LibSci for Accelerators (libsci_acc)

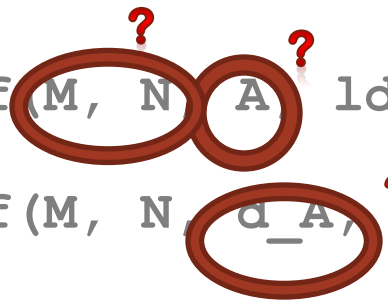
- **Provide basic libraries for accelerators, tuned for Cray**
- **Must be independent to OpenACC, but fully compatible**
- **Multiple use case support**
  - Get the base use of accelerators with no code change
  - Get extreme performance of GPU with or without code change
  - Extra tools for support of complex code
- **Incorporate the existing GPU libraries into libsci**
- **Provide additional performance and usability**
- **Maintain the Standard APIs where possible!**

Why libsci_acc?

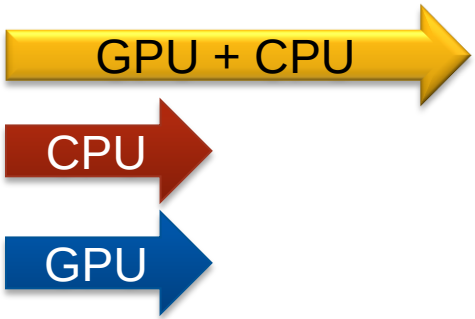
- **Code modification is required to use existing GPU libraries!**
- **Several scientific library packages are already there**
  - CUBLAS, CUFFT, CUSPARSE (NVIDIA), MAGMA (U Tennessee), CULA (EM Photonics)
- **No Compatibility to Legacy APIs**
  - cublasDgemm(….)
  - magma_dgetrf(…)
  - culaDgetrf(…)
  - Why not dgemm(), dgetrf()?
- **Not focused on Fortran API (C/C++)**
  - Require CUDA data types, primitives and functions in order to call them
- **Performance**

Three Interfaces For Three Use Cases

- **Simple interface**



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



- **Device interface**

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

- **CPU interface**

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

Simple Interface

- **You can pass either host pointers or device pointers to the simple interface**
- **Host memory pointer**
  - Performs hybrid operation on GPU
  - If problem is too small, performs host operation
- **Device memory pointer**
  - Performs operation on GPU
- **BLAS 1 and 2 perform computation local to the data location**
  - CPU-GPU data transfer is too expensive to exploit hybrid execution

Device Interface

- **Device interface gives higher degree 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**

libsci_acc Usage - Basics

- **Supports Cray and GNU compilers.**
- **Fortran and C interfaces (column-major assumed)**
  - Load the module ***craype-accel-nvidia35.***
  - Compile as normal (dynamic libraries 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**

libsci_acc DGEMM Example

- Starting with a code that relies on dgemm.
- The library will check the parameters at runtime.
- If the size of the matrix multiply is large enough, the library will run it on the GPU, handling all data movement behind the scenes.
- NOTE: Input and Output data are in CPU memory.

```
call dgemm('n','n',m,n,k,alpha,&
           a,lda,b,ldb,beta,c,ldc)
```

libsci_acc Interaction with OpenACC

- If the rest of the code uses OpenACC, it's possible to use the library with directives.
- All data management performed by OpenACC.
- Calls the device version of dgemm.
- All data is in CPU memory before and after data region.

```
!$acc data copy(a,b,c)

!$acc parallel
!Do Something
!$acc end parallel

!$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
```

libsci_acc Interaction with OpenACC

- libsci_acc is a bit smarter than this.
- Since ‘a,’ ‘b’, and ‘c’ are device arrays, the library knows it should run on the device.
- So just dgemm is sufficient.

```
!$acc data copy(a,b,c)

!$acc parallel
!Do Something
!$acc end parallel

!$acc host_data use_device(a,b,c)

call dgemm     ('n','n',m,n,k,&
                alpha,a,lda,&
                b,ldb,beta,c,ldc)

!$acc end host_data
!$acc end data
```

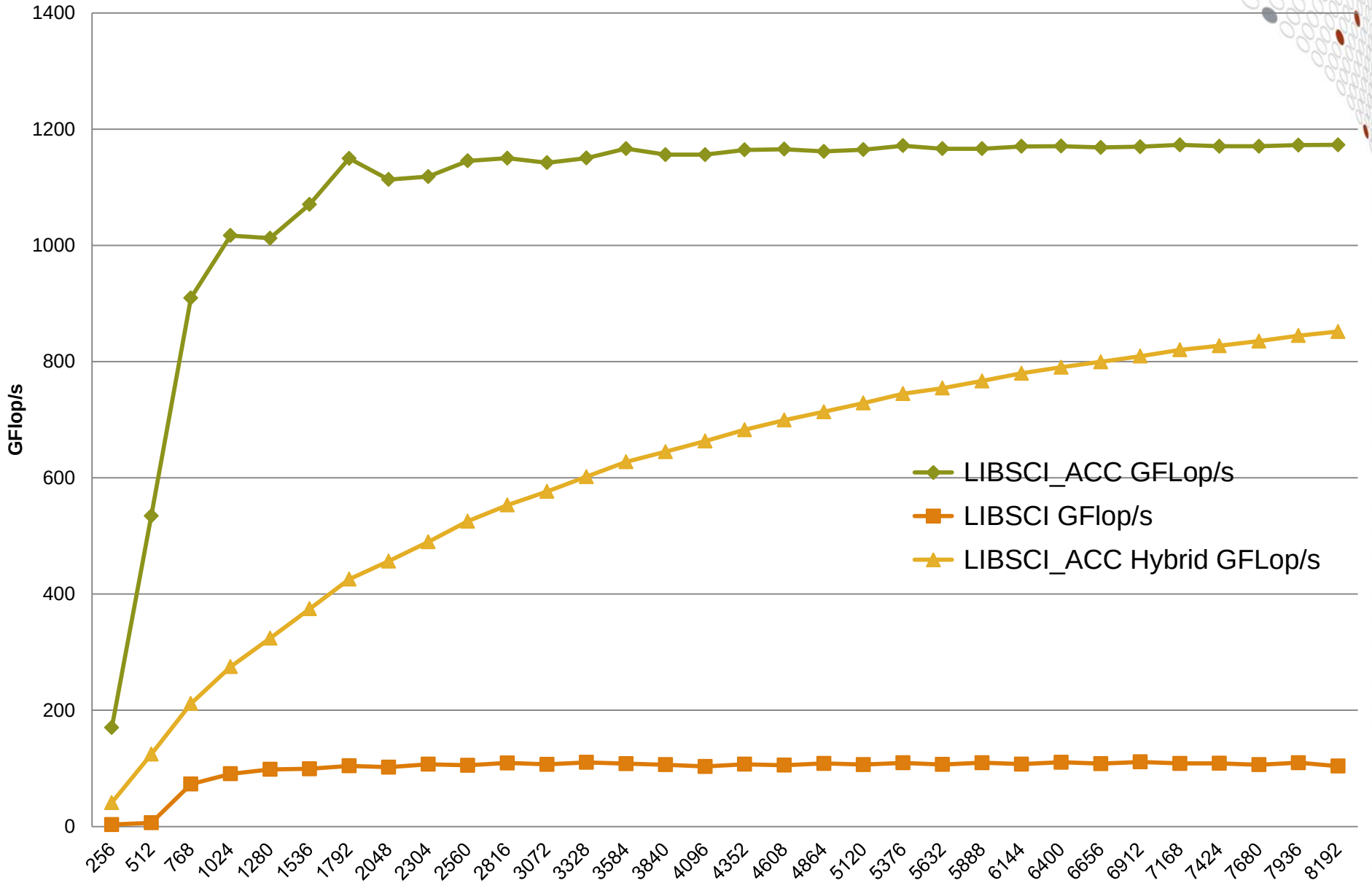
Advanced Controls

- **The communication avoidance (CA) version of DGETRF/ZGETRF can be enabled by setting the environment variable LIBSCI_ACC_DLU = CALU / LIBSCI_ACC_ZLU = CALU**

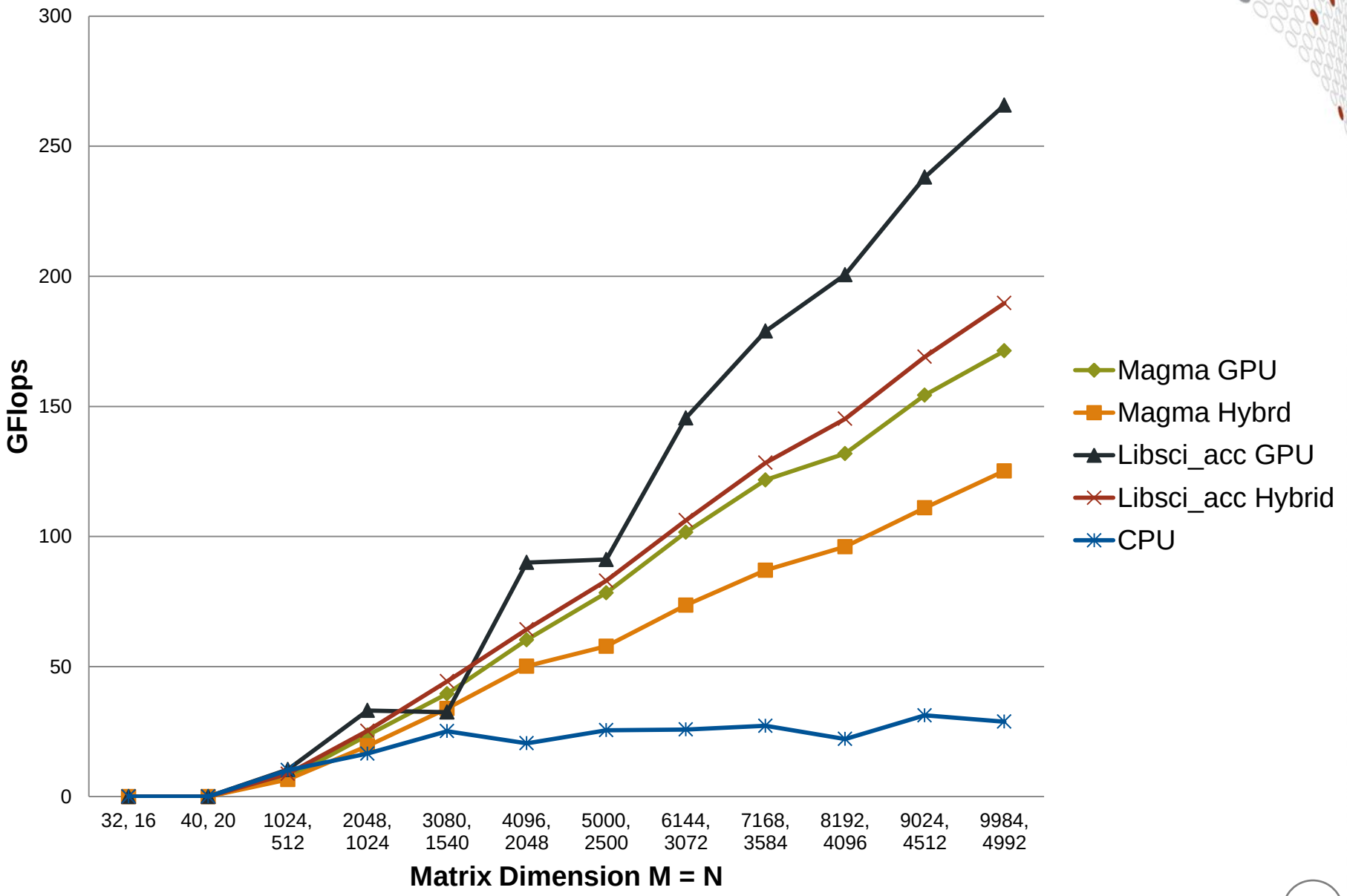
- **Change Split Ratio of Hybrid GEMM routines**
  - LIBSCI_SGEMM_SPLIT=0.9
  - LIBSCI_DGEMM_SPLIT=0.8
  - LIBSCI_CGEMM_SPLIT=0.9
  - LIBSCI_ZGEMM_SPLIT=0.8

- **Force simple API to always call CPU routine**
  - CRAY_LIBSCI_ACC_MODE=2

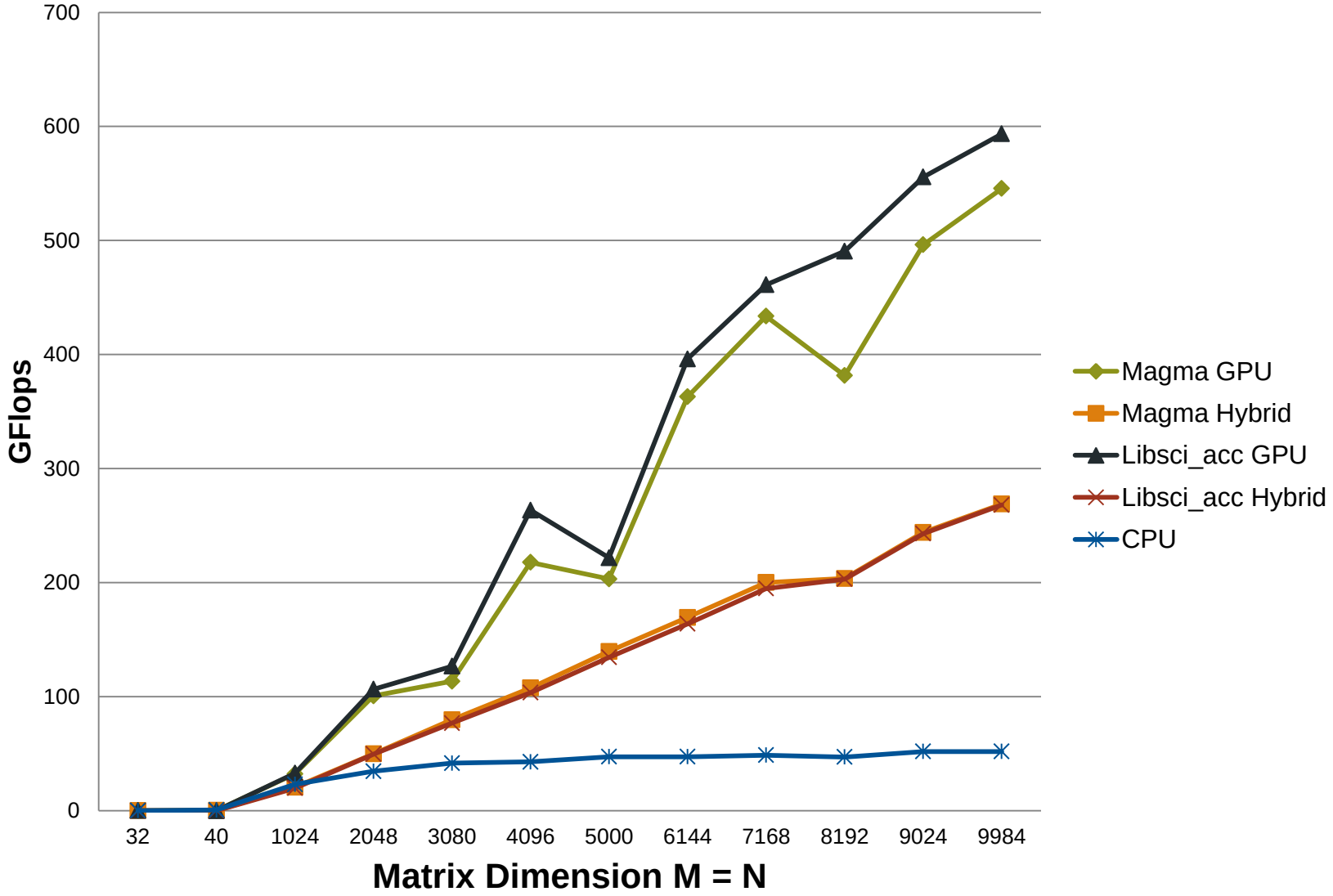
Matrix Multiplication :: Double (DGEMM)
XK7 Kepler :: Nov 2012



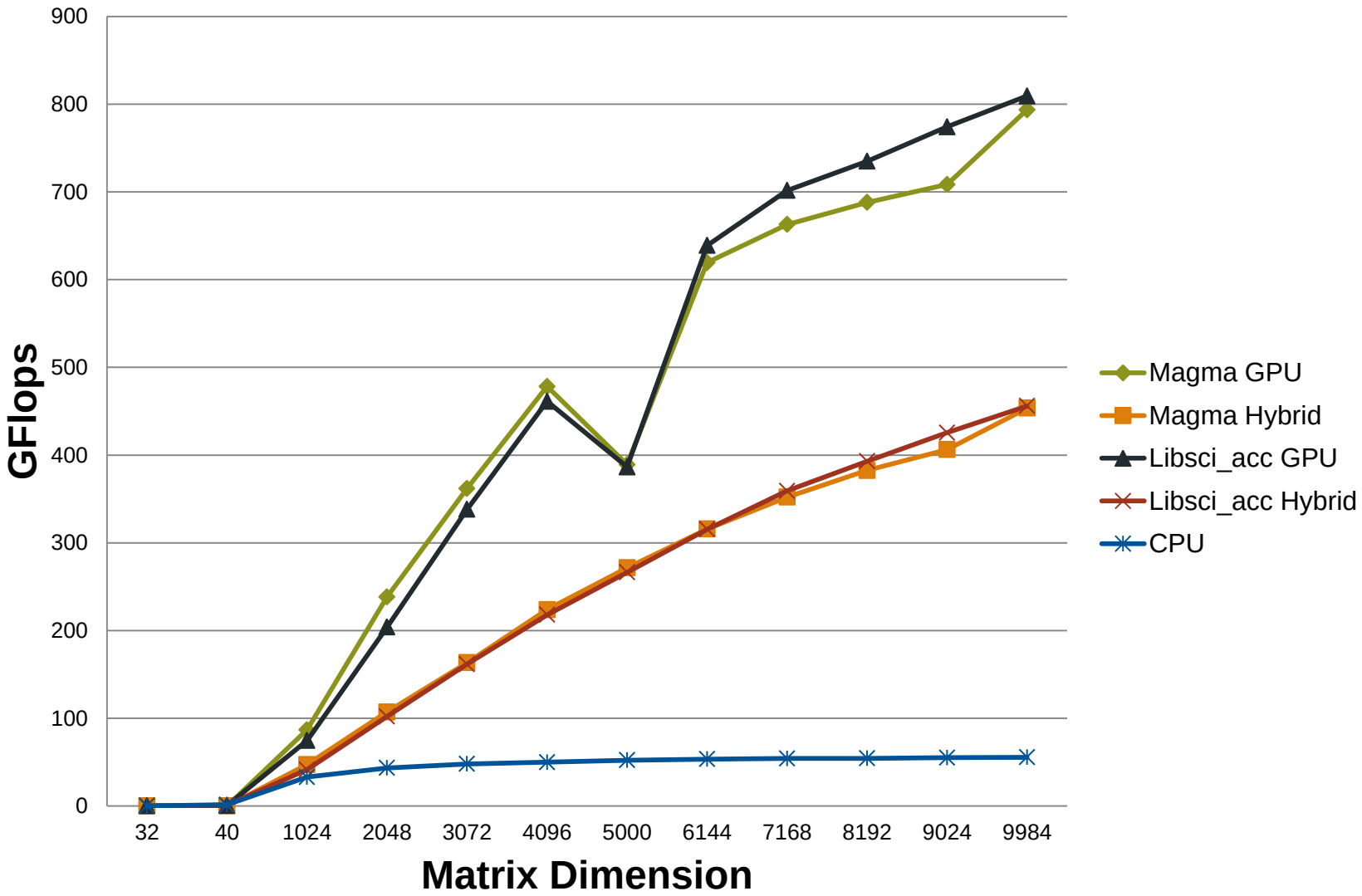
LAPACK QR factorization :: DGEQRF
XK7 Kepler :: Nov 2012



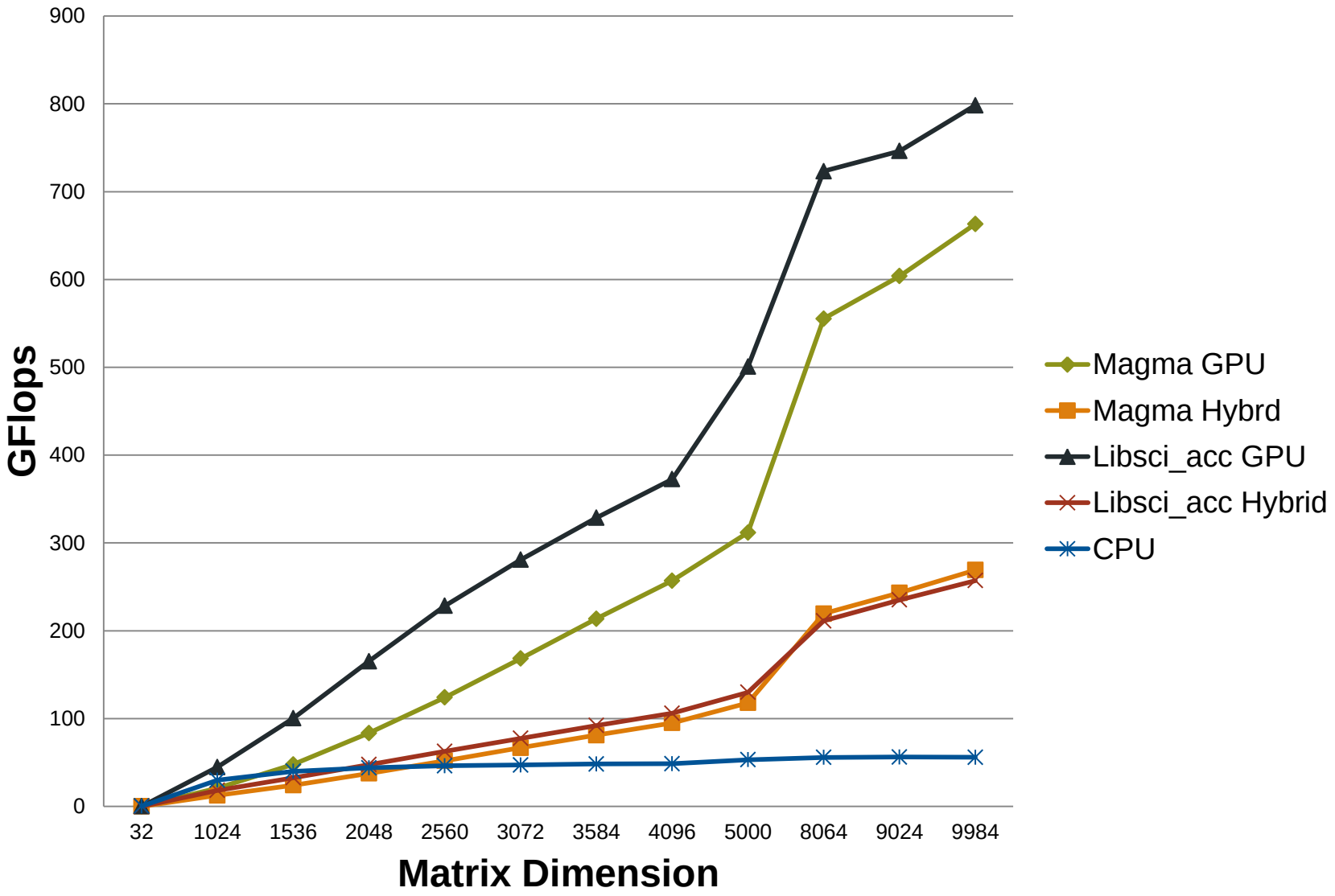
LAPACK LU factorization :: double (DGETRF)
XK7 Kepler :: Nov 2012



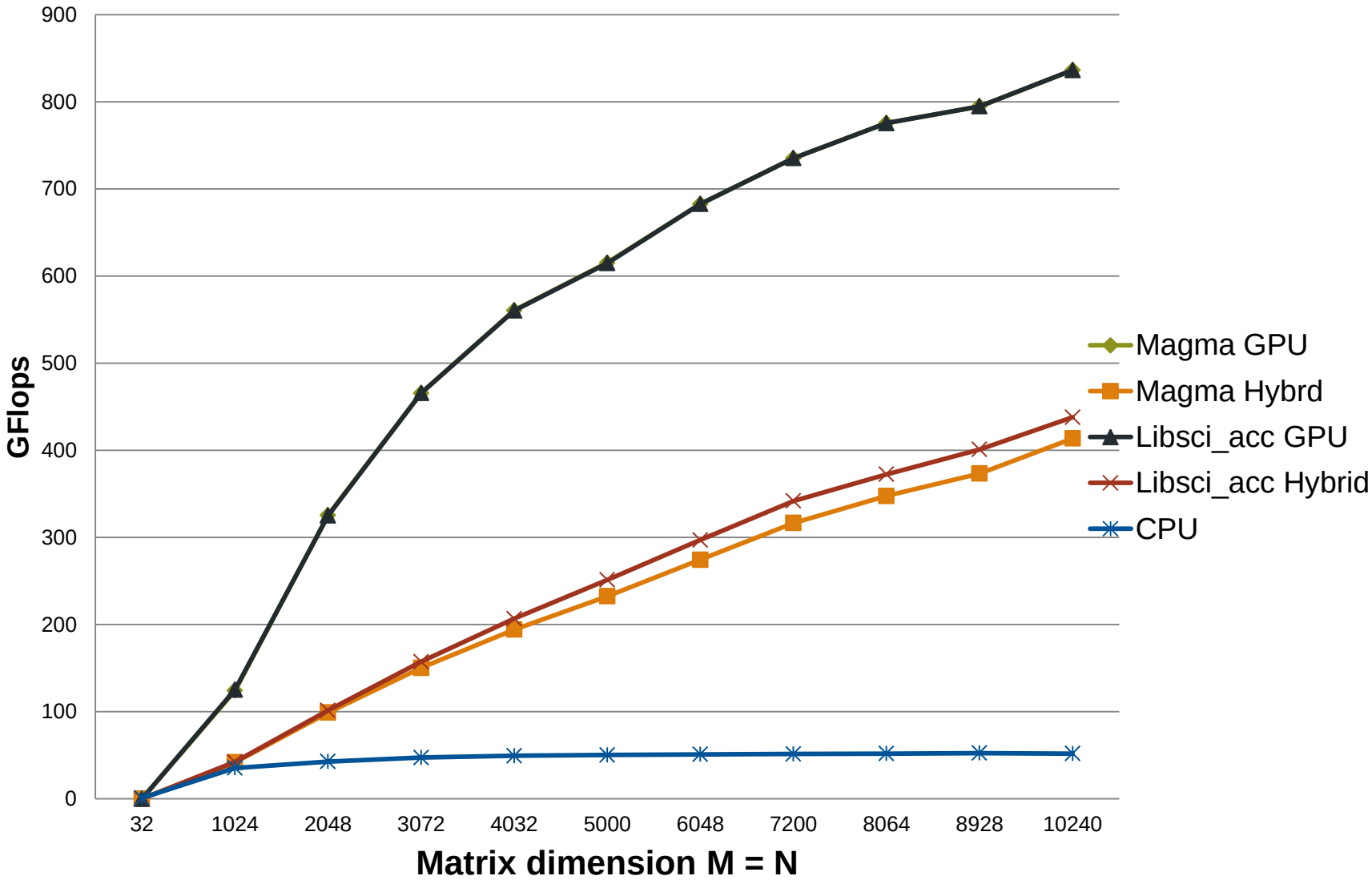
LAPACK LU factorization :: double complex (ZGETRF)
XK7 Kepler :: Nov 2012



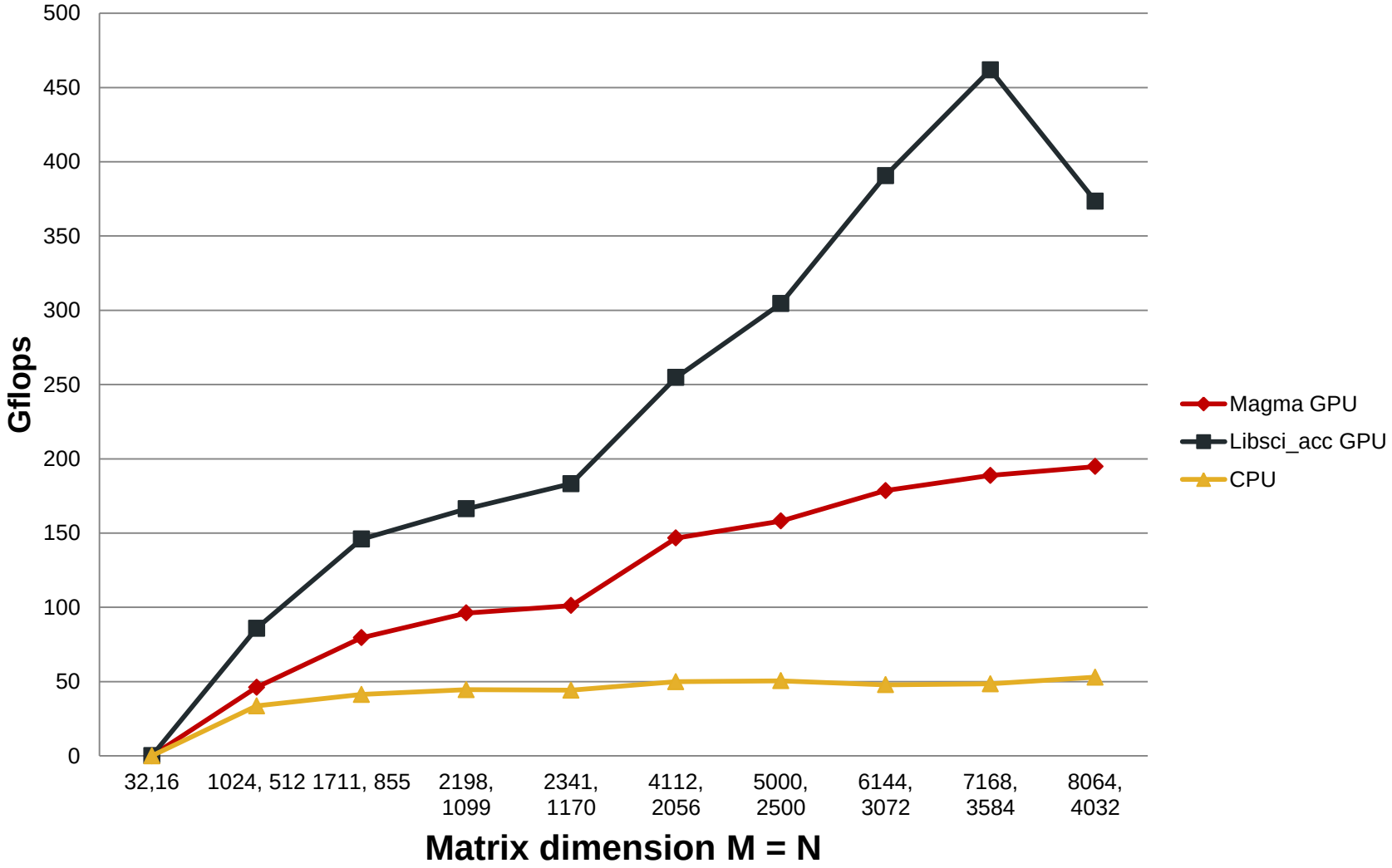
LAPACK Cholesky factorization :: double (DPOTRF)
XK7 Kepler :: Nov 2012



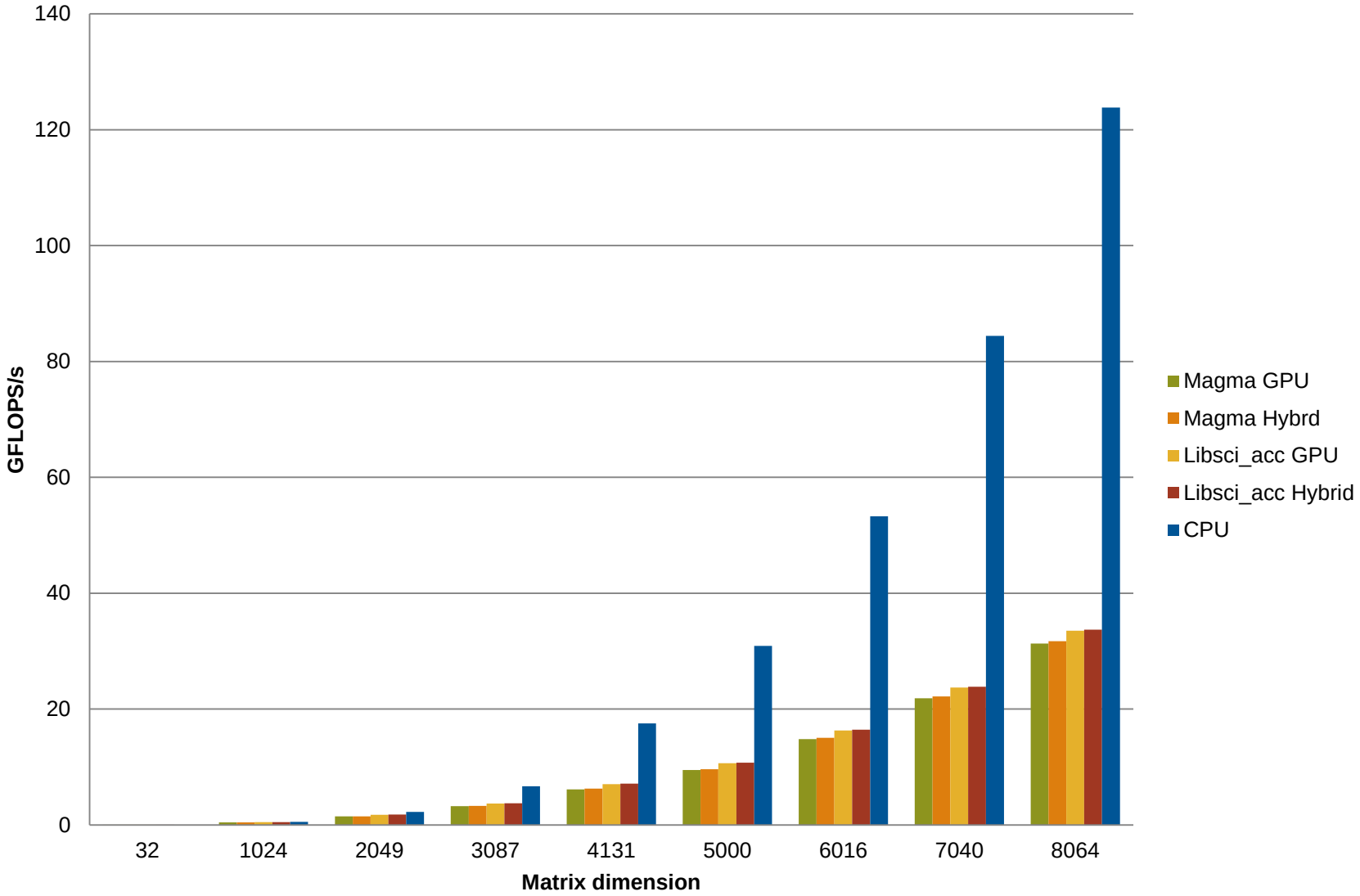
LAPACK Cholesky factorization :: double complex (ZPOTRF)
XK7 Kepler :: Nov 2012



LAPACK Cholesky solver :: double (DPOTRS)
XK7 Kepler :: Nov 2012



LAPACK divide and conquer eigensolver :: double (DSYEVD)
XK7 Kepler :: Nov 2012



libsci_acc BLAS Routines Available

- **BLAS 3 - HYBRID Implementations**
  - [s,d,c,z]GEMM
  - [s,d,c,z]GEMM
  - [s,d,c,z]TRSM
  - [z,c]HEMM
  - [s,d,c,z]SYMM
  - [s,d,c,z]SYRK
  - [z,d]HERK
  - [s,d,c,z]SYR2K
  - [s,d,c,z]TRMM
- **The following are supported without HYBRID implementations because there is no performance advantage**
  - All BLAS 2 Routines
  - All BLAS 1 Routines

libsci_acc LAPACK Routines Available

- **HYBRID Implementations:**
  - [d,z]GETRF (LU Factorization)
  - [d,z]POTRF (Cholesky Factorization)
  - [d,z]GETRS (System Solver)
  - [d,z]POTRS (System Solver)
  - [d,z]GESDD* (Generalized Singular Values)
  - [d,z]GEBRD (Generalized Bidiagonalization)
  - [d,z]GEQRF* (QR Factorization)
  - [d,z]GELQF (LQ Factorization
  - [d,z]GEEV (Non-symmetric Eigenvalues)
  - DSYEVR* / ZHEEVR* (Hermitian/Symmetric Eigenvalues)
  - DSYEV / DSYEVD (Hermitian/Symmetric Eigenvalues)
  - ZHEEV / ZHEEVD (Hermitian/Symmetric Eigenvalues)
  - DSYGVD / ZHEGVD (Hermitian/Symmetric Eigenvalue System Solver)

* Include Cray Proprietary Optimizations