FPGA accelerated FDTD Simulation on the Cray XD1 using Impulse C

Peter Messmer*, David Smithe, Paul Schoessow
Tech-X Corporation
messmer@txcorp.com

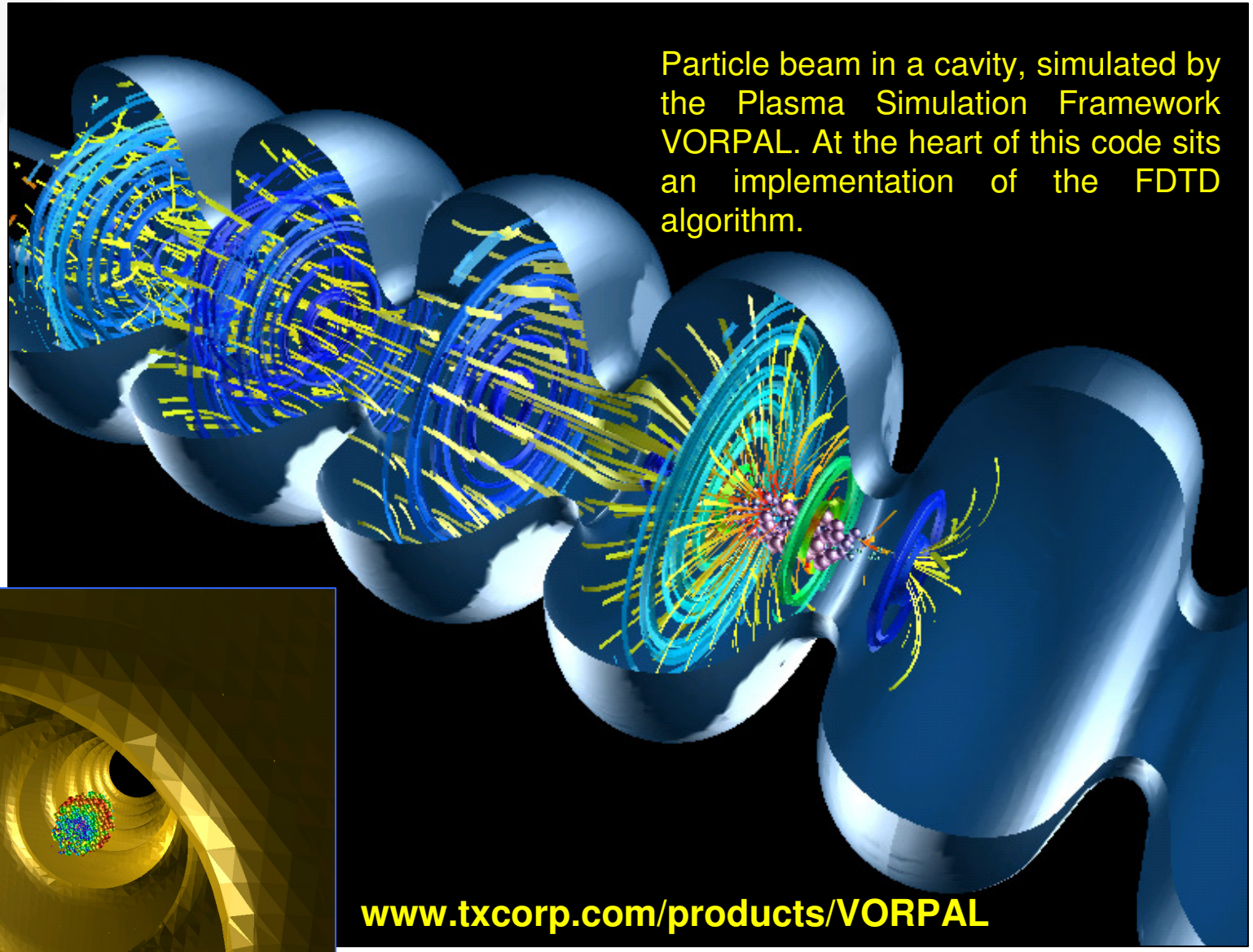
Ralph Bodenner
Impulse Accelerated Technologies

CUG 2006, Lugano, Switzerland
May 9, 2006

Outline

- The algorithm
  - FDTD and its applications
- The tools used
  - XD1 system overview, FPGA
  - Impulse C
- Porting FDTD to FPGA
  - Pure software based optimizations
  - Initial port to Impulse C/FPGA
  - Further optimizations
- Summary and Conclusion

Complex electromagnetic phenomena require simulations



Time-dependent solution of Maxwell's equations

- Initial condition:
  - Solution to Poisson's equation known: $\nabla \mathbf{E} = -\rho$
  - No magnetic monopoles: $\nabla \mathbf{B} = 0$
- Evolve dynamic Maxwell's equations in time:

$$\frac{\partial \mathbf{E}}{\partial t} = \nabla \times \mathbf{B} - \mathbf{J} \qquad \frac{\partial \mathbf{B}}{\partial t} = -\frac{1}{c^2} \nabla \times \mathbf{E}$$

- Current satisfies continuity equation: $\nabla \cdot \mathbf{J} = -\frac{\partial \rho}{\partial t}$

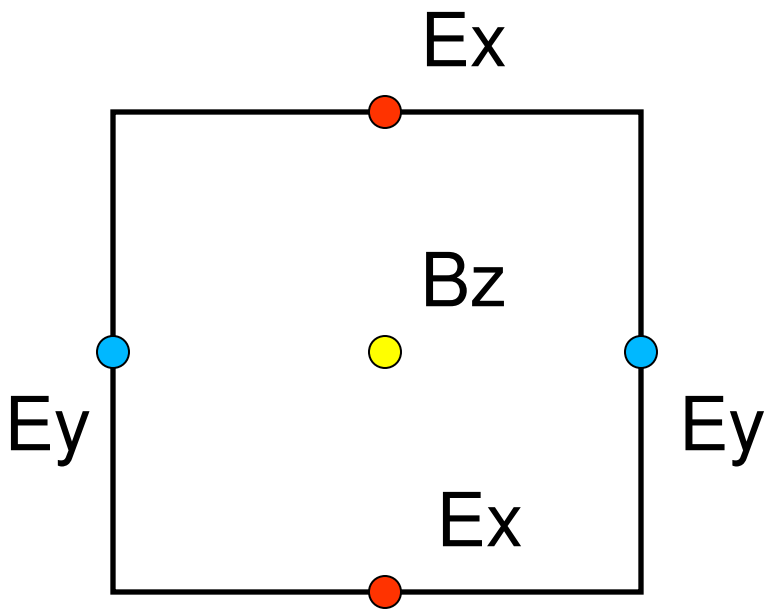
For this project: no charges/currents!

FDTD / Yee Grid

- Discretization of Maxwell's equations
  - Finite differences for curl and time derivative, e.g.

$$B_z(t+1) = B_z(t) + dt\left((E_x(y+1) - E_x) / dy - (E_y(x+1) - E_y) / dx \right)$$

- Yee grid
  - Arrangement of EM field components
  - Spatially centred finite differences
    - Second order in space
- Leap-frog of Ampere/Faraday
    - Second order in time



FPGAs have potential to accelerate FDTD

- Large simulations require >10^9 cells per processor and thousands of time-steps
- Straight forward CPU based implementation:

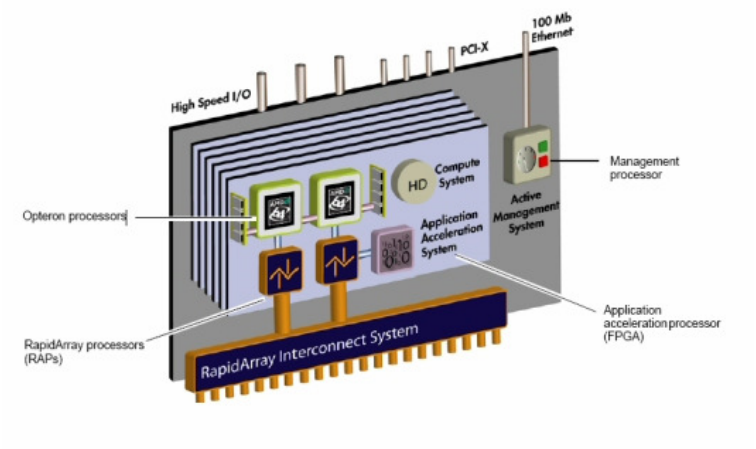
  ~ $5 \cdot 10^6$ 3D cell updates / second

$\Rightarrow$ need to accelerate cell updates

- High degree of parallelism in FDTD
  - Cell half-updates independent of each other
- Various groups: FDTD on FPGA with custom made pipelines
  - E.g. Culley et al. U. Cincinnati
- Today: High-level tools available for non-FPGA experts
  - FDTD simple enough to experiment with FPGAs and new tools
- Have access to an FPGA enhanced system
  - Cray XD1

Cray XD1 System

- Cluster of XD1 chassis
  - 2 XD1 chassis
    - 12 nodes total
    - 2 x 2.2 GHz Dual core AMD Opterons



  - 1 Chassis equipped with *Application Accelerator*
    - User programmable FPGA
      - Field-programmable Gate Array, 'configurable matrix of logic'
      - 'Programmable matrix of logic hardware'
      - Xilinx Vitrex-II Pro FPGA
    - Act as a configurable (co-)processor

- FPGA runs at 200MHz (!)
  - Have to exploit parallelism to compete with 2.2 GHz Opteron.
  - "Need more than factor 10 in *parallelism"*

FPGA programming using Impulse C

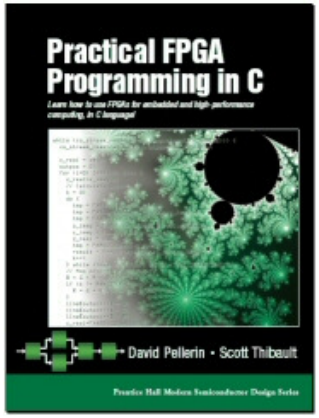
- C-to-hardware/HDL translator
  - Programming model: Application = set of processes
  - Processes communicate via streams, shared memories or signals
  - Processes located on either FPGA or CPU
- Impulse C dialect
  - API for streams, shared memory and signals
  - Functions for specifying location of processes (CPU or FPGA)
  - Directives for tuning (pipelines, loop unrolling, timing)

FPGA programming using Impulse C (cont.)

- Support for Cray XD1
  - Impulse C communication primitives on top of RapidArray Transport IP
  - Generates all necessary hardware and software interfaces
  - Generates makefiles and Xilinx ISE / XST project files

- Current limitations
  - XD1 support currently in beta stage
    - Only streams supported, no shared memory
  - Only limited support for floating-point IP
    - At time of project start not available for Cray XD1

- For more information: www.impulsec.com



Pellerin and Thibault, *Practical FPGA Programming in C,* Prentice Hall, 2005.

The long road from source code to an FPGA enabled application

Create/port source using ImpulseC
- Both for hardware and software processes
- Simulate in pure software

Create VHDL or Verilog from C code
- Simulate the VHDL code

Export application
- VHDL, interfaces, source for software processes
- makefiles/project files

Translate VHDL into a bitstream
- Compile VHDL
- Place&Route

Transfer bitstream and software process source to XD1

Append header to bitstream using Cray tools
Build CPU application

Execute CPU application
- Impulse C wrapper loads bitstream to FPGA

Initial optimization: Rescaling

- At start of project, ImpulseC limited to fix-point arithmetic
  - Floating point support currently being implemented
- Rescaling of variables
  - E -> E dx, B-> B dx/dt
  - Avoids most divisions and multiplications
  - Only integers needed for variables
- Update is reduced to

  Ez += (Bx[y+1] – Bx – By[x+1] – By) / cdx

  … similar for Ex, Ey, Bx, By, Bz
- Benchmark problem: 400 x 200 x 1 cells, 100 timesteps
- Pure software implementation: 0.22s (> 38 G Cells/s)

Software implementation of FDTD algorithm

```
for(int z = 0; z < nz; z++)
  for(int y = 0; y < ny; y++)
    for(int x = 0; x < nx; x++){

      int ind    =  x              + y * nx            + z * nx * ny;
      int ind_px = (x + 1) % nx + y * nx             + z * nx * ny;
      int ind_py =  x       + ((y + 1) % ny) * nx    + z * nx * ny;
      int ind_pz =  x              + y*nx       + (( z + 1) % nz) * nx * ny;

      ex[ind] += (bz[ind_py] - bz[ind] - by[ind_pz] + by[ind]) / cdx4;
      ey[ind] += (bx[ind_pz] - bx[ind] - bz[ind_px] + bz[ind]) / cdx4;
      ez[ind] += (by[ind_px] - by[ind] - bx[ind_py] + bx[ind]) / cdx4;

    }
```

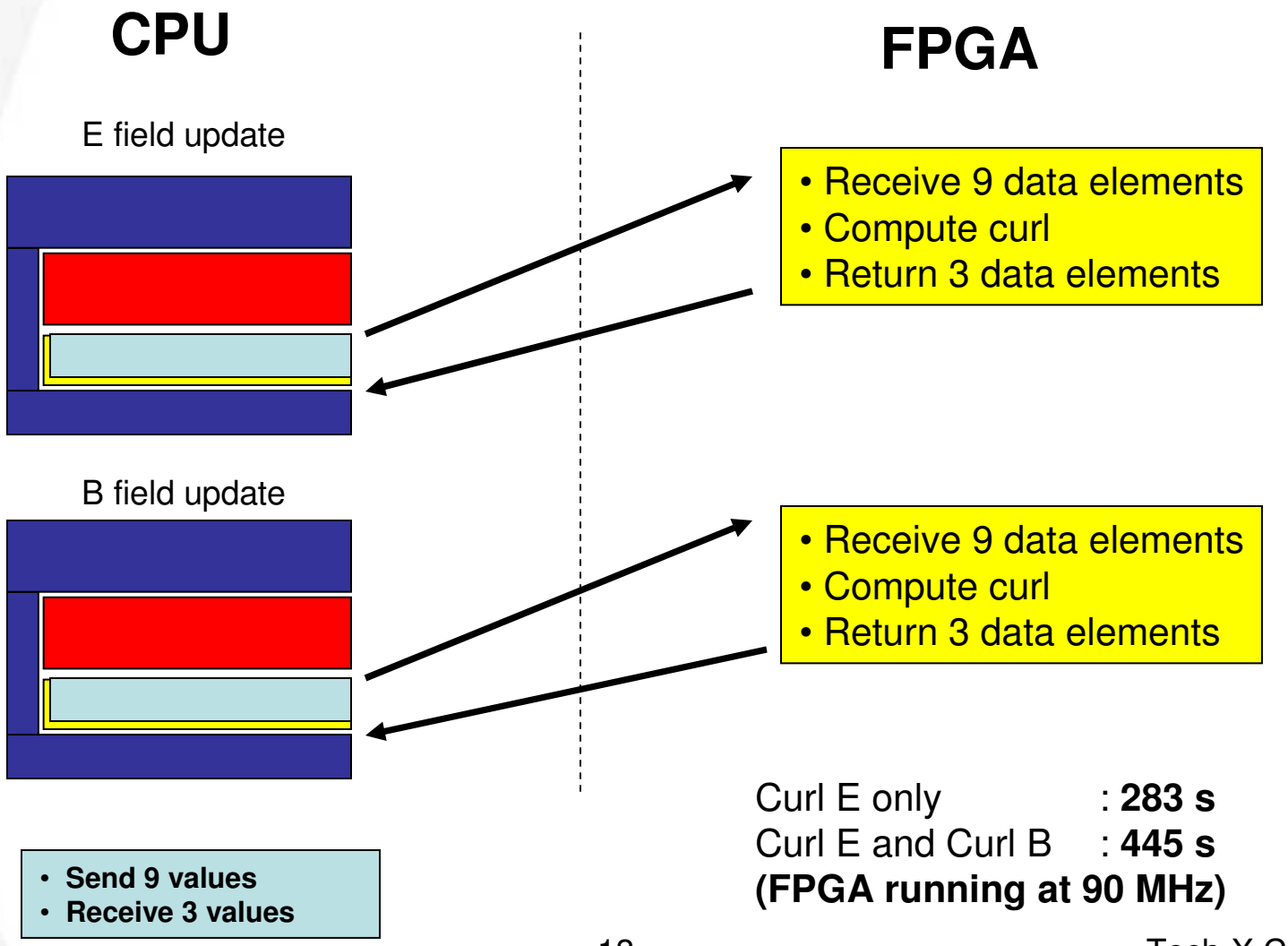
Loop over 3D grid

Index computation

Curl computation

… and similar for the B-field update.

Initial FPGA implementation: Put curl computation onto FPGA



Curl E only : **283 s**
Curl E and Curl B : **445 s**
(FPGA running at 90 MHz)

Some source code…

```
void CurlProcess(co_stream In, co_stream Out) {
   co_int32 bx, bx2, by, by2, bz, bz2;
   ….
   do {
      co_stream_open(In, O_RDONLY, INT_TYPE(32));
       co_stream_open(Out, O_WRONLY, INT_TYPE(32));

       while(!co_stream_eos(In)) {
#pragma CO pipeline
            co_stream_read(In, &bx,   sizeof(co_int32));   bx2 = bx >> 2;
            co_stream_read(In, &by,   sizeof(co_int32));   by2 = by >> 2;
            ….
            ex = bz_py2 - bz2 - by_pz2 + by2;
            co_stream_write(Out, &ex, sizeof(co_int32));
             ….
       }
       co_stream_close(Out);
       co_stream_close(In);
       IF_SIM(break;)  // Only run once during desktop simulation
   } while (1);
}
```

Regular C function, streams as parameters

Opening streams

Directives for tuning

Streams IO

Macros for simulation

Curl on FPGA: Works, but..

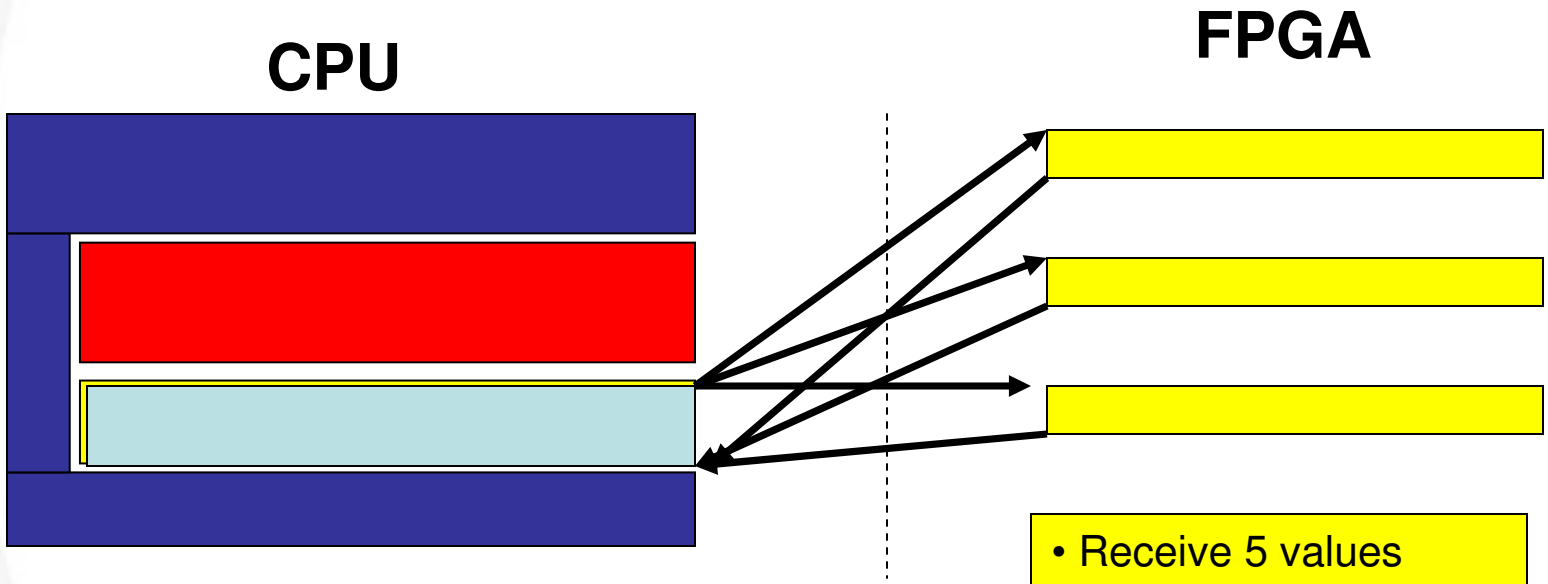
- … low performance

Curl on FPGA : **445 s**
(CPU only: 0.2s)

- Possible causes
  - Large amount of data transfer CPU <-> FPGA
    - 32 bit transfers
  - Very little computation on FPGA
    - No pipelining, no unrolling, large sequential part

Going parallel and optimizing streams

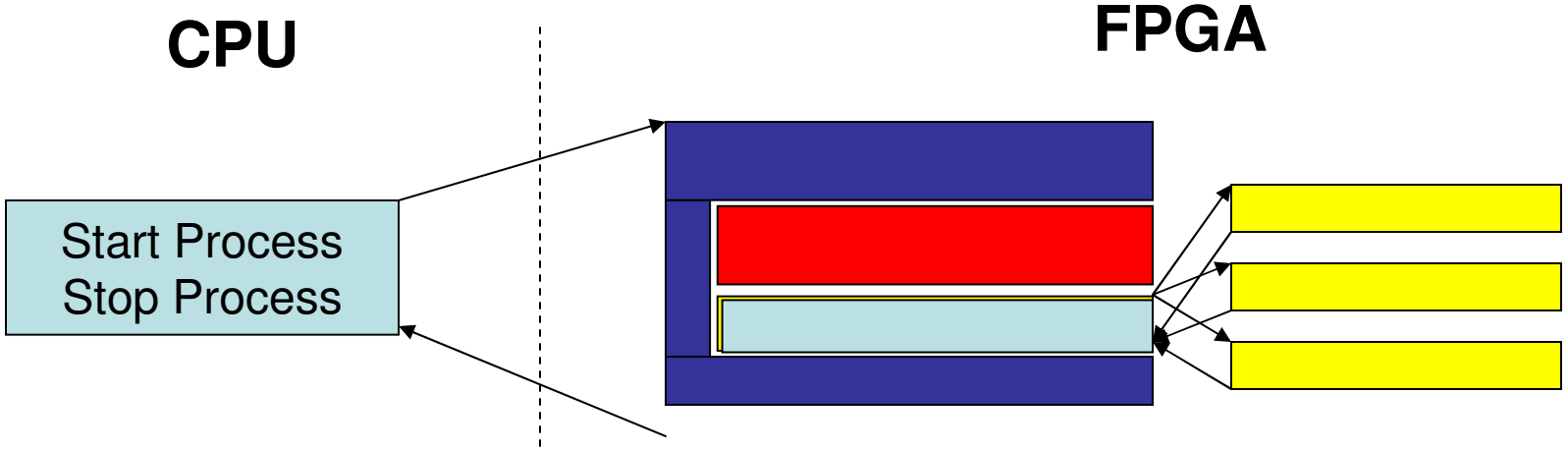
- Three components of curl can be computed in parallel
- Each component a separate, identical process
- Out += ($In_1 - In_2 - In_3 + In_4$)
- Pack 2 x 32 bit words into 64 bit word



3 curl processes : **243 s**
(FPGA running at 140 MHz)

Fighting the data transfer bottle neck

- Communication bottle neck too big!
  - No computation, just data ping-pong: 241 s
  - Plain Opteron implementation: < 0.2s!
- Avoid problem by putting entire application onto FPGA
  - 'Right way' via shared memory, once it's available
  - Grid size limited to FPGA BRAM



FPGA only : **5.9 s**
(FPGA running at 140 MHz)

Full FDTD on FPGA

- Initial implementation : 5.9 s
- Pipelining of curl computation : 5.1 s
- Unrolling curl : 5.0s

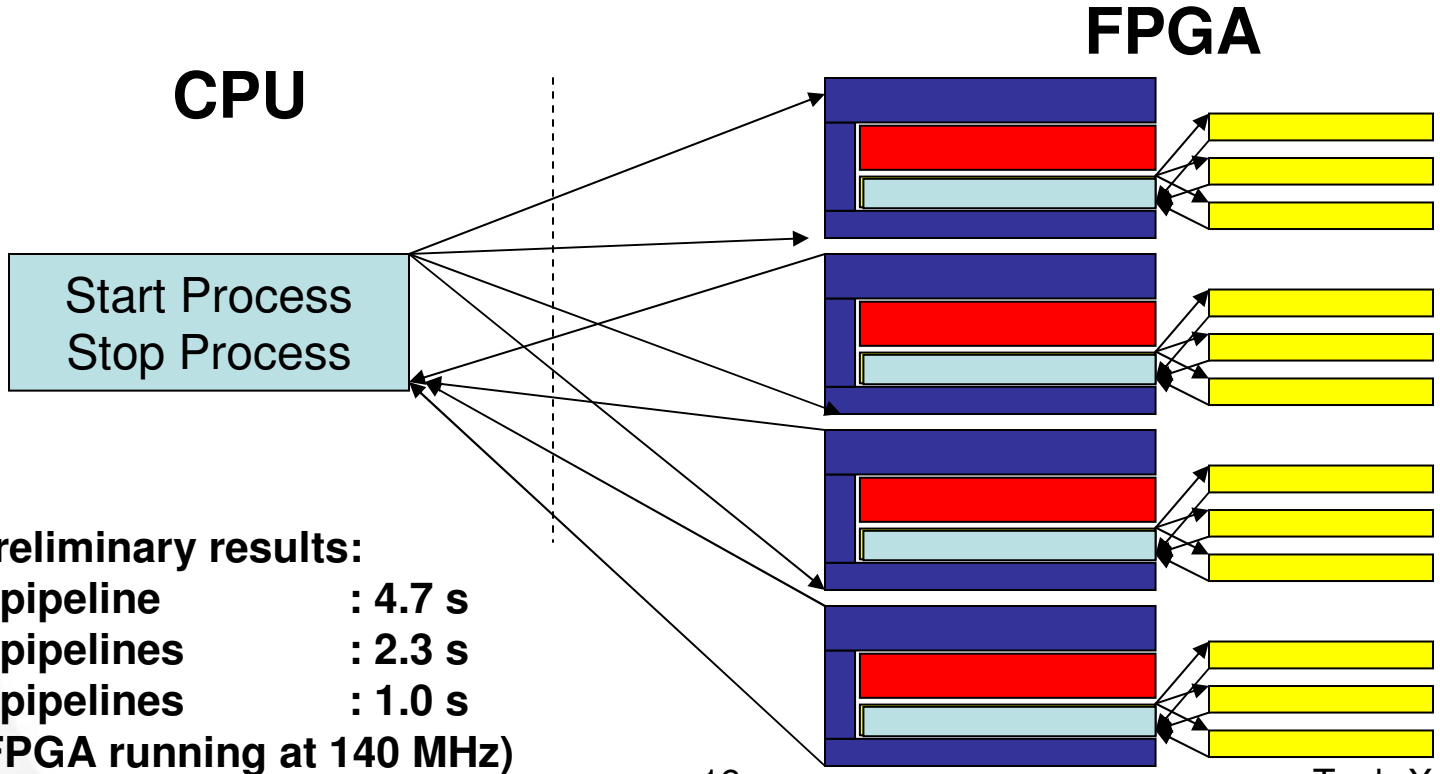
- Only one memory access per clock cycle
  - Splitting E, B array -> Ex, Ey, Ez, Bx, By, Bz array
- Array splitting : 4.7s

- Still factor 10 away from plain software implementation
- Currently main loop pipeline 10 cycles per result, 2 cycles per curl

- Further array splitting (odd/even split)
- System level parallelism

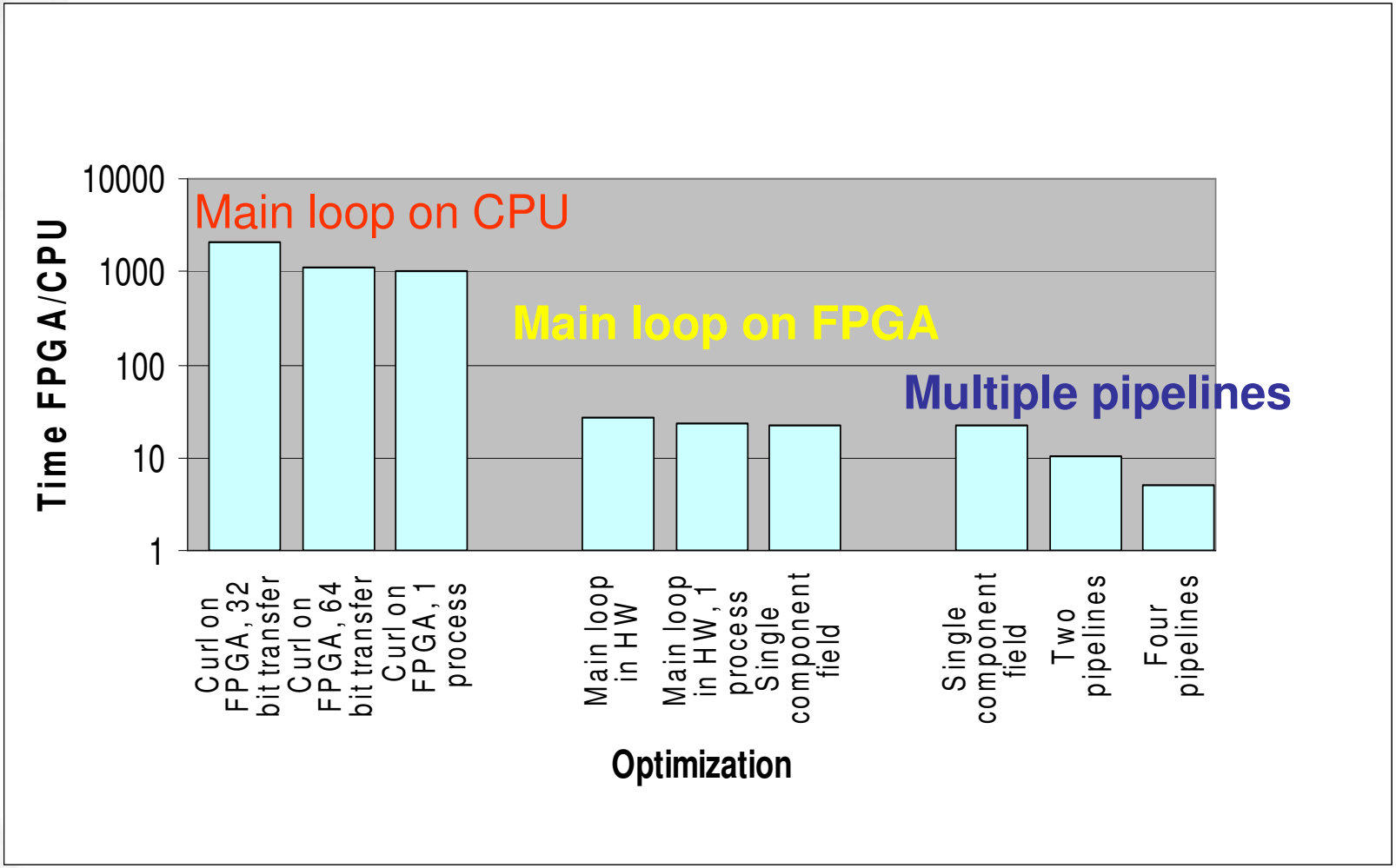
Exploiting System Level Parallelism

- Only about 8 % of FPGA real estate used
- Multiple FDTD pipelines in parallel
  - Domain decomposition



Preliminary results:
1 pipeline : 4.7 s
2 pipelines : 2.3 s
4 pipelines : 1.0 s
(FPGA running at 140 MHz)

Optimizations Summary



Conclusion and Summary

- Optimized an FDTD implementation on AMD Opteron
- Ported it to FPGA
- Experimented with various optimizations
  - Avoiding bus bottleneck, pipelining
  - Multiple concurrent processes, Domain decomposition
- Not quite at the performance of a single CPU, but getting closer
  - Potential is there!

- High-Level tools enable domain scientists to experiment with FPGA
- Cray XD1 system provides ideal platform for these experiments

- FPGA FDTD optimization
  - Getting speedup by combining domain decomposition and pipeline optimization

We would like to thank David Strenski (Cray) and Roy White (Xilinx) for providing access to various tools and resources. Access to the Cray XD1 system was provided through the Cray Marketing Partner Network.