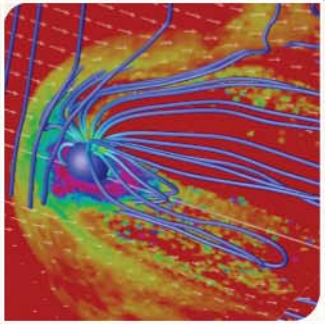
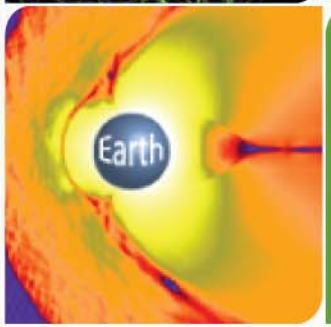
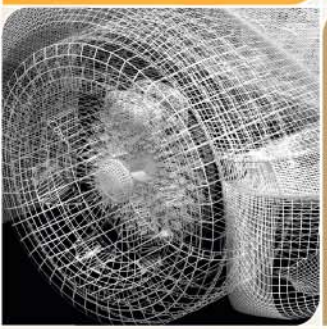


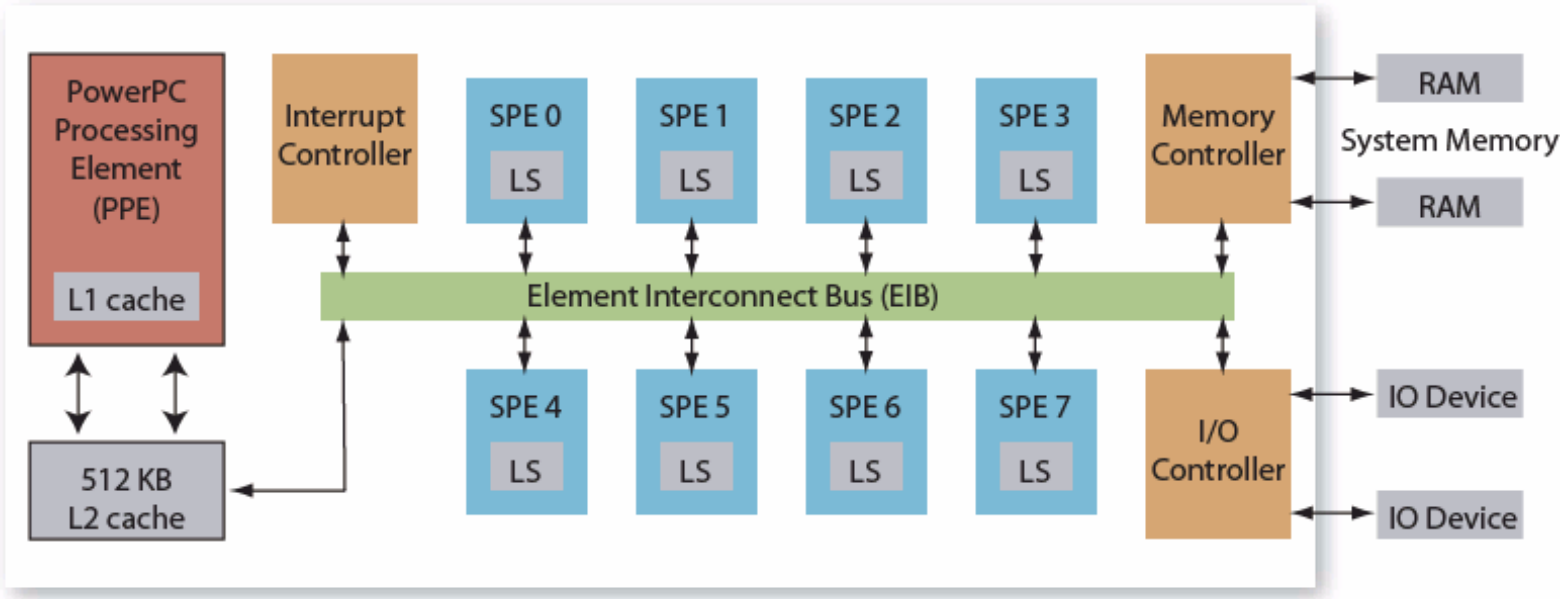


FFTC: Fastest Fourier Transform on the IBM Cell Broadband Engine

David A. Bader, **Virat Agarwal**

Georgia Tech College of Computing
Computational Science and Engineering

Cell System Features



- **Heterogeneous multi-core system architecture**
  - Power Processor Element for control tasks
  - Synergistic Processor Elements for data-intensive processing
- **Synergistic Processor Element (SPE) consists of**
  - Synergistic Processor Unit (SPU)
  - Synergistic Memory Flow Control (MFC)
  - Data movement & synchronization
  - Interface to high-performance Element Interconnect Bus



Fast Fourier Transform

- FFTs are used in:

  - Data Compression, Seismic Imaging, Fluid Dynamics, Image Processing, etc.

- Medium Size FFTs

  - Complex single-precision 1D FFT.
  - Input samples and output results reside in main memory.
  - **Radix 2**, 3 and 5.
  - Optimized for 1K-16K points.

Existing FFT Research on Cell/B.E.

- [Williams *et al.,* 2006], analyzed peak performance.
- [Cico, Cooper and Greene, 2006] estimated 22.1 GFlops/s for an 8K complex 1D FFT that resides in the Local Store of one SPE.
  - 8 independent FFTs in local store of 8 SPEs gives 176.8 GFlops/s.
- [Chow, Fossum and Brokenshire, 2005] achieved 46.8 GFlops/s for 16M complex FFT.
  - Highly specialized for this particular input size.
- FFTW is a highly portable FFT library of various types, precision and input size.

Our FFTC is based on Cooley Tukey

- **Out of Place 1D FFT**

  requires two arrays

  A & B for computation

  at each stage

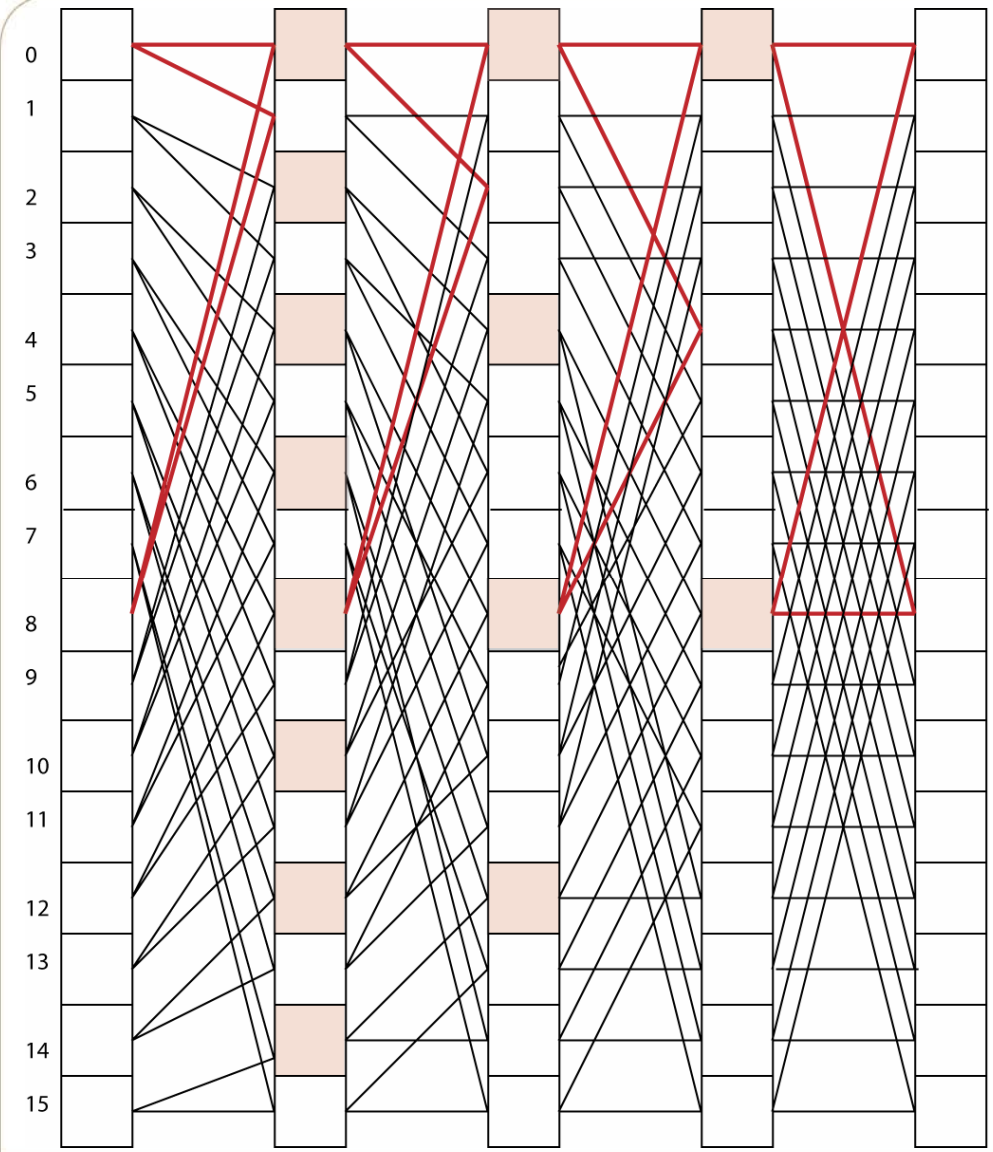
Algorithm 1: Sequential FFT algorithm

Input : array A[0] of size N

```
1  NP <- 1;
2  problemSize <- N;
3  dist <- 1;
4  i1 <- 0;
5  i2 <- 1;
6  while problemSize > 1 do
7      Begin Stage;
8      a <- i1;
9      b <- i2;
10     k = 0, jtwiddle = 0;
11     for j <-0 to N - 1 step 2 * NP do
12         W <- w[jtwiddle];
13         for jfirst <- 0 to NP do
14             b[j + jfirst] <-a[k + jfirst] + a[k + jfirst + N/2];
15             b[j + jfirst + Dist] <-(a[k + jfirst] - a[k + jfirst + N/2]) * W;
16         k <-k + NP;
17         jtwiddle <-jtwiddle + NP;
18     swap(i1, i2);
19     NP <-NP * 2;
20     problemSize <-problemSize/2;
21     dist <-dist * 2;
22     End Stage;
```

Output : array A[i1] of size N

Illustration of the Algorithm



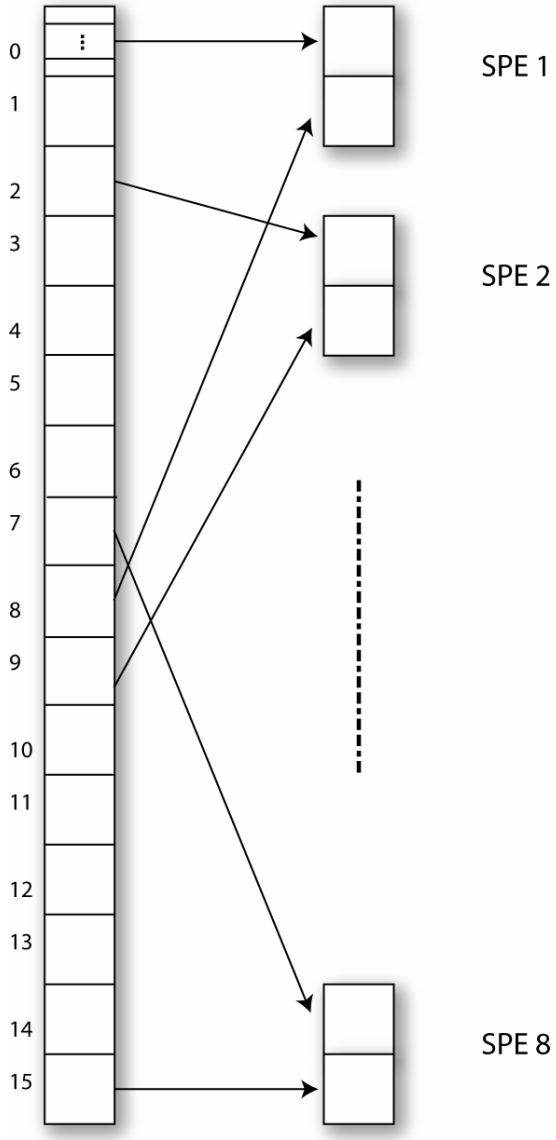
- Butterflies of the ordered DIF (Discrete in Frequency) FFT Algorithm.
- Does not require bit reveral at the end of all stages.



FFTC design on Cell/B.E.

- Synchronize step after every stage leads to significant overhead
- Load balancing to achieve better SPU utilization
- Vectorization difficult for every stage
- Limited local store
  - require space for twiddle factors and input data.
  - loop unrolling and duplication increases size of the code.
- Algorithm is branchy:
  - Doubly nested for loop within the outer while loop
  - Lack of branch predictor compromises performance.

Paralleling FFTC



- Number of chunks := 2*p

  p: Number of SPEs

- Chunk *i* and *i+p* are allocated to SPE *i.*

- Each chunk is fetched using DMA get with multibuffering.

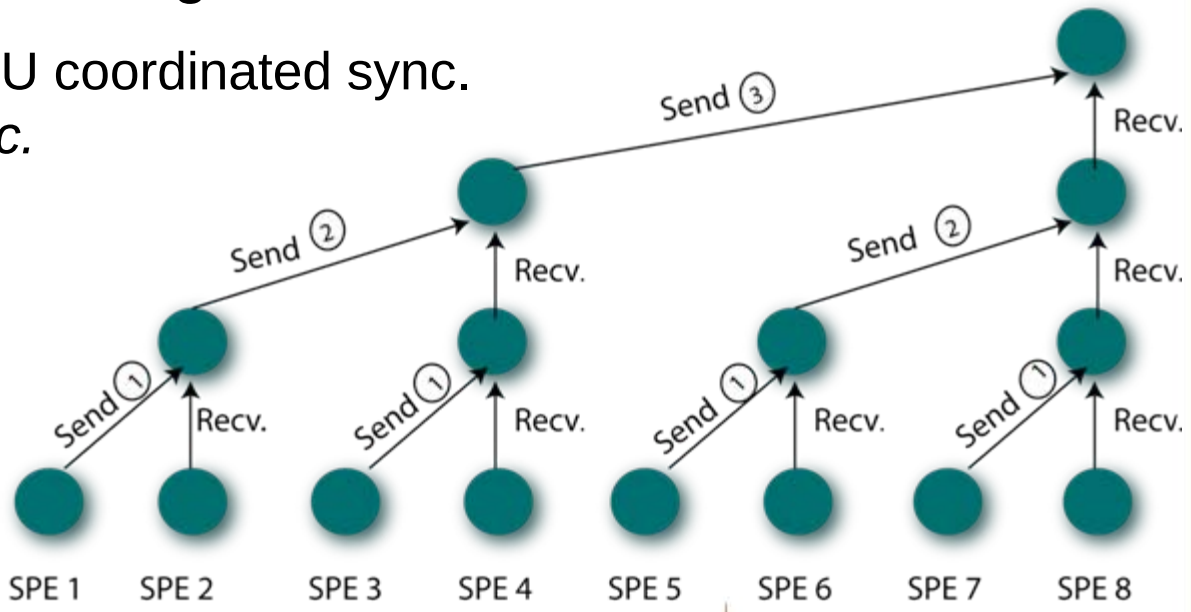
- Achieves load balancing.

Tree Synchronization

- Synchronization after every stage using Inter-SPE DMA communication
  - Achieved in (2*log*n)* stages.

- Each synchronization stage takes *1 microsec*
  - In comparison to PPU coordinated sync. that takes *20 microsec.*

- Minimizes sync. overhead

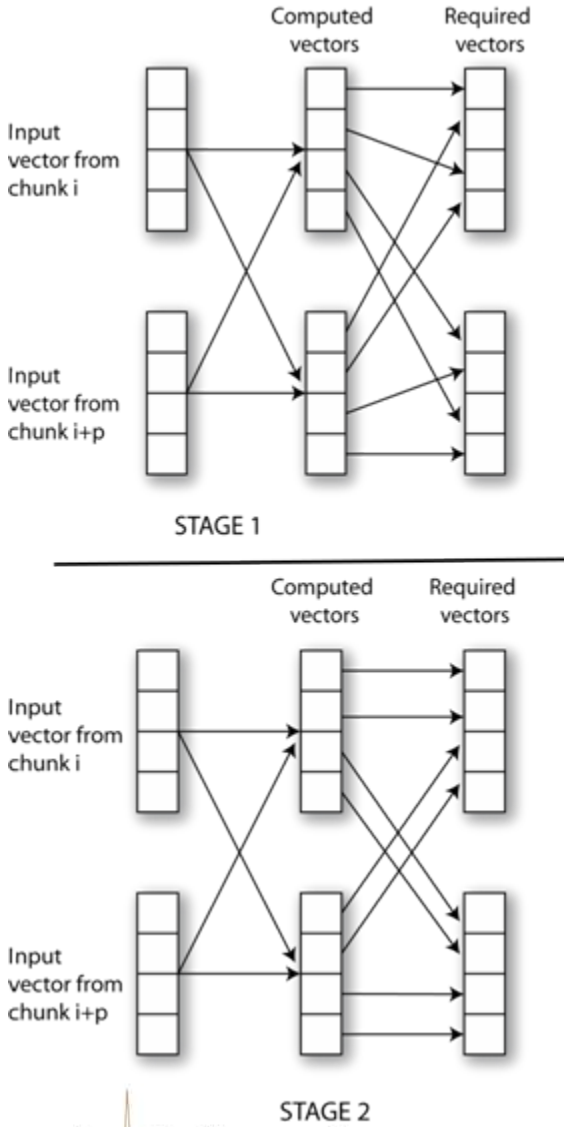


Optimization for SPE

- Loop duplication for Stages 1 & 2
  - For vectorization of these stages we need to use spu_shuffle on output vector.

- Loop duplication for NP<buffersize and otherwise.
  - Need to stall for DMA get at different places within the loop.

- Code size increases which limits the size of FFT that can be computed.



Pseudo Code for the Algorithm on Cell

Algorithm 2: Parallel FFT algorithm: View within SPE (Part I)

Input : array in PPE of size N
Output : array in PPE of size N

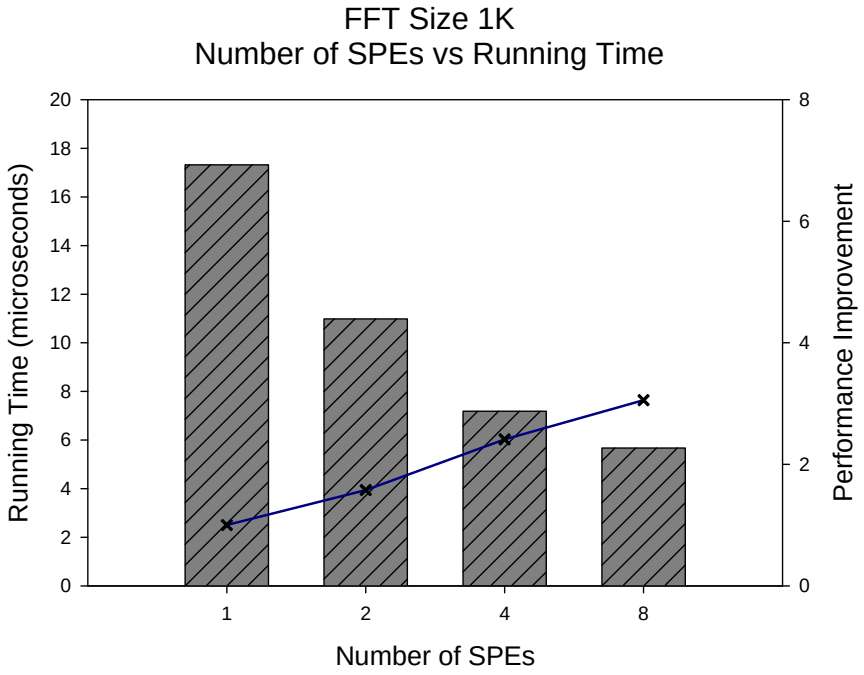
```
1  NP <- 1 ;
2  problemSize <- N ;
3  dist <- 1;
4  fetchaddr <- PPE input array ;
5  putaddr <- PPE output array ;
6  chunksize <- N/(2*p);
7  Stage 0 (SIMDization achieved with shuffling of output vector);
8  Stage 1 ;
9  while NP < buffersize && problemSize > 1 do
10     Begin Stage;
11     Initiate all DMA transfers to get data;
12     Initialize variables;
13     for j <- 0 to 2 * chunksize do
14         Stall for DMA buffer;
15         for i <- 0 to buffersize/NP do
16             for jfirst <- 0 to NP do
17                 SIMDize computation as NP > 4;
18             Update j, k, jtwiddle ;
19         Initiate DMA put for the computed results
20     swap(fetaddr, putaddr );
21     NP <- NP * 2;
22     problemSize <- problemSize/ 2;
23     dist <- dist * 2;
24     End Stage;
25     Synchronize using Inter SPE communication;
26 while problemSize > 1 do
27     Begin Stage;
28     Initiate all DMA transfers to get data;
29     Initialize variables;
30     for k <- 0 to chunksize do
31         for jfirst <- 0 to min(NP, chunksize - k) step buffersize do
32             Stall for DMA buffer;
33             for i <- 0 to buffersize do
34                 SIMDize computation as buffersize > 4;
35             Initiate DMA put for the computed results;
36         Update j, k, jtwiddle ;
37     swap(fetaddr, putaddr );
38     NP <- NP * 2;
39     problemSize <- problemSize/ 2;
40     dist <- dist * 2;
41     End Stage;
42     Synchronize using Inter SPE communication;
```

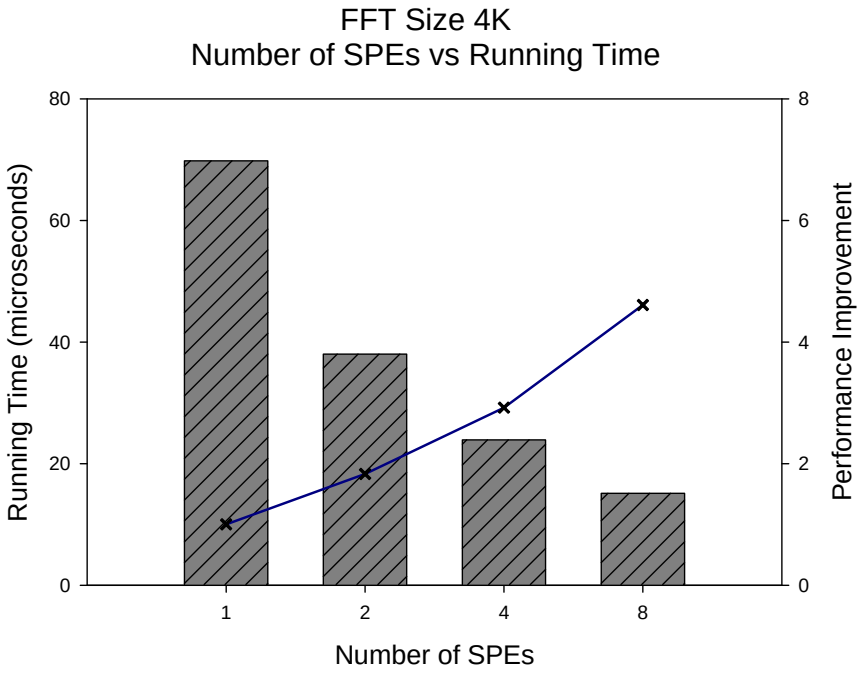


Experimental Setup

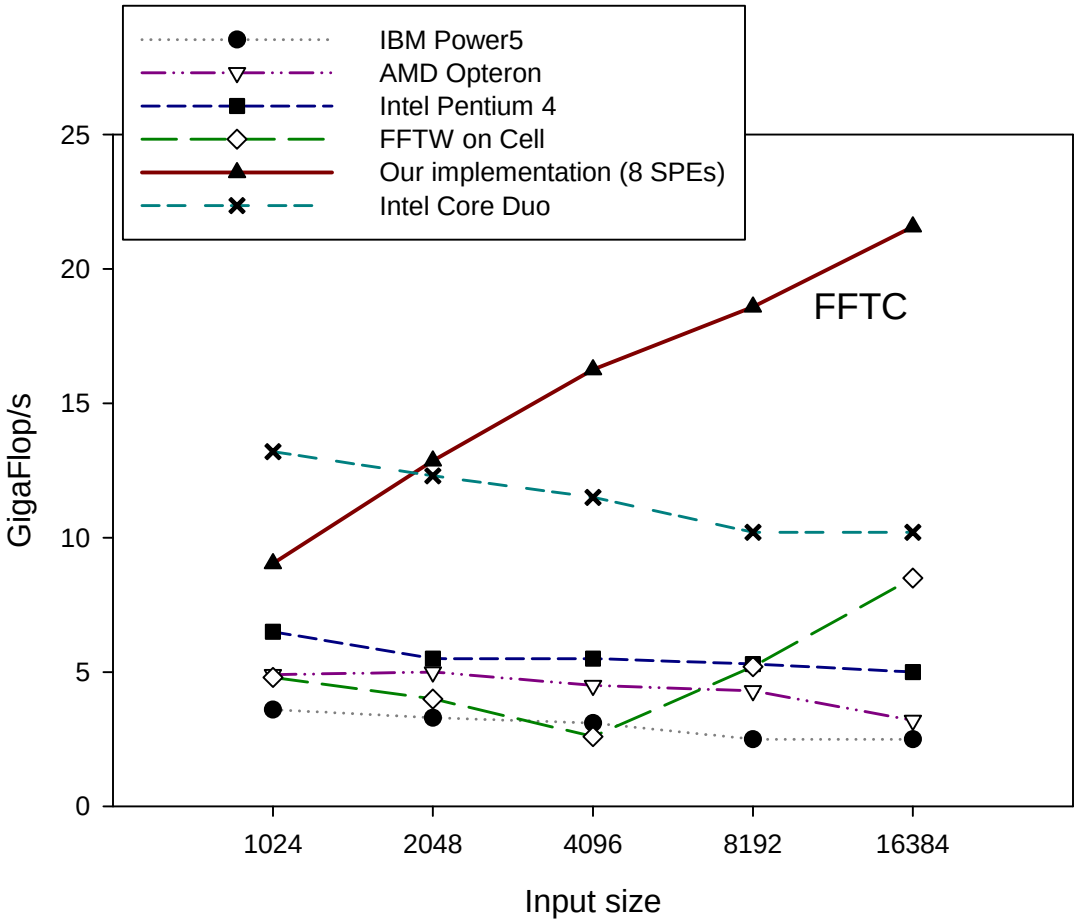
- Manual Loop unrolling, multi-buffering, inter SPE communication, odd-even pipelining, vectorization.
- Instruction level profiling and performance analysis using Cell SDK 2.1
- FLOP analysis
  - Operation Count : (5*N log *N)* floating point operations
  - For 2 complex value computations we require
    - One complex subtraction (2 FLOP), One complex addition (2 FLOP) and one complex multiplication (6 FLOP).

Performance analysis : Scaling across SPEs





Performance Comparison of FFTs



* Performance numbers from BenchFFT.



Conclusions

- FFTC is our high performance design of an FFT for a single 1-Dimensional DIF FFT.
- Use various optimization techniques such as Manual Loop unrolling, multi-buffering, inter SPE communication, odd-even pipelining, vectorization.
- Demonstrate superior performance of 18.6 GigaFlop/s for an FFT of size 8k-16K.
- Code available at http://sourceforge.net/projects/cellbuzz/

Acknowledgment of Support

- National Science Foundation
  - **CSR:** A Framework for Optimizing Scientific Applications (06-14915)
  - **CAREER:** High-Performance Algorithms for Scientific Applications (06-11589; 00-93039)
  - **ITR:** Building the Tree of Life -- A National Resource for Phyloinformatics and Computational Phylogenetics (EF/BIO 03-31654)
  - **DBI:** Acquisition of a High Performance Shared-Memory Computer for Computational Science and Engineering (04-20513).
- IBM Shared University Research (SUR) Grant
- Sony-Toshiba-IBM (STI)
- Microsoft Research
- Sun Academic Excellence Grant

Sun microsystems

DARPA

IBM

CRAY

TOSHIBA

Microsoft Research

SONY

Georgia Tech College of Computing

Thank you

Questions?