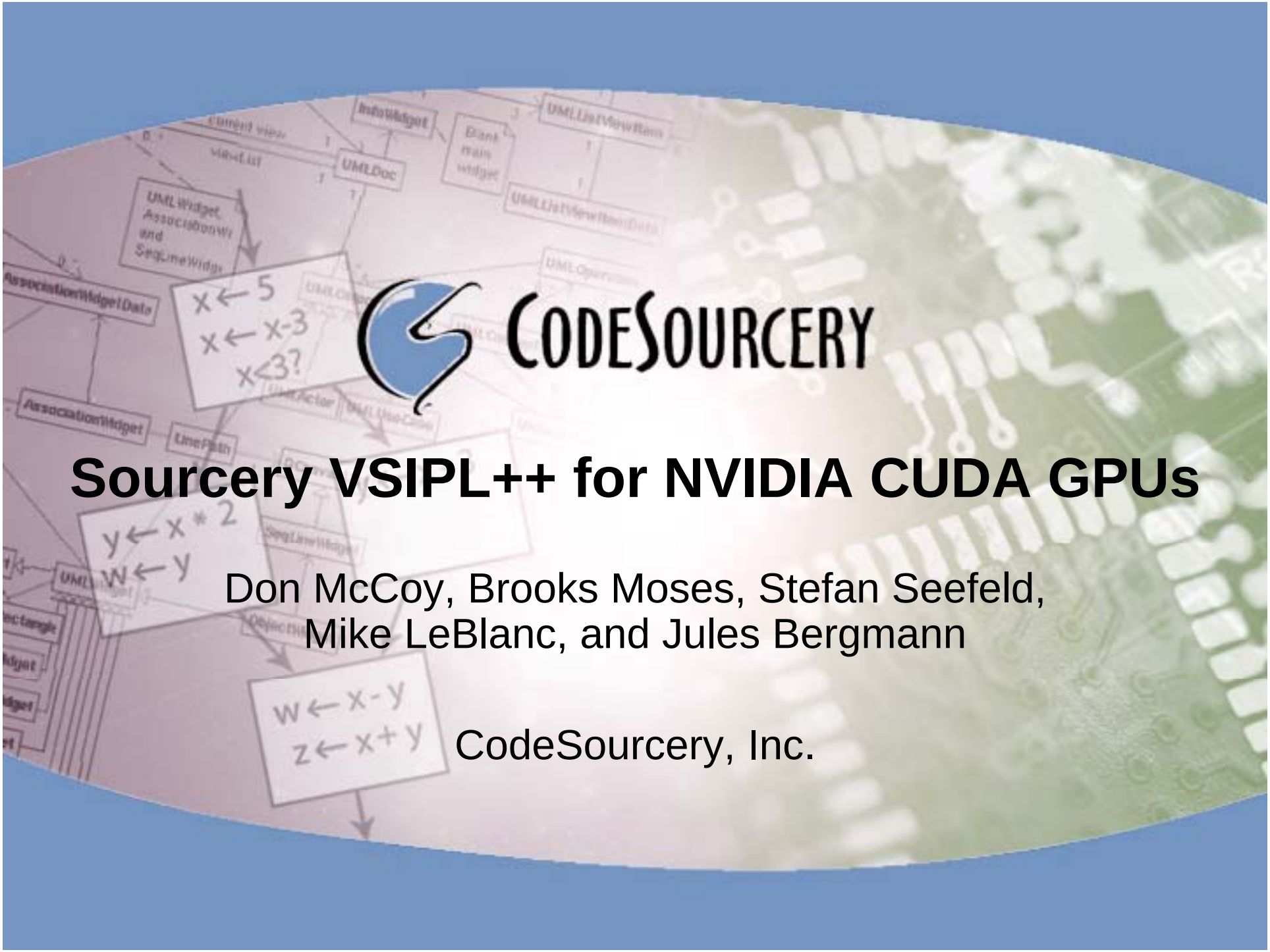


Sourcery VSIPL++ for NVIDIA CUDA GPUs

Don McCoy, Brooks Moses, Stefan Seefeld,
Mike LeBlanc, and Jules Bergmann

CodeSourcery, Inc.

Outline

Framing question: How can we preserve our programming investment and maintain competitive performance, in a world with ever-changing hardware?

- Topics
  - Example synthetic-aperture radar (SSAR) application
  - Sourcery VSIPL++ library for CUDA GPUs
  - Porting the SSAR application to this new target
  - Portability and performance results

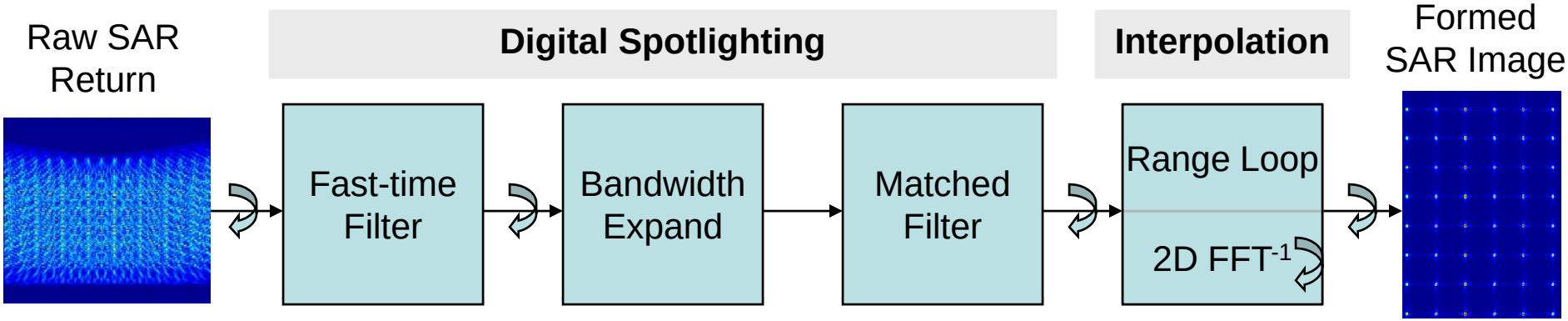
Review of 2008 HPEC Presentation

- Example code: SSCA3 SSAR benchmark
  - Provides a small but realistic example of HPEC code

- Sourcery VSIPL++ implementation
  - Initial target-independent version
  - Optimized for x86 (minimal changes)
  - Optimized for Cell/B.E. (a bit more work)

- Results
  - Productivity (lines of code, difficulty of optimization)
  - Performance (comparison to ref. implementation)

SSCA3 SSAR Benchmark



Major Computations:	FFT mmul	mmul FFT pad FFT^{-1}	FFT mmul	interpolate 2D FFT^{-1} magnitude

Scalable Synthetic SAR Benchmark

- Created by MIT/LL
- Realistic Kernels
- Scalable
- Focus on image formation kernel
- Matlab & C ref impl avail

Challenges

- Non-power of two data sizes (1072 point FFT – radix 67!)
- Polar -> Rectangular interpolation
- 5 corner-turns
- Usual kernels (FFTs, vmul)

Highly Representative Application

Characteristics of VSIPL++ SSAR Implementation

- Most portions use standard VSIPL++ functions
  - Fast Fourier transform (FFT)
  - Vector-matrix multiplication (vmmul)
- Range-loop interpolation implemented in user code
  - Simple by-element implementation (portable)
  - User-kernel implementation (Cell/B.E.)
- Concise, high-level program
  - 203 lines of code in portable VSIPL++
  - 201 additional lines in Cell/B.E. user kernel.

Conclusions from 2008 HPEC presentation

- Productivity
  - Optimized VSIPL++ easier than unoptimized C
  - Baseline version runs well on x86 and Cell/B.E.
  - User kernel greatly improves Cell/B.E. performance with minimal effort.

- Performance
  - Orders of magnitude faster than reference C code
  - Cell/B.E. 5.7x faster than Xeon x86

Conclusions from 2008 HPEC presentation

- Productivity
  -
  -
  -
- Pe
  -
  -

What about the future?

- **Technology refresh?**
- **Portability to future platforms?**
- **What if we need to run this on something like a GPU?**

What about the future, then?

- Porting the SSAR application to a GPU
  - Build a prototype Sourcery VSIPL++ for CUDA.
  - Port the existing SSAR application to use it.
- How hard is that port to do?
- How much code can we reuse?
- What performance do we get?

Characteristics of GPUs

Tesla C1060 GPU:

- 240 multithreaded coprocessor cores
- Cores execute in (partial) lock-step
- 4GB device memory
- Slow device-to-RAM data transfers
- Program in CUDA, OpenCL

Cell/B.E.:

- 8 coprocessor cores
- Cores are completely independent
- Limited local storage
- Fast transfers from RAM to local storage
- Program in C, C++

Very different concepts; low-level code is not portable

Prototype Sourcery VSIPL++ for CUDA

- **Part 1: Selected functions computed on GPU:**
  - Standard VSIPL++ functions:
    - 1-D and 2-D FFT (from CUDAFFT library)
    - FFTM (from CUDAFFT library)
    - Vector dot product (from CUDABLAS library)
    - Vector-matrix elementwise multiplication
    - Complex magnitude
    - Copy, Transpose, FreqSwap
  - Fused operations:
    - Fast convolution
    - FFTM and vector-matrix multiplication

Prototype Sourcery VSIPL++ for CUDA

- **Part 2: Data transfers to/from GPU device memory**
  - Support infrastructure
    - Transfer of data between GPU and RAM
    - Integration of CUDA kernels into library
  - Integration with standard VSIPL++ blocks
    - Data still stored in system RAM
    - Transfers to GPU device memory as needed for computations, and then back to system RAM
    - Completely transparent to user

Everything so far requires no user code changes

Initial CUDA Results

Initial CUDA results			Baseline x86	
Function	Time	Performance	Time	Speedup
Digital Spotlight				
Fast-time filter	0.078 s	3.7 GF/s	0.37 s	4.8
BW expansion	0.171 s	5.4 GF/s	0.47 s	2.8
Matched filter	0.144 s	4.8 GF/s	0.35 s	2.4
Interpolation				
Range loop	1.099 s	0.8 GF/s	1.09 s	-
2D IFFT	0.142 s	6.0 GF/s	0.38 s	2.7
Data Movement	0.215 s	1.8 GB/s	0.45 s	2.1
Overall	1.848 s		3.11 s	1.8

Almost a 2x speedup – but we can do better!

Digital Spotlighting Improvements

- Code here is almost all high-level VSIPL++ functions
- Problem: Computations on GPU, data stored in RAM
  - Each function requires a data round-trip
- Solution: New VSIPL++ Block type: **`Gpu_block`**
  - Moves data between RAM and GPU as needed
  - Stores data where it was last touched
- Requires a simple code change to declarations:

```
typedef Vector<float, Gpu_block>
  real_vector_type;
```

Gpu_block CUDA Results

Initial CUDA results			Baseline x86	
Function	Time	Performance	Time	Speedup
Digital Spotlight				
Fast-time filter	0.078 s	3.7 GF/s	0.37 s	4.8
BW expansion	0.171 s	5.4 GF/s	0.47 s	2.8
Matched filter	0.144 s	4.8 GF/s	0.35 s	2.4

Gpu_block CUDA results			Baseline x86	
Digital Spotlight				
Fast-time filter	0.023 s	12.8 GF/s	0.37 s	16.3
BW expansion	0.059 s	15.7 GF/s	0.47 s	8.0
Matched filter	0.033 s	21.4 GF/s	0.35 s	10.8

Maintaining data on GPU provides 3x-4x additional speedup

Interpolation Improvements

- Range Loop takes most of the computation time
  - Does not reduce to high-level VSIPL++ calls

- As with Cell/B.E., we write a custom “user kernel” to accelerate this on the coprocessor.
  - Sourcery VSIPL++ handles data movement, and provides access to data in GPU device memory.
    - Much simpler than using CUDA directly
  - User kernel only needs to supply computation code
    - ~150 source lines

Optimized CUDA Results

Optimized CUDA			Baseline x86	
Function	Time	Performance	Time	Speedup
Digital Spotlight				
Fast-time filter	0.023 s	12.8 GF/s	0.37 s	16.3
BW expansion	0.059 s	15.7 GF/s	0.47 s	8.0
Matched filter	0.033 s	21.4 GF/s	0.35 s	10.8
Interpolation				
Range loop	0.262 s	3.2 GF/s	1.09 s	4.1
2D IFFT	0.036 s	23.6 GF/s	0.38 s	10.5
Data Movement	0.095 s	4.0 GB/s	0.45 s	4.7
Overall	0.509 s		3.11 s	6.1

Result with everything on the GPU: a 6x speedup.

Conclusions

- Sourcery VSIPL++ for CUDA GPUs
  - Prototype code exists now
  - Contains everything needed for SSAR application

- Porting code to new targets with Sourcery VSIPL++ works with realistic code *in practice.*
  - GPUs are very different from Cell/B.E., but:
  - “50% performance” with zero code changes
  - Much better performance with minimal changes
  - And can easily hook in rewritten key kernels for best performance.