factor

Scaling of A may introduce additional rounding:

Choosing a proper scaling factor

Scaling with 1200.

$1200.*(1.+2^{-10})$ = 1201.1718

in FP16 = 1201 (precision loss)

only
1024 representables
for [1024,2048] in
FP16

Scaling with powers of 2 avoids this problem.

scaling factor

Scaling with 1024.

1024.*(1.+2^{-10}) = 1024.+1.=1025.

Scaling with powers of 2 corresponds to a

- **change in the exponent.**
- **significand is unchanged.**

Outline

- **Way of Operation and Consequences**
  - limited range -> scaling, but use powers of 2
  - rounding erros increase with
    - matrix values (scaling has no influence)
    - matrix size
    - 4th digit of result has no significance
- **Improving Quality of Tensor Core Usage**

increasing accuracy

Binomial approach

Markidis et al. Mar 2018.

X(32) ≈ Xh(16) + Xl(16)

with Xh(16)=(half) X(32)

Xl(16) =X(32)-(float)Xh(16)

$$(Xh + Xl) * (Yh + Yl) =$$
$$Xh * Yh + Xh * Yl + Xl * Yh + Xl * Yl$$

- **higher accuracy compared to Xh*Yh in FP16**
- **4 MMAs instead of one**

Example - binomial approach

x(32)=$2^{-1}+2^{-11}-2^{-13}$ 0.5003662

xh(16)=2^{-1}

xl(16) =$2^{-11} - 2^{-13}$

$x^2 \approx 2^{-2} + 2^{-11} - 2^{-13}$

$fp16^2 = 2^{-2}$

$binomial = 2^{-2} + 2^{-11} - 2^{-13}$

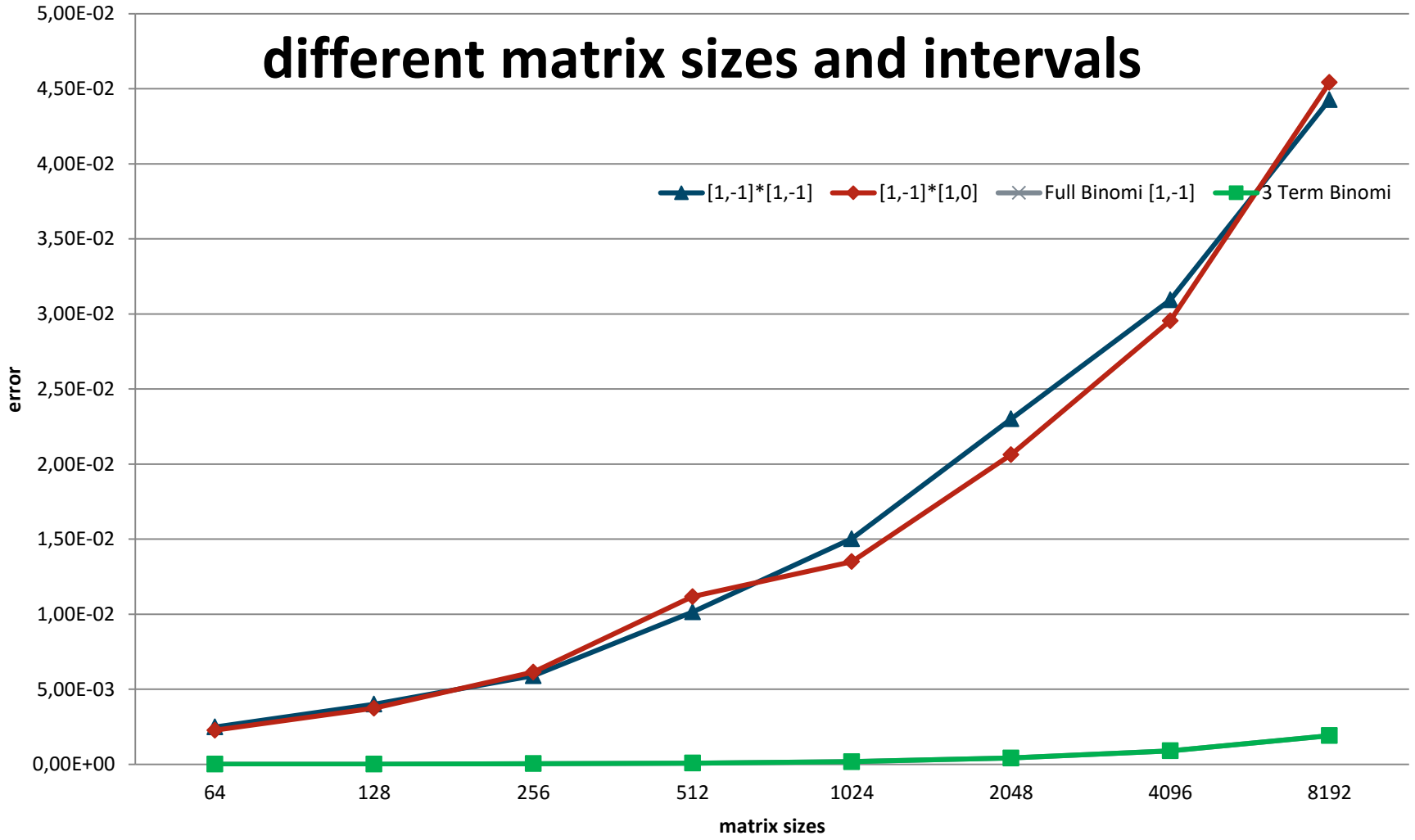
abserr($fp16$)=$2^{-11} - 2^{-13}$ (0.0003662)

abserr(binomial) = 0.

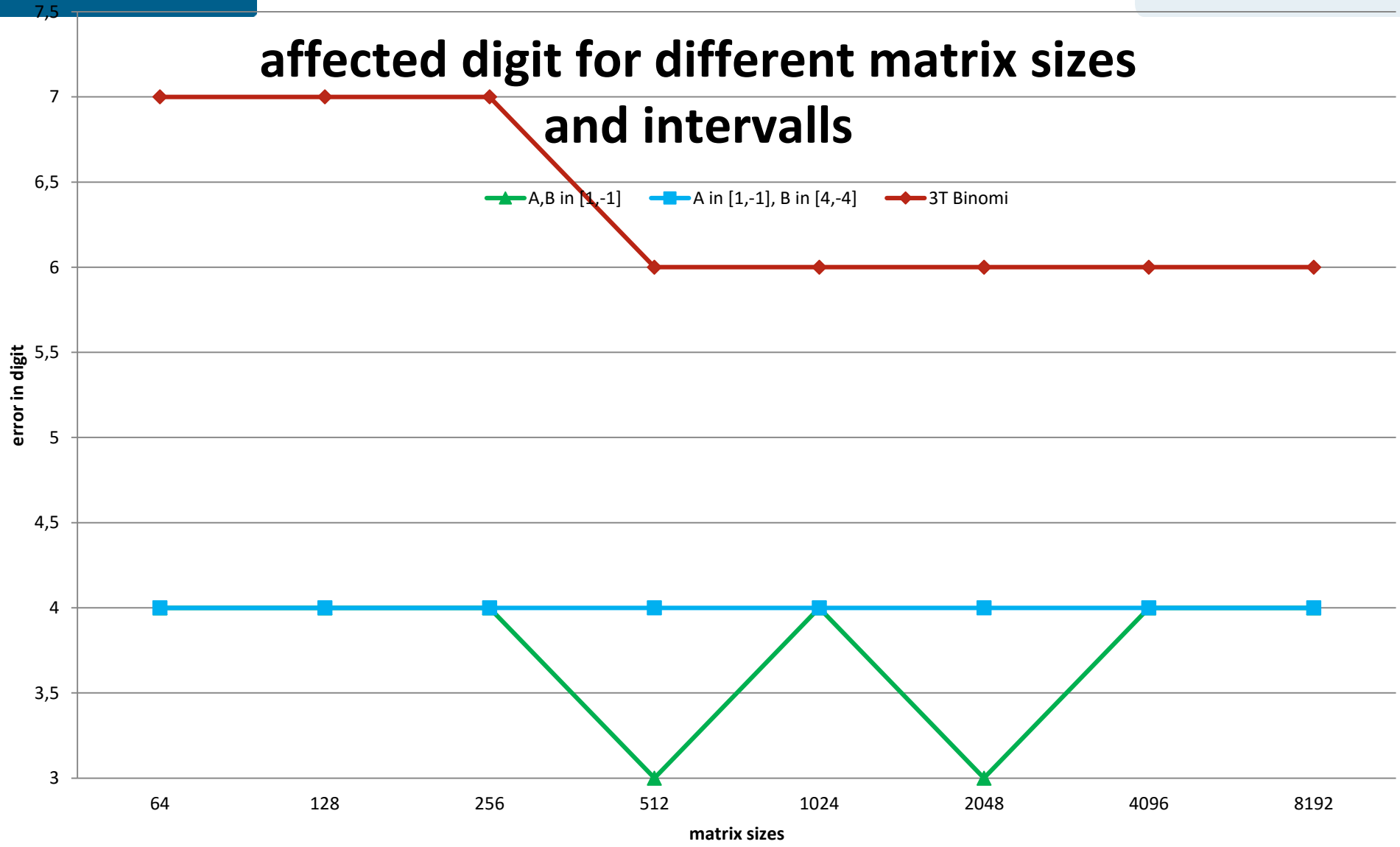
Using these numbers in a 8192x8192 matrix:

abserr(xh^2) $\approx 2^2$

normal vs. binomial



difference in digits



Karatsuba Algorithm

- Fast multiplication algorithm
- Divide a number X into two halves, the high bits h and the low bits l with respect to a base b:

  X=Xh*b+Xl

- Form H=Xh*Yh, L=Xl*Yl, D=(Xh+Xl)(Yh+Yl)-H-L
- Products are formed in lower precision
- Final Product in full precision:

  XY = H*b*b + D*b + L

- Only 3 low precision products needed to form H, L and D (compared to 4 with binomial approach)

Karatsuba Algorithm

Example: X=35, Y=34, b=10

Xh=3, Xl=5, Yh=3, Yl=4

H=3*3=9

L=5*4=20

D=(3+5)(3+4)-9-20=56-29=27

XY=9*b*b+27*b+20 = 900+270+20 = 1190

Only 2-digit multiplications required

Just 3 of them (multiplication by b is a shift operation)

Karatsuba extended to Scaled Binomial

Karatsuba: All operands have similar ranges

Modified Binomial Algorithm:

Use Karatsuba like decompostion of X= Xh+Xl $*2^{10}$

Use Binomi for Operation:

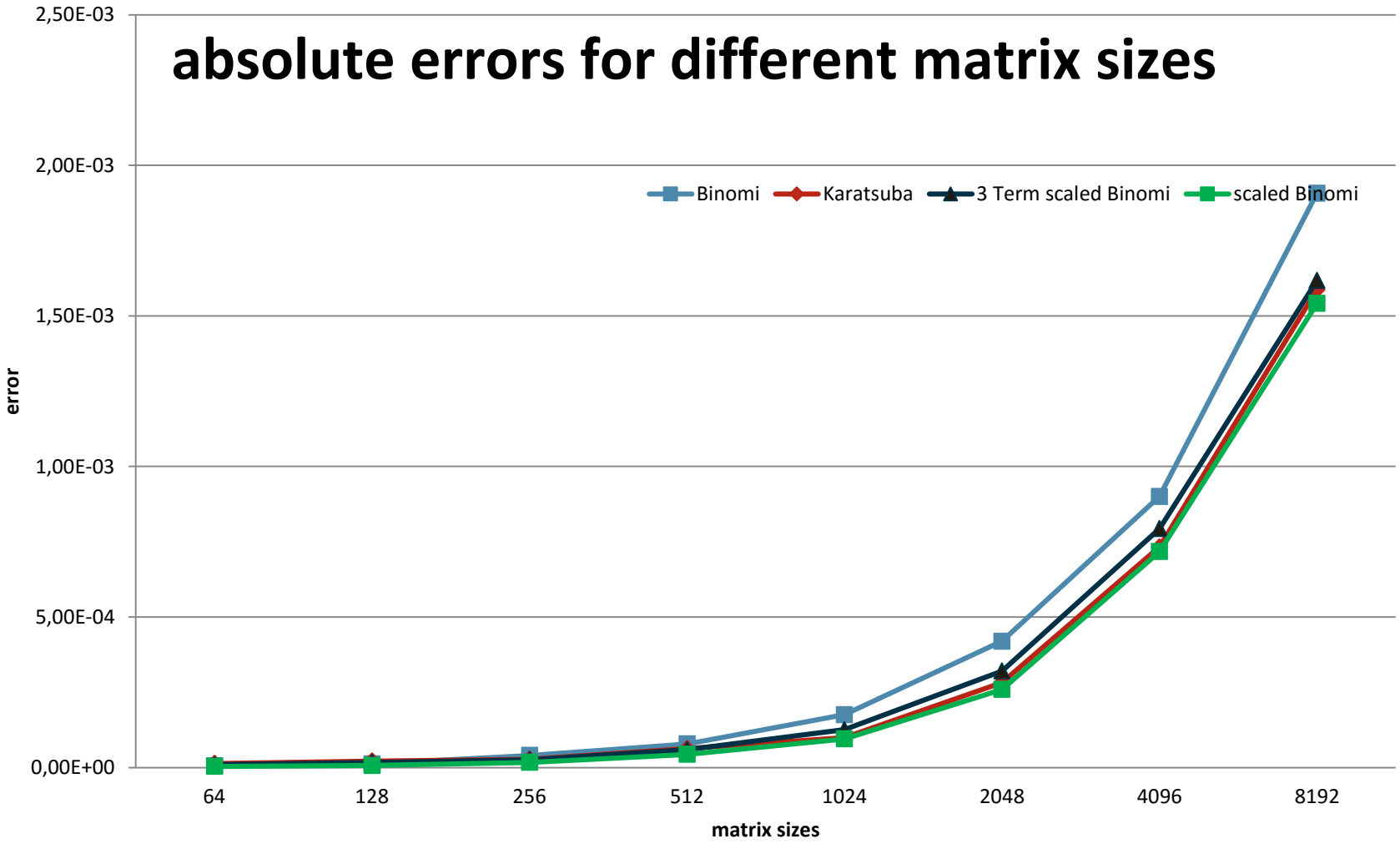
(Xh+Xl)(Yh+Yl) with products

H=Xh*Yh, HL=Xh*Yl, LH=Xl*Yh, L=Xl*Yl

and

XY=H+2^{-10}(HL+LH+2^{-10}L)

different approximations



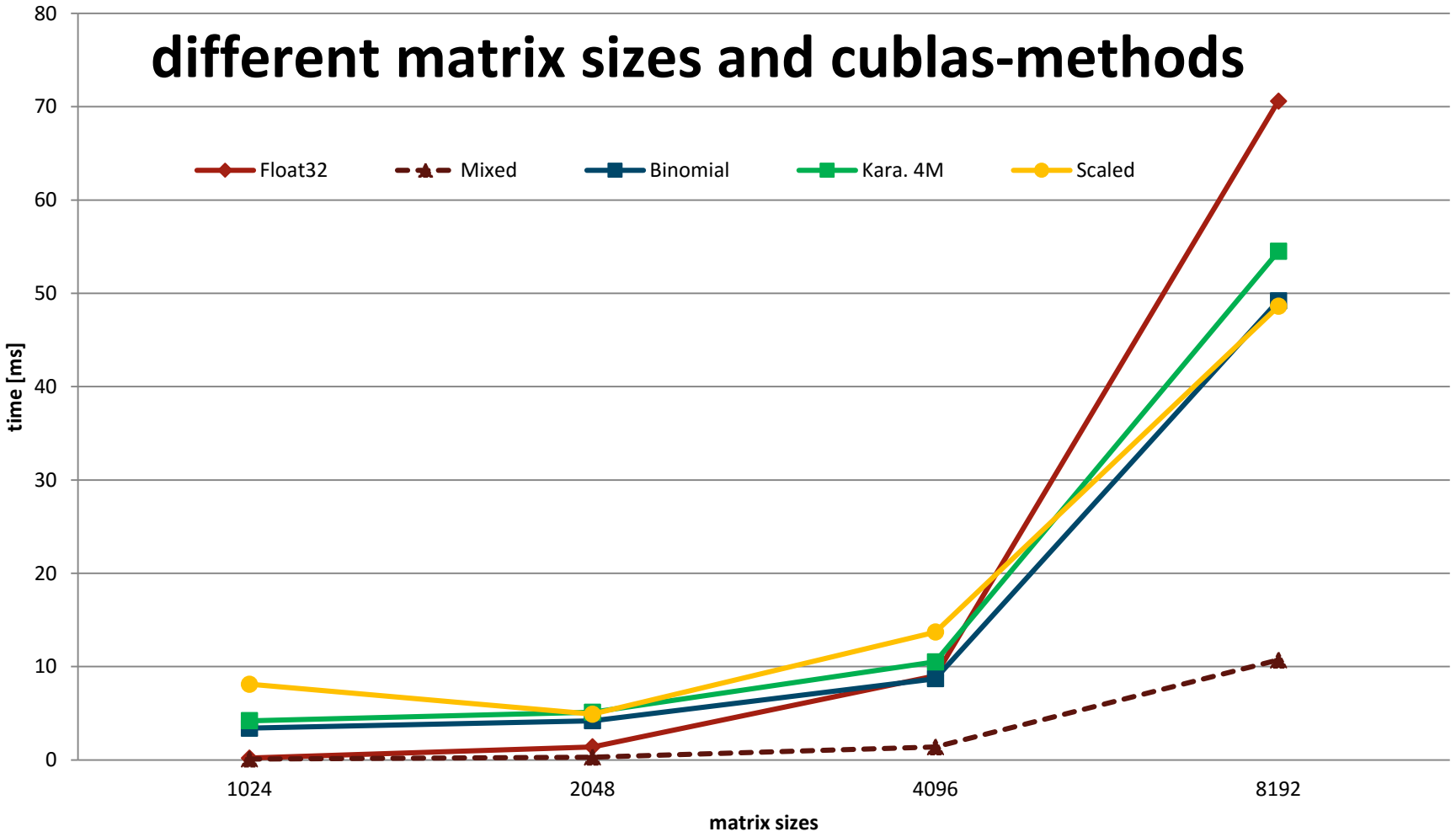
Outline

- Way of Operation and Consequences
- Improving Quality of Tensor Core Usage
  - Binomial multiplication reduces absolute error by one magnitude
  - adds 2-3 significant digits
  - Karatsuba and scaled binomial improve further
  - 3 terms of binomial algorithms are sufficient
- Performance

Implementations

- Volta 100
- Cuda 9.1.85, gcc 6.5
- 10 iteration measured
- time for data preparation not included
- time for data movements (CPU-GPU) not included

Using Cublas



cublas - Thinks to Remember

- Mixed precision really pais off at larger matrix sizes (4096+)
- Implementations for precision improvements slower than float32 for small sizes
  - use float32 instead
- For large matrices precision improvement algorithms are faster than float32 and may be worth a try

WMMA-API

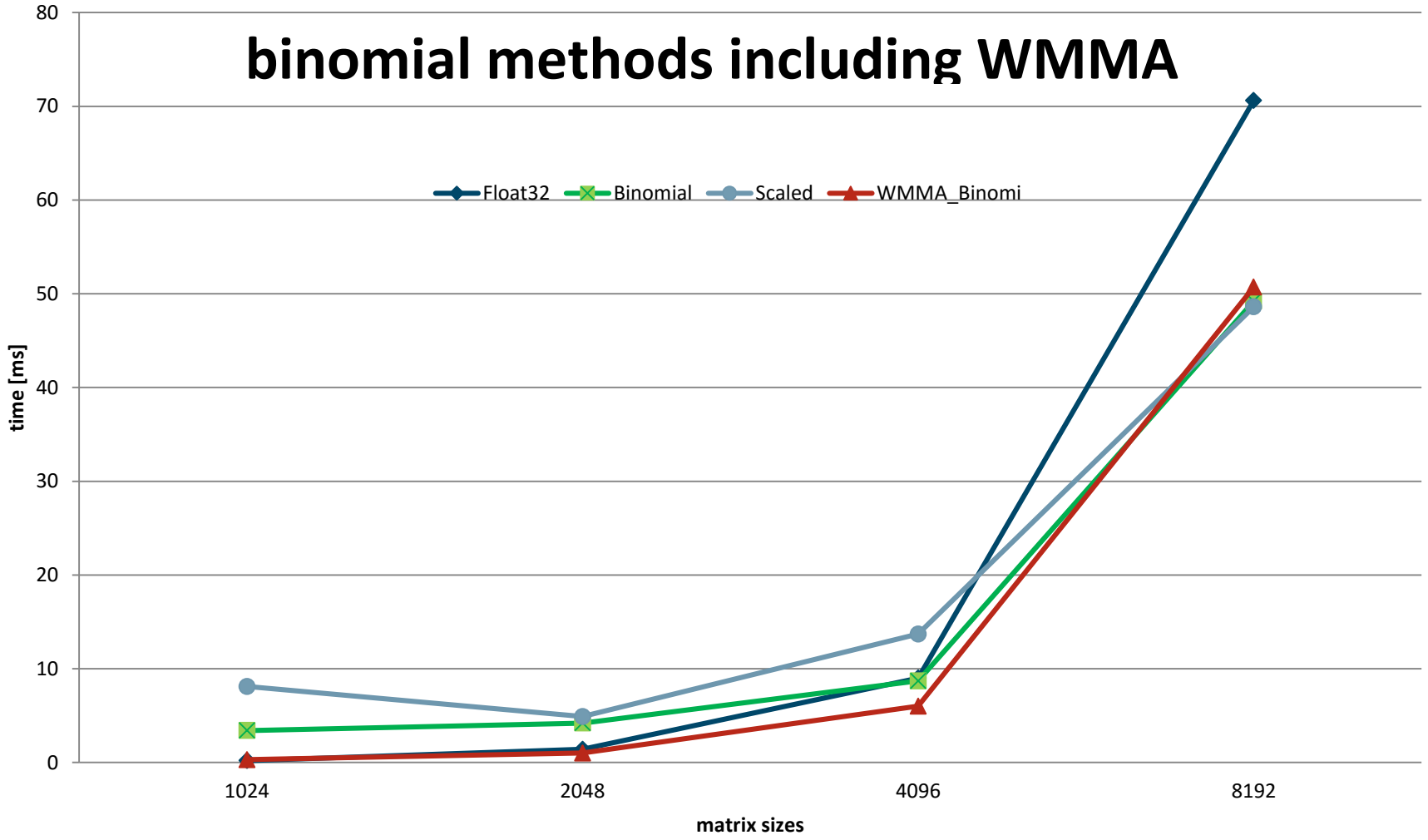
- **Usage is limited**
  - 2 types of so called fragments, FP16 matrix and FP16/FP32 accumulator
  - 1 operation possible D=AB+C
    1. is the only factor to be used
    C and D may be identical
    No accumulator for A or B
  - 1 store operation
  - No manipulation of loaded matrices
    like A=alpha*A
    like A=A+B
    - Accumulators may be manipulated in loops
      i->D.num_elements {D [i]=0.0f; }

WMMA-API

- 8 Tensor Cores/SM
- 8 warps per block for efficiency
- very effective loading of data
  - repeated loads utilize cache
  - shared memory really needed?
- documentation (#fragments) lacking
- fragments tile matrix
- additional tiling (4x4) mandatory

- hand coded version runs approx. at half performance of cublas-version

Binomi in WMMA-API



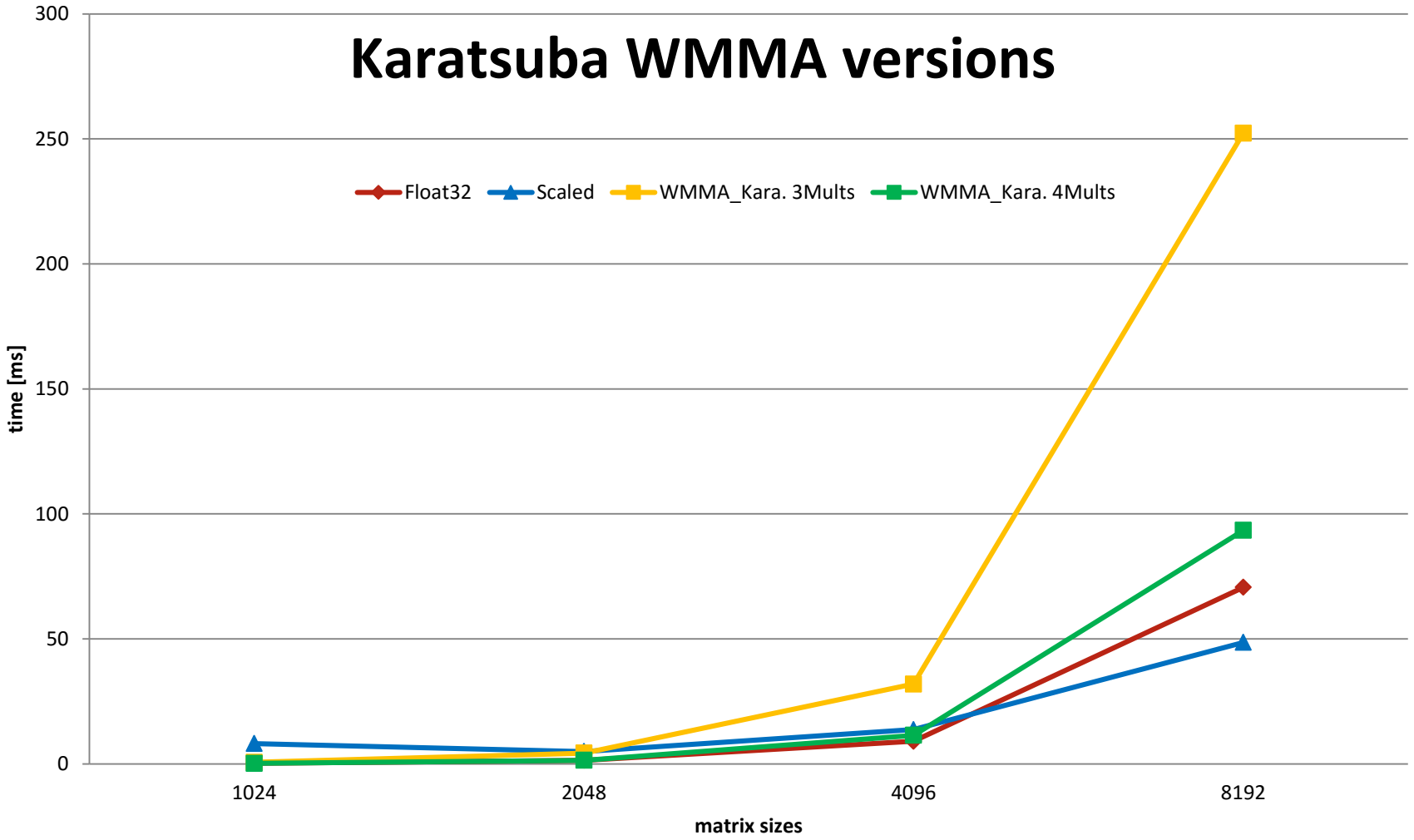
WMMA-API
Thinks to Remember

- Own API implementation is for small matrices faster than cublas-calls
- Own API implementation is for large matrices comparable to cublas-calls
  - with cublas-tricks it should be significantly faster

WMMA-Implementation Details

- 4x4 Tiles
- High and Low values of A and B matrices
- High and Low values double the amount of data
- Increases operational density - 3 MMAs per 4 loads compared to 1 MMA per 2 loads
- all binomi terms in flight accumulated to save accumulators or decrease #tiles

Karatsuba in WMMA-API



WMMA for Karatsuba Thinks to Remember

- Implementations are not competitive to float32 version
- True Karatsuba algorithm with 3 MMAs only is by far too slow
  - Tensor Cores are really fast - 1 add. MMA does not matter
  - AH+AL - operation is not feasible for fragments
  - complexity of algorithm reduces number of tiles to be used to 4x2
- Modified Karatsuba with 4 MMAs still too slow

memory issues

- **precision increasing algorithms require 32 bits for all matrix elements**
  - **reduced memory footprint in FP16 is lost**
  - **NVIDIA recommends to store network weights in FP32 anyway - but still**
- **Casting FP32 into 2 FP16 costs**
  - **few operations more**
  - **one FP16 matrix more to be moved**
  - **still one magnitude less than the MMA itself.**

Outline

- Tensor Cores - Way of Operation and Consequences
- Improving Quality of Tensor Core Usage
- Performance
  - Tensor Cores are really fast for large problems (>4096)
  - Binomial and scaled Binomi approximations faster than FP32
  - 3 Term Binomi in WMMA may close the gap further
  - WMMA limitations hamper scaled Binomi and Karatsuba
- Results and Outlook

Results

- **Tensor Core Usage requires Mixed Precision**
  - limited precision of FP16
  - range problem of FP16

- **Provided intermediate accumulation in FP32 is important**
- **Scaling values is good technique, if realized with powers of 2.**

Results

- **FP16 precision introduces rounding errors**
  rounding erros increase with
  - matrix values (scaling back and forth has no influence)
  - matrix size
- **4th digit of result has no significance**
- **Fine grain optimization of deep learning networks beyond 3rd digit for weights, biases, … is meaningless.**
- **Tensor Cores results are blind behind that point**

Results

- **Precision of tensor cores may be enhanced**

- **Split FP32 into High and Low FP16**
- **Binomial algorithm:**

  $x*y = (xH+xL)*(yH+yL) \approx xH*yH + xH*yL + xL*yH$
- **Karatsuba:**

  $x*y = xH*yH + S*((xH+xL)*(yH+yL)-xHyH-xLyL)+S(xL*yL))$
- **Scaled Binomial:**

  $x*y \approx xH*yH + S*(xH*yL + xL*yH)$

- **2-3 more siginificant digits**

Results

- cublas in Mixed Precision is incredibly fast
- WMMA-API is very restrictive

- Binomial and Scaled Binomial with cublas are faster than cublas in FP32
- Binomial algorithm in WMMA-API is faster than using cublas

Outlook

- **Binomial and Scaled Binomial algorithm in WMMA-API may be fastened up further knowing "tricks" used in cublas**

- **A new library function may be added to cublas:**
  - **Mixed precision**
  - **Tensor cores**
  - **Binomial or Scaled Binomial**
  - **Faster than FP32**
  - **Higher accuracy**

Outlook

- Deep learning algorithms may be improved
  - Highest performance, lowest accuracy at beginning
  - When precision starts to matter, shift to binomial type of algorithm

    High Performance, improved accuracy in the middle
  - Ultimative refinement in FP32

    Good Performance, best accuracy for final steps

Outlook

- I am asking for true large network examples to test and verify the above stated recommendations in collaboration.
- I am asking NVIDIA to provide cublas code to set up Binomial type of algorithms at best possible performance.
- I am asking for a better documentation of the WMMA-API.
- I am hopping that some of the restrictions in using the WMMA-API are released in the future.

Tensor Cores
Performance and Precision

Thanks

Vielen Dank