Cray Scientific Libraries

Adrian Tate
Principal Engineer
Cray Inc.
NERSC User Group 2012

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. Network performance
   - Optimize for network performance
   - Overlap between communication and computation
   - Use the best available low-level mechanism
   - Use adaptive parallel algorithms
2. Node performance
   - Highly tune BLAS etc at the low-level
3. Highly adaptive software
   - Using auto-tuning and adpatation, 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

Scientific libraries – package view

FFTW

- fftw-2.1.5
- fftw

PETSc

- petsc-
- Petsc-complex
- CASK (petsc)

Trilinos

- Trilinos 10.8.3.0
- CASK (trilinos)

LibSci

- BLAS
- LAPACK
- ScaLAPACK
- IRT
- CRAFFT

General usage information

- There are many libsci libraries on the systems
- One for each of
  - Compiler (intel, cray, gnu, pgi)
  - Single thread, multiple thread
  - Target (istanbul, mc12, interlagos)
  - Static and shared
- Naming schemes
  - Before libsci 11.0 : libsci_*target.a*
  - After libsci 11.0 : libsci_*compiler.a*
- **Best way to use libsci is to ignore all of this**
- Load the xtpe-module
  - module load xtpe-mc12
- “ftn” and “cc” are the magic tools that will help
  - Link appropriate libraries for your environment
  - Add all the library paths and that you need

Your friends

- module command (module --help)
- PrgEnv modules :

```
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
```

- Component modules

```
------------------------------------------------- /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)
```

- Cray driver scripts **ftn, cc, CC**

To add other libraries

- Perhaps you want to link another library such as ACML
- This can be done. If the library is provided by Cray, then load the module. The link will be performed with the libraries in the correct order.
- If the library is not provided by Cray and has no module, add it to the link line.
- Items you add to the explicit link will be in the correct place
- To get explicit BLAS from ACML but scalapack from libsci
  - Load acml module. Explicit calls to BLAS in code resolve from ACML
  - BLAS calls from the scalapack code will be resolved from libsci (no way around this with static libraries)

Check you got the right library!

- I recommend adding 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!)

Threaded BLAS library

- Libsci includes standard BLAS1, 2, 3
- Most BLAS in libsci are highly tuned and threaded
- The emphasis is on the routines that are most important to users – feedback always welcome
- There are single and multi-threaded libraries on the system
- The multi-thread library is linked by default
- The single-thread library is there for specialist use and debugging – no real reason to try it
- Usage – just as standard BLAS

Threading

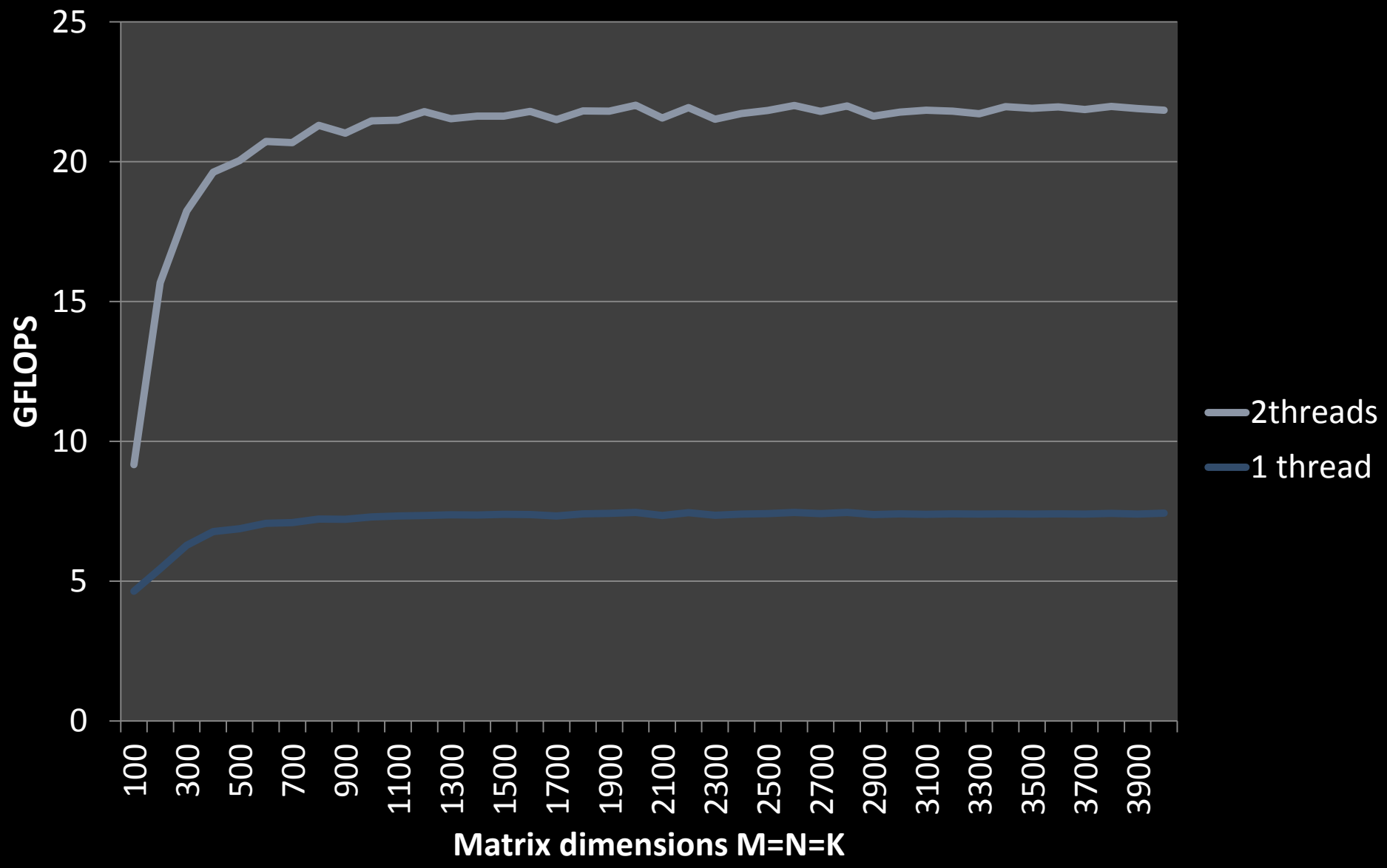
- LibSci is (now) compatible with OpenMP
- Control the number of threads to be used in your program using **OMP_NUM_THREADS**

  e.g. in job script

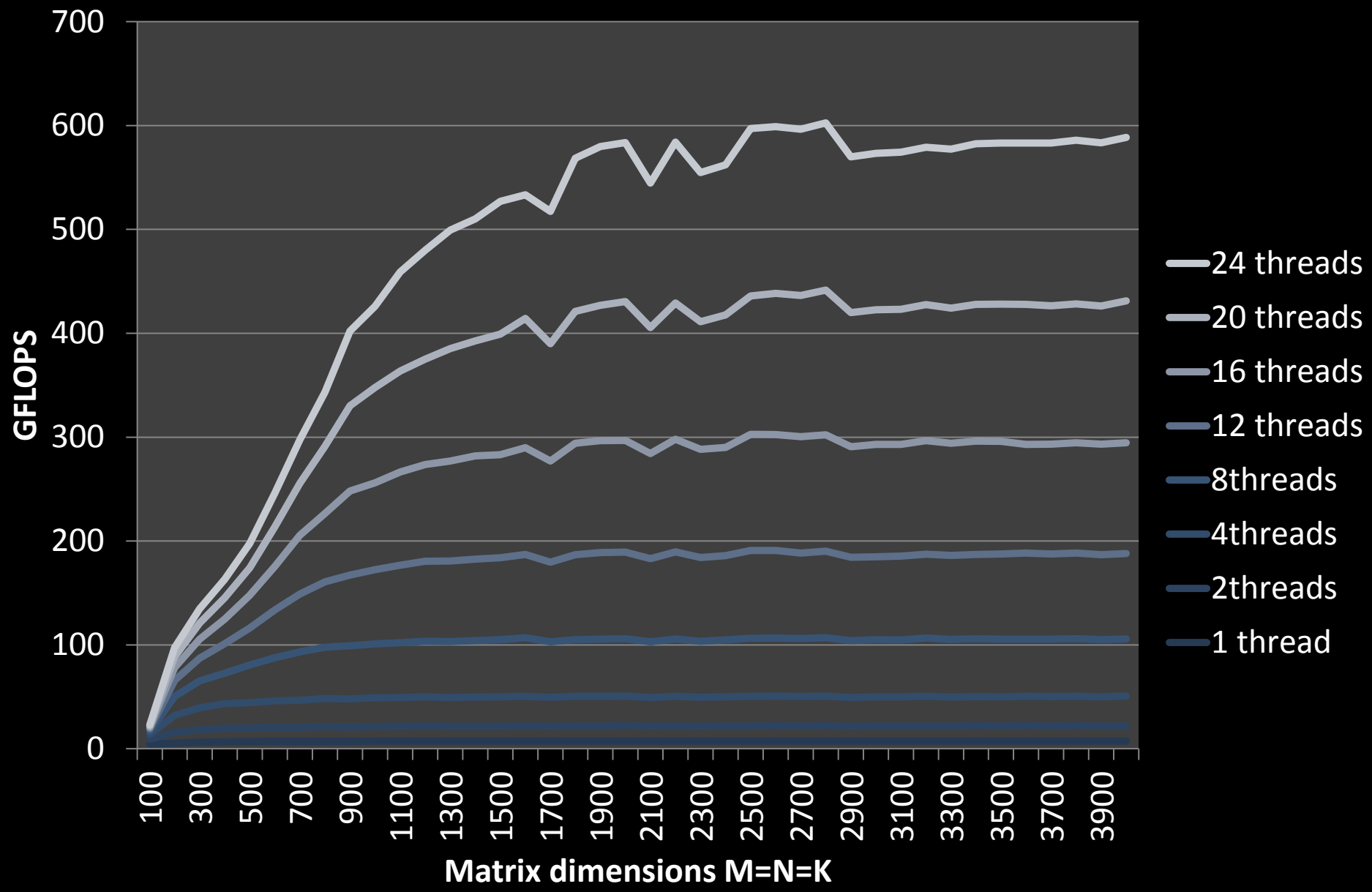
  setenv OMP_NUM_THREADS 16

  - 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

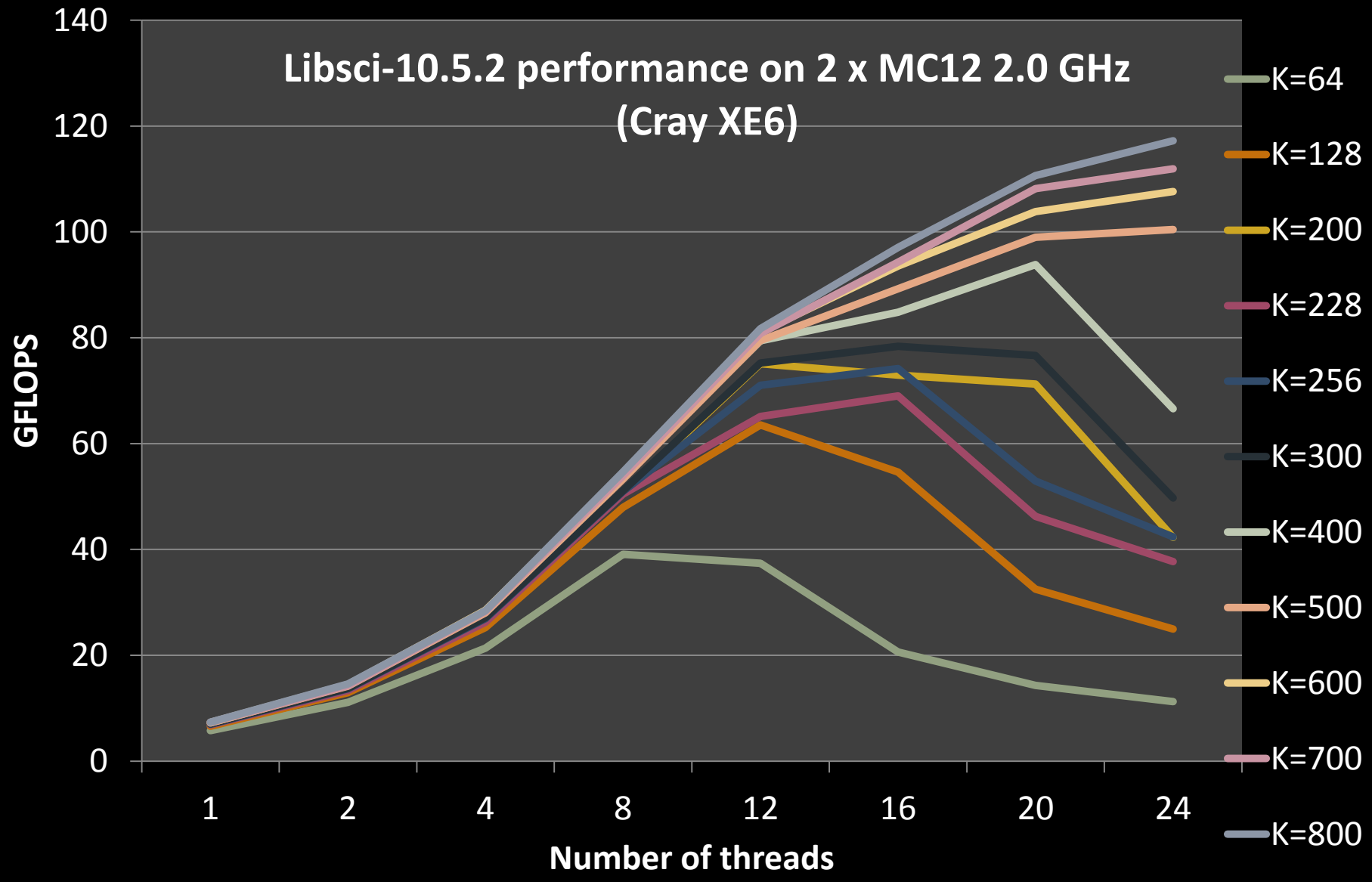
DGEMM Performance 2 x MC12 (1 and 2 threads)



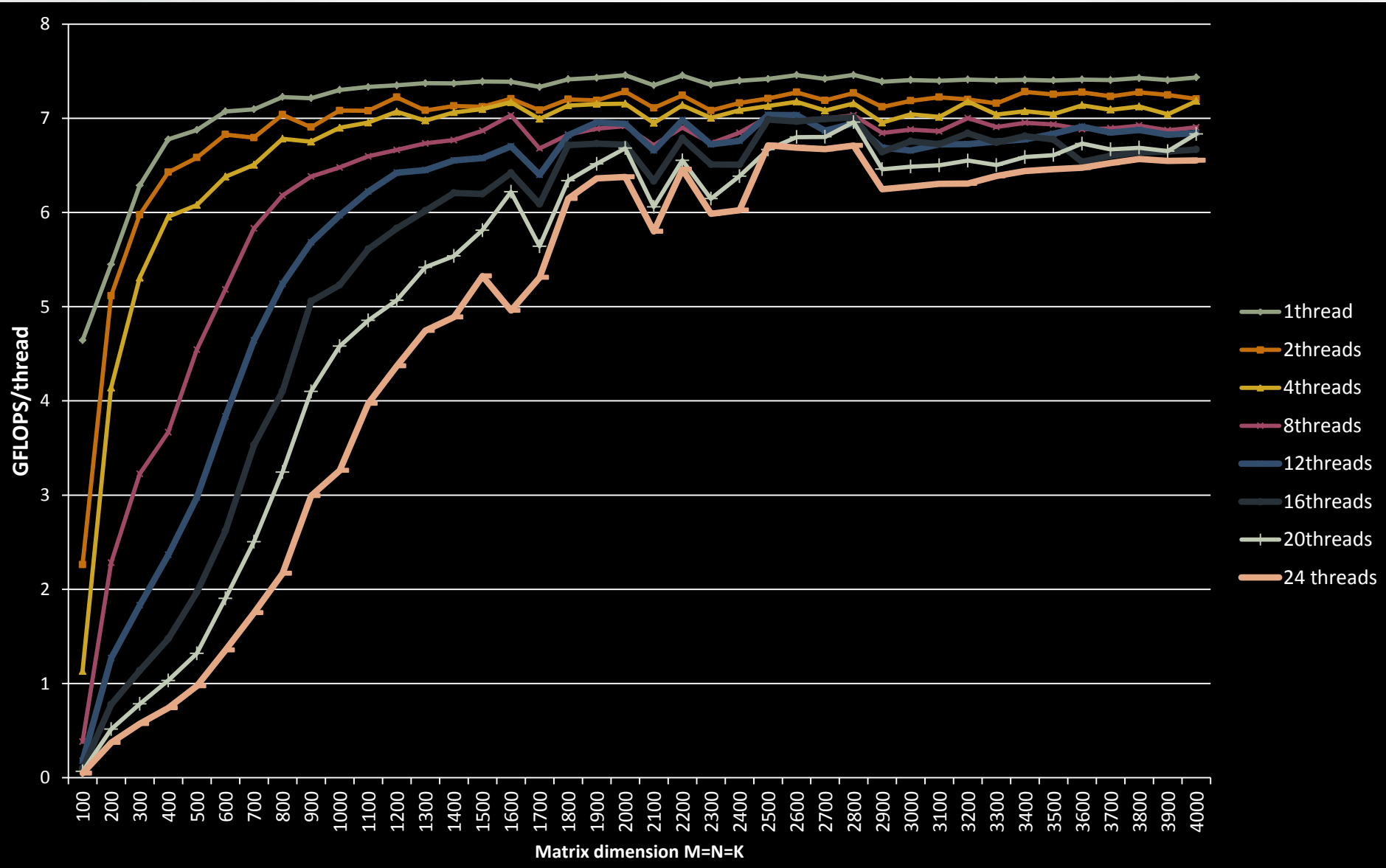
DGEMM Performance 2 x AMD MC12



Threaded DGEMM performance



Efficiency per thread



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

CrayBLAS 1.0 is coming

- We are preparing the release of an entirely new BLAS library
- This has been built in a completely different way
  - using our autotunign framework
  - By building an entirely adaptive interface into BLAS calls
  - Using a new generalized formulation of BLAS
- The generalized BLAS code allows much greater performance variation
  - Explore all loop orderings
  - Explore all threading options
  - Explore all buffer combinations
  - Change all block sizes and number of block levels.
- The idea is that you will receive the best of many many BLAS kernel versions for your specific problem at runtime

CrayBLAS bonuses

- What this will give you
- Extremely good performance for
  - Unusual problem sizes/shapes
  - Better performance within solvers (who also have unusual)
- Much richer openMP support
  - Multi-levels of parallelism
  - User selects the inner-most thread number, or
  - openMP run-time can decide how much to use
  - BIT –reproducible threaded GEMM

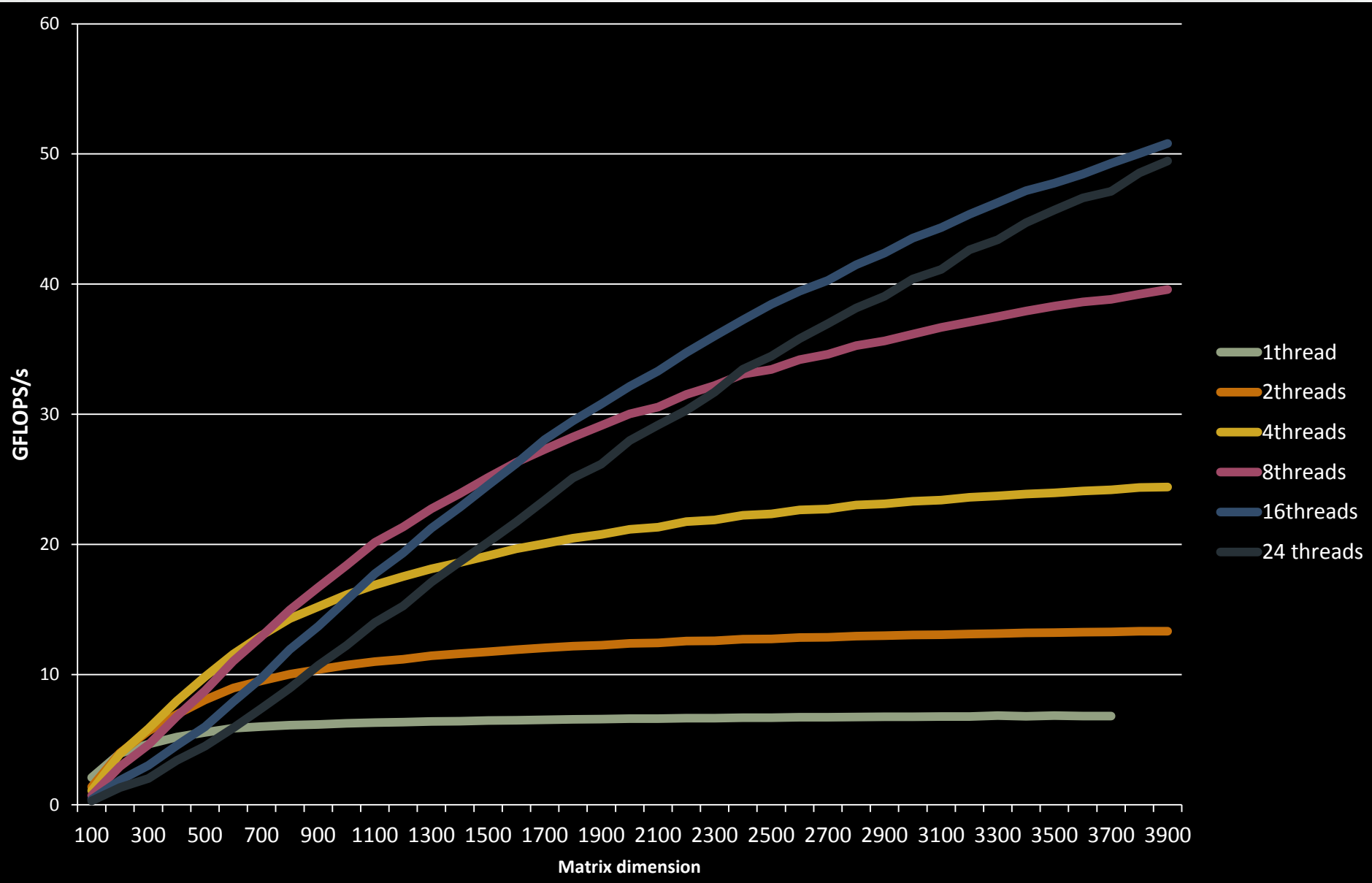
CrayBLAS early MC12 numbers

M	N	K	LibSci-10.5.2	CrayBLAS	%improve
8	8	8	0.04	0.18	**352.34%**
80	80	80	3.81	5.50	**44.26%**
80	800	80	4.87	6.67	**36.99%**
8	8000	8	2.25	2.35	**4.54%**
800	800	80	5.88	6.61	**12.28%**
200	200	200	5.69	6.69	**17.73%**
200	2000	200	6.16	7.28	**18.07%**
1000	1000	256	6.89	7.27	**5.54%**
1000	200	200	6.60	6.83	**3.54%**

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 LU (DGETRF) thread performance



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 :

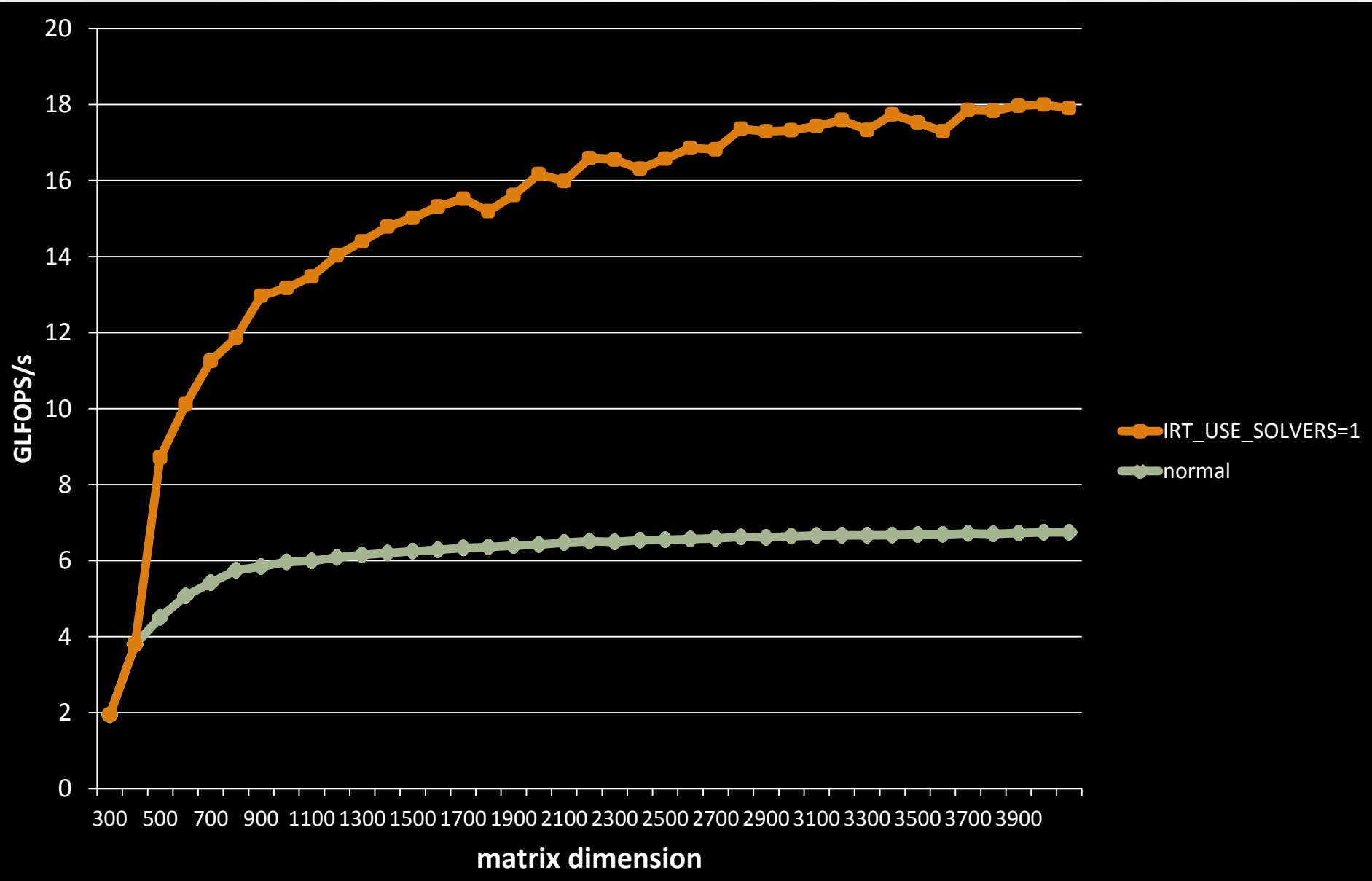
setenv IRT_USE_SOLVERS 1

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 example – LAPACK LU (DGESV)



FFTW

- Cray's main FFT library is FFTW from MIT
- We work with the FFT developers to make sure that this is optimized for Cray hardware
  - We wrote the bulldozer version of FFTW for MIT
- 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
- CRAFFT can be used for advanced controls of FFTW and better parallel performance

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
```

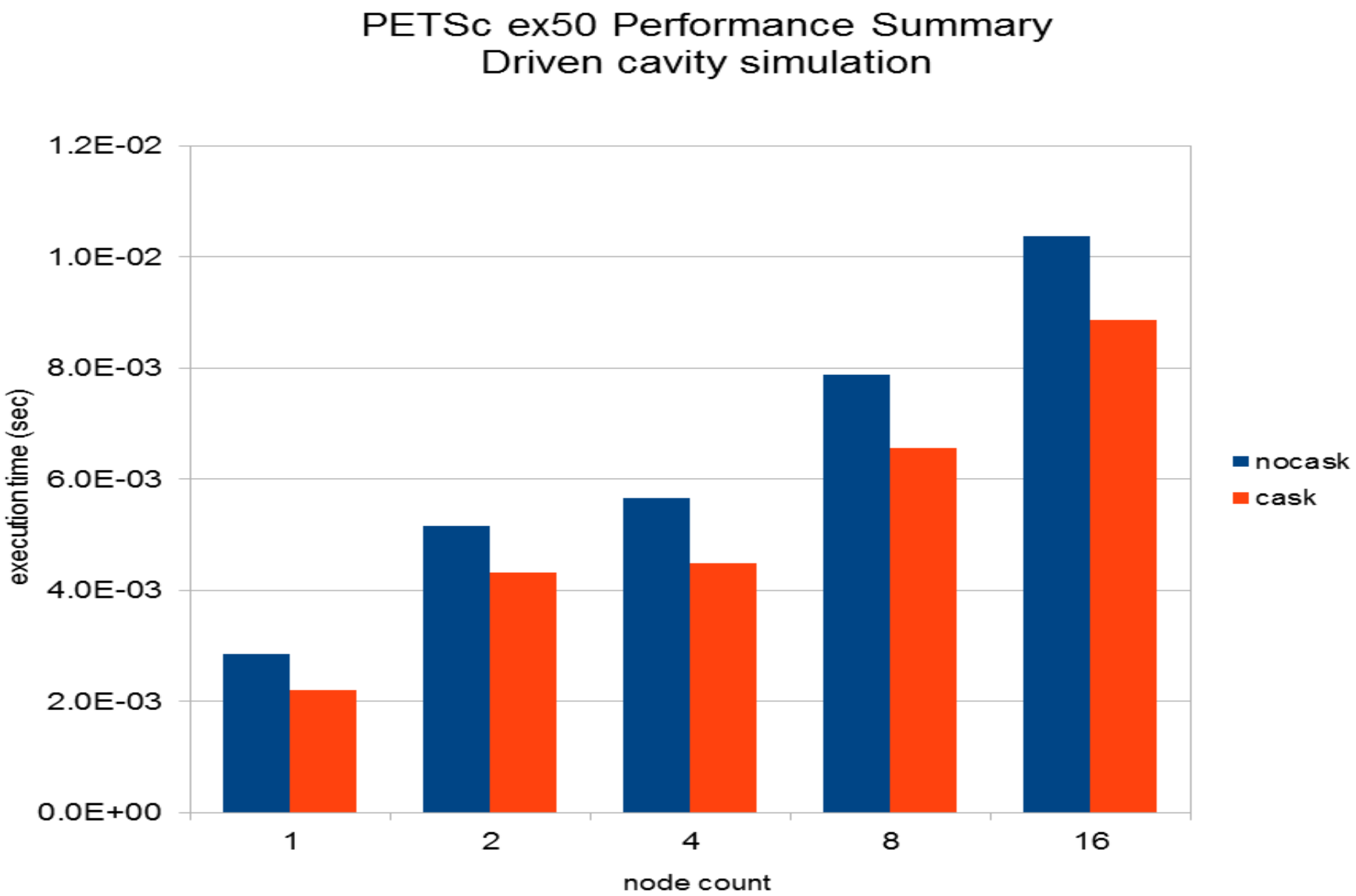
Example of CRAFFT v FFTW3.3.0 (on AMD IL)

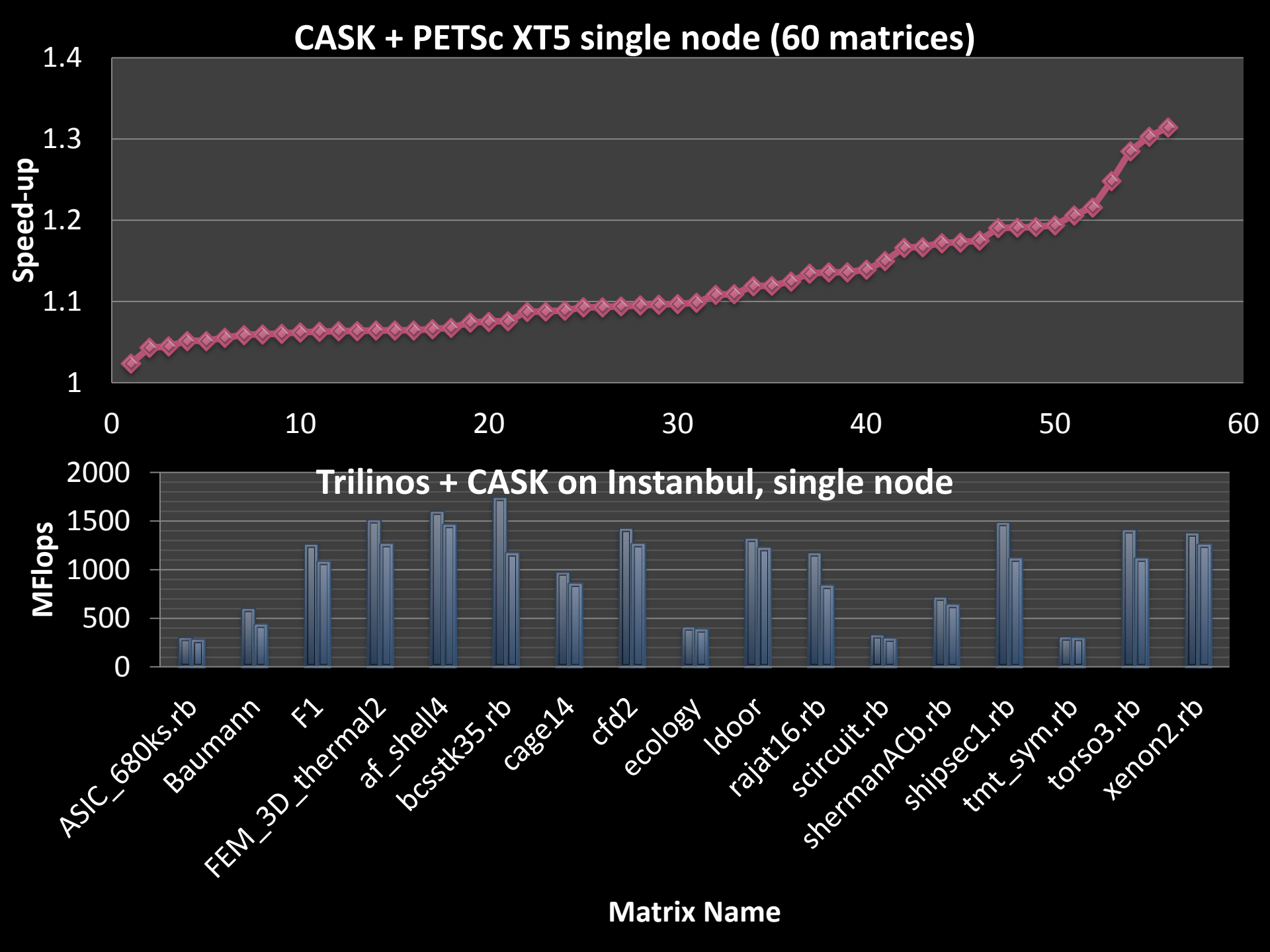
	CRAFFT, plan=0 (=FFTW_ESTIMATE)	CRAFFT plan = 1 (=FFTW_MEASURE)	2d mpi fft, normal, n=12288	FFTW out-place
r2c, Gflops	5.36	5.92	4.8	4.7
c2r, Gflops	4.14	5.02	4.8	5.14

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 (Cray ATF)
- Uses ATF auto-tuning, specialization and Adaptation concepts
- 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




CASK + PETSc XT5 single node (60 matrices)
Speed-up
1.4
1.3
1.2
1.1
1
0
10
20
30
40
50
60
Trilinos + CASK on Instanbul, single node
MFlops
2000
1500
1000
500
0
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_acc (libsci for accelorators)

- Tuned library for hybrid nodes of NVIDIA GPUs + AMD IL
- Simple interface
  - Use the standard API for BLAS, LAPACK etc
  - Libsci_acc does it all under the covers
    - Manages and pins the host memory
    - Allocates GPU resources
    - Copies data to the GPU
    - Performs the operation on GPU and on CPU
    - Copies data back to the GPU
- Provides the following
  - [s,d,z,c]GEMM
  - [s,d,z,c]GETRF
  - [s,d,z,c]POTRF

Upcoming releases

- March release
  - LAPACK 3.4.0
  - C interfaces for lapack
  - CRAFFT CAF optimizations
- April release
  - CrayBLAS1.0
- April release of libsci_acc
  - Fully adaptive BLAS (GEMM)
  - POTRF, DGESDD

CRAY

THE SUPERCOMPUTER COMPANY