Using FPGAs for HPC* acceleration: now and in 20 years

*High Performance Computing
for large “warehouse” apps.

MAXELER
Technologies
MAXIMUM PERFORMANCE COMPUTING

Michael J. Flynn

Maxeler Technologies and Stanford University

Where we are now….

Accelerator HW model

- Assumes host CPU + FPGA accelerator
- Application consists of two parts
  - Essential (high usage, >99%) part (kernel(s))
  - Bulk part (<1% dynamic activity)
- Essential part is executed on accelerator; Bulk part on host

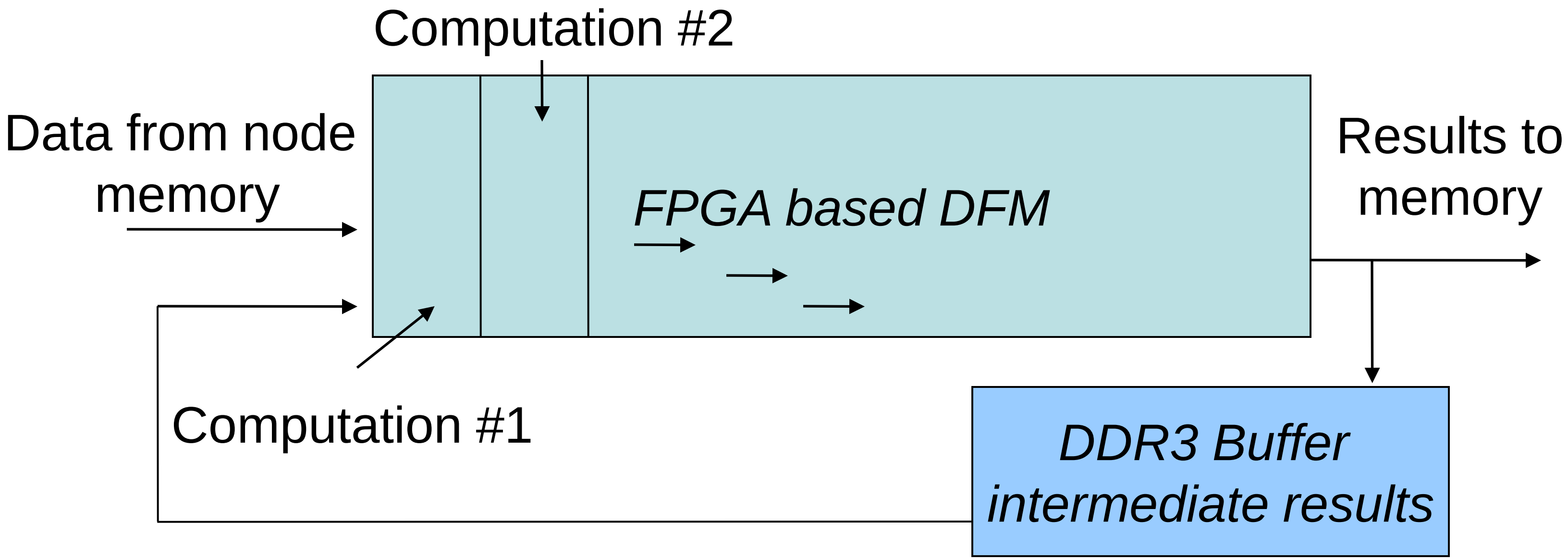
Acceleration with Static, Synchronous, Streaming DFMs

- Create a static DFM (unroll loops, etc.); generally the goal is throughput not latency.
- Create a fully synchronous DFM synchronized to multiple memory channels. The time through the DFM is always the same.
- Stream computations across the long DFM array, creating MISD or pipelined parallelism.
- If silicon area and pin BW allow, create multiple copies of the DFM (as with SIMD or vector computations).
- Iterate on the DFM aspect ratio to optimize speedup.

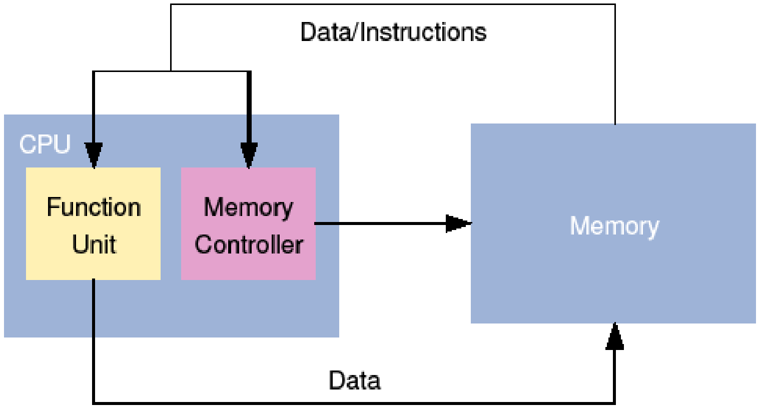
Acceleration with Static, Synchronous, Streaming DFMs

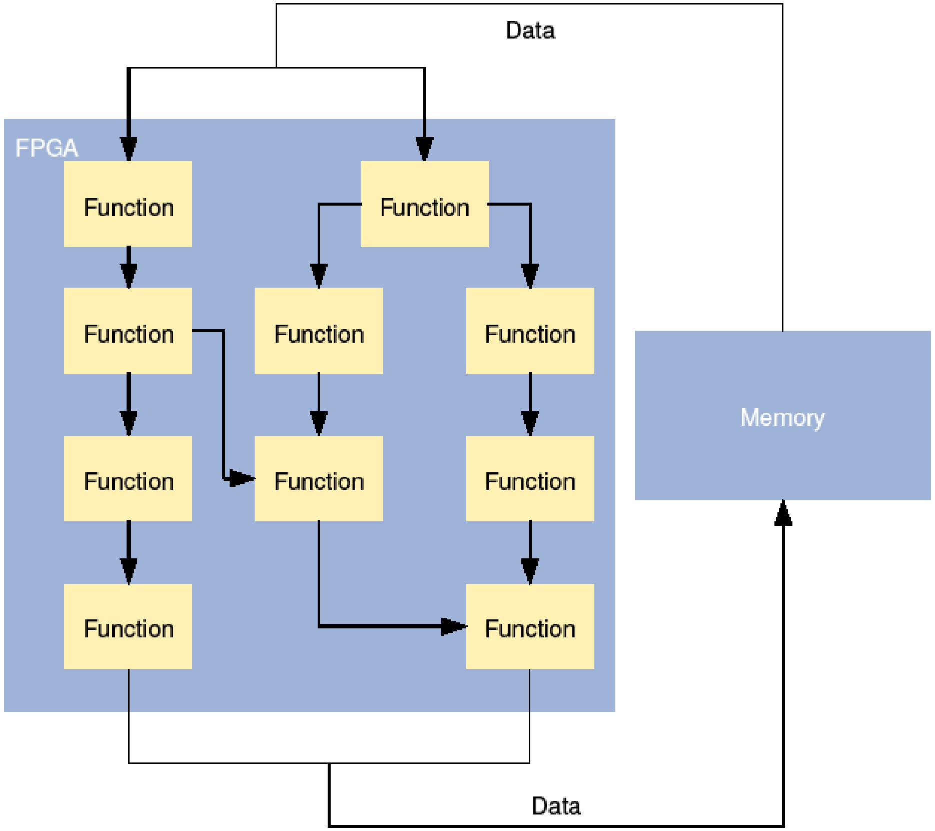
- Create a fully synchronous data flow machine synchronized to multiple memory channels, then stream computations across a long array

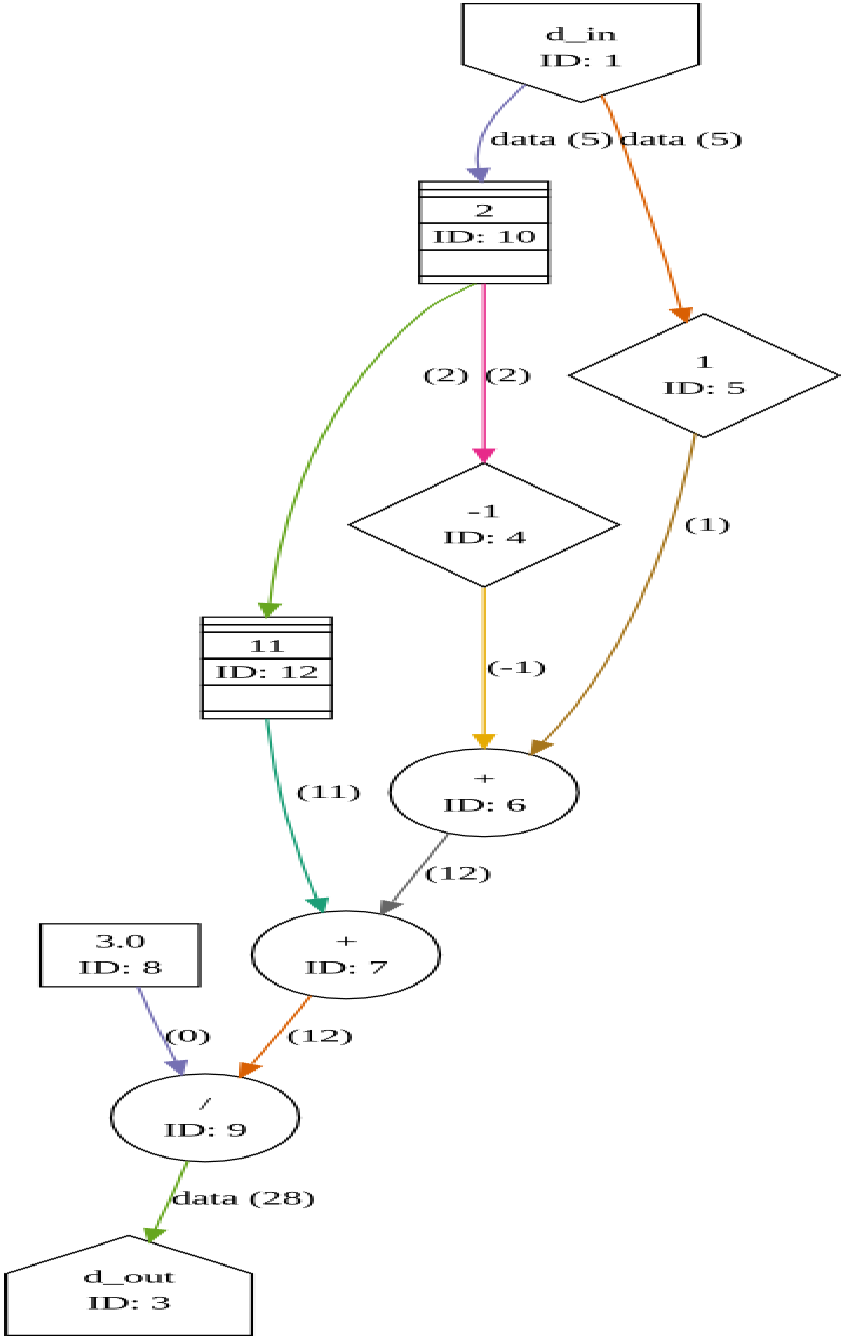
PCIe accelerator card



CPUs vs. Stream Processing



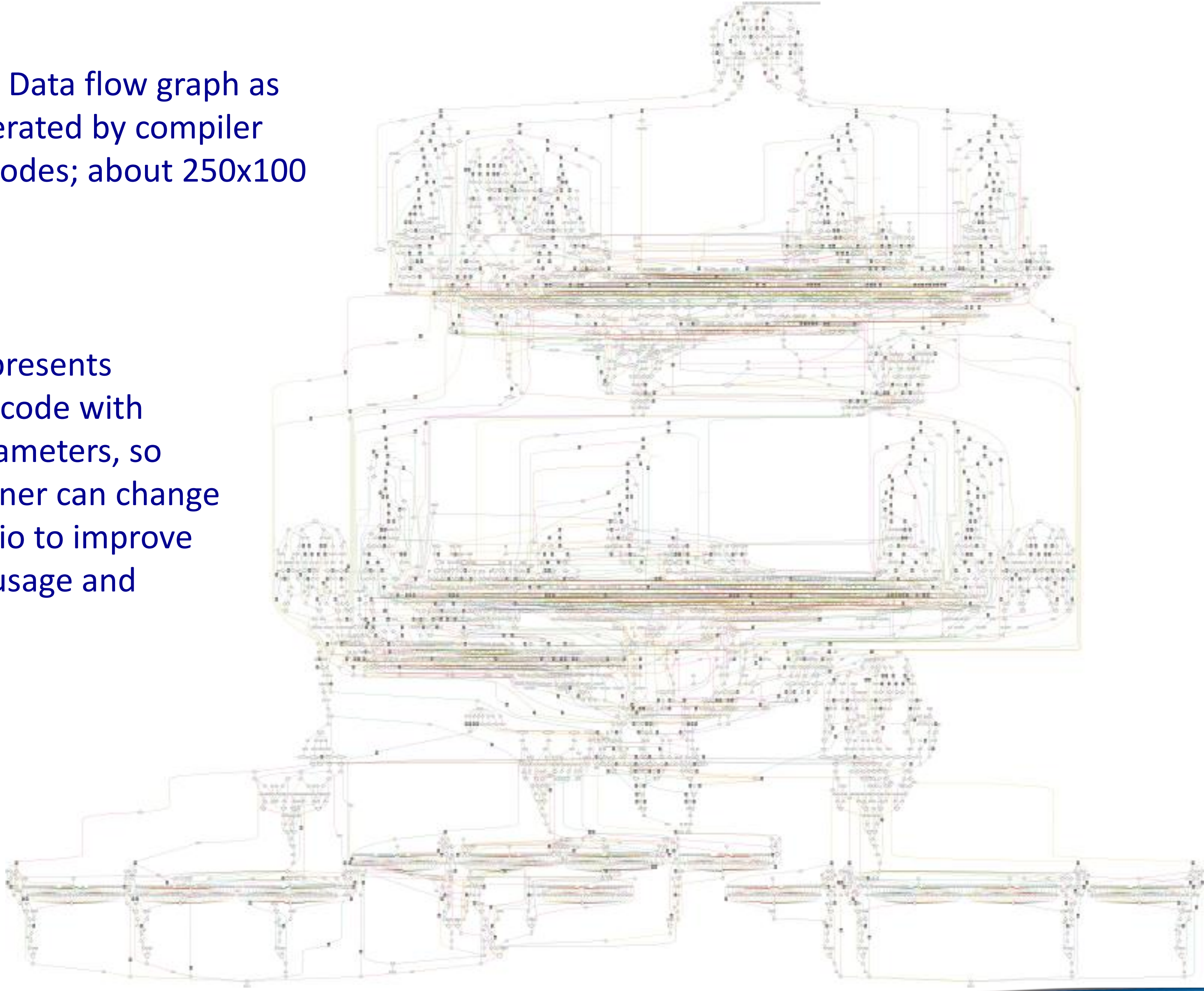




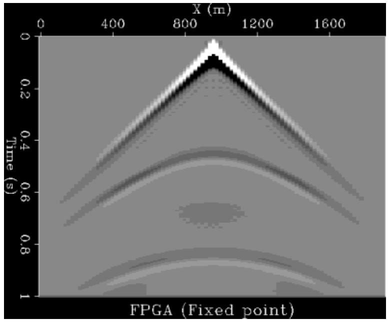
JAVA based DF graph description.
Automatic generation /
compilation creating DFM
buffer synchronized

Initial Data flow graph as
generated by compiler
4866 nodes; about 250x100

Each node represents a line of JAVA code with area time parameters, so that the designer can change the aspect ratio to improve pin BW, area usage and speedup

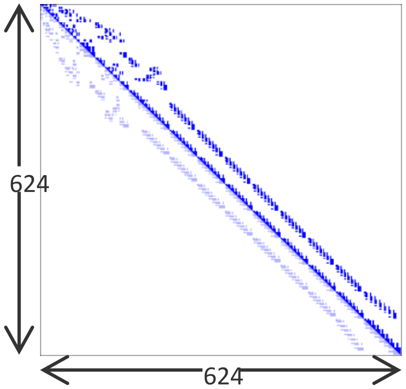


Achieved Computational Speedup for the entire application (not just kernel) compared to Intel server



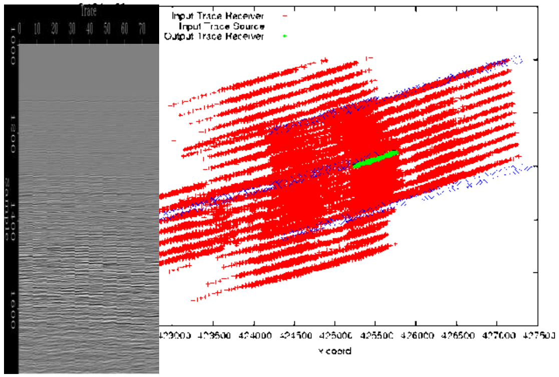
RTM with Chevron

VTI 19x and TTI 25x



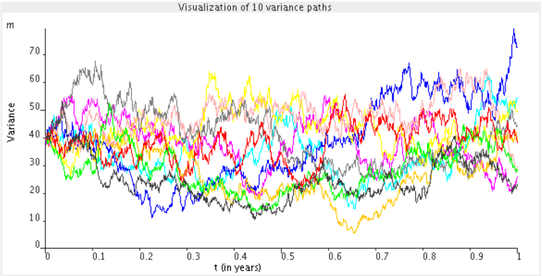
Sparse Matrix

20-40x



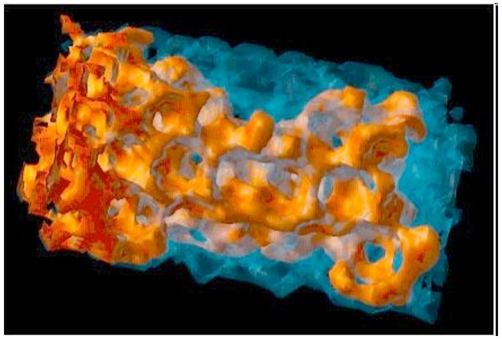
Seismic Trace Processing

24x



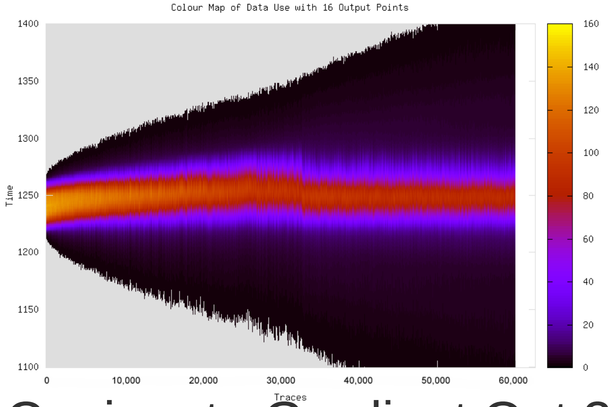
J.P.Morgan

Credit 32x and Rates 26x



Lattice Boltzman

Fluid Flow 30x



Conjugate Gradient Opt 26x

So how can emulation (FPGA) be better than high performance x86 processor(s)?

- Multi core approach lacks robustness in streaming hardware (spanning area, time, power)
- Multi core lacks robust parallel software methodology and tools
- FPGAs form an unlikely basis for acceleration
- Success comes about from their flexibility in matching the DFG with a synchronous DFM and streaming data through and shear size > 1 million cells
- Effort and support tools (JAVA DFM compiler, memory choreographing SW) provide significant application speedup

And 20 years from now?

HPC plus 20 years

- Role of FPGAs? depends on competing technologies (multi core, etc..)
- All technologies will be array based; impossible to manage / design /validate singular designs
- All technologies severely limited by software that enable applications to exploit parallelism. Need a rethinking of
  - Architecture
  - Compilers
  - User interface

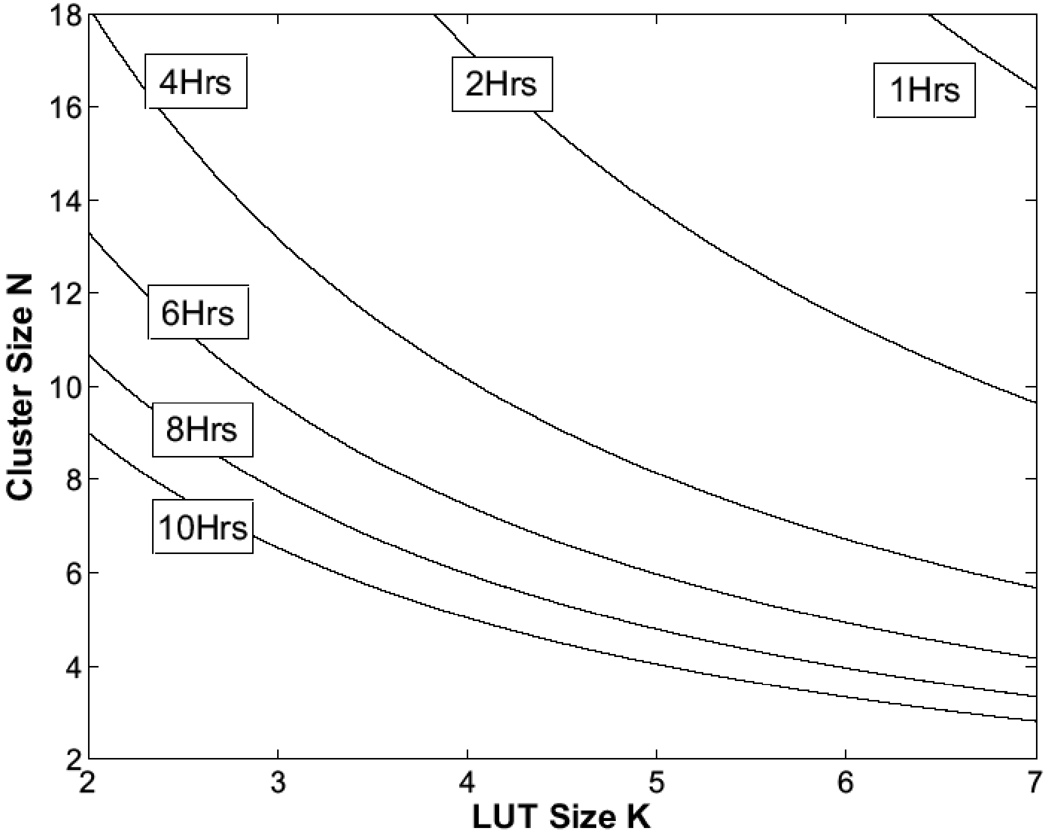
Multi Core plus 20

- It’s difficult to see where today’s multi core goes. Limited by fixed isp, fixed interconnect, fixed memory access protocols
- Frequency (limited by power density) remains relatively static, 2-4 x
- Why 1000 or 10,000 cores per die (and 90% devoted to cache), when memory is already the bottleneck.
- A more generalized, flexible array technology?
- Programming model will evolve away from current (sequentially oriented) models.

FPGA for HPC plus 20

- Computational density increases by 1000 to 10,000 (technology, circuitry, coarser grain)
- The FPGA with 2 personalities: fine and arithmetic (HPC)
- Frequency (again) relatively static
- Software DFM oriented; more/better application mapping tools
- Multi channel choreographed memory access now, later memory on die, customizable memory interconnects
- For coarser grain array technology, a convergence with flexible multi core? What is the grain in coarse grain?

BUT, what about place and route?



The more clusters to place and route, the more the runtime. Data from 09, no better now

Runtime constraints for P&R time

Chin and Wilton, “Modeling …FPGA… Place and Route Runtime, FPL 09

BUT, what about the user?

- Over the decade 1999-2009 the literature seems to indicate that P&R time has increased by at least 10x, even as the P&R execution HW became 10x faster.
- HPC applications need to deal with large number of clusters (DFG nodes), now at 10 hour P&R.
- And what about 20 years from now? Months to do a P&R for HPC? One would think that P&R is the stepchild application for HPC. Why not vendor P&R HPC cloud?
- Larger issue is application mapping (source to run) User concerns: time to implement and time to execute, not circuitry or silicon

Comments

- In HPC the success of FPGA acceleration points to the weakness of evolutionary approaches to parallel processing: hardware (multi core) and software (C++, etc.), at least for these HPC applications.
- The automation of acceleration is still early on; still required: tools, methodology for writing apps., analysis methodology and (probably) a new hardware basis (coarser grain, less P&R time).

Conclusions

- There's a role for FPGAs in HPC if underlying software problems can be solved.
- In HPC the parallel approach demands rethinking algorithms, programming approach and environment and underlying hardware.
- There's a lot of research ahead to effectively create parallel translation and array based technology.