Cray Scientific Libraries

: Overview and Performance

September 2012

Contents

- LibSci overview and usage
- BFRAME / CrayBLAS
- LAPACK
- ScaLAPACK
- FFTW / CRAFFT
- CASK
- LibSci for Accelerators

What are libraries for?

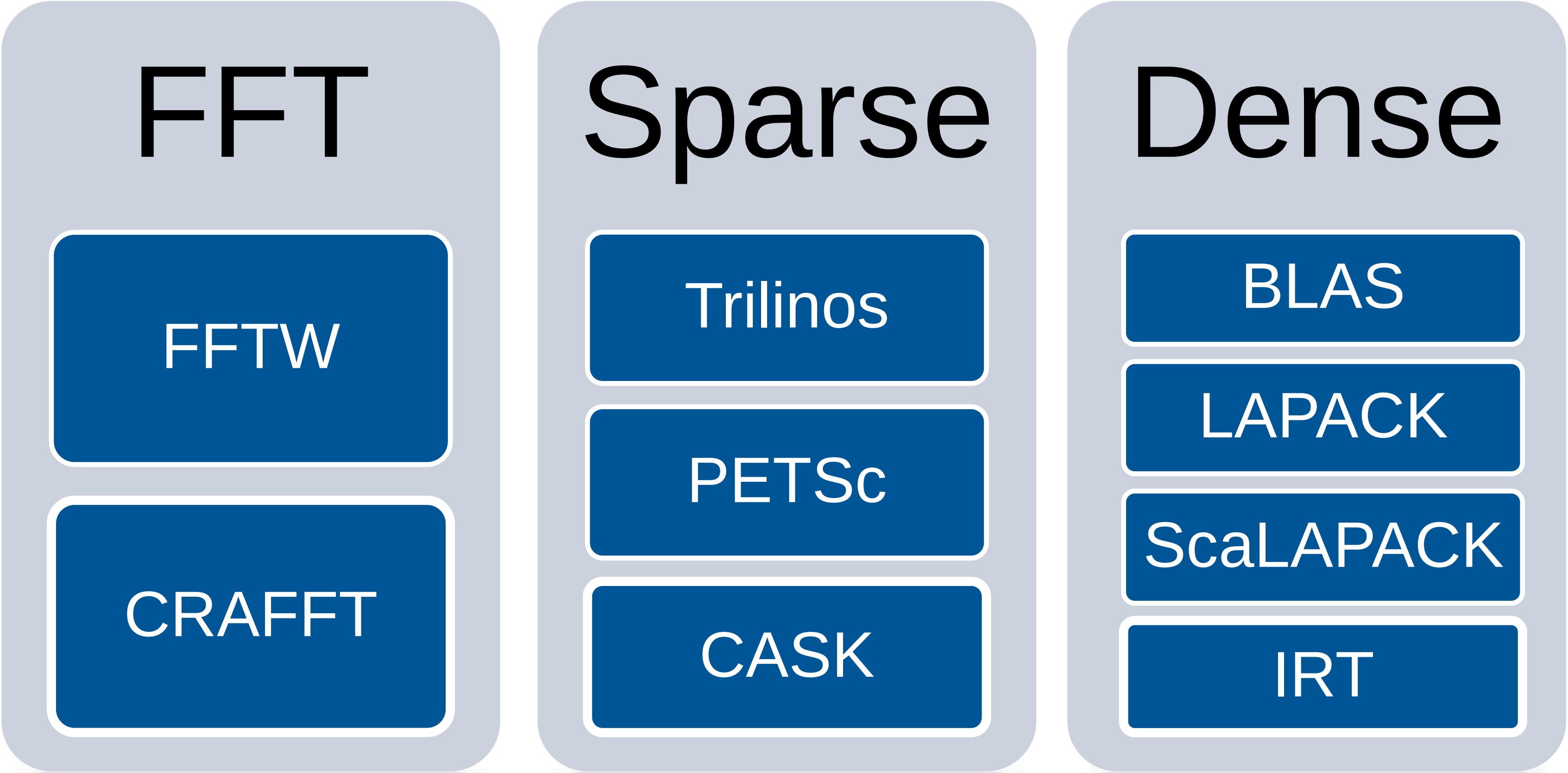
- **Building blocks for writing scientific applications**
- **Historically – allowed the first forms of code re-use**
- **Later – became ways of running optimized code**
- **These days the complexity of the hardware is very high**
- **Cray PE insulates the user from that complexity**
  - Cray module environment
  - CCE
  - Performance tools
  - Tuned MPI libraries (+PGAS)
  - Optimized Scientific libraries

Cray scientific libraries are designed to give maximum possible performance from Cray systems with minimum effort

What makes Cray libraries special

1. **Node performance**
   - Highly tune BLAS etc at the low-level
2. **Network performance**
   - Optimize for network performance
   - Overlap between communication and computation
   - Use the best available low-level mechanism
   - Use adaptive parallel algorithms
3. **Highly adaptive software**
   - Using auto-tuning and adaptation, give the user the known best (or very good) codes at runtime
4. **Productivity features**
   - Simpler interfaces into complex software

Scientific libraries – functional view



FFT

- FFTW
- CRAFFT

Sparse

- Trilinos
- PETSc
- CASK

Dense

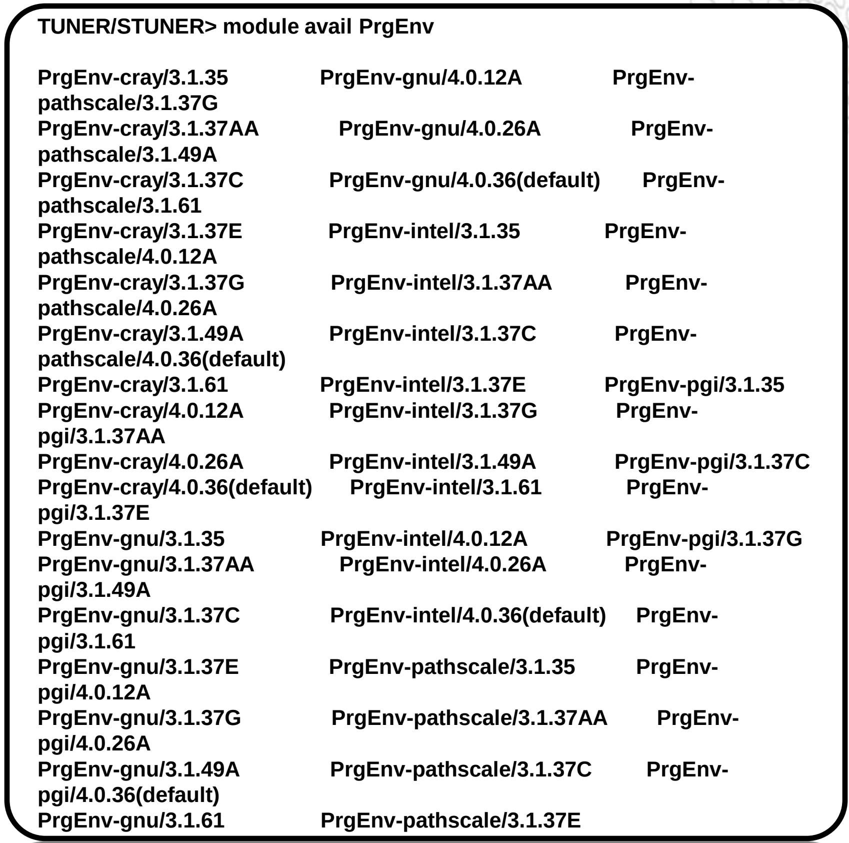
- BLAS
- LAPACK
- ScaLAPACK
- IRT

Libsci Usage all fits on one slide

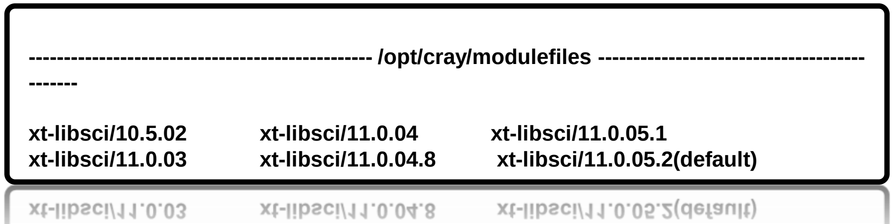
- **LIbSci**
  - The drivers should do it all for you. Don’t explicitly link.
  - For threads, set OMP_NUM_THREADS
    - Threading is used within libsci.
    - If you call within parallel region, single thread used
    - -Wl, -ydgemm_ reveals where the link was resolved
- **FFTW**
  - Module load fftw (there are also wisdom files you can pick up)
- **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**

Your friends

- module command (module --help)
- PrgEnv modules :
- Component modules
- csmlversion (tool)
- Cray driver scripts ftn, cc, CC



```
TUNER/STUNER> module avail PrgEnv

PrgEnv-cray/3.1.35            PrgEnv-gnu/4.0.12A            PrgEnv-
pathscale/3.1.37G
PrgEnv-cray/3.1.37AA            PrgEnv-gnu/4.0.26A            PrgEnv-
pathscale/3.1.49A
PrgEnv-cray/3.1.37C            PrgEnv-gnu/4.0.36(default)      PrgEnv-
pathscale/3.1.61
PrgEnv-cray/3.1.37E            PrgEnv-intel/3.1.35            PrgEnv-
pathscale/4.0.12A
PrgEnv-cray/3.1.37G            PrgEnv-intel/3.1.37AA          PrgEnv-
pathscale/4.0.26A
PrgEnv-cray/3.1.49A            PrgEnv-intel/3.1.37C           PrgEnv-
pathscale/4.0.36(default)
PrgEnv-cray/3.1.61            PrgEnv-intel/3.1.37E           PrgEnv-pgi/3.1.35
PrgEnv-cray/4.0.12A            PrgEnv-intel/3.1.37G           PrgEnv-
pgi/3.1.37AA
PrgEnv-cray/4.0.26A            PrgEnv-intel/3.1.49A           PrgEnv-pgi/3.1.37C
PrgEnv-cray/4.0.36(default)     PrgEnv-intel/3.1.61            PrgEnv-
pgi/3.1.37E
PrgEnv-gnu/3.1.35             PrgEnv-intel/4.0.12A           PrgEnv-pgi/3.1.37G
PrgEnv-gnu/3.1.37AA            PrgEnv-intel/4.0.26A           PrgEnv-
pgi/3.1.49A
PrgEnv-gnu/3.1.37C            PrgEnv-intel/4.0.36(default)    PrgEnv-
pgi/3.1.61
PrgEnv-gnu/3.1.37E            PrgEnv-pathscale/3.1.35         PrgEnv-
pgi/4.0.12A
PrgEnv-gnu/3.1.37G            PrgEnv-pathscale/3.1.37AA       PrgEnv-
pgi/4.0.26A
PrgEnv-gnu/3.1.49A            PrgEnv-pathscale/3.1.37C        PrgEnv-
pgi/4.0.36(default)
PrgEnv-gnu/3.1.61             PrgEnv-pathscale/3.1.37E
```



```
------------------------------------------------ /opt/cray/modulefiles -------------------------------------
-------

xt-libsci/10.5.02          xt-libsci/11.0.04          xt-libsci/11.0.05.1
xt-libsci/11.0.03          xt-libsci/11.0.04.8         xt-libsci/11.0.05.2(default)
```

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!)

Threading

- **LibSci is compatible with OpenMP**
- Control the number of threads to be used in your program using **OMP_NUM_THREADS**

  e.g. in job script

  setenv OMP_NUM_THREADS 16

  - Then run with aprun –n1 **–d16**
- **What behavior you get from the library depends on your code**
  1. No threading in code
     - The BLAS call will use OMP_NUM_THREADS threads
  2. Threaded code, outside parallel region
     - The BLAS call will use OMP_NUM_THREADS threads
  3. Threaded code, inside parallel region
     - The BLAS call will use a single thread

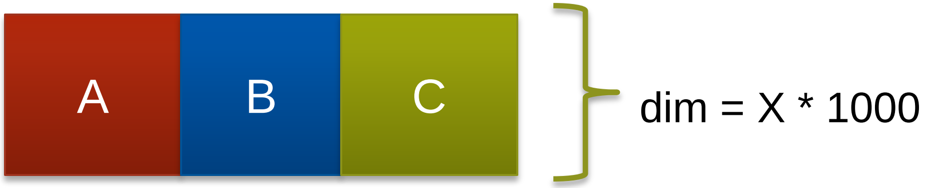
Emphasis

- **A large subset of HPC customers care very deeply about each of the following**
  - BLAS – explicit calls in their code
  - LAPACK – linear solvers
  - LAPACK – eigensolvers
  - ScaLAPACK
  - Serial FFT

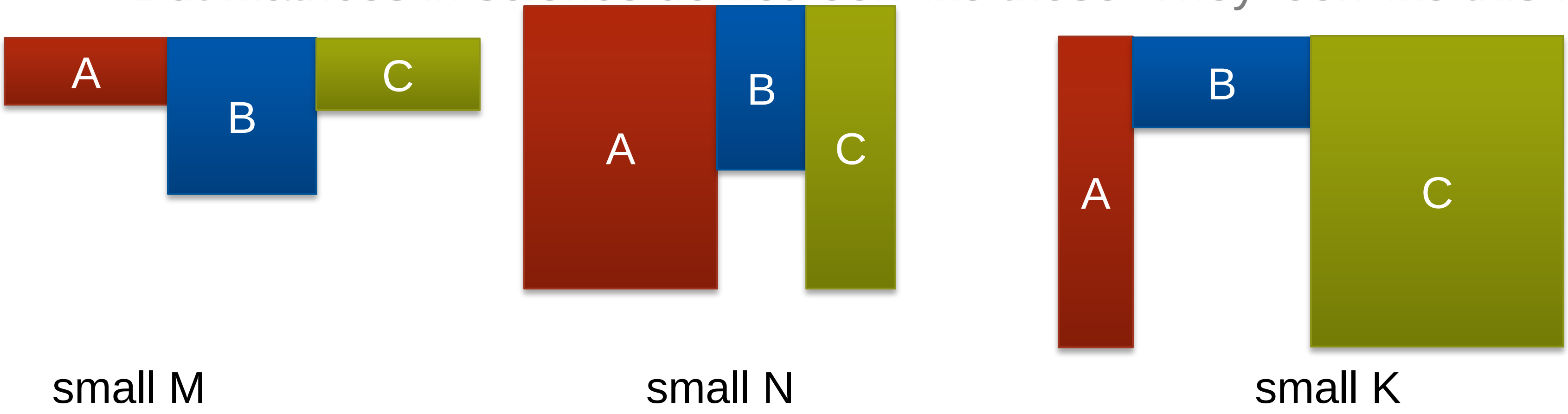
- **Our job is to make them work at extreme performance on Cray hardware**
- **A flaming-hot GEMM library can support wide usage**

But that is very hard

- Vendor libraries can be very heavily tuned for a specific problem.
- Tunings are not general, unfortunately.
- Your library is probably tuned for this



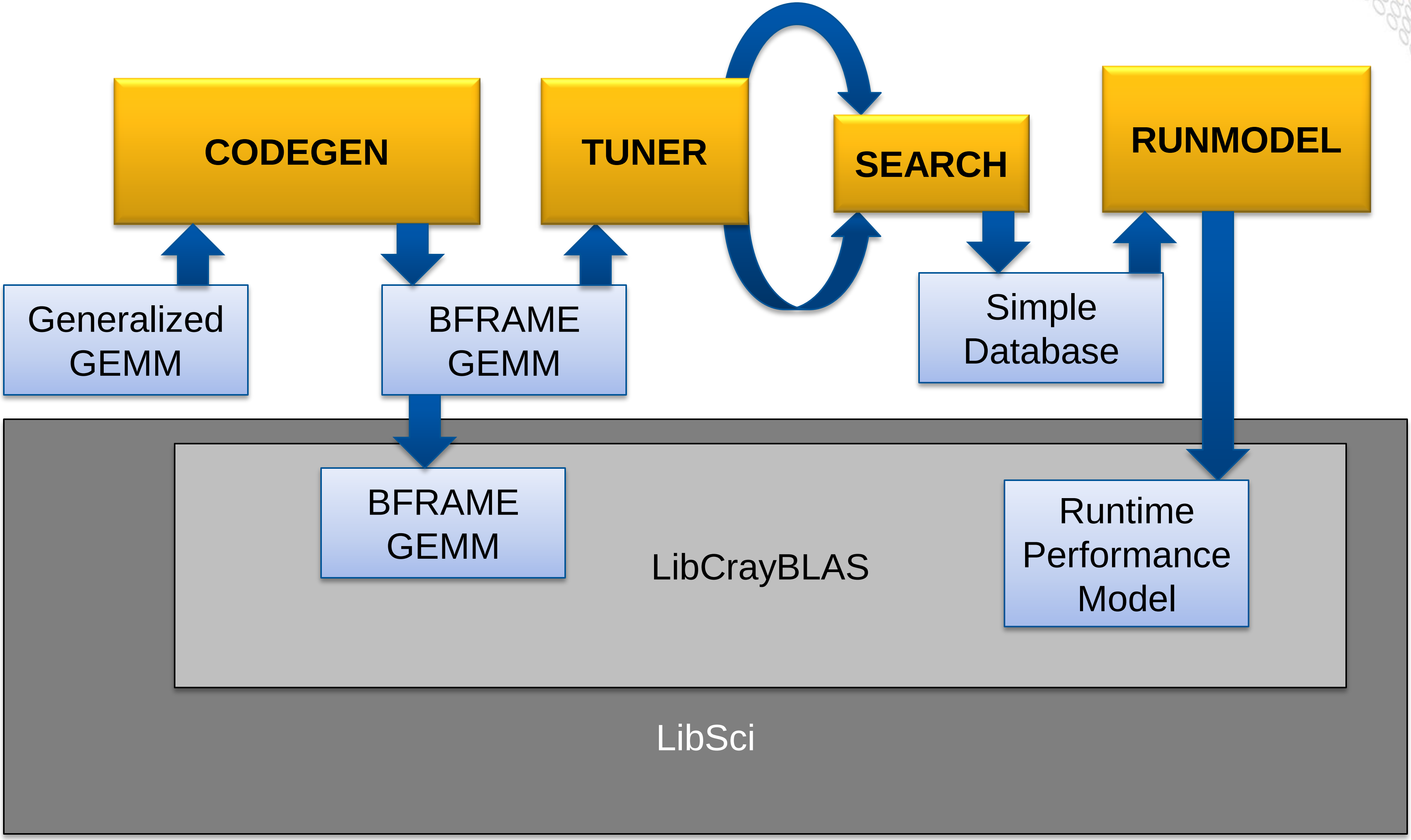
- But matrices in science do not look like those. They look like this :

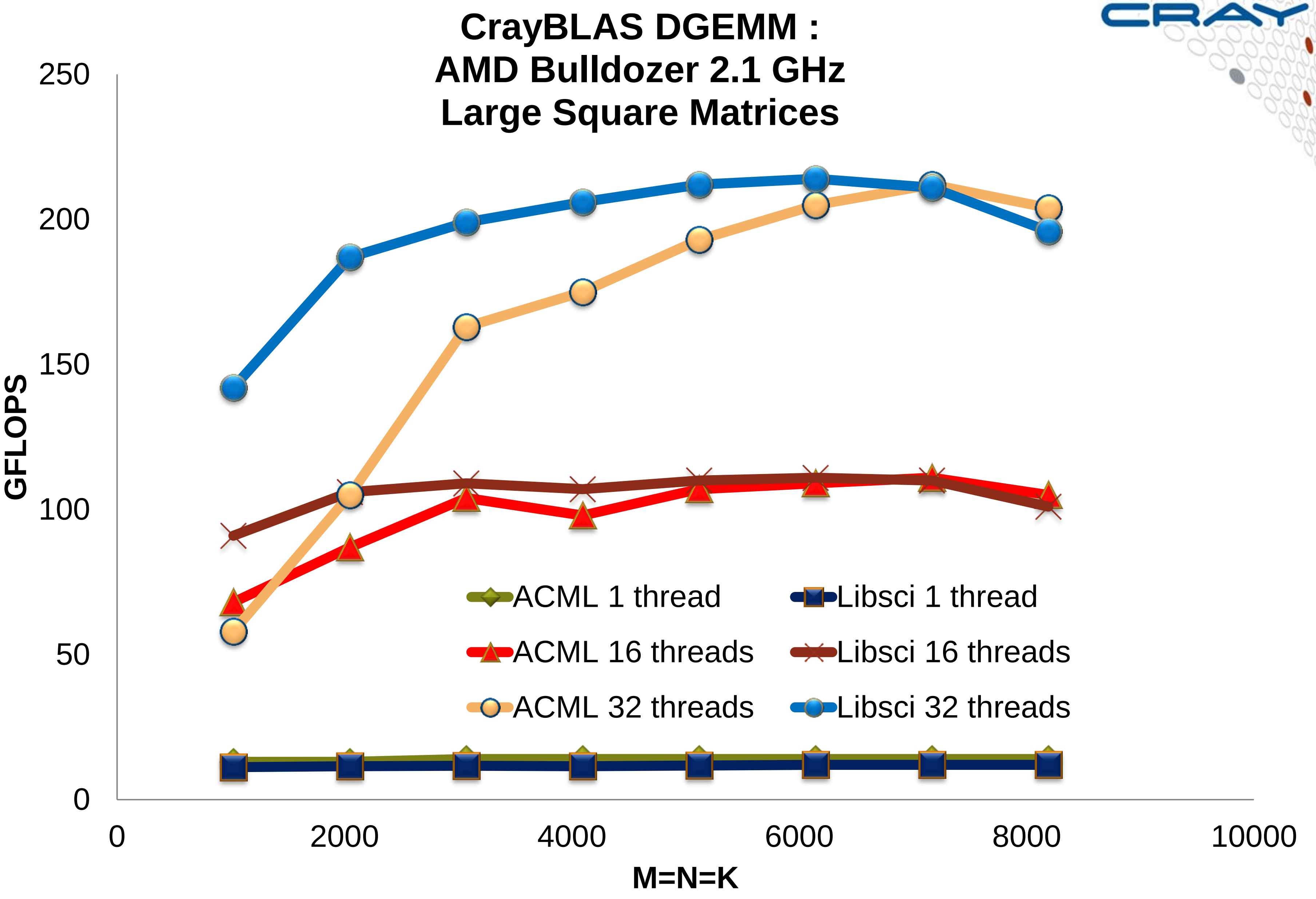


BFRAME / CrayBLAS

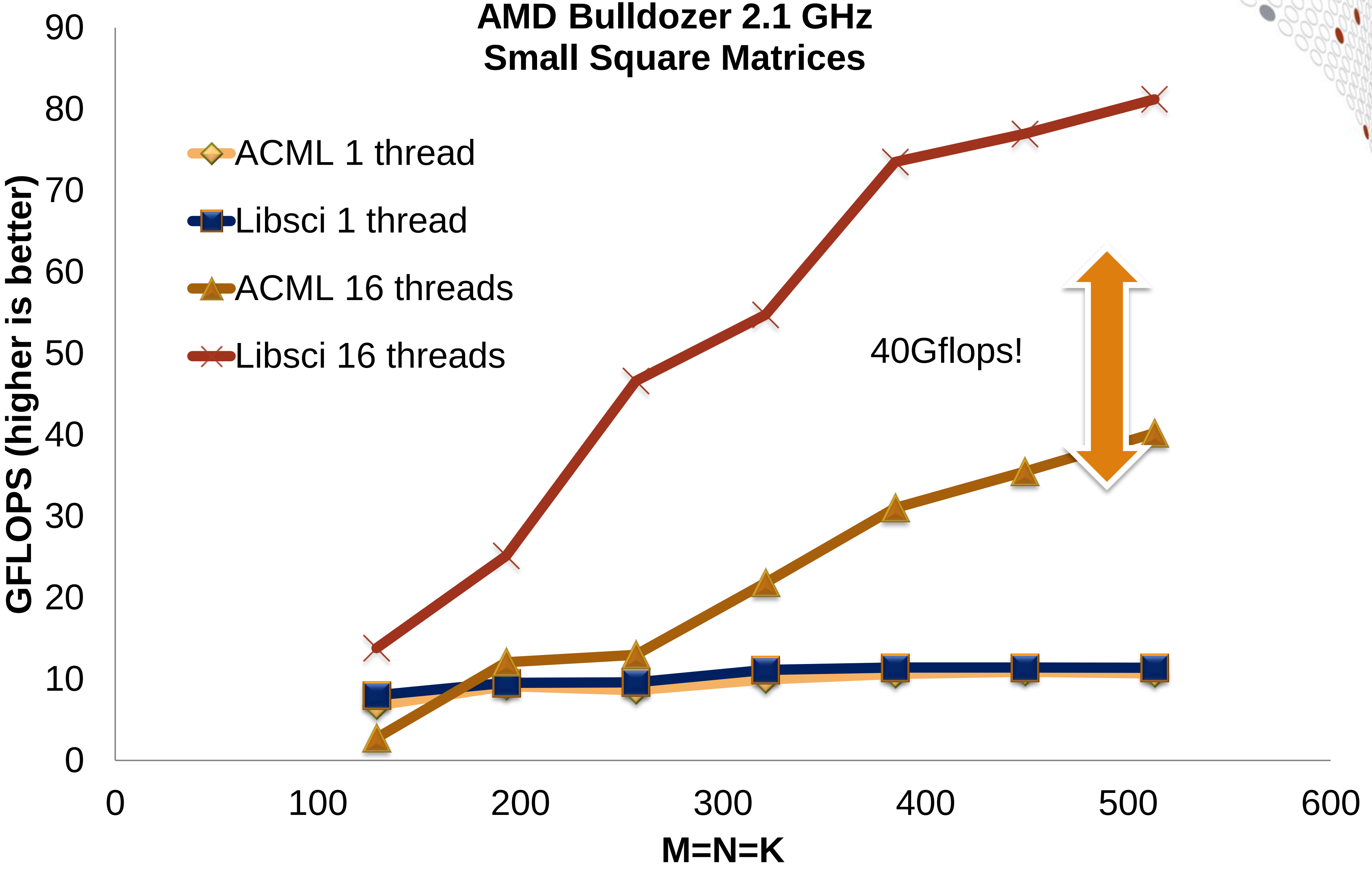
- **Goal**
  - Entire auto-tuning framework for BLAS on CPUS & GPUS
  - Must provide better performance than alternatives for “real” matrices
  - Improve threaded performance for any problem size on any number of threads
  - Encapsulate all tunings into a run-time performance model
- **Design :**
  1. Generalized GEMM implementation
  2. Offline Auto tuning
  3. Runtime performance modeling

BFRAME infrastructure

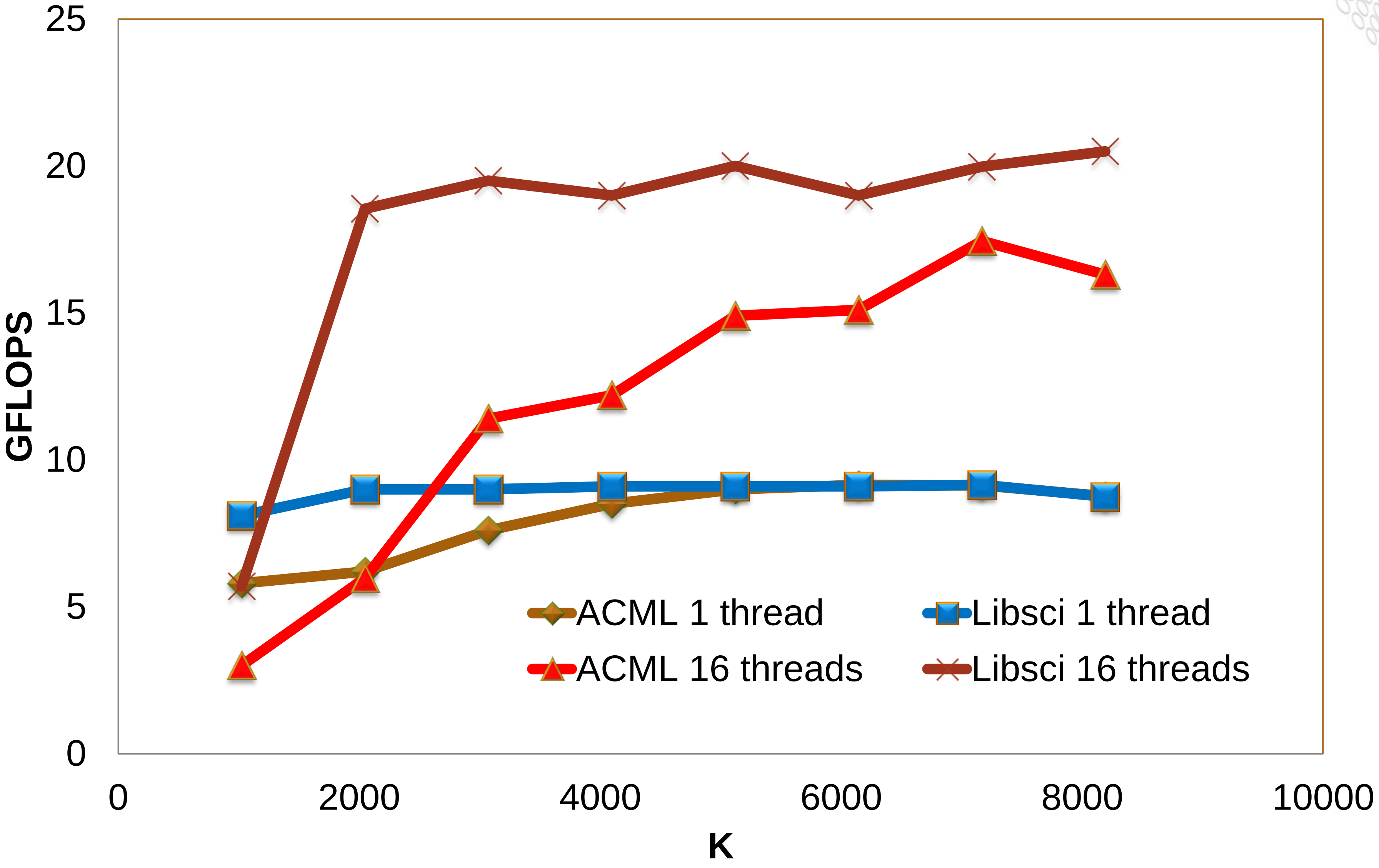



CrayBLAS DGEMM :
AMD Bulldozer 2.1 GHz
Large Square Matrices
GFLOPS
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
M=N=K
ACML 1 thread
Libsci 1 thread
ACML 16 threads
Libsci 16 threads
ACML 32 threads
Libsci 32 threads

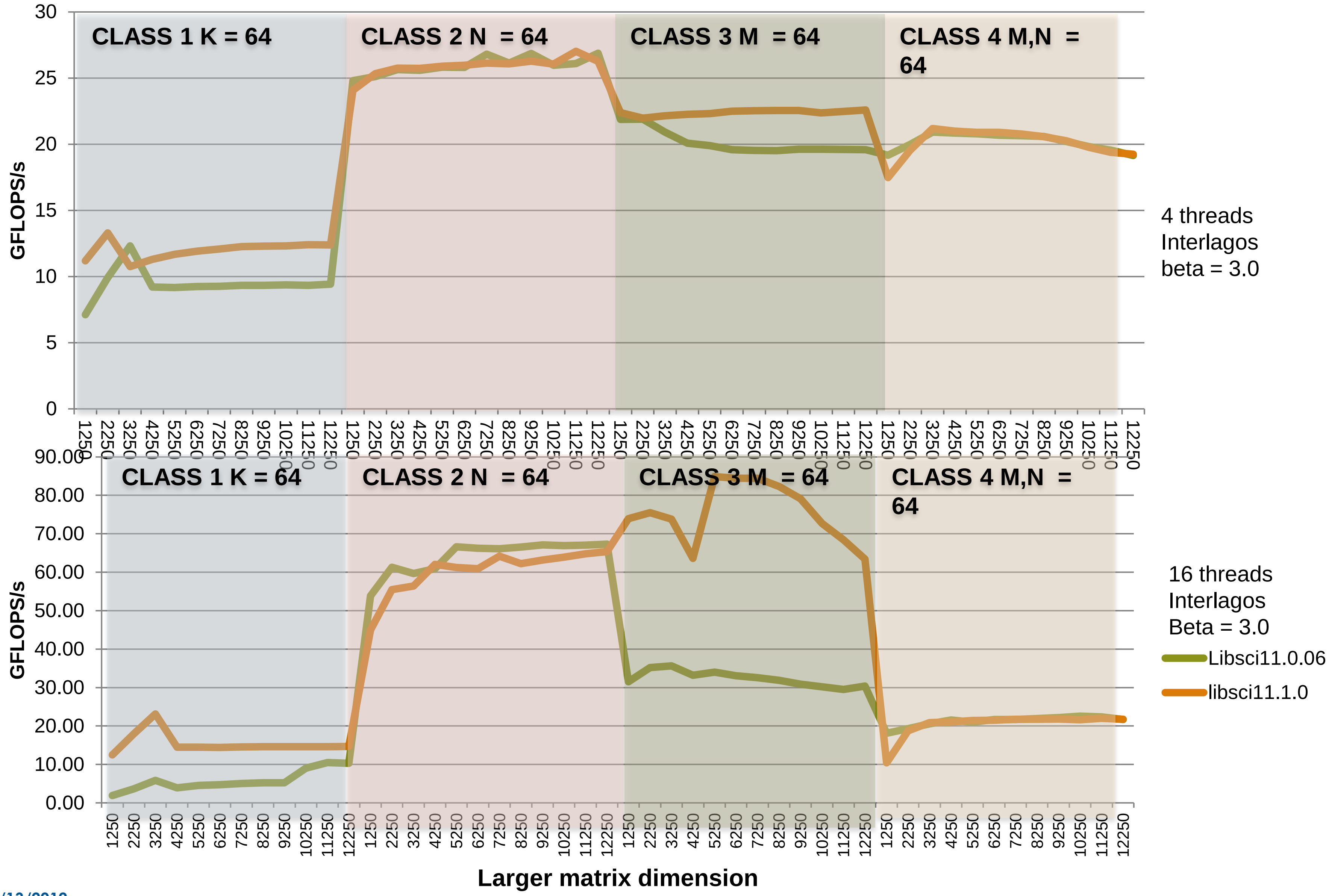
CrayBLAS DGEMM :
AMD Bulldozer 2.1 GHz
Small Square Matrices



CrayBLAS DGEMM :
AMD Bulldozer 2.1 GHzM :
Varying K dimension



DGEMM Performance across Matrix classes



4 threads
Interlagos
beta = 3.0

16 threads
Interlagos
Beta = 3.0

Tuning requests

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

crayblas@cray.com

- Just send the routine name, and the list of calling sequences

Advanced optimizations for BLAS

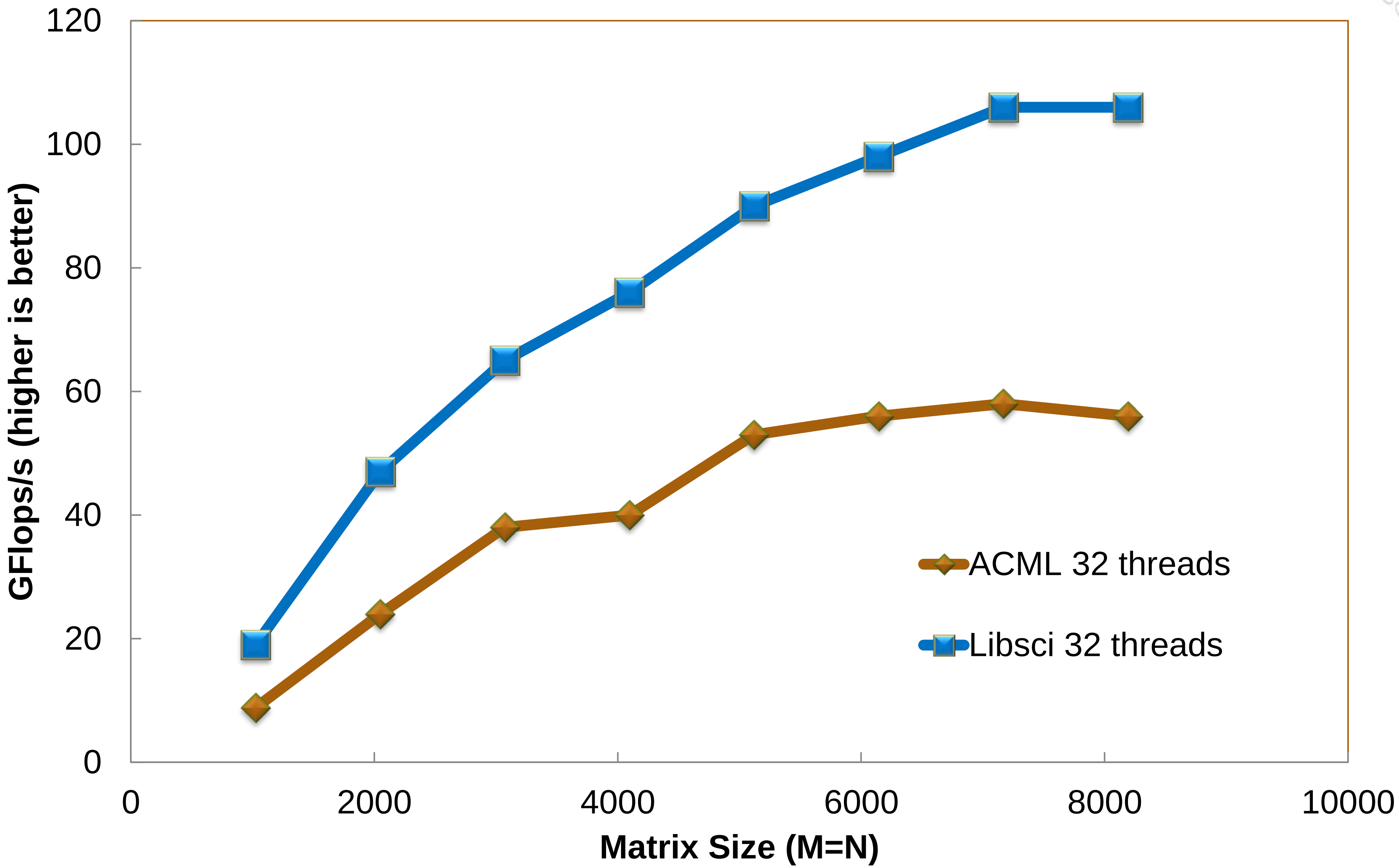
- **For ZGEMM only**
  - Complex matrix multiplication can be performed using real matrix additions, for fewer flops
  - You can turn on the 3M algorithm
  - Set the environment variable

    ZGEMM_USE_3M=1
  - Note : there is an accuracy trade-off, though this should be safe most of the time

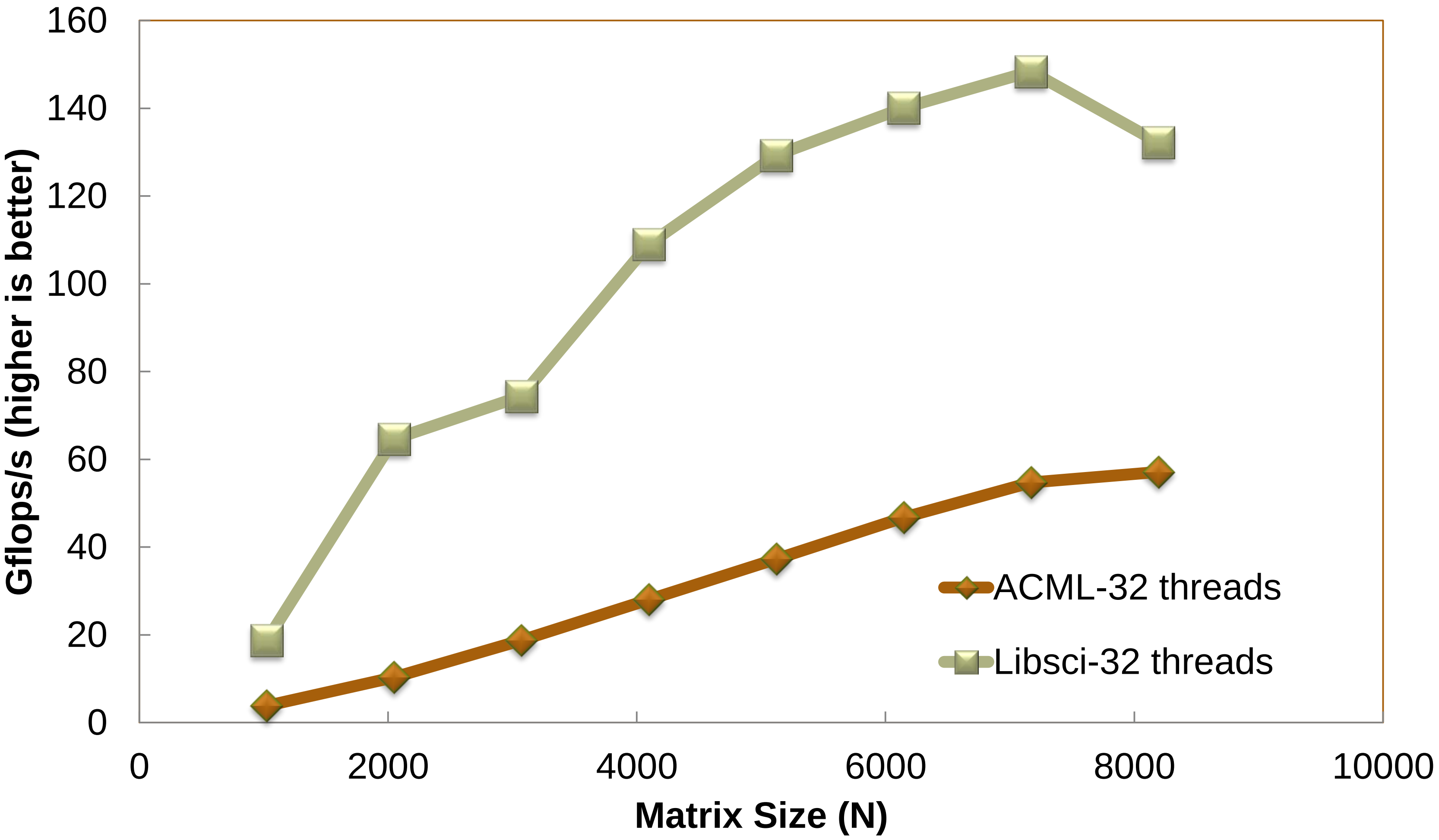
Threaded LAPACK

- Threaded LAPACK works exactly the same as threaded BLAS
- Anywhere LAPACK uses BLAS, those BLAS can be threaded
- Some LAPACK routines are threaded at the higher level
- No special instructions

LAPACK DGETRF (LU)
AMD Bulldozer 2.1Ghz ::July 2012



LAPACK DPOTRF (Cholesky)
AMD Bulldozer 2.1Ghz ::Aug 2012

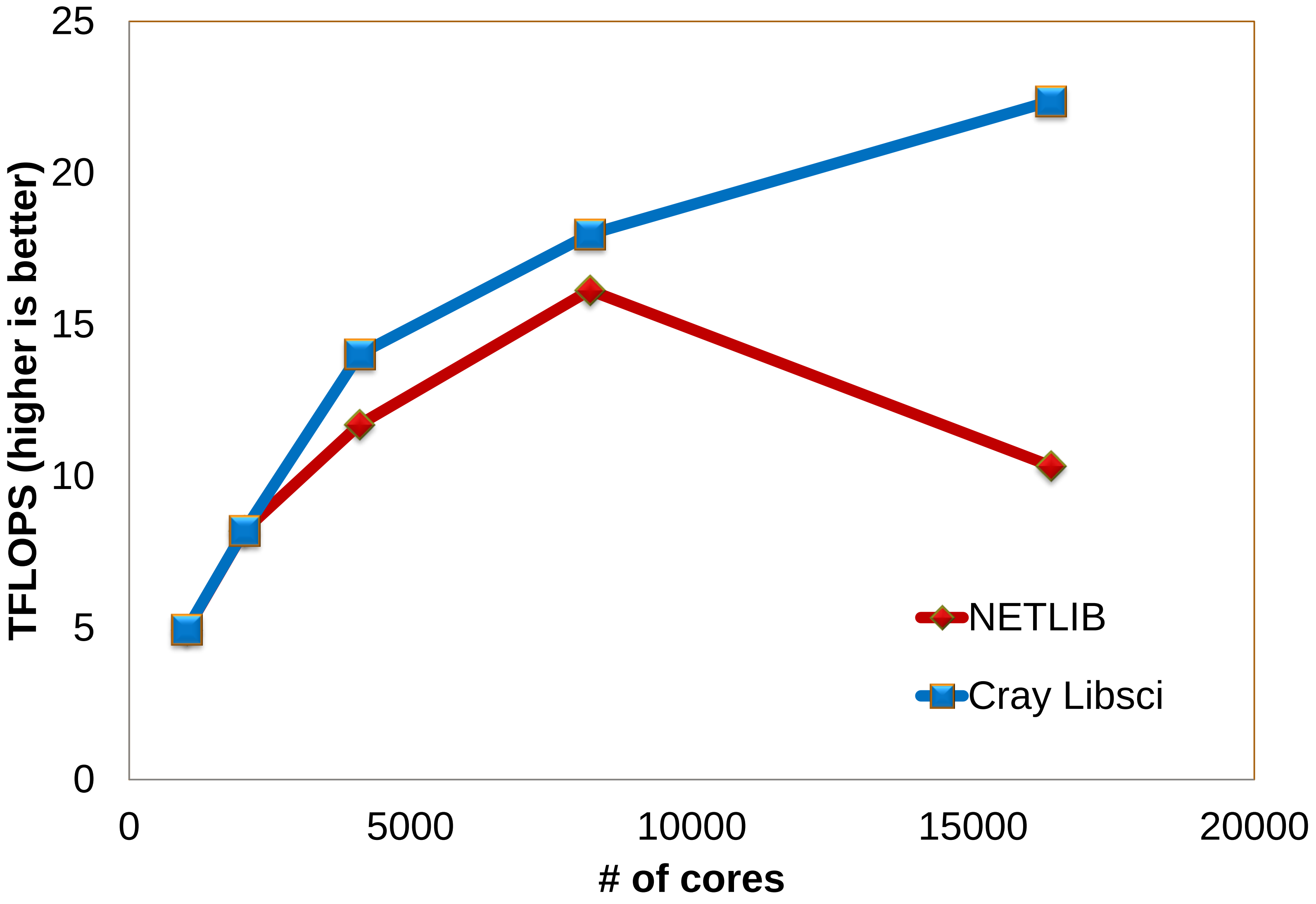


ScaLAPACK

- **ScaLAPACK in libsci is optimized for Gemini interconnect**
  - New collective communication procedures are added
  - Default topologies are changed to use the new optimizations
  - Much better strong scaling
- **It also benefits from the optimizations in CrayBLAS**
- **IRT can provide further improvements (see later)**

ScaLAPACK Double LU

Cray XE6 :: AMD Bulldozer 2.1 GHz
M = 131072 :: July 2012



Iterative Refinement Toolkit

- **Mixed precision can yield a big win on x86 machines.**
- **SSE (and AVX) units issue double the number of single precision operations per cycle.**
- **On CPU, single precision is always 2x as fast as double**
- **Accelerators sometimes have a bigger ratio**
  - Cell – 10x
  - Older NVIDIA cards – 7x
  - New NVIDIA cards (2x)
  - Newer AMD cards (> 2x)
- **IRT is a suite of tools to help exploit single precision**
  - A library for direct solvers
  - An automatic framework to use mixed precision under the covers

Iterative Refinement Toolkit - Library

- **Various tools for solves linear systems in mixed precision**
- **Obtaining solutions accurate to double precision**
  - For well conditioned problems
- **Serial and Parallel versions of LU, Cholesky, and QR**
- **2 usage methods**
  - **IRT Benchmark routines**
    - Uses IRT 'under-the-covers' without changing your code
      - Simply set an environment variable
      - Useful when you cannot alter source code
  - **Advanced IRT API**
    - If greater control of the iterative refinement process is required
      - Allows
        - condition number estimation
        - error bounds return
        - minimization of either forward or backward error
        - 'fall back' to full precision if the condition number is too high
        - max number of iterations can be altered by users

IRT library usage

Decide if you want to use advanced API or benchmark API

benchmark API :

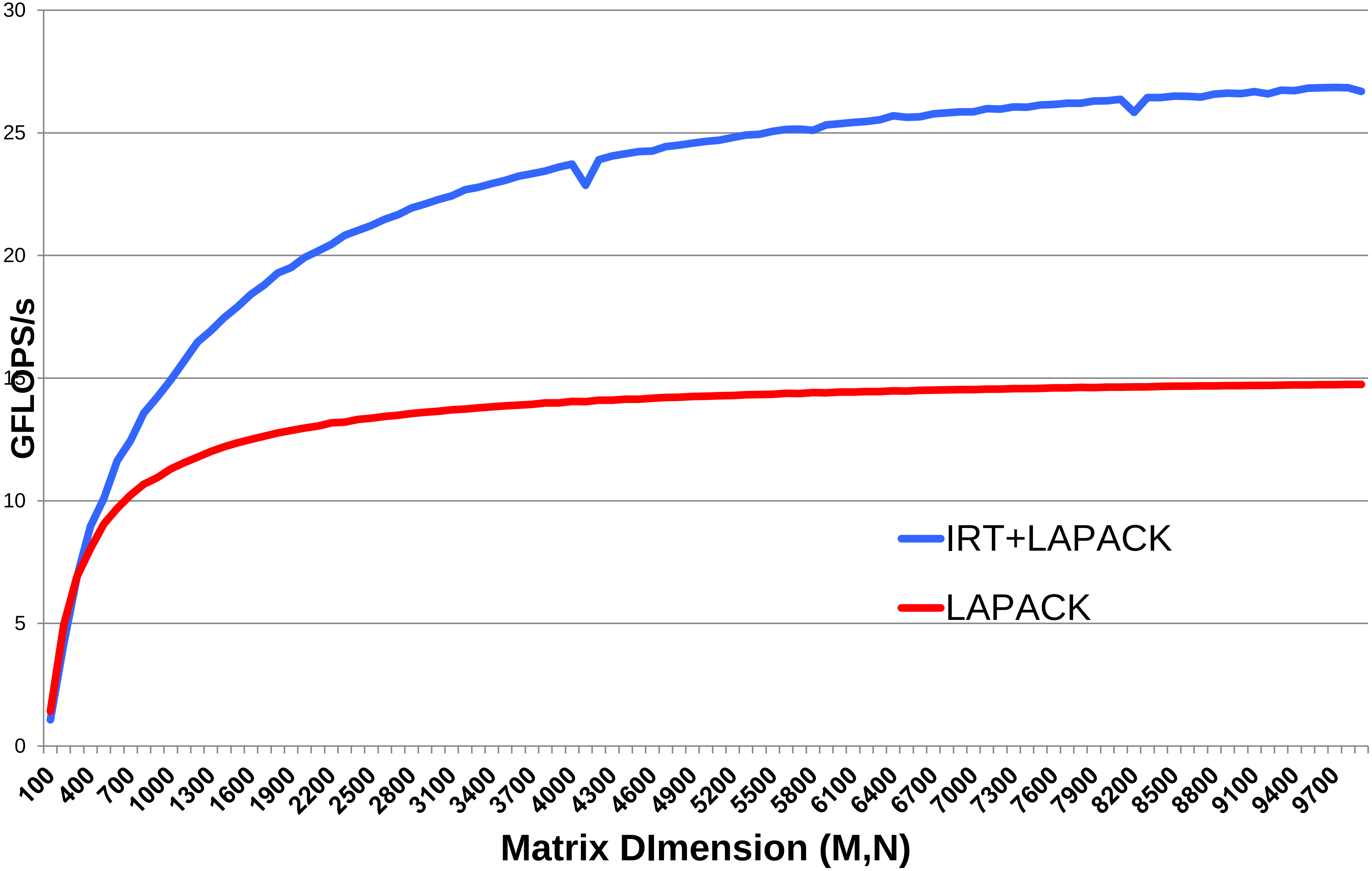
setenv IRT_USE_SOLVERS 1

advanced API :

1. **locate the factor and solve in your code (LAPACK or ScaLAPACK)**
2. **Replace factor and solve with a call to IRT routine**
   - e.g. dgesv -> irt_lu_real_serial
   - e.g. pzgesv -> irt_lu_complex_parallel
   - e.g pzposv -> irt_po_complex_parallel
3. **Set advanced arguments**
   - Forward error convergence for most accurate solution
   - Condition number estimate
   - “fall-back” to full precision if condition number too high

Note : **“info” does not return zero when using IRT !!**

IRT with LAPACK LU DGETRF
AMD Bulldozer 2.1 GHz
2threads :: September 2012



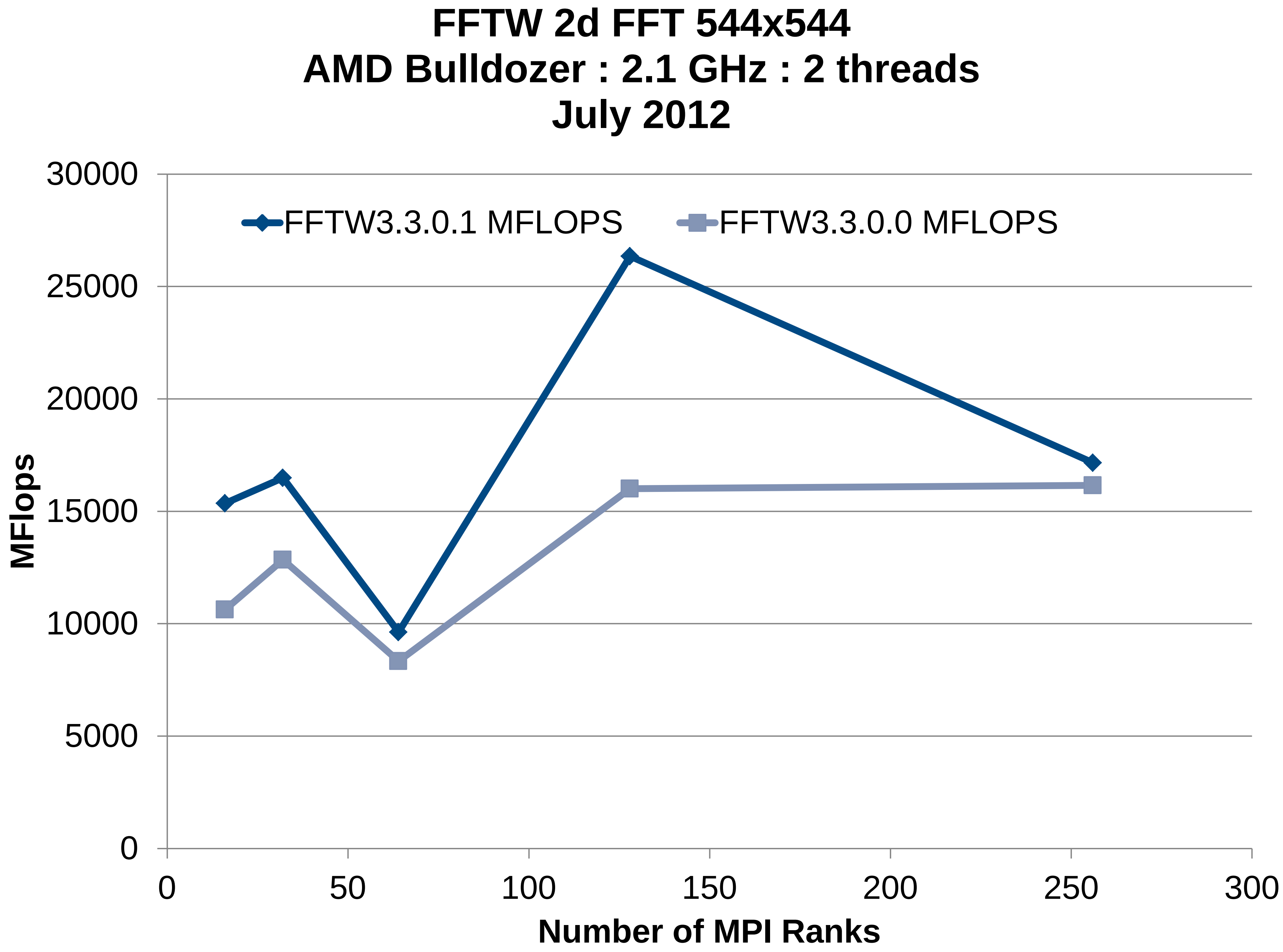
FFTW

- Cray’s main FFT library is FFTW from MIT
  - Some additional optimizations for Cray hardware

- Usage is simple
  - Load the module
  - In the code, call an FFTW plan

- Cray’s FFTW provides wisdom files for these systems
- You can use the wisdom files to skip the plan stage
- This can be a significant performance boost

- FFTW 3.1.0.1 includes Cray optimizations for IL processors


FFTW 2d FFT 544x544
AMD Bulldozer : 2.1 GHz : 2 threads
July 2012
FFTW3.3.0.1 MFLOPS
FFTW3.3.0.0 MFLOPS
MFlops
30000
25000
20000
15000
10000
5000
0
0
50
100
150
200
250
300
Number of MPI Ranks

Cray Adaptive FFT (CRAFFT)

- **Serial CRAFFT is largely a productivity enhancer**
- **Also a performance boost due to “wisdom” usage**
- **Some FFT developers have problems such as**
  - Which library choice to use?
  - How to use complicated interfaces (e.g., FFTW)
- **Standard FFT practice**
  - Do a plan stage
  - Do an execute
- **CRAFFT is designed with simple-to-use interfaces**
  - Planning and execution stage can be combined into one function call
  - Underneath the interfaces, CRAFFT calls the appropriate FFT kernel

CRAFFT usage

1. Load module fftw/3.2.0 or higher.
2. Add a Fortran statement “use crafft”
3. call crafft_init()
4. Call crafft transform using none, some or all optional arguments (as shown in red)

In-place, implicit memory management :

```
call crafft_z2z3d(n1,n2,n3,input,ld_in,ld_in2,isign)
```

in-place, explicit memory management

```
call crafft_z2z3d(n1,n2,n3,input,ld_in,ld_in2,isign,work)
```

out-of-place, explicit memory management :

```
crafft_z2z3d(n1,n2,n3,input,ld_in,ld_in2,output,ld_out,ld_out2,isign,work)
```

Note : the user can also control the planning strategy of CRAFFT using the CRAFFT_PLANNING environment variable and the do_exe optional argument, please see the intro_crafft man page.

Parallel CRAFFT

- **Parallel CRAFFT is meant as a performance improvement to FFTW2 distributed transforms**
  - Uses FFTW3 for the serial transform
  - Uses ALLTOALL instead of ALLTOALLV where possible
  - Overlaps the local transpose with the parallel communications
  - Uses a more adaptive communication scheme based on input
  - Lots of more advanced research in one-sided messaging and active messages
- **Can provide impressive performance improvements over FFTW2**
- **Currently implemented**
  - complex-complex
  - Real-complex and complex-real
  - 3-d and 2-d
  - In-place and out-of-place
  - 1 data distribution scheme but looking to support more (please tell us)
  - C language support for serial and parallel
  - Generic interfaces for C users (use C++ compiler to get these)

parallel CRAFFT usage

1. Add “use crafft” to Fortran code
2. Initialize CRAFFT using crafft_init
3. Assume MPI initialized and data distributed (see manpage)
4. Call crafft, e.g. (optional arguments in red)

   2-d complex-complex, in-place, internal mem management :

   ```
   call crafft_pz2z2d(n1,n2,input,isign,flag,comm)
   ```

   2-d complex-complex, in-place with no internal memory :

   ```
   call crafft_pz2z2d(n1,n2,input,isign,flag,comm,work)
   ```

   2-d complex-complex, out-of-place, internal mem manager :

   ```
   call crafft_pz2z2d(n1,n2,input,output,isign,flag,comm)
   ```

   2-d complex-complex, out-of-place, no internal memory :

   ```
   crafft_pz2z2d(n1,n2,input,output,isign,flag,comm,work)
   ```

Each routine above has manpage. Also see 3d equivalent :

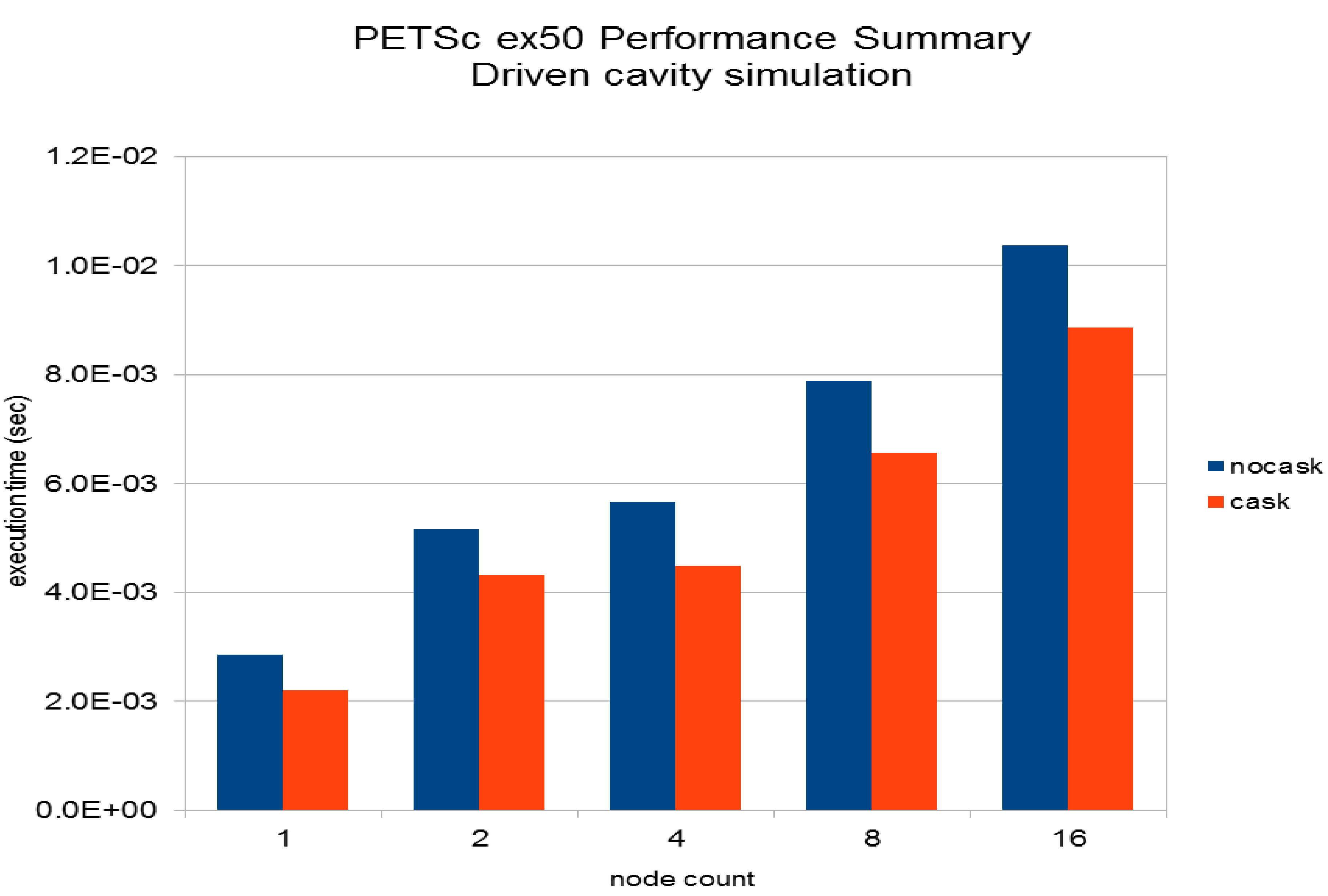
```
man crafft_pz2z3d
```

Cray Adaptive Sparse Kernel (CASK)

- **Sparse matrix operations in PETSc and Trilinos on Cray systems are optimized via CASK**

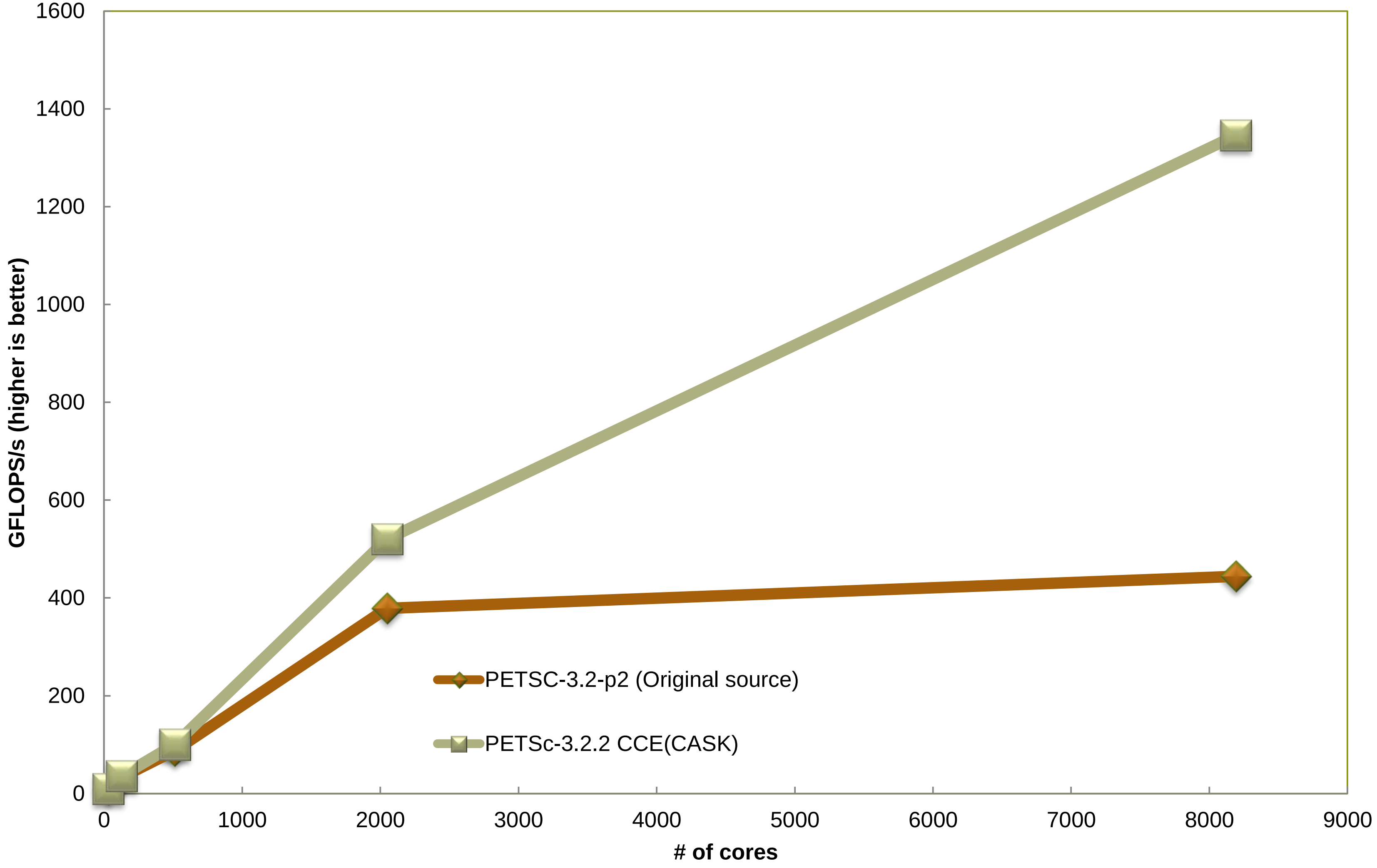
- **CASK is a product developed at Cray using the Cray Auto-tuning Framework**
- **Offline :**
  - ATF program builds many thousands of sparse kernel
  - Testing program defines matrix categories based on density, dimension etc
  - Each kernel variant is tested against each matrix class
  - Performance 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

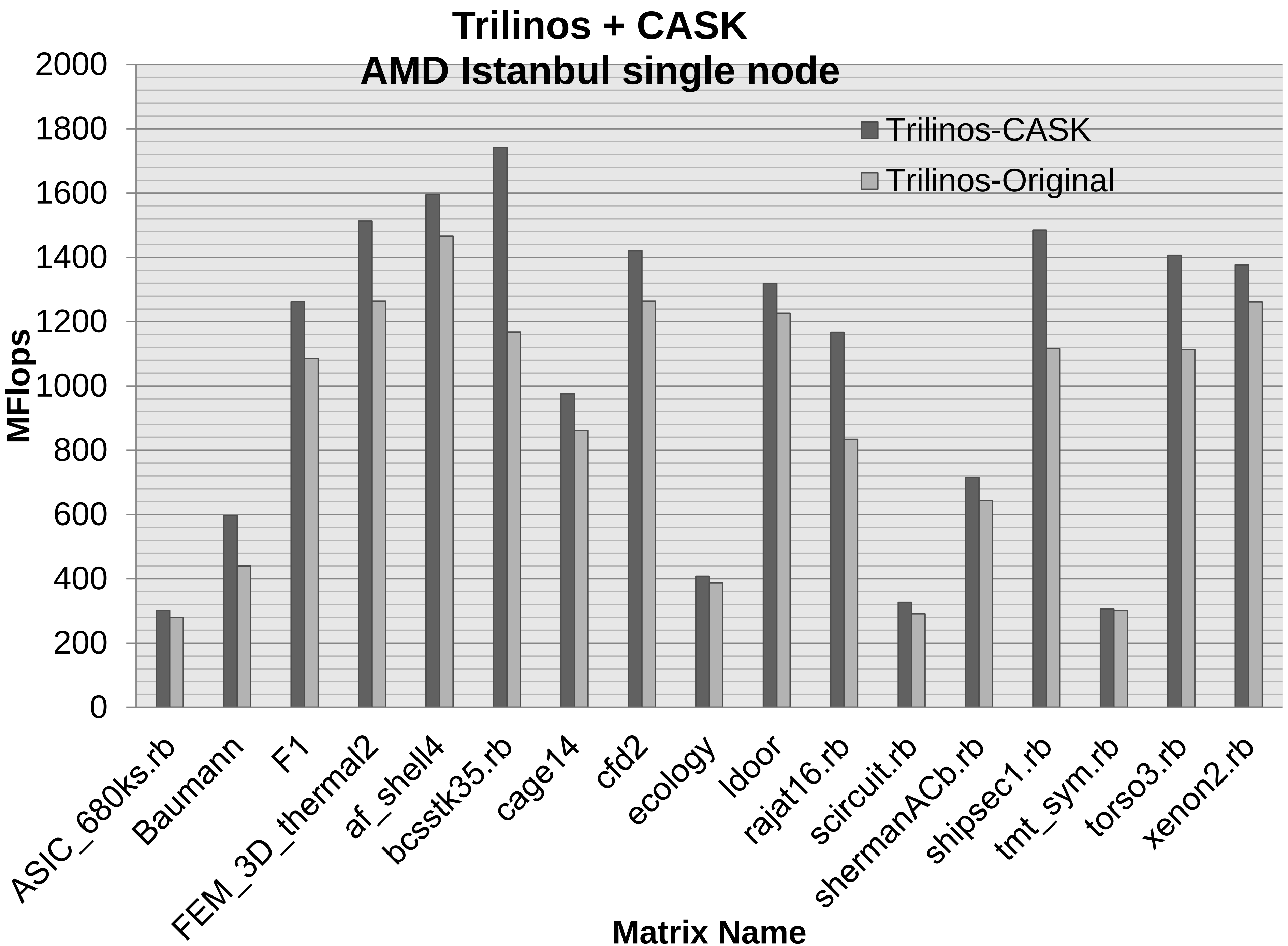
CASK + PETSc AMD IL



PETSc, Linear System Solution

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




Trilinos + CASK
AMD Istanbul single node
MFlops
2000
1800
1600
1400
1200
1000
800
600
400
200
0
Trilinos-CASK
Trilinos-Original
ASIC_680ks.rb
Baumann
F1
FEM_3D_thermal2
af_shell4
bcsstk35.rb
cage14
cfd2
ecology
ldoor
rajat16.rb
scircuit.rb
shermanACb.rb
shipsec1.rb
tmt_sym.rb
torso3.rb
xenon2.rb
Matrix Name

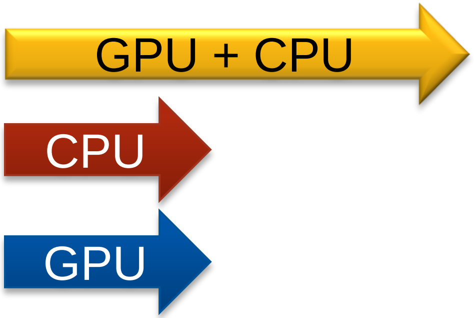
LibSci for Accelerators

- **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!**

Three interfaces for three use cases

- Simple interface

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

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


GPU + CPU
CPU
GPU

- Device interface

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

- CPU interface

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

Usage - Basics

- **Supports Cray and GNU compilers.**
- **Fortran and C interfaces (column-major assumed)**
  - Load the module ***craype-accel-nvidia20.***
  - 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 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 Example

- 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.

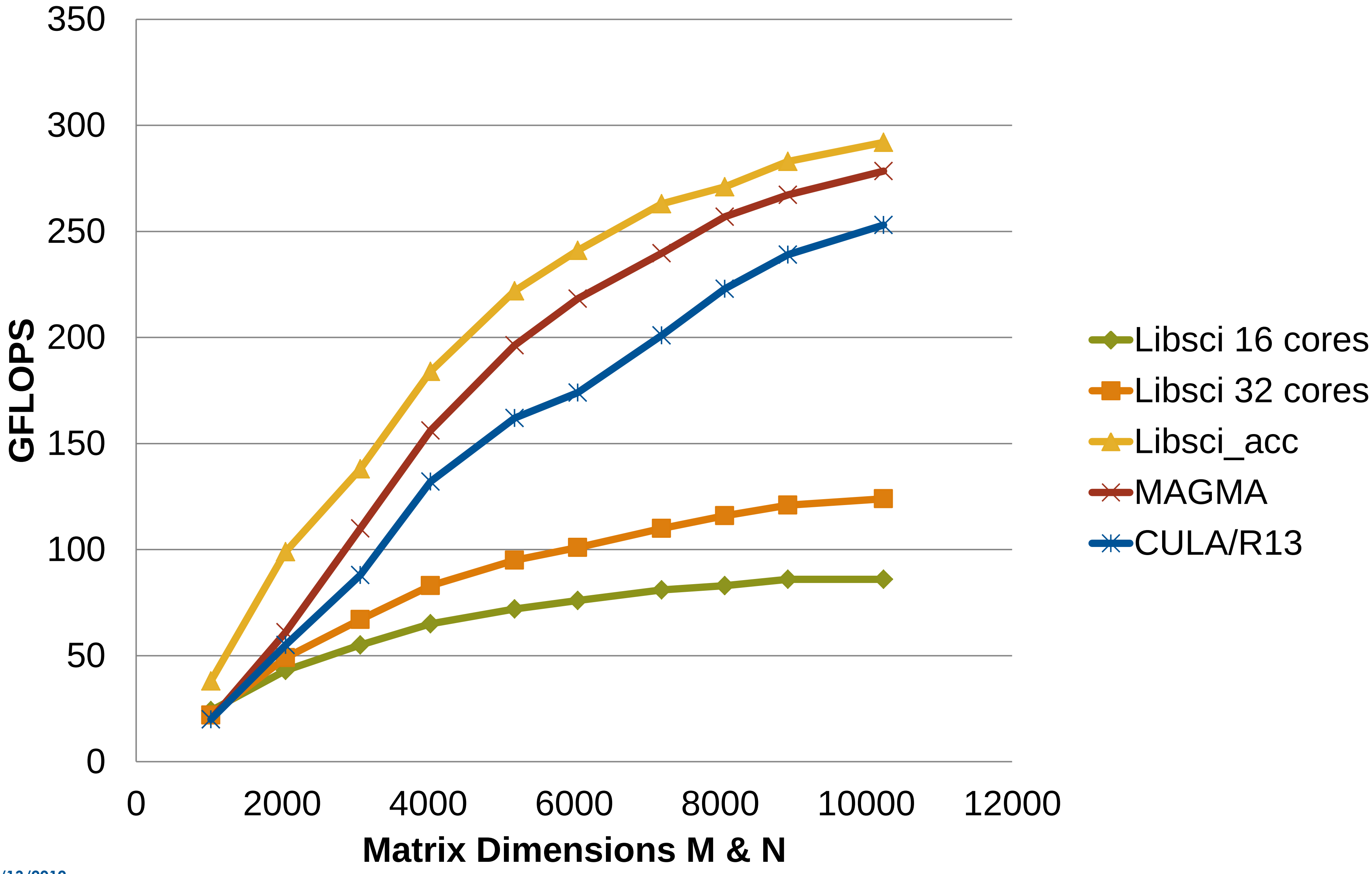
```
!$acc data copy(a,b,c)

!$acc parallel
!Do Something
!$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
```

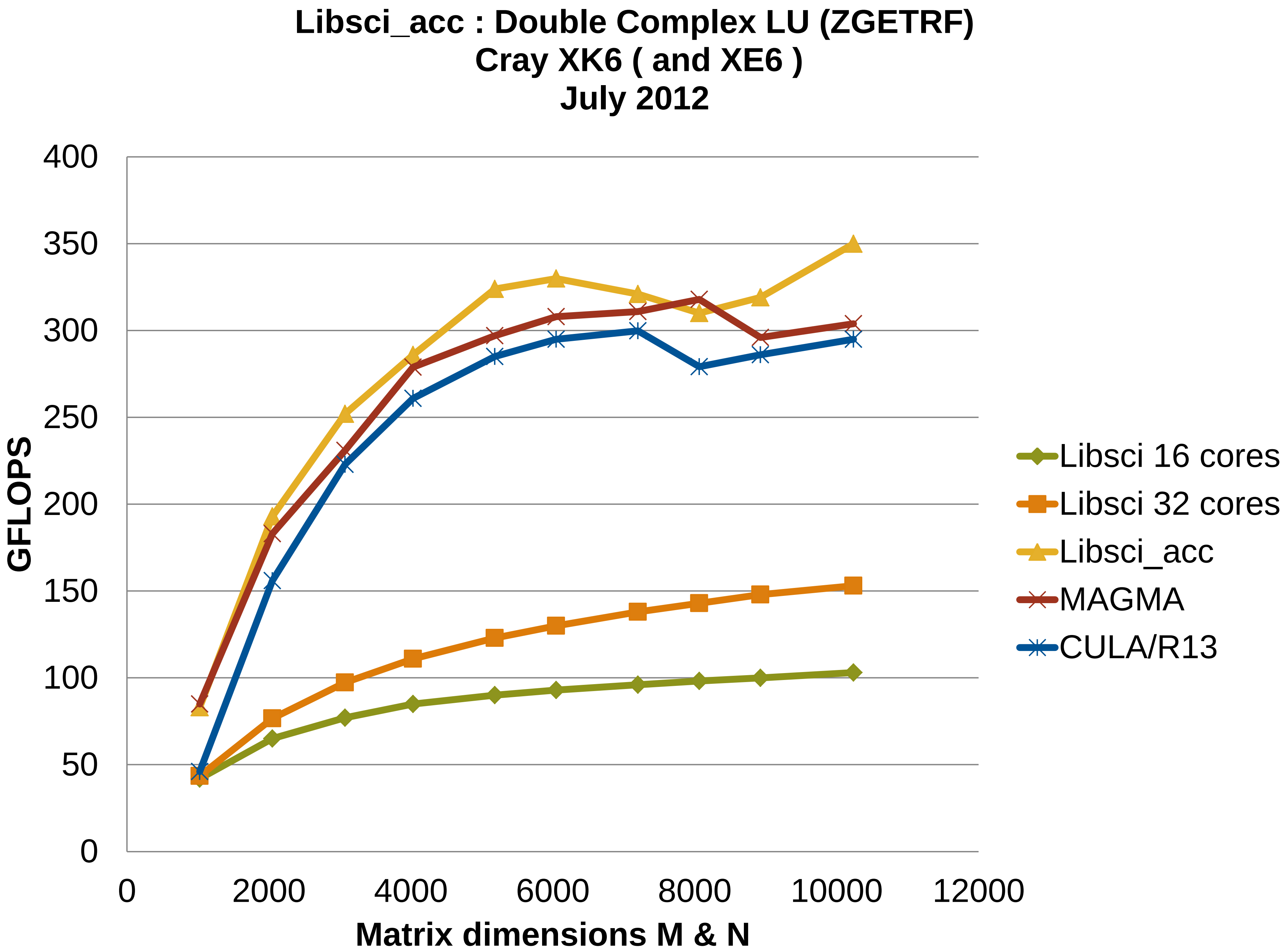
Libsci_acc Example

- 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.

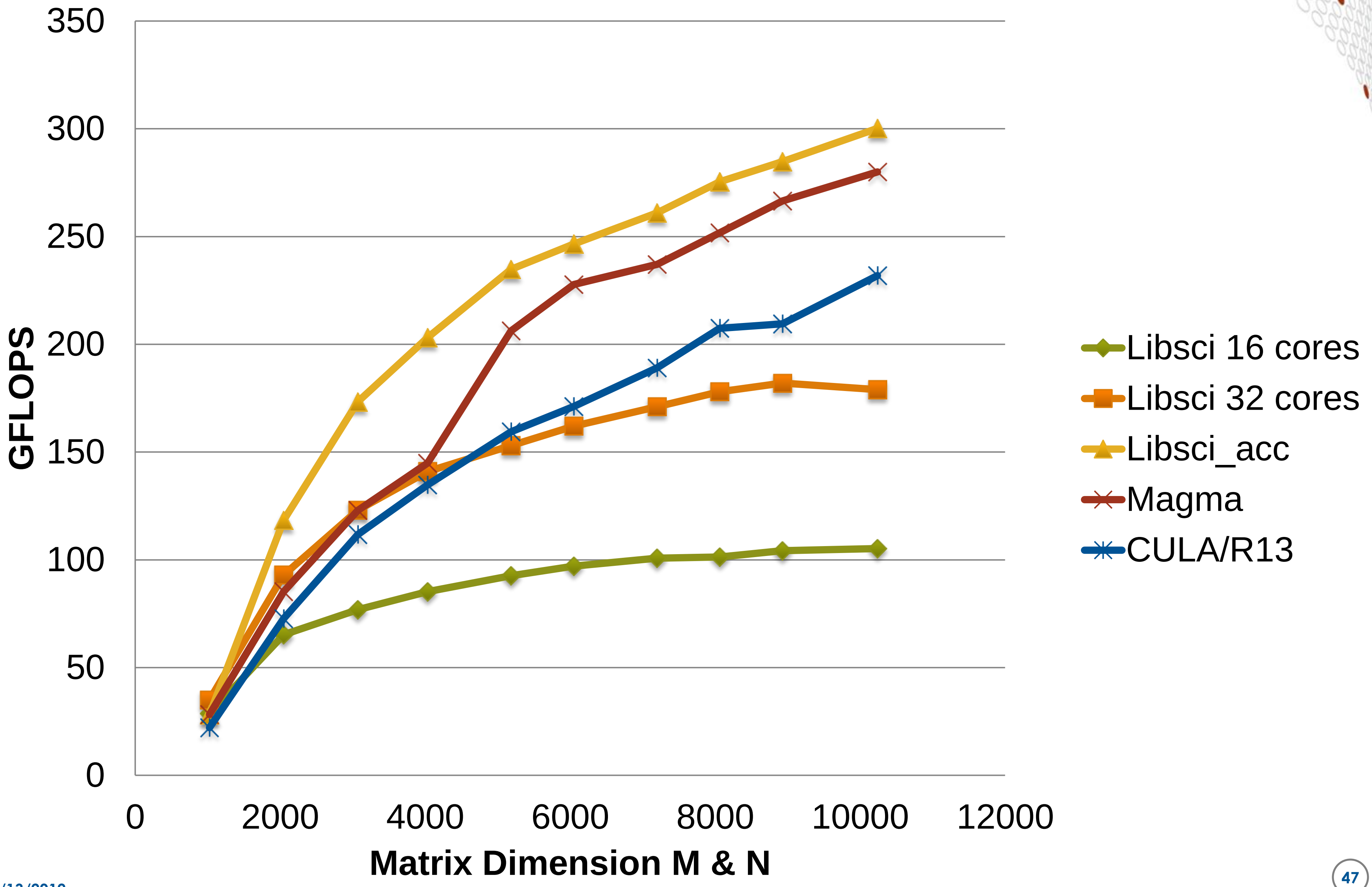
```
!$acc data copy(a,b,c)

!$acc parallel
!Do Something
!$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
```

Libsci_acc : Double : LU (DGETRF)
Cray XK6 (and XE6)
July 2012

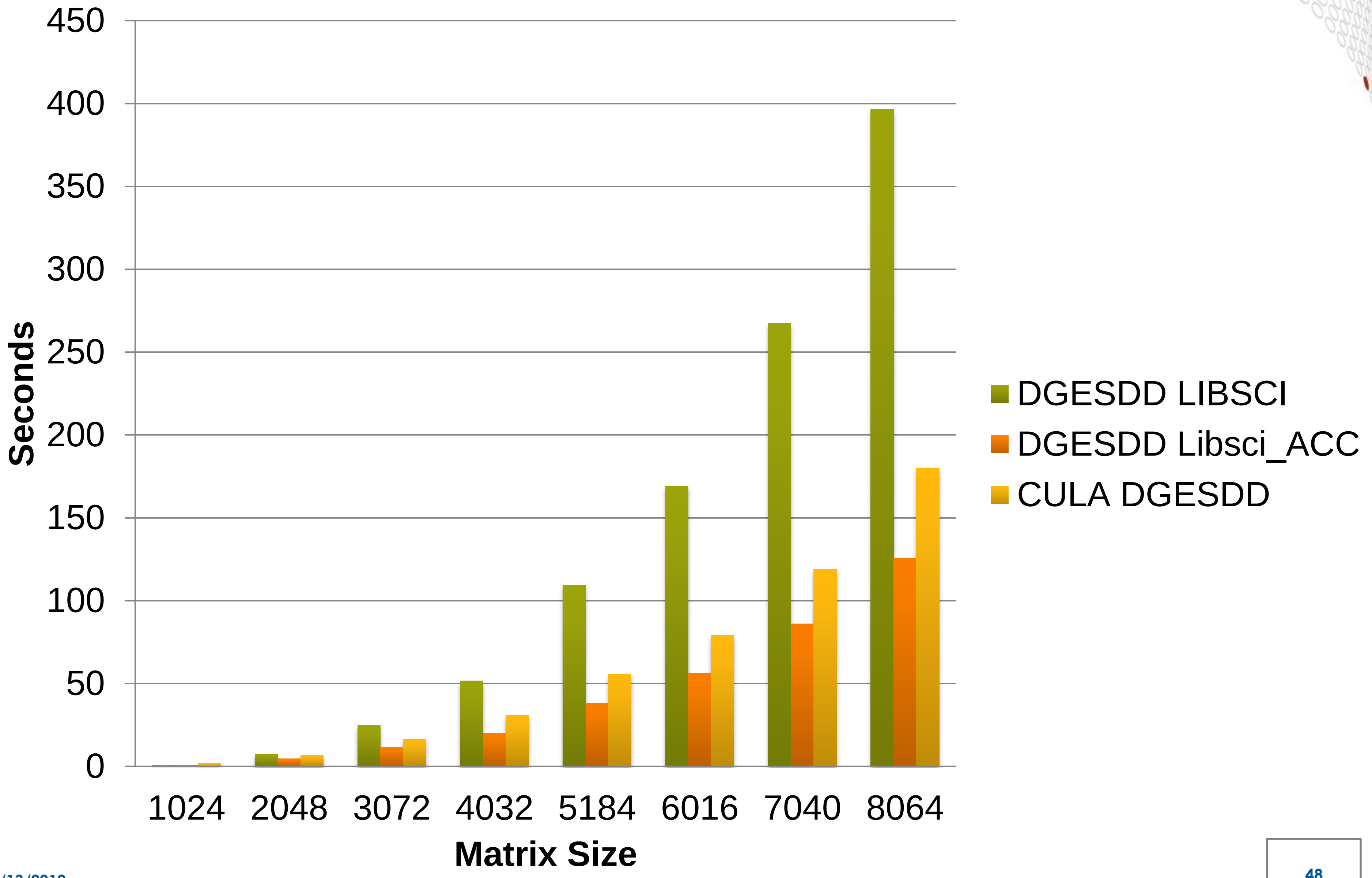



Libsci_acc : Double Complex LU (ZGETRF)
Cray XK6 (and XE6)
July 2012
GFLOPS
0
50
100
150
200
250
300
350
400
0
2000
4000
6000
8000
10000
12000
Matrix dimensions M & N
Libsci 16 cores
Libsci 32 cores
Libsci_acc
MAGMA
CULA/R13

Libsci_acc : Double Complex Cholesky (DPOTRF)
Cray XK6 (and XE6)
July 2012



libsci_acc : Double Complex LU (ZGETRF)
Cray XK6 July 2012



Thanks

- **For specific CrayBLAS optimizations**
  - crayblas@cray.com
- **For more information on Cray libraries**
  - Cray Centre of Excellence
  - or
  - adrian@cray.com