Initial Experiences Programming Xilinx Virtex-6 FPGAs inside Convey's HC-1ex

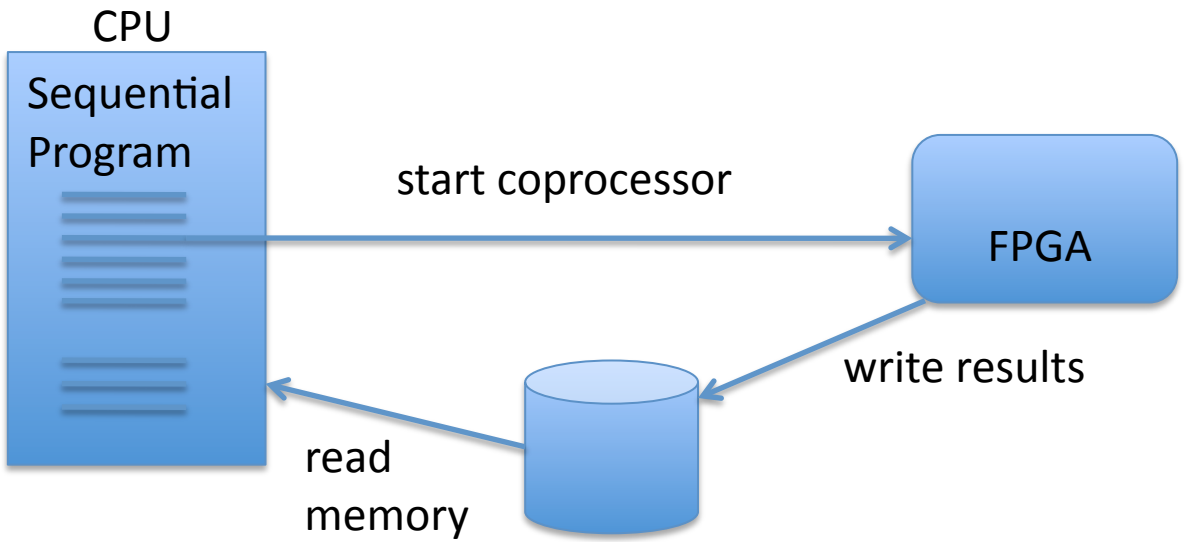
Stephen Bique, Ph.D.
Computer Scientist
Naval Research Laboratory
Washington, DC

In Brief

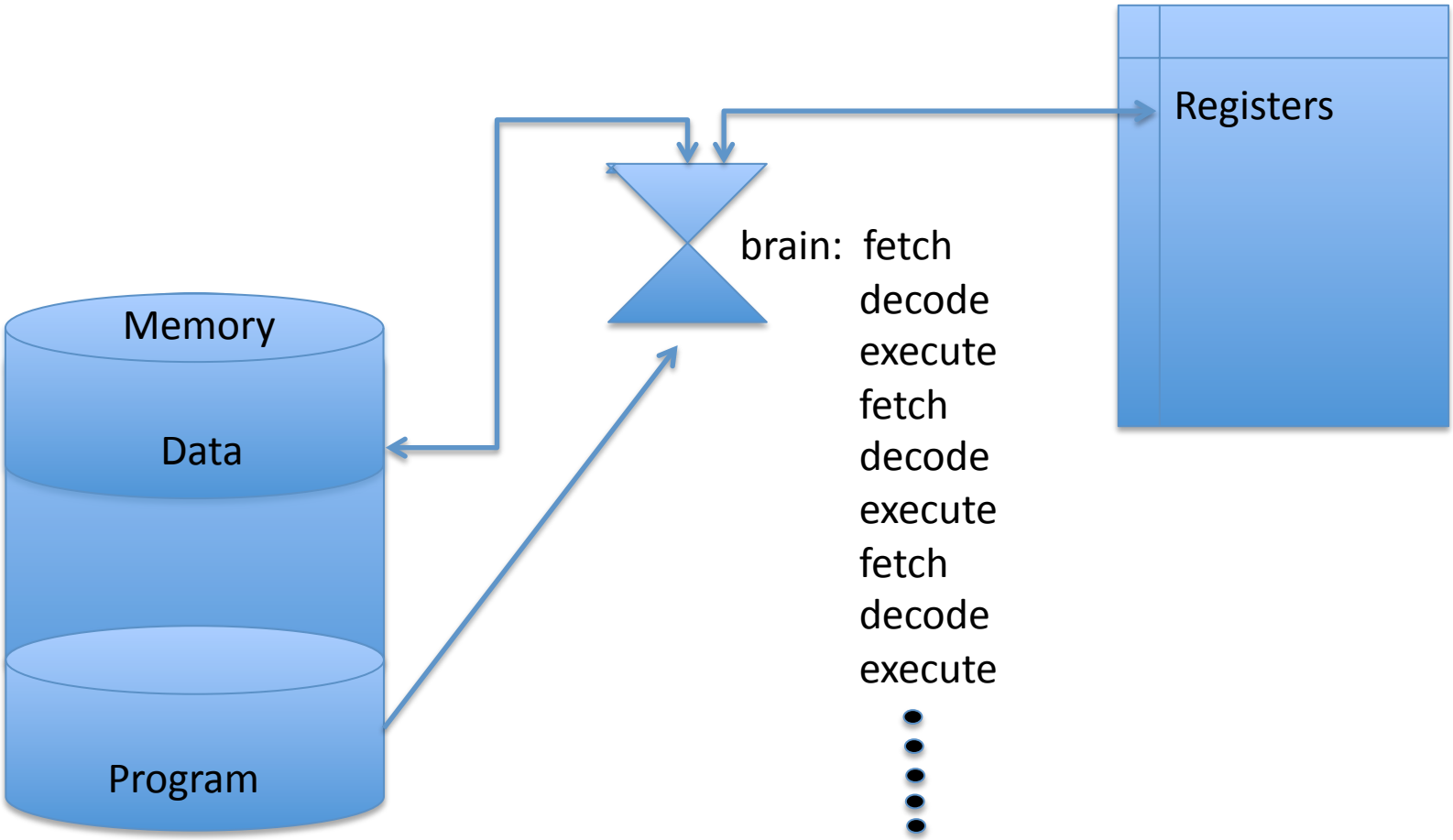
- Recap:
  - What is an FPGA?
  - Where are FPGAs used?
  - How are FPGA's programmed?
- Bluespec SystemVerilog
  - Modern language used in the design of electronic systems
- Example: Matrix Multiplication
- Prior, Current and Future Work

What is an FPGA?

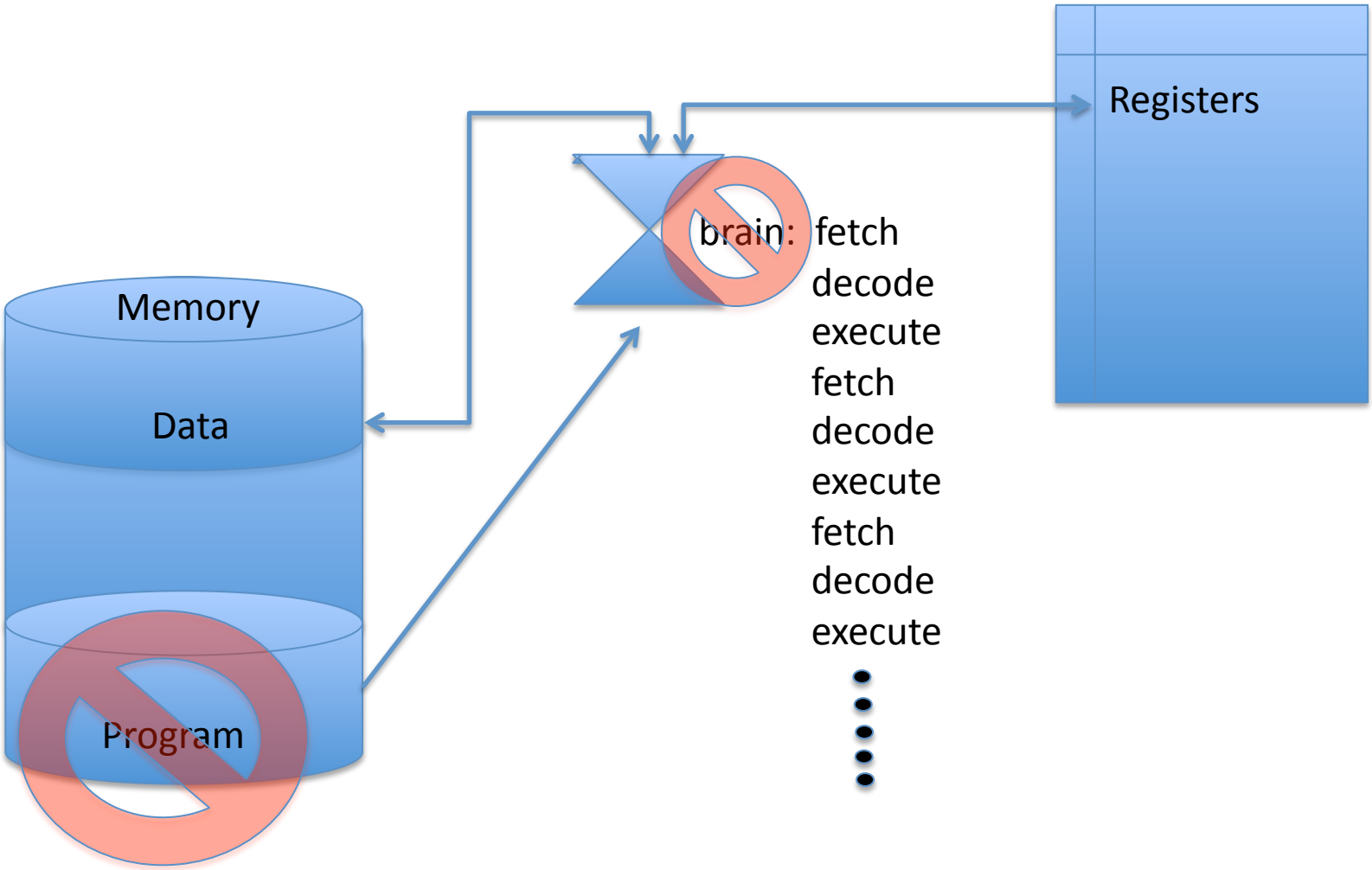
- FPGA = Field Programmable Gate Array
- Alternative to a CPU (like GPUs), or
- Coprocessor used to offload computationally intensive tasks



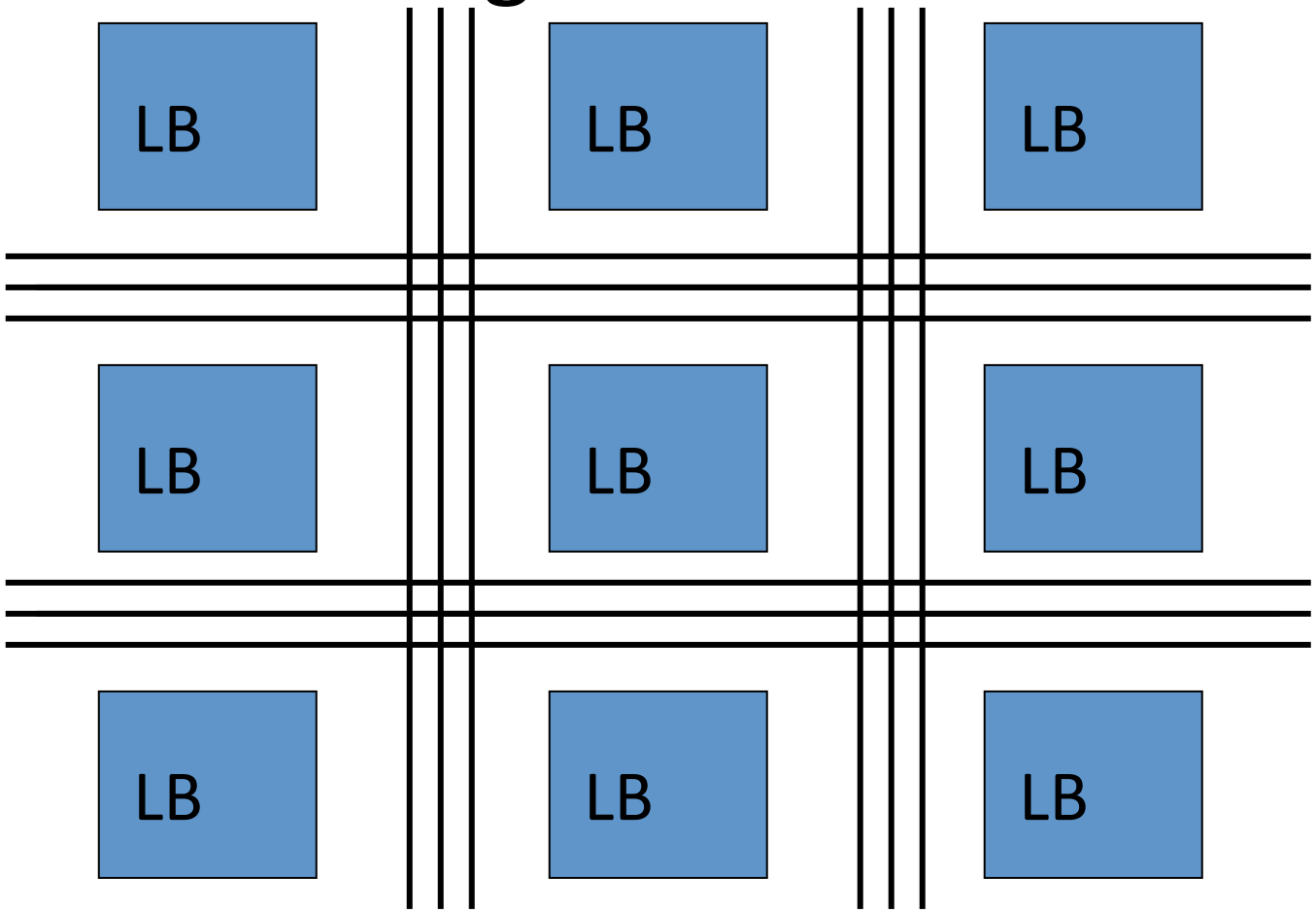
In contrast, what is a CPU?



How an FPGA differs from a CPU



FPGA = large array of configurable logic blocks

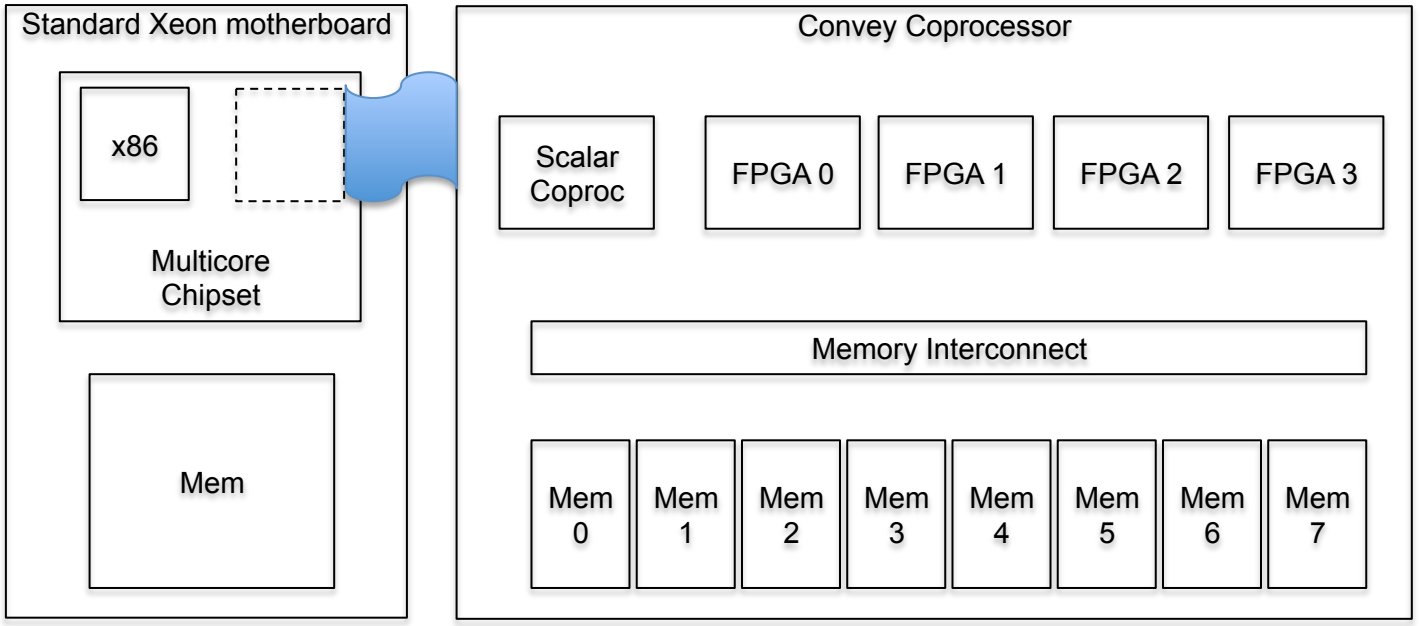


Look Up Tables (LUTs) and Flip-Flops are the basic building blocks.
"Programming" is a customized configuration of the logic blocks.

FPGA Variant: Xilinx Virtex-6

- 6-input LUTs (474,240 LUTs)
- Each block contains four LUTs and eight flip-flops (118,560 blocks)
- Some blocks can use their LUTs as distributed RAM (up to 8,280 Kb)
- 864 DSP48E1 slices where each one contains a multiplier that takes two binary inputs, one up to 18-bits, the other 25-bits, and an adder/ subtracter/accumulator that takes three binary inputs up to 48-bits wide
- Block RAMs are 36 Kbits in size

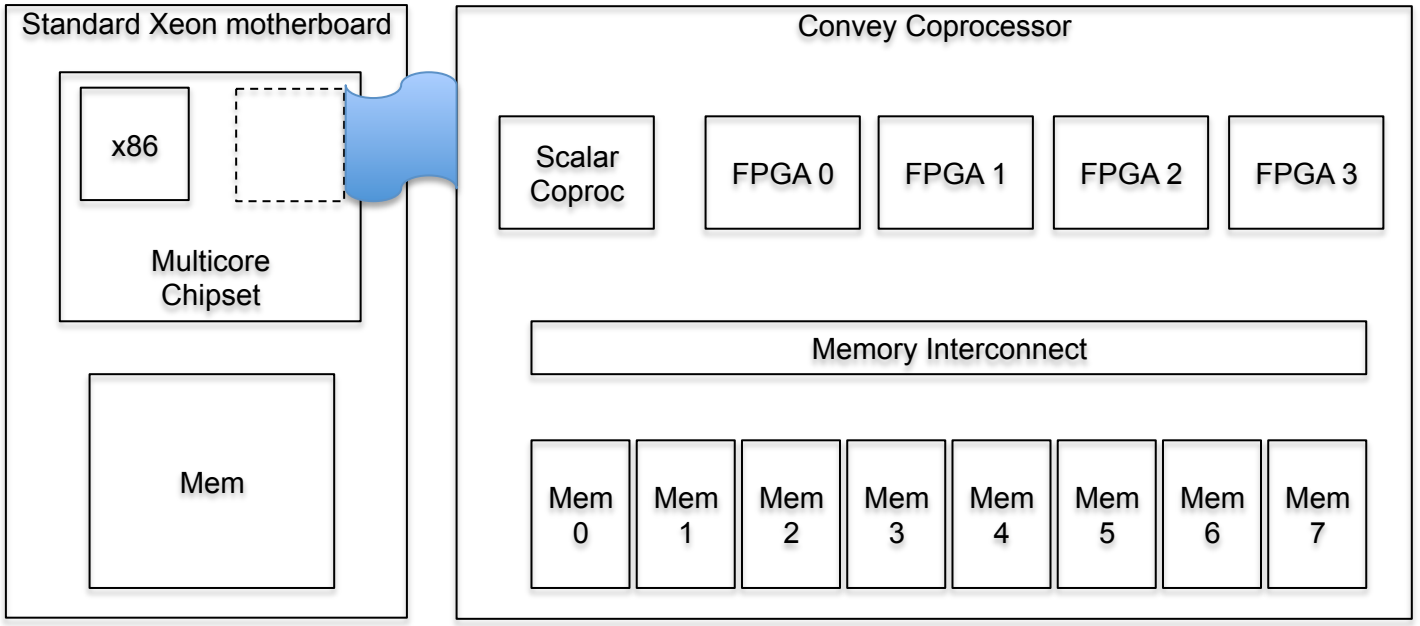
Convey HC-1ex



Rack-mount box with two boards

- A standard Xeon server board with two Front Side Bus (FSB) processor sockets.
- A custom board with 4 user-programmable FPGAs (Virtex 6 LX760)
- Tightly coupled memory with multiple high-bandwidth ports on each FPGA
- Tight integration with the x86 server:
  - Appears to the x86 as a genuine x86 coprocessor, i.e., executes co-processor instructions using standard x86 co-processor protocols.
  - *Coprocessor memory looks like standard memory to the x86, i.e., it is cache-coherent with the x86 server's own memory, and addressed from the FPGA using standard virtual addresses*.

Convey HC-1ex



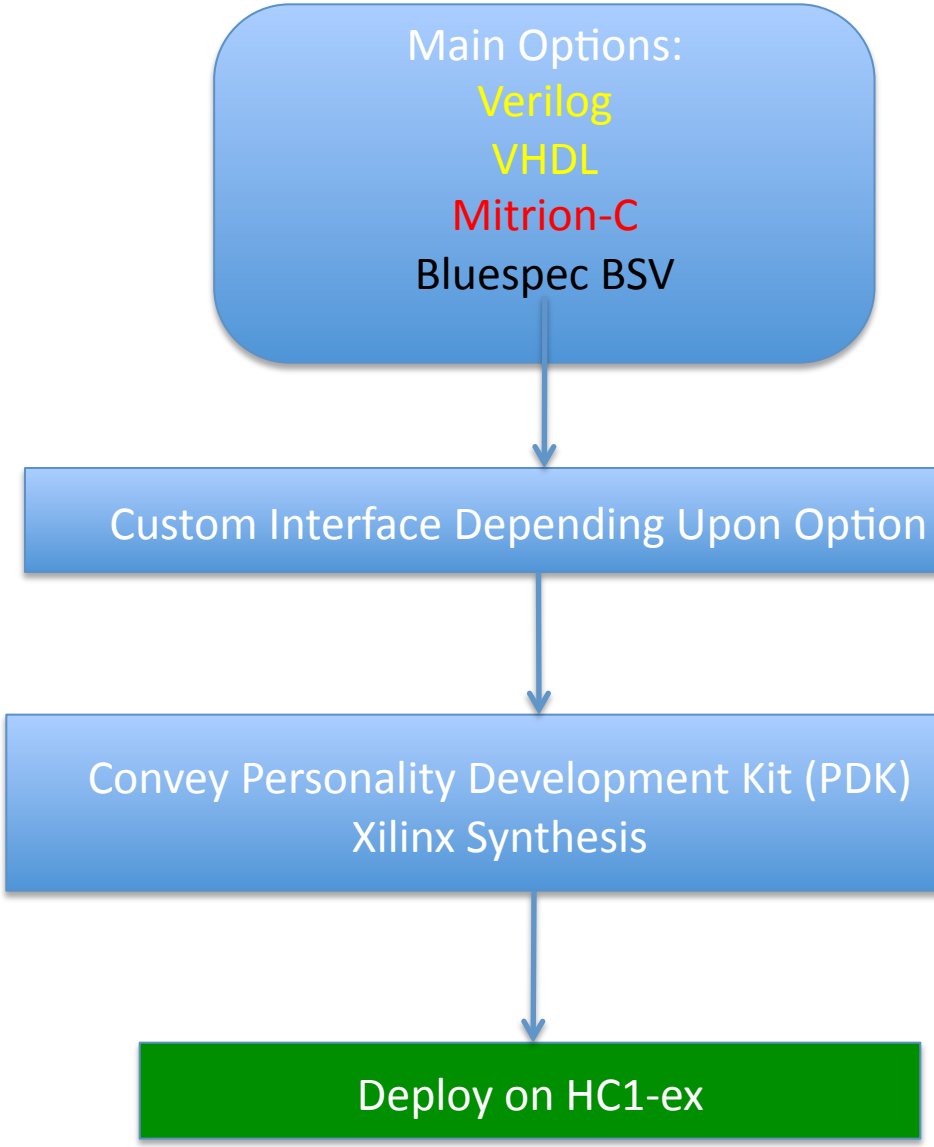
FPGA's local external memory

- 8 memory banks with parallel access from each FPGA (80 GB/s total)
- Each FPGA (AE) has 2 ports to each of the 8 Memory Controllers (MCs).
  - Thus, each FPGA has 16 ports!
- Pipelined (up to ~256 requests in flight)
- Out-of-order read responses. Requests/responses carry user-supplied 32b tag for identification
- Fire-and-forget writes; weak ordering; memory 'fence' op to wait for write-completions

Architecture Summary

- FPGAs avoid the instruction/fetch/decode bottleneck of traditional Von Neumann architectures
- Convey's novel memory system was designed to address common bottleneck performance limitations

Software Development

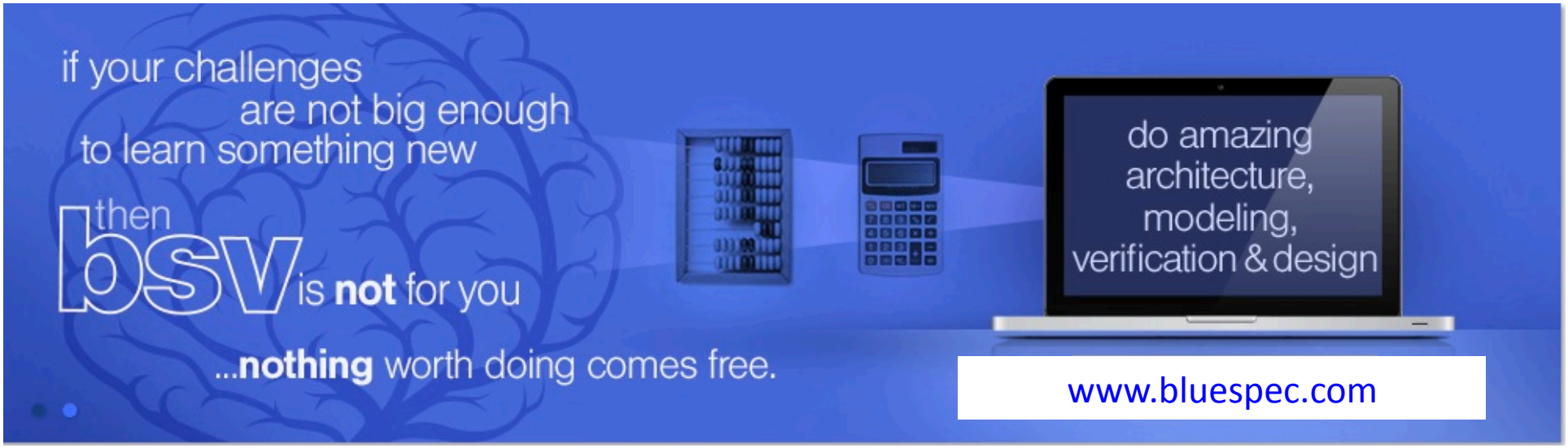


Want to use a new language?

Then a custom interface will be needed to interface with Convey software.

Luckily, we do not need to program all the other FPGAs inside the box!

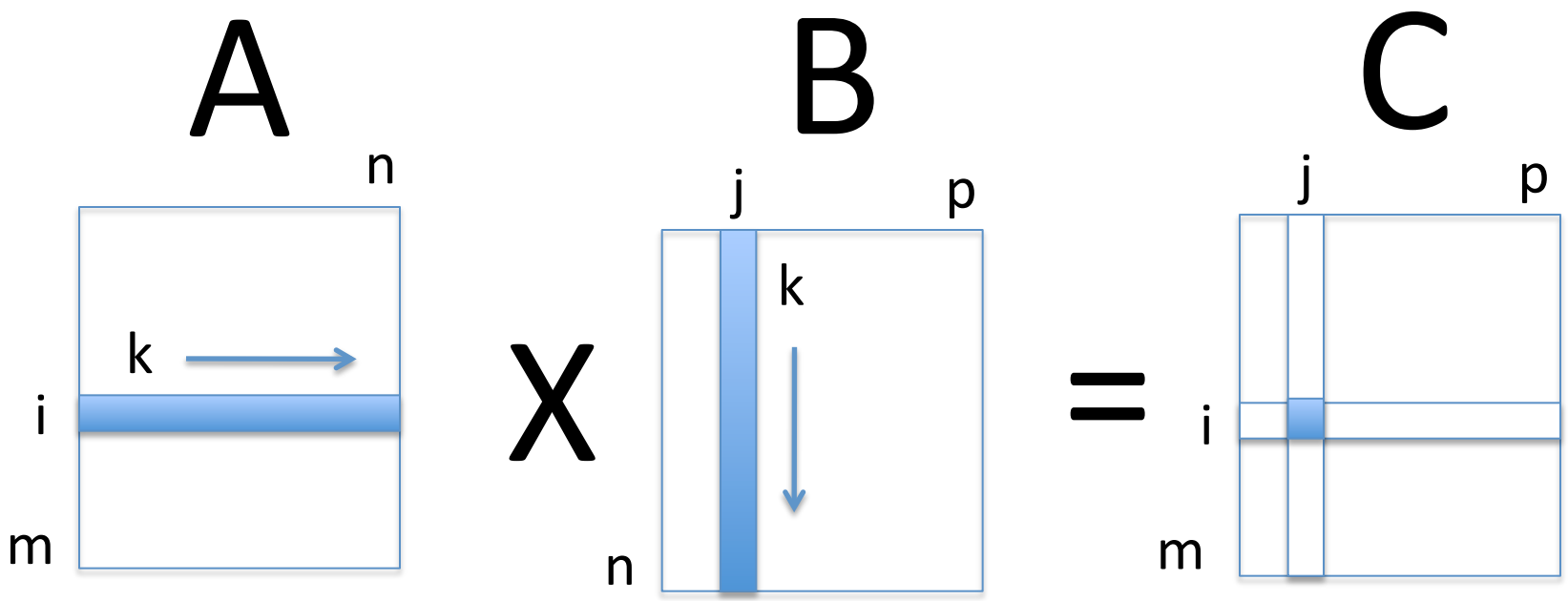
Bluespec SystemVerilog (BSV)



Why Choose Bluespec BSV?

- to be able to program at a low enough level (as in Verilog or VHDL) to be able to design a circuit,
- to write code and use high-level interfaces (like FIFOs) that are comparable to interfaces for other high-level languages (unlike Verilog or VHDL),
- to be able to verify correctness at the right level of programming (atomic rules), and
- to be able to run fast implementations, provided the code is not too complex.

Example: Matrix Multiplication

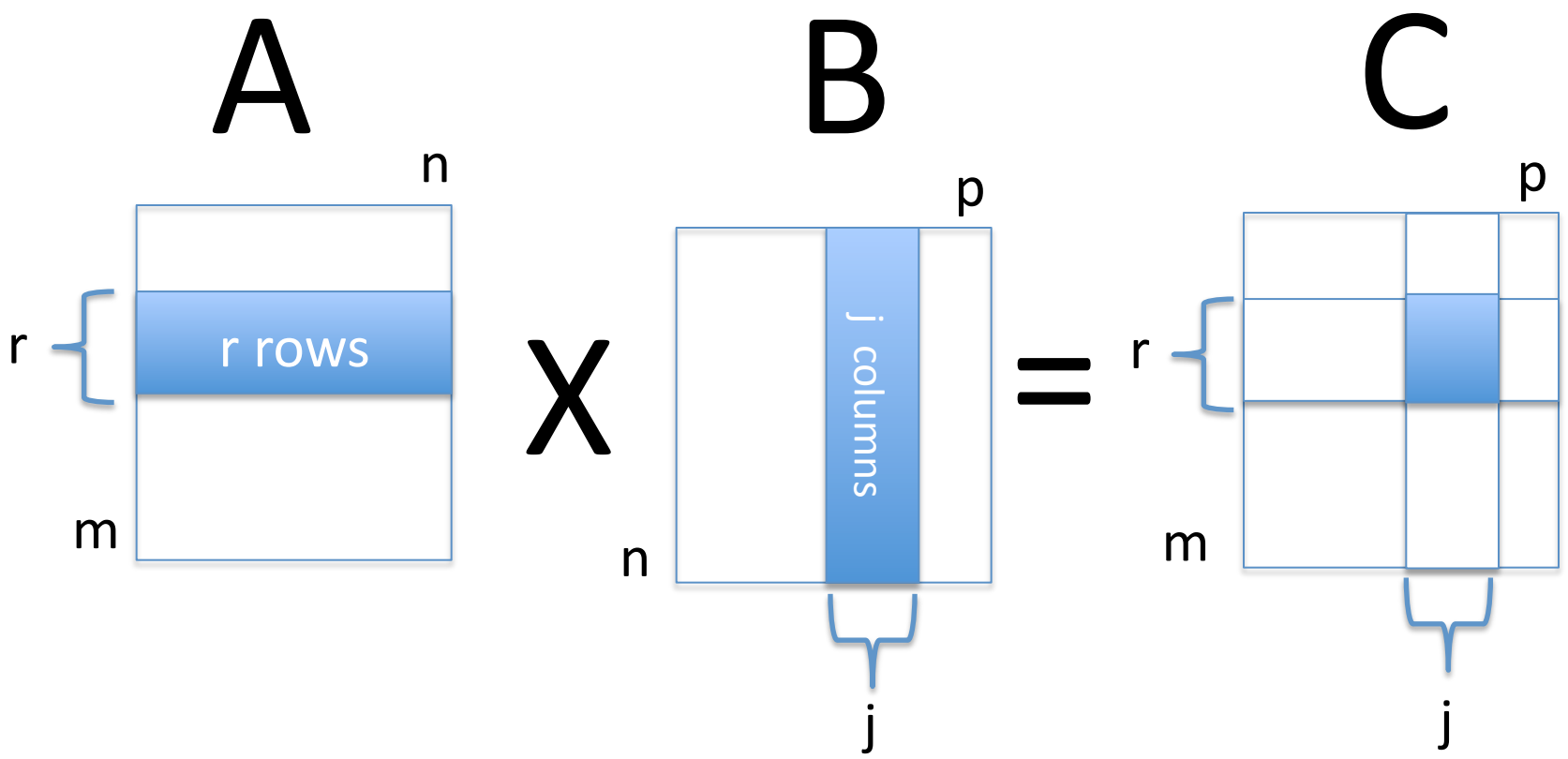



```
for i = 1..m
  for j = 1..p
        c[i,j] = 0
        for k = 1..n
              C[i,k] = C[i,j] + A[j,k] x B[k,j]
```

C = A x B

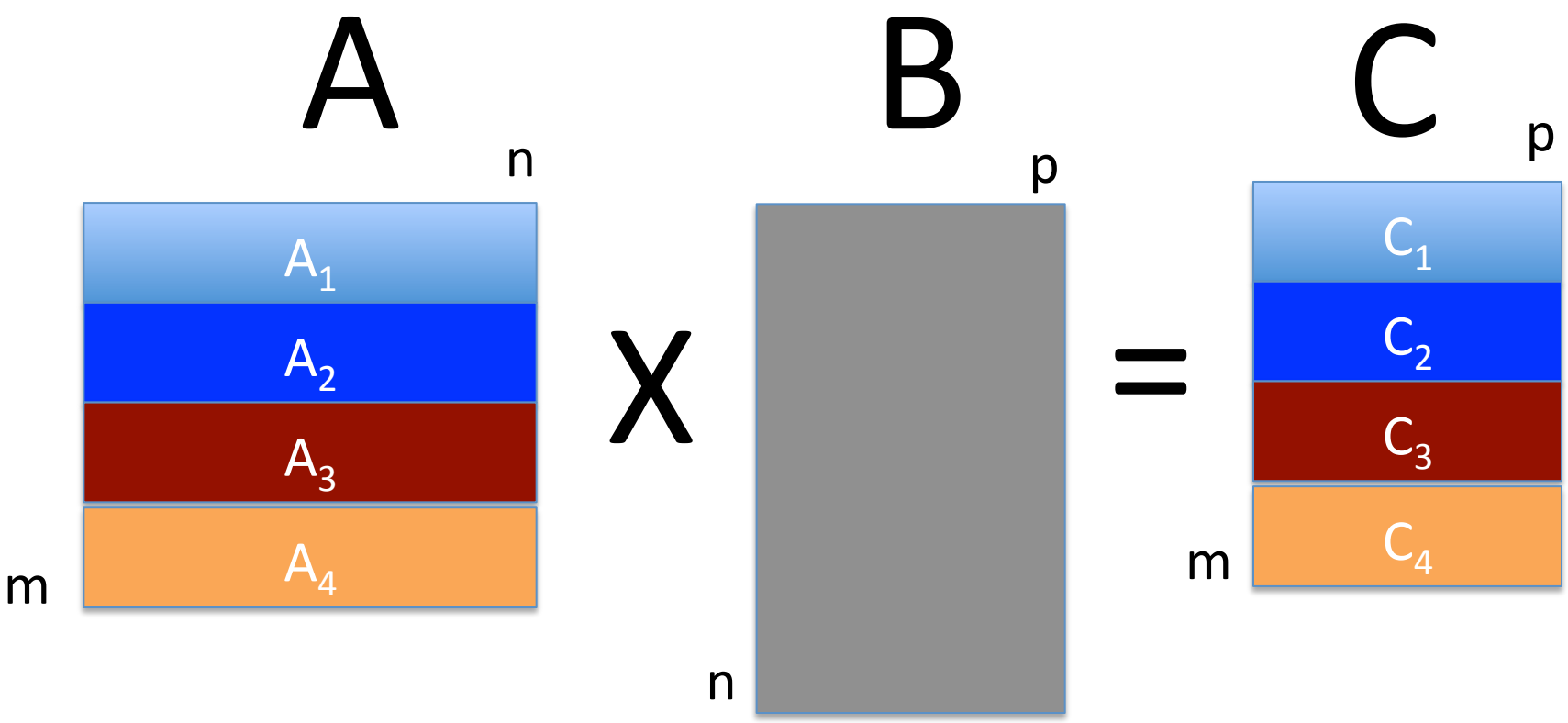
← "MAC"

Usual Block Matrix Multiplication



Instead of computing a cell at a time, as usual, compute a block at a time.

Parallelization Strategy for Four FPGAs



FPGA N computes $C_N = A_N$ X B for N = 1..4 using a block matrix algorithm.

Initial Experience: Use Brute Force

- Use brute force and program with 64-bit integers
- Discovered a Xilinx synthesis bug, which was known by Xilinx, but none of us (Convey, Bluespec) were aware of this bug.
- Despite many attempts, synthesis would never succeed due to 64-bit multiplies. Recall on board, there is only a 18x23 multiplier.

Next Step: Custom Implementation of 32-bit Single-Precision Floating-Point

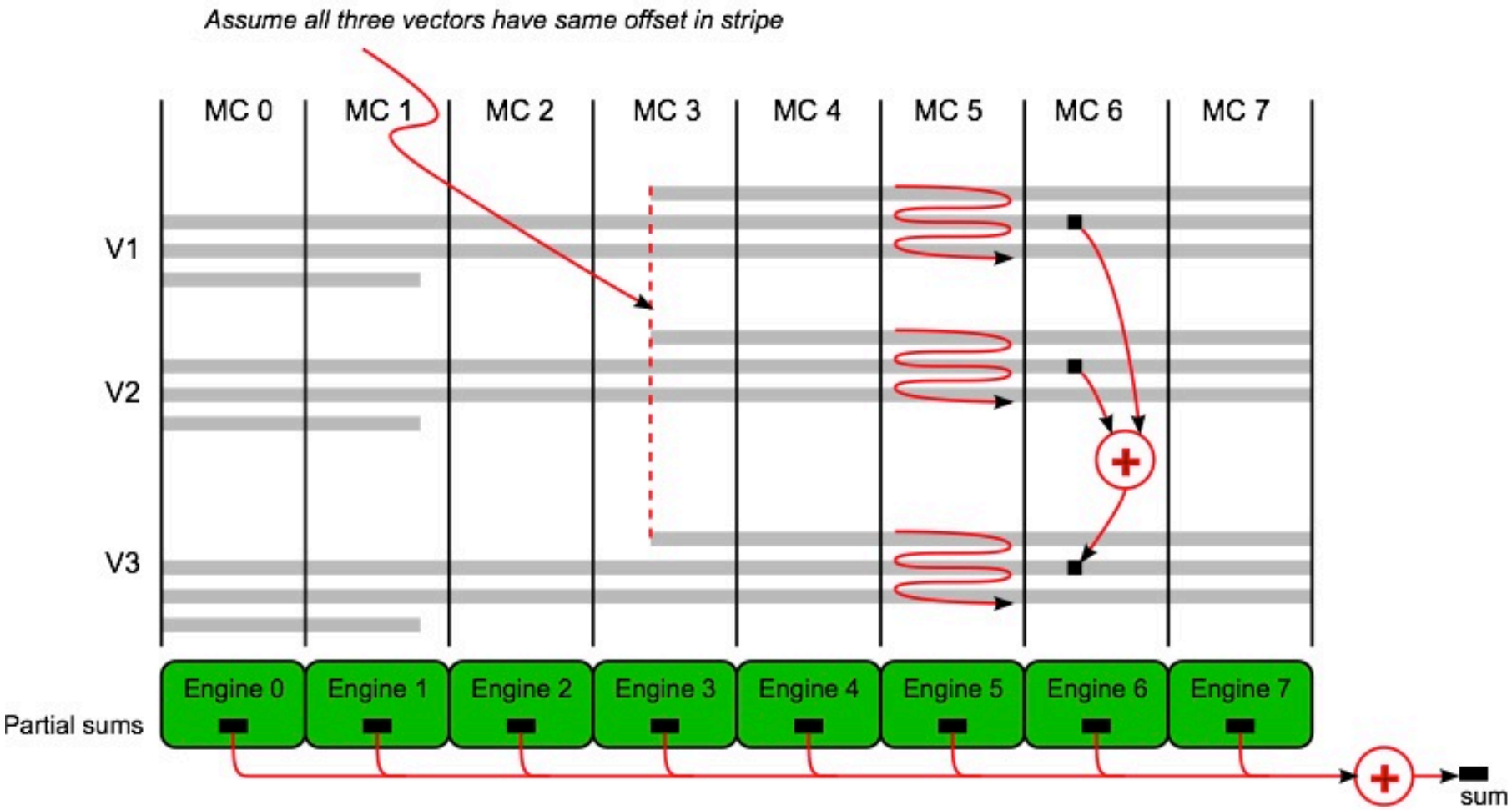
- Need to break up the operations into smaller ones, so that each operation can be mapped to the multipliers and adders on the Virtex-6
- Experimented with both standard high-school technique, and the so-called karatsuba technique
- Did considerable testing in C language to ensure every calculation in custom implementation was at least as accurate, matching precisely the rounding, as if done in C using IEEE 754 standard format, with full support for normal and denormalized numbers.

Custom Implementation of 32-bit Single-Precision Floating-Point

We do near double-precision calculations as our representation uses a 48-bit significand as seen in Bluespec BSV code:

```
typedef struct {
    Bit #(3)      switch;
    Int #(13)     exponent;
    UInt #(48) significand;
} SPMAC deriving (Bits, Eq);
```

Initial Parallelization Technique For Each FPGA Using Block Algorithm



(source: Bluespec™ Option for Convey™ PDK)

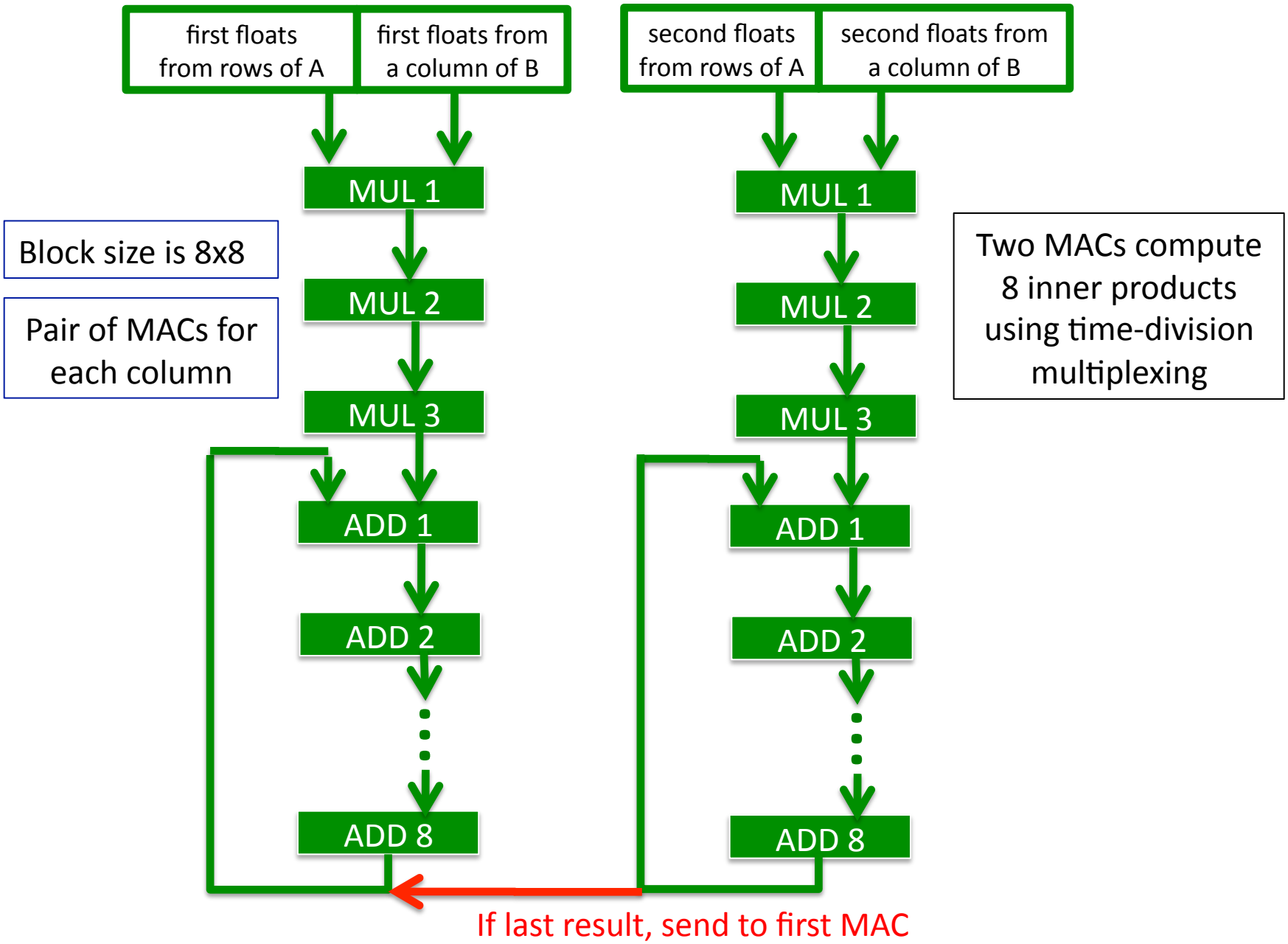
Optimizations

- Reduce complexity
  - Use fewer MACs and adders by pipelining and time-division multiplexing
    - Use a single MAC, which involves several stages, to compute the partial results for all rows in a column block
    - Use a single adder, which involves several stages, to add the partial sums for an entire block
  - Avoid interleaving reads and writes, which was the lesson from Bluespec's implementation of Convey's Vector add example

Current Work

- Forget about alignment! Vectors can be anywhere in memory (after fixing bug in BClib)
- Computations proceed independently of reading and writing
- Don't interleave reads and writes, and always perform exactly the "magic number" of writes
- Eliminate the adder that combines the partial sums from each "MC" engine as we abandon the prior "MC" parallelization technique, and use multiple time-multiplexed MACs

MAC Kernel


first floats from rows of A
first floats from a column of B
second floats from rows of A
second floats from a column of B
MUL 1
MUL 2
MUL 3
ADD 1
ADD 2
ADD 8
MUL 1
MUL 2
MUL 3
ADD 1
ADD 2
ADD 8
Block size is 8x8
Pair of MACs for each column
Two MACs compute 8 inner products using time-division multiplexing
If last result, send to first MAC

Future Work

- Implement double-precision version
- Design a sparse matrix-matrix solution
- Find other applications

Questions?