Introduction to the Cray Scientific Libraries for Accelerators

Heidi Poxon
Cray Inc.

What Makes Cray Libraries Special?

- **Node performance**
  - Highly tuned routines at the low-level (ex. BLAS)
- **Network performance**
  - Optimized for network performance
  - Overlap between communication and computation
  - Use the best available low-level mechanism
  - Use adaptive parallel algorithms
- **Highly adaptive software**
  - Use auto-tuning and adaptation to give the user the known best (or very good) codes at runtime
- **Productivity features**
  - Simple interfaces into complex software

LibSci Usage

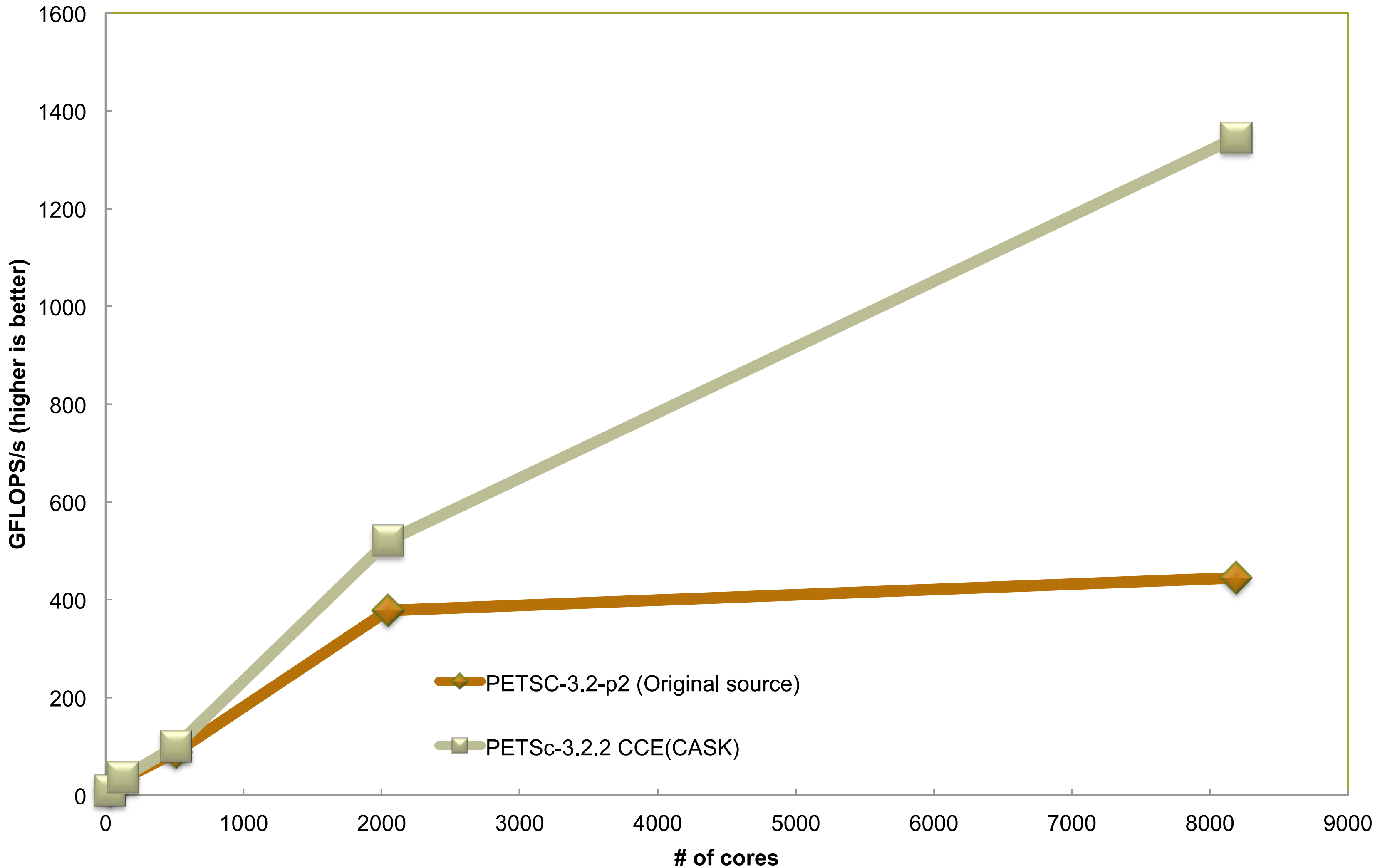
- **LIbSci**
  - The drivers should do it all for you. No need to explicitly link.
  - CCE will automatically pattern match to select scientific libraries
  - For threads, set OMP_NUM_THREADS
    - Threading is used within libsci.
    - If you call within a parallel region, single thread used
- **FFTW**
  - `module load fftw` (there are also wisdom files available)
- **PETSc**
  - `module load petsc` (or module load petsc-complex)
  - Use as you would your normal PETSc build
- **Trilinos**
  - `module load trilinos`
- **CASK – no need to do anything, you get optimizations free**

Cray Adaptive Sparse Kernel (CASK)

- **Sparse matrix operations in PETSc and Trilinos on Cray systems are optimized via CASK**
- **CASK is a product developed at Cray using the Cray Auto-tuning Framework**
- **Offline**
  - ATF program builds many thousands of sparse kernel
  - Testing program defines matrix categories based on density, dimension etc
  - Each kernel variant is tested against each matrix class
  - Performance table is built and adaptive library constructed
- **Runtime**
  - Scan matrix at very low cost
  - Map user's calling sequence to nearest table match
  - Assign best kernel to the calling sequence
  - Optimized kernel used in iterative solver execution

PETSc, Linear System Solution

2D Laplacian Problem
Weak Scalability
N=262,144 --- 268M
AMD Bulldozer 2.1G :: July 2012



Check You Got the Right Library!

- Add options to the linker to make sure you have the correct library loaded.
- -Wl adds a command to the linker from the driver
- You can ask for the linker to tell you where an object was resolved from using the –y option.
  - E.g. –Wl, -ydgemm_

```
.//main.o: reference to dgemm_
/opt/xt-libsci/11.0.05.2/cray/73/mc12/lib/libsci_cray_mp.a(dgemm.o):
definition  of dgemm_
```

Note : explicitly linking "-lsci" is bad! This won't be found from libsci 11+ (and means single core library for 10.x!)

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**

Auto-tuning

- **Cray Autotuning framework has been built to tune BLAS for accelerators**
  - GPU kernel codes are built using code generator
  - Enormous offline auto-tuning is used to build a map of performance to input
  - An adaptive library is built from the results of the auto-tuning
  - At run-time, your code is mapped to training set of input
  - Best kernel for your problem is used

Three Interfaces For Three Use Cases

- **Simple interface**

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

CPU

GPU + CPU

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

GPU

- **Device interface**

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

GPU

- **CPU interface**

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

CPU

Simple Interface

- You can pass either host pointers or device pointers to 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 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**

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

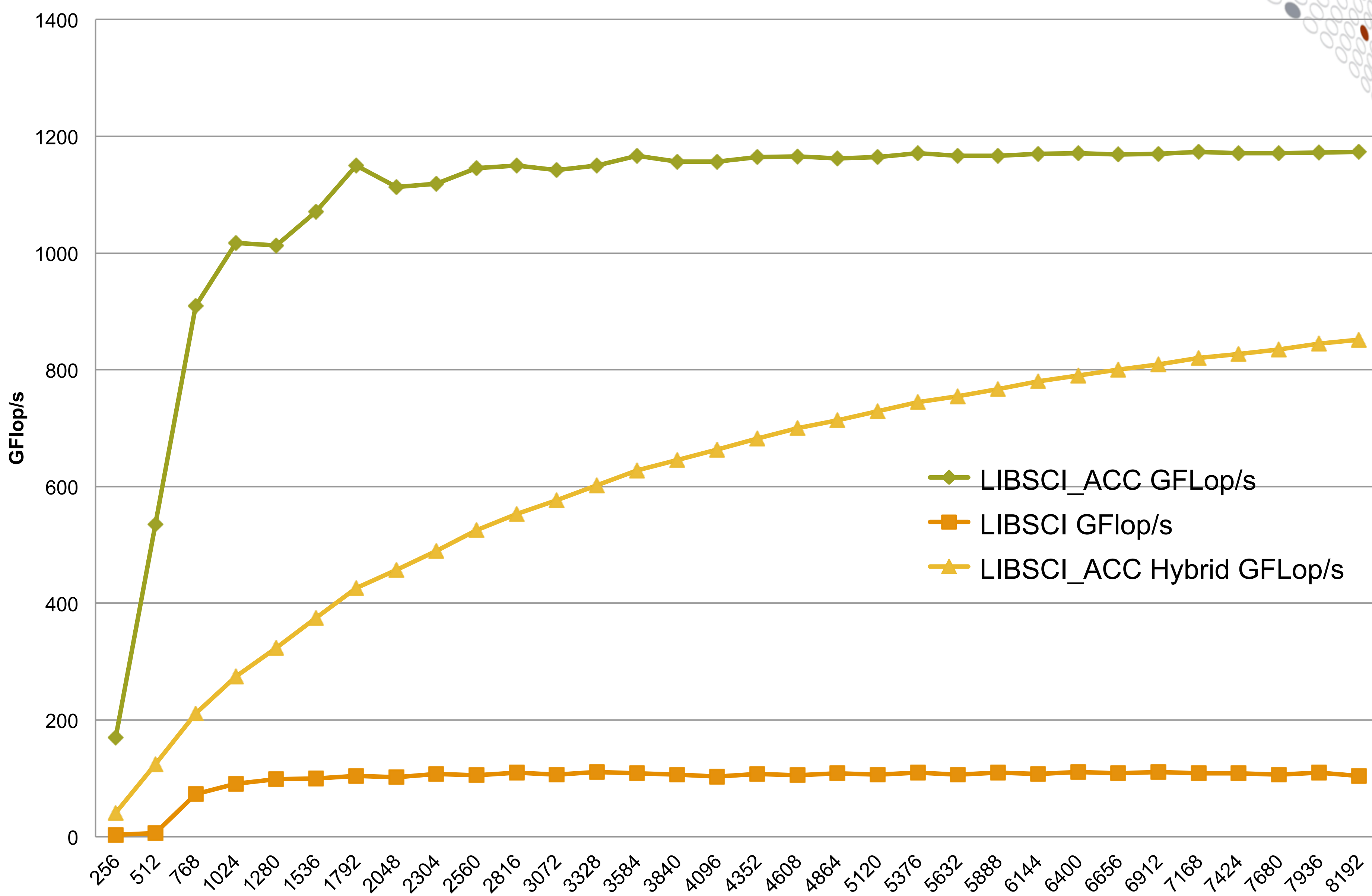
- libsci_acc is a bit smarter that this.
- Since ‘a,’ ‘b’, and ‘c’ are device arrays, the library knows it should run on the device.
- So just dgemm is sufficient.

```
!$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
```

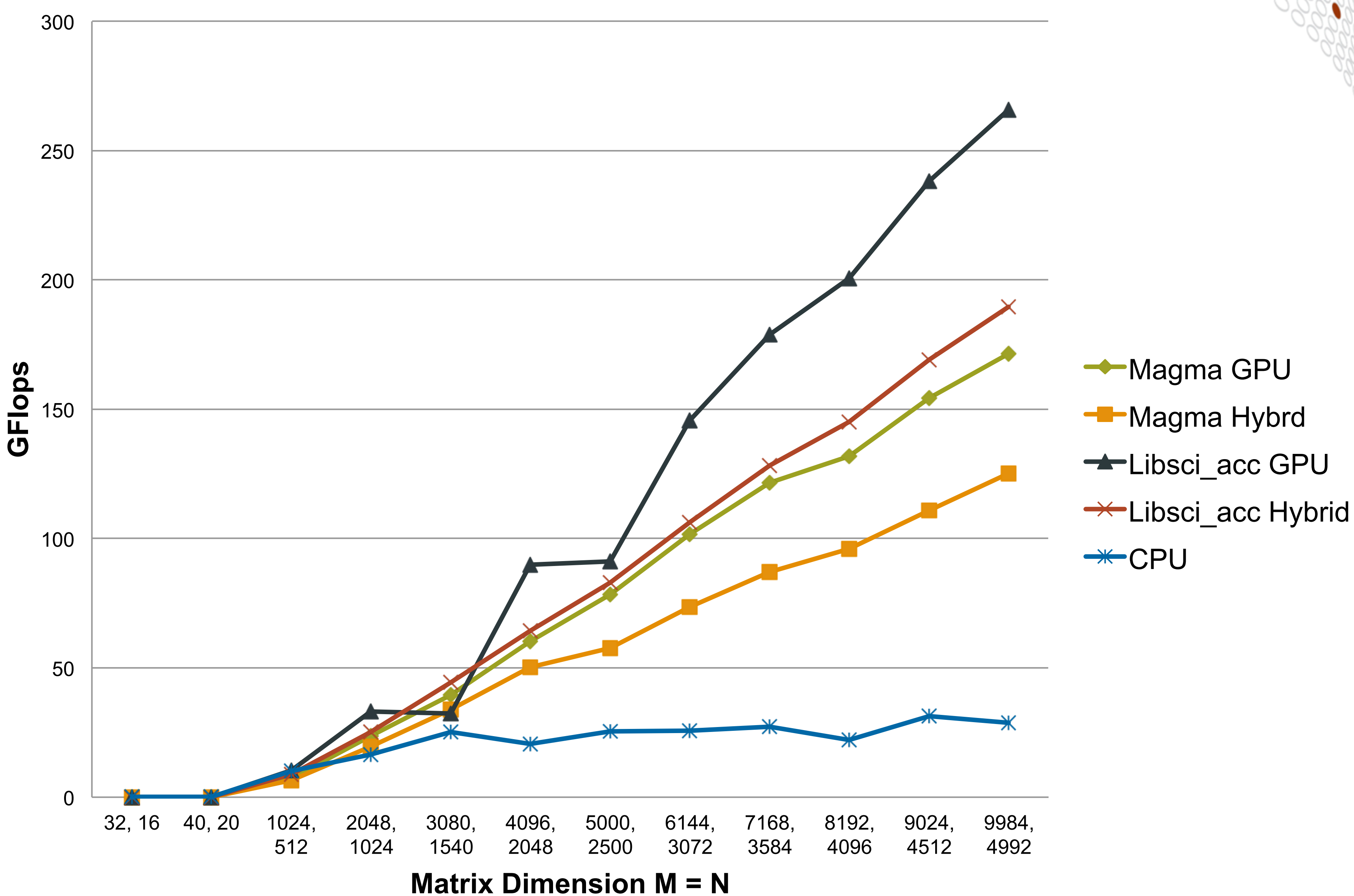
Advanced Controls

- 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

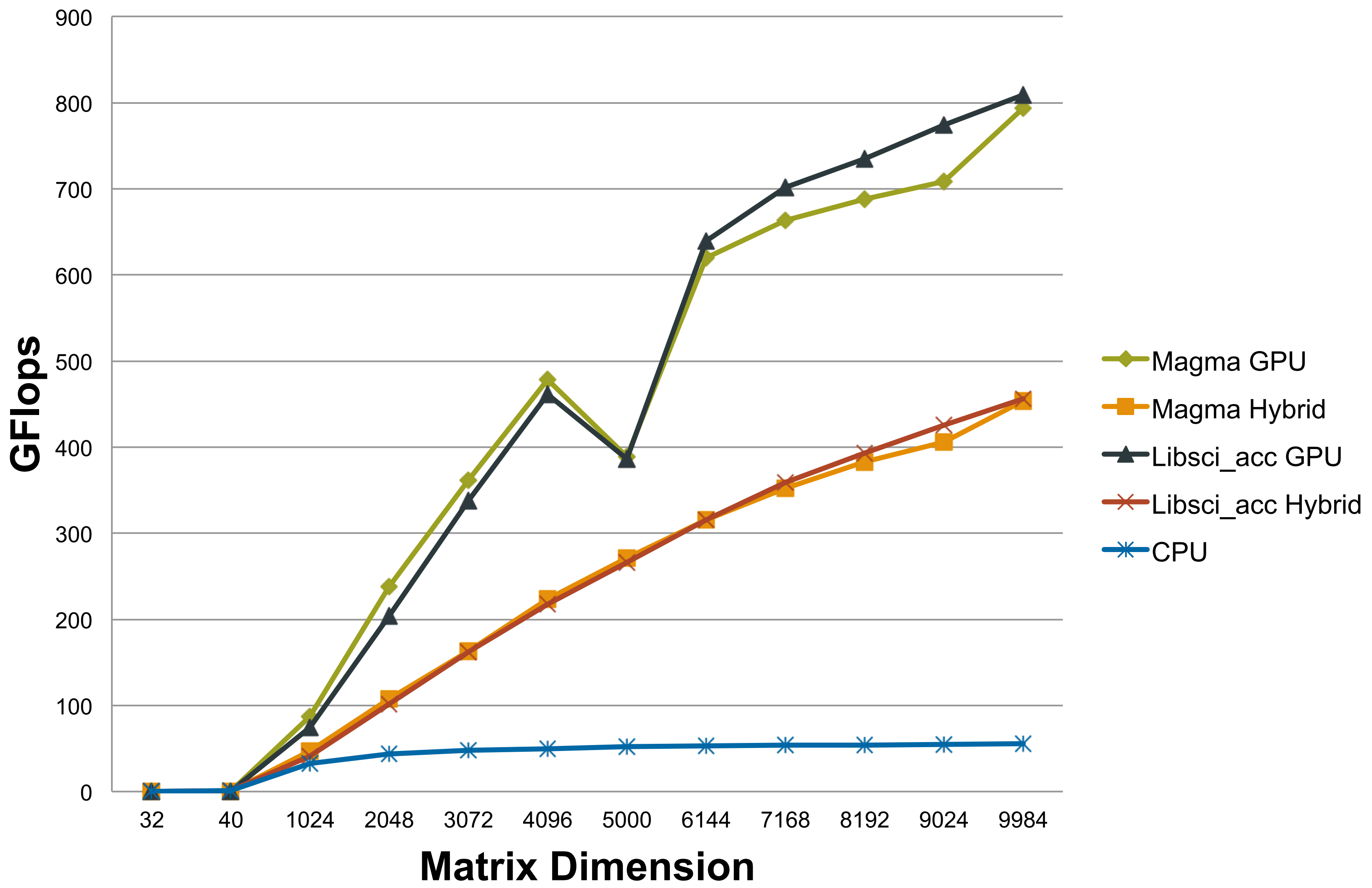
Matrix Multiplication :: Double (DGEMM)
XK7 Kepler :: Nov 2012



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



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



libsci_acc BLAS Routines Available

- **BLAS 3 - Full 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

- **Full 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

Summary

- **Access to libsci_acc routines is simple**
  - No need to explicitly link - Programming Environment drivers (cc, ftn, CC) do this for you
  - Just target the GPU by loading module
- **Can automatically take advantage of threading on CPU**
  - Just set OMP_NUM_THREADS and run
- **Simple interface available to enable hybrid, CPU or GPU execution of a routine depending on where memory pointers reside and problem size**
- **Interface for advanced control is also available**

Tuning Requests

- **CrayBLAS is an auto-tuned library**
  - Generally, excellent performance is possible for all shapes and sizes
- **However, the adaptive CrayBLAS can be improved by tuning for exact sizes and shapes**
- **Send your specific tuning requirements to**

  crayblas@cray.com

- **Send the routine name and the list of calling sequences**

Questions
?