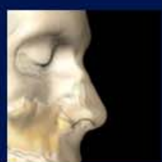
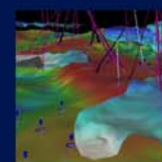
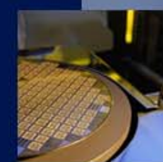
Mercury Computer Systems, Inc.

Challenges Drive Innovation™





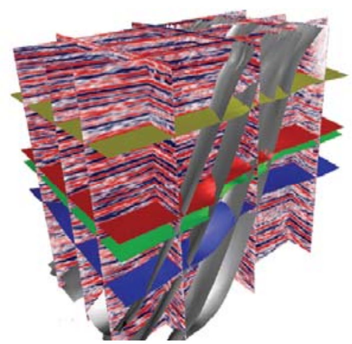



Programming Cell Systems – Mercury's MultiCore Framework & Scientific Algorithm Library

Brian Bouzas -- Staff Systems Engineer

About Mercury Computer Systems

- **Leader in high-performance computing for defense and commercial applications**
- **First non-gaming company to integrate the Cell Broadband Engine™ (BE) processor into its products**
  - High-volume gaming market is transforming the technology industry
- **Targeting applications in existing and new markets with optimized Cell BE-based products**
  - Medical imaging, inspection, defense, geosciences, telecommunications, etc.
  - The Cell BE is designed to solve the types of problems Mercury has been solving for many years



Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc.

Mercury's Cell Programming Model

- **Multicomputer with Function Offload Engines**
  - SPEs perform computations & move data
  - PPE manages the worker processors and handles “outer loop” setup
- **Write code for both processing elements**
  - Algorithms for SPEs (workers)
  - Control code for PPE (manager)
- **View PPE & XDR memory as traditional multicomputer node**
  - Use favorite middleware (PAS, MPI, …) to move data and coordinate processing among nodes
- **Cell architecture (256KB local store) dictates this programming model for performance**

Overview of Mercury’s Multi-Core Framework

- Simplifies development of high-performance applications on multi-core processors like Cell
- Preserves limited SPE memory for application code & data
- Runs SPE tasks without Linux overhead
- Data movement and synchronization features are “built-in” to the network
- Provides a convenient API to describe how data is organized within XDR and SPE memories
- Manager (PPE) handles “outer loop” setup
- Derived from existing, proven software technologies

Multi-Core Framework (MCF) – Basic Concepts

- PPE **Manager** creates a **Network** of 1 to 8 **Worker** SPEs
  - an SPE can belong to only 1 MCF network
- MCF Network enables
  - barrier synchronization
  - semaphores
  - message queues
  - access to remote “named” memory
- PPE Manager directs **Teams** of Worker SPEs to run **Tasks**
  - Tasks have a main()
  - an SPE runs one task at a time (run to completion model)
  - SPEs may belong to multiple teams

“hello world”

- **Manager Program**

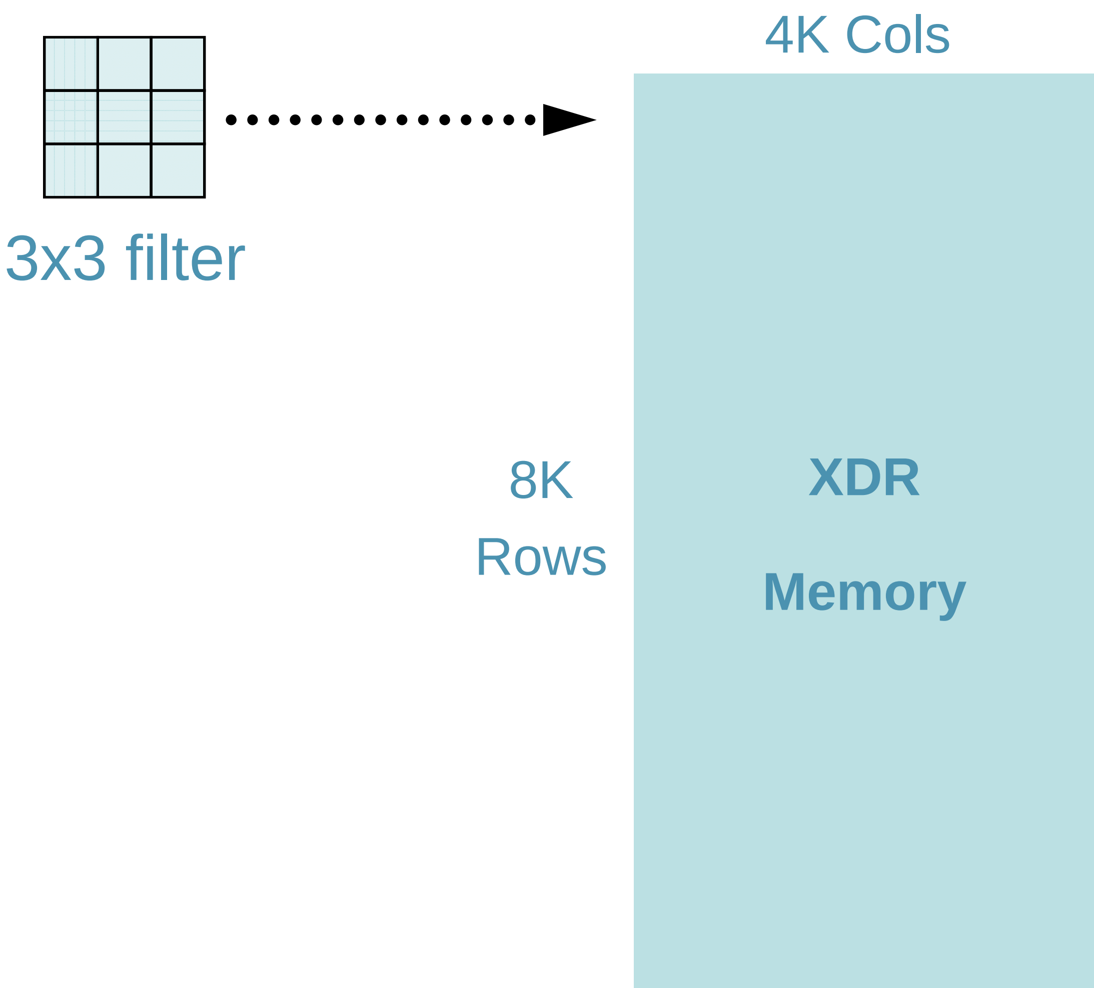
```
main(int argc, char **argv) {

  mcf_m_net_create();               // specify number of workers (SPEs)
  mcf_m_net_initialize();           // launch MCF kernel on workers

  mcf_m_net_add_task_by_path();     // load worker code into XDR memory
  mcf m team run task();            // run task on each worker
                                    //     pass arguments to each worker

  mcf_m_team_wait();                // wait for each team member's task to exit

  mcf_m_net_remove_task();          // free memory holding worker executable
  mcf_m_net_destroy();              // free memory associated with MCF network
}
```

- **Worker Program**

```
mcf_w_main (int n_bytes, void * p_arg_ls) {

  // arguments are available in local store memory
  printf();
}
```

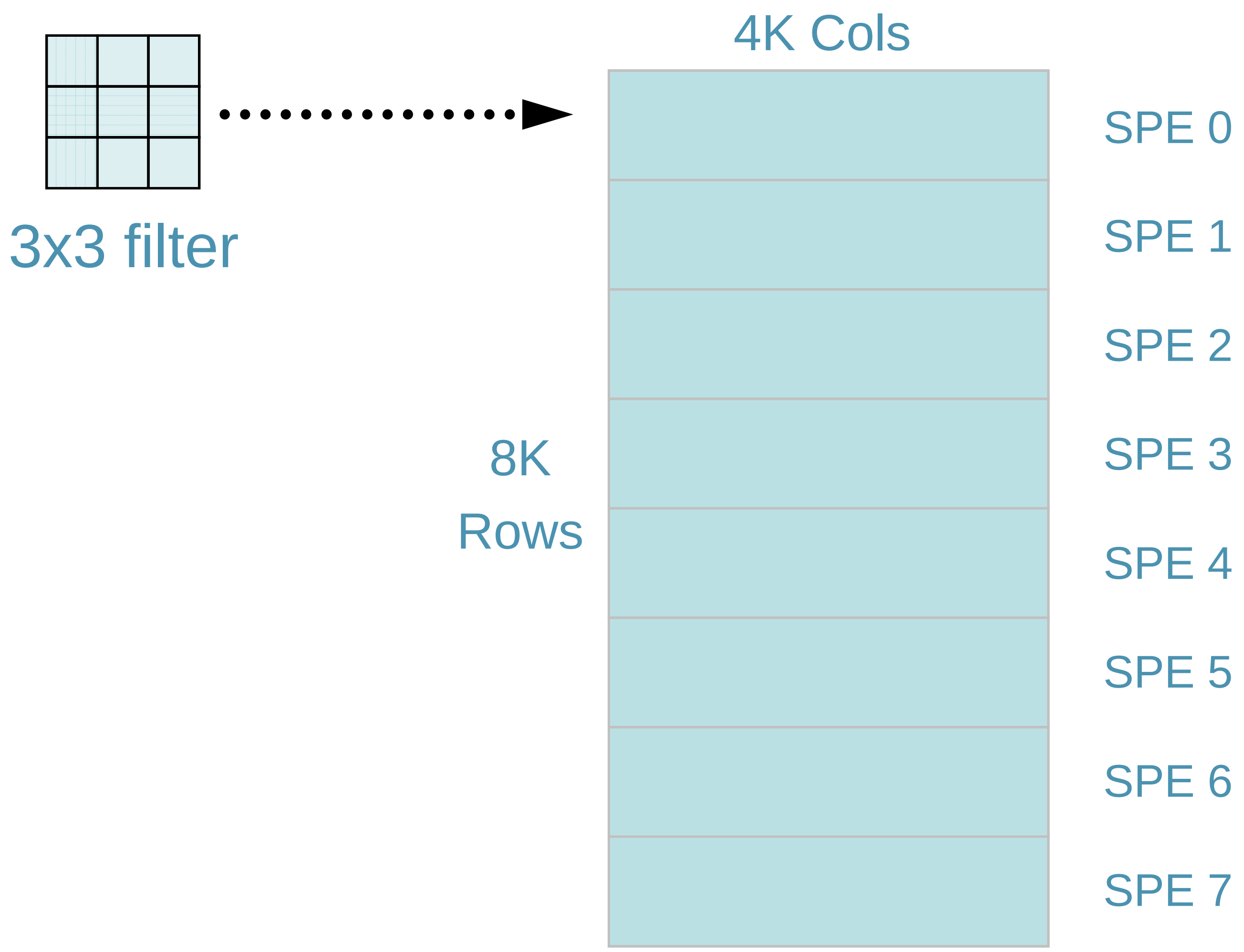
Example – 3x3 Image Filter

- **Run a 3x3 image filter over a 4Kx8K 8-bit image**



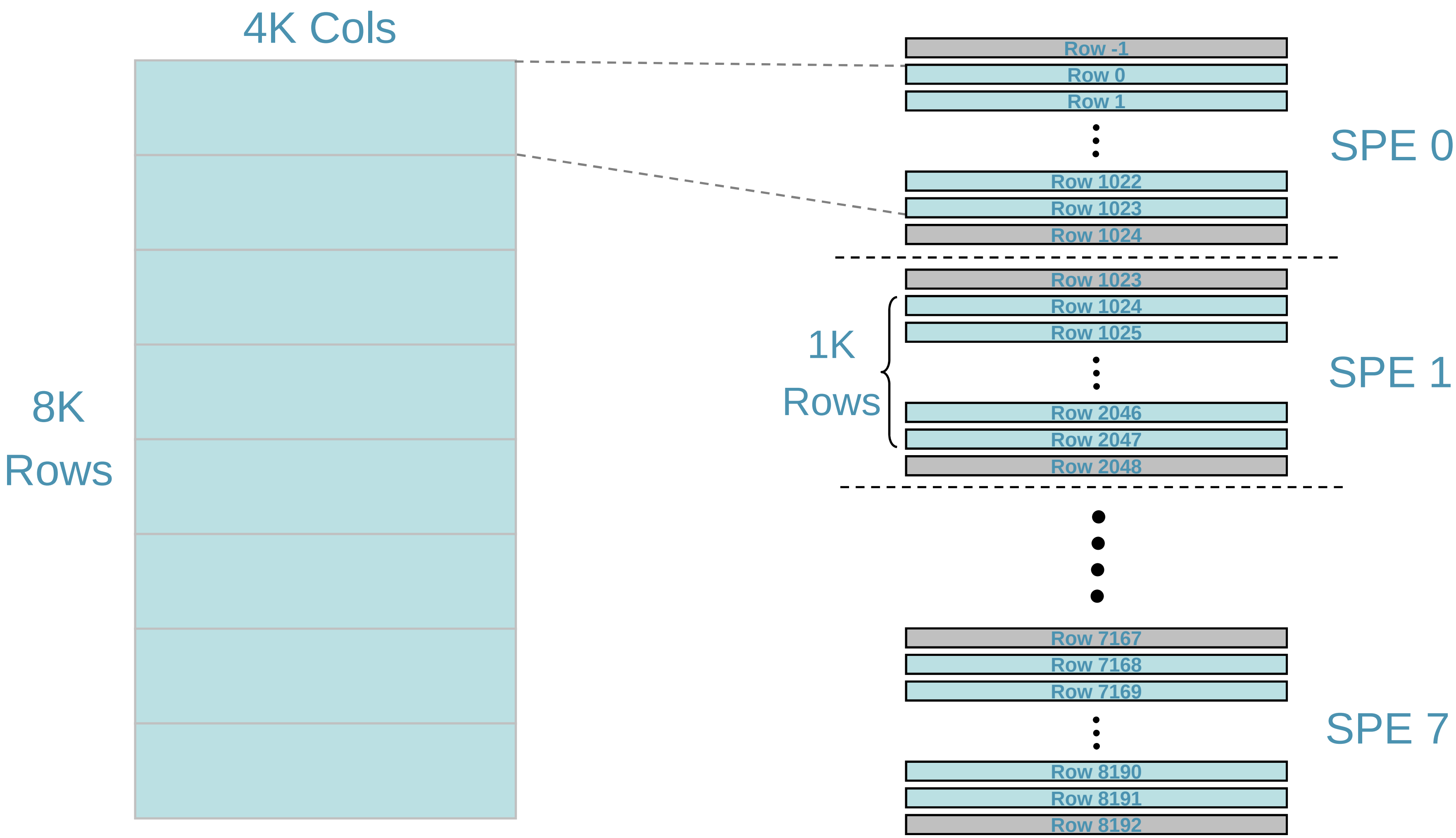
Example – 3x3 Image Filter

- **Assign rows to the 8 SPEs**



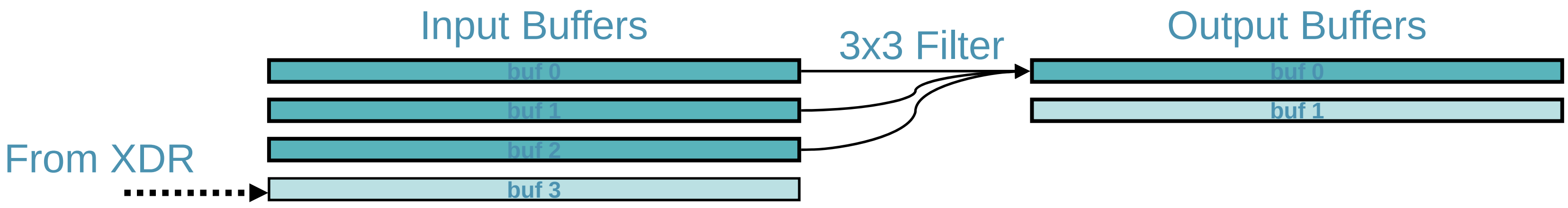
3x3 Image Filter

- **Include overlap in the partitioning**



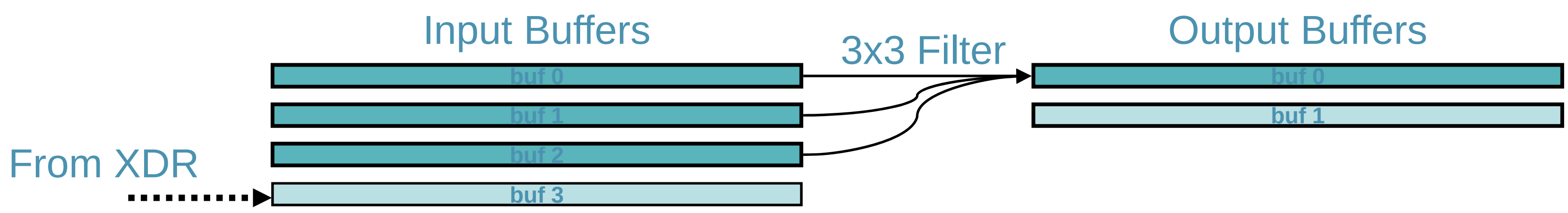
3x3 Image Filter (Overlapped I/O & Processing)

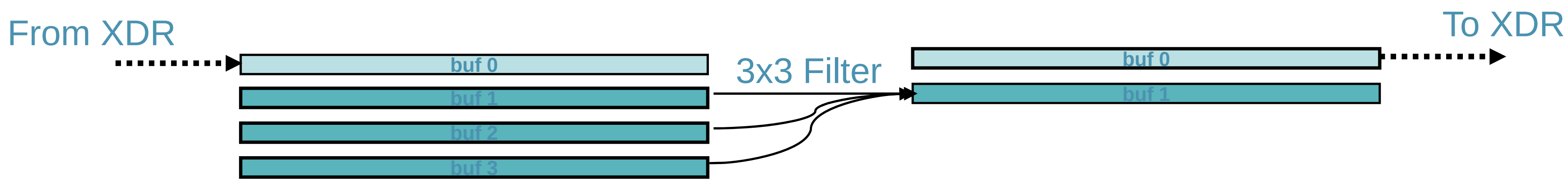
SPE Memory



3x3 Image Filter (Overlapped I/O & Processing)

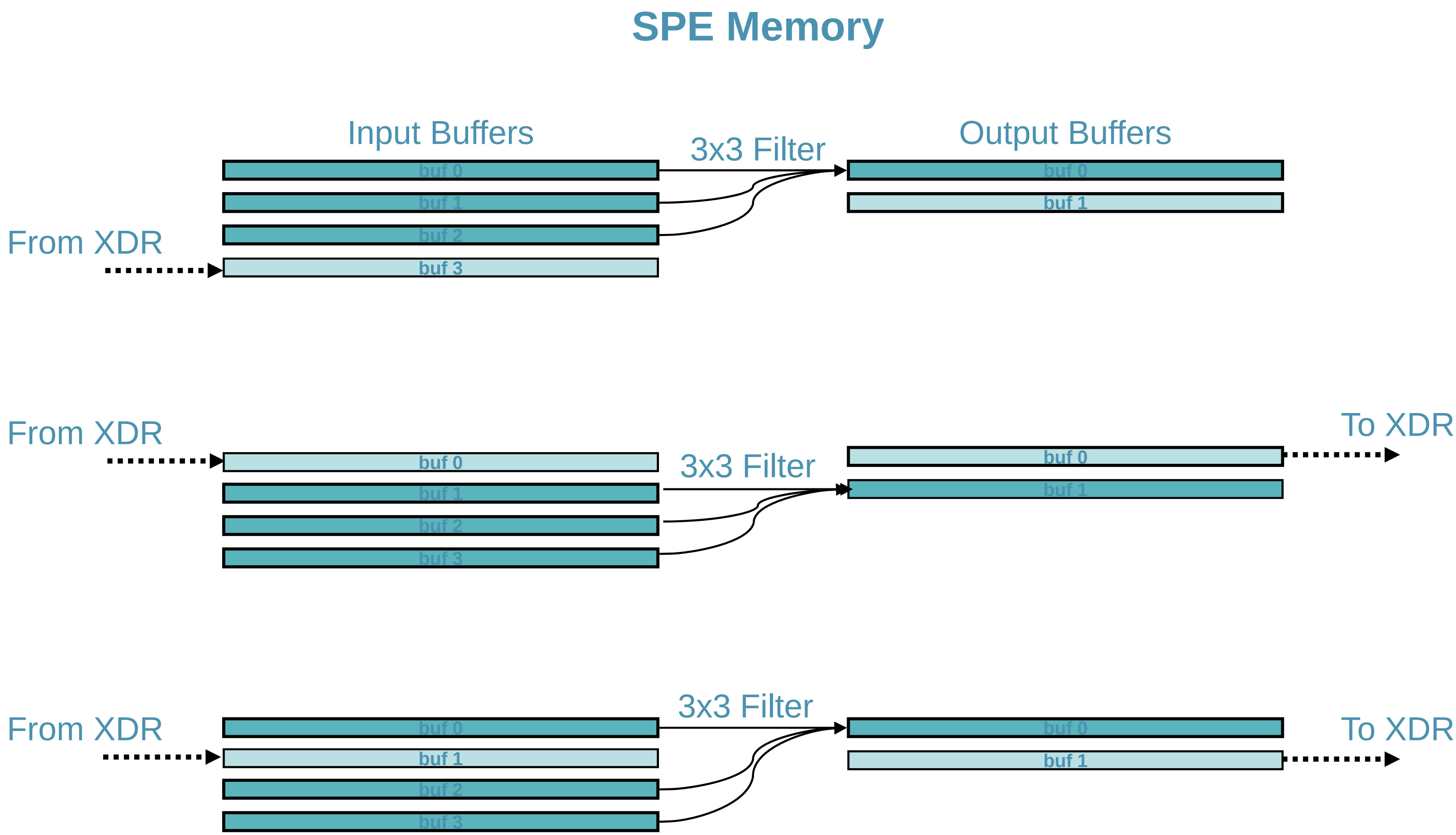
SPE Memory





3x3 Image Filter (Overlapped I/O & Processing)

Computer Systems, Inc.
MERCURY
Challenges Drive Innovation™



Tile (Strip-Mining) Channel Abstraction

- **A "Tile Channel" is the Mercury abstraction for multi-buffered I/O**
- **Tile Channel =**
  - XDR buffer
  - Description of how XDR buffer is divided
    - Size of a tile
    - Which tiles are associated with which SPEs
  - Set of work buffers in SPE local memories
- **The channel**
  - Does the data movement between source and destination
  - Handles synchronization
  - "Owns" the XDR & SPE data buffers during transfers (i.e. when buffers are "in" the channel)
  - Starts (by default) with all the buffers in the channel & empty

MCF Source Tile Channel

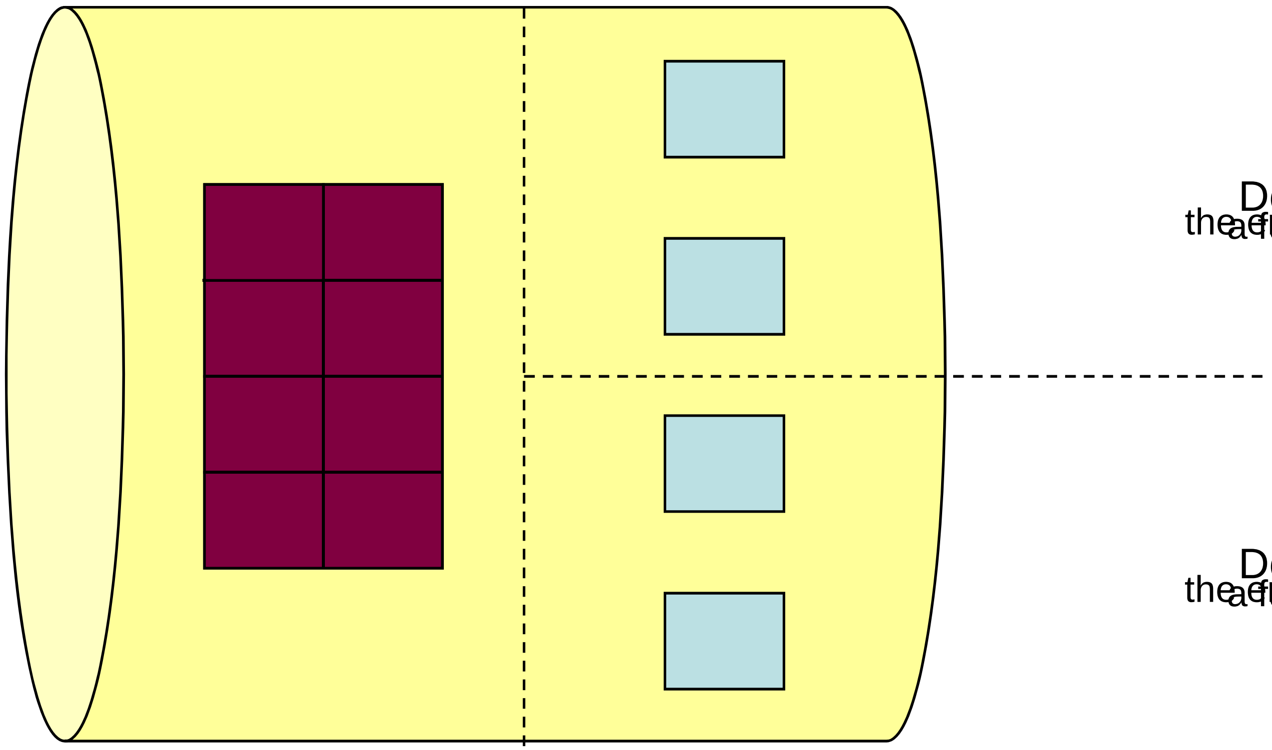
Manager Memory
(XDR)

Worker 1 Memory
(SPE Local Store)

GET
an empty buffer

fill
buffer

PUT
the full buffer



GET / **PUT**
Do Math
the empty buffer / a full buffer

GET / **PUT**
Do Math
the empty buffer / a full buffer

Worker 2 Memory
(SPE Local Store)

Tile Channel Abstraction

- **SPE program makes calls to get() & put()**
  - "get" a buffer from the channel
  - "put" a buffer back into the channel
- **Source / Input (XDR -> SPE)**
  - **get()** provides a buffer filled with new data
  - **put()** tells the channel the SPE is done reading the buffer. The channel will fill it in the background.
- **Destination / Output (SPE -> XDR)**
  - **get()** provides an empty buffer that can safely be filled
  - **put()** tells the channel the SPE is done filling the buffer. The channel will move data to XDR in the background.

3x3 Image Filter – Manager Program

```
main(int argc, char **argv)
{
  mcf_m_net_create();                                      // specify number of workers (SPEs)
  mcf_m_net_initialize();                                  // launch MCF kernel on workers

  mcf_m_net_add_task_by_embedded_name();                   // load SPE executable to XDR
  mcf_m_team_run_task();                                   // run task on each worker

  mcf_m_tile_distribution_create_2d(“in”);                 // specify XDR buffer size & tile size
  mcf_m_tile_distribution_set_assignment_overlap(“in”);    // specify overlap between partitions
  mcf_m_tile_distribution_create_2d(“out”);

  mcf_m_tile_channel_create(“in”);                         // source channel
  mcf_m_tile_channel_create(“out”);                        // destination channel
  mcf_m_tile_channel_connect(“in”);
  mcf_m_tile_channel_connect(“out”);

  mcf_m_tile_channel_get_buffer(“in”);                     // get XDR address of buffer to fill
  // fill input image here
  mcf_m_tile_channel_put_buffer(“in”);                     // make data available to workers

  mcf_m_tile_channel_get_buffer(“out”);                    // wait for results

  mcf_m_tile_channel_disconnect(“in”);
  mcf_m_tile_channel_disconnect(“out”);
  mcf_m_tile_channel_destroy(“in”);
  mcf_m_tile_channel_destroy(“out”);

  mcf_m_team_wait();                                       // wait for each team member's task to exit
  mcf_m_net_remove_task();
  mcf_m_net_destroy();                                     // free memory associated with MCF network
}
```

3x3 Image Filter – Manager Program

```
main(int argc, char **argv)
{
  mcf_m_net_create();                                     // specify number of workers (SPEs)
  mcf_m_net_initialize();                                 // launch MCF kernel on workers

  mcf_m_net_add_task_by_embedded_name();                  // load SPE executable to XDR
  mcf_m_team_run_task();                                  // run task on each worker

  mcf_m_tile_distribution_create_2d("in");                // specify XDR buffer size & tile size
  mcf_m_tile_distribution_set_assignment_overlap("in");   // specify overlap between partitions
  mcf_m_tile_distribution_create_2d("out");

  mcf_m_tile_channel_create("in");                        // source channel
  mcf_m_tile_channel_create("out");                       // destination channel
  mcf_m_tile_channel_connect("in");
  mcf_m_tile_channel_connect("out");

  mcf_m_tile_channel_get_buffer("in");                    // get XDR address of buffer to fill
  // fill input image here
  mcf_m_tile_channel_put_buffer("in");                    // make data available to workers

  mcf_m_tile_channel_get_buffer("out");                   // wait for results

  mcf_m_tile_channel_disconnect("in");
  mcf_m_tile_channel_disconnect("out");
  mcf_m_tile_channel_destroy("in");
  mcf_m_tile_channel_destroy("out");

  mcf_m_team_wait();                                      // wait for each team member's task to exit
  mcf_m_net_remove_task();
  mcf_m_net_destroy();                                    // free memory associated with MCF network
}
```

3x3 Image Filter – Manager Program

Computer Systems, Inc. MERCURY Challenges Drive Innovation™

```
main(int argc, char **argv)
{
  mcf_m_net_create();                                        // specify number of workers (SPEs)
  mcf_m_net_initialize();                                    // launch MCF kernel on workers

  mcf_m_net_add_task_by_embedded_name();                     // load SPE executable to XDR
  mcf_m_team_run_task();                                     // run task on each worker

  mcf_m_tile_distribution_create_2d("in");                   // specify XDR buffer size & tile size
  mcf_m_tile_distribution_set_assignment_overlap("in");      // specify overlap between partitions
  mcf_m_tile_distribution_create_2d("out");

  mcf_m_tile_channel_create("in");                           // source channel
  mcf_m_tile_channel_create("out");                          // destination channel
  mcf_m_tile_channel_connect("in");
  mcf_m_tile_channel_connect("out");

  mcf_m_tile_channel_get_buffer("in");                       // get XDR address of buffer to fill
  // fill input image here
  mcf_m_tile_channel_put_buffer("in");                       // make data available to workers

  mcf_m_tile_channel_get_buffer("out");                      // wait for results

  mcf_m_tile_channel_disconnect("in");
  mcf_m_tile_channel_disconnect("out");
  mcf_m_tile_channel_destroy("in");
  mcf_m_tile_channel_destroy("out");

  mcf_m_team_wait();                                         // wait for each team member's task to exit
  mcf_m_net_remove_task();
  mcf_m_net_destroy();                                       // free memory associated with MCF network
}
```

3x3 Image Filter – Manager Program

```
main(int argc, char **argv)
{
  mcf_m_net_create();                        // specify number of workers (SPEs)
  mcf_m_net_initialize();                    // launch MCF kernel on workers

  mcf_m_net_add_task_by_embedded_name();     // load SPE executable to XDR
  mcf_m_team_run_task();                     // run task on each worker

  mcf_m_tile_distribution_create_2d("in");   // specify XDR buffer size & tile size
  mcf_m_tile_distribution_set_assignment_overlap("in");  // specify overlap between partitions
  mcf_m_tile_distribution_create_2d("out");

  mcf_m_tile_channel_create("in");           // source channel
  mcf_m_tile_channel_create("out");          // destination channel
  mcf_m_tile_channel_connect("in");
  mcf_m_tile_channel_connect("out");

  mcf_m_tile_channel_get_buffer("in");       // get XDR address of buffer to fill
  // fill input image here
  mcf_m_tile_channel_put_buffer("in");       // make data available to workers

  mcf_m_tile_channel_get_buffer("out");      // wait for results

  mcf_m_tile_channel_disconnect("in");
  mcf_m_tile_channel_disconnect("out");
  mcf_m_tile_channel_destroy("in");
  mcf_m_tile_channel_destroy("out");

  mcf_m_team_wait();                         // wait for each team member's task to exit
  mcf_m_net_remove_task();
  mcf_m_net_destroy();                       // free memory associated with MCF network
}
```

3x3 Image Filter – Manager Program

Mercury Computer Systems, Inc. — Challenges Drive Innovation™

```
main(int argc, char **argv)
{
  mcf_m_net_create();                                          // specify number of workers (SPEs)
  mcf_m_net_initialize();                                      // launch MCF kernel on workers

  mcf_m_net_add_task_by_embedded_name();                       // load SPE executable to XDR
  mcf_m_team_run_task();                                       // run task on each worker

  mcf_m_tile_distribution_create_2d(“in”);                     // specify XDR buffer size & tile size
  mcf_m_tile_distribution_set_assignment_overlap(“in”);        // specify overlap between partitions
  mcf_m_tile_distribution_create_2d(“out”);

  mcf_m_tile_channel_create(“in”);                             // source channel
  mcf_m_tile_channel_create(“out”);                            // destination channel
  mcf_m_tile_channel_connect(“in”);
  mcf_m_tile_channel_connect(“out”);

  mcf_m_tile_channel_get_buffer(“in”);                         // get XDR address of buffer to fill
  /* fill input image here */
  mcf_m_tile_channel_put_buffer(“in”);                         // make data available to workers

  mcf_m_tile_channel_get_buffer(“out”);                        // wait for results

  mcf_m_tile_channel_disconnect(“in”);
  mcf_m_tile_channel_disconnect(“out”);
  mcf_m_tile_channel_destroy(“in”);
  mcf_m_tile_channel_destroy(“out”);

  mcf_m_team_wait();                                           // wait for each team member's task to exit
  mcf_m_net_remove_task();
  mcf_m_net_destroy();                                         // free memory associated with MCF network
}
```

3x3 Image Filter – Manager Program

```
main(int argc, char **argv)
{
  mcf_m_net_create();                                        // specify number of workers (SPEs)
  mcf_m_net_initialize();                                    // launch MCF kernel on workers

  mcf_m_net_add_task_by_embedded_name();                     // load SPE executable to XDR
  mcf_m_team_run_task();                                     // run task on each worker

  mcf_m_tile_distribution_create_2d("in");                   // specify XDR buffer size & tile size
  mcf_m_tile_distribution_set_assignment_overlap("in");      // specify overlap between partitions
  mcf_m_tile_distribution_create_2d("out");

  mcf_m_tile_channel_create("in");                           // source channel
  mcf_m_tile_channel_create("out");                          // destination channel
  mcf_m_tile_channel_connect("in");
  mcf_m_tile_channel_connect("out");

  mcf_m_tile_channel_get_buffer("in");                       // get XDR address of buffer to fill
  // fill input image here
  mcf_m_tile_channel_put_buffer("in");                       // make data available to workers

  mcf_m_tile_channel_get_buffer("out");                      // wait for results

  mcf_m_tile_channel_disconnect("in");
  mcf_m_tile_channel_disconnect("out");
  mcf_m_tile_channel_destroy("in");
  mcf_m_tile_channel_destroy("out");

  mcf_m_team_wait();                                         // wait for each team member's task to exit
  mcf_m_net_remove_task();
  mcf_m_net_destroy();                                       // free memory associated with MCF network
}
```

3x3 Image Filter – Manager Program

```
main(int argc, char **argv)
{
  mcf_m_net_create();                                    // specify number of workers (SPEs)
  mcf_m_net_initialize();                                // launch MCF kernel on workers

  mcf_m_net_add_task_by_embedded_name();                 // load SPE executable to XDR
  mcf_m_team_run_task();                                 // run task on each worker

  mcf_m_tile_distribution_create_2d(“in”);               // specify XDR buffer size & tile size
  mcf_m_tile_distribution_set_assignment_overlap(“in”);  // specify overlap between partitions
  mcf_m_tile_distribution_create_2d(“out”);

  mcf_m_tile_channel_create(“in”);                       // source channel
  mcf_m_tile_channel_create(“out”);                      // destination channel
  mcf_m_tile_channel_connect(“in”);
  mcf_m_tile_channel_connect(“out”);

  mcf_m_tile_channel_get_buffer(“in”);                   // get XDR address of buffer to fill
  // fill input image here
  mcf_m_tile_channel_put_buffer(“in”);                   // make data available to workers

  mcf_m_tile_channel_get_buffer(“out”);                  // wait for results

  mcf_m_tile_channel_disconnect(“in”);
  mcf_m_tile_channel_disconnect(“out”);
  mcf_m_tile_channel_destroy(“in”);
  mcf_m_tile_channel_destroy(“out”);

  mcf_m_team_wait();                                     // wait for each team member's task to exit
  mcf_m_net_remove_task();
  mcf_m_net_destroy();                                   // free memory associated with MCF network
}
```

3x3 Image Filter – Worker Program

```
mcf_w_main (int n_bytes, void * p_arg_ls)
{
  mcf_w_tile_channel_create(“in”);
  mcf_w_tile_channel_create(“out”);
  mcf_w_tile_channel_connect(“in”);
  mcf_w_tile_channel_connect(“out”);

  mcf_w_tile_channel_get_buffer(&in[0]);        // get first two rows
  mcf_w_tile_channel_get_buffer(&in[1]);
  while ( mcf_w_tile_channel_is_not_end_of_channel(“in”) )
  {
     mcf_w_tile_channel_get_buffer(&in[2]);     // get third row
     mcf_w_tile_channel_get_buffer(&out);       // get an output buffer
     f3x3();
     mcf_w_tile_channel_put_buffer(in[0]);      // put “empty” buffer back into channel
     mcf_w_tile_channel_put_buffer(out);        // start moving results back to XDR
     in[0]=in[1];
     in[1]=in[2];
  }
  mcf_w_tile_channel_disconnect(“in”);
  mcf_w_tile_channel_disconnect(“out”);
  mcf_w_tile_channel_destroy(“in”);
  mcf_w_tile_channel_destroy(“out”);
}
```

3x3 Image Filter – Worker Program

```
mcf_w_main (int n_bytes, void * p_arg_ls)
{
  mcf_w_tile_channel_create(“in”);
  mcf_w_tile_channel_create(“out”);
  mcf_w_tile_channel_connect(“in”);
  mcf_w_tile_channel_connect(“out”);

  mcf_w_tile_channel_get_buffer(&in[0]);          // get first two rows
  mcf_w_tile_channel_get_buffer(&in[1]);
  while ( mcf_w_tile_channel_is_not_end_of_channel(“in”) )
  {
     mcf_w_tile_channel_get_buffer(&in[2]);       // get third row
     mcf_w_tile_channel_get_buffer(&out);         // get an output buffer
     f3x3();
     mcf_w_tile_channel_put_buffer(in[0]);        // put “empty” buffer back into channel
     mcf_w_tile_channel_put_buffer(out);          // start moving results back to XDR
     in[0]=in[1];
     in[1]=in[2];
  }
  mcf_w_tile_channel_disconnect(“in”);
  mcf_w_tile_channel_disconnect(“out”);
  mcf_w_tile_channel_destroy(“in”);
  mcf_w_tile_channel_destroy(“out”);
}
```

3x3 Image Filter – Worker Program

```
mcf_w_main (int n_bytes, void * p_arg_ls)
{
  mcf_w_tile_channel_create(“in”);
  mcf_w_tile_channel_create(“out”);
  mcf_w_tile_channel_connect(“in”);
  mcf_w_tile_channel_connect(“out”);

  mcf_w_tile_channel_get_buffer(&in[0]);       // get first two rows
  mcf_w_tile_channel_get_buffer(&in[1]);
  while ( mcf_w_tile_channel_is_not_end_of_channel(“in”) )
  {
     mcf_w_tile_channel_get_buffer(&in[2]);    // get third row
     mcf_w_tile_channel_get_buffer(&out);      // get an output buffer
     f3x3();
     mcf_w_tile_channel_put_buffer(in[0]);     // put “empty” buffer back into channel
     mcf_w_tile_channel_put_buffer(out);       // start moving results back to XDR
     in[0]=in[1];
     in[1]=in[2];
  }
  mcf_w_tile_channel_disconnect(“in”);
  mcf_w_tile_channel_disconnect(“out”);
  mcf_w_tile_channel_destroy(“in”);
  mcf_w_tile_channel_destroy(“out”);
}
```

3x3 Image Filter – Worker Program

```
mcf_w_main (int n_bytes, void * p_arg_ls)
{
  mcf_w_tile_channel_create(“in”);
  mcf_w_tile_channel_create(“out”);
  mcf_w_tile_channel_connect(“in”);
  mcf_w_tile_channel_connect(“out”);

  mcf_w_tile_channel_get_buffer(&in[0]);        // get first two rows
  mcf_w_tile_channel_get_buffer(&in[1]);
  while ( mcf_w_tile_channel_is_not_end_of_channel(“in”) )
  {
     mcf_w_tile_channel_get_buffer(&in[2]);     // get third row
     mcf_w_tile_channel_get_buffer(&out);       // get an output buffer
     f3x3();
     mcf_w_tile_channel_put_buffer(in[0]);      // put “empty” buffer back into channel
     mcf_w_tile_channel_put_buffer(out);        // start moving results back to XDR
     in[0]=in[1];
     in[1]=in[2];
  }
  mcf_w_tile_channel_disconnect(“in”);
  mcf_w_tile_channel_disconnect(“out”);
  mcf_w_tile_channel_destroy(“in”);
  mcf_w_tile_channel_destroy(“out”);
}
```

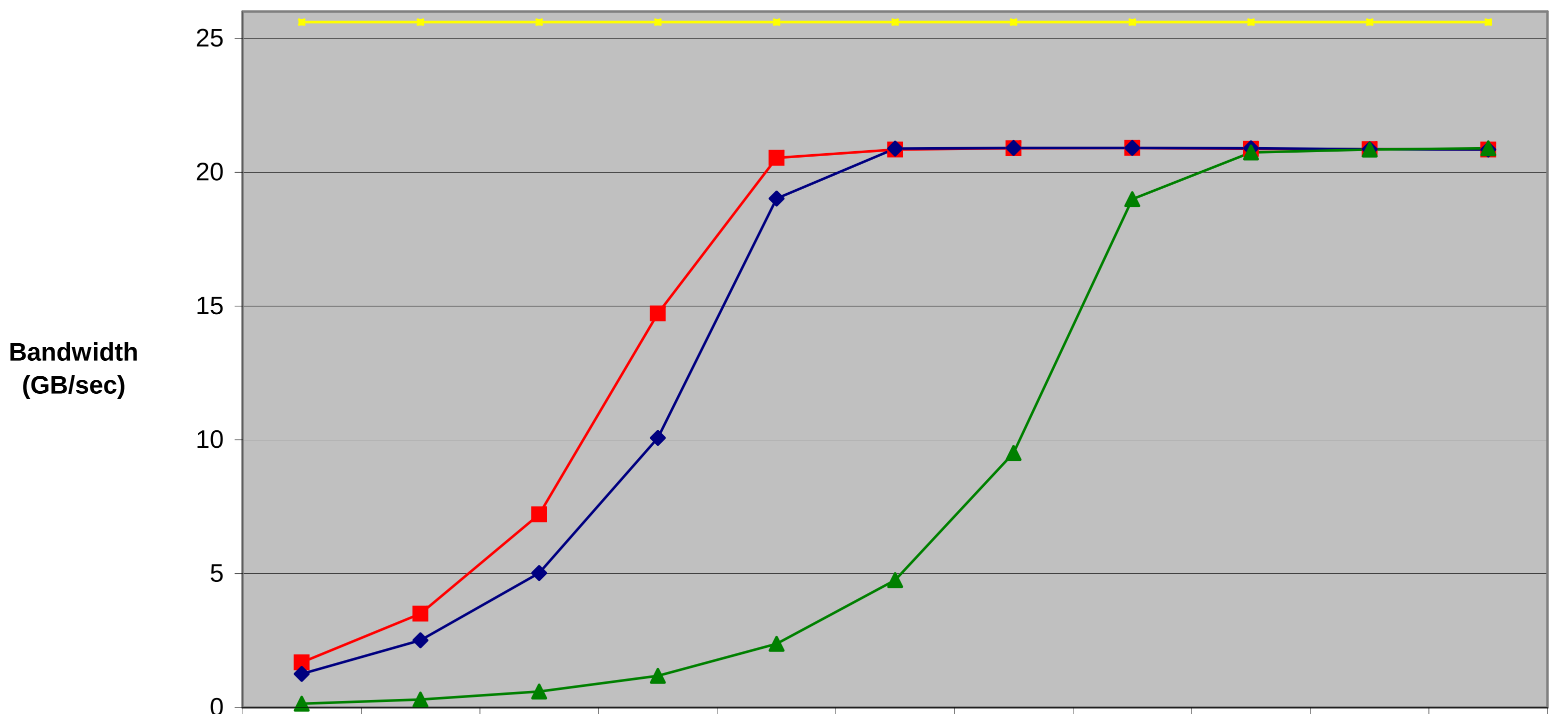
3x3 Image Filter – Worker Program

```
mcf_w_main (int n_bytes, void * p_arg_ls)
{
  mcf_w_tile_channel_create(“in”);
  mcf_w_tile_channel_create(“out”);
  mcf_w_tile_channel_connect(“in”);
  mcf_w_tile_channel_connect(“out”);

  mcf_w_tile_channel_get_buffer(&in[0]);        // get first two rows
  mcf_w_tile_channel_get_buffer(&in[1]);
  while ( mcf_w_tile_channel_is_not_end_of_channel(“in”) )
  {
     mcf_w_tile_channel_get_buffer(&in[2]);     // get third row
     mcf_w_tile_channel_get_buffer(&out);       // get an output buffer
     f3x3();
     mcf_w_tile_channel_put_buffer(in[0]);      // put “empty” buffer back into channel
     mcf_w_tile_channel_put_buffer(out);        // start moving results back to XDR
     in[0]=in[1];
     in[1]=in[2];
  }
  mcf_w_tile_channel_disconnect(“in”);
  mcf_w_tile_channel_disconnect(“out”);
  mcf_w_tile_channel_destroy(“in”);
  mcf_w_tile_channel_destroy(“out”);
}
```

MCF Tile Channel vs. MCF & SPE DMA

Computer Systems, Inc.
MERCURY
Challenges Drive Innovation™

XDR-to-LS and LS-to-XDR Bandwidth
double-buffered input and output with synchronization
(8 SPEs @ 3.2 GHz, 64K page size)



	16	32	64	128	256	512	1024	2048	4096	8192	16384
SPE-DMA	1.68	3.50	7.21	14.72	20.53	20.84	20.90	20.90	20.87	20.86	20.85
MCF-DMA	1.25	2.51	5.02	10.07	19.01	20.88	20.90	20.90	20.89	20.86	20.84
MCF-Tile-Channel	0.15	0.30	0.59	1.19	2.38	4.76	9.51	18.99	20.74	20.85	20.89
Theoretical	25.60	25.60	25.60	25.60	25.60	25.60	25.60	25.60	25.60	25.60	25.60

Transfer Size (bytes)

MCF Summary

- **Simplifies development of high performance applications on multi-core processors like Cell**
  - Easy to overlap IO with processing using Tile & Reorg Channels
  - Abstracts asynchronous DMA data movement
- **Preserves limited SPE memory for application code & data**
  - SPE kernel < 5% of local store memory
- **Runs SPE tasks without Linux overhead**
- **Data movement and synchronization features are “built-in” to the network**
  - Barrier & semaphore synchronization
  - Message queues & mailboxes
  - Asynchronous DMA data movement
- **Provides a convenient API to describe how data is organized within XDR and SPE memories**
  - Data Distribution Objects for Tile & Reorg Channels
  - Programmer describes data rather than individual transfers
- **Manager (PPE) handles “outer loop” setup**
- **Derived from existing, proven software technologies**
  - Leveraged PAS’s model for partitioning data across multiple processors

Playstation 3 Software

- **Software Devlelopers Kit**
  - MultiCore Framework (MCF)
  - Trace Analysis Tool & Library (TATL)
  - Available June 26, 2007
  - Purchase at terrasoftsolutions.com or through Mercury sales reps
  - $400
- **Scientific Algorithm Library**
  - Optimized Functions for SPE (and PPE)
  - Vector & Matrix Arithmetic
  - FFTs, Convolutions, Matrix Decomposition, . . .
  - $400

SAL – Optimized for Cell SPEs

- convolution (1d & 2d)
- dot product
- FFT (1d & 2d, real & complex, radix 2 & radix 3)
- comparison (min, max, threshold, clip, >, <, =,. . .)
- LU decomposition
- matrix multiply
- matrix transpose
- mean, mean square, rms
- vector & scalar add, sub, mul, div, multiply accumulate
- sum vector elements, sum squares
- type conversion (char to float, int to float, . . .)
- sine, cosine, natural log, exponential, square root

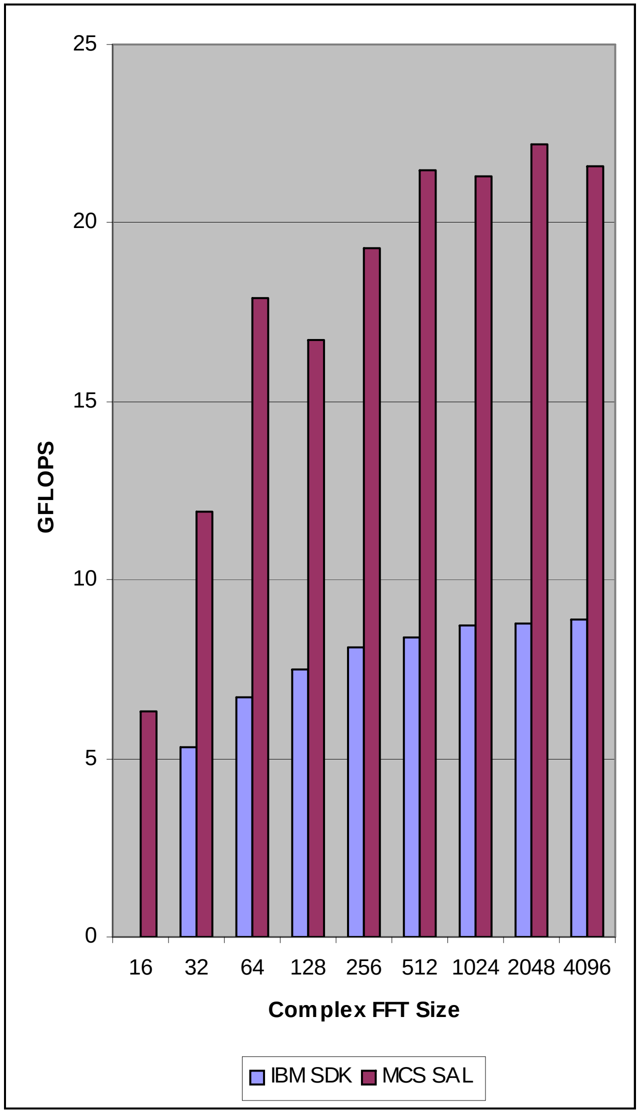
SAL Performance (Single SPE @ 3.2 GHz)

```
• convx    ( 1024 )  : 3-tap FIR filter     =   546 ns (1.7 cpp)
• dotprx   ( 1024 )  : dot product          =   221 ns (0.7 cpp)
• vaddx    ( 1024 )  : vector add           =   285 ns (0.9 cpp)
• vcosx    ( 1024 )  : vector cosine        =  1273 ns (4.0 cpp)
• vdivx    ( 1024 )  : vector divide        =   705 ns (2.2 cpp)
• vexpx    ( 1024 )  : vector exponential   =  1514 ns (4.7 cpp)
• vlnx     ( 1024 )  : vector natural log   =  1254 ns (3.9 cpp)
• vma_x    ( 1024 )  : vector multiply add  =   368 ns (1.2 cpp)
• vmovx    ( 1024 )  : vector move (memcpy) =   210 ns (0.7 cpp)
• vaddx    ( 1024 )  : vector multiply      =   286 ns (0.9 cpp)
• vsinx    ( 1024 )  : vector sine          =  1352 ns (4.2 cpp)
• vsqrtx   ( 1024 )  : vector square root   =   535 ns (1.7 cpp)
• vthrx    ( 1024 )  : vector threshold     =   227 ns (0.7 cpp)

• conv2dx (64x64 )   : 3x3 convolution      =  4749 ns ( 3.7 cpp)
• mat_mulx(64x64 )   : matrix multiply      = 29866 ns (23.3 cpp)
• mtransx (64x64 )   : matrix transpose     =  1436 ns ( 1.1 cpp)
```

SAL FFT vs IBM SDK 2.1 FFT (Single SPE !)

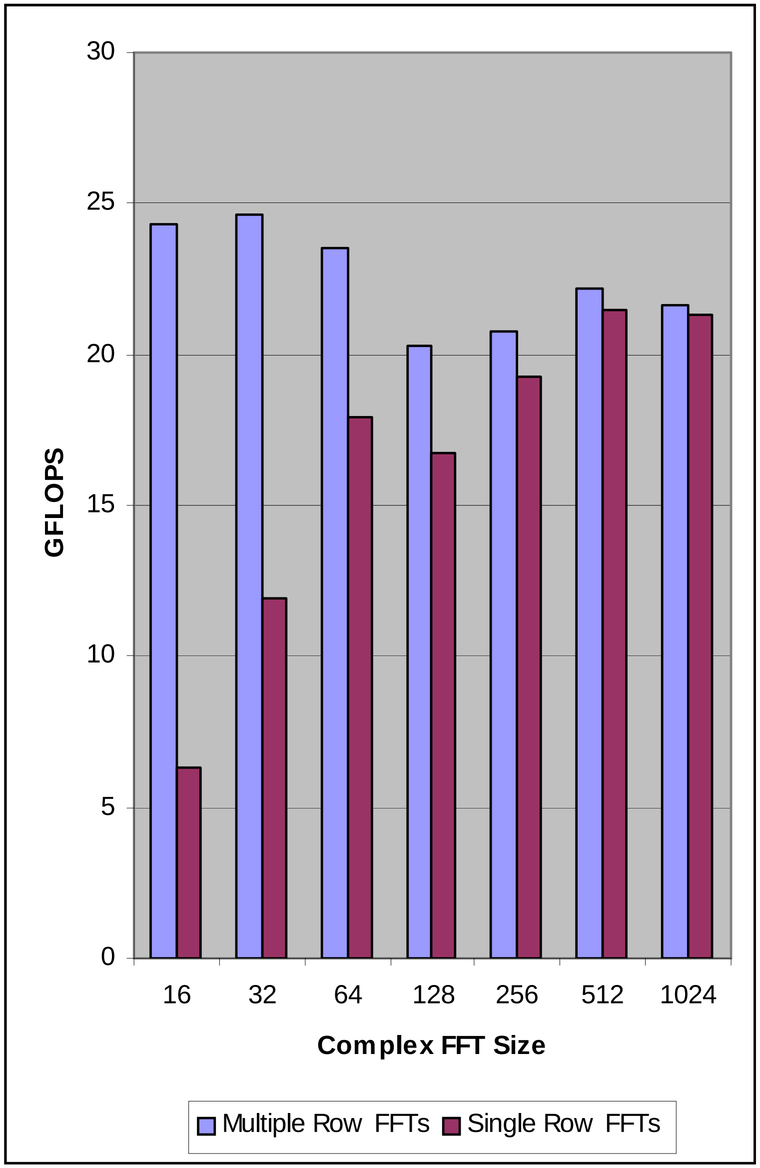
Mercury Computer Systems, Inc.
Challenges Drive Innovation™

- SAL 4096 point complex = 11373 ns (21.6 GFLOPS)
- IBM 4096 point complex = 27520 ns (8.9 GFLOPS)
- SAL 2048 point complex = 5064 ns (22.2 GFLOPS)
- IBM 2048 point complex = 12769 ns (8.8 GFLOPS)
- SAL 1024 point complex = 2398 ns (21.3 GFLOPS)
- IBM 1024 point complex = 5907 ns (8.7 GFLOPS)
- SAL 512 point complex = 1072 ns (21.5 GFLOPS)
- IBM 512 point complex = 2730 ns (8.4 GFLOPS)
- SAL 256 point complex = 530 ns (19.3 GFLOPS)
- IBM 256 point complex = 1267 ns (8.1 GFLOPS)
- SAL 128 point complex = 267 ns (16.7 GFLOPS)
- IBM 128 point complex = 594 ns (7.5 GFLOPS)
- SAL 64 point complex = 107 ns (17.9 GFLOPS)
- IBM 64 point complex = 284 ns (6.7 GFLOPS)
- SAL 32 point complex = 67 ns (11.9 GFLOPS)
- IBM 32 point complex = 152 ns (5.3 GFLOPS)
- SAL 16 point complex = 50 ns (6.3 GFLOPS)



SAL Multiple Row FFTs

```
  4 rows of 1024-point complex = 9502 ns (21.6 GFLOPS)

  4 rows of  512-point complex = 4203 ns (21.9 GFLOPS)
  8 rows of  512-point complex = 8316 ns (22.2 GFLOPS)

  4 rows of  256-point complex = 2029 ns (20.2 GFLOPS)
  8 rows of  256-point complex = 3973 ns (20.6 GFLOPS)
 16 rows of  256-point complex = 7860 ns (20.8 GFLOPS)

  4 rows of  128-point complex =  959 ns (18.7 GFLOPS)
  8 rows of  128-point complex = 1832 ns (19.6 GFLOPS)
 16 rows of  128-point complex = 3578 ns (20.0 GFLOPS)
 32 rows of  128-point complex = 7068 ns (20.3 GFLOPS)

  4 rows of   64-point complex =  383 ns (20.0 GFLOPS)
  8 rows of   64-point complex =  706 ns (21.7 GFLOPS)
 16 rows of   64-point complex = 1351 ns (22.7 GFLOPS)
 32 rows of   64-point complex = 2641 ns (23.3 GFLOPS)
 64 rows of   64-point complex = 5221 ns (23.5 GFLOPS)

  4 rows of   32-point complex =  201 ns (15.9 GFLOPS)
  8 rows of   32-point complex =  328 ns (19.5 GFLOPS)
 16 rows of   32-point complex =  583 ns (21.9 GFLOPS)
 32 rows of   32-point complex = 1094 ns (23.4 GFLOPS)
 64 rows of   32-point complex = 2113 ns (24.2 GFLOPS)
128 rows of   32-point complex = 4154 ns (24.6 GFLOPS)

  4 rows of   16-point complex =  134 ns ( 9.5 GFLOPS)
  8 rows of   16-point complex =  186 ns (13.7 GFLOPS)
 16 rows of   16-point complex =  289 ns (17.7 GFLOPS)
 32 rows of   16-point complex =  493 ns (20.7 GFLOPS)
 64 rows of   16-point complex =  903 ns (22.7 GFLOPS)
128 rows of   16-point complex = 1723 ns (23.8 GFLOPS)
256 rows of   16-point complex = 3364 ns (24.3 GFLOPS)
```



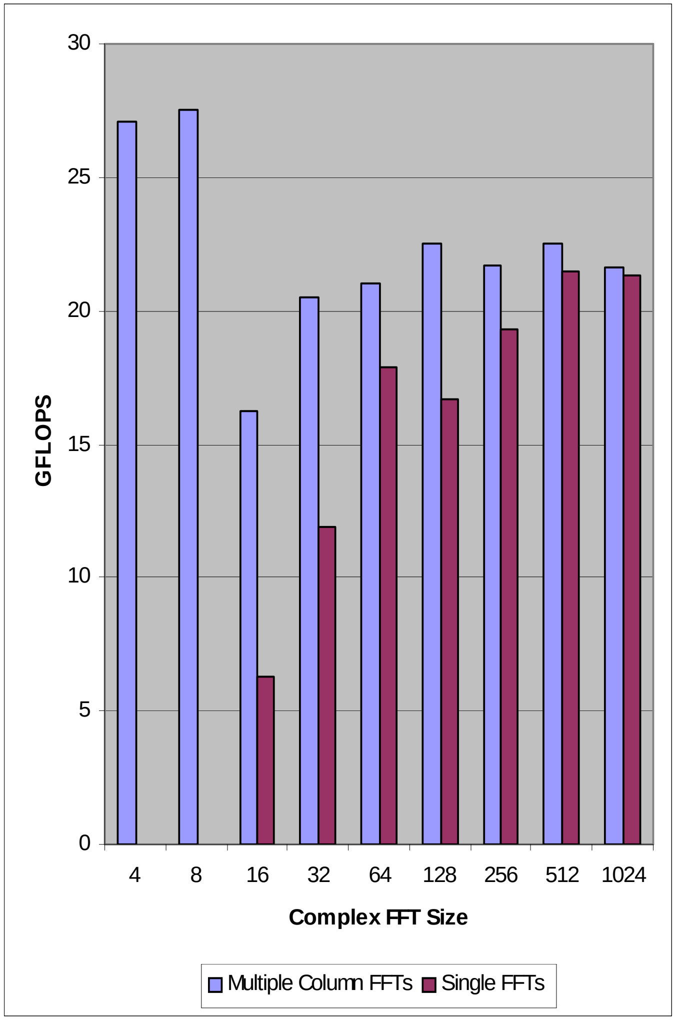
SAL Multiple Column FFTs

```
  4 cols of 1024-point complex = 9492 ns (21.6 GFLOPS)

  4 cols of  512-point complex = 4140 ns (22.3 GFLOPS)
  8 cols of  512-point complex = 8206 ns (22.5 GFLOPS)

  4 cols of  256-point complex = 1943 ns (21.1 GFLOPS)
  8 cols of  256-point complex = 3816 ns (21.5 GFLOPS)
 16 cols of  256-point complex = 7563 ns (21.7 GFLOPS)

  4 cols of  128-point complex =  861 ns (20.8 GFLOPS)
  8 cols of  128-point complex = 1646 ns (21.8 GFLOPS)
 16 cols of  128-point complex = 3219 ns (22.3 GFLOPS)
 32 cols of  128-point complex = 6362 ns (22.5 GFLOPS)

  4 cols of   64-point complex =  430 ns (17.8 GFLOPS)
  8 cols of   64-point complex =  792 ns (19.4 GFLOPS)
 16 cols of   64-point complex = 1516 ns (20.3 GFLOPS)
 32 cols of   64-point complex = 2962 ns (20.7 GFLOPS)
 64 cols of   64-point complex = 5855 ns (21.0 GFLOPS)

  4 cols of   32-point complex =  229 ns (14.0 GFLOPS)
  8 cols of   32-point complex =  383 ns (16.7 GFLOPS)
 16 cols of   32-point complex =  691 ns (18.5 GFLOPS)
 32 cols of   32-point complex = 1307 ns (19.6 GFLOPS)
 64 cols of   32-point complex = 2540 ns (20.2 GFLOPS)
128 cols of   32-point complex = 5005 ns (20.5 GFLOPS)
```

SAL Multiple Column FFTs (continued)

Computer Systems, Inc. MERCURY
Challenges Drive Innovation™

```
   4 cols of 16-point complex =  147 ns ( 8.7 GFLOPS)
   8 cols of 16-point complex =  225 ns (11.3 GFLOPS)
  16 cols of 16-point complex =  381 ns (13.4 GFLOPS)
  32 cols of 16-point complex =  692 ns (14.8 GFLOPS)
  64 cols of 16-point complex = 1315 ns (15.6 GFLOPS)
 128 cols of 16-point complex = 2560 ns (16.0 GFLOPS)
 256 cols of 16-point complex = 5049 ns (16.2 GFLOPS)

   4 cols of  8-point complex =   93 ns ( 5.1 GFLOPS)
   8 cols of  8-point complex =  110 ns ( 8.7 GFLOPS)
  16 cols of  8-point complex =  144 ns (13.3 GFLOPS)
  32 cols of  8-point complex =  211 ns (18.1 GFLOPS)
  64 cols of  8-point complex =  347 ns (22.1 GFLOPS)
 128 cols of  8-point complex =  617 ns (24.9 GFLOPS)
 256 cols of  8-point complex = 1157 ns (26.5 GFLOPS)
 512 cols of  8-point complex = 2237 ns (27.5 GFLOPS)

   4 cols of  4-point complex =   77 ns ( 2.1 GFLOPS)
   8 cols of  4-point complex =   83 ns ( 3.9 GFLOPS)
  16 cols of  4-point complex =   94 ns ( 6.8 GFLOPS)
  32 cols of  4-point complex =  116 ns (11.0 GFLOPS)
  64 cols of  4-point complex =  161 ns (15.9 GFLOPS)
 128 cols of  4-point complex =  251 ns (20.4 GFLOPS)
 256 cols of  4-point complex =  431 ns (23.7 GFLOPS)
 512 cols of  4-point complex =  791 ns (25.9 GFLOPS)
1024 cols of  4-point complex = 1511 ns (27.1 GFLOPS)
```



SAL Real FFTs (Single SPE !)

- SAL 4096 point real = 6318 ns (19.4 GFLOPS)
- SAL 2048 point real = 3052 ns (18.5 GFLOPS)
- SAL 1024 point real = 1426 ns (18.0 GFLOPS)
- SAL 512 point real = 733 ns (15.7 GFLOPS)
- SAL 256 point real = 395 ns (13.0 GFLOPS)
- SAL 128 point real = 199 ns (11.3 GFLOPS)
- SAL 64 point real = 140 ns (6.8 GFLOPS)
- SAL 32 point real = 114 ns (3.5 GFLOPS)

For More Information

Web: www.mc.com/cell

E-mail: webinfo@mc.com

+1 (866) 627-6951 (US)
+1 (978) 967-1401 (International)