CRAY
THE SUPERCOMPUTER COMPANY

Cray Scientific and Math Libraries

Luiz DeRose
Programming Environments Director
Cray Inc.

HP2C

CSCS
Swiss National Supercomputing Centre

Cray Scientific Libraries – Value Add

- Optimized libraries for Cray Hardware
  - AMD-Interlagos
  - Magny-Cours, …
- Compiler Support and Verification
  - CCE
  - GNU
  - Intel
  - PGI
- Auto-tuning / Adaptive Kernels
  - Alternative optimized kernels
  - Tuned kernels selected based on problem size
  - Iterative Refinement Toolkit (IRT)
- Simplified Interface
  - CRAFFT
  - PETSc/Trilinos simplified application build
  - Cray Adaptive Eigensolver

General Usage Information

- There are many libsci libraries on the systems
- One for each of
  - Compiler (cray, gnu, pgi, intel)
  - Single thread, multiple thread
  - Target (interlagos, mc12, mc8, …)
- Best way to use libsci is to ignore all of this
- Load the xtpe-module
  - module load xtpe-interlagos / xtpe-mc12 / xtpe-mc8
- Cray's compiler drivers will link the library automatically
- PETSc, Trilinos, fftw, acml all have their own module

Making Sure You Have the Right Library

- We 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_
    - Will return :

      cc -L./ -o mmulator blas_test.o netlib_dgemm.o -Wl,-ydgemm_
      blas_test.o: reference to dgemm_
      /opt/xt-libsci/10.4.9/cray/lib/libsci.a(dgemm.o): definition of dgemm_

OpenMP BLAS

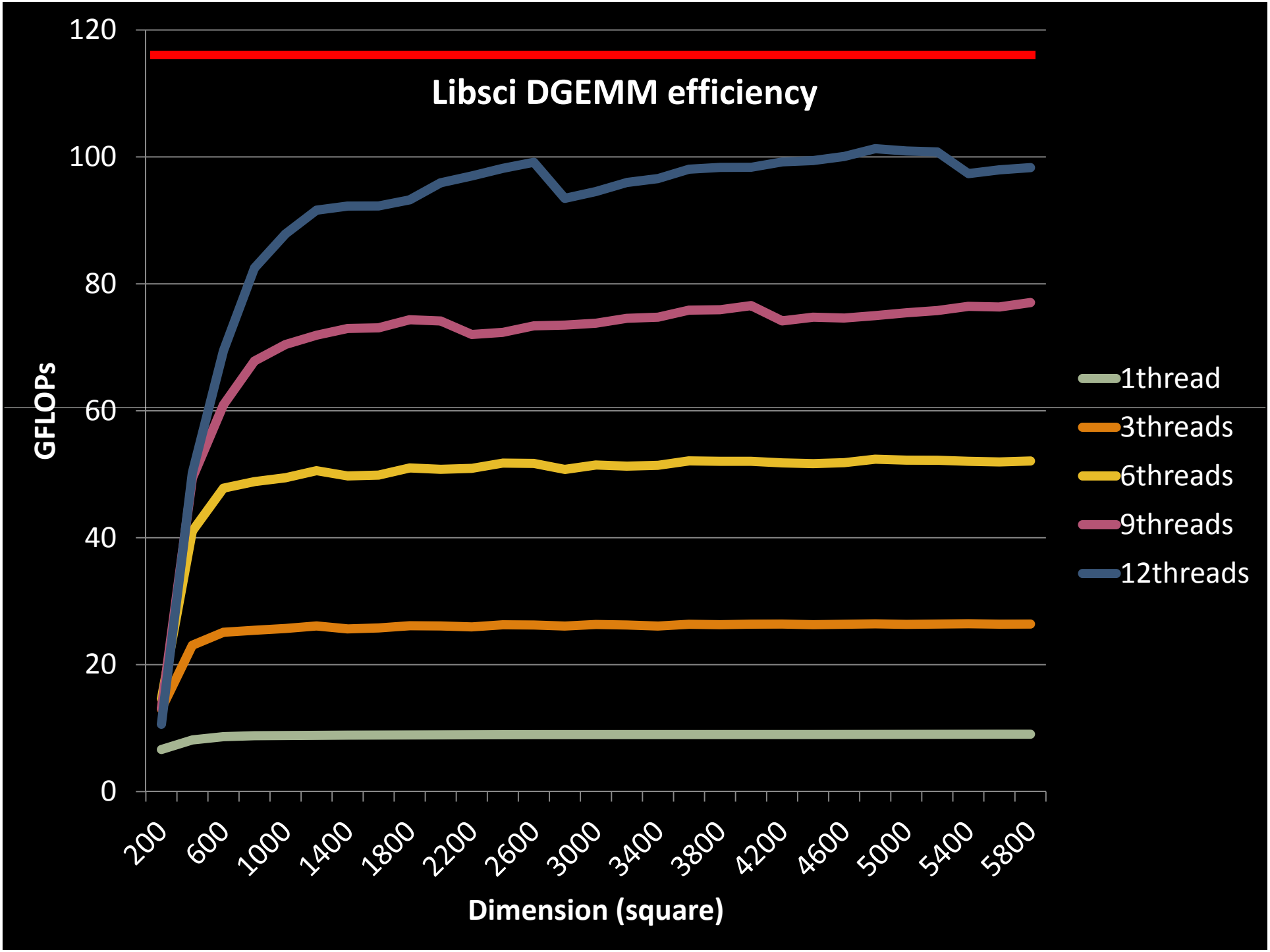
- Threading capabilities in previous libsci versions were poor
  - Used PTHREADS (more explicit affinity etc)
  - Required explicit linking to a _mp version of libsci
  - Was a source of concern for some applications that need hybrid performance and interoperability with openMP
- LibSci 10.4.2 (February 2010)
  - OpenMP-aware LibSci
  - Allows calling of BLAS inside or outside parallel region
  - Single library supported (there is still a single thread lib)
- Usage – load the xtpe module for your system (interlagos)
- **Use OMP_NUM_THREADS**
  - **GOTO_NUM_THREADS outmoded**

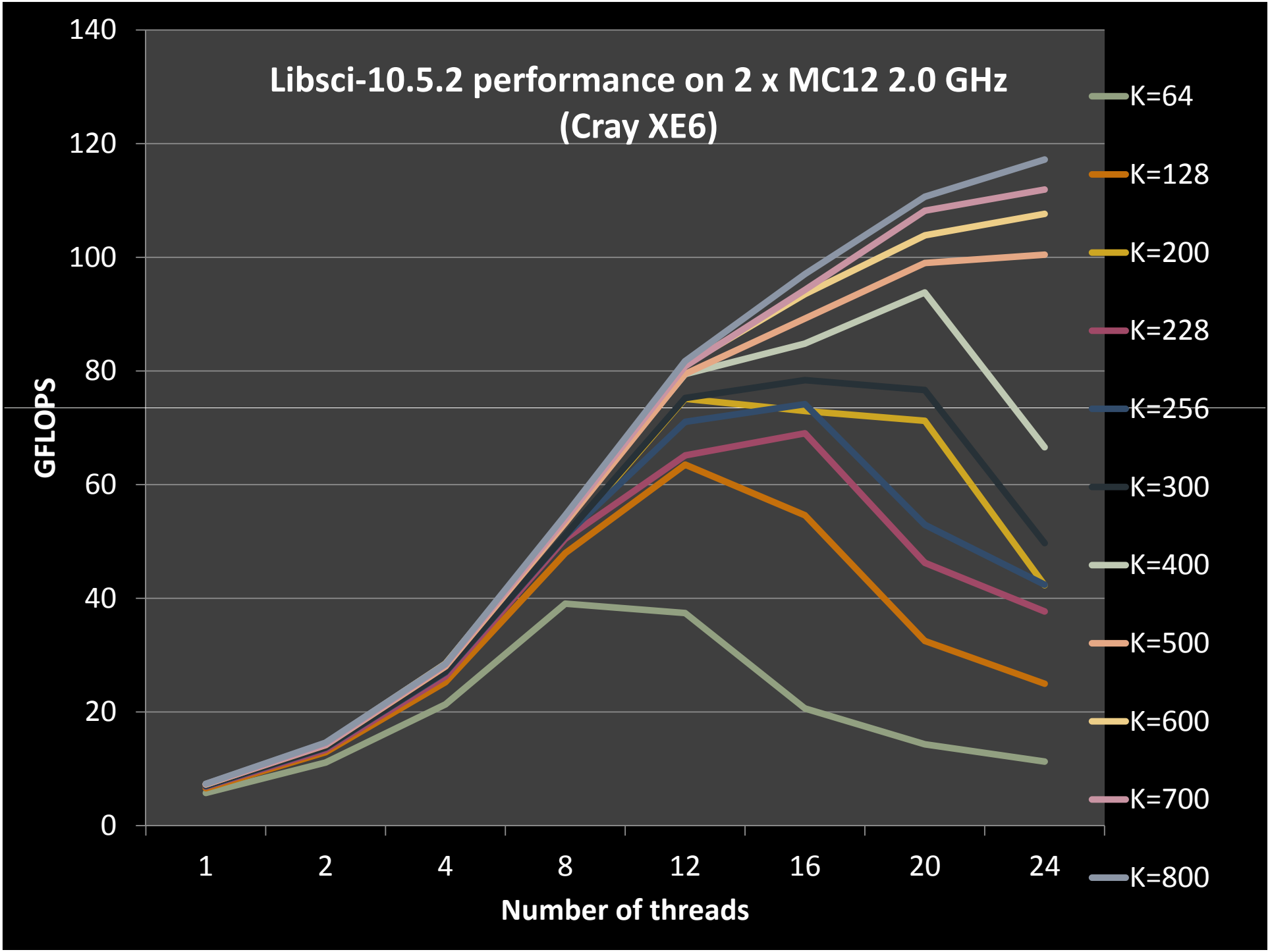
OpenMP LibSci

- Allows seamless calling of the BLAS within or without a parallel region

e.g. OMP_NUM_THREADS = 16

```
call dgemm(…)   threaded dgemm is used with 16 threads
!$OMP PARALLEL DO
do
        call dgemm(…)   single thread dgemm is used
end do
```


Libsci DGEMM efficiency
GFLOPs
120
100
80
60
40
20
0
200
600
1000
1400
1800
2200
2600
3000
3400
3800
4200
4600
5000
5400
5800
Dimension (square)
1thread
3threads
6threads
9threads
12threads


Libsci-10.5.2 performance on 2 x MC12 2.0 GHz (Cray XE6)
GFLOPS
0
20
40
60
80
100
120
140
1
2
4
8
12
16
20
24
Number of threads
K=64
K=128
K=200
K=228
K=256
K=300
K=400
K=500
K=600
K=700
K=800

Using ScaLAPACK in hybrid mode on XE6

- Use the number of ScaLAPACK grid points you want to correspond to the number of MPI ranks you want
- Rely on the BLAS to operate with the number of threads you desire
- Use OMP_NUM_THREADS and the aprun options to set the number of threads you need for on-node parallelism
- Set the threads per node from libsci BLAS with OMP_NUM_THREADS
- Use aprun options –n and –d for nodes and threads

ScaLAPACK broadcast optimizations

- On Gemini systems, the choice of underlying broadcast algorithm used in ScaLAPACK is very important
- Unfortunately this is not exposed to the user since it is hard coded into BLACS
- One type of BCAST, the I-Ring can effectively become a node-aware broadcast
  - Can perform extremely well on Gemini
- Added the environment variables
  - SCALAPACK_LU_RBCAST
  - SCALAPACK_LLT_UBCAST
  - SCALAPACK_LLT_LBCAST

Iterative Refinement Toolkit

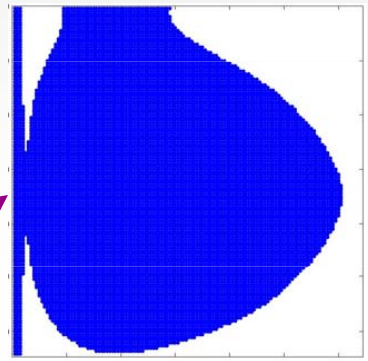
- Mixed precision can yield a big win on x86 machines.
- SSE (and AVX) units issue double the number of single precision operations per cycle.
- On CPU, single precision is always 2x as fast as double
- Accelerators sometimes have a bigger ratio
  - Cell – 10x
  - Older NVIDIA cards – 7x
  - New NVIDIA 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

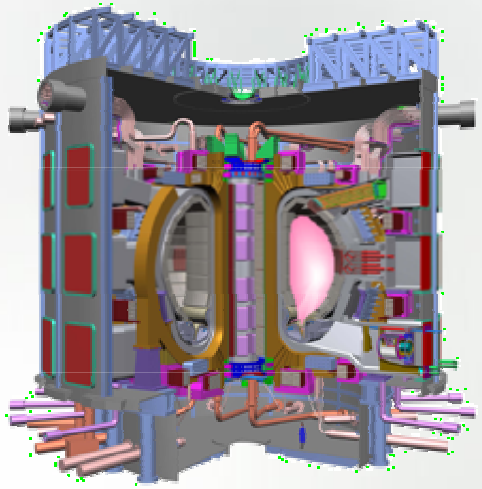
Iterative Refinement Toolkit - Library

- 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

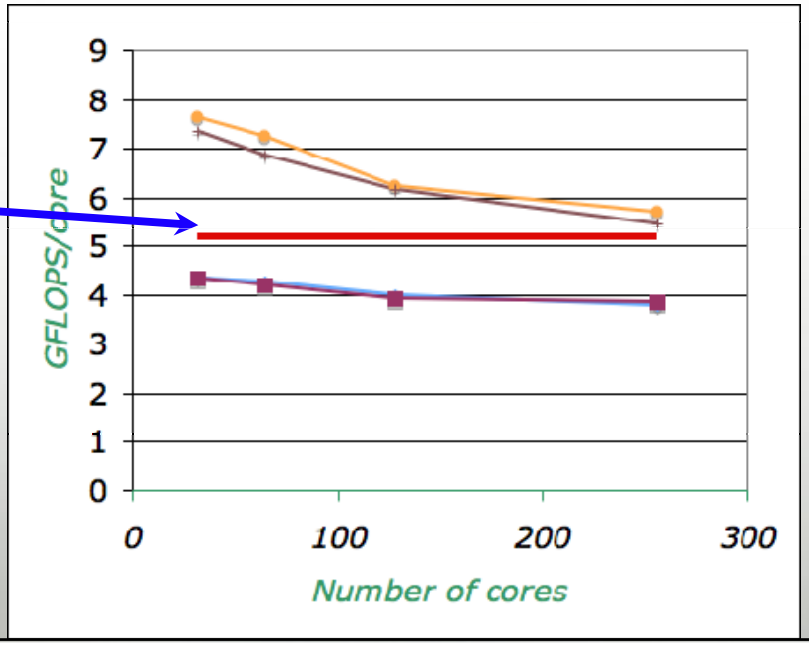
Example: AORSA Fusion Energy

- "High Power Electromagnetic Wave Heating in the ITER Burning Plasma"
- rf heating in tokamak
- Maxwell-Bolzmann Eqns
- FFT
- Dense linear system
- Calc Quasi-linear op





Theoretical Peak



Courtesy
Richard Barrett

OAK RIDGE National Laboratory

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

PETSc

- The Cray PETSc 3.1.09 is equivalent to the official patch release of PETSc-3.1-p8 by Argonne National Laboratory
- Serial and Parallel versions of sparse iterative linear solvers
  - Suites of iterative solvers
    - CG, GMRES, BiCG, QMR, etc.
  - Suites of preconditioning methods
    - IC, ILU, diagonal block (ILU/IC), Additive Schwartz, Jacobi, SOR
  - Support block sparse matrix data format for better performance
  - Interface to external packages (Hypre, SuperLU_DIST,MUMPS)
  - Fortran and C support
  - Newton-type nonlinear solvers
- Large user community
- http://www-unix.mcs.anl.gov/petsc/petsc-as

PETSc External Packages

- Cray provides external scientific computing packages to strengthen the capability of PETSc
  - Hypre: scalable parallel preconditioners
    - AMG (Very scalable and efficient for specific class of problems)
    - 2 different ILUs (General purpose)
    - Sparse Approximate Inverse (General purpose)
  - ParMetis: parallel graph partitioning package
  - MUMPS: parallel multifrontal sparse direct solver
  - SuperLU: sequential left-looking sparse solver
  - SuperLU_DIST: parallel right-looking sparse direct solver with static pivoting

Trilinos

- The Cray Trilinos 10.6.4.0 is equivalent to the official patch release of Trilinos 10.6.4 by Sandia National Labratories
- The Trilinos module is dependent on the petsc, xt-libsci, and xt-asyncpe modules
  - Make certain these modules are loaded before using Trilinos
- To use the Trilinos packages, load your choice of PrgEnv and then load the Trilinos module

  % Module load trilinos
- Several packages are incorporated
  - man trilinos (3) for details

Adaptive Scientific Libraries

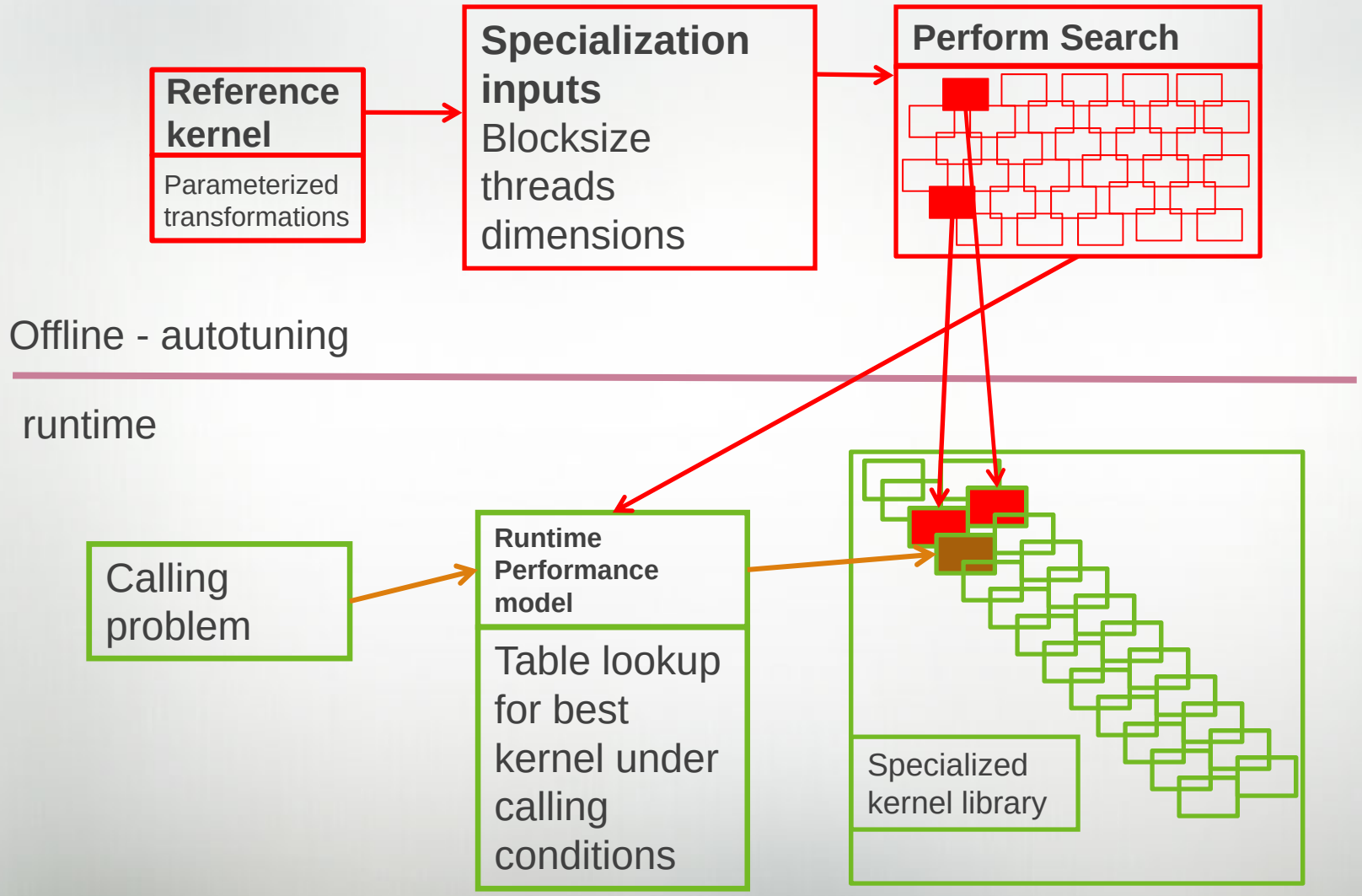
- Scientific Libraries today have three concentrations to increase productivity with enhanced performance
  - **Standardization**
  - **Autotuning**
  - **Adaptive Libraries**

- Cray **adaptive** model
  - Runtime analysis allows **best** library/kernel to be **used dynamically**
  - Extensive offline testing allows **library to make decisions** or remove the need for those decisions
  - Decision depends on the system, on previous performance info, obtained previously, and characteristics of calling problem

The Cray Auto-Tuning Framework

- Automation of code optimization
  - Includes automation of the following ‘components’
    - Code generation
    - Compilation
    - Batch submission
    - Parameter Search
    - Result Analysis
- Allows many more optimizations to be studied
- ‘Search’ component means allows massive optimization space to be studied in realistic time
- Currently employed in two projects at Cray
  - CASK: Cray Adaptive Sparse Kernels
    - Optimize PETSc and Trilinos on Cray without the user even knowing
  - CRAFFT: Cray Adaptive FFT
    - Provides one very simple interface into all existing FFT libraries
    - Uses previous performance information to decide where to go

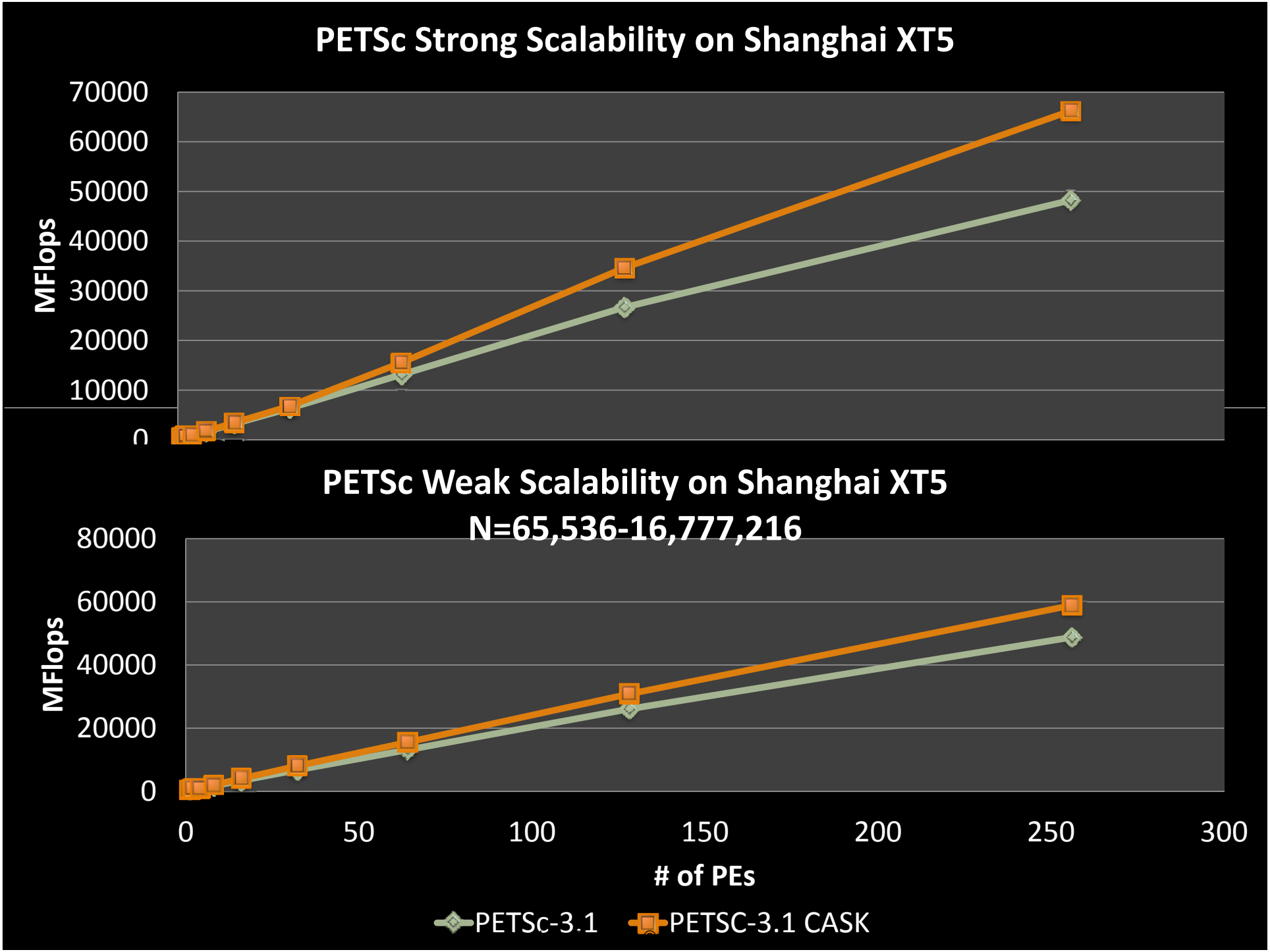
Adaptation, Auto-tuning and Specialization

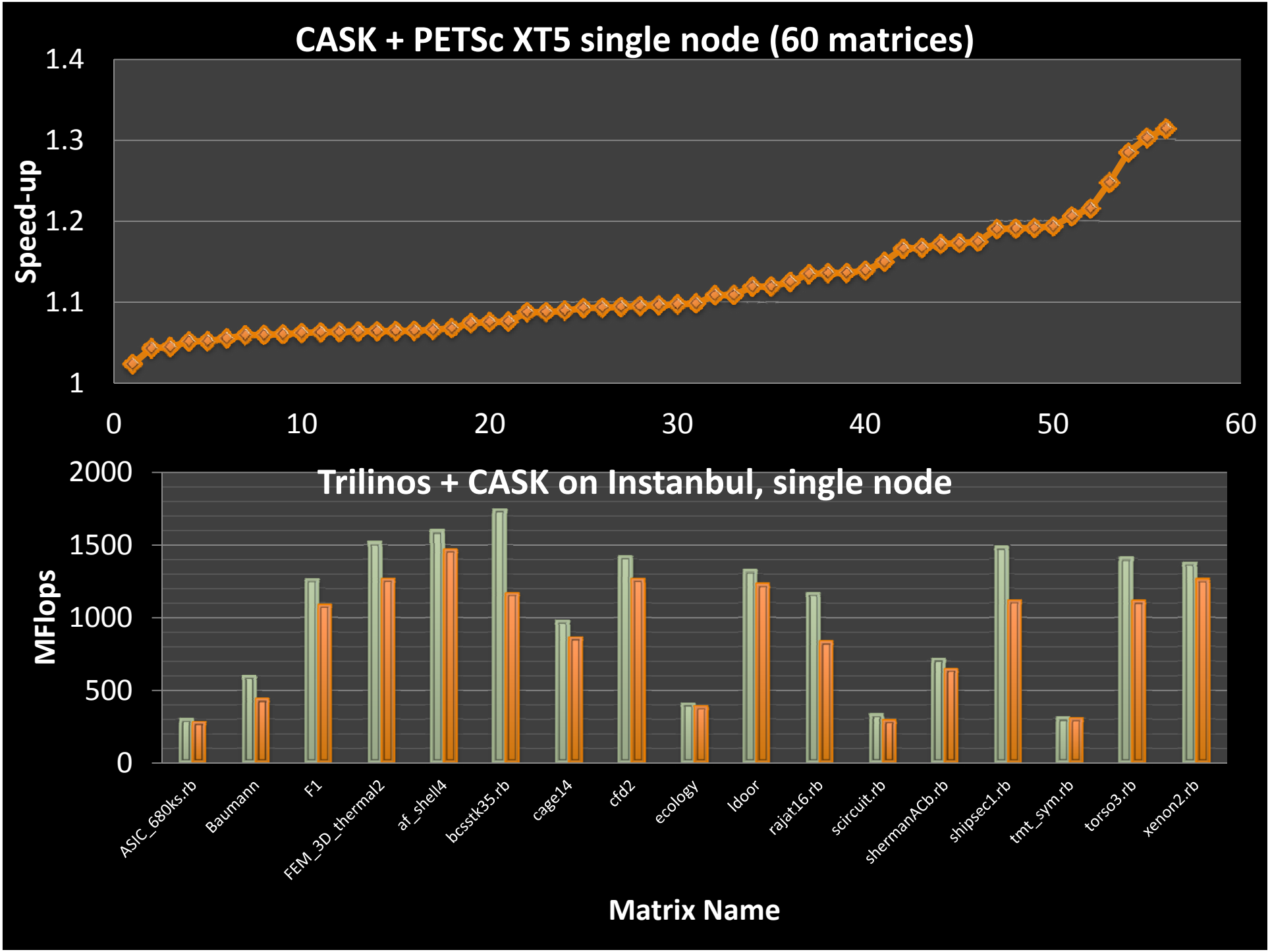


This is all invisible to the user :: all you will see is good performance

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


PETSc Strong Scalability on Shanghai XT5
MFlops
70000
60000
50000
40000
30000
20000
10000
0
PETSc Weak Scalability on Shanghai XT5
N=65,536-16,777,216
MFlops
80000
60000
40000
20000
0
0
50
100
150
200
250
300
# of PEs
PETSc-3.1
PETSC-3.1 CASK


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

Usage

As easy as you can get :

```
module load petsc
```

or

```
module load trilinos
```

That’s all you need

Cray Adaptive FFT (CRAFFT)

- Serial CRAFFT is largely a productivity enhancer
- 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, 2-d, 1d
  - In-place and out-of-place
  - FFTW interfaces
  - 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
```

CrayBLAS

- At the end of 2011, Cray will transition to a completely new BLAS library called CrayBLAS
- This library will be 100% autotuned
- General concept – create a completely general DGEMM
  1. Arbitrarily blocked
  2. Has an arbitrary number of levels of blocking
  3. Has an arbitrary ordering
  4. Has an arbitrary number of bufferings
  5. Has an arbitrary mapping into buffer space
- Then, in the same style as CASK we create a completely adaptive library interface
  - The user's calling problem is matched to the best implementation from the auto-tuning
- Should lead to better performance for all problem sizes
  - Even unusual dimensions or shapes

CRAY
THE SUPERCOMPUTER COMPANY

Cray XE Workshop

Questions / Comments

Thank You!