Cray Scientific Libraries – Overview and Performance Evaluation

Goals of this talk

1. Introduce libsci and explain usage for those not familiar
2. Show XT5 and XT6 performance results for some key areas
3. Explain what is being done for Gemini
4. Introduce our specialization model
5. Get information from you to help improve

Structure of the Talk

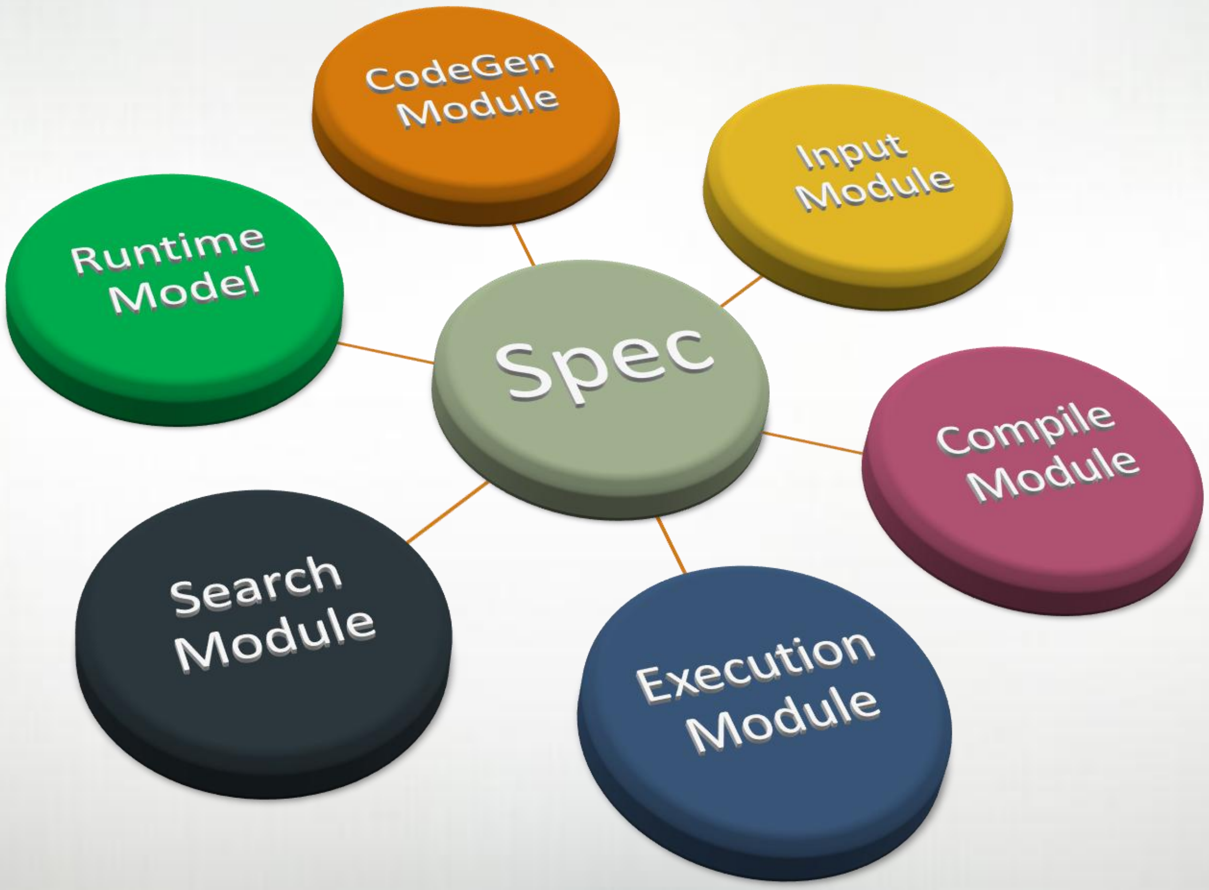
- LibSci philosophy and direction
- LibSci OpenMP BLAS
- FFT
- ScaLAPACK
- CASK
- Specialization

Cray Scientific Libraries Philosophy

- Goal of scientific libraries

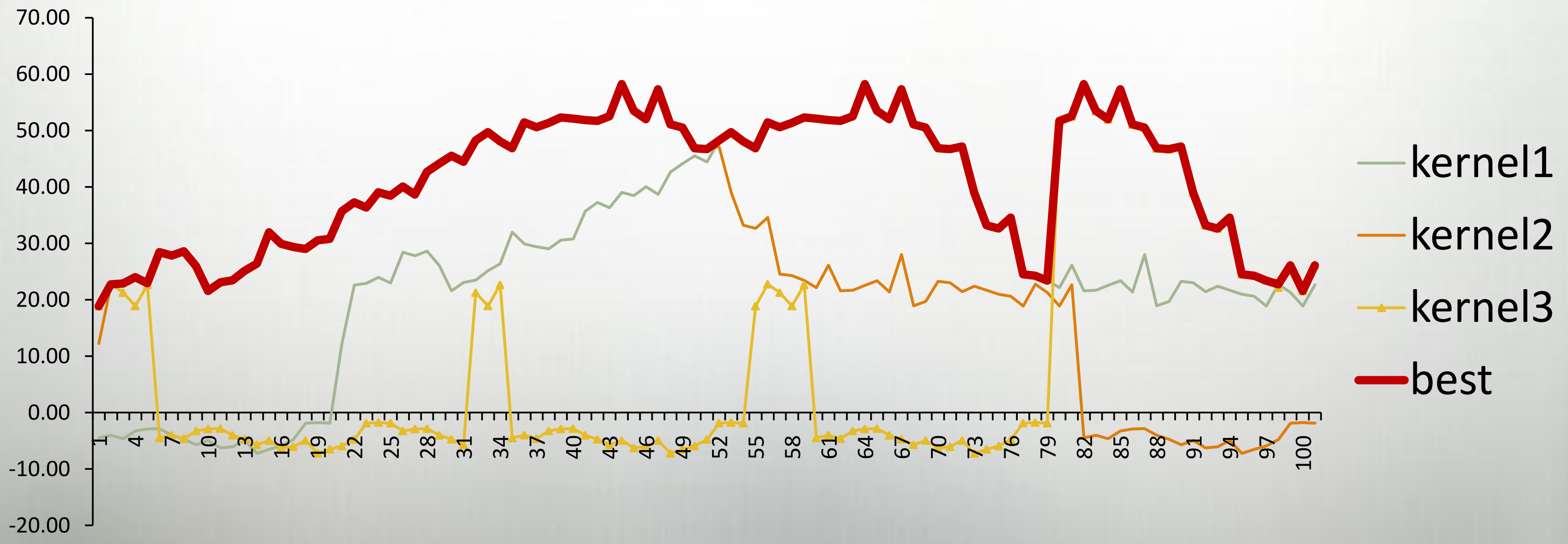
  optimal performance requiring the least effort possible
- Tradition method of providing such performance
  - Optimize very general purpose routines
  - Optimize for most "common problem" sizes
- Problem with this approach
  - Nobody actually uses "common problem sizes" !!
- New approach – Adaptive Specialization
  - Use Cray's existing Auto-tuning Framework to build a general tuning environment
  - Build an adaptive library that can take a huge number of tuned variants
  - Adaptive library serves specialist code for certain code sequences
  - Customers can request the tuning of a routine for a given input or set of inputs

CrayATF is the world's first generalized tuning framework

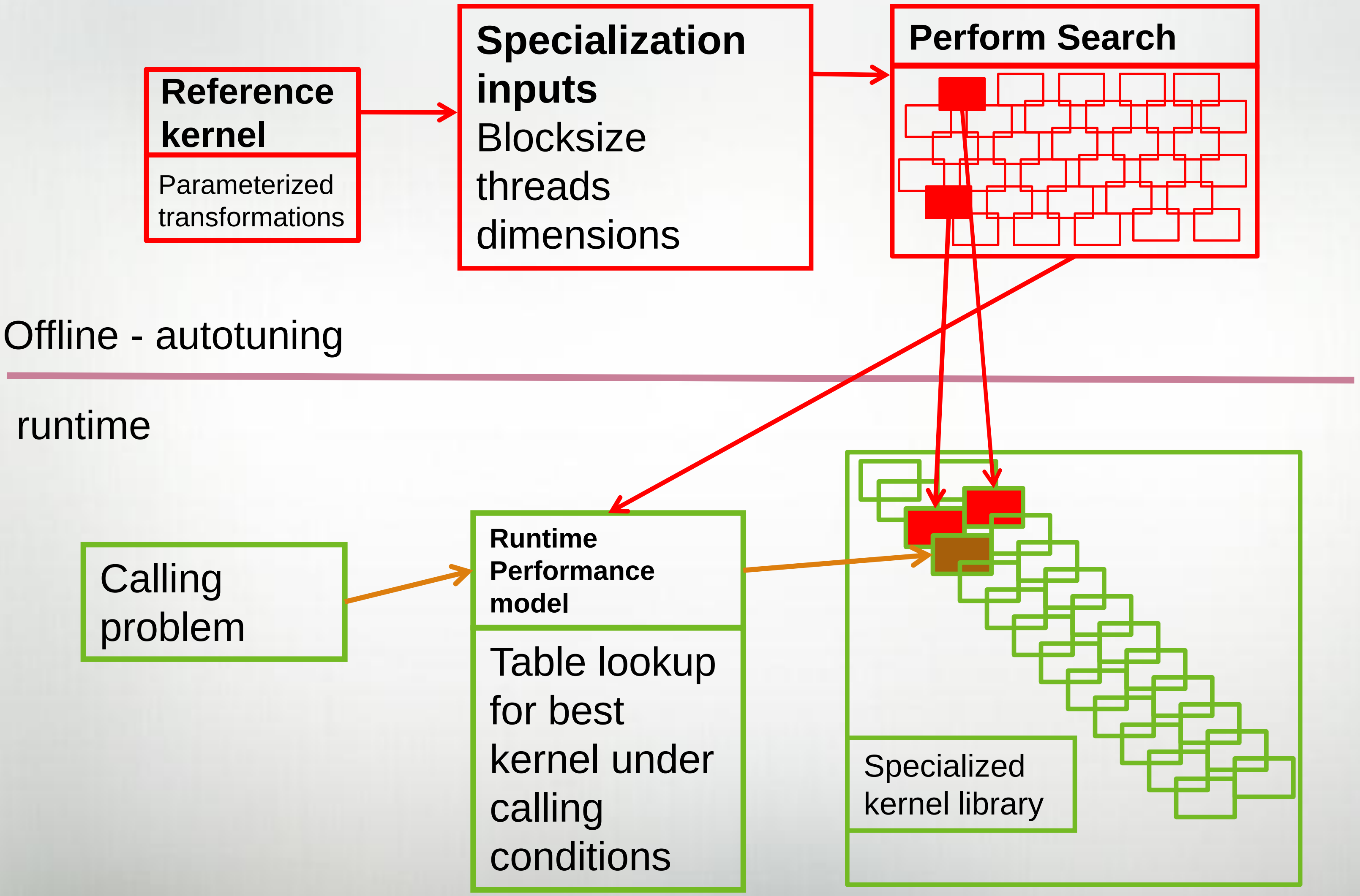


Adaptation

- Adaptation is corresponding concept to auto-tuning
- Given a set of good kernels and a range of input values
    - Map the best or very good kernel onto each combination of input values
    - Need a switching function such that
      for any set of testing parameters we obtain the best kernel

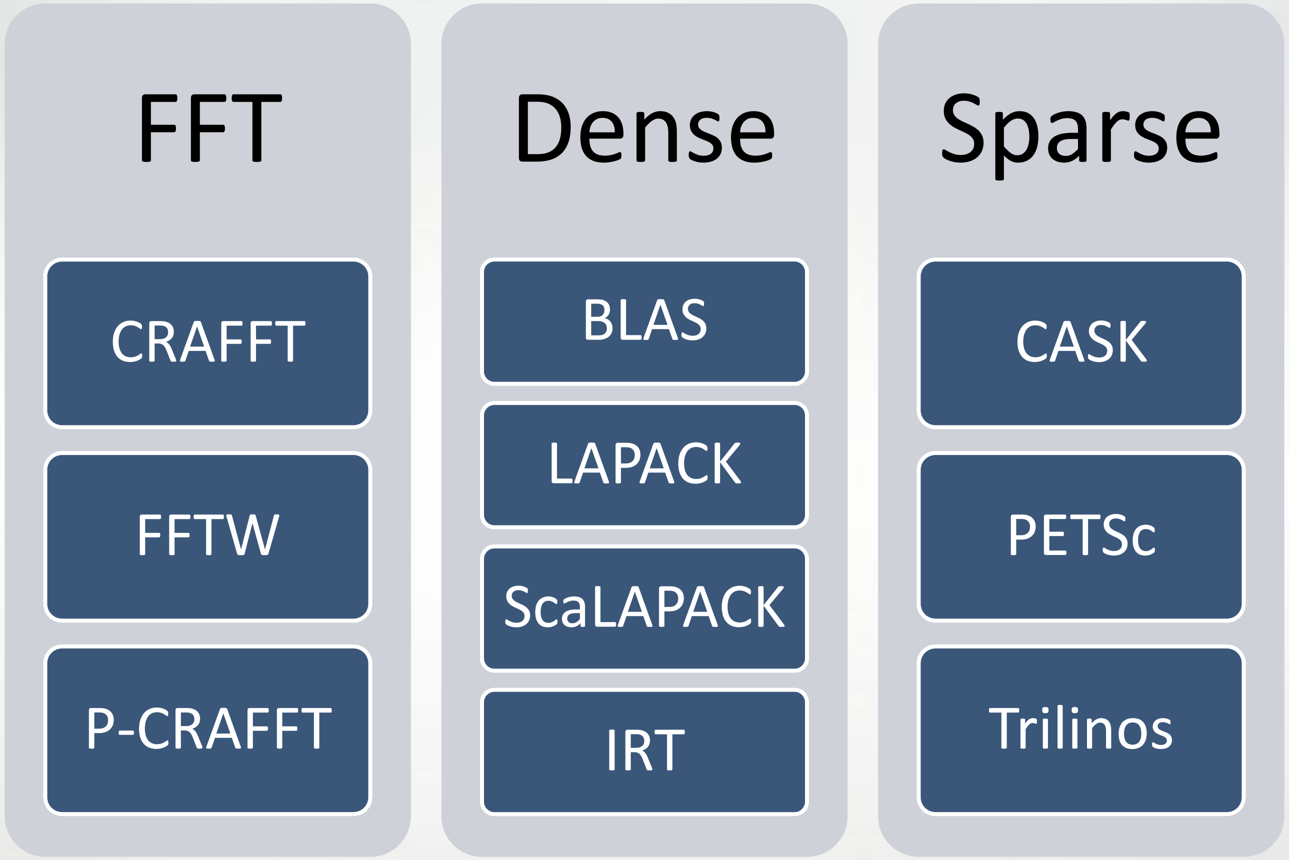


Adaptation, Auto-tuning and Specialization



Reference kernel
Parameterized transformations
Specialization inputs
Blocksize
threads
dimensions
Perform Search
Offline - autotuning
runtime
Calling problem
Runtime Performance model
Table lookup for best kernel under calling conditions
Specialized kernel library

Cray Scientific Libraries



IRT – Iterative Refinement Toolkit
CASK – Cray Adaptive Sparse Kernels
CRAFFT – Cray Adaptive FFT

OpenMP BLAS

- Threading capabilities in previous libsci versions were poor
  - Used PTHREADS
  - Required explicit linking to a _mp version of libsci
  - Was a source of concern for some applications that need hybrid performance
- LibSci 10.4.2 February 18th 2010
  - OpenMP-aware LibSci
  - Allows calling of BLAS inside or outside parallel region
  - Single library supported
    - No multi-thread library and single thread library (-lsci and –lsci_mp)
    - Performance not compromised

    (there were some usage restrictions with this version)
- Usage – link to libsci_istanbul or libsci_mc12 or libsci_mc8

GOTO_NUM_THREADS outmoded – use OMP_NUM_THREADS

BLAS summary

- For single thread performance, use libsci
- For a few threads performance use libsci
- For many threads, use ACML
- When using threads, try to restructure to allow large rank update

- Threading performance of new libsci will be improved
- libGoto is end-of-life, libsci is now Cray BLAS
  - Interlagos will be new source base

FFTs

- Cray FFT customers fall into several camps :
  1. Want fastest serial transformations and can change code
  2. Want fastest distributed transformations and can change code
  3. Need FFTW2 serial transforms, cannot change the code
  4. Need FFTW2 distributed transforms, cannot change code
  5. Need FFTW3 serial transforms, cannot change the code
  6. None of the above (usually a custom FFT or leftfield library)
- We hope to move towards giving every camp excellent performance
  1. Use FFTW3 straight or serial CRAFFT for productivity (current)
  2. Use parallel CRAFFT for performance and productivity (current)
  3. Use FFTW2-boost (imminent)
  4. Use FFTW2-boost or P-CRAFFT FFTW interface (Q310)
  5. You are in good shape (until Interlagos)
  6. Tell us what you want

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 ALLTOALLV where possible
  - Uses a more adaptive communication scheme based on input
- 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)
- Upcoming
  - C language support for serial and parallel

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
```

Adaptation in CRAFFT

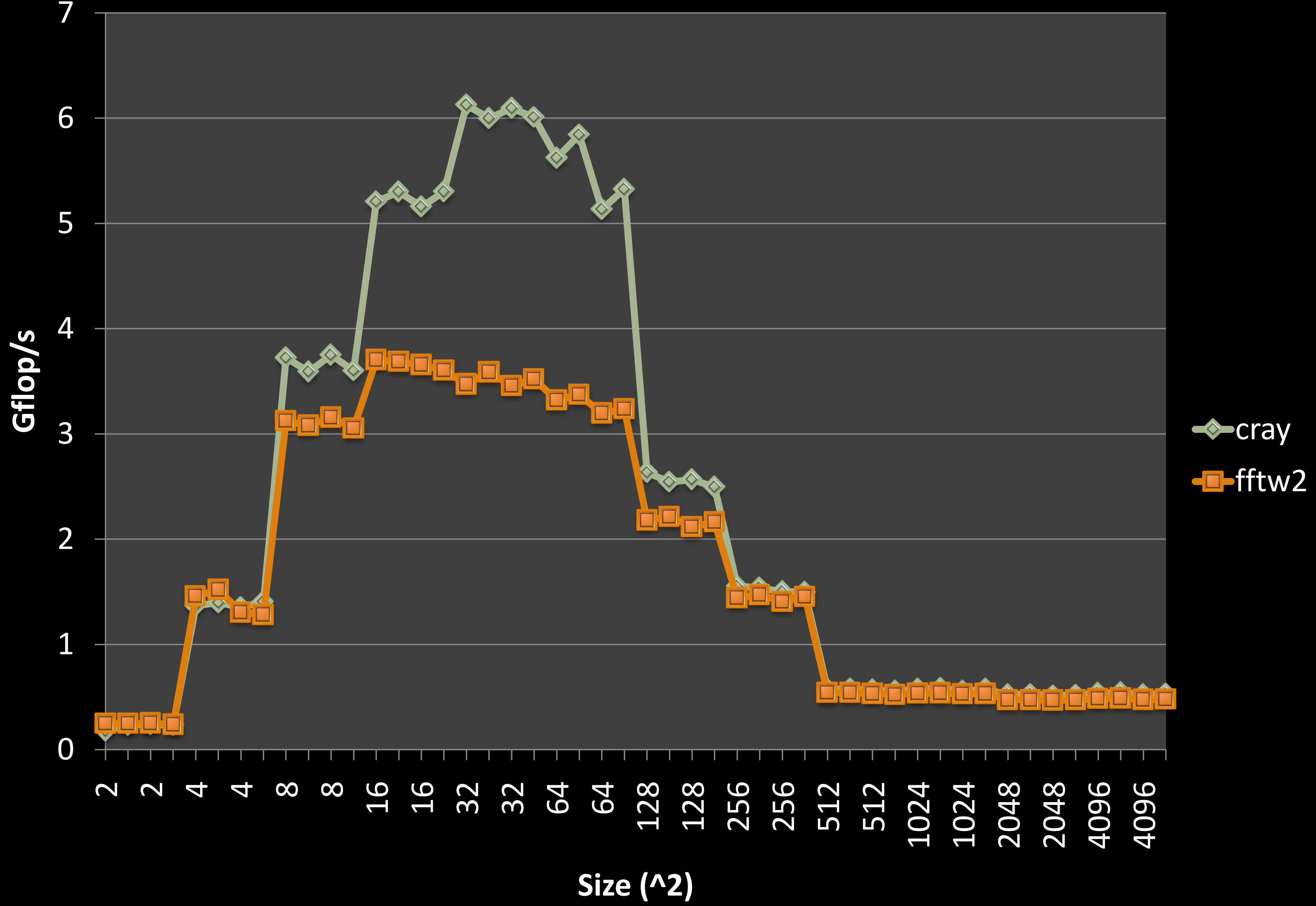
- Currently, CRAFFT out performs FFTW sometimes but not always
- We are constructing a performance model to help provide the best implementation at all times

- Adaptive CRAFFT – fall back to FFTW when we know this is the best approach

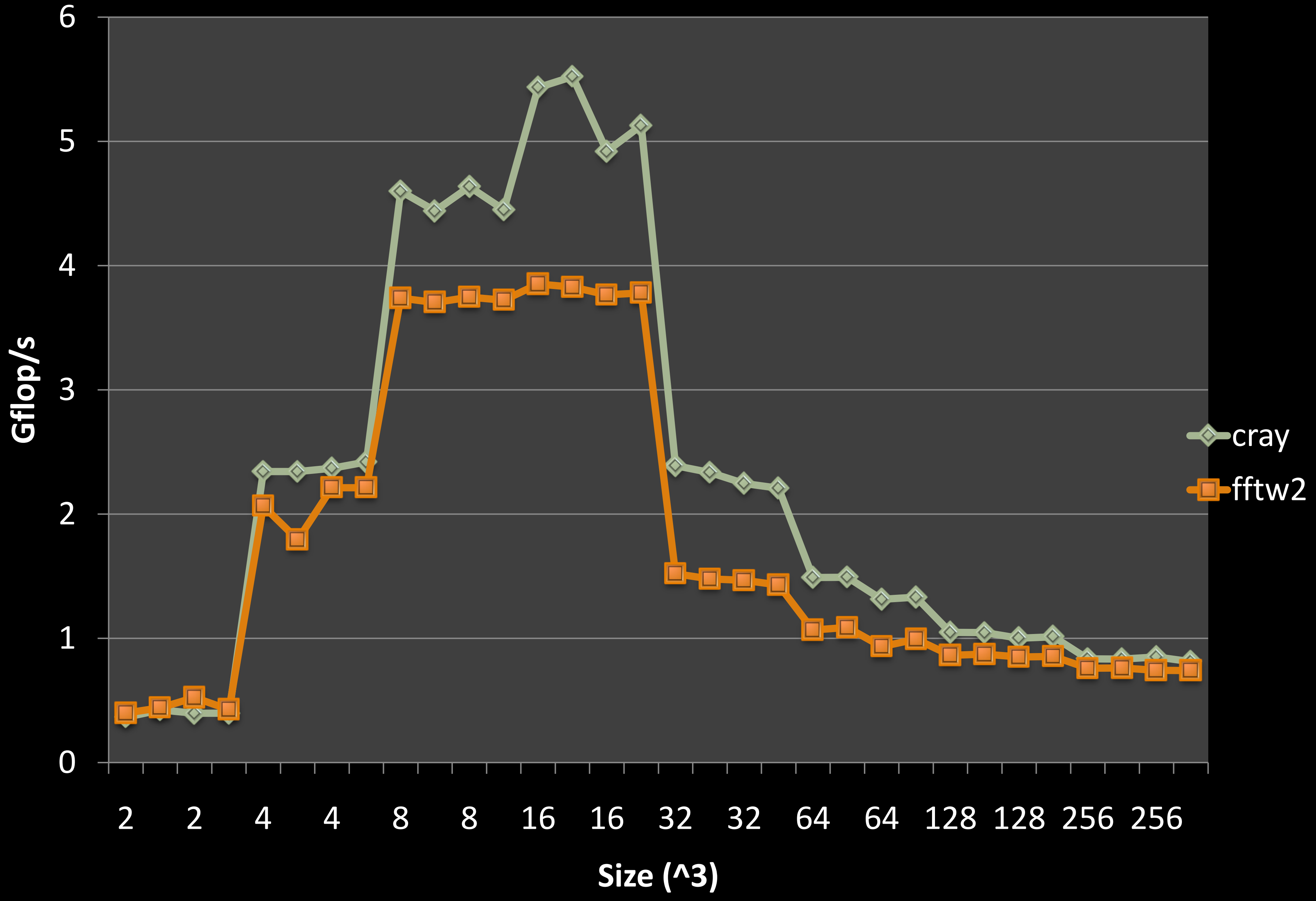
FFTW2-boost

- Cray optimized the FFTW2 code generator to
  - perform SSE vectorization
  - Better schedule instructions
- Result is a FFTW2 library that performs comparative to FFTW3 (slightly better sometimes)
- R2c, c2r, c2c
- 1d, 2d, 3d
- We welcome Beta users of this library, please contact us
- Beta process will help us devise a packaging strategy for this library

perf of fftw2 vs Cray for serial 2dfft quadcore XT5 c2c



perf of fftw2 vs cray for serial 3dfft data XT5 quadcore 2.3 GHz, c2c



Gemini Optimization of FFT

- Well understood problem in FFTs
  - Computation is O(N log N)
  - Communication is O (N)
  - For small N communications dominate
  - Limited by bisection bandwidth
- Communications optimization becomes critical
- One sided communications and PGAS can be of help with this:
  - Perform all-to-all pieces as they arise in stages 1 and 2 of FFT
  - Data decomposition is extremely important in deciding comms complexity
  - Been some good work published but no vendor has capitalized
- Our approach will be multifaceted and **we seek applications collaboration**
- The communications optimization of FFT is appearing to be an auto-tuning problem (combine this with serial optimization and the search is huge)

FFT survey

- Use custom or leftfield FFT ?
- FFTW2 v FFTW3 ?
- Use FFTW2 distributed transforms?
- What data distributions are required
- 2d versus 3d
- R2c, c2r, c2c ?
- Does anyone ever do 1d parallel FFT (except in G-FFT)?

Upcoming releases

- FFTW2-boost (??)
  - We need your feedback on the packaging
- P-CRAFFT FFTW2 interface (July 2010)
- Adaptive P-CRAFFT (July)

Sparse

- At one time Cray provided both
  - Custom sparse direct solvers
  - Custom sparse iterative solvers
- There has been an evolution towards using standardized frameworks such as Trilinos & PETSc
- Today, we attempt to provide that same performance boost while maintaining productivity
- CASK library – optimizes sparse matrix operations on Cray computers whilst being invisible to the user
  - Cray Trilinos distribution
  - Cray PETSc distribution

PETSc (Portable, Extensible Toolkit for Scientific Computation)

- 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 (ScaLAPACK, SuperLU_DIST)
  - Fortran and C support
  - Newton-type nonlinear solvers
- Extremely large user community in US and Europe
- http://www-unix.mcs.anl.gov/petsc/petsc-as

Usage and External Packages

- Cray provides
  - Hypre: scalable parallel preconditioners
  - ParMetis: parallel graph partitioning package
  - MUMPS: parallel multifrontal sparse direct solver
  - SuperLU: sequential version of SuperLU_DIST
- To use Cray-PETSc, load the appropriate module :

  module load petsc

  (or) module load petsc-complex

  (no need to load a compiler specific module)
- Treat the Cray distribution as your local PETSc installation

Trilinos

- The Trilinos Project http://trilinos.sandia.gov/

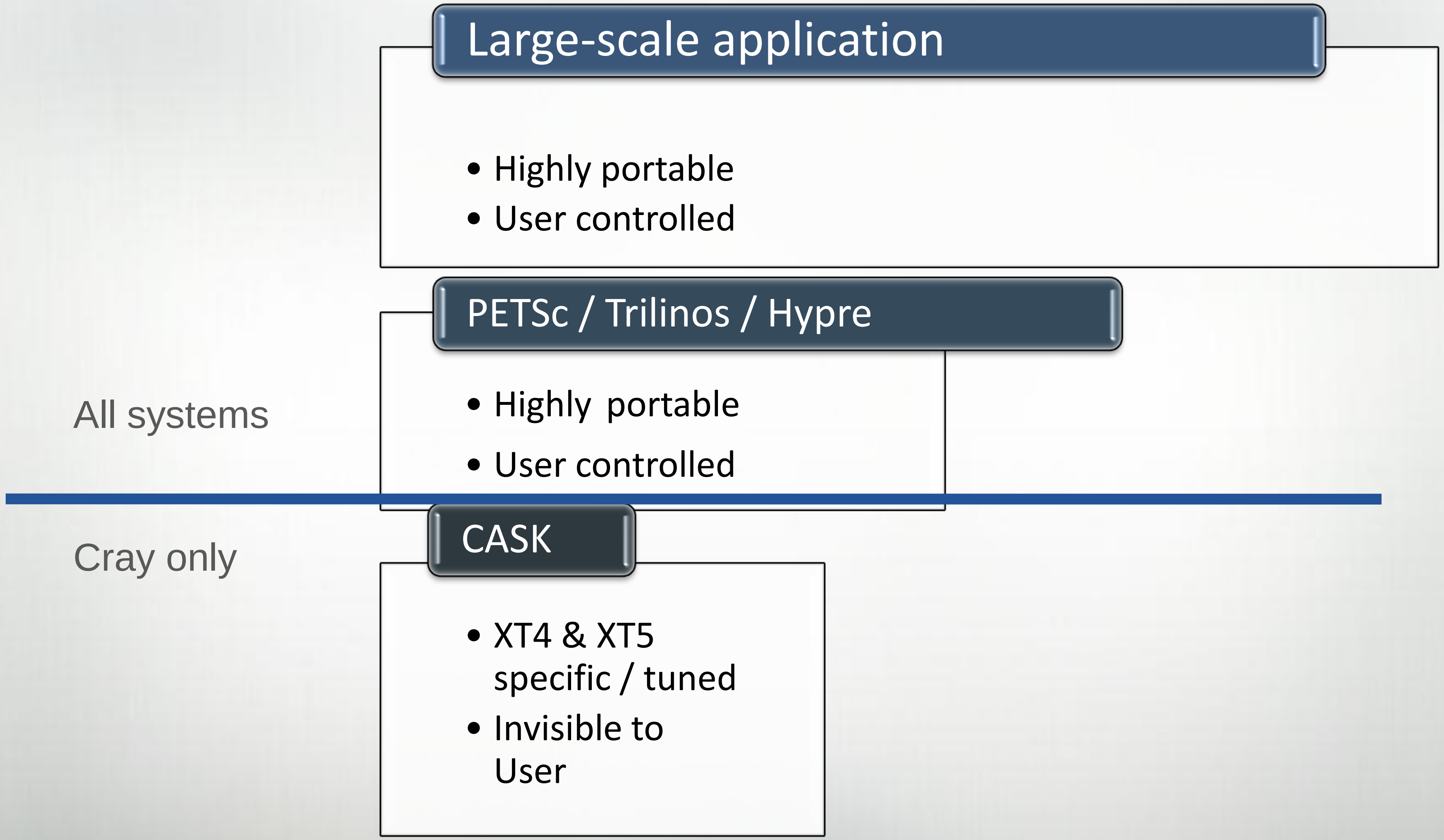
  "an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems"
- A unique design feature of Trilinos is its focus on packages.
- Very large user-base and growing rapidly. Important to DOE.
- Cray's optimized Trilinos released on January 21 2010
  - Includes 50+ trilinos packages
  - Optimized via CASK
  - Any code that uses Epetra objects can access the optimizations
- Usage :

```
module load trilinos
```

Cray Adaptive Sparse Kernel (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

Support Model



Large-scale application

- Highly portable
- User controlled

PETSc / Trilinos / Hypre

- Highly portable
- User controlled

All systems

Cray only

CASK

- XT4 & XT5 specific / tuned
- Invisible to User

CASK on MC12

- MC12 is the first entirely automated CASK
- ATF used for all stages
  - Codegen
  - Testing, search
  - Execution
  - Automation of adaptive lirbrary

- Release July 2010

- Future CASK work
- Block CSR kernels for MC
- Internal block representation for CSR

Iterative Refinement Toolkit

- Solves linear systems in single 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 usage

Decide if you want to use advanced API or benchmark API

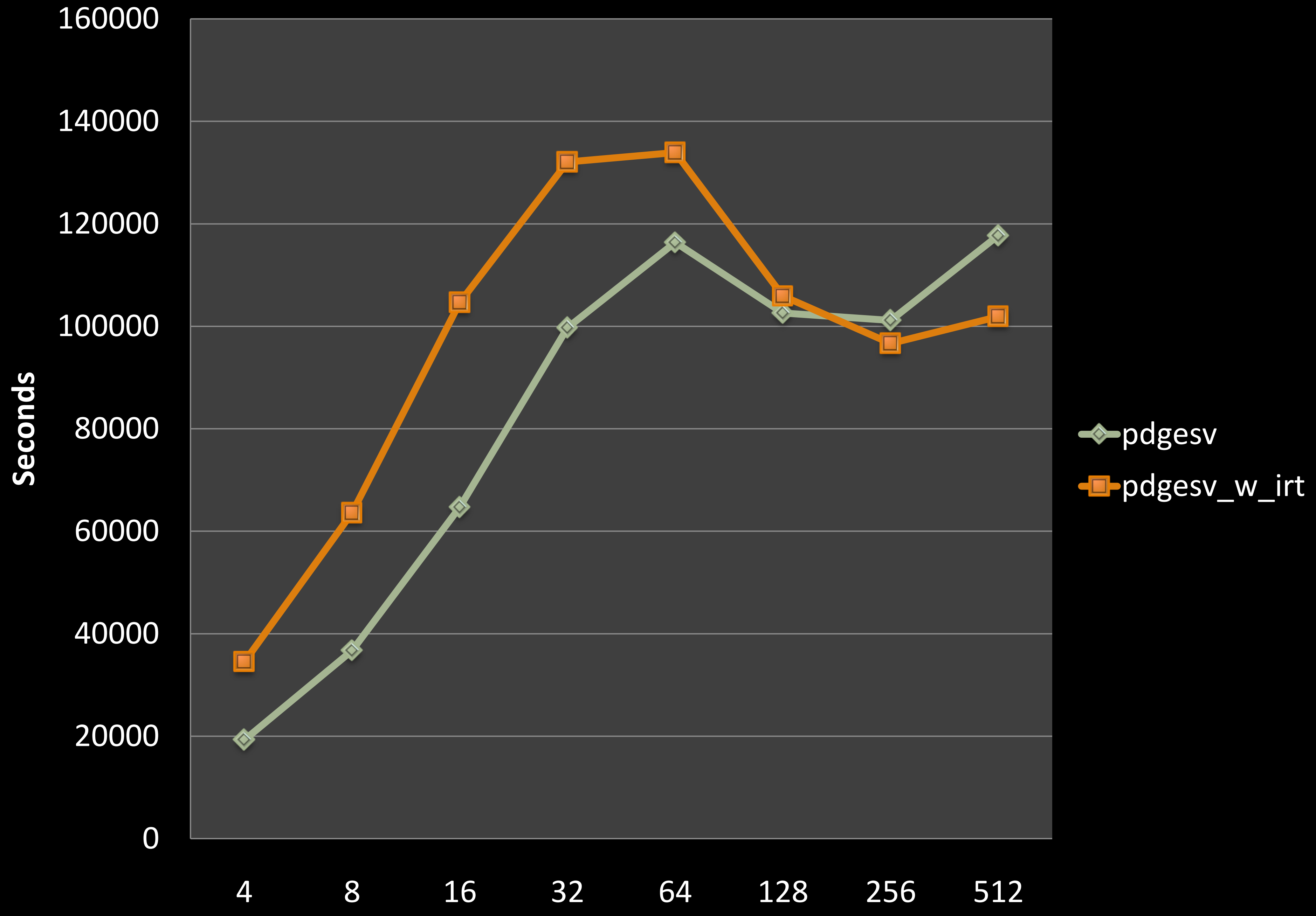
benchmark API :

```
setenv IRT_USE_SOLVERS 1
```

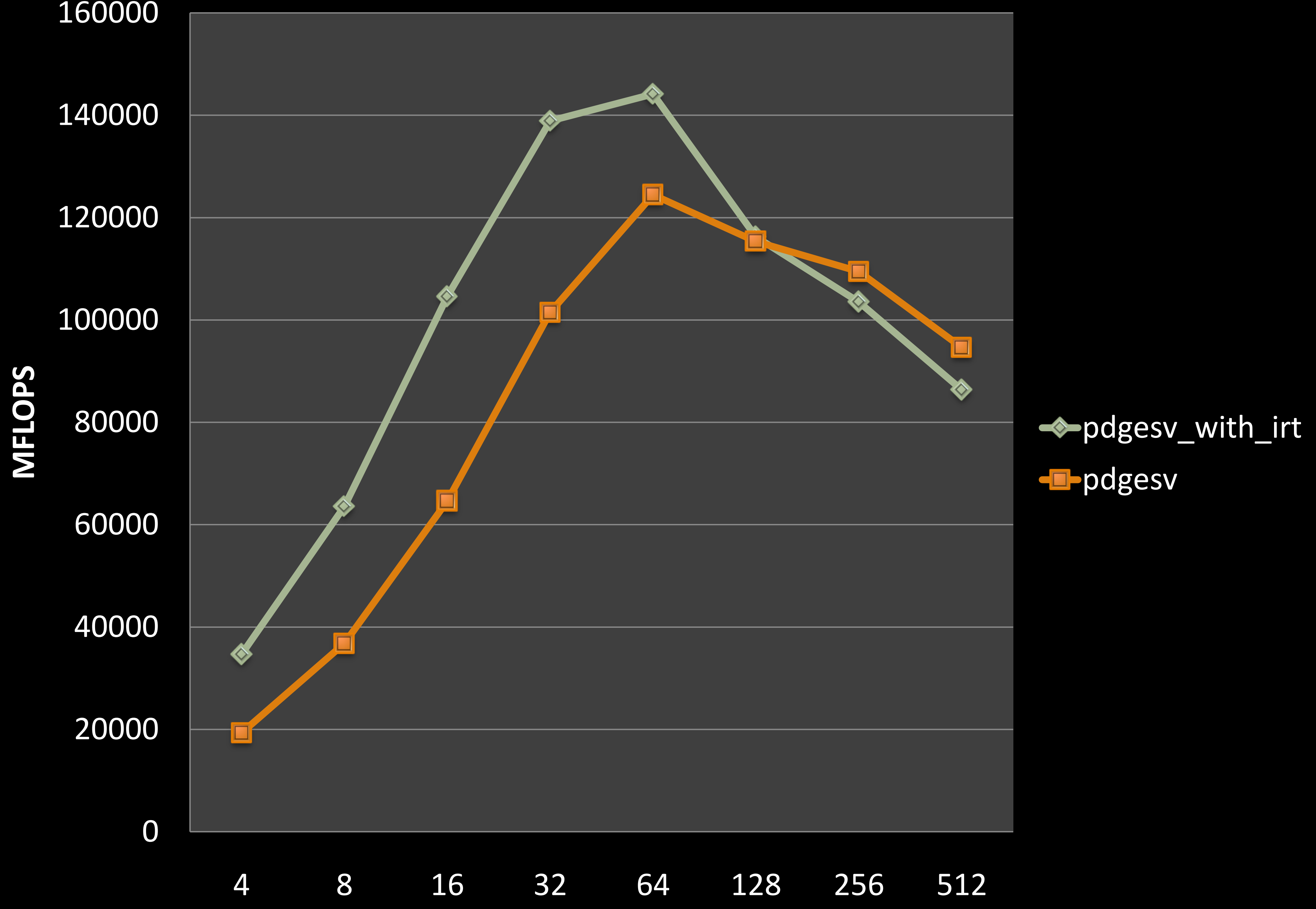
advanced API :

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

PDGESV on MC8



PDGESV on MC12



Recent Releases

- LibSci 10.4.2 February 18th 2010
  - OpenMP-aware LibSci
  - Allows calling of BLAS inside or outside parallel region
  - Single library supported
    - No multi-thread library and single thread library (-lsci and –lsci_mp)
    - Performance not compromised

    (there were some usage restrictions with this version)
- LibSci 10.4.3 April 2010
  - Parallel CRAFFT improvements
  - Fixes usage restrictions of 10.4.2
  - OMP_NUM_THREADS required (not GOTO_NUM_THREADS)
- Upcoming
  - PETSc 3.1.0 June
  - Trilinos 10.2 May 20

ScaLAPACK optimization on Gemini

- XT and X1/X2 versions of ScaLAPACK communications were tuned heavily
- XT does not lend itself well to communications optimization of DLA
- Gemini allows us to revisit many of the techniques that we used before

Specializations

- The adaptive model can give you more performance than GP
- Where you can give specific routine plus any or all of
  - Problem size
  - Block sizes
  - Density (sparse)
- Contact us directly and work with us to add specializations to libsci
- This applies to
  - BLAS
  - LAPACK
  - ScaLAPACK
  - FFT serial
  - FFT distributed
  - CASK (PETSc and Trilinos)

Summary

- BLAS
  - Use libsci for single thread or small threads, ACML for large threads
  - Improve work-per-thread with large rank operations
- P-CRAFFT currently can be of assistance (performance model complicated)
- Upcoming P-CRAFFT with adaptation and FFTW-boost will add FFT performance
- General-purpose tunings of ScaLAPACK are not effective
  - Need specialization for better tuning
  - IRT is extremely useful for speed-up
- Gemini
  - Expect good ScaLAPACK improvements
  - Expect excellent FFT improvements
- Specialization – we have the tools to provide tunings for your problem