CUG 2008 HELSINKI • MAY 5–8, 2008
CROSSING THE BOUNDARIES

FFT libraries on Cray XT: CRay Adaptive FFT (CRAFFT)

Jonathan Bentz

Cray Inc.

CRAY
THE SUPERCOMPUTER COMPANY

Outline

- Background
- Current FFT libraries on XT
- CRAFFT design
  - Example interfaces
- Performance Results
- Future plans
- Questions?

Fourier Transform Background

- Discrete Fourier Transform (DFT)
  - Transforms an array x(0:N-1) into X(0:N-1)
  - Calculation by the definition is a *O*(N^2) algorithm

$$X_k = \sum_{j=0}^{N-1} x_k \exp\left(\frac{-2\pi ijk}{N}\right), i = \sqrt{-1}$$

- Fast Fourier Transform (FFT)
  - Algorithm to calculate the DFT using *O*(N log N)
  - Algorithm is dependent on N
- Applications (among many)
  - Signal processing
  - Solving PDE

Current FFT libraries on XT

- FFTW (MIT, Frigo & Johnson, fftw.org)
  - Serial performance is very competitive
    - SIMD code for x86
    - Sophisticated run-time tuning mechanisms
  - Extremely flexible interface
    - FFT for almost any data distribution you can imagine
  - Complicated and tedious interface
  - Substantive differences between versions 2 and 3
    - Interfaces are incompatible
    - Parallel transforms in version 2 only
    - Superior serial performance in version 3
- ACML (AMD, amd.com)
  - Performance is not spectacular
    - Especially on non-powers of 2

FFT libraries common practice

- Execution of FFT in application code generally has two steps
  1. **PLANNING** stage
     - Initialize the FFT library based on the FFT size
       - Some libraries pre-compute a table of trigonometric values
       - FFTW is able to try out various FFT of that size and choose the fastest one
         - Often this can take orders of magnitude longer than the actual execution of the FFT
         - An FFTW_PATIENT plan for size 512^3 FFT takes **2758** sec to plan and **9.7** sec to execute!!!
  2. **EXECUTION** stage
     - Execute the FFT using the information from the Planning stage

Major problem with FFT libs

- Which library to choose?
  - We want the best possible FFT performance
    - To date, we have seen excellent performance from FFTW
  - FFTW also has a rich set of options for different data distributions
  - Do NOT want to change application code frequently
- How to use the complicated interfaces???
  - FFTW can be really difficult to use
    - E.g., 2d FFT with LDA > size, 14 arguments!!!

```
call dfftw_plan_many_dft(plan,rank,n,howmany, &
                         input,inembed, &
                         istride,idist, &
                         output,onembed, &
                         ostride,odist, &
                         expon,FFTW_flags)
```

CRAFFT library solves this problem

- CRAFFT is designed with simple-to-use interfaces
  - Planning and execution stage can be combined into one subroutine call
  - Underneath the interfaces, CRAFFT calls the appropriate FFT kernel
- CRAFFT provides both offline and online tuning
  - Offline tuning
    - Which FFT kernel to use
    - Pre-computed PLANs for common-sized FFT
  - Online tuning is performed as necessary at runtime as well
- At runtime, CRAFFT adaptively selects the best FFT kernel to use based on both offline and online testing (e.g. ACML, FFTW, Custom FFT)

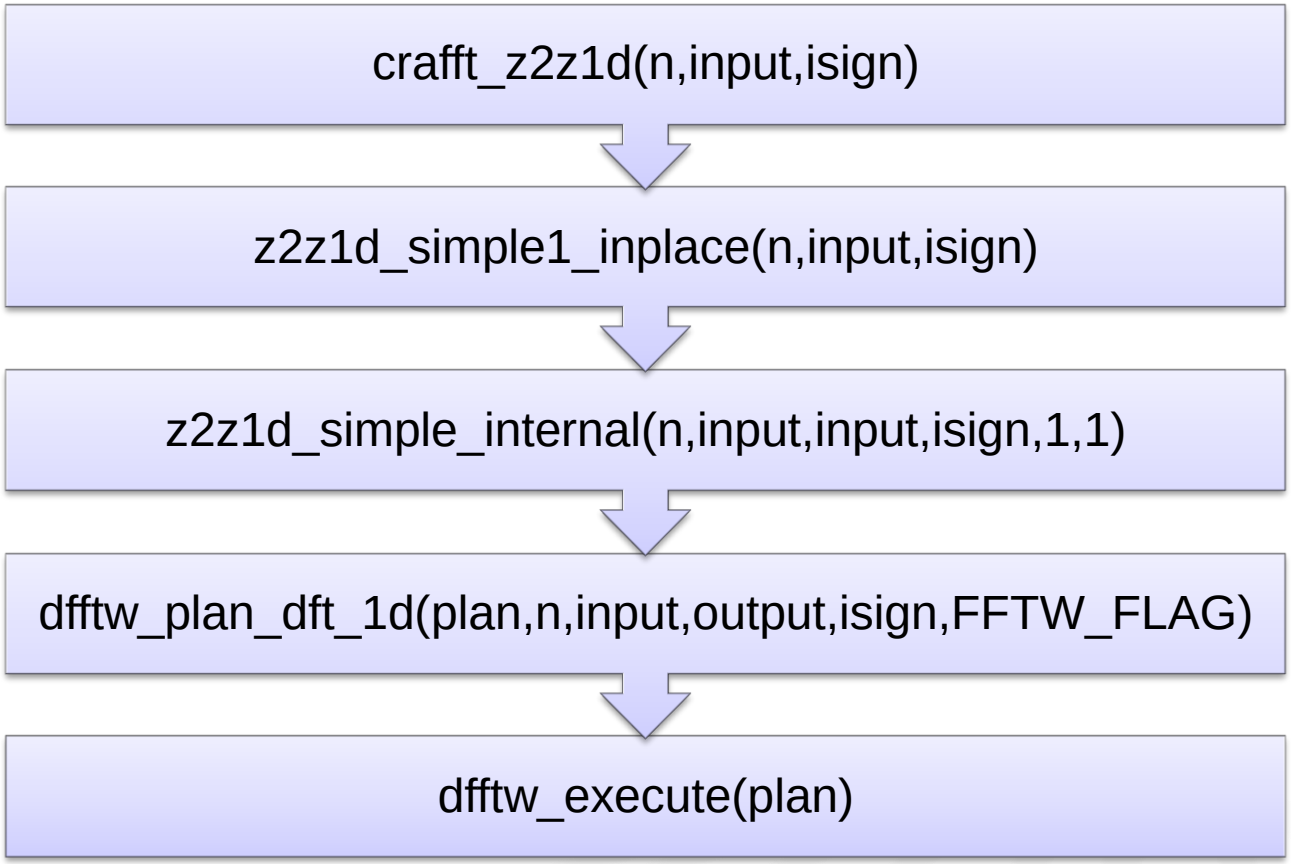
User Interface Choices

- Cray-style interface (mostly for legacy compatibility)
  - **ZZFFT(…)**; 1d complex-to-complex double precision FFT
- Simple interface
  - **CRAFFT_z2z1d(size,array,isign)**
    - Just the basics, size and array locations
    - All internals, including possible temporary memory allocation and tuning are taken care of
    - The easiest choice for users
- Advanced interface
  - **CRAFFT_z2z1d(size,array,isign,workspace,PLANNING)**
    - In addition to size and array, user also provides workspace and planning parameters
    - In 2D and 3D, the leading dimension type args can be used

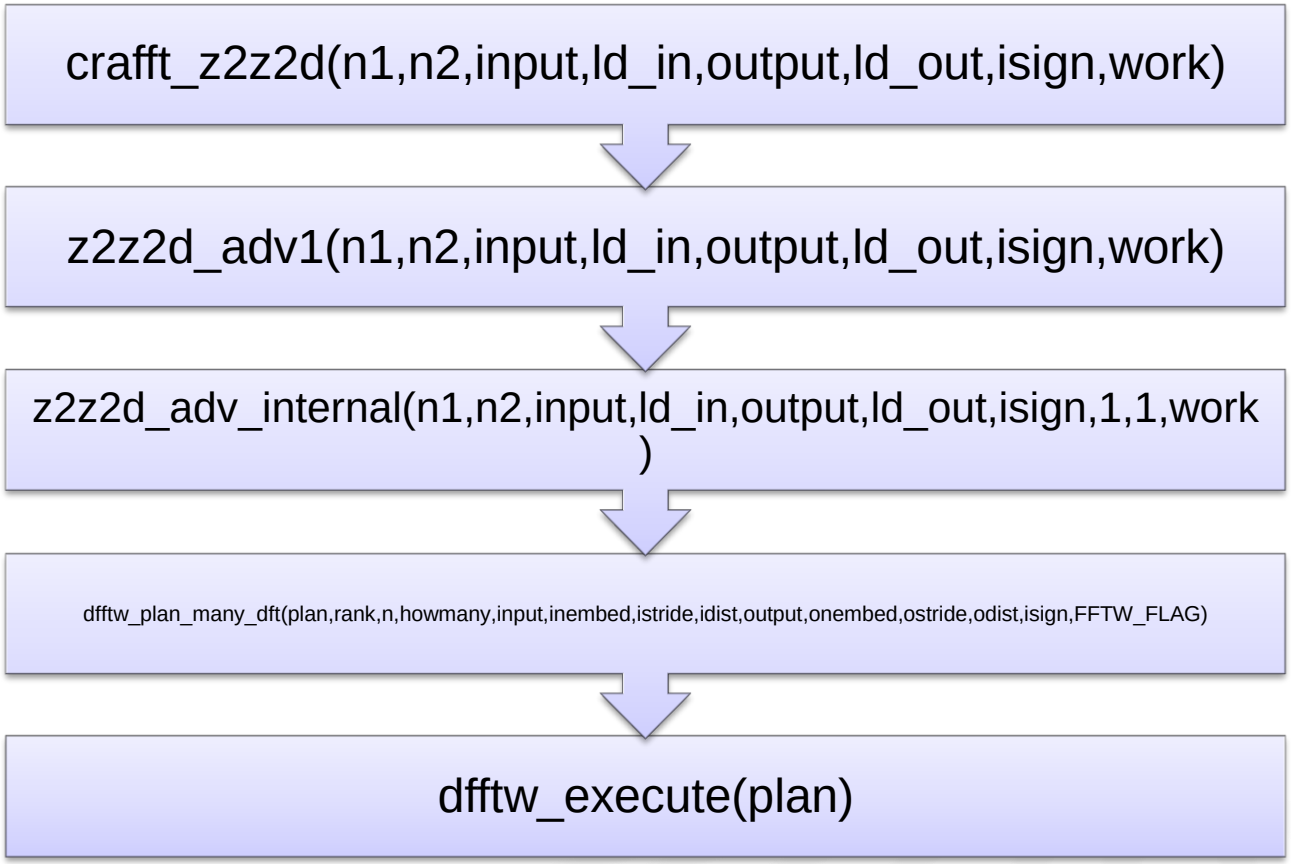
Interfaces (cont.)

- All subroutine names have the form crafft_**α**2**βθ**D
  - **α, β** = S,D,C or Z like netlib, i.e., D = double precision real, C = single precision complex
  - **θ** = 1, 2 or 3, i.e., the dimension of the transform
  - E.g., **crafft_d2z1d** is a double real to double complex transform in 1d
- Interface makes use of F90 modules to overload the names
  - Users must put “use crafft” in their fortran source code
  - 1D complex to complex examples:
    - **crafft_z2z1d**(size,array,isign)
      - in-place
    - **crafft_z2z1d**(size,input,output,isign)
      - out-of-place

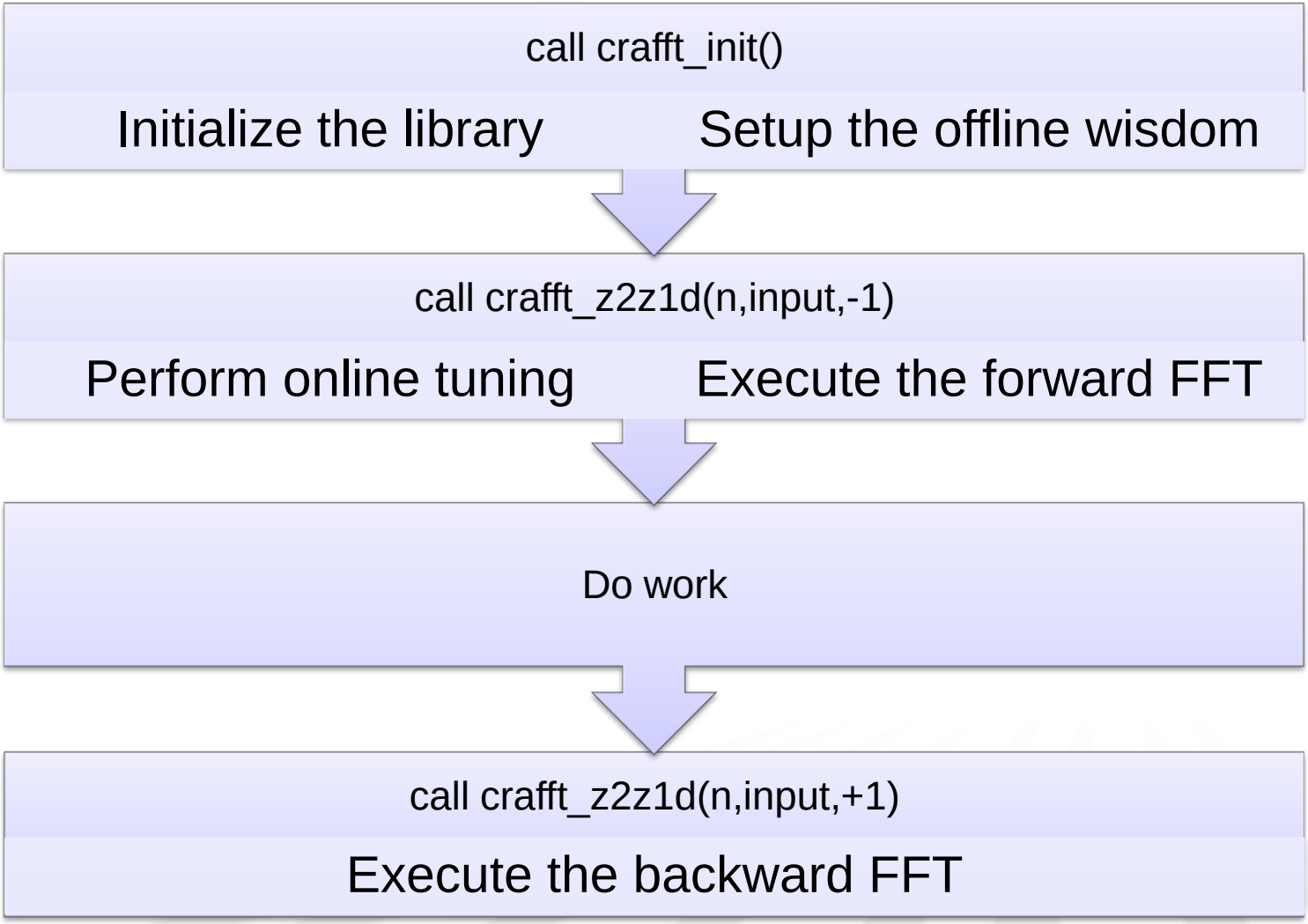
Simple 1d CRAFFT call resolves to…



Advanced 2d CRAFFT call resolves to…



CRAFFT user code calling sequence



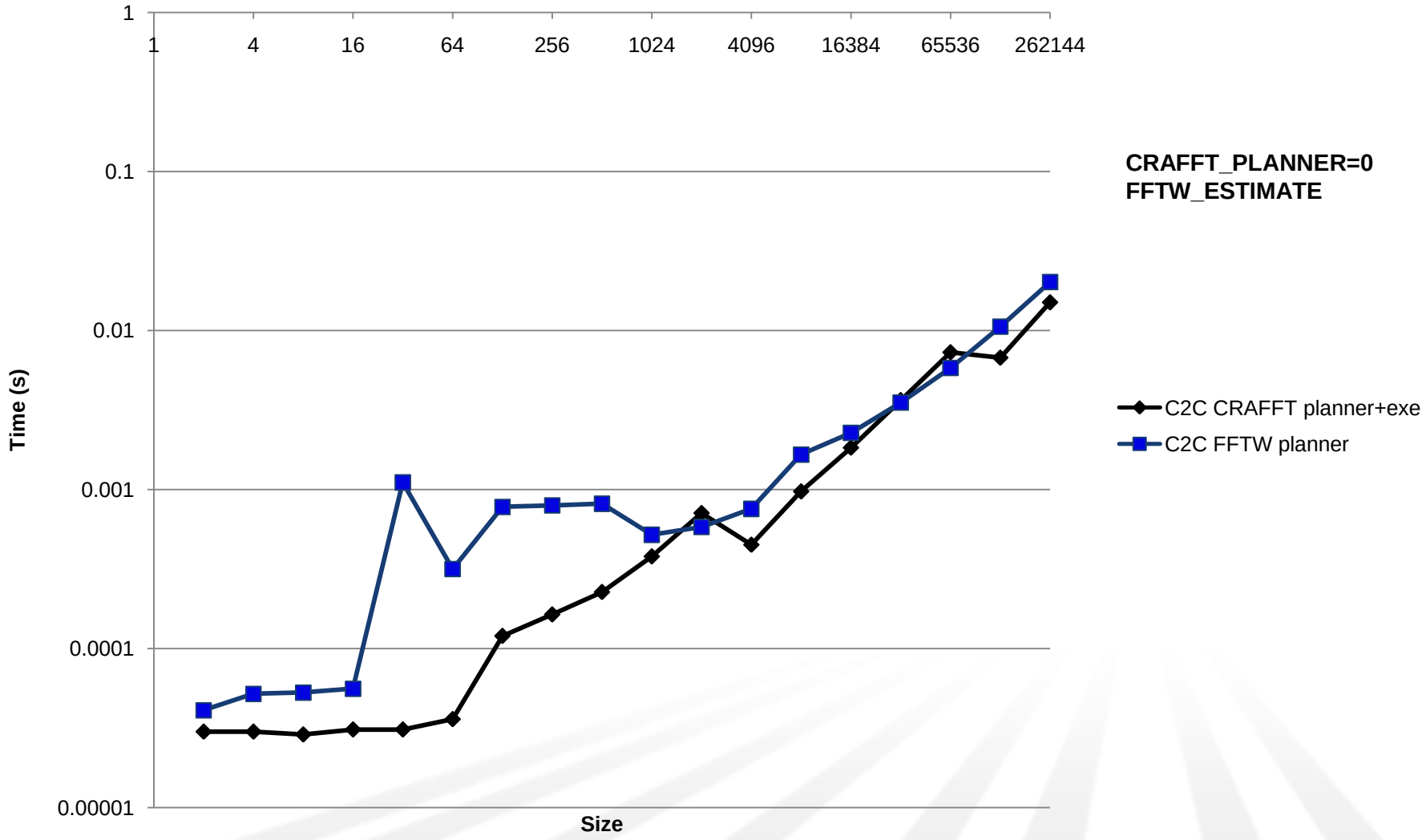
CRAFFT 1.0alpha (current status)

- Largely FFTW centric
- Includes FFTW offline wisdom to minimize expensive online planning
- Allows simple interface into advanced FFTW functionality

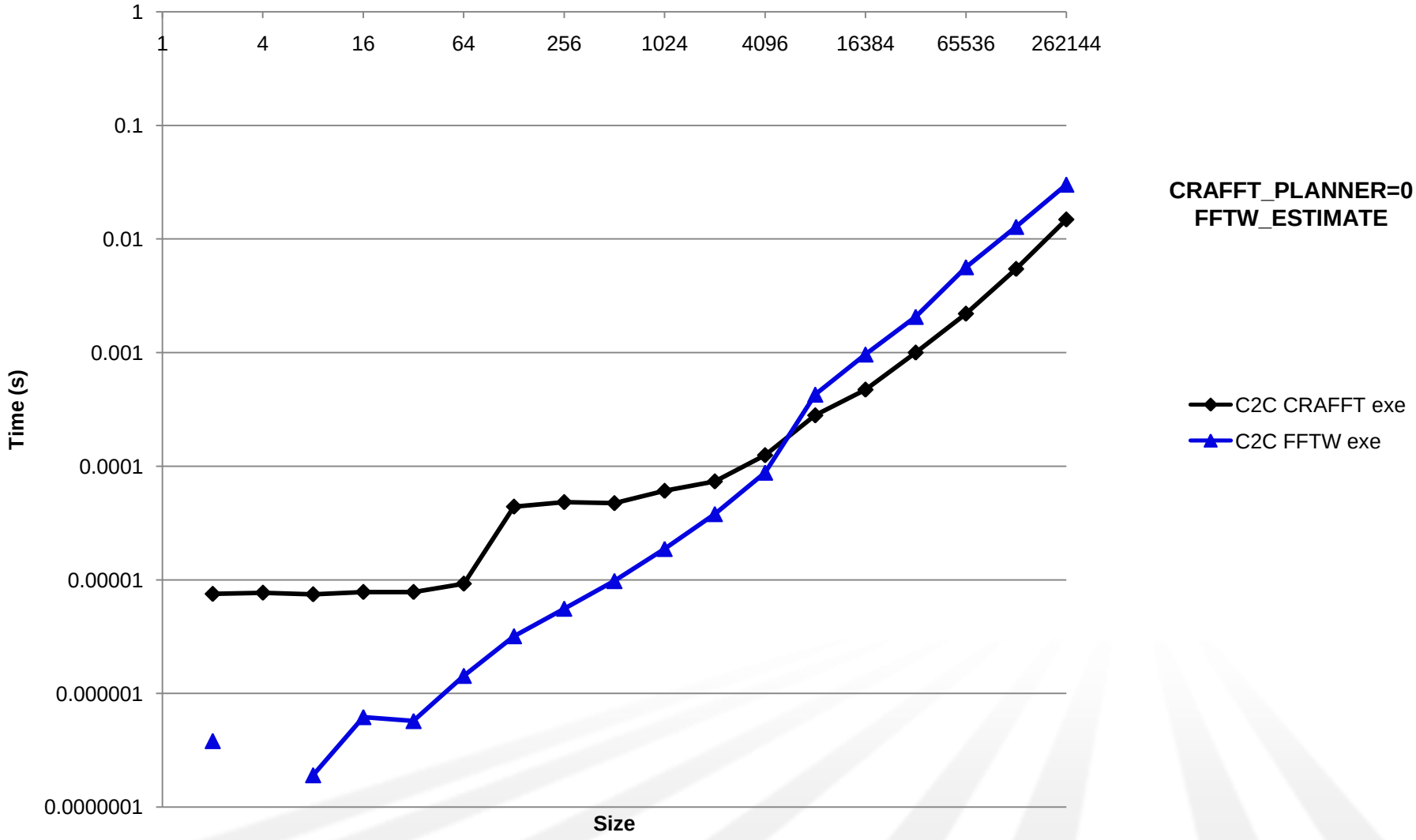
- Proposed release in summer 2008

- PERFORMANCE???

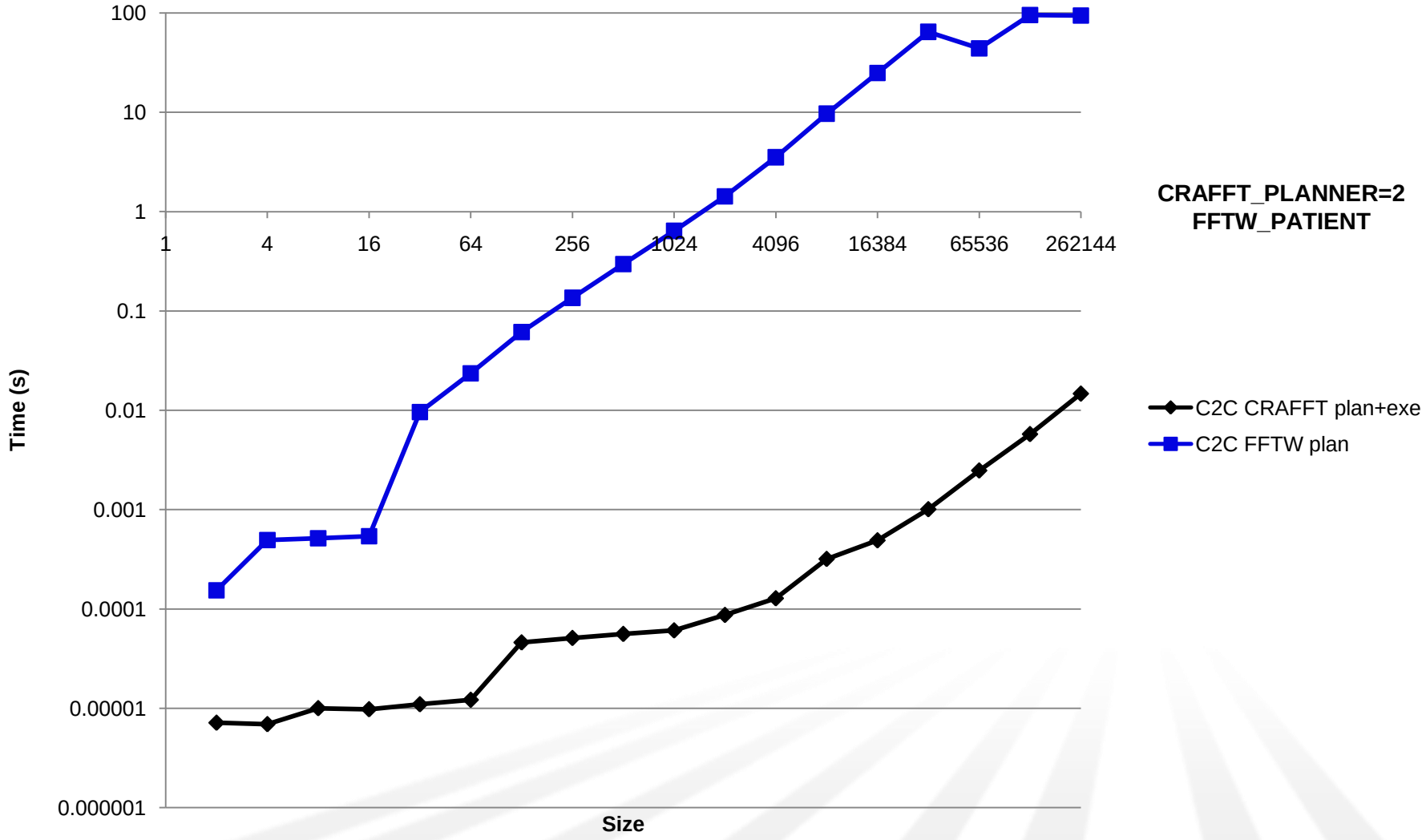
Walltime vs. size, 1D C2C FFT planner



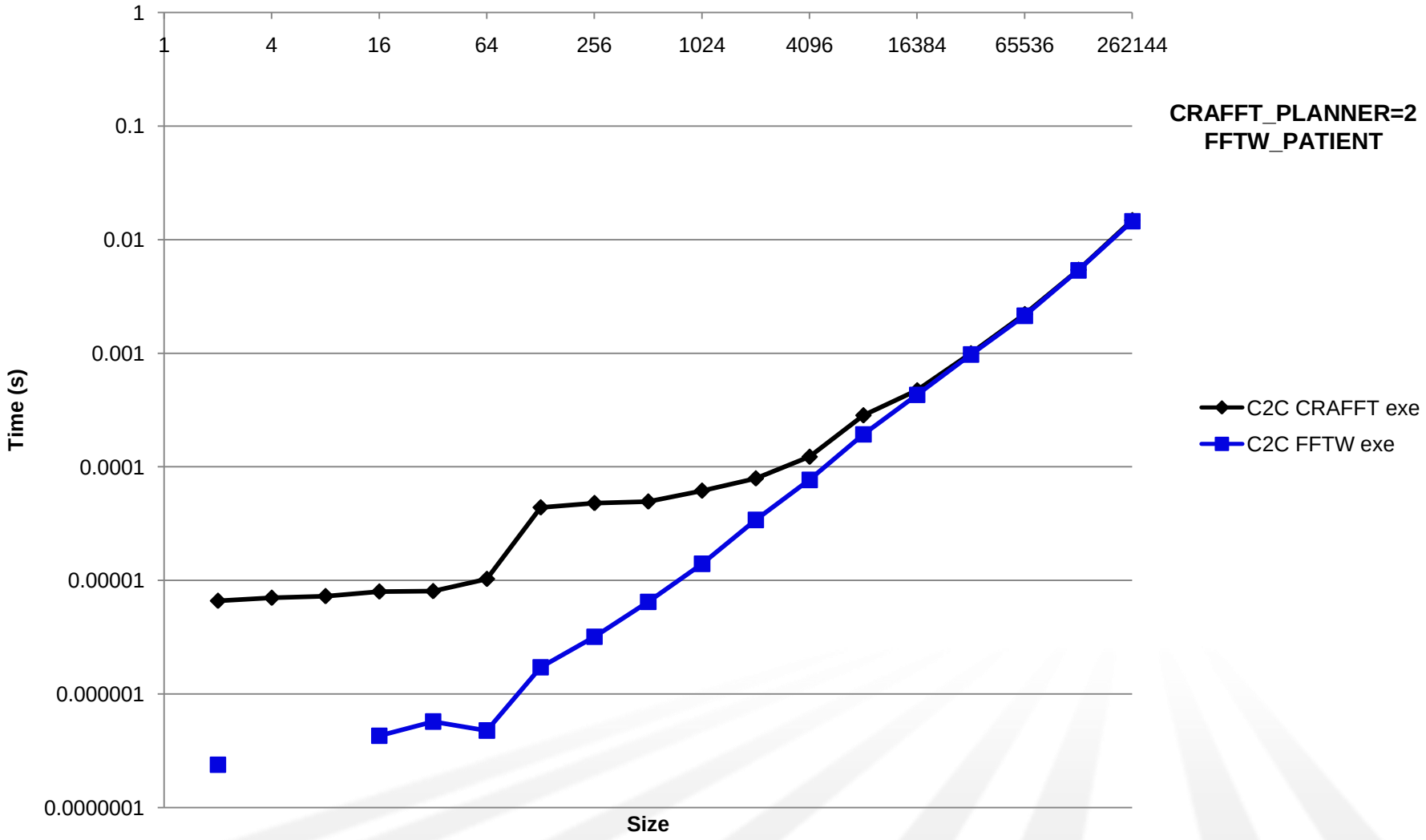
Walltime vs. size, 1D C2C FFT execute



Walltime vs. size, 1D C2C FFT planner



Walltime vs. size, 1D C2C FFT execute



Summary

- CRAFFT provides a simple interface into FFT for XT
  - Avoid those nasty 14 argument FFTW calls!
- CRAFFT overhead is very minimal
- CRAFFT performance is really excellent when using common-sized FFT
  - CRAFFT avoids expensive planning stage

Future Work

- Additional libraries “under-the-covers”
  - Complete libraries, e.g., SPIRAL (CMU, Franchetti et. al., spiral.net)
  - Targeted tuning of kernels for specific sizes
- Parallel FFT
  - Again, provide a simple, intuitive interface and handle the details transparently
  - Provide multiple data distributions

QUESTIONS???

- Email: jnbntz@cray.com