Implementation of 2-D FFT on the Cell Broadband Engine Architecture

William Lundgren (wlundgren@gedae.com, Gedae),
Kerry Barnes (Gedae), James Steed (Gedae)

HPEC 2009

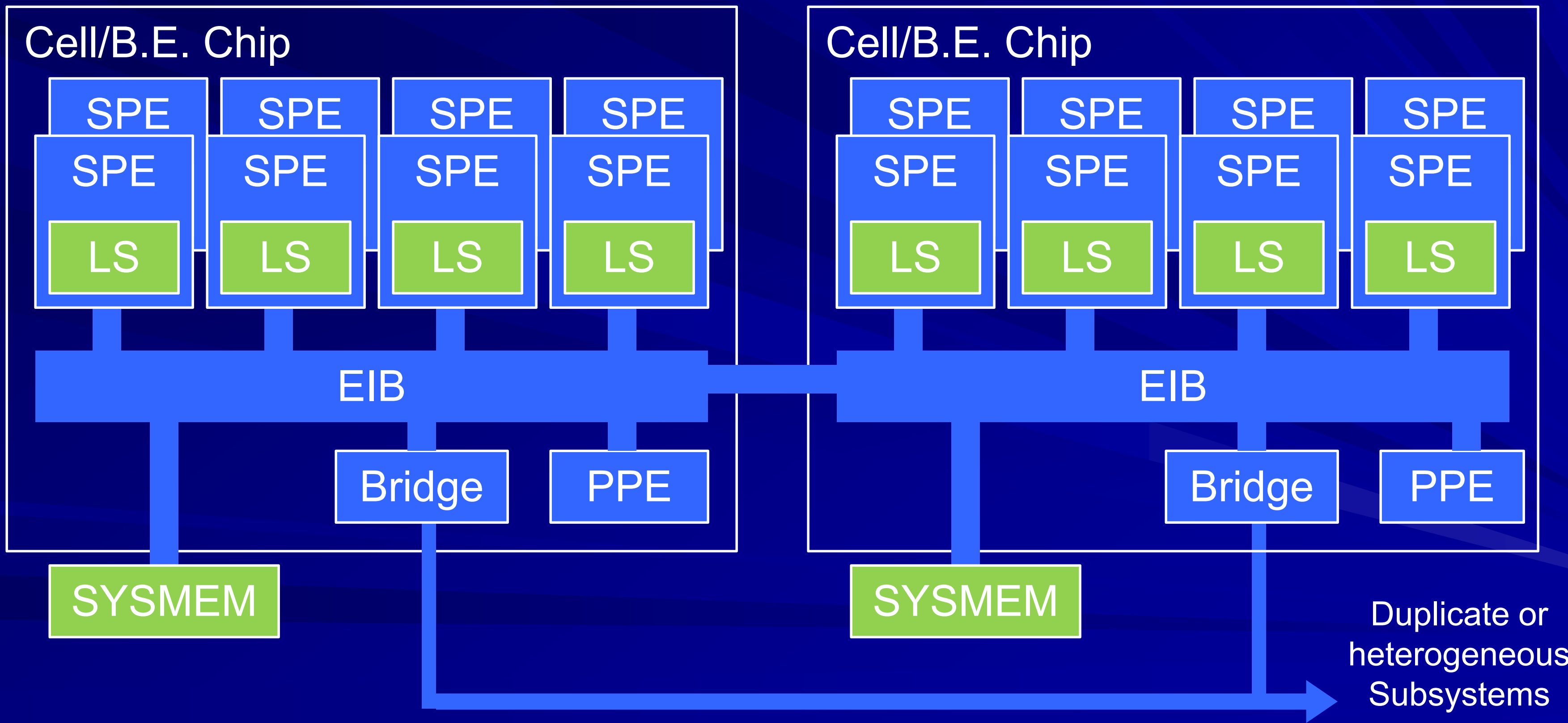
Introduction

Gedae

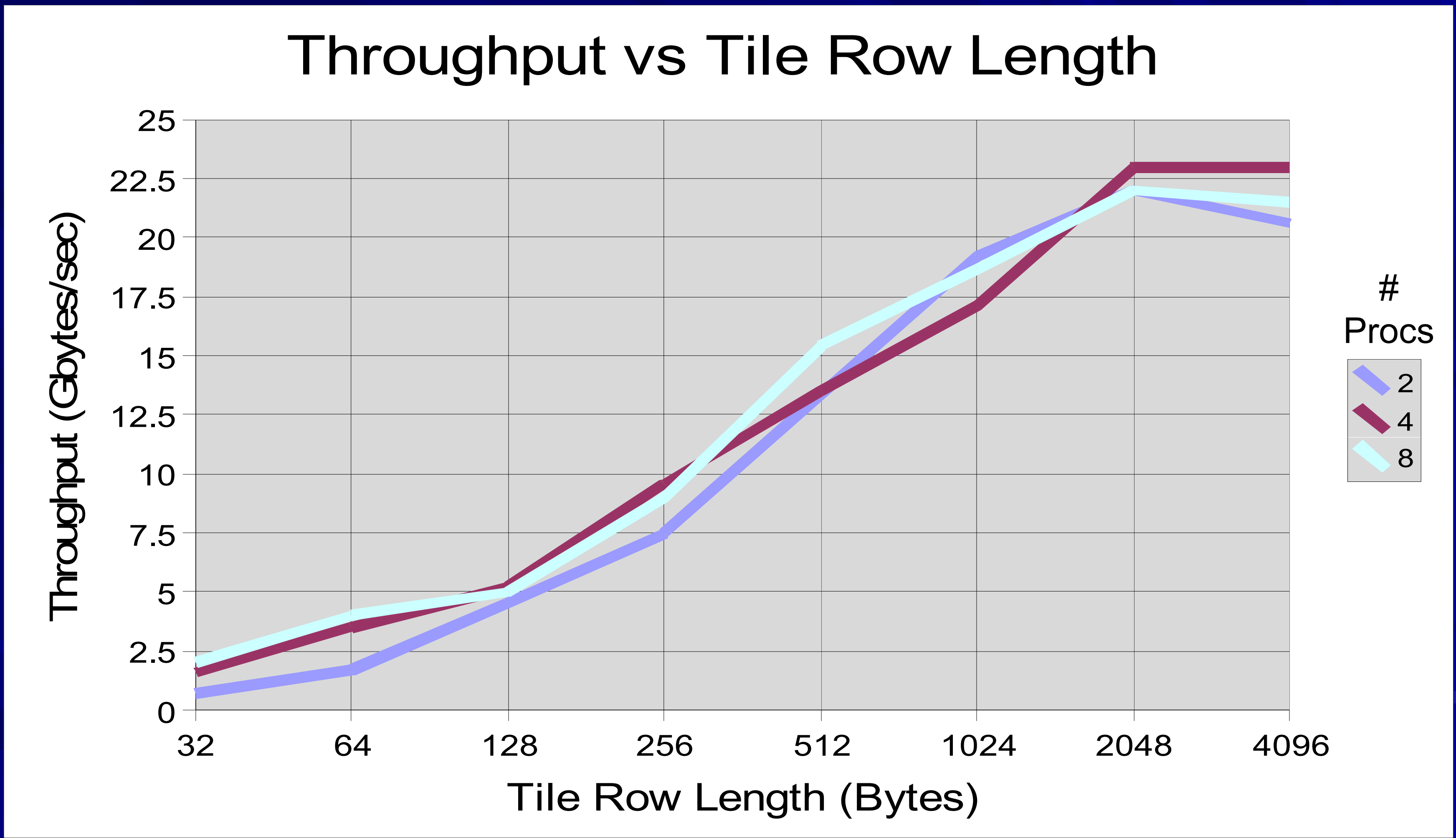
- **Processing is either limited by memory or CPU bandwidth**
  - Challenge is to achieve the practical limit of processing
  - Large 2-D FFTs are limited by memory bandwidth
- **Automating details of implementation provides developer with more opportunity to optimize structure of algorithm**
- **Cell Broadband Engine is a good platform for studying efficient use of memory bandwidth**
  - Data movements are exposed and can be controlled
  - Cache managers hide the data movement
- **Intel X86 & IBM Power processor clusters, Larrabee, Tilera, etc. have similar challenges**

Cell/B.E. Memory Hierarchy

- **Each SPE core has a 256 kB local storage**
- **Each Cell/B.E. chip has a large system memory**



Effect of Tile Size on Throughput



(Times are measured within Gedae)

Limiting Resource is Memory Bandwidth

- Simple algorithm: FFT, Transpose, FFT
  – Scales well to large matrices
- Data must be moved into and out of memory 6 times for a total of
  – 2*4*512*512*6 = 12.6e6 bytes
  – 12.6e6/25.6e9 = 0.492 mSec
  - Total flops = 5*512*log2(512) * 2 * 512 = 23.6e6
  - 23.6e6/204.8e9 = 0.115 mSec
  - Clearly the limiting resource is memory IO
- Matrices up to 512x512, a faster algorithm is possible
  — Reduces the memory IO to 2 transfers into and 2 out of system memory
  — The expected is time based on the practical memory IO bandwidth shown on the previous chart is 62 gflops

Overview of 4 Phase Algorithm

- Repeat C1 times
  - Move C2/Procs columns to local storage
  - FFT
  - Transpose C2/Procs * R2/Procs matrix tiles and move to system memory
- Repeat R1 times
  - Move R2/Procs * C2/Procs tiles to local storage
  - FFT
  - Move R2/Procs columns to system memory

Optimization of Data Movement

- We know that to achieve optimal performance we must use buffers with row length >= 2048 bytes
- Complexity beyond the reach of many programmers
- Approach:
  - Use Idea Language (Copyright Gedae, Inc.) to design the data organization and movement among memories
  - Implement on Cell processor using Gedae DF
  - Future implementation will be fully automated from Idea design
- The following charts show the algorithm design using the Idea Language and implementation and diagrams

Multicores require the introduction of fundamentally new automation.

Row and Column Partitioning

Gedae

- Procs = number of processors (8)
  - Slowest moving row and column partition index
- Row decomposition
  - R1 = Slow moving row partition index (4)
  - R2 = Second middle moving row partition index (8)
  - R3 = Fast moving row partition index (2)
  - Row size = Procs * R1 * R2 * R3 = 512
- Column Decomposition
  - C1 = Slow moving column partition index (4)
  - C2 = Second middle moving column partition index (8)
  - C3 = Fast moving column partition index (2)
  - Column size = Procs * C1 * C2 * C3 = 512

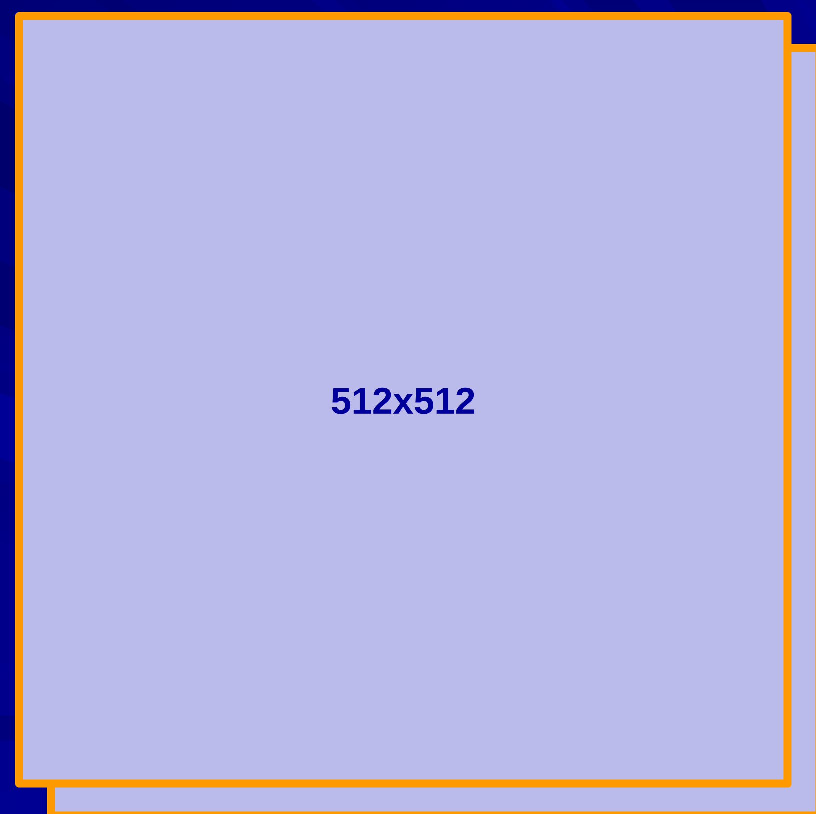
Notation – Ranges and Comma Notation

- `range r1 = R1;`
  - is an iterator that takes on the values `0, 1, … R1-1`
  - `#r1` equals `R1`, the size of the range variable

- We define ranges as:
  - `range rp = Procs; range r1 = R1;`
    `range r2 = R2;    range r3 = R3;`
  - `range cp = Procs; range r1 = R1;`
    `range r2 = R2;    range r3 = R3;`
  - `range iq = 2; /* real, imag */`
- We define comma notation as:
  - `x[rp,r1]`
    is a vector of size `#rp * #r1` and equivalent to
    `x[rp * #r1 + r1]`

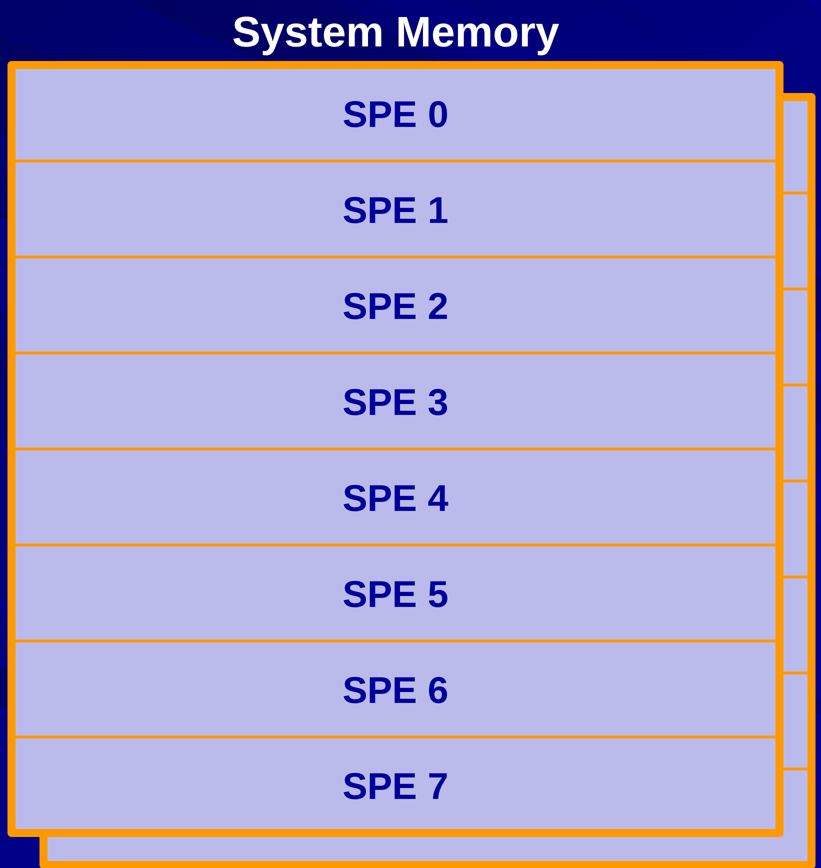
Input Matrix

- Input matrix is 512 by 512
  - `range r = R; range c = C;`
    - `r ➔ rp,r1,r2,r3`
    - `c ➔ cp,c1,c2,c3`
  - `range iq = 2`
    - `Split complex (re, im)`
  - `x[iq][r][c]`



Distribution of Data to SPEs

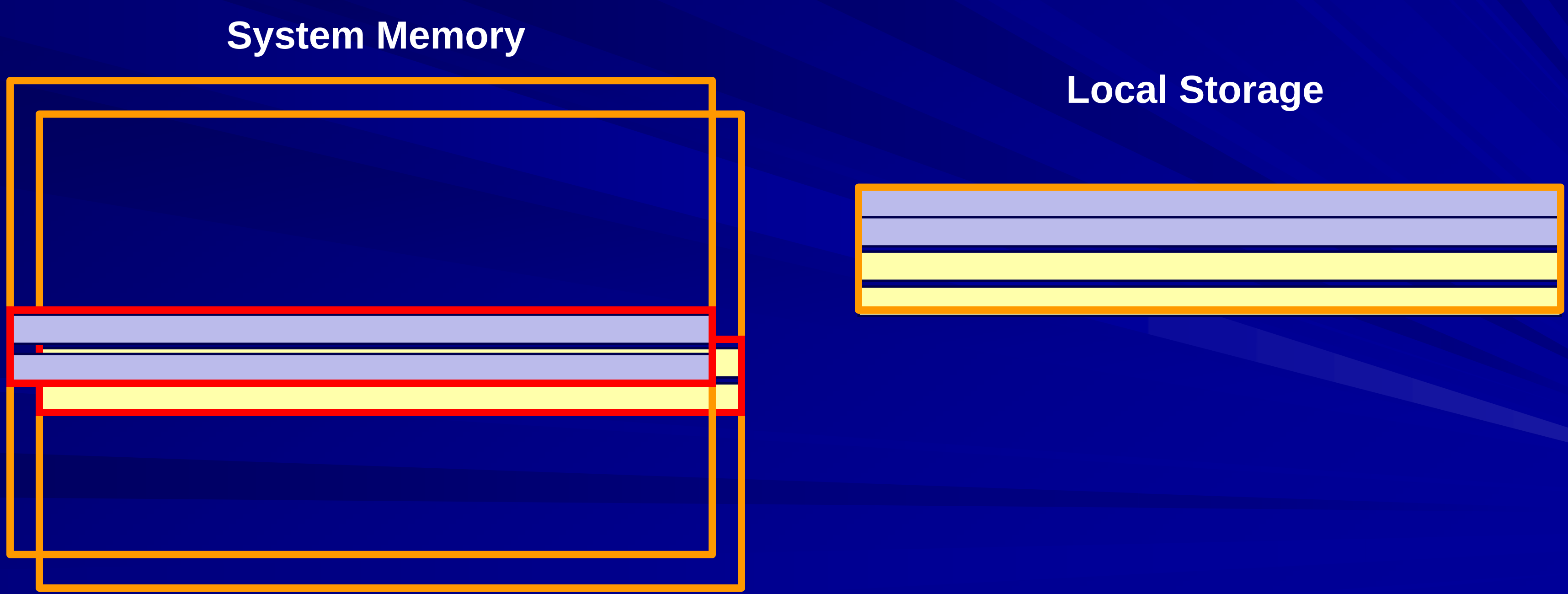
- **Decompose the input matrix by row for processing on 8 SPEs**
  `[rp]x1[iq][r1,r2,r3][c] = x[iq][rp,r1,r2,r3][c]`



Stream Submatrices from System Memory into SPE

- Consider 1 SPE
- Stream submatrices with R3 (2) rows from system memory into local store of the SPE. Gedae will use list DMA to move the data

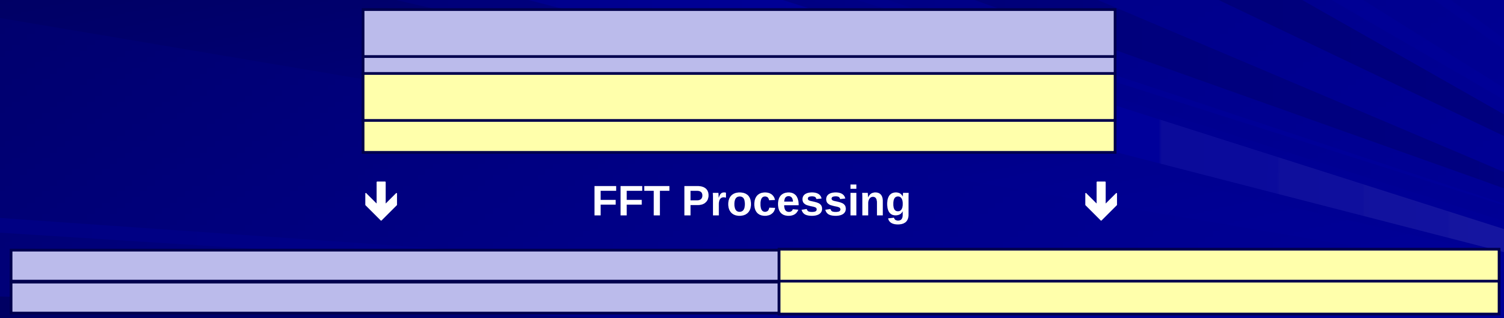
```
[rp]x2[iq][r3][c](r1,r2) = [rp]x1[iq][r1,r2,r3][c]
```



FFT Processing

- Process Submatrices by Computing FFT on Each Row
  - `[rp]x3[r3][iq][c]= fft([rp]x2[iq][r3])`
  - Since the `[c]` dimension is dropped from the argument to the fft function it is being passed a complete row (vector).
  - Stream indices are dropped. The stream processing is automatically implemented by Gedae.

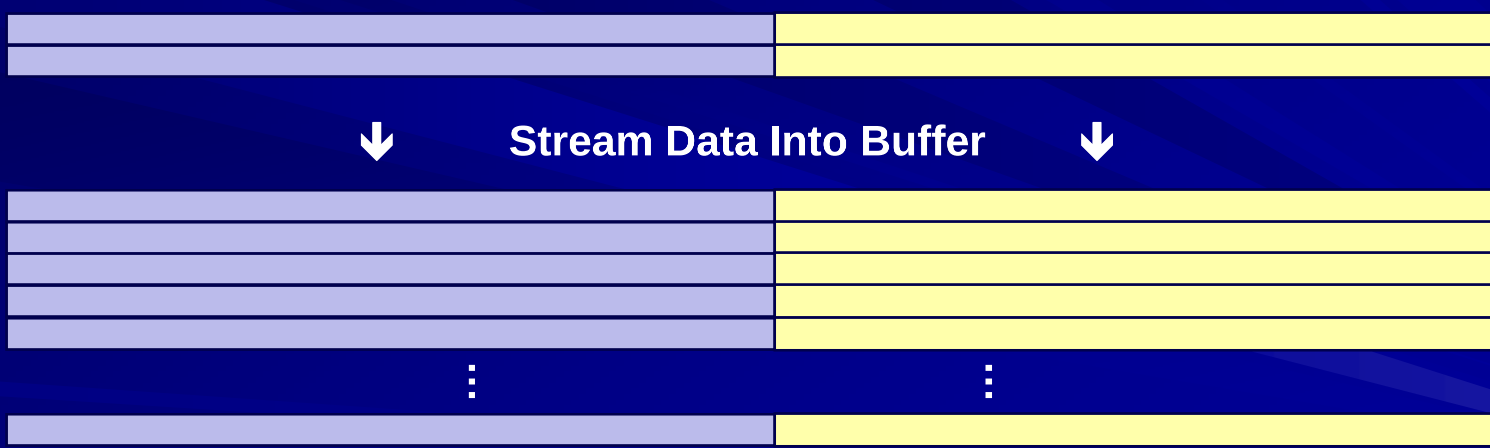
Each sub matrix contains 2 rows of real and 2 rows of imaginary data.



Notice that the real and imaginary data has been interleaved to keep each submatrix in contiguous memory.

Create Buffer for Streaming Tiles to System Memory Phase 1

- **Collect R1 submatrices into a larger buffer with R2*R3 (16) rows**
  - `[rp]x4[r2,r3][iq][c] = [rp]x3[r3][iq][c](r2)`
  - **This process is completed `r1` (4) times to process the full 64 rows.**



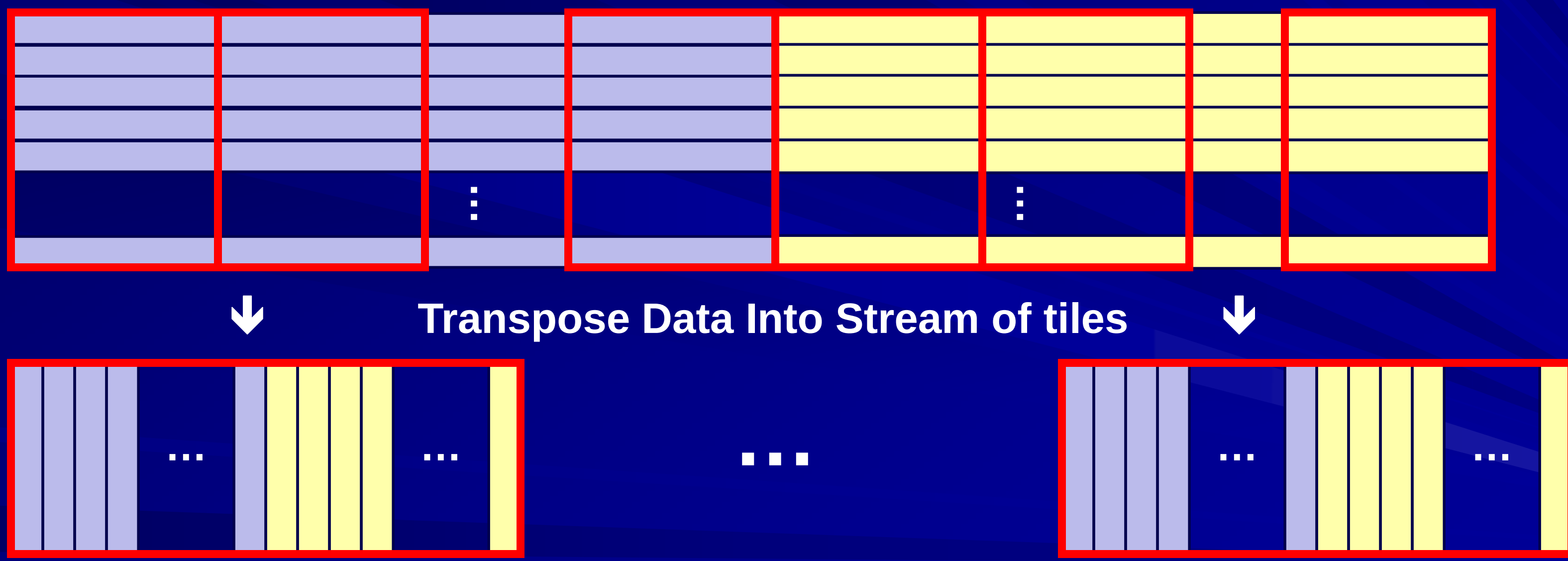
There are now R2*R3 (16) rows in memory.

Transpose Tiles to Contiguous Memory

- **A tile of real and a tile of imaginary are extracted and transposed to continuous memory**

`[rp]x5[iq][c2,c3][r2,r3](cp,c1) = [rp]x4[r2,r3][iq][cp,c1,c2,c3]`

  – **This process is completed `r1` (4) times to process the full 64 rows.**



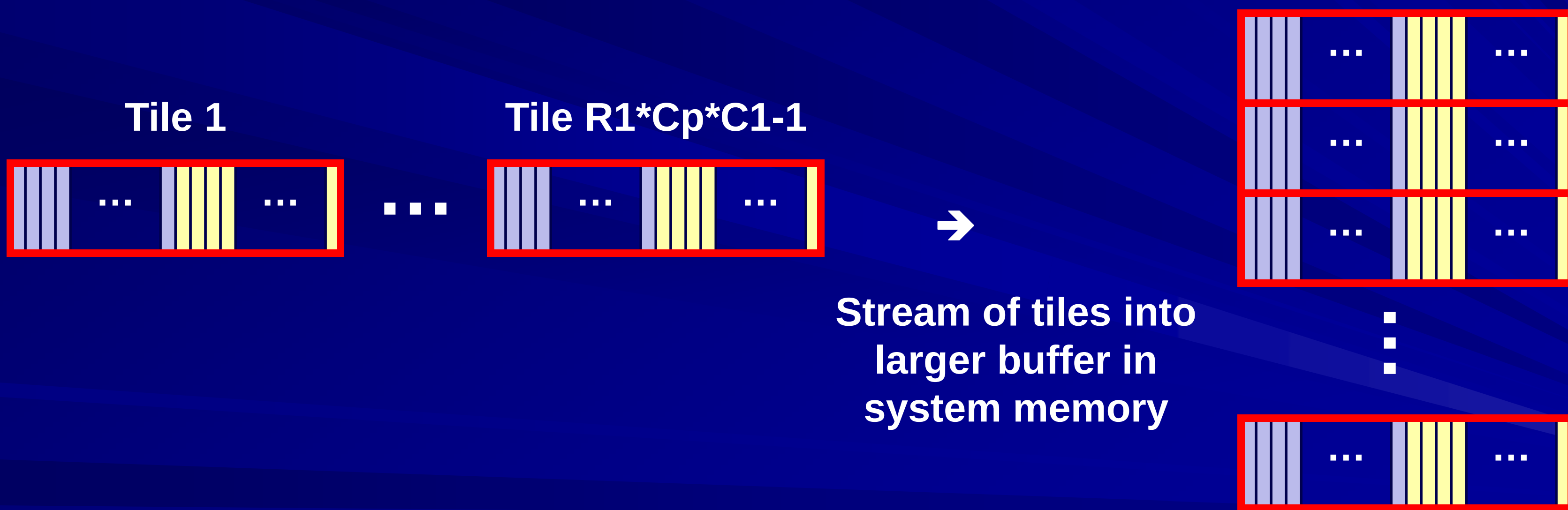
Now there is a stream of Cp*C1 (32) tiles. Each tile is IQ by R2*R3 by C2*C3 (2x16x16) data elements.

Stream Tiles into System Memory Phase 1

- **Stream tiles into a buffer in system memory. Each row contains all the tiles assigned to that processor.**

  `[rp]x6[r1,cp,c1][iq][c2,c3] = [rp]x5[iq][c2,c3][r2,r3](r1,cp,c1)`

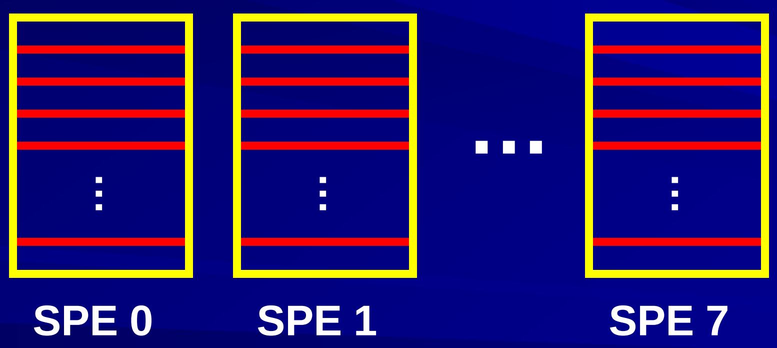
  - **The `r1` iterations were created on the initial streaming of `r1,r2` submatrices.**



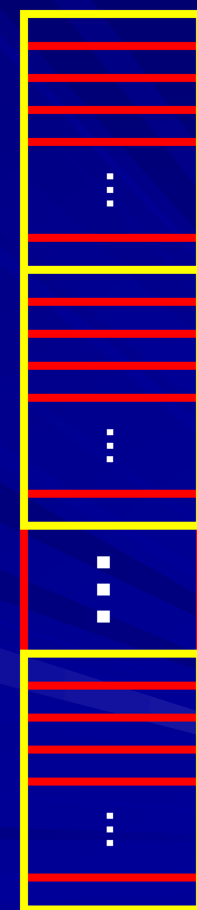
Now there is a buffer of R1*Cp*C1 (128) tiles each IQ by R2*R3 by C2*C3

Stream Tile into System Memory Phase 2

- Collect buffers from each SPE into full sized buffer in system memory.
  - `x7[rp,r1,cp,c1][iq][c2,c3][r2,r3] = [rp]x6[r1,cp,c1][iq][c2,c3][r2,r3]`
  - The larger matrix is created by Gedae and the pointers passed back to the box on the SPE that is DMA'ng the data into system memory



➔ Collect tiles into larger buffer in system memory



The buffer is now `Rp*R1*Cp*C1` (1024) tiles.
Each tile is IQ by R2*R3 by C2*C3

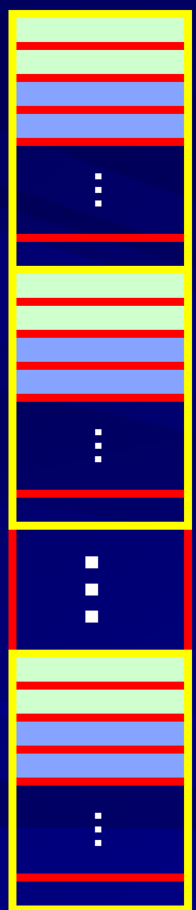
Stream Tiles into Local Store

Gedae

- Extract tiles from system memory to create 16 full sized columns (r index) in local store.

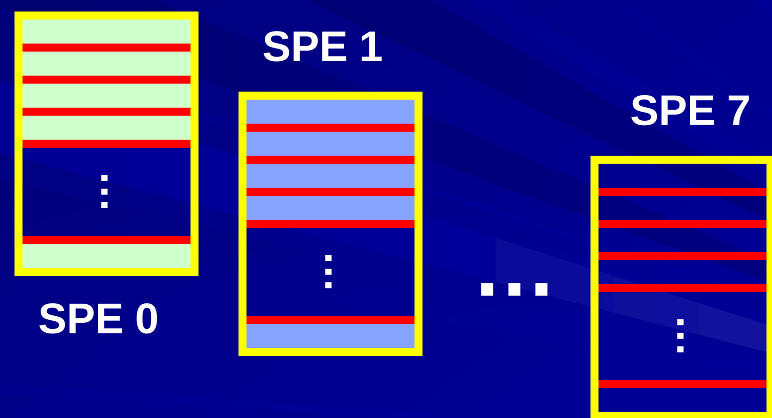
```
[cp]x8[iq][c2,c3][r2,r3](c1,rp,r1) =
              x7[rp,r1,cp,c1][iq][c2,c3][r1,r2];
```

  – All SPEs have access to full buffer to extract data in a regular but scattered pattern.



➔

Collect tiles into local store from regular but scattered locations in system memory



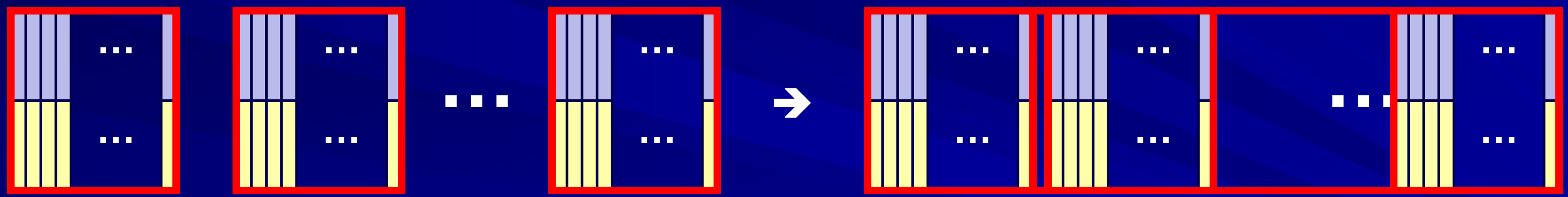
The buffer in local store is now Rp*R1 (32) tiles. Each tile is IQ by C2*C3 by R2*R3 (2x16x16). This scheme is repeated C1 (4) times on each SPE.

Stream Tiles into Buffer

- **Stream tiles into a buffer in system memory. Each row contains all the tiles assigned to that processor.**

```
[cp]x9[iq][c2,c3][rp,r1,r2,r3](c1) =
                    [cp]x8[iq][c2,c3][r2,r3](c1,rp,r1)
```

  - **The `r1` iterations were created on the initial streaming of `r1,r2` submatrices.**



Stream of tiles into full length column (r index) buffer with a tile copy.

Now there is a buffer of R1*Cp*C1 (128) tiles each IQ by R2*R3 by C2*C3

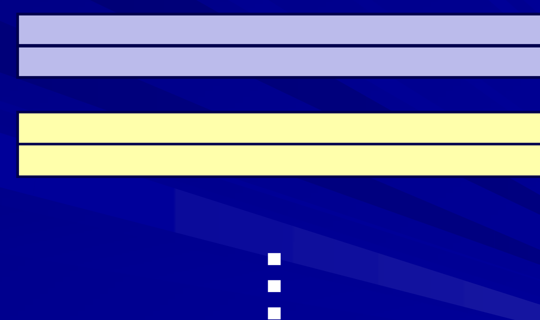
Process Columns with FFT

- **Stream 2 rows into an FFT function that places the real and imaginary data into separate buffers. This allows reconstructing a split complex matrix in system memory.**

```
[cp]x10[iq][c3][r](c2) = fft([cp]x9[iq][c2,c3]);
```

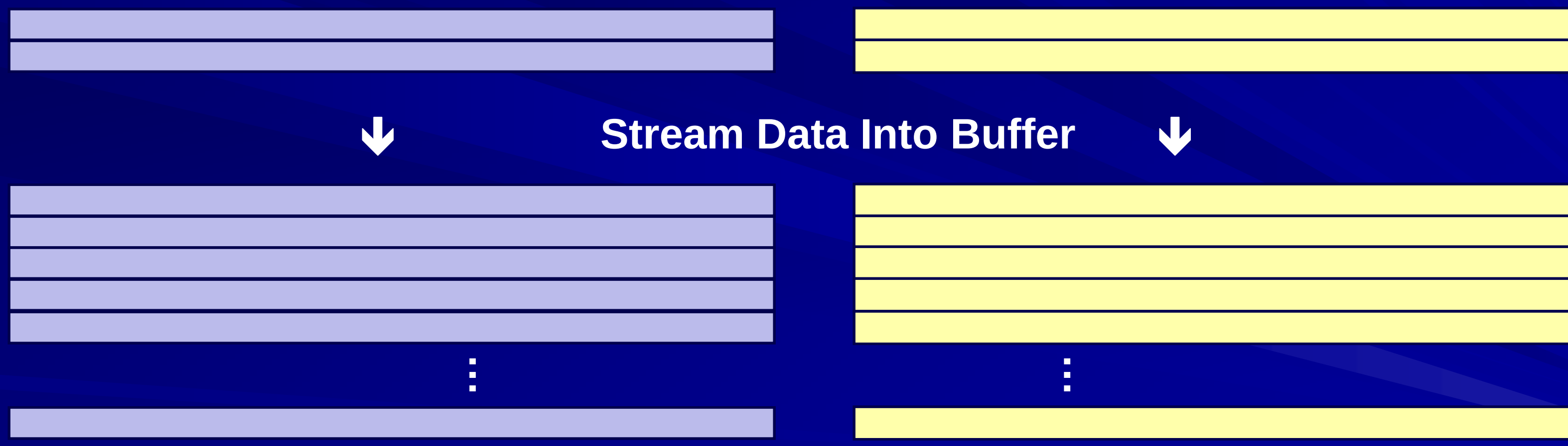
➔

Stream 2 rows of real and imaginary data into FFT function. Place data into separate buffers on output.



Create Buffer for Streaming Tiles to System Memory

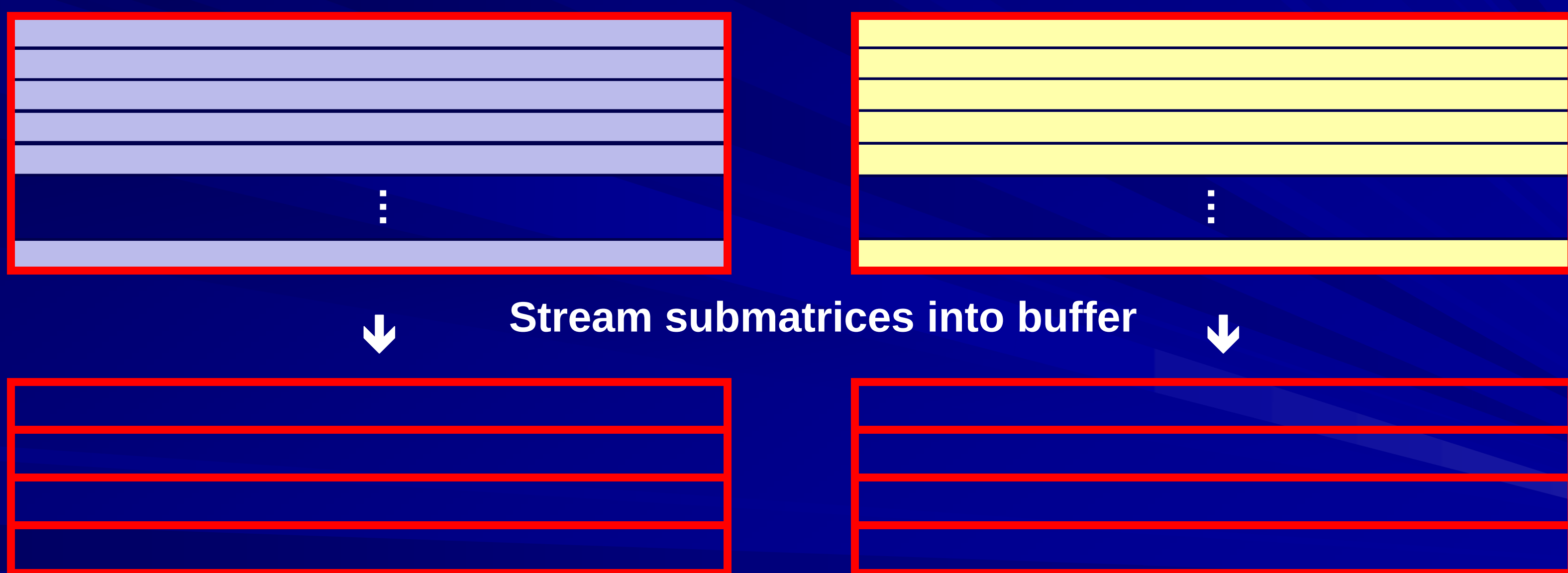
- **Collect R2 submatrices into a larger buffer with R2*R3 (16) rows**
  - `[p]x11[iq][c2,c3][r] = [cp]x10[iq][c3][r](c2)`
  - **This process is completed c1 (4) times to process the full 64 rows.**



There are now R2*R3 (16) rows in memory.

Create Buffer for Streaming Tiles to System Memory

- Collect R2 submatrices into a larger buffer with R2*R3 (16) rows
  - `[p]x12[iq][c1,c2,c3][r] = [p]x11[iq][c2,c3][r](c1)`



There are now 2 buffers of C1*C2*C3 (128) rows of length R (512) in system memory.

Distribution of Data to SPEs

- **Decompose the input matrix by row for processing on 8 SPEs**

```
x13[iq][cp,c1,c2,c3][r] = [cp]x12[iq][c1,c2,c3][r]
```

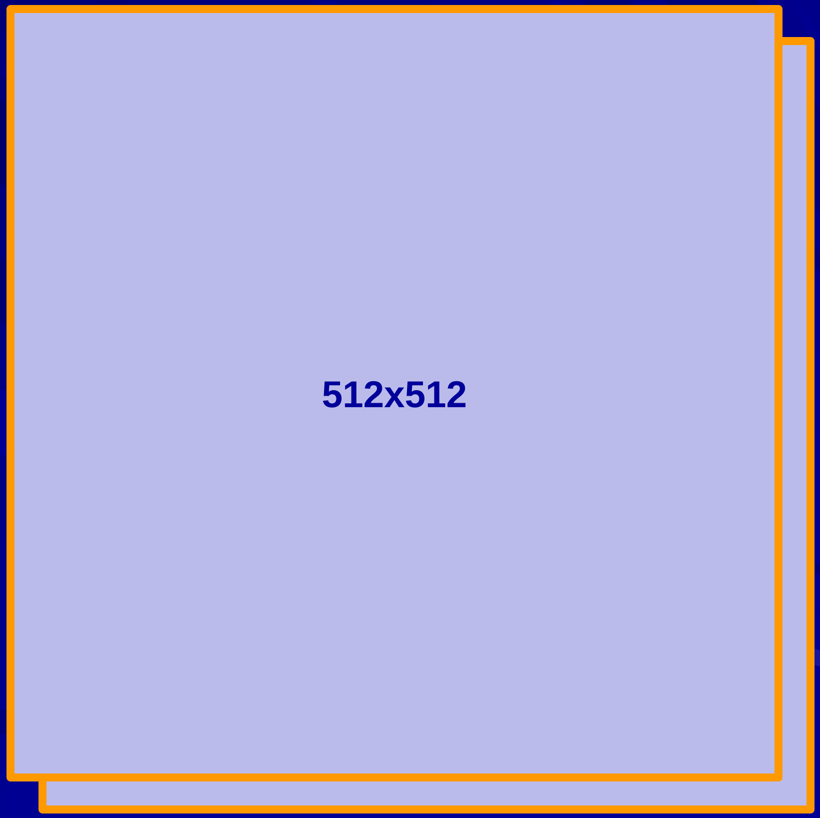
System Memory

Only real plane represented in picture.

Each SPE will produce
C1*C2*C3 = 64
rows.

Output Matrix

- **Output matrix is 512 by 512**
  - `y[iq][c][r]`



Results

- Two days to design algorithm using Idea language
  - 66 lines of code
- One day to implement on Cell
  - 20 kernels, each with 20-50 lines of code
  - 14 parameter expressions
  - Already large savings in amount of code over difficult handcoding
  - Future automation will eliminate coding at the kernel level altogether
- Achieved 57* gflops out of the theoretical maximum 62 gflops

* Measured on CAB Board at 2.8 ghz adjusted to 3.2 ghz. The measure algorithm is slightly different and expected to increase to 60 gflops.

Gedae Status

- 12 months into an 18/20 month repackaging of Gedae
  - Introduced algebraic features of the Idea language used to design this algorithm
  - Completed support for hierarchical memory
  - Embarking now on code overlays and large scale heterogeneous systems
  - Will reduce memory footprint to enable 1024 x 1024 2D FFT