Automatic Generation of the HPC Challenge's Global FFT Benchmark for BlueGene/P

Franz Franchetti[1], Yevgen Voronenko[2], Gheorghe Almasi[3]

[1]Carnegie Mellon University and SpiralGen, Inc.
[2]AccuRay, Inc., [3]IBM Research

Presenter: Richard M. Veras
Carnegie Mellon University

This work was supported by NSF, ONR, and ANL BlueGene/Q ESP

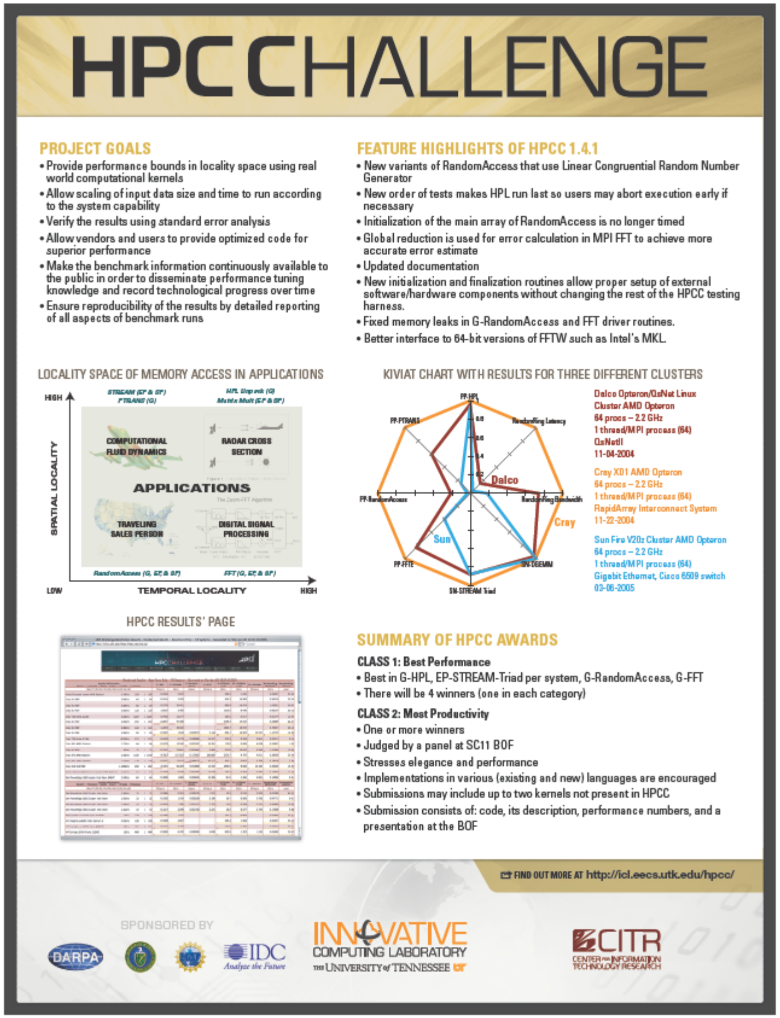
The HPC Challenge's Global FFT Benchmark

HPC Challenge

- New HPC Benchmark suite
- HPL, STREAM, RandomAccess, PTRANS, FFT, DGEMM, and b_eff
- Better characterization than HPL

Global FFT

- Large, parallel 1D FFT across the whole machine
- Strongly limited by the machine's communication system
- Baseline implementation: FFTE



http://icl.cs.utk.edu/hpcc/

Goal: Auto-generate efficient Global FFT implementation

Outline

- Spiral: Library Generation
- MPI-Friendly Global FFT Algorithm
- Experimental Results
- Summary

Outline

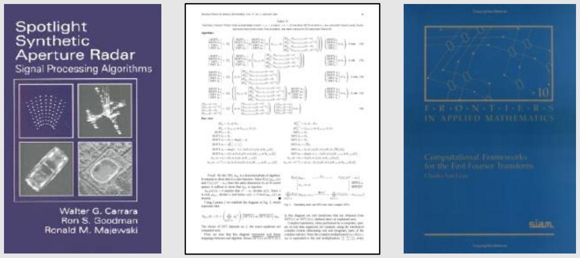
- **Spiral: Library Generation**
- MPI-Friendly Global FFT Algorithm
- Experimental Results
- Summary

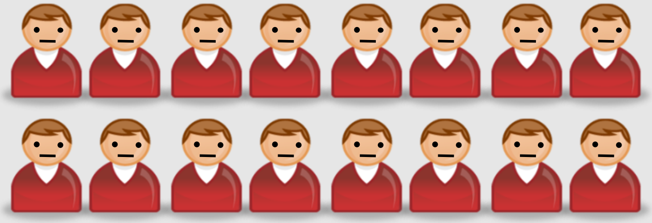
M. Püschel, F. Franchetti, Y. Voronenko: **Spiral.** Encyclopedia of Parallel Computing, D. A. Padua (Editor), 2011.

Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo: **SPIRAL: Code Generation for DSP Transforms.** Special issue, Proceedings of the IEEE 93(2), 2005.

Spiral: Automating Library Tuning

Traditionally

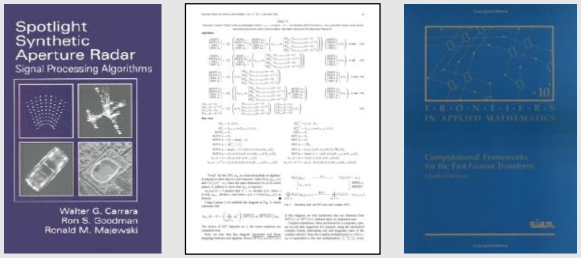




High performance library optimized for given platform

Comparable performance

Spiral Approach



Spiral

High performance library optimized for given platform

Spiral's Formal Framework

- **Transform = Matrix-vector multiplication**
  Example: Discrete Fourier transform (DFT)

$$x \mapsto y = T \cdot x$$

input vector (signal) → x

output vector (signal) → y

transform = matrix → T

- **Fast algorithm = sparse matrix factorization = SPL formula**
  Example: Cooley-Tukey FFT algorithm

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 1 & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ 1 & \cdot & -1 & \cdot \\ \cdot & 1 & \cdot & -1 \end{bmatrix} \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & j \end{bmatrix} \begin{bmatrix} 1 & 1 & \cdot & \cdot \\ 1 & -1 & \cdot & \cdot \\ \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \end{bmatrix}$$

$$\mathrm{DFT}_4 = (\mathrm{DFT}_2 \otimes \mathrm{I}_2)\,\mathrm{T}^4_2\,(\mathrm{I}_2 \otimes \mathrm{DFT}_2)\,\mathrm{L}^4_2$$

Transforms and Breakdown Rules

$$\mathbf{DFT}_n \rightarrow (\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^n_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^n_k, \quad n = km$$

$$\mathbf{DFT}_n \rightarrow P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m)Q_n, \quad n = km,\ \mathsf{gcd}(k,m) = 1$$

$$\mathbf{DFT}_p \rightarrow R_p^T(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})D_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})R_p, \quad p \text{ prime}$$

DCT-3

DCT-4

“Teaches” Spiral about existing algorithm knowledge (~200 journal papers)

$$\mathbf{IMDCT}_{2m} \rightarrow (\mathsf{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathsf{J}_m)\left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m\right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m\right)\right) \mathsf{J}_{2m}\, \mathbf{DCT\text{-}4}_{2m}$$

$$\mathbf{WHT}_{2^k} \rightarrow \prod_{i=1}^{t} (\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t$$

$$\mathbf{DFT}_2 \rightarrow \mathsf{F}_2$$

$$\mathbf{DCT\text{-}2}_2 \rightarrow \mathsf{diag}(1, 1/\sqrt{2})\, \mathsf{F}_2$$

$$\mathbf{DCT\text{-}4}_2 \rightarrow \mathsf{J}_2\, \mathsf{R}_{13\pi/8}$$

Base case rules

Teaches Spiral about FFT algorithms

One Approach for all Types of Parallelism

- **Multithreading** (Multicore) — $\mathrm{I}_p \otimes_{\|} A_{\mu n}, \quad \mathbf{L}^{mn}_{m} \bar{\otimes}\, \mathrm{I}_\mu$
- **Vector SIMD** (SSE, VMX/Altivec,…) — $A \hat{\otimes}\, \mathrm{I}_\nu \quad \underbrace{\mathbf{L}^{2\nu}_{2}}_{\text{isa}}, \quad \underbrace{\mathbf{L}^{2\nu}_{\nu}}_{\text{isa}}, \quad \underbrace{\mathbf{L}^{\nu^2}_{\nu}}_{\text{isa}}$
- **Message Passing** (Clusters, MPP) — $\mathrm{I}_p \otimes_{\|} A_n, \quad \underbrace{\mathbf{L}^{p^2}_{p} \bar{\otimes}\, \mathrm{I}_{n/p^2}}_{\text{all-to-all}}$
- **Streaming/multibuffering** (Cell) — $\mathrm{I}_n \otimes_2 A_{\mu n}, \quad \mathbf{L}^{mn}_{m} \bar{\otimes}\, \mathrm{I}_\mu$
- **Graphics Processors** (GPUs) — $\prod_{i=0}^{n-1} A_i, \quad A_n \hat{\otimes}\, \mathrm{I}_w, \quad P_n \otimes Q_w$
- **Gate-level parallelism** (FPGA) — $\prod_{i=0}^{n-1}{}^{\text{ir}} A, \quad \mathrm{I}_s \tilde{\otimes} A, \quad \underbrace{\mathbf{L}^{m}_{n}}_{\text{bram}}$
- **HW/SW partitioning** (CPU + FPGA) — $\underbrace{A_1}_{\text{fpga}}, \quad \underbrace{A_2}_{\text{fpga}}, \quad \underbrace{A_3}_{\text{fpga}}, \quad \underbrace{A_4}_{\text{fpga}}$

Translating a Formula into Code

Rewriting Input:

$$\underbrace{\mathrm{DFT}_8}_{\text{double}}$$

Output =

OL Formula: $(\mathrm{DFT}_2 \otimes I_4)\, T^8_4 \left(I_2 \otimes \left((\mathrm{DFT}_2 \otimes I_2)\, T^4_2\, (I_2 \otimes \mathrm{DFT}_2)\, L^4_2\right)\right) L^8_2$

Σ-OL: $\sum_{j=0}^{3} \left(S_j\, \mathrm{DFT}_2\, G_j\right) \sum_{k=0}^{1} \left(\sum_{l=0}^{1} \left(S_{k,l}\, \mathrm{diag}\left(\mathrm{t}_{k,l}\right) \mathrm{DFT}_2\, G_l\right) \sum_{m=0}^{1} \left(S_m\, \mathrm{diag}\left(\mathrm{t}_m\right) \mathrm{DFT}_2\, G_{k,m}\right)\right)$

C Code:

```
void sub(double *y, double *x) {
  double f0, f1, f2, f3, f4, f7, f8, f10, f11;
  f0 = x[0] - x[3];
  f1 = x[0] + x[3];
  f2 = x[1] - x[2];
  f3 = x[1] + x[2];
  f4 = f1 - f3;
  y[0] = f1 + f3;
  y[2] = 0.7071067811865476 * f4;
  f7 = 0.9238795325112867 * f0;
  f8 = 0.3826834323650898 * f2;
  y[1] = f7 + f8;
  f10 = 0.3826834323650898 * f0;
  f11 = (-0.9238795325112867) * f2;
  y[3] = f10 + f11;
}
```

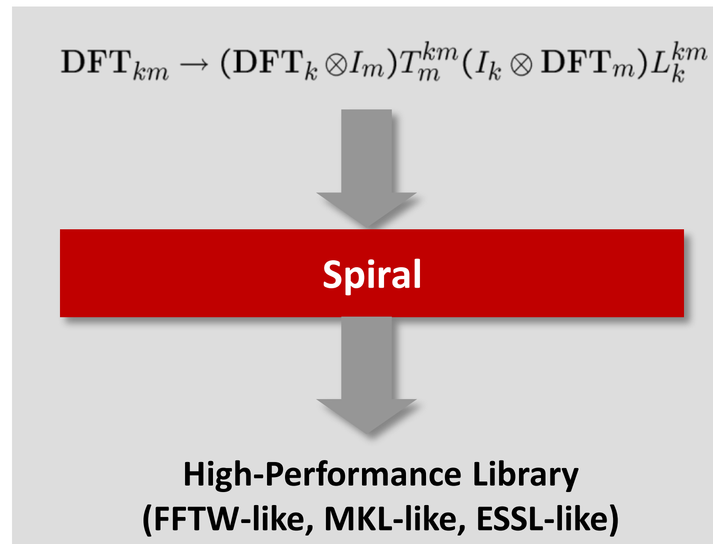
Synthesizing General Size Libraries

Input:

- **Transform**: DFT_n
- **Algorithms**: $\mathrm{DFT}_{km} \rightarrow (\mathrm{DFT}_k \otimes I_m) T_m^{km} (I_k \otimes \mathrm{DFT}_m) L_k^{km}$
  $\mathrm{DFT}_2 \rightarrow \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
- **Vectorization**: 2-way SSE
- **Threading**: Yes

Output:

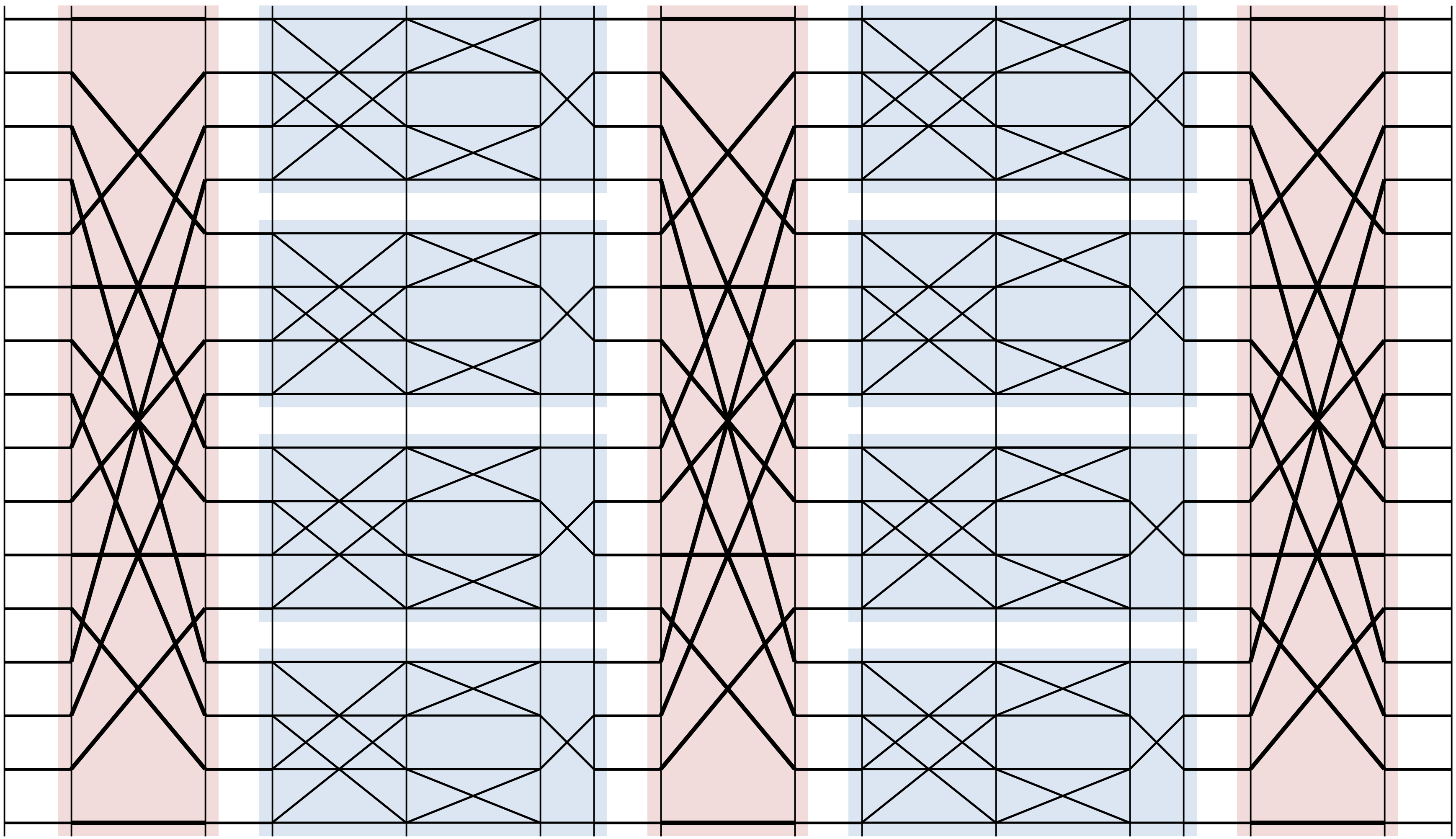
- Optimized library (10,000 lines of C++)
- For general input size
  (**not** collection of fixed sizes)
- Vectorized
- Multithreaded
- With runtime adaptation mechanism
- Performance competitive with hand-written code



Outline

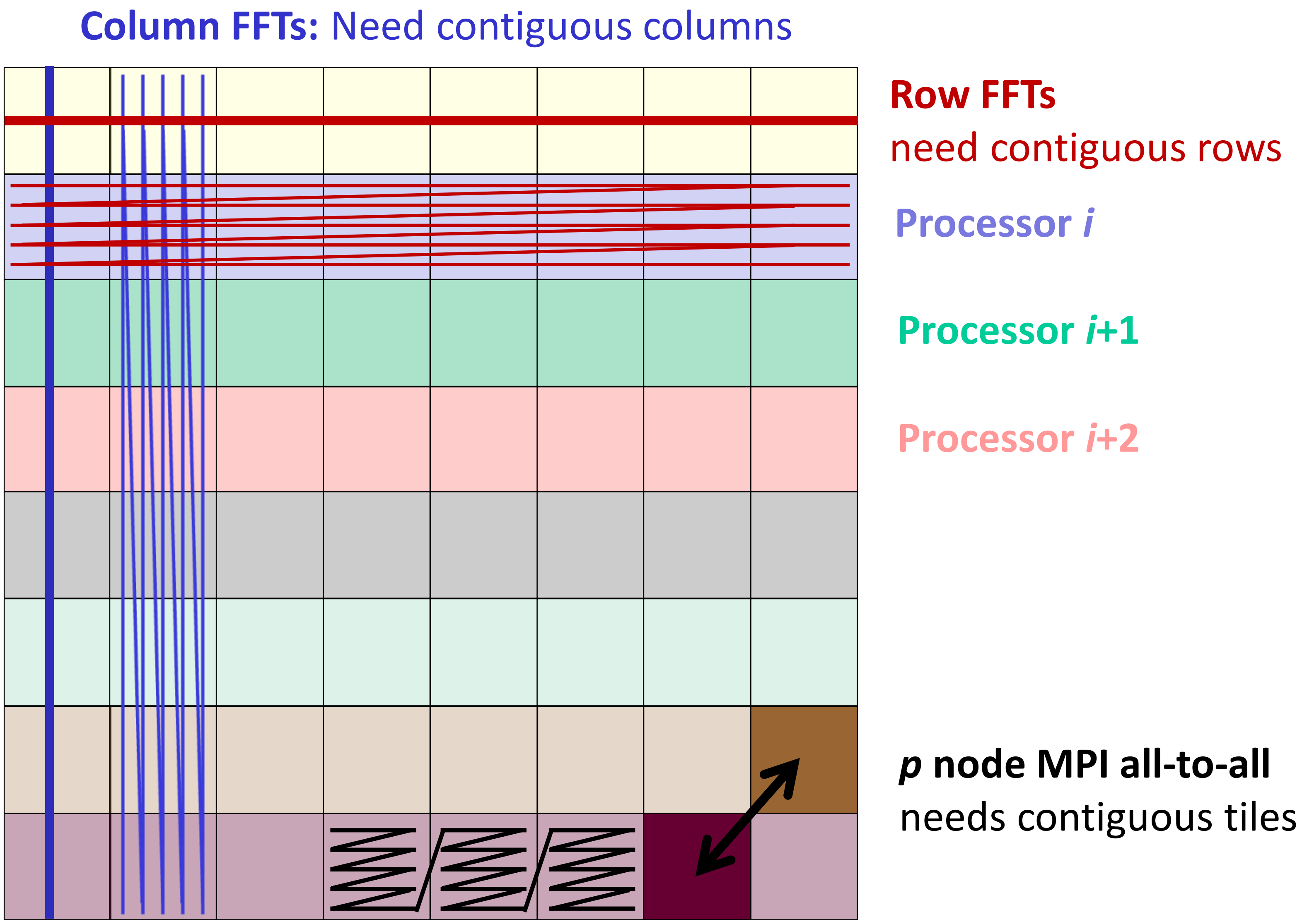
- Spiral: Library Generation
- **MPI-Friendly Global FFT Algorithm**
- Experimental Results
- Summary

FFT needs Local FFTs and Global Transposes



FFTs along rows and columns of distributed square matrix

Linear Memory vs. Tiled Memory



Requires MPI all-to-allv, explicit copy, or FFT on tiled memory

MPI All-to-all "Friendly" Six Step FFT

$$\mathrm{DFT}_{mn} = \underbrace{\left(\mathrm{I}_p \otimes (\mathrm{L}^{m}_{m/p} \otimes \mathrm{I}_{n/p})\right)}_{\text{local transpose}} \underbrace{\left(\mathrm{L}^{p^2}_{p} \otimes \mathrm{I}_{mn/p^2}\right)}_{\text{all-to-all}} \underbrace{\left(\mathrm{I}_p \otimes (\mathrm{DFT}_m \otimes \mathrm{I}_{n/p})\right)}_{\text{inplace FFT library call}}$$

$$\underbrace{\left(\mathrm{L}^{p^2}_{p} \otimes \mathrm{I}_{mn/p^2}\right)}_{\text{all-to-all}} \underbrace{\left(\mathrm{T}^{mn}_{n}\right)^{(\mathrm{I}_p \otimes \mathrm{L}^{m}_{m/p} \otimes \mathrm{I}_{n/p})} \left(\mathrm{I}_p \otimes (\mathrm{L}^{m}_{p} \otimes \mathrm{I}_{n/p})(\mathrm{I}_{m/p} \otimes \mathrm{DFT}_n)\ \mathrm{L}^{mn/p}_{m/p}\right)}_{\text{out-of-place scaled FFT library call}}$$

$$\underbrace{\left(\mathrm{L}^{p^2}_{p} \otimes \mathrm{I}_{mn/p^2}\right)}_{\text{all-to-all}} \underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}^{n}_{p} \otimes \mathrm{I}_{m/p}\right)}_{\text{local transpose}}$$

Standard batch FFT library (on 1D contiguous memory)

Specialized node FFT library: batch FFT+twiddles on 2D tiled memory

Standard MPI all-to-all on contiguous data

Node-local pre-processing (data scrambling)

SIMD Vectorization for FFT

Standard FFT

$$\mathrm{DFT}_{mn} \rightarrow (\mathrm{DFT}_m \otimes \mathrm{I}_n)\mathrm{T}^{mn}_n(\mathrm{I}_m \otimes \mathrm{DFT}_n)\,\mathrm{L}^{mn}_m$$

Automatic formula rewriting

$$\mathrm{DFT}_{mn} \rightarrow \left(\mathrm{I}_{\frac{mn}{\nu}} \otimes \mathrm{L}^{2\nu}_{\nu}\right)\left(\overline{\mathrm{DFT}_m \otimes \mathrm{I}_{\frac{n}{\nu}}} \otimes \mathrm{I}_\nu\right)\mathrm{T}'^{mn}_n \left(\mathrm{I}_{\frac{m}{\nu}} \otimes \left(\mathrm{L}^{2n}_{\nu} \otimes \mathrm{I}_\nu\right)\left(\mathrm{I}_{\frac{2n}{\nu}} \otimes \mathrm{L}^{\nu^2}_{\nu}\right)\left(\mathrm{I}_{\frac{n}{\nu}} \otimes \mathrm{L}^{2\nu}_{2} \otimes \mathrm{I}_\nu\right)\left(\overline{\mathrm{DFT}}_n \otimes \mathrm{I}_\nu\right)\right)\left(\mathrm{L}^{\frac{mn}{\nu}}_{\frac{m}{\nu}} \otimes \mathrm{L}^{2\nu}_{2}\right)$$

Vectorized arithmetic

Data reorganization
(requires architecture specific vetorization)

Short Vector FFT for ν-way vector instructions

Only 3 permutations require architecture-specific vectorization: $\mathrm{L}^{2\nu}_{2},\ \mathrm{L}^{2\nu}_{\nu},\ \mathrm{L}^{\nu^2}_{\nu}$

Works for any N=mn with $\nu^2 | N$

F. Franchetti, M. Püschel: **"Short Vector Code Generation for the Discrete Fourier Transform,"**
In Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS '03), pages 58-67

F. Franchetti, S. Kral, J. Lorenz, C. W. Ueberhuber: **"Efficient Utilization of SIMD Extensions,"** Proceedings of the IEEE Special Issue on "Program Generation, Optimization, and Adaptation," Vol. 93, No. 2, 2005, pages 409-425

Rewriting for SMP Parallelization

$$\underbrace{\mathbf{DFT}_{mn}}_{\mathsf{smp}(p,\mu)} \rightarrow \underbrace{\left((\mathbf{DFT}_m \otimes \mathrm{I}_n)\mathsf{T}^{mn}_n(\mathrm{I}_m \otimes \mathbf{DFT}_n)\,\mathsf{L}^{mn}_m\right)}_{\mathsf{smp}(p,\mu)}$$

$$\ldots$$

$$\rightarrow \underbrace{\left(\mathbf{DFT}_m \otimes \mathrm{I}_n\right)}_{\mathsf{smp}(p,\mu)} \underbrace{\mathsf{T}^{mn}_n}_{\mathsf{smp}(p,\mu)} \underbrace{\left(\mathrm{I}_m \otimes \mathbf{DFT}_n\right)}_{\mathsf{smp}(p,\mu)} \underbrace{\mathsf{L}^{nm}_m}_{\mathsf{smp}(p,\mu)}$$

$$\ldots$$

$$\rightarrow \left((\mathsf{L}^{mp}_m \otimes \mathrm{I}_{n/p\mu})\bar{\otimes}\,\mathrm{I}_\mu\right)\left(\mathrm{I}_p \otimes_{\|} (\mathbf{DFT}_m \otimes \mathrm{I}_{n/p})\right)\left((\mathsf{L}^{mp}_p \otimes \mathrm{I}_{n/p\mu})\bar{\otimes}\,\mathrm{I}_\mu\right)$$
$$\left(\bigoplus_{i=0}^{p-1}{}_{\|}\, \mathsf{T}^{mn,i}_n\right)\left(\mathrm{I}_p \otimes_{\|} (\mathrm{I}_{m/p} \otimes \mathbf{DFT}_n)\right)\left(\mathrm{I}_p \otimes_{\|} \mathsf{L}^{mn/p}_{m/p}\right)\left((\mathsf{L}^{pn}_p \otimes \mathrm{I}_{m/p\mu})\bar{\otimes}\,\mathrm{I}_\mu\right)$$

Two types of base cases: load-balanced, no false sharing

load-balanced		no false sharing
$\mathrm{I}_p \otimes_{\|} A_{\mu n}$	$\bigoplus_{i=0}^{p-1}{}_{\|}\, \mathsf{T}^{mn,i}_n$	$P_n \bar{\otimes}\, \mathrm{I}_\mu$

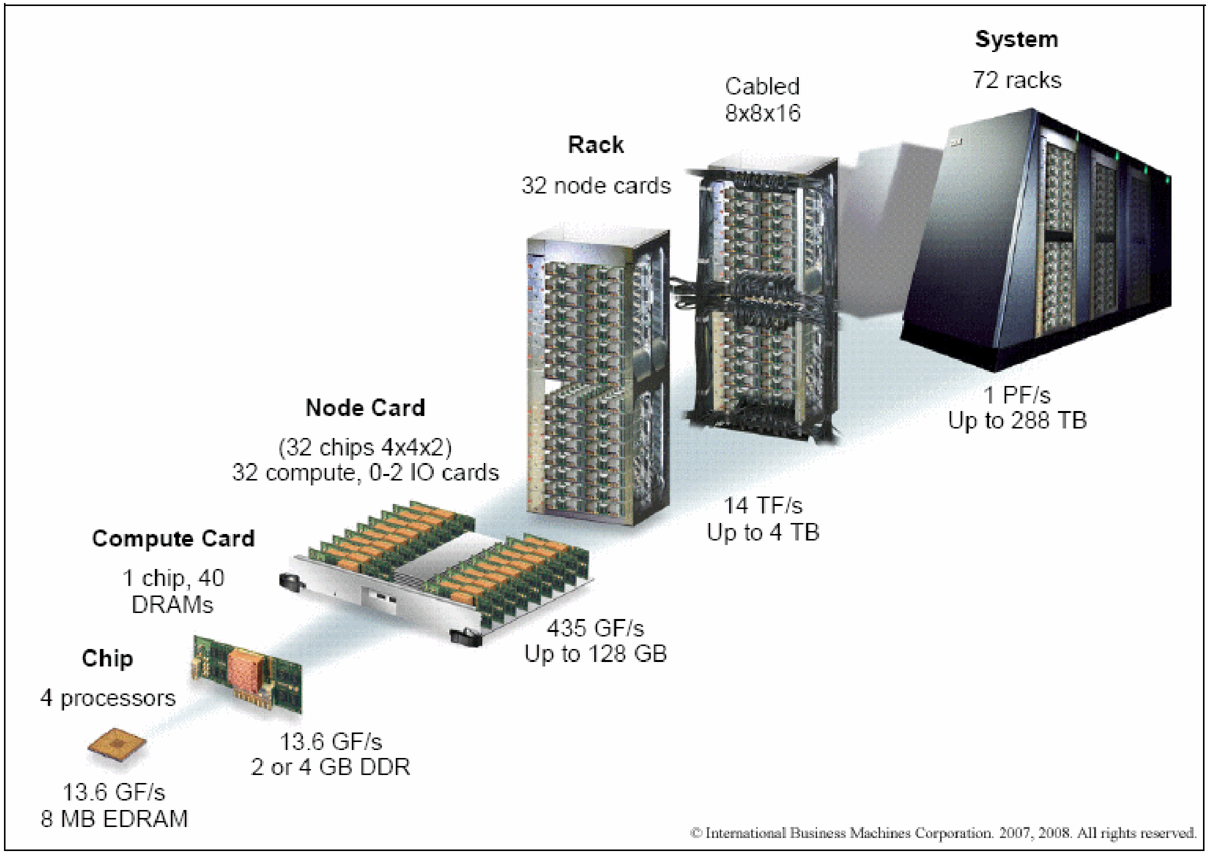
F. Franchetti, Y. Voronenko, M. Püschel: **"FFT Program Generation for Shared Memory: SMP and Multicore,"** In Proceedings Supercomputing (SC), 2006.

Outline

- Spiral: Library Generation
- MPI-Friendly Global FFT Algorithm
- **Experimental Results**
- Summary

BlueGene/P Intrepid at ANL

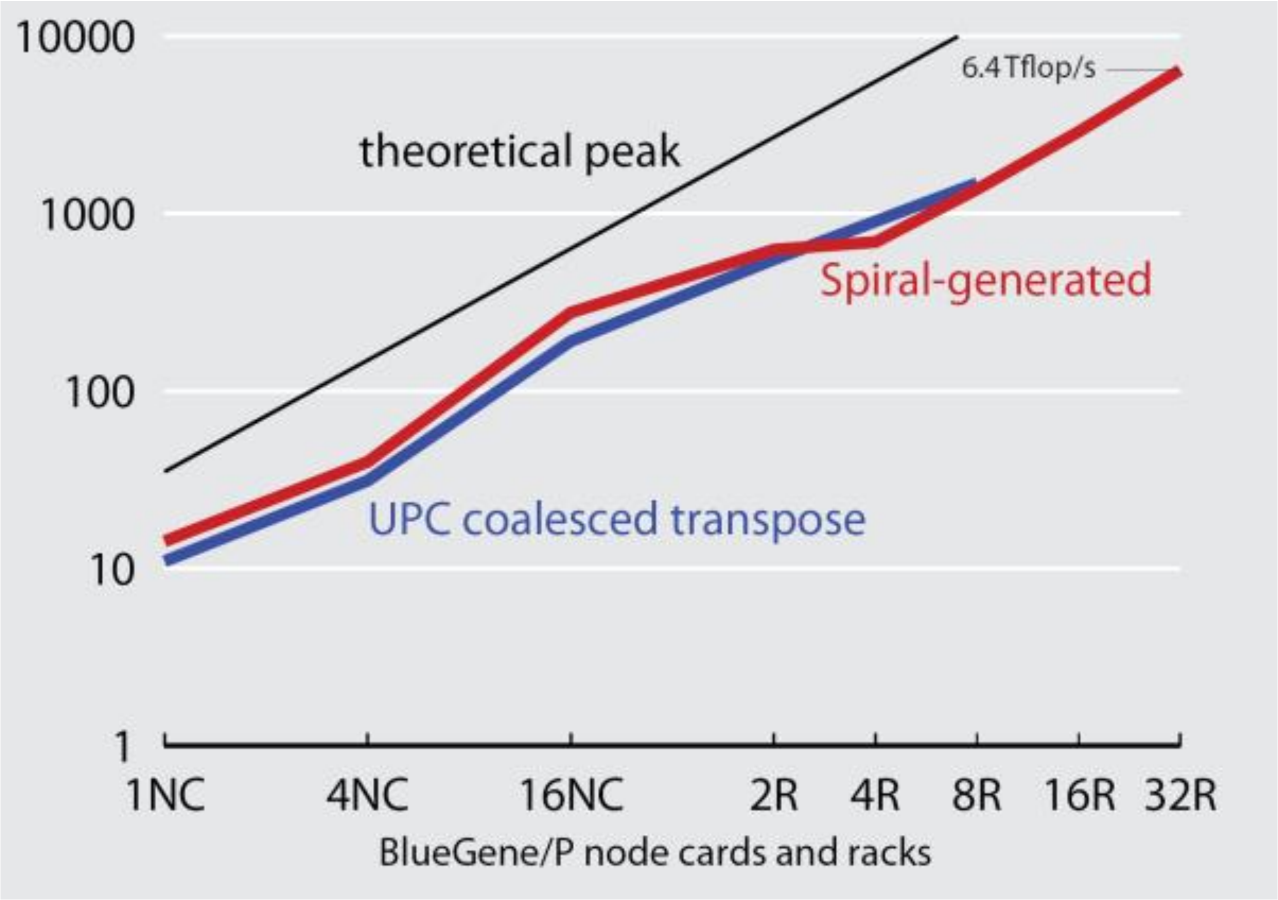
- 40 racks of Blue Gene/P
- 1024 nodes per rack
- one 850 MHz quad-core processor and 2GB RAM per node
- Double FPU SIMD
- 3D Torus network



HPC Challenge Global FFT on BlueGene/P

1D Global FFT

performance [Gflop/s]

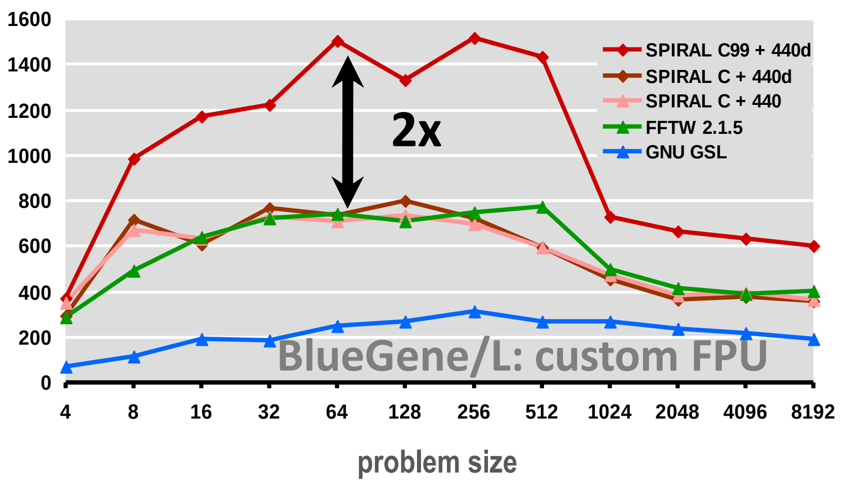


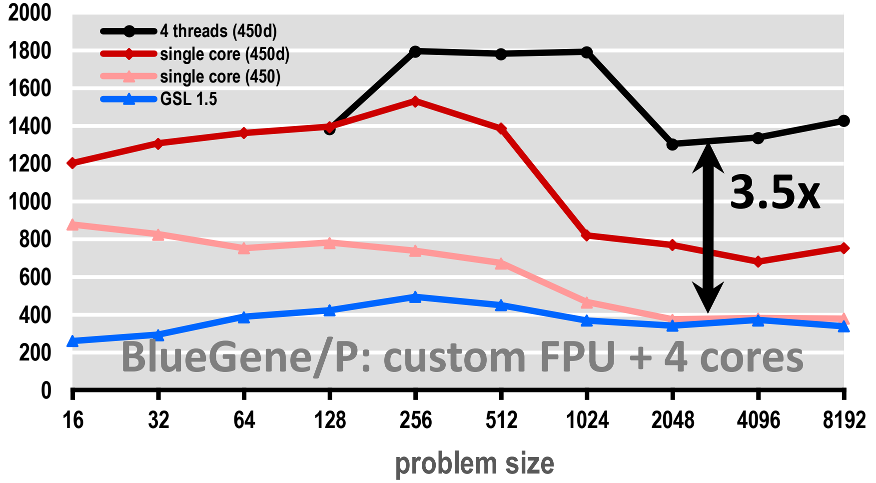
6.4 Tflop/s

FFTE baseline: 5 Tflop/s

G. Almási, B. Dalton, L. L. Hu, F. Franchetti, Y. Liu, A. Sidelnik, T. Spelce, I. G. Tānase, E. Tiotto, Y. Voronenko, X. Xue: **2010 IBM HPC Challenge Class II Submission.** **Winner of the 2010 HPC Challenge Class II Award (Most Productive System).**

Double FPU and Multicore Performance





Single BlueGene/L CPU at 700 MHz
IBM T. J. Watson Research Center

SIMD vectorization

Single BlueGene/P node (4 CPUs) at 850 MHz
Argonne National Laboratory

SIMD vectorization + multi-threading

F. Gygi, E. W. Draeger, M. Schulz, B. R. de Supinski, J. A. Gunnels, V. Austel, J. C. Sexton, F. Franchetti, S. Kral, C. W. Ueberhuber, J. Lorenz: **Large-Scale Electronic Structure Calculations of High-Z Metals on the BlueGene/L Platform.** In Proceedings of Supercomputing, 2006. **Winner of the 2006 Gordon Bell Prize (Peak Performance Award).**

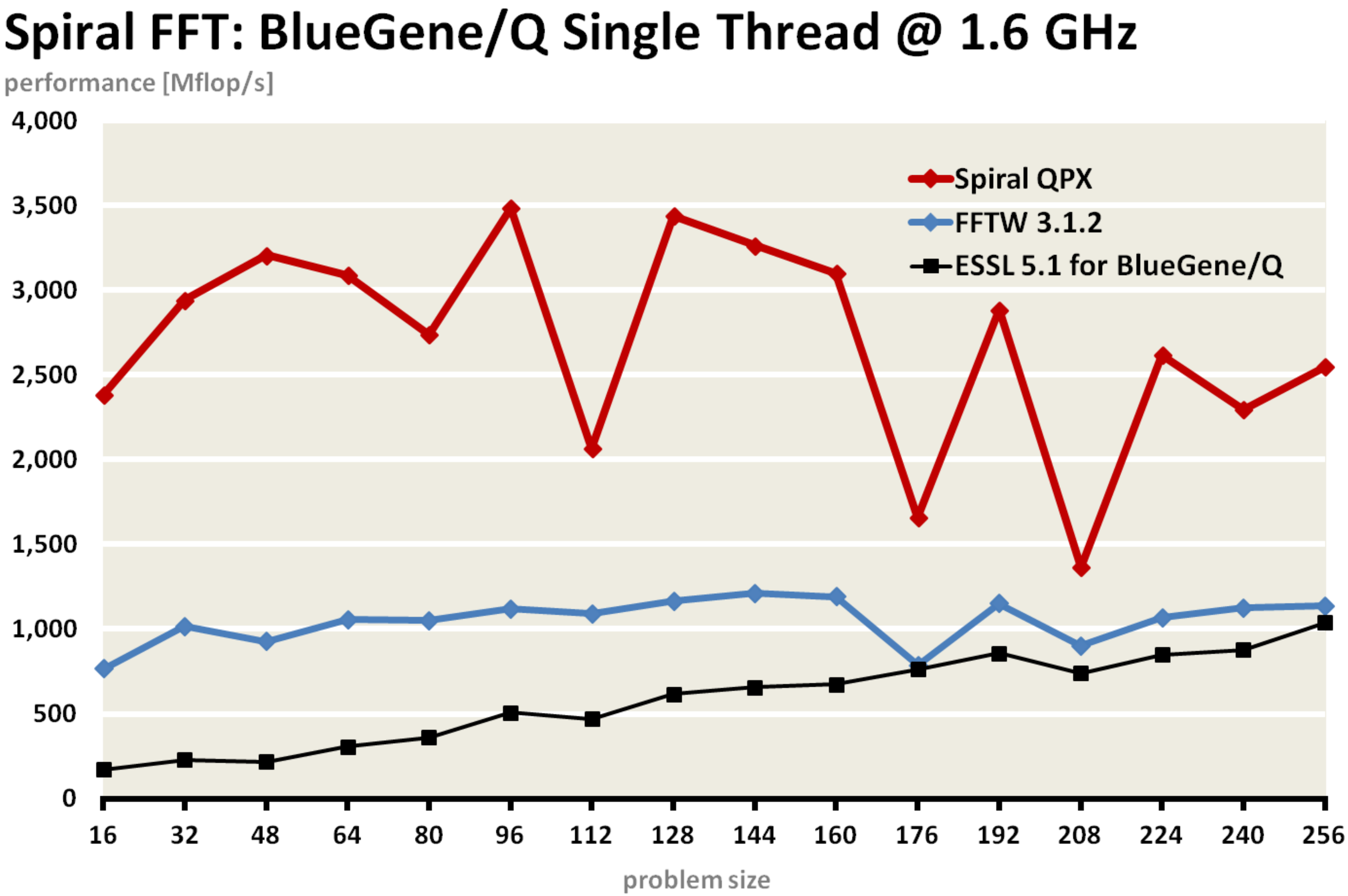
J. Lorenz, S. Kral, F. Franchetti, C. W. Ueberhuber: **Vectorization Techniques for the Blue Gene/L double FPU.** IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005.

Towards BlueGene/Q: QPX Code Generation

```
void dft16(double  *Y, double  *X) {
    const vector4double C1 = (vector4double)(1.0, 0.70710678118654757, 0.0, (-0.70710678118654757));
    const vector4double C2 = (vector4double)(0.0, 0.70710678118654757, 1.0, 0.70710678118654757);
    const vector4double C3 = (vector4double)(1.0, 0.92387953251128674, 0.70710678118654757, 0.38268343236508978);
    const vector4double C4 = (vector4double)(0.0, 0.38268343236508978, 0.70710678118654757, 0.92387953251128674);
    const vector4double C5 = (vector4double)(1.0, 0.38268343236508978, (-0.70710678118654757), (-0.92387953251128674));
    const vector4double C6 = (vector4double)(0.0, 0.92387953251128674, 0.70710678118654757, (-0.38268343236508978));
    vector4double a90, a91, a92, a93, a94, a95, s139, s140, s141, s142, s143, s144, s145, s146, s147, s148,...,;
    vector4double  *a89, *a96;
    a89 = ((vector4double  *) X);
    s139 = a89[0];
    s140 = a89[1];
    a90 = vec_gpci(0xa60);
    s141 = vec_perm(s139, s140, a90);
    a91 = vec_gpci(0xef2);
    s142 = vec_perm(s139, s140, a91);
    s143 = a89[4];
    s144 = a89[5];
    s145 = vec_perm(s143, s144, a90);
    ...
    s170 = vec_perm(s158, s162, a95);
    s171 = vec_sub(vec_mul(C1, s165), vec_mul(
    s172 = vec_add(vec_mul(C2, s165), vec_mul(
    t145 = vec_add(s163, s171);
    t146 = vec_add(s167, s172);
    t147 = vec_sub(s163, s171);
    t148 = vec_sub(s167, s172);
    s173 = vec_sub(vec_mul(C3, s164), vec_mul(
    s174 = vec_add(vec_mul(C4, s164), vec_mul(
    s175 = vec_sub(vec_mul(C5, s166), vec_mul(
    s176 = vec_add(vec_mul(C6, s166), vec_mul(
    t149 = vec_add(s173, s175);
    ...
    a96[3] = s182;
    s183 = vec_perm(t159, t160, a92);
    a96[6] = s183;
    s184 = vec_perm(t159, t160, a93);
    a96[7] = s184;
}
```

```
78| 00014C qvfmul   118D0132  1   QVFMUL    qr12=qr13,qr4,fcr
79| 000150 qvfmul   11AD0172  1   QVFMUL    qr13=qr13,qr5,fcr
84| 000154 qvfmul   11CF01B2  1   QVFMUL    qr14=qr15,qr6,fcr
85| 000158 qvfmul   11EF01F2  1   QVFMUL    qr15=qr15,qr7,fcr
86| 00015C qvfmul   12110232  1   QVFMUL    qr16=qr17,qr8,fcr
87| 000160 qvfmul   12310272  1   QVFMUL    qr17=qr17,qr9,fcr
60| 000164 qvfperm  1253B00C  1   QVFPERM   qr18=qr19,qr22,qr0
62| 000168 qvfperm  1273B04C  1   QVFPERM   qr19=qr19,qr22,qr1
65| 00016C qvfperm  12F4A80C  1   QVFPERM   qr23=qr20,qr21,qr0
66| 000170 qvfperm  1294A84C  1   QVFPERM   qr20=qr20,qr21,qr1
72| 000174 qvfperm  12B2B8CC  1   QVFPERM   qr21=qr18,qr23,qr3
73| 000178 qvfperm  12D3A08C  1   QVFPERM   qr22=qr19,qr20,qr2
74| 00017C qvfperm  1073A0CC  1   QVFPERM   qr3=qr19,qr20,qr3
79| 000180 qvfnmadd 10B62B3E  1   QVFNMADD  qr5=qr5,qr22,qr12,fcr
80| 000184 qvfmadd  1096237A  1   QVFMADD   qr4=qr4,qr22,qr13,fcr
85| 000188 qvfnmadd 10F53BBE  1   QVFNMADD  qr7=qr7,qr21,qr14,fcr
86| 00018C qvfmadd  10D533FA  1   QVFMADD   qr6=qr6,qr21,qr15,fcr
87| 000190 qvfnmadd 11234C3E  1   QVFNMADD  qr9=qr9,qr3,qr16,fcr
88| 000194 qvfmadd  1063447A  1   QVFMADD   qr3=qr8,qr3,qr17,fcr
70| 000198 qvfperm  1112B88C  1   QVFPERM   qr8=qr18,qr23,qr2
75| 00019C qvfperm  104A588C  1   QVFPERM   qr2=qr10,qr11,qr2
81| 0001A0 qvfadd   1148282A  1   QVFADD    qr10=qr8,qr5,fcr
82| 0001A4 qvfadd   1162202A  1   QVFADD    qr11=qr2,qr4,fcr
89| 0001A8 qvfadd   1187482A  1   QVFADD    qr12=qr7,qr9,fcr
90| 0001AC qvfadd   11A6182A  1   QVFADD    qr13=qr6,qr3,fcr
83| 0001B0 qvfsub   10A82828  1   QVFSUB    qr5=qr8,qr5,fcr
```

First BlueGene/Q Performance Results



Outline

- Spiral: Library Generation
- MPI-Friendly Global FFT Algorithm
- Experimental Results
- **Summary**

Summary

- **Node FFT libraries are tuned for linear, contiguous data**
  But 2D abstraction required for the transpose in the FFT
- **MPI all-to-all (transpose) is suboptimal on linearized 2D data**
  2D tiles are not contiguous in memory
- **Solution: Special FFT functions that work on 2D tiles**
  Same FFT performance as linear memory, full MPI performance
- **Spiral auto-generates specialized node libraries**
  As fast as ESSL and FFTW, but works on 2D tiled memory
- **Performance results on ANL's BlueGene/P (Intrepid)**
  Performance improvement from 5 Tflop/s to 6.4 Tflop/s

More Information:

www.spiral.net

www.spiralgen.com