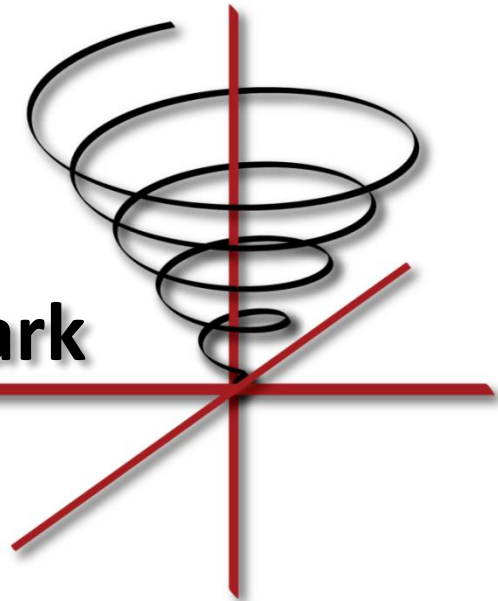


Automatic Generation of the HPC Challenge's Global FFT Benchmark for BlueGene/P



Franz Franchetti¹, Yevgen Voronenko², Gheorghe Almasi³

¹Carnegie Mellon University and SpiralGen, Inc.

²AccuRay, Inc., ³IBM 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

HPC CHALLENGE

PROJECT GOALS

- Provide performance bounds in locality space using real world computational kernels
- Allow scaling of input data size and time to run according to the system capability
- Verify the results using standard error analysis
- Allow vendors and users to provide optimized code for superior performance
- Make the benchmark information continuously available to the public in order to disseminate performance tuning knowledge and record technological progress over time
- Ensure reproducibility of the results by detailed reporting of all aspects of benchmark runs

FEATURE HIGHLIGHTS OF HPCC 1.4.1

- New variants of RandomAccess that use Linear Congruential Random Number Generator
- New order of tests makes HPL run last so users may abort execution early if necessary
- Initialization of the main array of RandomAccess is no longer timed
- Global reduction is used for error calculation in MPI FFT to achieve more accurate error estimate
- Updated documentation
- New initialization and finalization routines allow proper setup of external software/hardware components without changing the rest of the HPCC testing harness.
- Fixed memory leaks in G-RandomAccess and FFT driver routines.
- Better interface to 64-bit versions of FFTW such as Intel's MKL

LOCALITY SPACE OF MEMORY ACCESS IN APPLICATIONS

KIVIAT CHART WITH RESULTS FOR THREE DIFFERENT CLUSTERS

Delco Opteron/CaNial Linux Cluster AMD Opteron
64 proc - 2.2 GHz
1 thread/MPI process (64)
CaNial
11-04-2004

Cray X01 AMD Opteron
64 proc - 2.2 GHz
1 thread/MPI process (64)
RapidArray Interconnect System
11-22-2004

Sun Fire V20i Cluster AMD Opteron
64 proc - 2.2 GHz
1 thread/MPI process (64)
Gigabit Ethernet, Cisco 6509 switch
03-06-2005

HPCC RESULTS' PAGE

SUMMARY OF HPCC AWARDS

CLASS 1: Best Performance

- Best in G-HPL, EP-STREAM-Triad per system, G-RandomAccess, G-FFT
- There will be 4 winners (one in each category)

CLASS 2: Most Productivity

- One or more winners
- Judged by a panel at SC11 BOF
- Stresses elegance and performance
- Implementations in various (existing and new) languages are encouraged
- Submissions may include up to two kernels not present in HPCC
- Submission consists of: code, its description, performance numbers, and a presentation at the BOF

FIND OUT MORE AT <http://icl.eecs.utk.edu/hpcc/>

SPONSORED BY

INNOVATIVE
COMPUTING LABORATORY
THE UNIVERSITY OF TENNESSEE

CITR
CENTER FOR INFORMATION
TECHNOLOGY RESEARCH

<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

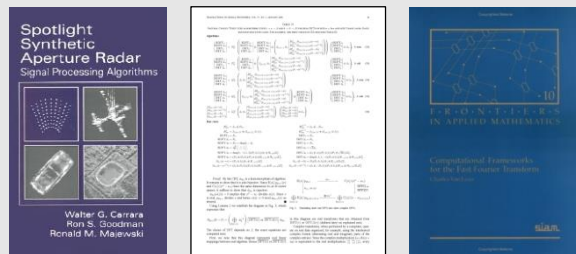
- **Spiral: Library Generation**
- MPI-Friendly Global FFT Algorithm
- Experimental Results
- Summary

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

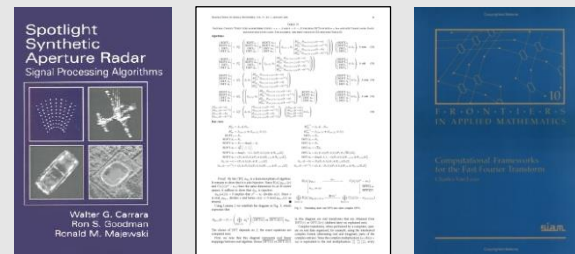
Traditionally



High performance library
optimized for given platform

*Comparable
performance*

Spiral Approach



High performance library
optimized for given platform

Spiral's Formal Framework

- **Transform = Matrix-vector multiplication**

Example: Discrete Fourier transform (DFT)

$$\begin{array}{c}
 x \mapsto y = T \cdot x \\
 \uparrow \qquad \qquad \qquad \uparrow \\
 \text{input vector (signal)} \quad \text{output vector (signal)} \quad \text{transform = matrix}
 \end{array}$$

- **Fast algorithm = sparse matrix factorization = SPL formula**

Example: Cooley-Tukey FFT algorithm

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

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

Transforms and Breakdown Rules

$$\text{DFT}_n \rightarrow (\text{DFT}_k \otimes \text{I}_m) \text{T}_m^n (\text{I}_k \otimes \text{DFT}_m) \text{L}_k^n, \quad n = km$$

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

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

$$\text{DCT-3} \rightarrow (\text{I}_1 \oplus \text{I}_1) \text{L}_n (\text{DCT-2}_{(1/4)} \oplus \text{DCT-2}_{(3/4)})$$

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

DCT-4

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

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

$$\text{DFT}_2 \rightarrow \text{F}_2$$

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

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

Base case rules

Teaches Spiral about FFT algorithms

One Approach for all Types of Parallelism

- **Multithreading** (Multicore)
- **Vector SIMD** (SSE, VMX/AltiVec,...)
- **Message Passing** (Clusters, MPP)
- **Streaming/multibuffering** (Cell)
- **Graphics Processors** (GPUs)
- **Gate-level parallelism** (FPGA)
- **HW/SW partitioning** (CPU + FPGA)

$$I_p \otimes_{\parallel} A_{\mu n}, \quad L_m^{mn} \bar{\otimes} I_{\mu}$$

$$A \hat{\otimes} I_{\nu} \quad \underbrace{L_2^{2\nu}}_{isa}, \quad \underbrace{L_{\nu}^{2\nu}}_{isa}, \quad \underbrace{L_{\nu}^{\nu^2}}_{isa}$$

$$I_p \otimes_{\parallel} A_n, \quad \underbrace{L_p^{p^2} \bar{\otimes} I_{n/p^2}}_{\text{all-to-all}}$$

$$I_n \otimes_2 A_{\mu n}, \quad L_m^{mn} \bar{\otimes} I_{\mu}$$

$$\prod_{i=0}^{n-1} A_i, \quad A_n \hat{\otimes} I_w, \quad P_n \otimes Q_w$$

$$\prod_{i=0}^{n-1} A_i^{ir}, \quad I_s \tilde{\otimes} A, \quad \underbrace{L_n^m}_{\text{bram}}$$

$$\underbrace{A_1}_{\text{fpga}}, \quad \underbrace{A_2}_{\text{fpga}}, \quad \underbrace{A_3}_{\text{fpga}}, \quad \underbrace{A_4}_{\text{fpga}}$$

Translating a Formula into Code

Rewriting Input:

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



Output =

OL Formula: $(\text{DFT}_2 \otimes I_4) T_4^8 \left(I_2 \otimes \left((\text{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \text{DFT}_2) L_2^4 \right) \right) L_2^8$



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



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;
}
```

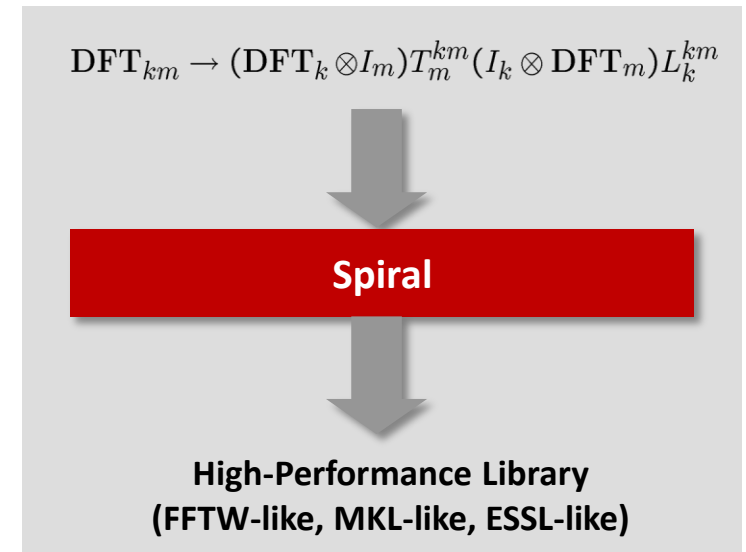
Synthesizing General Size Libraries

Input:

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

Output:

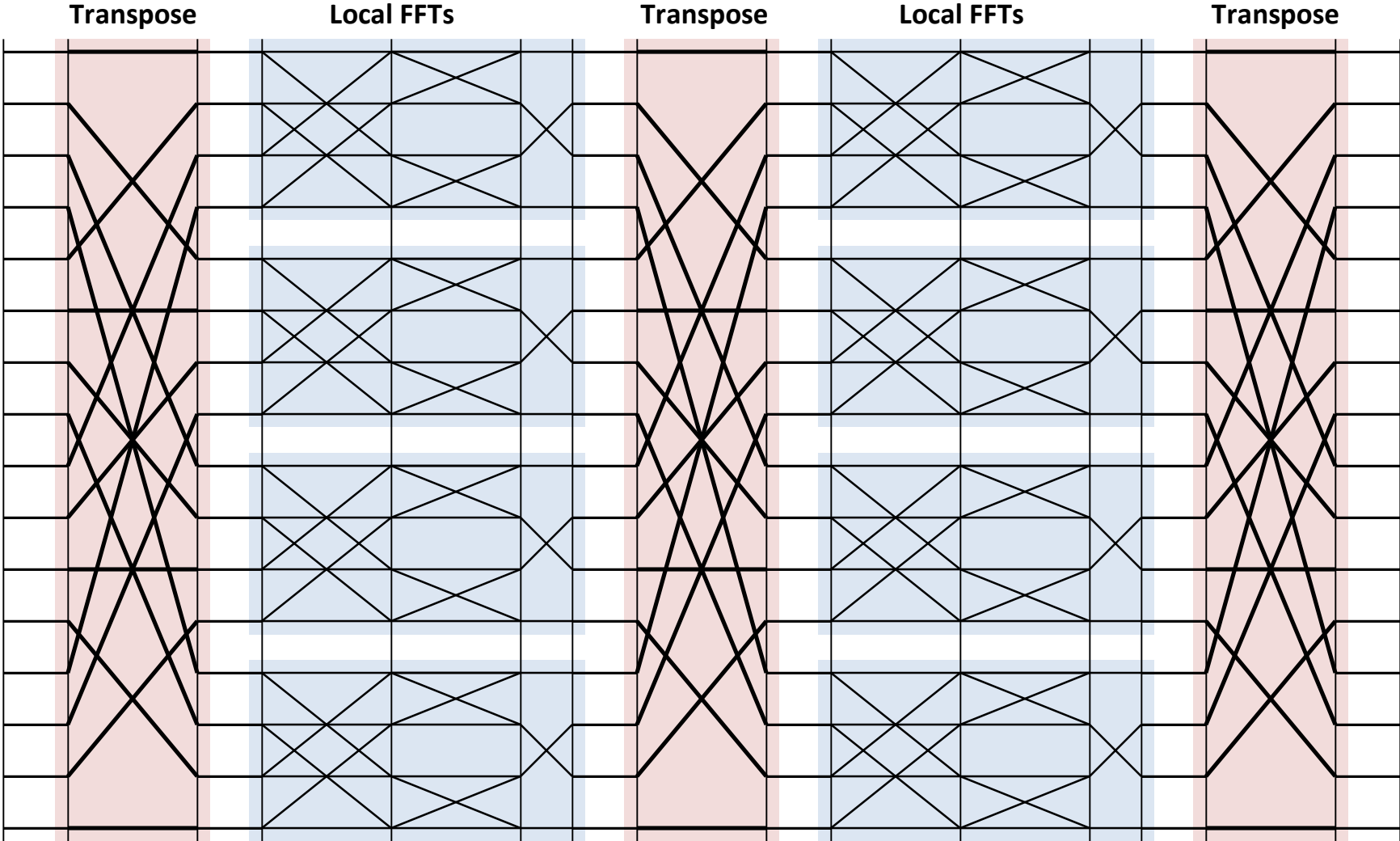
- Optimized library (10,000 lines of C++)
- For general input size
(**not** collection of fixed sizes)
- Vectorized
- Multithreaded
- With runtime adaptation mechanism
- Performance competitive with hand-written code



Outline

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

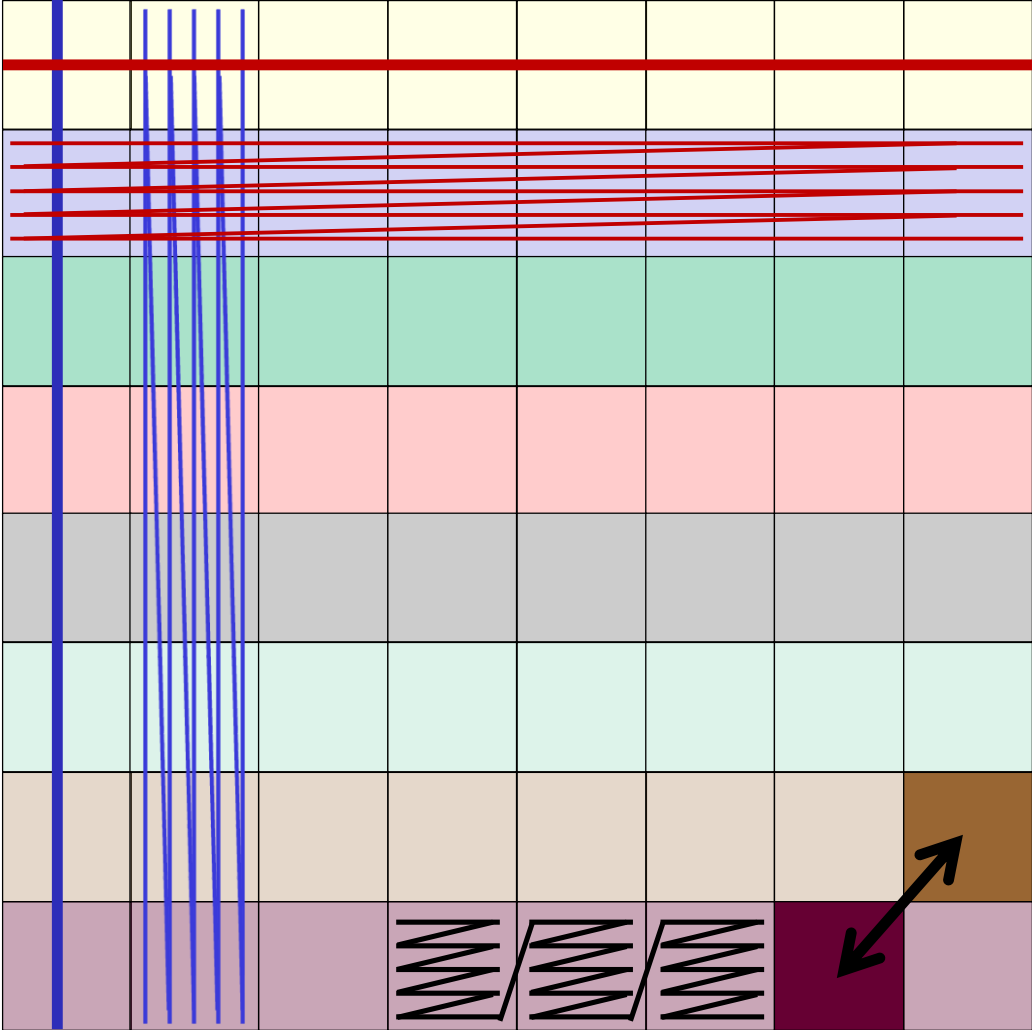
FFT needs Local FFTs and Global Transposes



FFTs along rows and columns of distributed square matrix

Linear Memory vs. Tiled Memory

Column FFTs: Need contiguous columns



Row FFTs
need contiguous rows

Processor i

Processor $i+1$

Processor $i+2$

p node MPI all-to-all
needs contiguous tiles

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

MPI All-to-all “Friendly” Six Step FFT

$$\begin{aligned}
 \text{DFT}_{mn} = & \underbrace{\left(I_p \otimes (L_{m/p}^m \otimes I_{n/p}) \right)}_{\text{local transpose}} \underbrace{\left(L_p^{p^2} \otimes I_{mn/p^2} \right)}_{\text{all-to-all}} \underbrace{\left(I_p \otimes (\text{DFT}_m \otimes I_{n/p}) \right)}_{\text{inplace FFT library call}} \\
 & \underbrace{\left(L_p^{p^2} \otimes I_{mn/p^2} \right)}_{\text{all-to-all}} \underbrace{\left(T_n^{mn} \right)}_{\text{out-of-place scaled FFT library call}} \underbrace{\left(I_p \otimes L_{m/p}^m \otimes I_{n/p} \right)}_{\text{out-of-place scaled FFT library call}} \underbrace{\left(I_p \otimes (L_p^m \otimes I_{n/p}) (I_{m/p} \otimes \text{DFT}_n) L_{m/p}^{mn/p} \right)}_{\text{out-of-place scaled FFT library call}} \\
 & \underbrace{\left(L_p^{p^2} \otimes I_{mn/p^2} \right)}_{\text{all-to-all}} \underbrace{\left(I_p \otimes L_p^n \otimes I_{m/p} \right)}_{\text{local transpose}}
 \end{aligned}$$

Standard batch FFT library (on 1D contiguous memory)

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

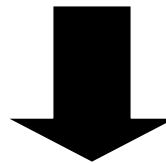
Standard MPI all-to-all on contiguous data

Node-local pre-processing (data scrambling)

SIMD Vectorization for FFT

Standard FFT

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



Automatic formula rewriting

$$\text{DFT}_{mn} \rightarrow \left(\text{I}_{\frac{mn}{\nu}} \otimes \underline{\text{L}_{\nu}^{2\nu}} \right) \left(\underline{\overline{\text{DFT}_m \otimes \text{I}_{\frac{n}{\nu}} \otimes \text{I}_{\nu}}} \right) \overline{\text{T}_n^{mn}}$$

$$\left(\text{I}_{\frac{m}{\nu}} \otimes \left(\underline{\text{L}_{\nu}^{2n}} \otimes \text{I}_{\nu} \right) \left(\text{I}_{\frac{2n}{\nu}} \otimes \underline{\text{L}_{\nu}^{\nu^2}} \right) \left(\underline{\text{I}_{\frac{n}{\nu}} \otimes \text{L}_2^{2\nu}} \otimes \text{I}_{\nu} \right) \left(\underline{\overline{\text{DFT}_n \otimes \text{I}_{\nu}}} \right) \right) \left(\text{L}_{\frac{m}{\nu}}^{\frac{mn}{\nu}} \otimes \underline{\text{L}_2^{2\nu}} \right)$$

Vectorized arithmetic

Data reorganization

(requires architecture specific
 vectorization)

Short Vector FFT for v-way vector instructions

Only 3 **permutations** require architecture-specific vectorization: $\text{L}_2^{2\nu}$, $\text{L}_{\nu}^{2\nu}$, $\text{L}_{\nu}^{\nu^2}$

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

$$\begin{aligned}
 \underbrace{\text{DFT}_{mn}}_{\text{smp}(p,\mu)} &\rightarrow \underbrace{\left((\text{DFT}_m \otimes \text{I}_n) \text{T}_n^{mn} (\text{I}_m \otimes \text{DFT}_n) \text{L}_m^{mn} \right)}_{\text{smp}(p,\mu)} \\
 \dots & \\
 &\rightarrow \underbrace{\left(\text{DFT}_m \otimes \text{I}_n \right)}_{\text{smp}(p,\mu)} \underbrace{\text{T}_n^{mn}}_{\text{smp}(p,\mu)} \underbrace{\left(\text{I}_m \otimes \text{DFT}_n \right)}_{\text{smp}(p,\mu)} \underbrace{\text{L}_m^{nm}}_{\text{smp}(p,\mu)} \\
 \dots & \\
 &\rightarrow \underbrace{\left((\text{L}_m^{mp} \otimes \text{I}_{n/p\mu}) \bar{\otimes} \text{I}_\mu \right)}_{\text{red}} \underbrace{\left(\text{I}_p \otimes_{\parallel} (\text{DFT}_m \otimes \text{I}_{n/p}) \right)}_{\text{blue}} \underbrace{\left((\text{L}_p^{mp} \otimes \text{I}_{n/p\mu}) \bar{\otimes} \text{I}_\mu \right)}_{\text{red}} \\
 &\quad \underbrace{\left(\bigoplus_{i=0}^{p-1} \text{T}_n^{mn,i} \right)}_{\text{blue}} \underbrace{\left(\text{I}_p \otimes_{\parallel} (\text{I}_{m/p} \otimes \text{DFT}_n) \right)}_{\text{blue}} \underbrace{\left(\text{I}_p \otimes_{\parallel} \text{L}_{m/p}^{mn/p} \right)}_{\text{blue}} \underbrace{\left((\text{L}_p^{pn} \otimes \text{I}_{m/p\mu}) \bar{\otimes} \text{I}_\mu \right)}_{\text{red}}
 \end{aligned}$$

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

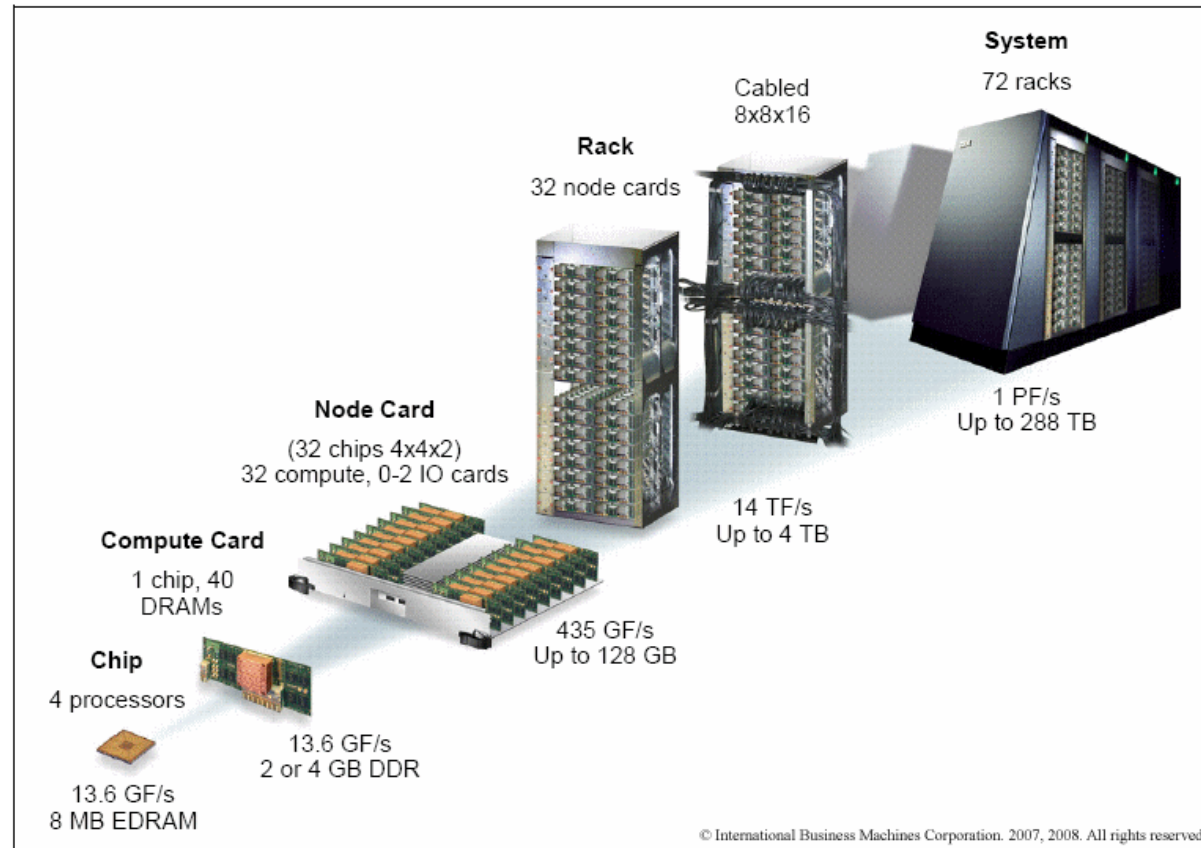
$\text{I}_p \otimes_{\parallel} A_{\mu n}$	$\bigoplus_{i=0}^{p-1} \text{T}_n^{mn,i}$	$P_n \bar{\otimes} \text{I}_\mu$
--	---	----------------------------------

Outline

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

BlueGene/P Intrepid at ANL

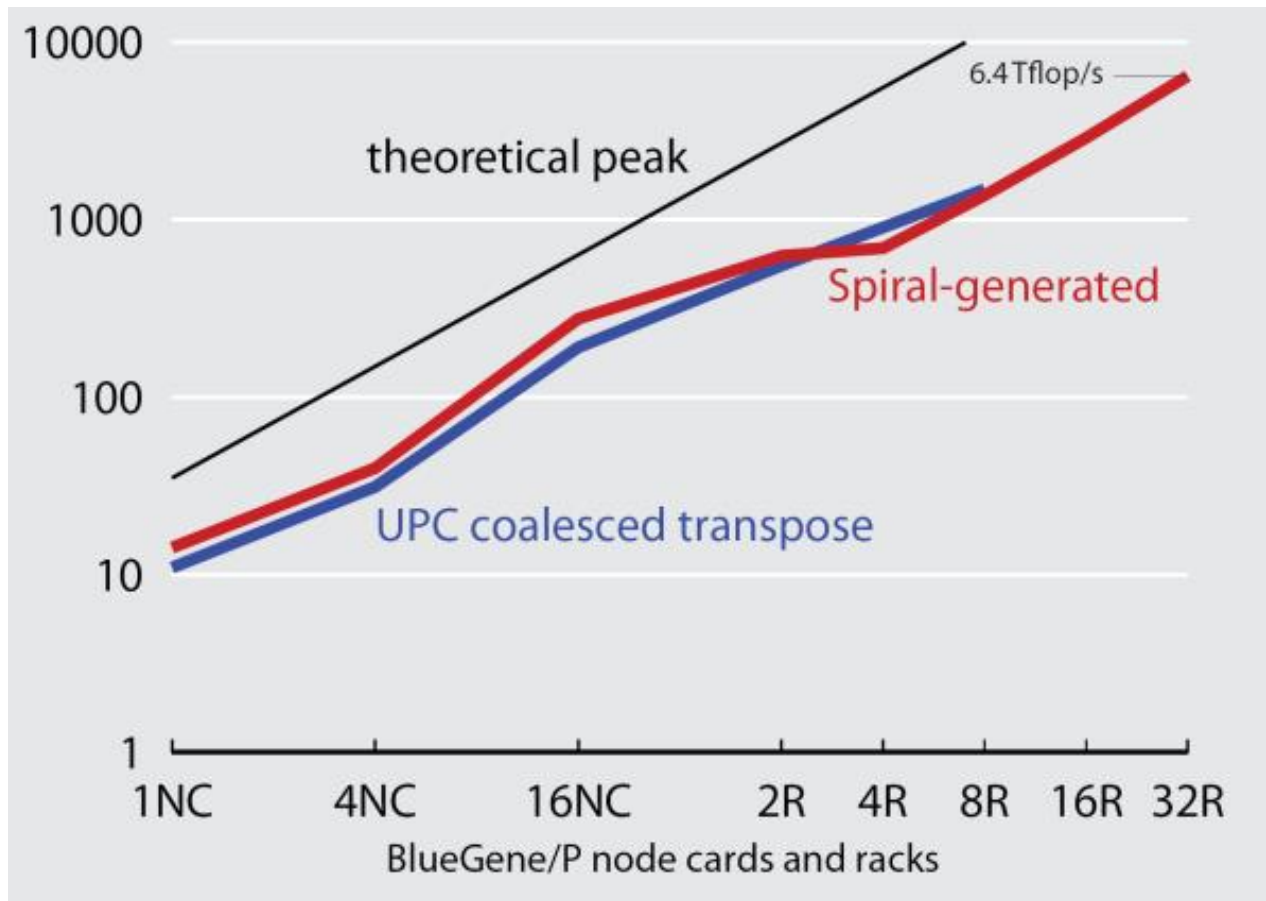
- 40 racks of Blue Gene/P
- 1024 nodes per rack
- one 850 MHz quad-core processor and 2GB RAM per node
- Double FPU SIMD
- 3D Torus network



HPC Challenge Global FFT on BlueGene/P

1D Global FFT

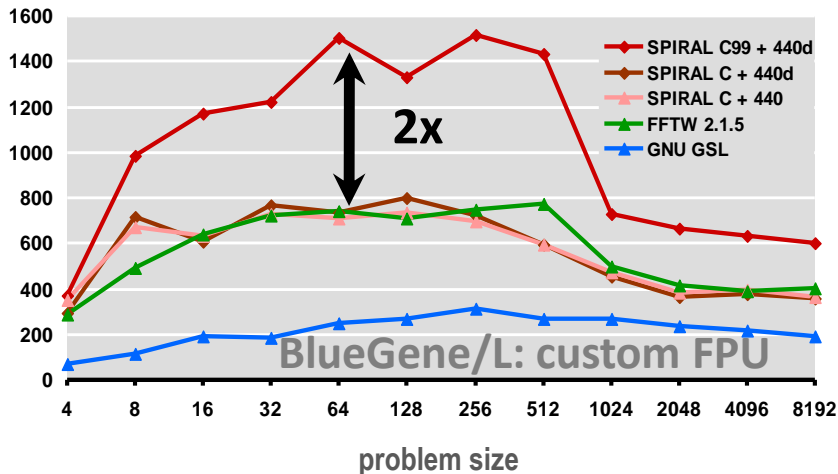
performance [Gflop/s]



6.4 Tflop/s
FFTE baseline: 5 Tflop/s

Double FPU and Multicore Performance

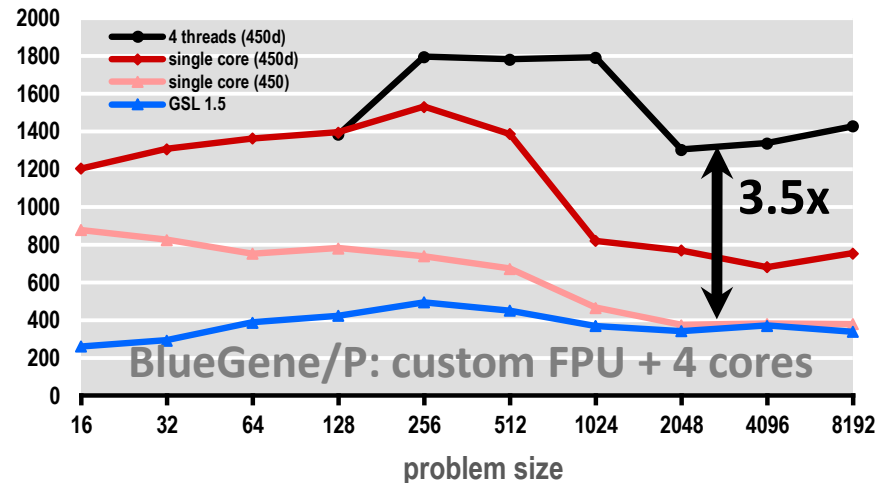
DFT, double precision, XL C compiler
performance [Mflop/s]



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

SIMD vectorization

DFT, double precision, XL C compiler
performance [Mflop/s]



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(C2, s165));
    s172 = vec_add(vec_mul(C2, s165), vec_mul(C3, s165));
    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(C4, s164));
    s174 = vec_add(vec_mul(C4, s164), vec_mul(C5, s164));
    s175 = vec_sub(vec_mul(C5, s166), vec_mul(C6, s166));
    s176 = vec_add(vec_mul(C6, s166), vec_mul(C1, s166));
    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;
}

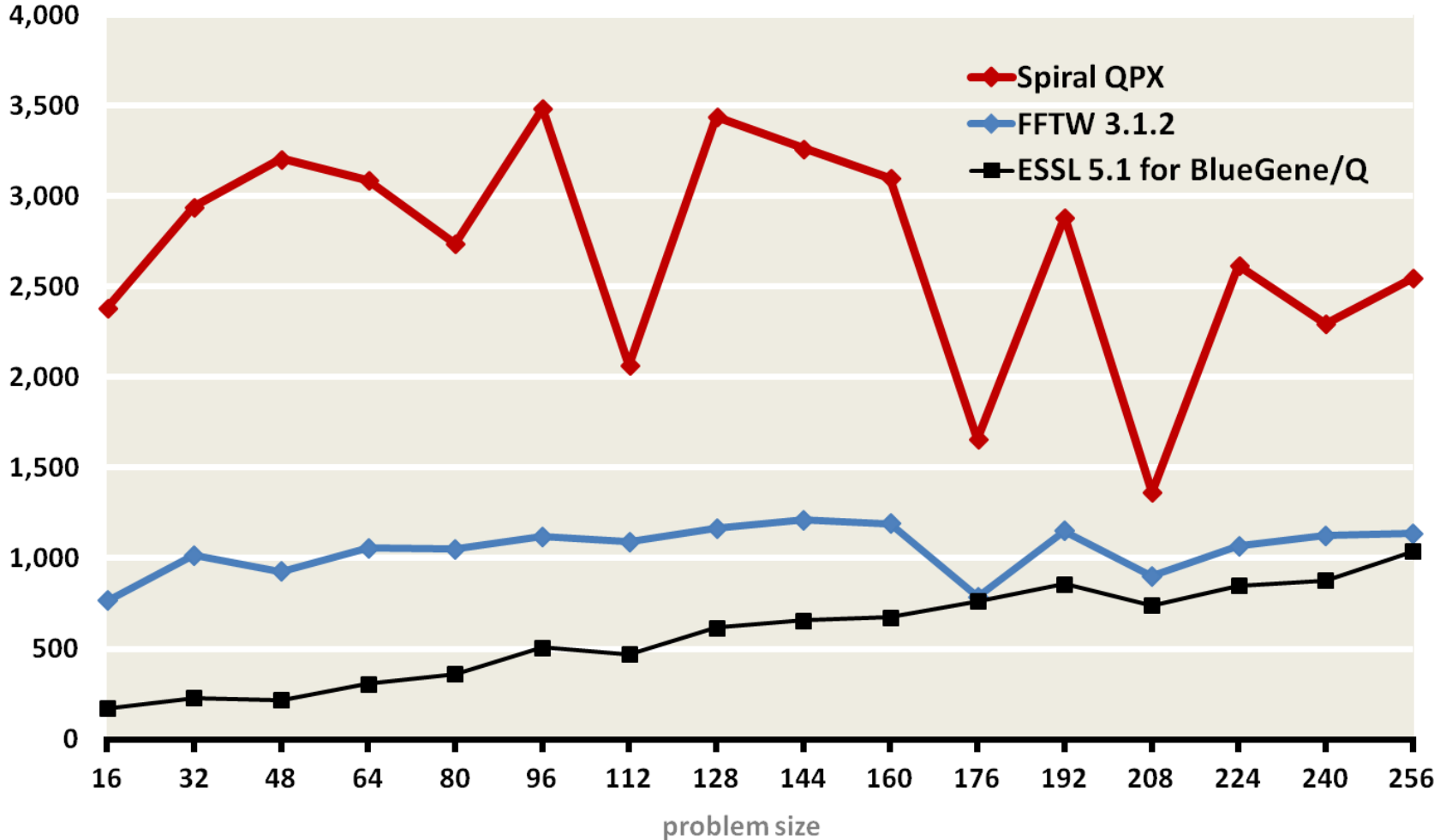
```

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	QVFFPERM	qr18=qr19,qr22,qr0
62	000168	qvfperm	1273B04C	1	QVFFPERM	qr19=qr19,qr22,qr1
65	00016C	qvfperm	12F4A80C	1	QVFFPERM	qr23=qr20,qr21,qr0
66	000170	qvfperm	1294A84C	1	QVFFPERM	qr20=qr20,qr21,qr1
72	000174	qvfperm	12B2B8CC	1	QVFFPERM	qr21=qr18,qr23,qr3
73	000178	qvfperm	12D3A08C	1	QVFFPERM	qr22=qr19,qr20,qr2
74	00017C	qvfperm	1073A0CC	1	QVFFPERM	qr3=qr19,qr20,qr3
79	000180	qvfnmadd	10B62B3E	1	QVFNMADD	qr5=qr5,qr22,qr12, fcr
80	000184	qvfnmadd	1096237A	1	QVFNMADD	qr4=qr4,qr22,qr13, fcr
85	000188	qvfnmadd	10F53BBE	1	QVFNMADD	qr7=qr7,qr21,qr14, fcr
86	00018C	qvfnmadd	10D533FA	1	QVFNMADD	qr6=qr6,qr21,qr15, fcr
87	000190	qvfnmadd	11234C3E	1	QVFNMADD	qr9=qr9,qr3,qr16, fcr
88	000194	qvfnmadd	1063447A	1	QVFNMADD	qr3=qr8,qr3,qr17, fcr
70	000198	qvfperm	1112B88C	1	QVFFPERM	qr8=qr18,qr23,qr2
75	00019C	qvfperm	104A588C	1	QVFFPERM	qr2=qr10,qr11,qr2
81	0001A0	qvfvadd	1148282A	1	QVFFADD	qr10=qr8,qr5, fcr
82	0001A4	qvfvadd	1162202A	1	QVFFADD	qr11=qr2,qr4, fcr
89	0001A8	qvfvadd	1187482A	1	QVFFADD	qr12=qr7,qr9, fcr
90	0001AC	qvfvadd	11A6182A	1	QVFFADD	qr13=qr6,qr3, fcr
83	0001B0	qvfvsub	10A82828	1	QVFFSUB	qr5=qr8,qr5, fcr

First BlueGene/Q Performance Results

Spiral FFT: BlueGene/Q Single Thread @ 1.6 GHz

performance [Mflop/s]



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