LibSci for accelerators - libsci_acc

Adrian Tate
XK6 / openACC workshop
Manno, Mar6-7 2012

Contents

Libsci_acc

- Provide basic scientific libraries optimized for hybrid CPU and accelerator systems (XK6)
- Independent to, but fully compatible with openACC
- Designed to augment the existing choices (MAGMA, CUBLAS, CULA)
- Dual goal :

1. Base performance of GPU with minimal (or no) code change

   libsci_acc simple interface

2. Advanced performance of the GPU with controls for data movement

   libsci_acc device interface

does not imply that always need expert interfaces to get great performance

Simple interface

- Supports the standard API in original form
- Will perform all GPU dirty-work for you
  - Initialize data structures on GPU
  - Split your problem into a CPU portion and GPU portion
  - Copy data to the GPU memory from CPU memory
  - Perform GPU and CPU operations
  - Copy data back to CPU memory
- Library-heavy codes can use GPUs with no code change
- Is not only a tool for simple usage
  - If you don’t need the data on GPU afterwards, use the simple interface
- Simple API has automatic adaptation

Adaptation in Simple interface

- You can pass either host pointers or device pointers to simple interface
- A is host memory

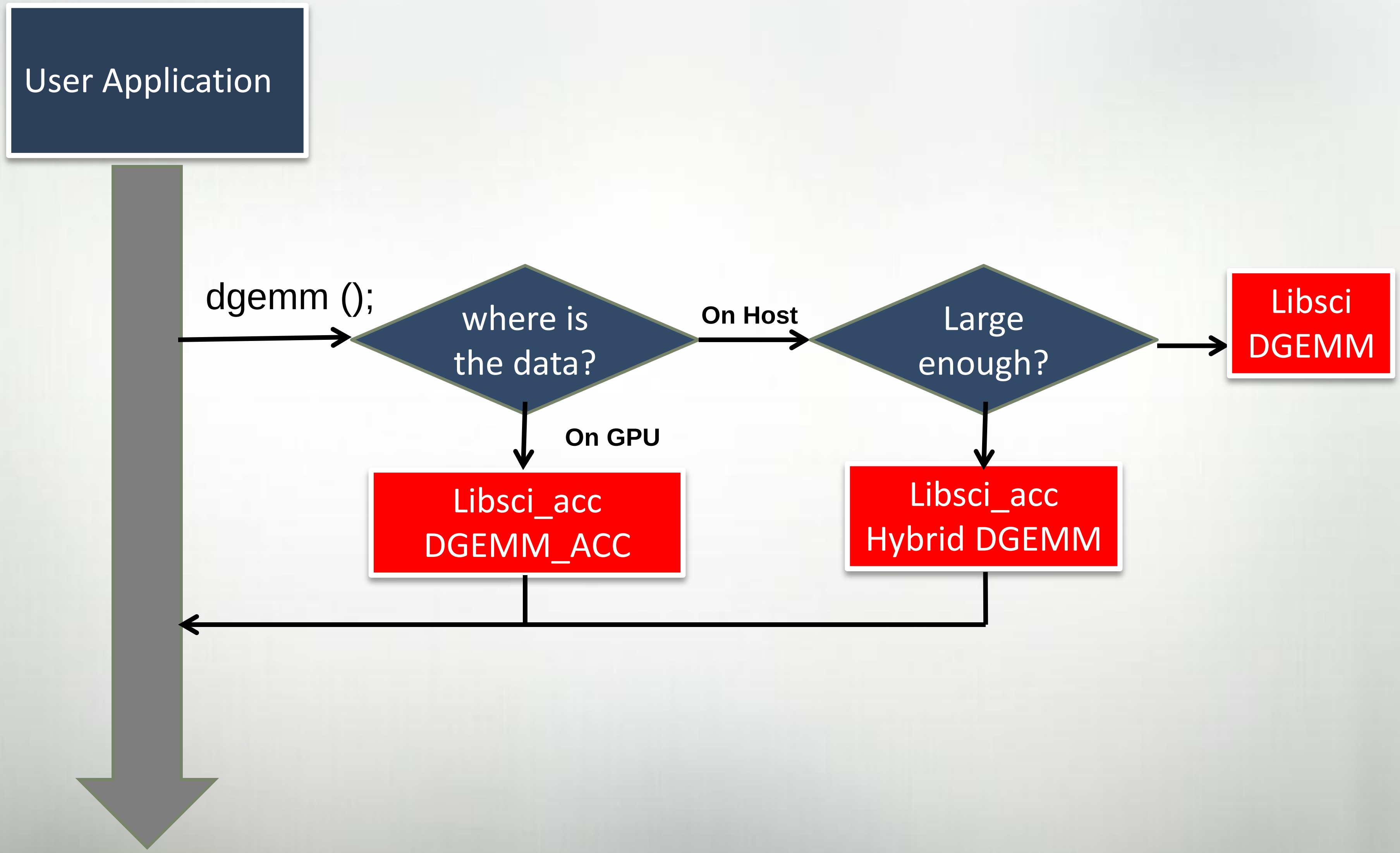
```
dgetrf(M, N, A, lda, ipiv, &info)
```

  - if problem is too small, performs host operation
  - Otherwise, performs hybrid LU operation on CPU and GPU
- Pass Device memory

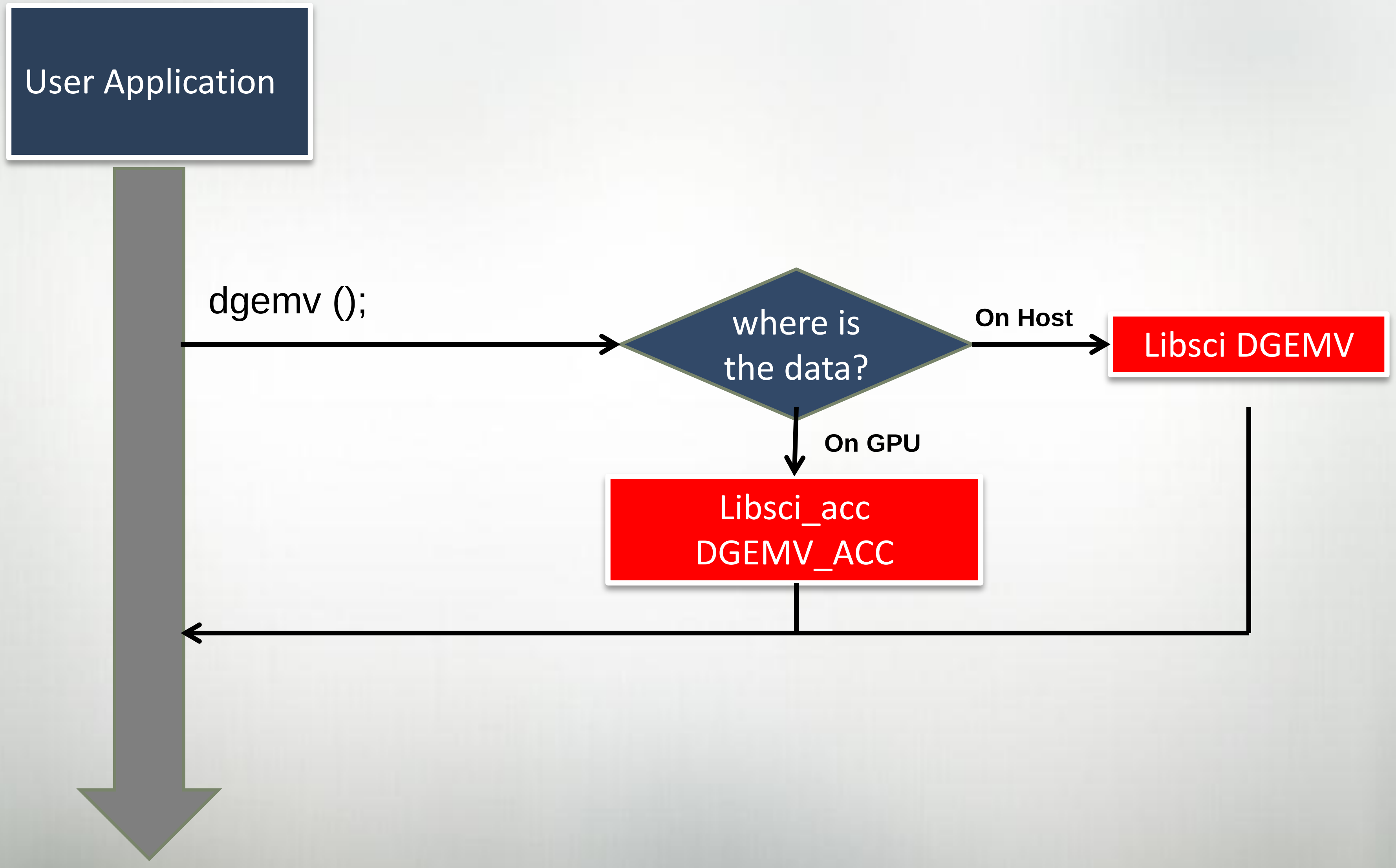
```
dgetrf(M, N, d_A, lda, ipiv, &info)
```

- Performace LU on the device

Libsci_acc: Simple Interface for BLAS3 and LAPACK



Libsci_acc: Simple Interface for BLAS1 and BLAS2



Device interface

- Device interface gives higher degrees of control
- Requires that you have already copied your data to the device memory
- API
  - Every routine in libsci has a version with _acc suffix
  - E.g. dgetrf_acc
  - This resembles standard API except for the suffix and the device pointers

CPU interface

- Sometimes apps may want to force ops on the CPU
  - Need to preserve GPU memory
  - Want to perform something in parallel
  - Don’t want to incur transfer cost for a small op
- can force any operation to occur on CPU with _cpu version
- Every routine has a _cpu entry-point
- API is exactly standard otherwise

Targets – As of February 2012

- BLAS
  - **[s,d,c,z]GEMM**
  - **[s,d,c,z]TRSM**
- LAPACK
  - **[d,z]GETRF**
  - **[d,z]GETRS**
  - **[d,z]POTRF**
  - **[d,z]POTRS**

Key - **HYBRID**

Contents – As of April 2012

- BLAS
- [s,d,c,z]GEMM
- [s,d,c,z]TRSM
- **[z,c]HEMM**
- **[s,d]SYMM**
- **[s,d,c,z]SYRK**
- **[z,d]HERK**
- **[s,d,c,z]SYR2K**
- **[s,d,c,z]TRMM**
- **ALL level 2 BLAS**
- **All level 1 BLAS**

- LAPACK
  - [d,z]GETRF
  - [d,z]GETRS
  - [d,z]POTRF
  - [d,z]POTRS
  - **[d,z]GESDD**
  - **[d,z]GESDD**
  - **[d,z]GEBRD**
  - **[d,z]GEQRF**
  - **[d,z]GELQF**

Key - NEW

Targets as of April 2012

- BLAS
  - [s,d,c,z]GEMM
  - [s,d,c,z]TRSM
  - [z,c]HEMM
  - [s,d]SYMM
  - [s,d,c,z]SYRK
  - [z,d]HERK
  - [s,d,c,z]SYR2K
  - [s,d,c,z]TRMM
  - ALL level 2 BLAS
  - All level 1 BLAS

Host pointers run on the cpu

- LAPACK
  - [d,z]GETRF
  - [d,z]GETRS
  - [d,z]POTRF
  - [d,z]POTRS
  - [d,z]GESDD
  - [d,z]GEBRD
  - [d,z]GEQRF
  - [d,z]GELQF

AUTOTUNED-HYBRID HYBRID Simple, device and CPU

Optimization in libsci_acc

- Cray Autotuning framework has been built to tune all BLAS for accelerators
  - GPU kernel codes are built using code generator
  - Enormous offline autotuning is used to build a map of performance to input
  - An adaptive library is built from the results of the autotuning
  - At run-time, your code is mapped to training set of input
  - Best kernel for your problem is used
- All the BLAS and LAPACK schemes have been rebuilt using a block-asynchronous methodology
  - Partition matrix for CPU and GPU
  - Re-block original host matrix, send part of data to device
  - Begin computation on device, and simultaneous bring more data
  - Continue, and fine tune so that the whole transfer is hidden

Usage - Basics

- Supports Cray, GNU and PGI compilers.
- Fortran and C interfaces (column-major assumed)
  - Load the module ***craype-accel-nvidia20.***
  - Compile as normal (dynamic libraries will be used)
  - To enable threading in the CPU library, set OMP_NUM_THREADS
    - E.g. export OMP_NUM_THREADS=16
  - Assign 1 single MPI process per node
    - Multiple processes cannot share the single GPU
  - Execute your code as normal

Device Interface example - dgetrf_acc

- Use existing methods of transferring data to device memory (e.g. CUDA., openACC)
- Supply pointers to devide meomry

```
dgetrf_acc(M, N, d_A, lda, ipiv, &info)
```

- Data **must** already exist in device at address d_A!

Using Pinned Memory

- Pinned memory is a CUDA feature that allows you to perform asynchronous data transfer
- As of feb2012, within the simple interface – pinned memory is essential for performance
- Libsci_ACC provides tools to allow you to pin memory
  - libsci_acc_HostAlloc
  - libsci_acc_FreeHost
- You can use simple interface without pinning memory, but performance will be poor

In a future release, pinned memory will not be a requirement

OpenACC support

- libsci_acc is independent to libsci_acc but fully compatible with it
- Use data and host_data directives to manage data transfer and memory allocation on GPU.
  - For BLAS, copy all matrix and vector arrays to GPU
  - Scalar variables must stay on CPU
- Because simple interface can accept either device or host pointers you can use standard compliant calls

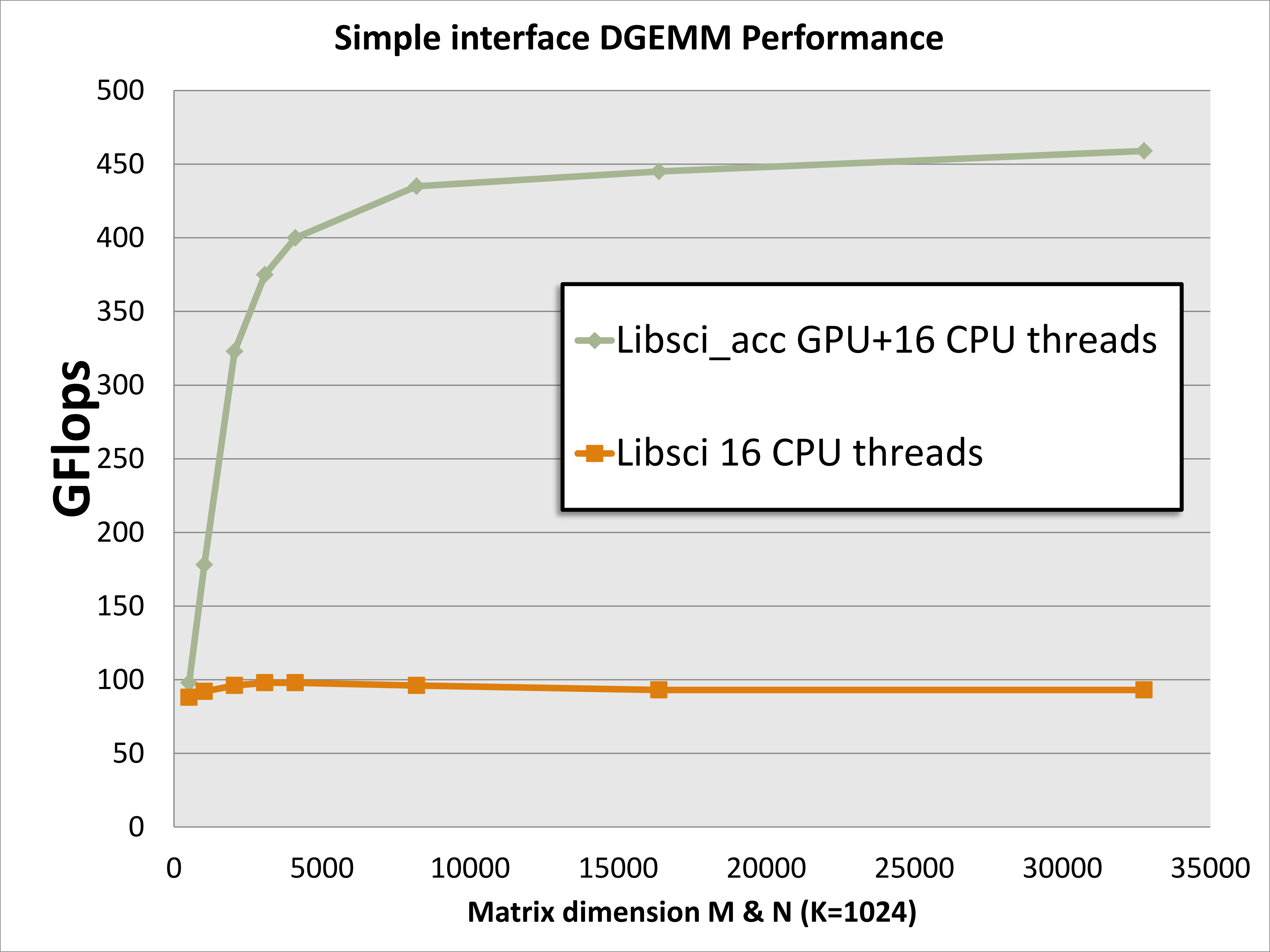
OpenACC and libsci_acc example

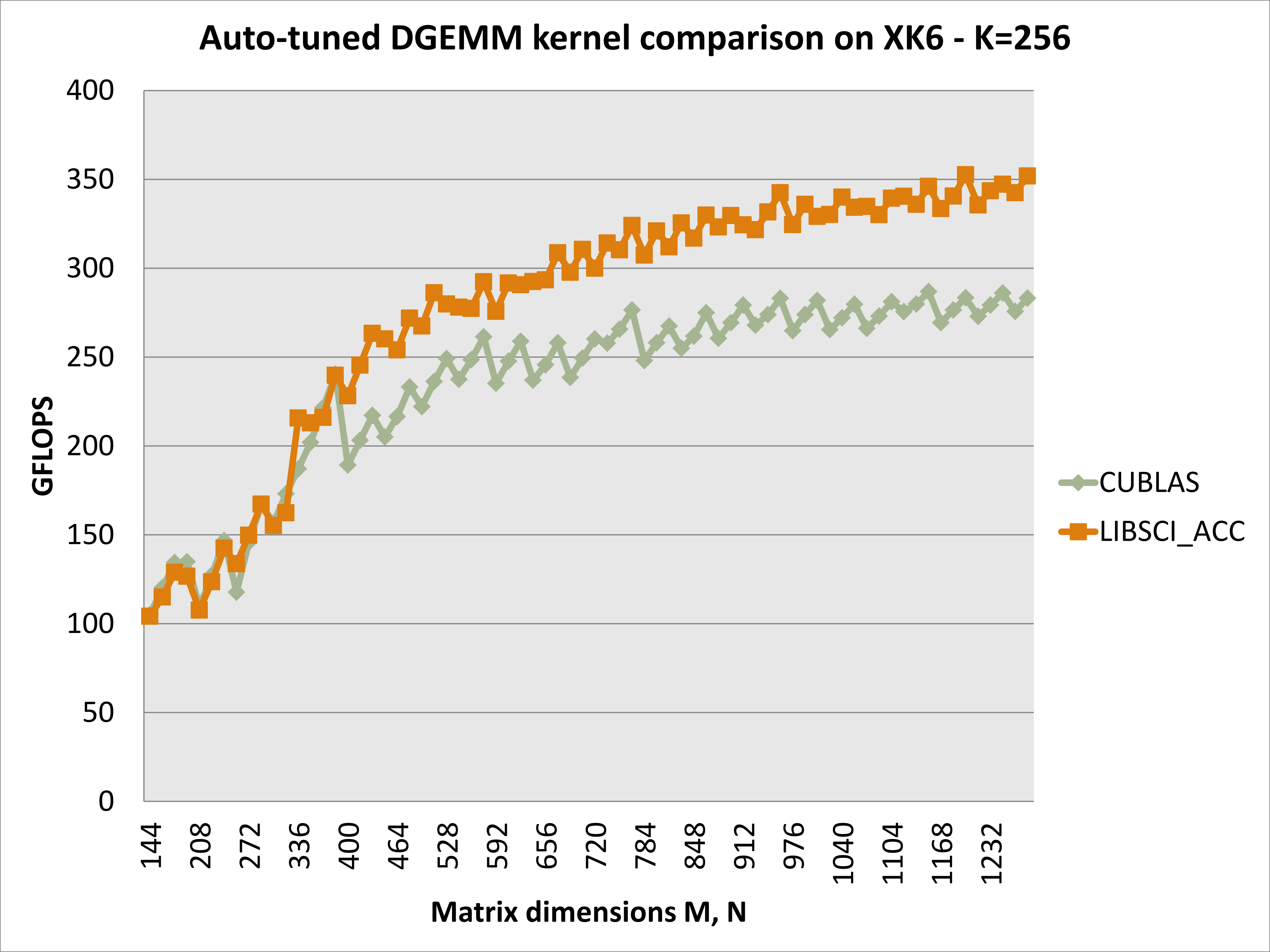
```
!$acc data copy(c), copyin(a,b)
!$acc host_data use_device(a,b,c)
     call dgemm_acc('n','n',m,n,k,alpha,a,lda,
&                       b,ldb,beta,c,ldc)
!$acc end host_data
!$acc end data
```

Functionaly equivalent to

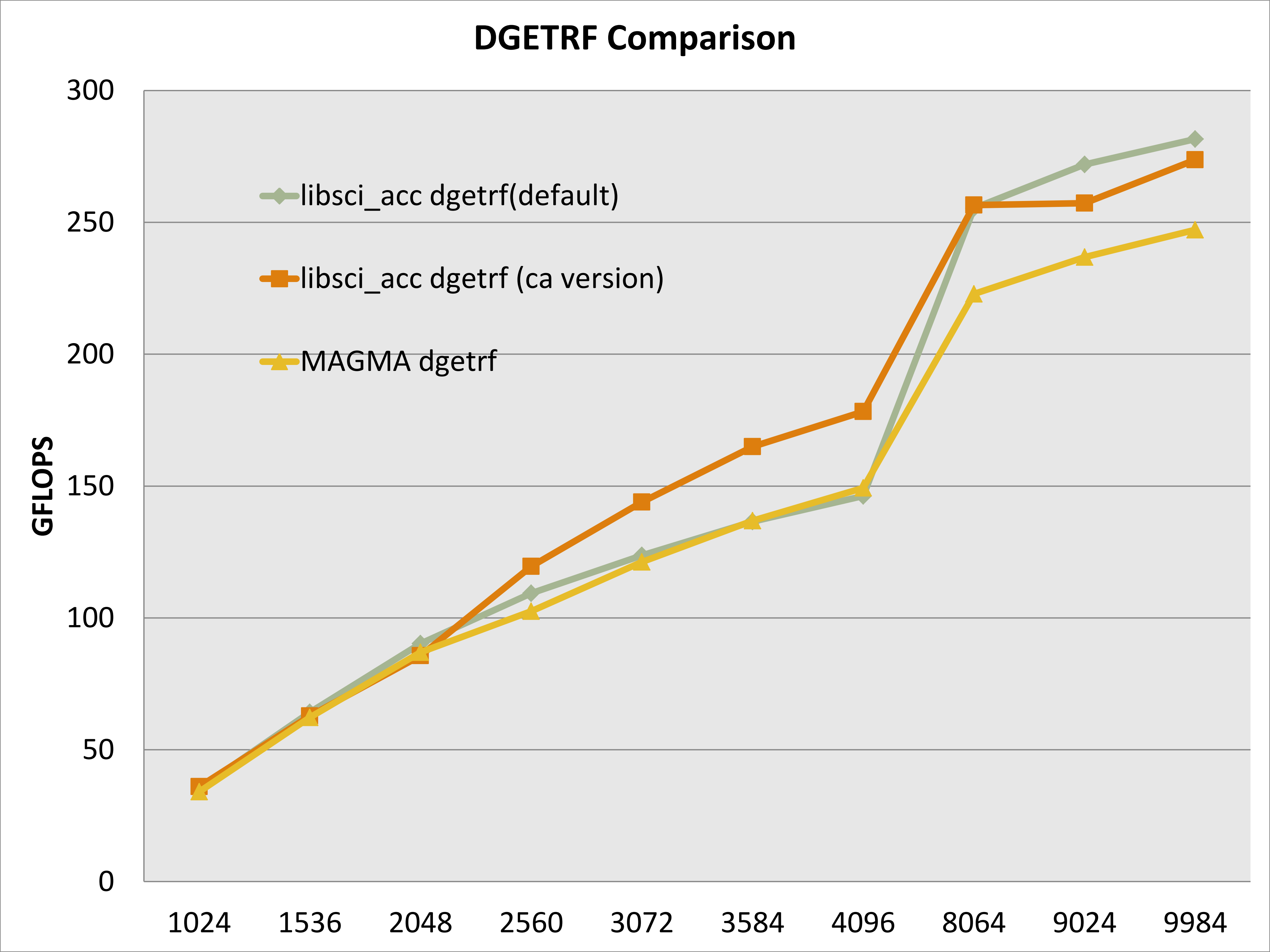
Open_ACC C example: LAPACK call

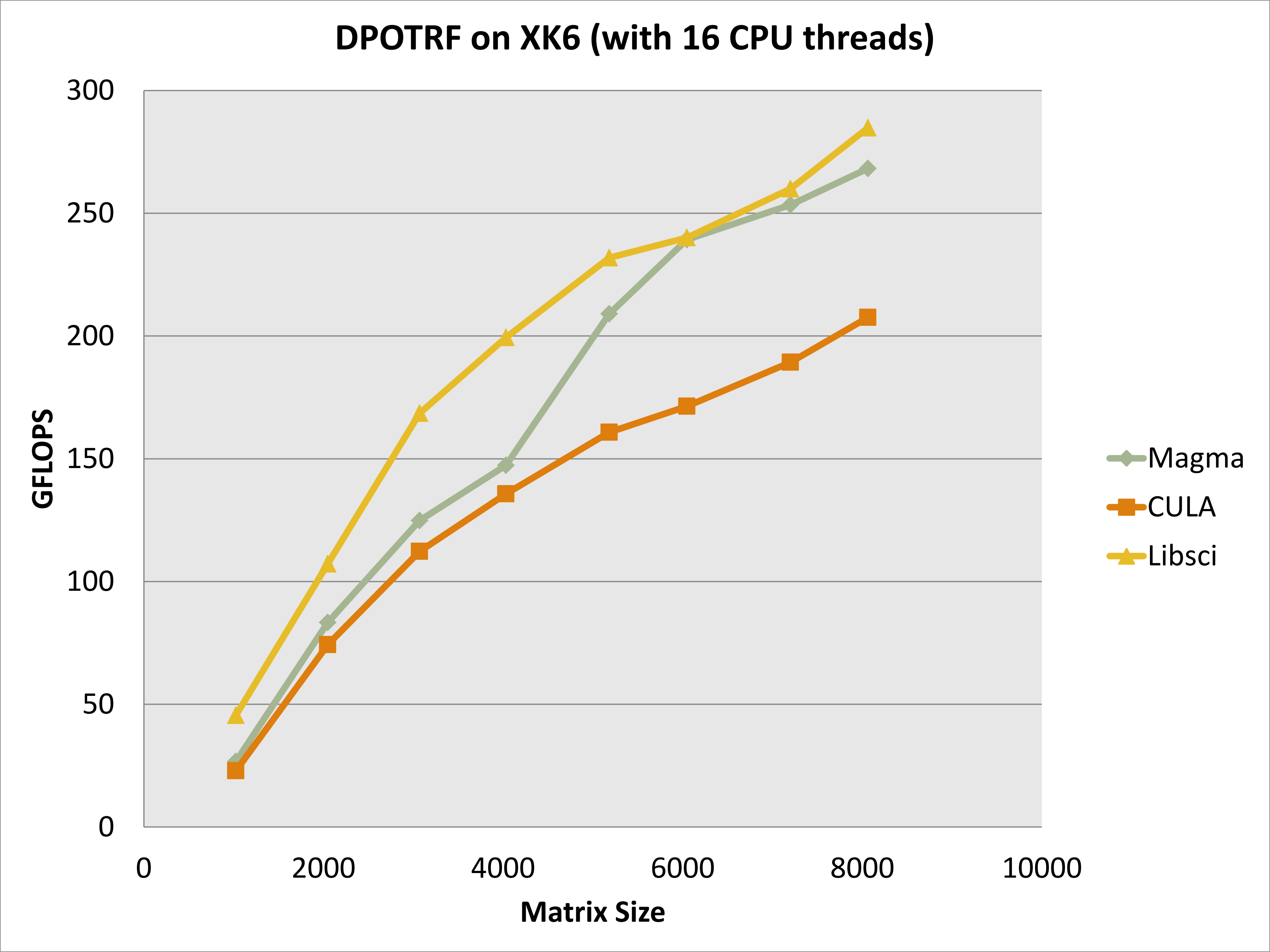
```
#pragma acc data copy(A[0:n*lda])
{
#pragma acc host_data use_device(A)
  {
      dgetrf_acc( &M, &N, A, &lda, ipiv,
  &info);
   }
}
```


Simple interface DGEMM Performance
GFlops
500
450
400
350
300
250
200
150
100
50
0
0
5000
10000
15000
20000
25000
30000
35000
Matrix dimension M & N (K=1024)
Libsci_acc GPU+16 CPU threads
Libsci 16 CPU threads

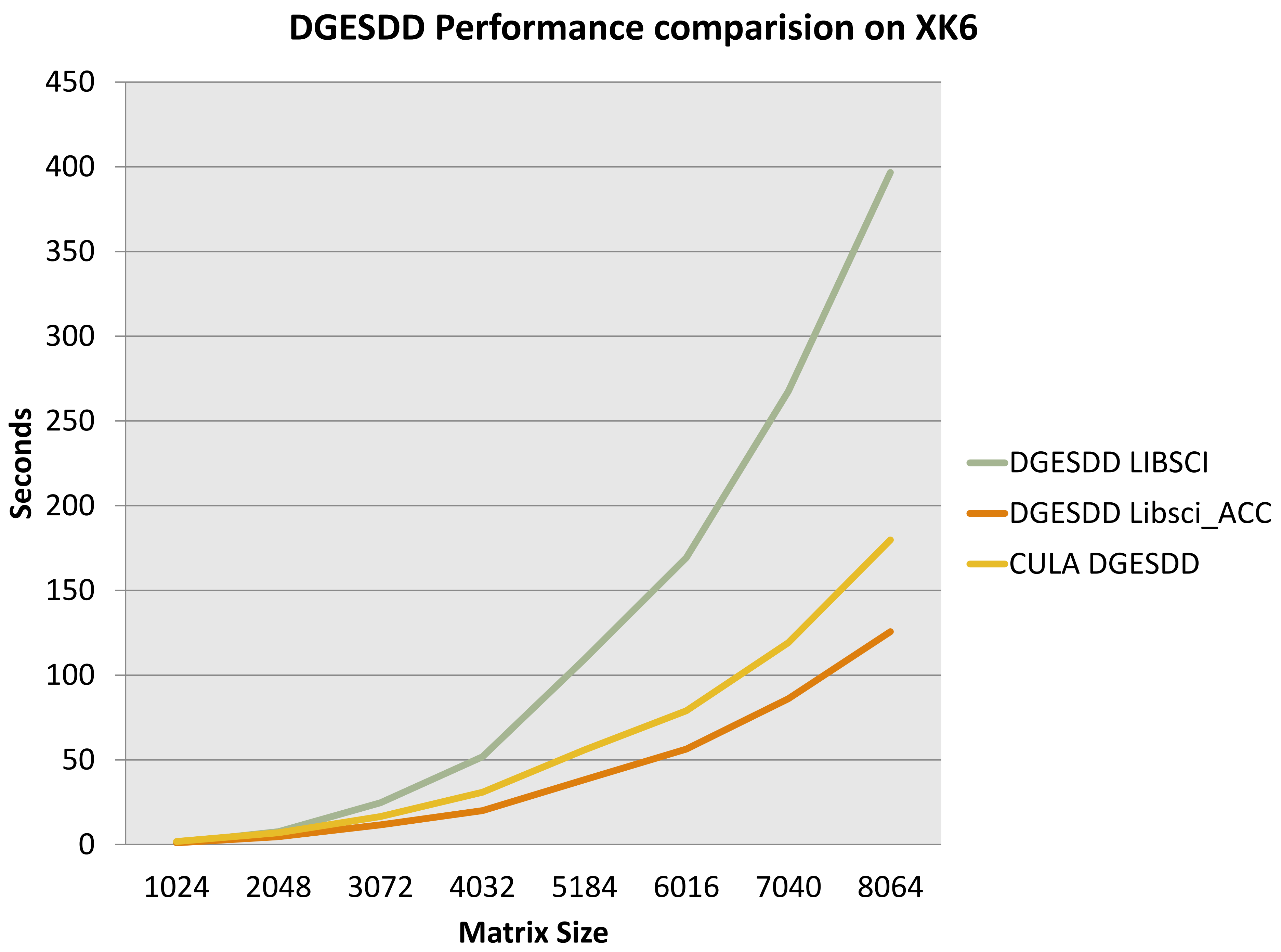


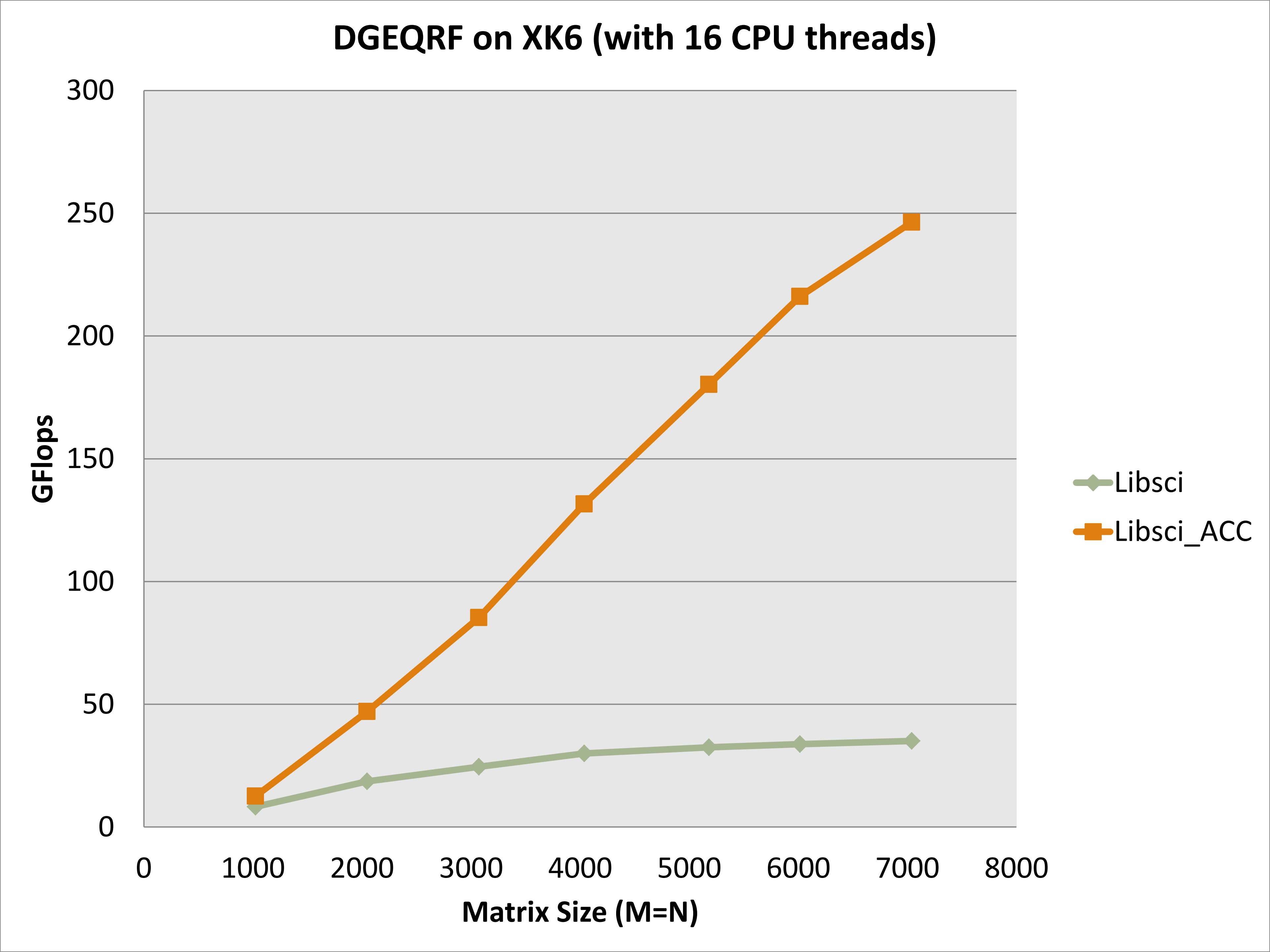
Auto-tuned DGEMM kernel comparison on XK6 - K=256
GFLOPS
400
350
300
250
200
150
100
50
0
144
208
272
336
400
464
528
592
656
720
784
848
912
976
1040
1104
1168
1232
Matrix dimensions M, N
CUBLAS
LIBSCI_ACC


DGETRF Comparison
GFLOPS
300
250
200
150
100
50
0
1024
1536
2048
2560
3072
3584
4096
8064
9024
9984
libsci_acc dgetrf(default)
libsci_acc dgetrf (ca version)
MAGMA dgetrf



DPOTRF on XK6 (with 16 CPU threads)
GFLOPS
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
Matrix Size
Magma
CULA
Libsci


DGESDD Performance comparision on XK6
Seconds
450
400
350
300
250
200
150
100
50
0
1024
2048
3072
4032
5184
6016
7040
8064
Matrix Size
DGESDD LIBSCI
DGESDD Libsci_ACC
CULA DGESDD

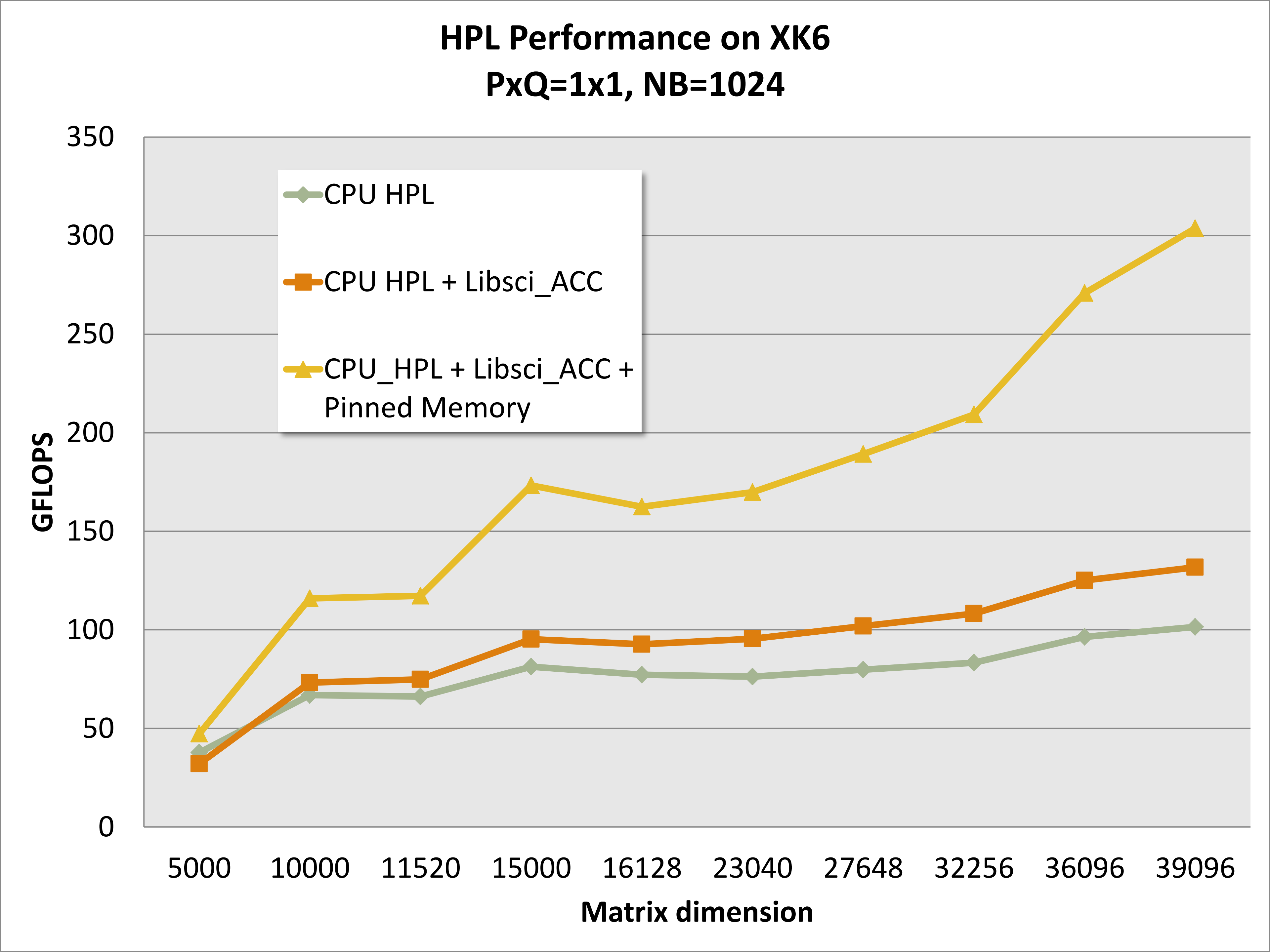

DGEQRF on XK6 (with 16 CPU threads)
GFlops
300
250
200
150
100
50
0
0
1000
2000
3000
4000
5000
6000
7000
8000
Matrix Size (M=N)
Libsci
Libsci_ACC

3 minutes to 300GF with HPL!

- You don’t need a fancy version of HPL for the GPU
- With 3 minor code changes you can use stock HPL code :
  1. Add `#include “libsci_acc.h“`
  2. Replace 1 instance of malloc with libsci_acc_HostAllocc
  3. replace 1 free with `libsci_acc_FreeHost`
- I can provide more details of where/what to change
- Then run HPL as normal :

```
T/V              N     NB    P    Q          Time          Gflops
--------------------------------------------------------------------------------
WR11R4L2     39096  1024    1    1        126.66       3.146e+02
```

By June 2012 release you won’t need to make any code change


HPL Performance on XK6
PxQ=1x1, NB=1024
350
300
250
200
150
100
50
0
GFLOPS
CPU HPL
CPU HPL + Libsci_ACC
CPU_HPL + Libsci_ACC + Pinned Memory
5000
10000
11520
15000
16128
23040
27648
32256
36096
39096
Matrix dimension

Upcoming features

- Hybrid BLAS for Unpinned Memory
- CUDA 5 support
- Auto-tuned BLAS for all precisions
- Auto-tuned BLAS for TN, NT, TT cases
- Auto-tuned SYMM/HEMM_ACC
- Eigenvalue solvers
- More hybrid Level 3 BLAS
- Requests?
- Small matrix problems?

EXAMPLES

- /users/cours01/CSCS_XK6_Course_2012/Day2/Tutorials

  LIBSCI_Acc_examples.tar

- HYBRID_DGEMM
- dgetrf_CCE
- dgetrf_F90
- dgetrf_GNU
- hpl-2.0
- hpl-2.0-xk6
- hpl-2.0-xk6-pinned
- OpenACC_DGEMM

THE SUPERCOMPUTER COMPANY

Using Pinned Memory in Fortran90

- Use functions and data types from iso_c_binding to enable libsci_acc_hostalloc
- Enables hybrid computing with a few lines of modification

```
! Enable C pointer
use iso_c_binding
! Declare C pointer
type(C_PTR)::cptr_A
complex*16, pointer, dimension (:,:) :: A
! Initialize libsci_acc
call libsci_acc_init()
! Allocate pinned memory to C pointer
ierr = libsci_acc_hostalloc(cptr_A, INT8(16*max_dim*max_dim))
! Convert the C pointer to Fortran pointer for 2 dimensional array
call c_f_pointer(cptr_A,A,(/lda,max_dim/))
```

More tuning with libsci_acc in future (2012 Q1)

- More LAPACK routines support
- A few lines of code change.
- Controls algorithm choice and data transfer mode through environment variables
  - setenv LIBSCI_LAPACK_ZGESV_RHSONLY 1

```
do iblk=nblk,2,-1
      m=n
      ioff=joff
      n=blk_sz(iblk-1)
      joff=joff-n
      call zgesv( m, ioff, a(ioff+1,ioff+1),lda, ipvt,  a(ioff+1,1),lda,info)
!     call zgetrf(m,m,a(ioff+1,ioff+1),lda, ipvt,info)
!     call zgetrs('n',m,ioff,a(ioff+1,ioff+1),lda,ipvt,  a(ioff+1,1),lda,info)
     if(iblk.gt.2) then
            call zgemm('n','n',n,ioff-k+1,na-ioff,cmone,a(joff+1,ioff+1),lda,
     &      a(ioff+1,k),lda,cone,a(joff+1,k),lda)
             call zgemm('n','n',joff,n,na-ioff,cmone,a(1,ioff+1),lda,
     &      a(ioff+1,joff+1),lda,cone,a(1,joff+1),lda)
     endif
end do
```