ASPEN.K2+MUBLAS: level2 CUDA BLAS kernels

Toshiyuki Imamura
RIKEN AICS

Joint work with Dr. Daichi Mukunoki (AICS)

CUDA BLAS

- Important routines for numerical linear algebra.
  - CUBLAS, MAGMA, KBLAS, CULA, etc.

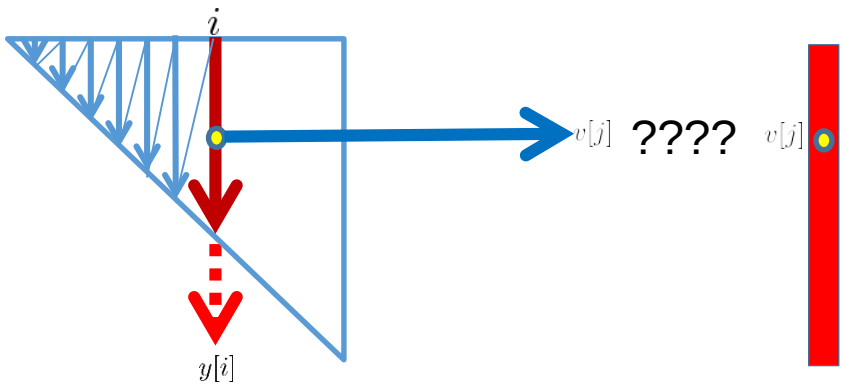
- Our projects are currently focusing on the Level 2 kernels.
  - ASPEN.K2 (by TI)
    - Mainly for SYMV (SYmmetric Matrix Vector product) kernels
    - **New algorithm** which is based on atomic operations
    - **Off-line parameter tuning** is done automatically.
    - Huge parameter space to be explored is pruned by empirical and theoretical performance models.
  - MUBLAS (by Daichi)
    - Mainly for GEMV (GEneral Matrix Vector product) kernels
    - Algorithm is straightforward, and an elaborate implementation.
    - Also, Off-line tuning is applied.
    - **Sampling free tuning**, which is based on **couple of occupancy metrics**, the best parameter is determined. (challenging work)

SYMV

Algorithm

```
!$OMP PARALLEL DO
do i=1,n
   do j=1,i-1
      y(i)=y(i)+a(j,i)*x(j)
      v(j)=w(j)+a(j,i)*x(i)
   enddo
   y(i)=y(i)+a(i,i)*x(i)
enddo
y(1:n)=y(1:n)+v(1:n)
```

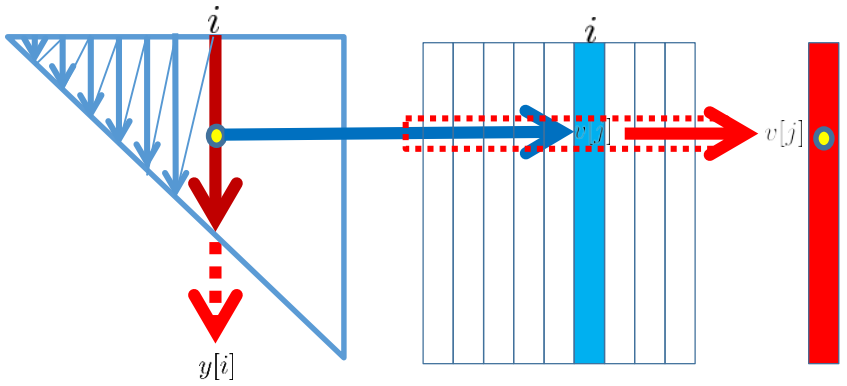
Need of exclusive control



Algorithm

```
!$OMP PARALLEL DO
do i=1,n
   do j=1,i-1
      y(i)=y(i)+a(j,i)*x(j)
      v(j)=w(j)+a(j,i)*x(i)
   enddo
   y(i)=y(i)+a(i,i)*x(i)
enddo
y(1:n)=y(1:n)+v(1:n)
```

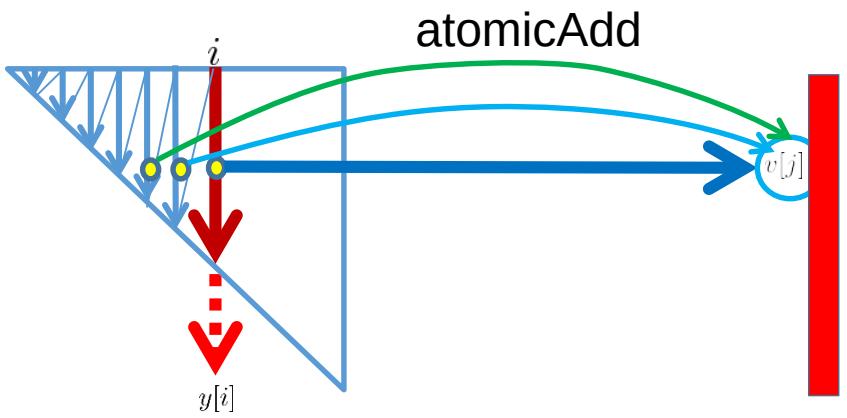
1. **Temporal set algorithm [MAGMA@SC11]:**
   Scatter and gather technique
2. **Atomic (point) algorithm [KBLAS@GTC2014]: → CUDA6**
   Use an atomic operation
3. **Atomic (section) algorithm [imamura@VECPAR 2012]:**
   Use a mutex mechanism



Algorithm

```
!$OMP PARALLEL DO
do i=1,n
   do j=1,i-1
      y(i)=y(i)+a(j,i)*x(j)
      v(j)=w(j)+a(j,i)*x(i)
   enddo
   y(i)=y(i)+a(i,i)*x(i)
enddo
y(1:n)=y(1:n)+v(1:n)
```

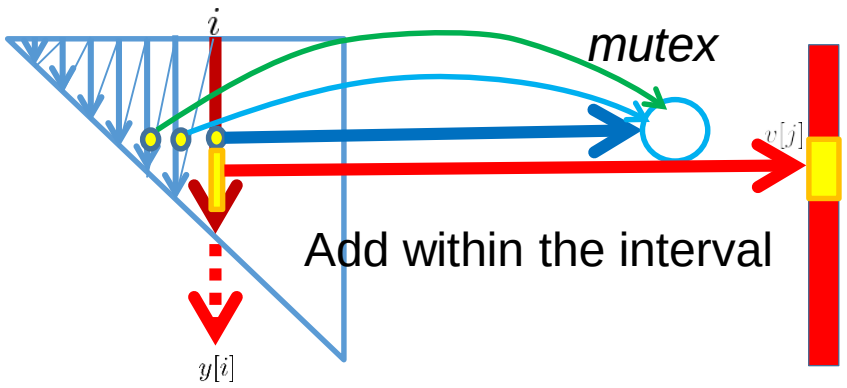
1. **Temporal set algorithm [MAGMA@SC11]:**
   Scatter and gather technique
2. **Atomic (point) algorithm [KBLAS@GTC2014]: → CUDA6**
   Use an atomic operation
3. **Atomic (section) algorithm [imamura@VECPAR 2012]:**
   Use a mutex mechanism



Algorithm

```
!$OMP PARALLEL DO
do i=1,n
   do j=1,i-1
      y(i)=y(i)+a(j,i)*x(j)
      v(j)=w(j)+a(j,i)*x(i)
   enddo
   y(i)=y(i)+a(i,i)*x(i)
enddo
y(1:n)=y(1:n)+v(1:n)
```

1. **Temporal set algorithm [MAGMA@SC11]:**
   Scatter and gather technique
2. **Atomic (point) algorithm [KBLAS@GTC2014]: → CUDA6**
   Use an atomic operation
3. **Atomic (section) algorithm [imamura@VECPAR 2012]:**
   Use a mutex mechanism



```
kernel symv_main <Tx, Ty, U, M>
  define j ≡ ȷ + threadIdx.x.
  thID := get_localID(), and
  blkID := get_blockID().
  d := (U/Ty)*threadIdx.y, i := U*blkID, and
  s := ceil(i − 1, Tx).
  Ticket := ticket.
  yreg[0] := ... := yreg[U/Ty − 1] := 0.
  // part one
  for ȷ:=0 to s − 1 step Tx
    if j < i − 1 then
      areg[k] := A(j, i + k + d),
      yreg[k] += areg[k]*x(j), and
      wreg := ∑k areg[k]*x(i + k) for k ∈ [0, U/Ty).
    endif
    get_Ticket( Ticket )
    wreg := sumup wreg through Ty.
    if j < i − 1 then
      v(j) += wreg.
    endif
    release_Ticket( Ticket ), and Ticket++.
  endfor
  // part two
  for ȷ:= thID to U do
    shm[thID][ȷ] := shm[ȷ][thID] := A(i + thID, i + ȷ).
  endfor
  synchthreads
  synchthreads
  if thID < U then
    yreg[k] := shm[thID][k]*x(i+k) for k ∈ [0, U/Ty).
  endif
  shm[k][thID] := sumup yreg[k]
                  through Tx for k ∈ [0, U/Ty).
  if thID < U then
    y(i+thID) += alpha*shm[thID][thID].
  endif
endkernel
```

```
procedure get_Ticket( int *Ticket )
  if isMasterTthread() then
    while (TRUE)
      c := atomicCAS( Ticket, blkID, −1 ).
      if c = blkID break
    endwhile
  endif
  syncthreads
endprocedure

procedure relase_Ticket( int *Ticket )
  syncthreads
  if isMasterThread() then
    atomicExch( Ticket, blkID − 1 ).
  endif
endprocedure
```

```
kernel symv_main <Tx, Ty, U, M>
  define j ≡ J + threadIdx.x.
  thID := get_localID(), and
  blkID := get_blockID(
  d := (U/Ty)*threadId
  s := ceil(i − 1, Tx).
  Ticket := ticket.
  yreg[0] := ... := yreg[U
  // part one
  for J:=0 to s − 1 step Tx
    if j < i − 1 then
      areg[k] := A(j, i + k + d),
      yreg[k]
      wreg :=
    endif
    get_Ticket( Ticket )
    wreg := sumup wreg through Ty.
    if j < i − 1 then
      v(j) += wreg.
    endif
    release_Ticket( Ticket ), and Ticket++.
  endfor
  // part two
  for J:= thI
    shm[thII
  endfor
  synchthre
```

```
  synchthreads
  if thID < U then
    yreg[k] := shm[thID][k]*x(i+k) for k ∈ [0, U/Ty).
  endif
endkernel
```

Overwrite-order is guaranteed uniquely

➔Non-deterministic result pointed on CUBLAS 6.x later can be resolved.

Get a ticket for overwrite

Decrement of the number printed on the ticket and release the ticket

```
procedure get_Ticket( int *Ticket )
  if isMasterTthread() then
    while (TRUE)
      c := atomicCAS( Ticket, blkID, −1 ).
      if c = blkID break
    endwhile
  endif
  syncthreads
endprocedure

procedure relase_Ticket( int *Ticket )
  syncthreads
  if isMasterThread() then
    atomicExch( Ticket, blkID − 1 ).
  endif
endprocedure
```

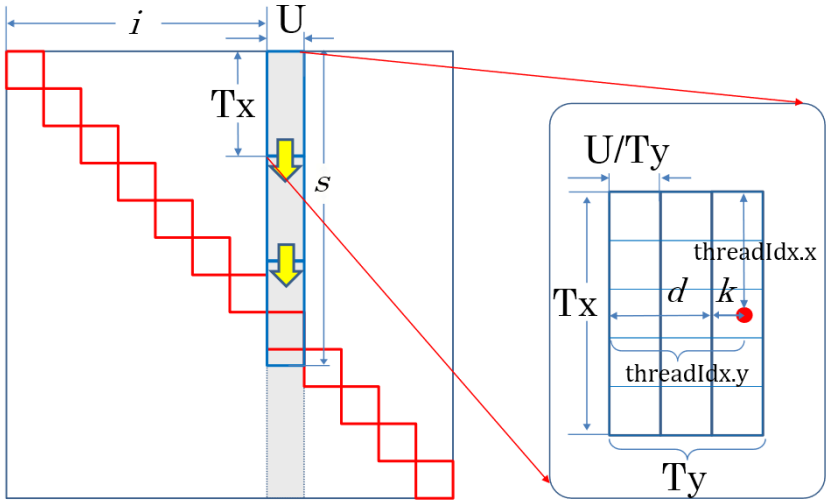
Only the block whose blkID matches.

Decrement and give a right to next block

Optimizable Parameters

- <int Tx,int Ty> :: 2D-Thread block shape
- <int U> :: Panel shape is <Tx,U> && U%Ty=0
- <int M> :: Access order

```
do i0=0,n-1, U :: block
  do j0=0,i0-1, Tx
    do i_=0, Ty-1
    do j_=0, Tx-1; j=j0+j_
    do k_=0,U/Ty-1; i=i0+i_*U/Ty+k_
      y(i)=y(i)  +a(j,i)*x(j)
      w(j)=w(j)+a(j,i)*x(i)
    enddo
    enddo
    enddo
  enddo
enddo
```



Automatic Parameter Tuning DSYMV-U

- Reduction of the Sampling cost for DSYMV-U
  - **Parameter := (Tx, Ty, U, M)**

Tx	ThreadBlock.x	{32,64,…288/320}	10
Ty	ThreadBlock.y	{1,2,…,8}	8
U	Panel/Ty	U/Ty={3,4,…,32}	28
M	Stream Order	{1,2,…,10}	10
		**| {parameter space}	→**	22400

3-step screening

step 1 :: fix M and restrict others to maximize Tx*Ty*U with 96≦Tx*Ty≦320

step 2 :: top20's from step1 varying M

step 3 :: top20's from step2 and finer sampling for N (#1 and #2 step are coarse)

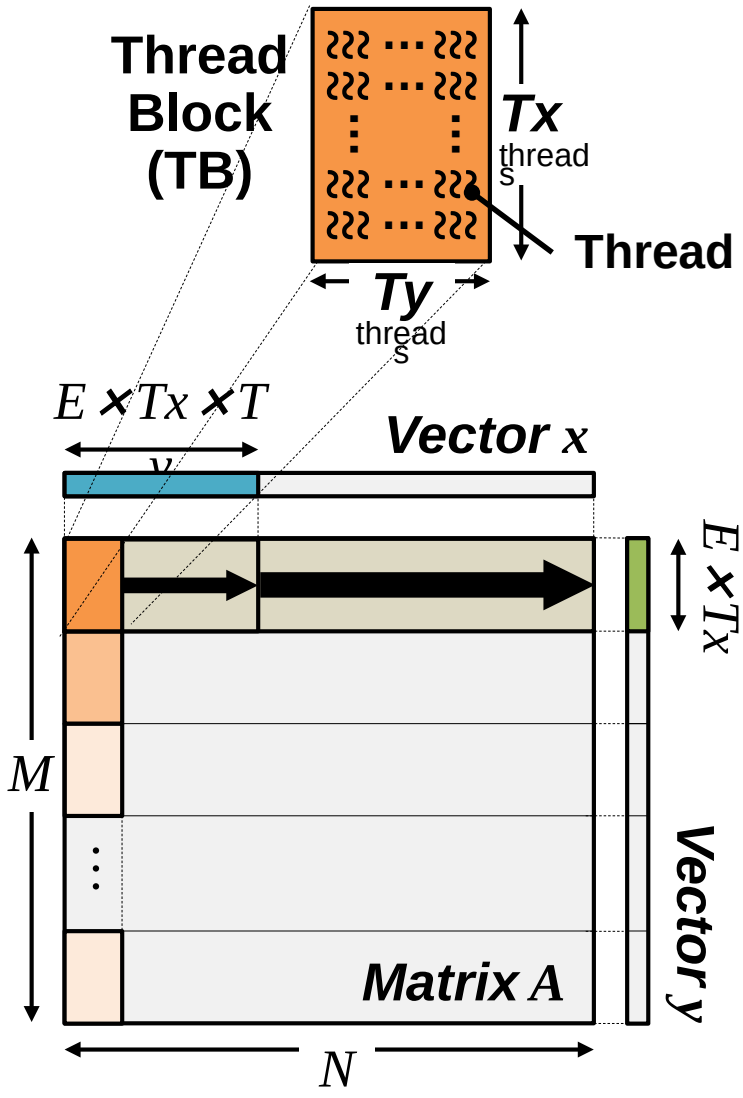
	GTX580 (Fermi)	K20c (Kepler)	Titan Black (Kepler)	GTX750Ti (Maxwell)
Sampling 1	33	25	25	33
Sampling 2	163	114	114	115
Sampling 3	9	13	14	19
Total time for tuning	1:17 14:41-15:58	1:15 17:17-18:32	1:19 17:18-18:37	1:28 17:16-18:44

GEMV

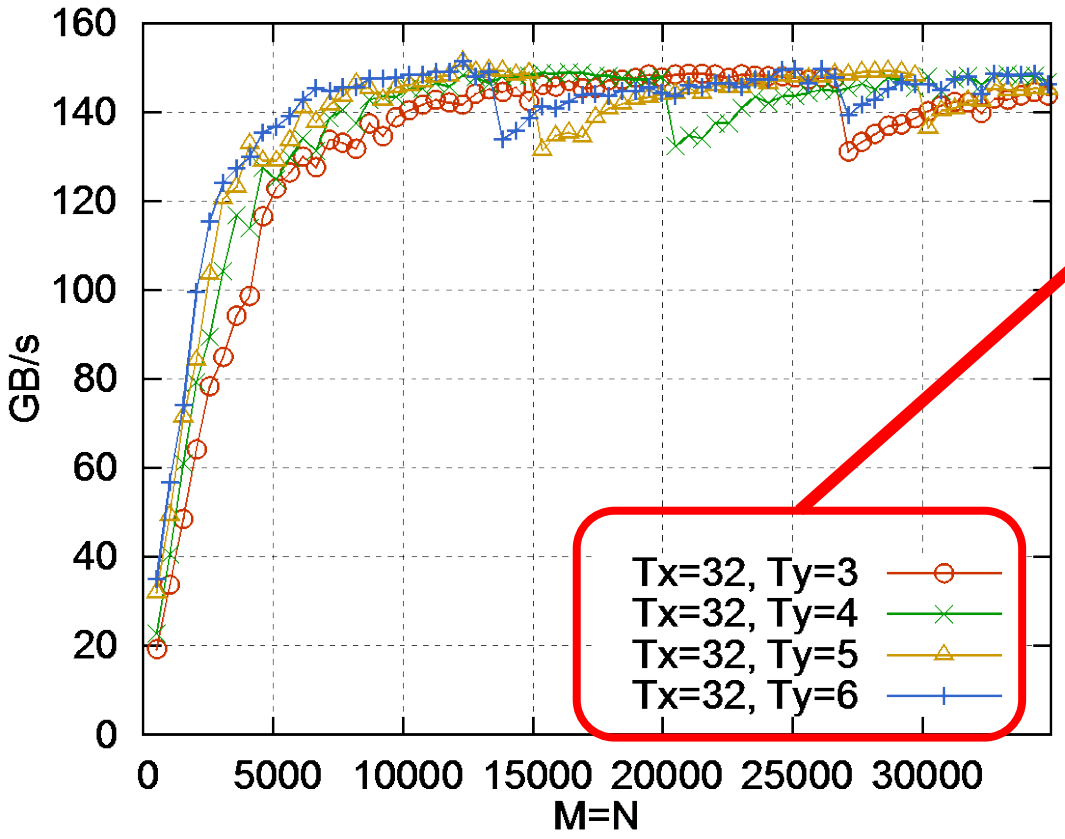
Implementation

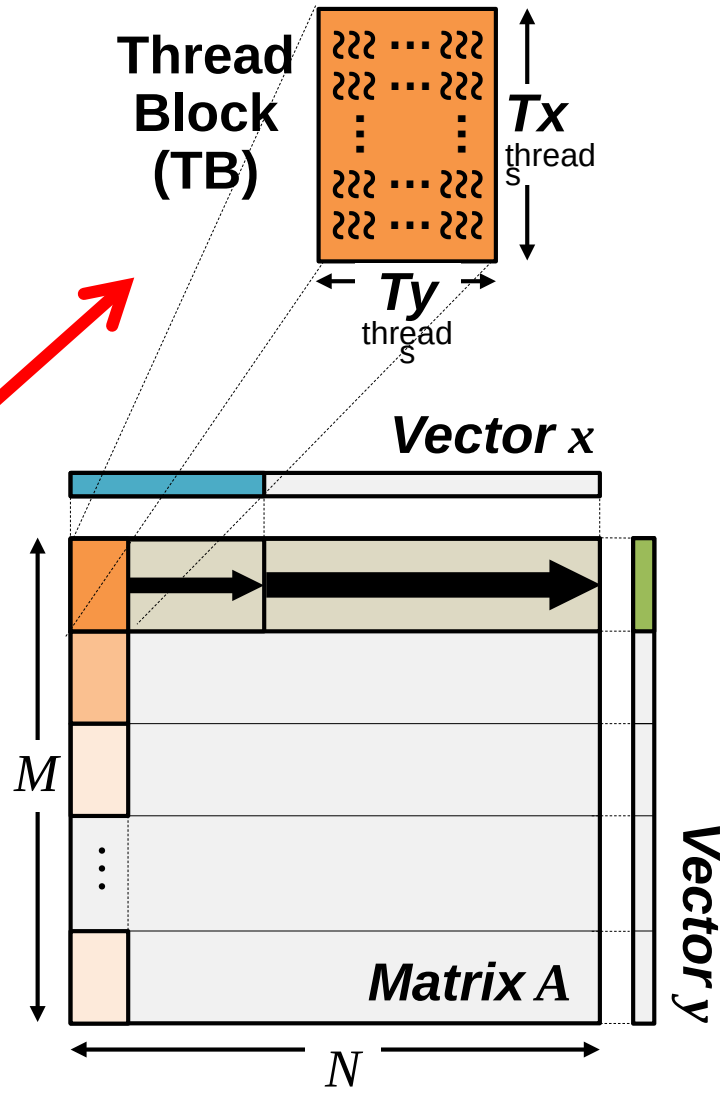
- **Kernel design**
  - 2-dimentional Thread-Block (TB)
    - TB-size: Tx × Ty threads
    - For high warp-occupancy
    - Final result (vector y) is computed with reduction among Ty-threads
  - Load matrix A through read-only data cache
  - Shared-memory blocking for vector x
  - Register blocking for vector x with 128-bit vector load
    - 1 thread accesses 128bit-width elements
    - e.g. float4 (E=4) or double2 (E=2)

      E: vector length



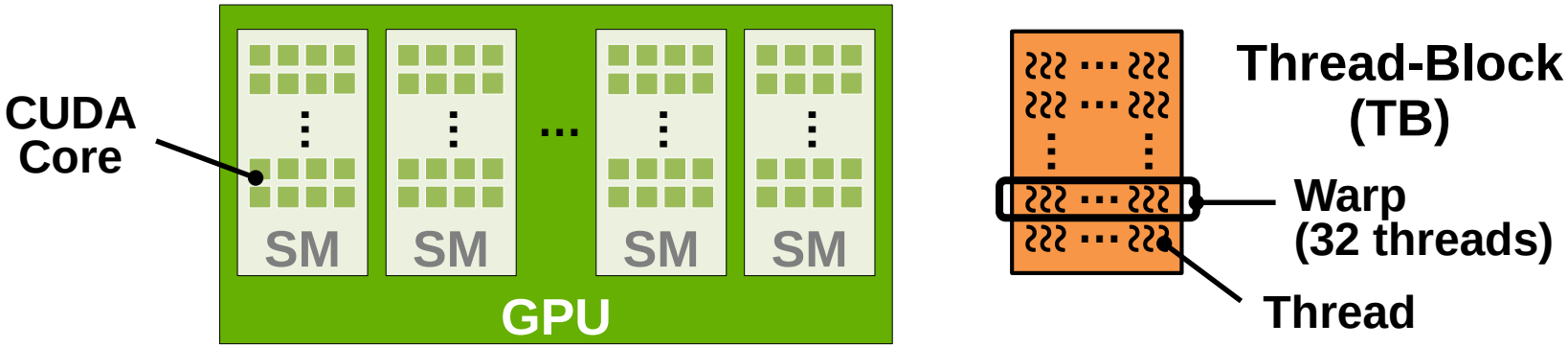
Pre Performance Evaluation





Block-Occupancy (1/4)

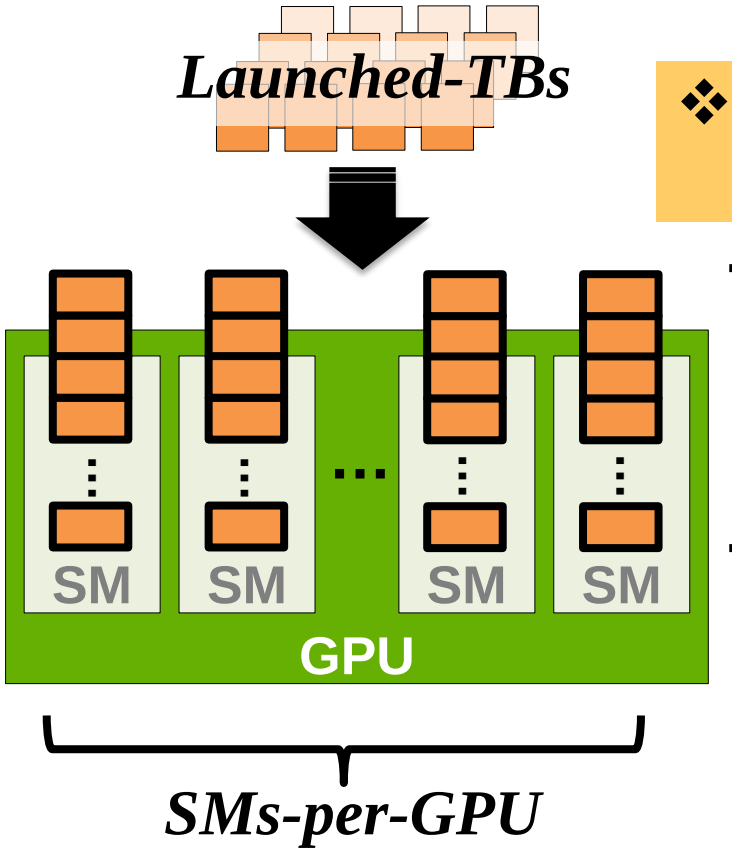
- **Warp-occupancy**
  - Occupancy of *warps* on a *Streaming Multiprocessor (SM)*
    - Percentage of active (running) *warps* per max executable *warps* concurrently on an *SM*
  - One of the metrics of hardware utilization at an *SM* level
- **Block-occupancy** (our proposal)
  - Occupancy of *Thread-Blocks (TBs)* on all *SMs* on a GPU
  - One of the metrics of hardware utilization at a GPU level



Block-Occupancy (2/4)

$$\textit{Block-Occupancy} = \frac{\textit{Launched-TBs}}{\textbf{ceil}\ (\textit{Launched-TBs},\ \textit{Max-TBs-per-GPU})}$$

†ceil(x,y) rounds x up to the nearest multiple of y



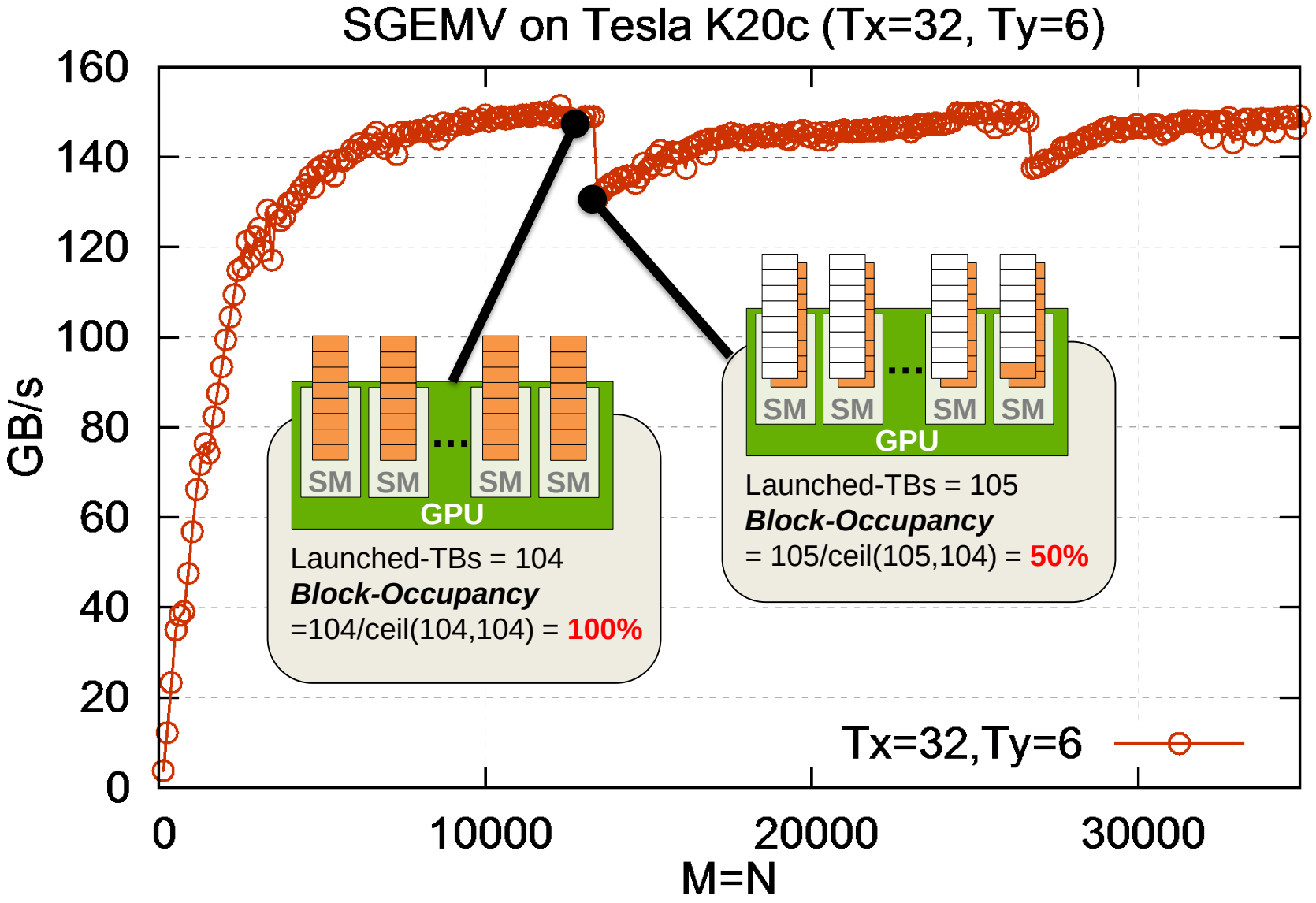
❖ A GPU executes ***Max-TBs-per-GPU*** TBs concurrently

❖ ***Launched-TBs*** should be multiples of ***Max-TBs-per-GPU***

Max-TBs-per-SM

- The number depends on:
  - Register and shared-memory usage
  - TB size
  - Physical max (Fermi=8, Kepler=16, Maxwel=32)

Block-Occupancy (3/4)



Automatic TB Size Adjustment (2/3)

Implementation

- Block-occupancy is calculated for every possible combinations of Tx&Ty
- All parameters required to determine Tx & Ty can be automatically collected
  - Register usage: `cudaFuncGetAttributes()`
  - # of SMs per GPU: `cudaGetDeviceProperties()`
    - Only this function is called in init-function before first use of our library like cublasCreate()
- The determining process is performed in GEMV routine every time before kernel launch

Algorithm 1 Determining Tx_B and Ty_B

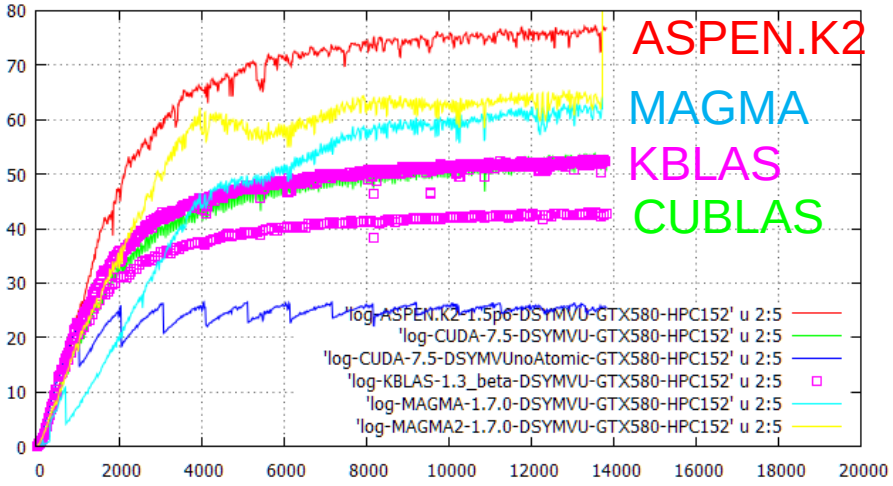
1: $Tx_{B_{min}} = 8$
2: $Tx_{B_{max}} = 1024$
3: $Th_{B_{min}} = 128$
4: $Th_{B_{max}} = 1024$
5: $B_{M_{min}} = 3$
6:
7: $B_{D_{min}} = M_D \times B_{M_{min}}$
8: $O_{tmp} = 0$
9: $tx = Tx_{B_{min}}$
10: **do**
11: $Ty_{B_{min}}$ =ceil $(Th_{B_{min}}/tx, 1)$
12: $Ty_{B_{max}}$ =max $(Ty_{B_{min}},$
13: min (floor $(Th_{B_{max}}/tx, 1)$, $tx))$
14: $ty = Ty_{B_{max}}$
15: **while** $ty \geq Ty_{B_{min}}$ **do**
16: O_B =GetOB (tx, ty)
17: $B_{M_{max}}$ =GetBMmax (tx, ty)
18: **if** $(B_{M_{max}} \geq B_{M_{min}})$ and $(O_{tmp} \leq O_B)$ **then**
19: $O_{tmp} = O_B$
20: $Tx_B = tx$
21: $Ty_B = ty$
22: **end if**
23: $ty = ty - 1$
24: **end while**
25: $tx = tx + Tx_{B_{min}}$
26: B_D =GetBD (tx)
27: **while** $(B_D \geq B_{D_{min}})$ and $(tx \leq Tx_{B_{max}})$

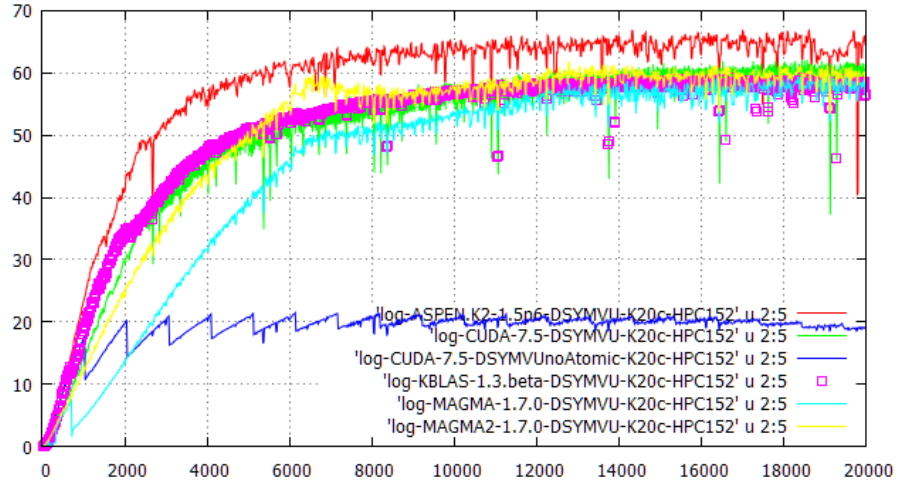
PERFORMANCE

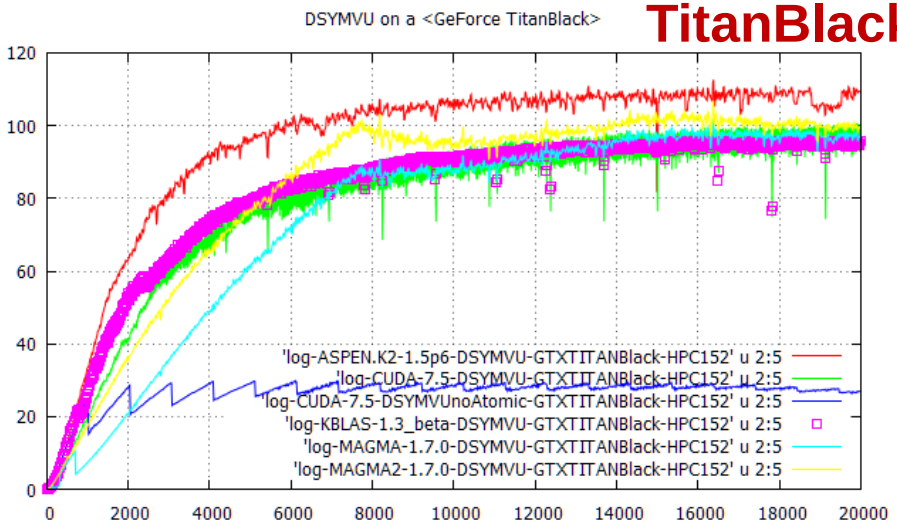
SYMV's on slides with RED bars

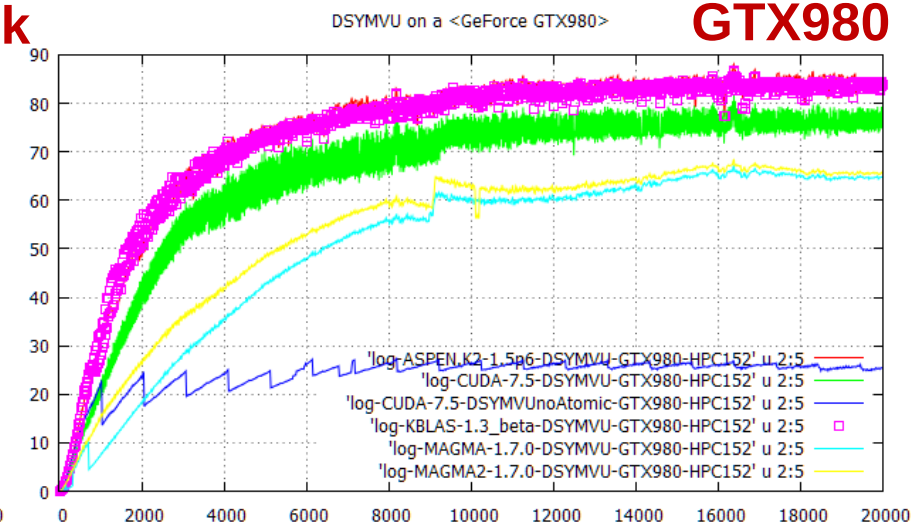
GEMV's on slides with BLUE bars

DSYMVU





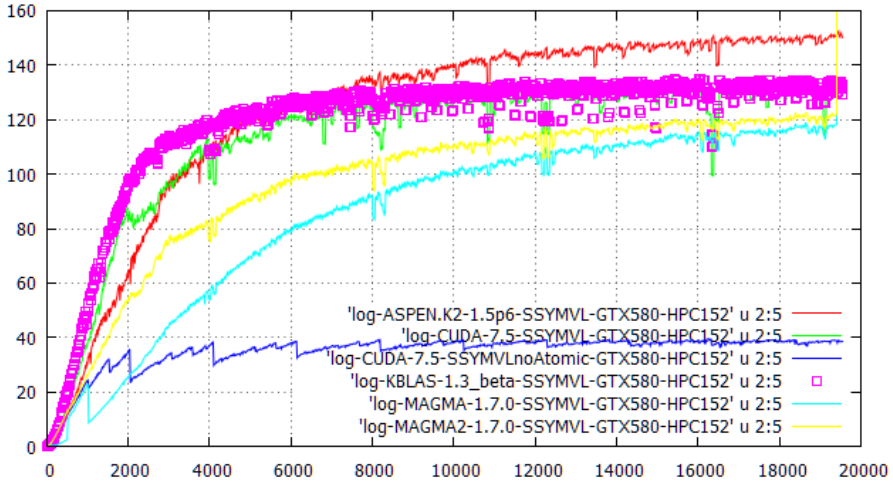




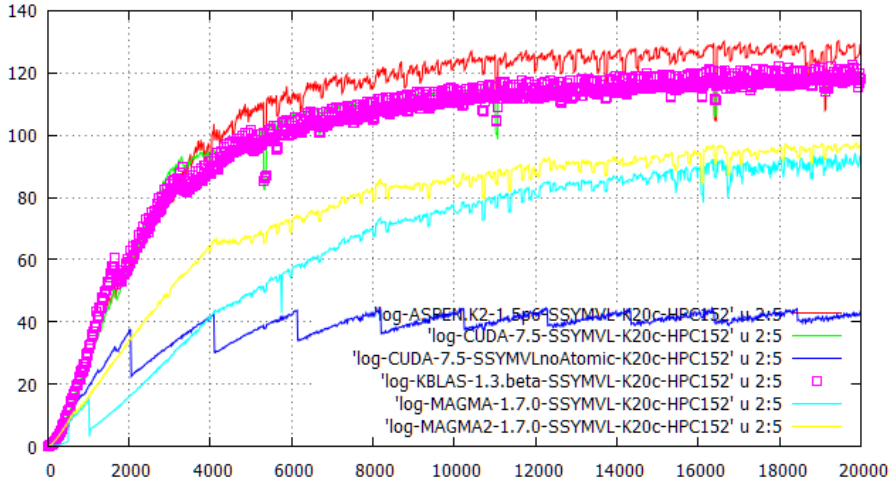
[GFLOPS]

[matrix dimension]

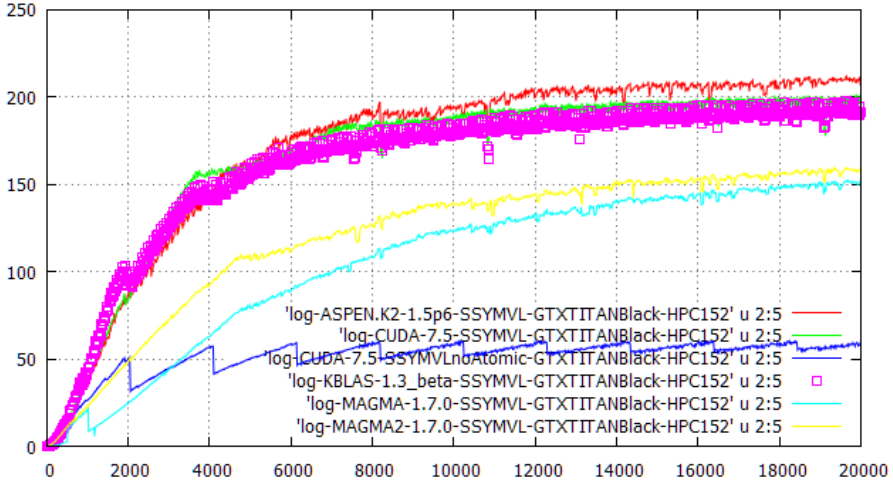
SSYMVL



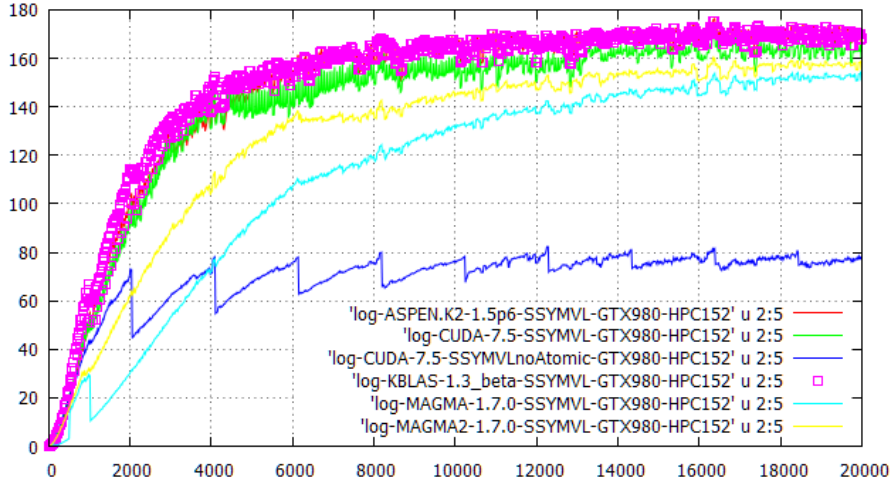
GTX580



TeslaK20c



TitanBlack

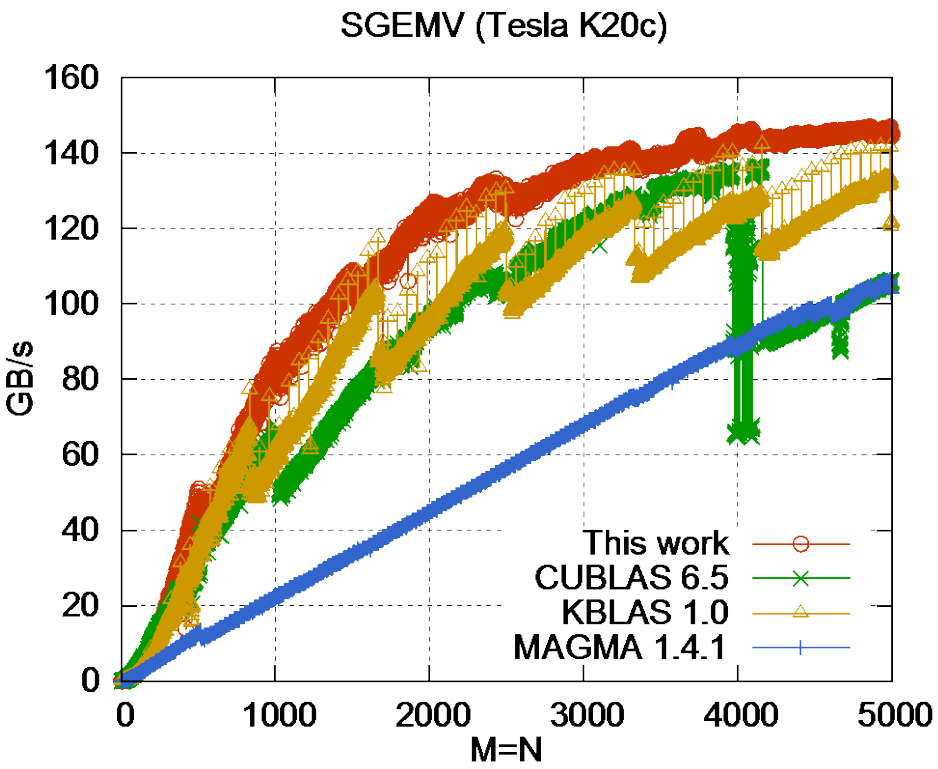


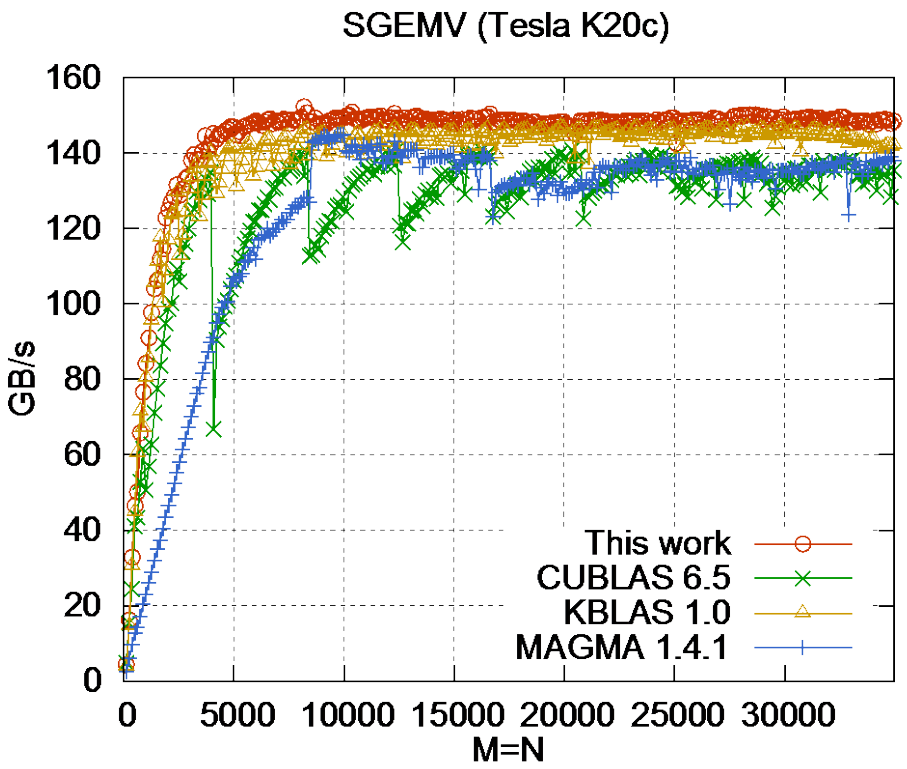
GTX980

[GFLOPS]

[matrix dimension]

SGEMV (Tesla K20c)

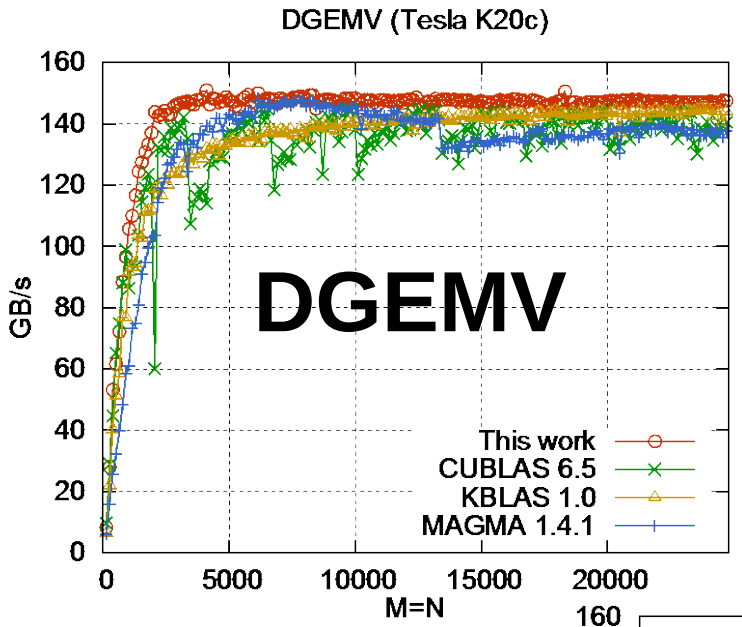


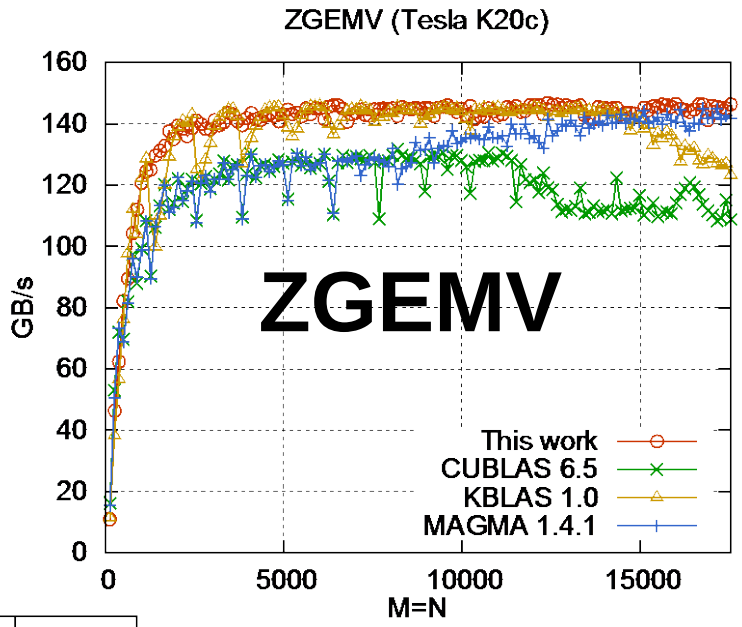


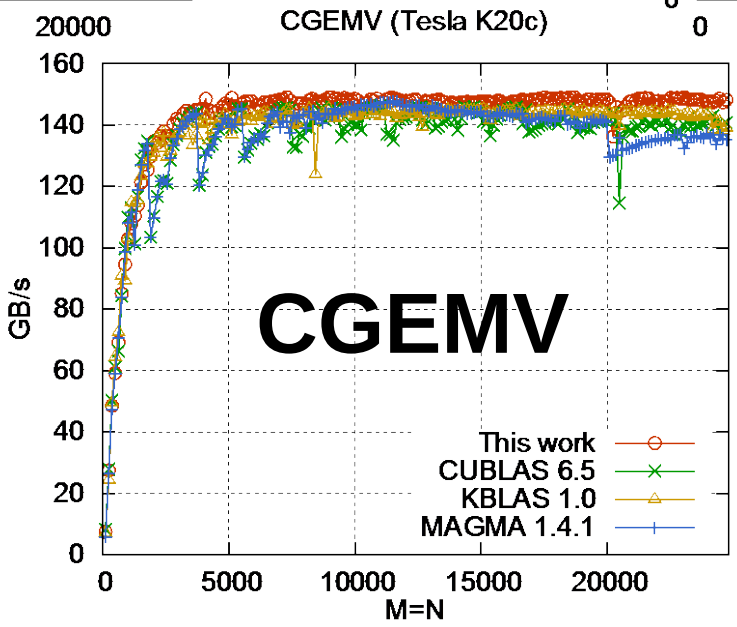
M=N=1~5000
(step=1)

M=N=1~34944
(step=128)

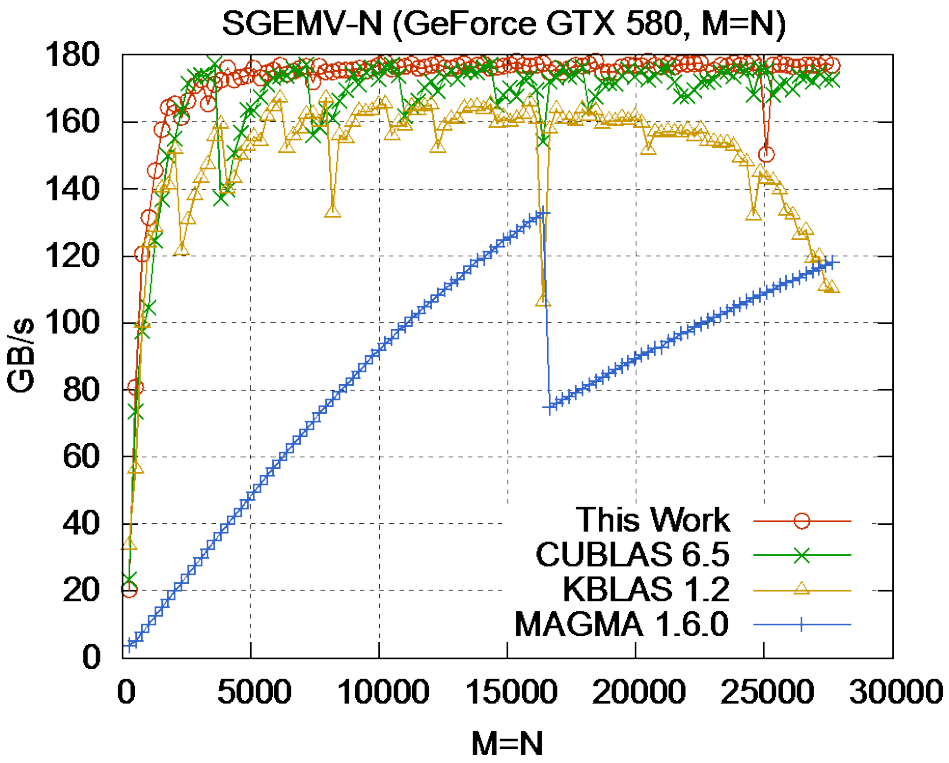
Double & Complex (Tesla K20c)





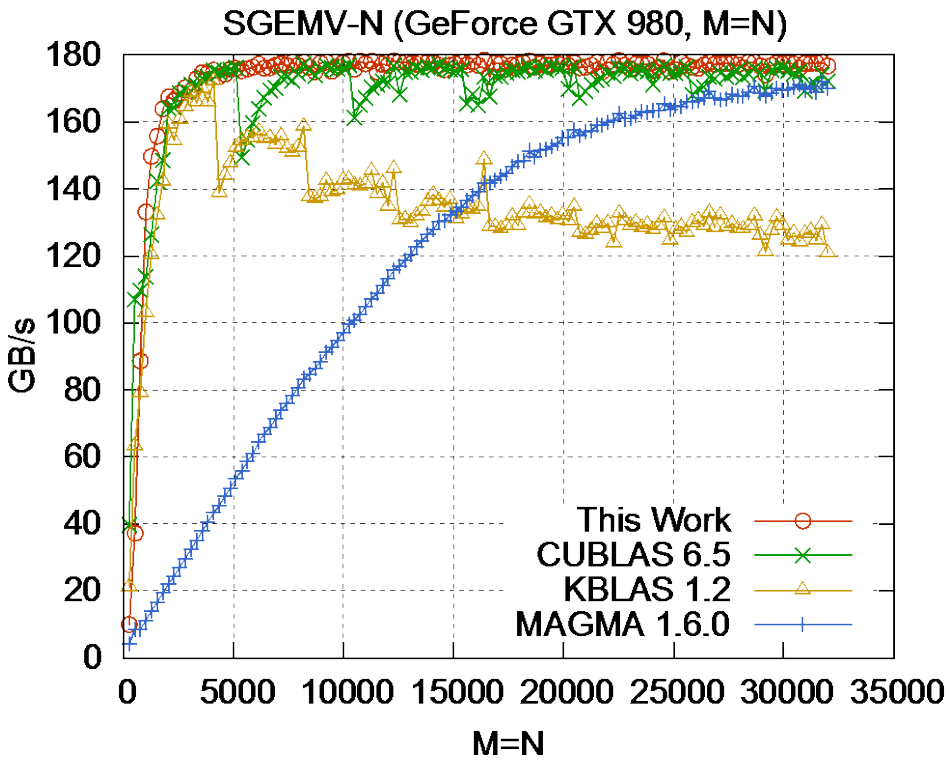


Fermi & Maxwell



Fermi

(GeForce GTX 580)



Maxwell

(GeForce GTX 980)

CONCLUSION

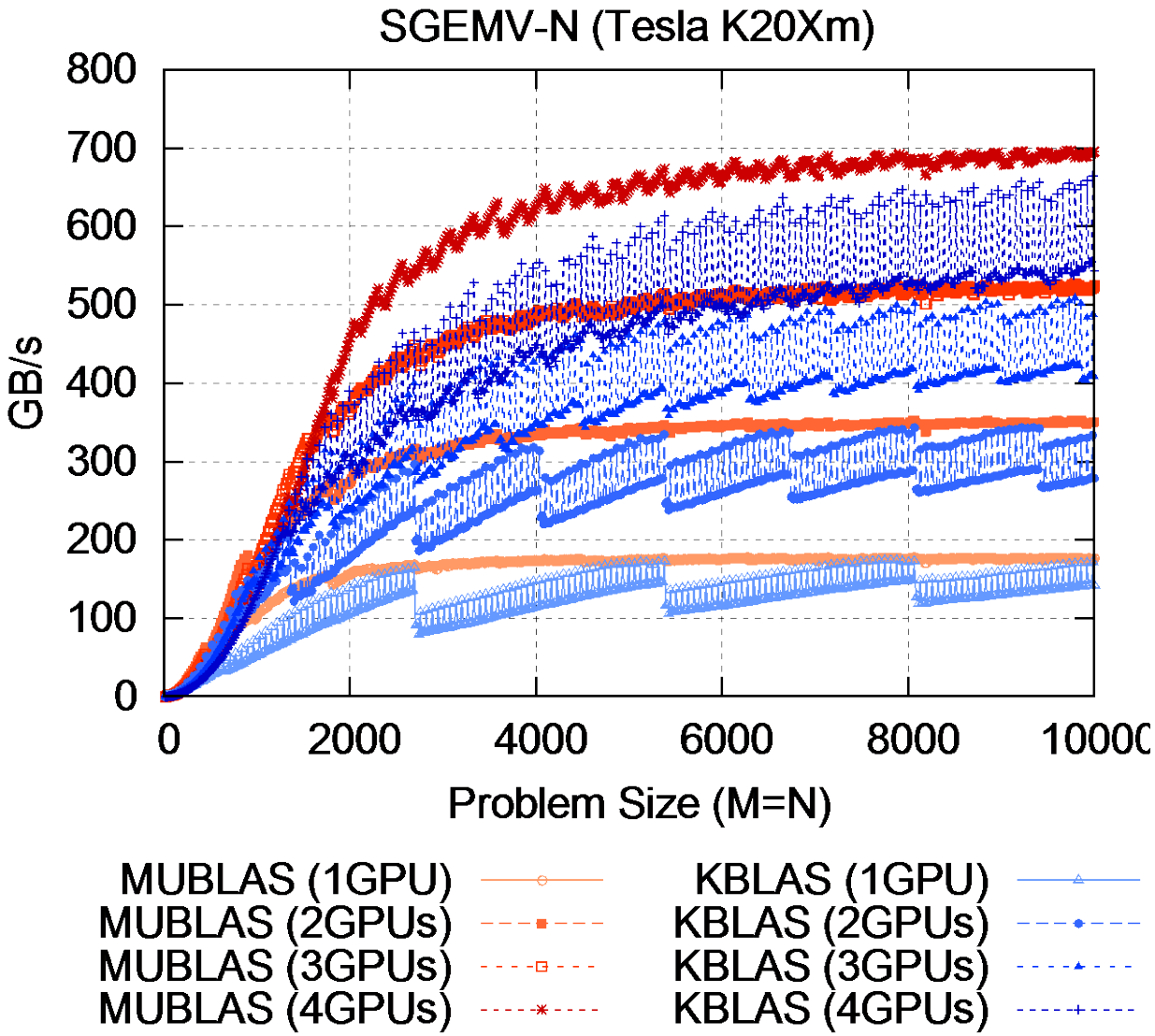
Conclusion

- **Overview**
  - Fast implementation of SYMV and GEMV on NVIDIA GPUs with auto-tuning
  - "*Atomic-operation*"-based implementation for SYMV
  - Performance stabilization with automatic TB size adjustment based on "*Block-Occupancy*" for GEMV
- **Result**
  - Better performance in terms of throughput and stability with respect to the matrix size
  - Keep nearly peak performance on…
    - Any matrix size (except for small matrix)
    - Single & double precision on real (optionally complex)
    - Multiple GPU products/generations (Fermi, Kepler, Maxwell)
  - Sampling-base tuning ← SYMV
  - Trial-execution free tuning ← GEMV

Download

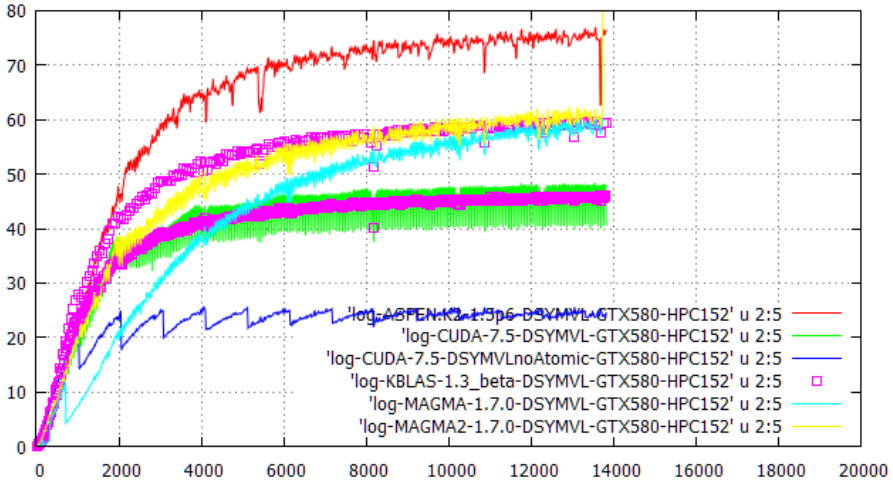
- **ASPEN.K2-SYMV and MUBLAS-GEMV**
  - **http://www.aics.riken.jp/labs/lpnctrt/ASPENK2.html**
  - ✓ SYMV-[UL], GEMV-[NT]
  - ✓ Fermi, Kepler, Maxwell
  - ✓ Single, Double (Double-Double (DD) precision for GEMV)
  - ✓ Real (Complex for GEMV)
  - ✓ Multi-GPU version is planed

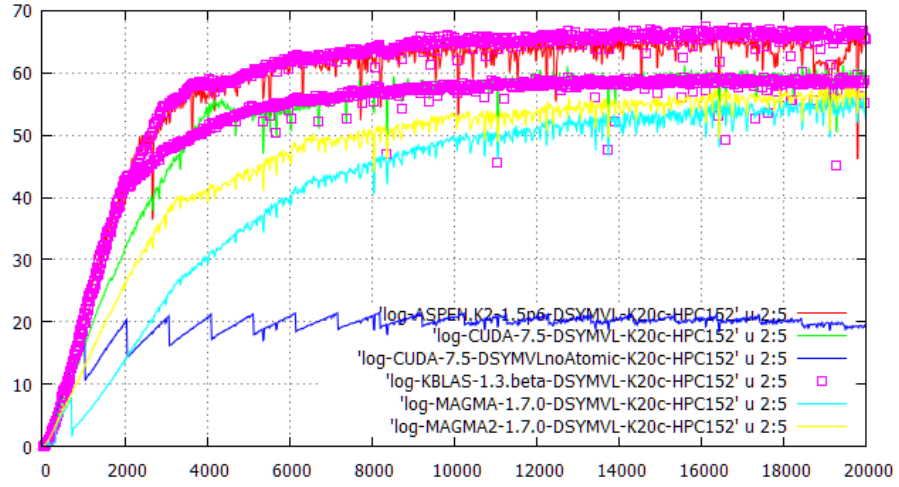
THANK YOU

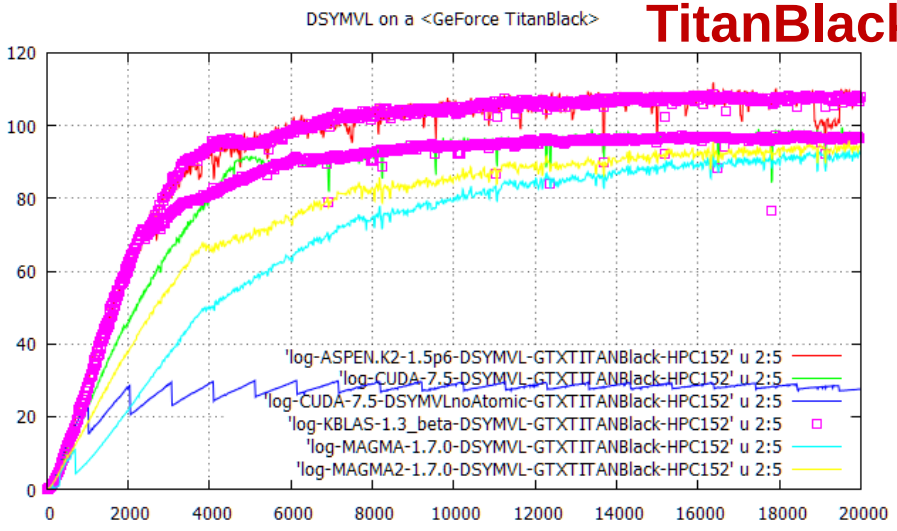


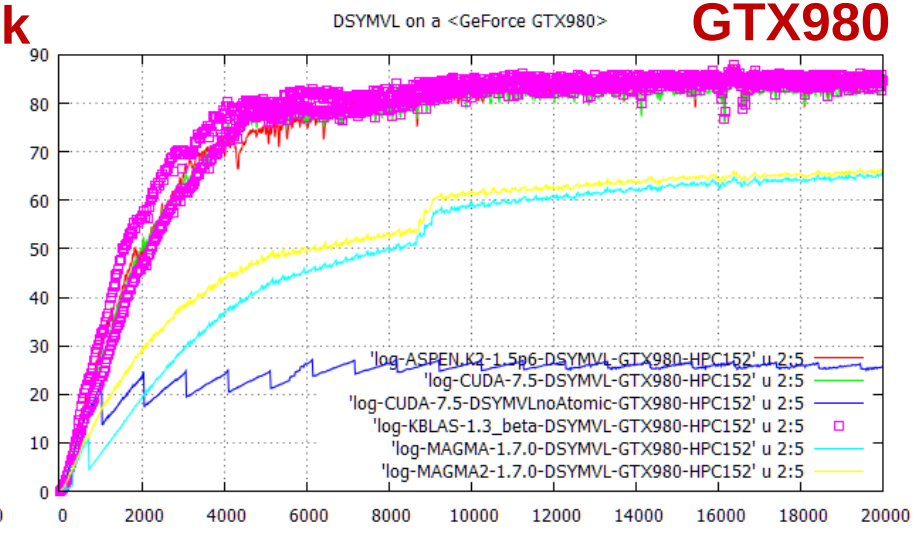
MUBLAS 1.5.18, KBLAS 1.3-beta, HOKUSAI GreatWave, CUDA7.0

DSYMVL





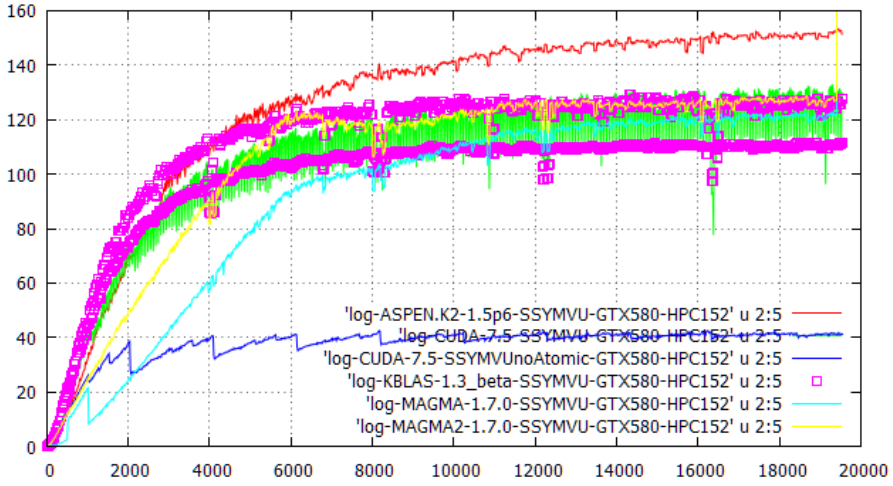


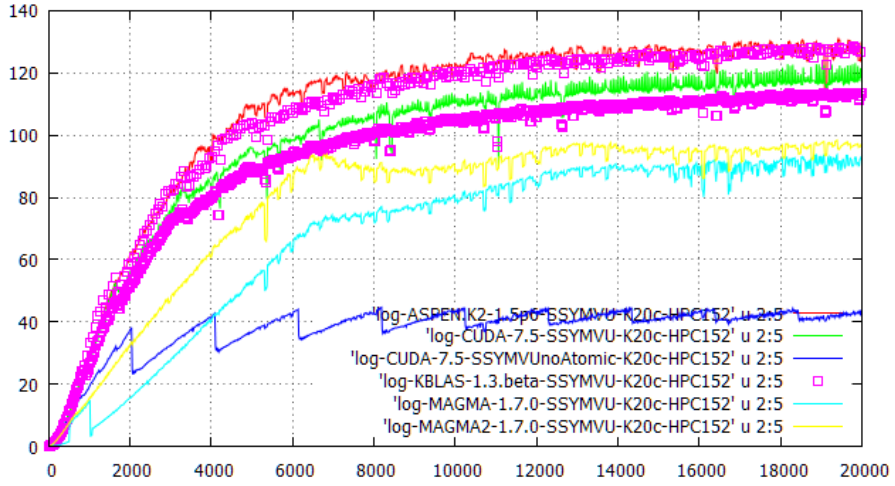


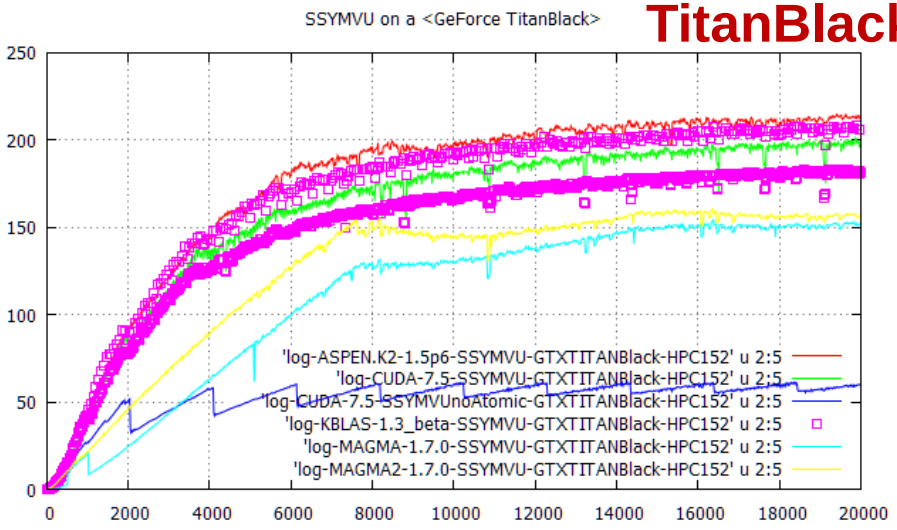
[GFLOPS]

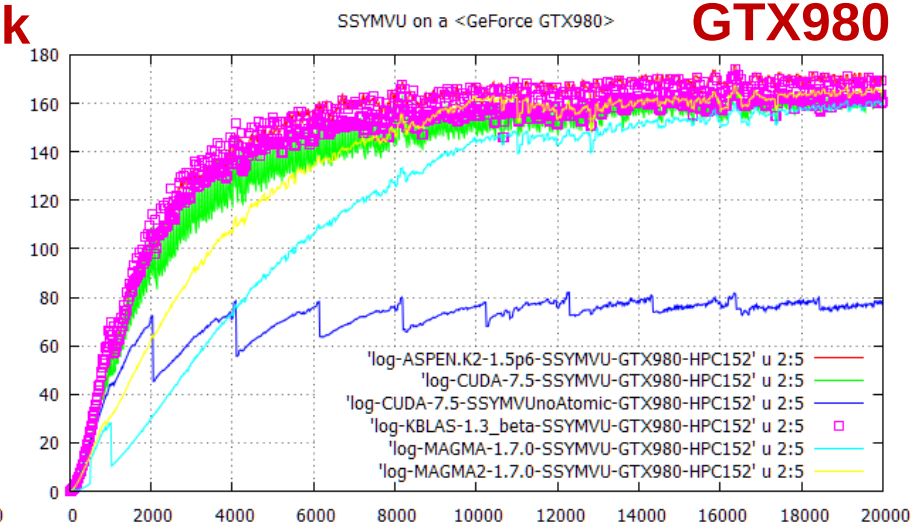
[matrix dimension]

SSYMVU









[GFLOPS]

[matrix dimension]