Dataflow Supercomputers

Michael J. Flynn

Maxeler Technologies and Stanford University

Outline

- Some parallel processor history
- How did we get where we are?
- Moving forward with Multicore & FPGA convergence
- Reconfiguration as a supercomputer technology
- The road ahead

Parallel Processor history: The Amdahl – Slotnick debate of 1967.

Amdahl's Laws

- First law: Serial processors always win; too much time spent in programming the parallel processor.
- Second law: fraction of serial code, s, limits speedup to $S_p = T_1 / (T_1 (s) + T_1 (1\text{-}s)/p)$ or

$$S_p = 1 / (s + (1\text{-}s)/p)$$

Slotnick's law

"The parallel approach to computing does require that some original thinking be done about numerical analysis and data management in order to secure efficient use.

In an environment which has represented the absence of the need to think as the highest virtue this is a decided disadvantage."

-Daniel Slotnick

….***Speedup is achieved by algorithmic, analytic & programming effort***……

So Amdahl was right, at least until 2002

- But up until the present, development largely focused on the sequential processor using *ilp* of various sorts (vectors, superscalar, VLIW) to improve sequential performance.
- And programmers increasingly embraced the sequential machine model (layers of abstraction, etc.); relying on compilers, etc. for the HW mapping
- But frequency scaling stopped around 2002 AND..

We have to deal with Slotnick’s Law

- So far, we’ve largely ignored the shift as the degree of multi core was low; indeed kept low by emphasis on *ilp* and big caches.
- And the shift to thin clients / notepads /etc. promoted “thin” processors; relegating the “fat” multi cores to the server market.
- Of course if we could figure out the programming, we’d be way better off with chips of mega thin proc., than fewer fat proc

Blinded by the (sequential) light

- Hardware: so we have multi threaded, superscalar cores with limited *ILP*; worse yet, most of the die area (80%) is devoted to two or three levels of cache to support the illusion of sequential model. And the cache organization doesn’t work for many data structures (especially as seen in big server applications).

Blinded by the (sequential) light

- Software: Most of what we've taught students about programming while useful for sequential programming productivity is useless for parallel processor speedup.
- For parallel processing, "layers of abstraction" "object oriented" "global memory" are all of dubious value.

The (multi core) Parallel Processor Problem

- Efficient distribution of tasks
- Inter-node communications (data assembly & dispatch) reduces computational efficiency: speedup/nodes
- Memory limitations
- Layers of abstraction hide critical sources of and limits to efficient parallel execution
- Result: scaled up cost, power, cooling and reliability concerns

MOVING FORWARD WITH Multicore FPGA convergence

Convergence phases:

1) **Simple FPGA accelerators**
2) **Multiple accelerators / multiple types**
3) **Reconfigurable resources of multiple types.**

Hardware and Software Alternatives

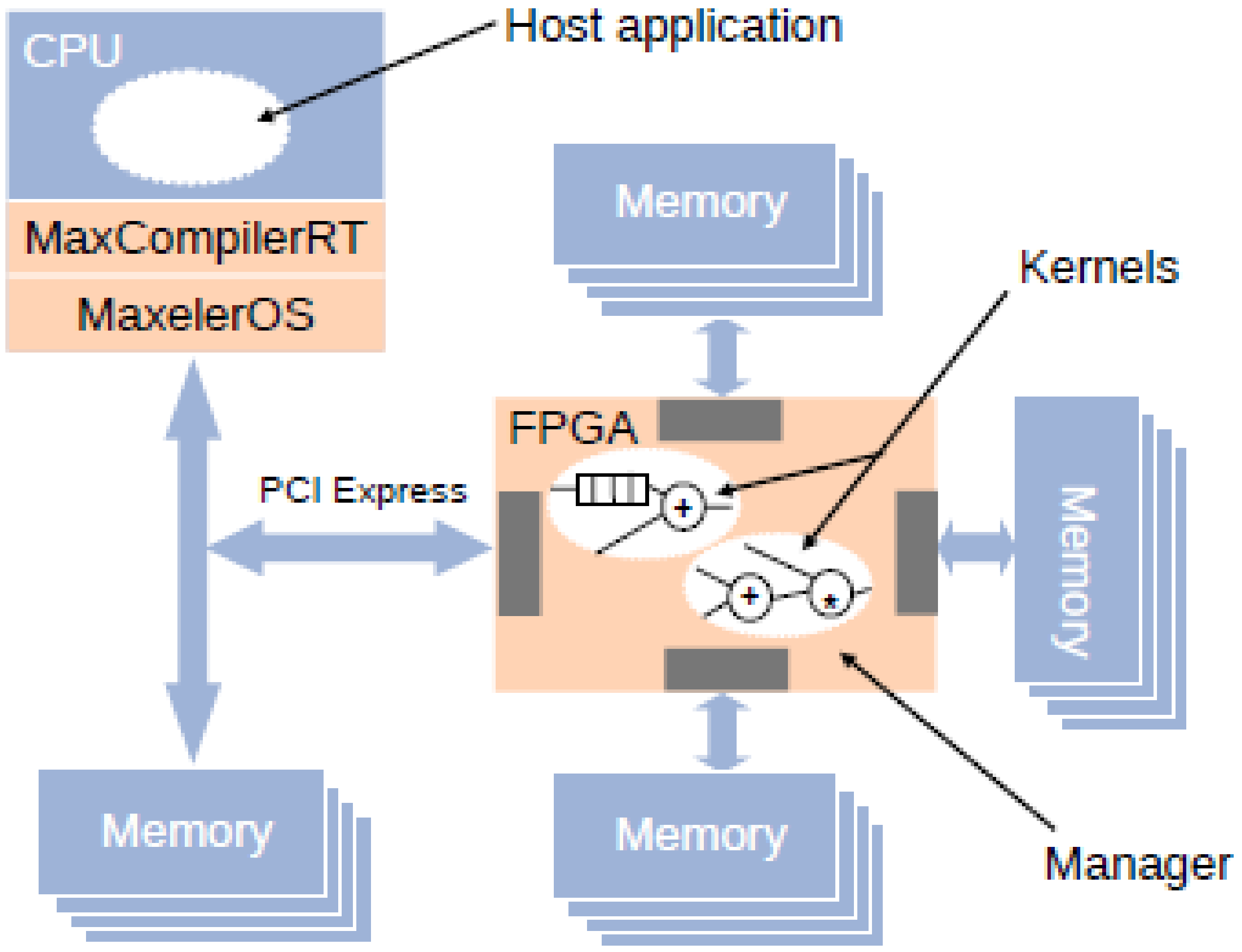
- Hardware:
  A more generalized (and reconfigurable) heterogeneous accelerator array model

- Software:
  A cylindrical rather than a layered model suits many applications

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
- So Slotnick's law of effort now only applies to a small portion of the application

FPGA accelerator hardware model: server with acceleration cards



Data flow graph (DFG) --> Data flow machine

- Each (essential) program has a data flow graph (DFG)
- The ideal HW to execute the DFG is a data flow machine that exactly matches the DFG
- A compiler / translator transforms the DF machine so that it can be emulated by the FPGA.
- FPGA based accelerators, while slow in cycle time, offer much more flexibility in matching DFGs.
- *Limitation 1*: The DFG is limited in (static) size to O (10^4) nodes.
- *Limitation 2*: Only the control structure is matched not the data access patterns

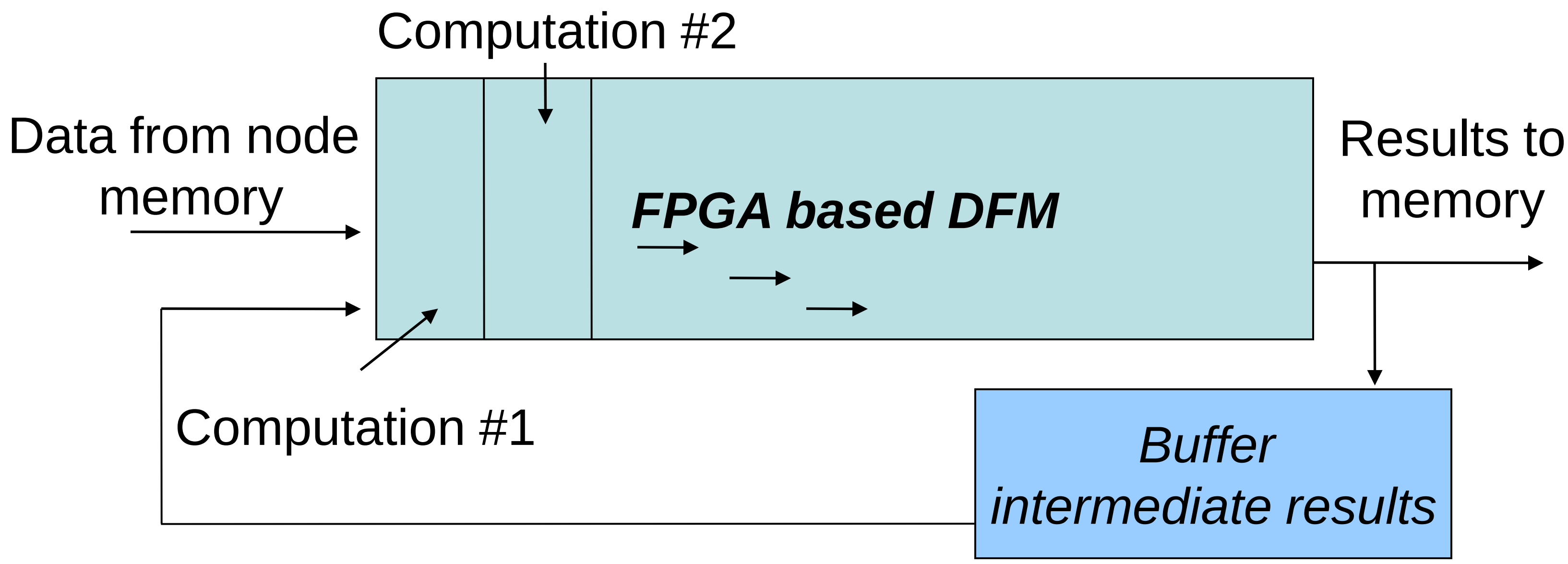
Acceleration with Static, Synchronous, Streaming DFMs

- 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

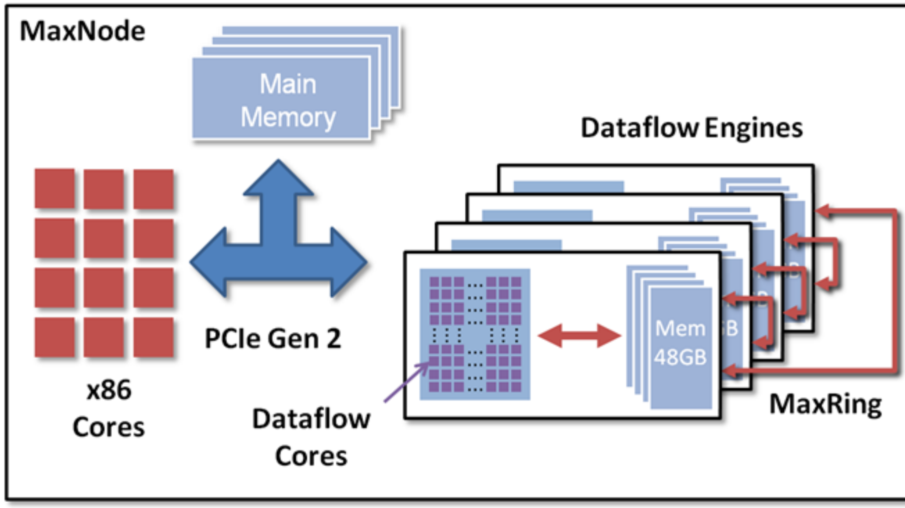
- Create a fully synchronous data flow machine synchronized to multiple memory channels, then stream computations across a long array

PCIe accelerator card



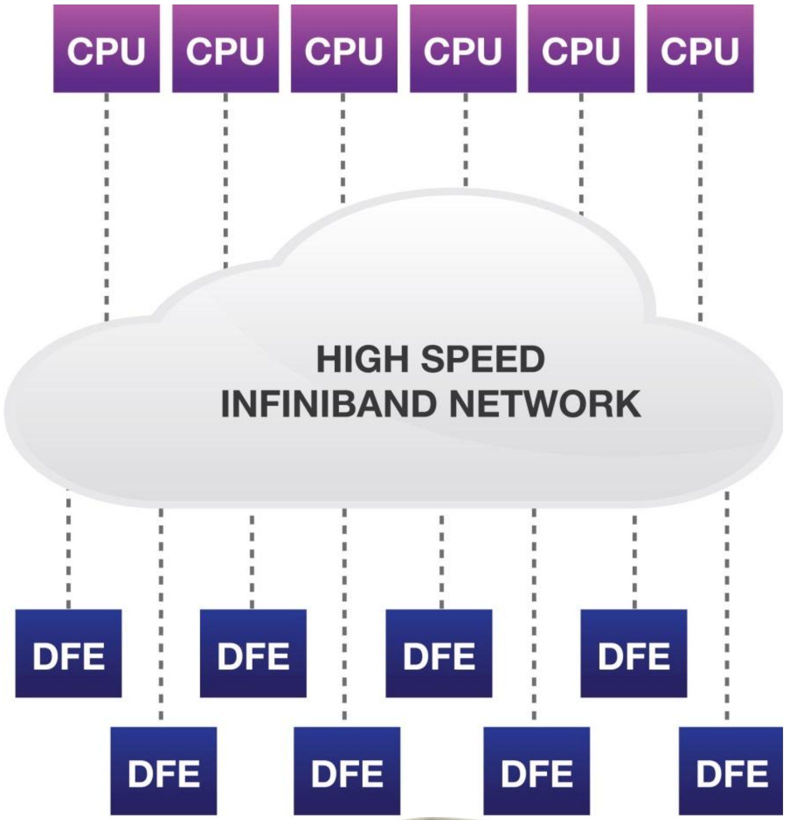
MPC-C500

- 1U Form Factor
- 4x dataflow engines
- 12 Intel Xeon cores
- 96GB DFE RAM
- 192GB CPU RAM
- *MaxRing* interconnect
- 3x 3.5” hard drives
- Infiniband

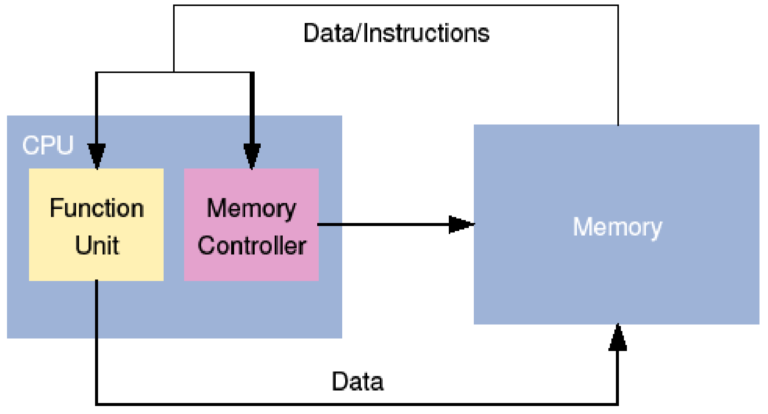


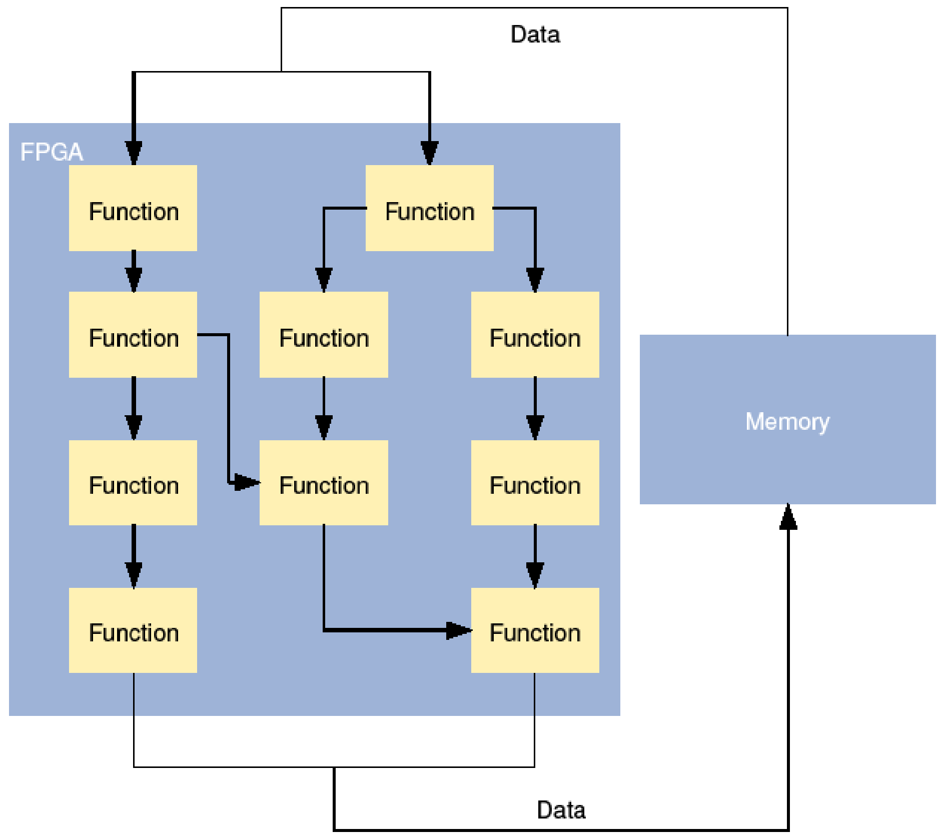
MPC-X1000

- 8 dataflow engines (192-384GB RAM)
- High-speed MaxRing
- Zero-copy RDMA between CPUs and DFEs over Infiniband
- Dynamic CPU/DFE balancing

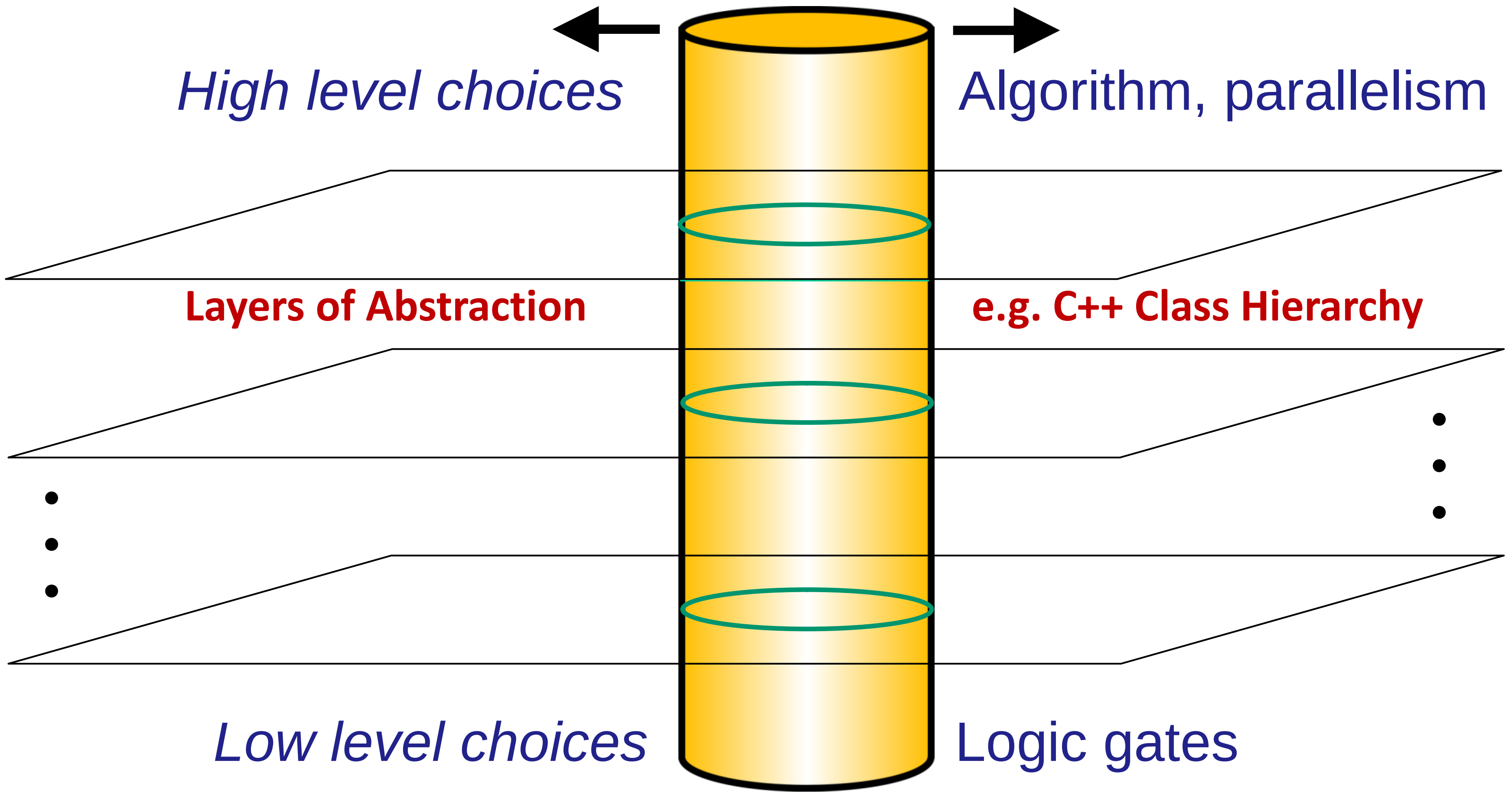


CPUs vs. Stream Processing





Cylindrical Model for Vertical Acceleration

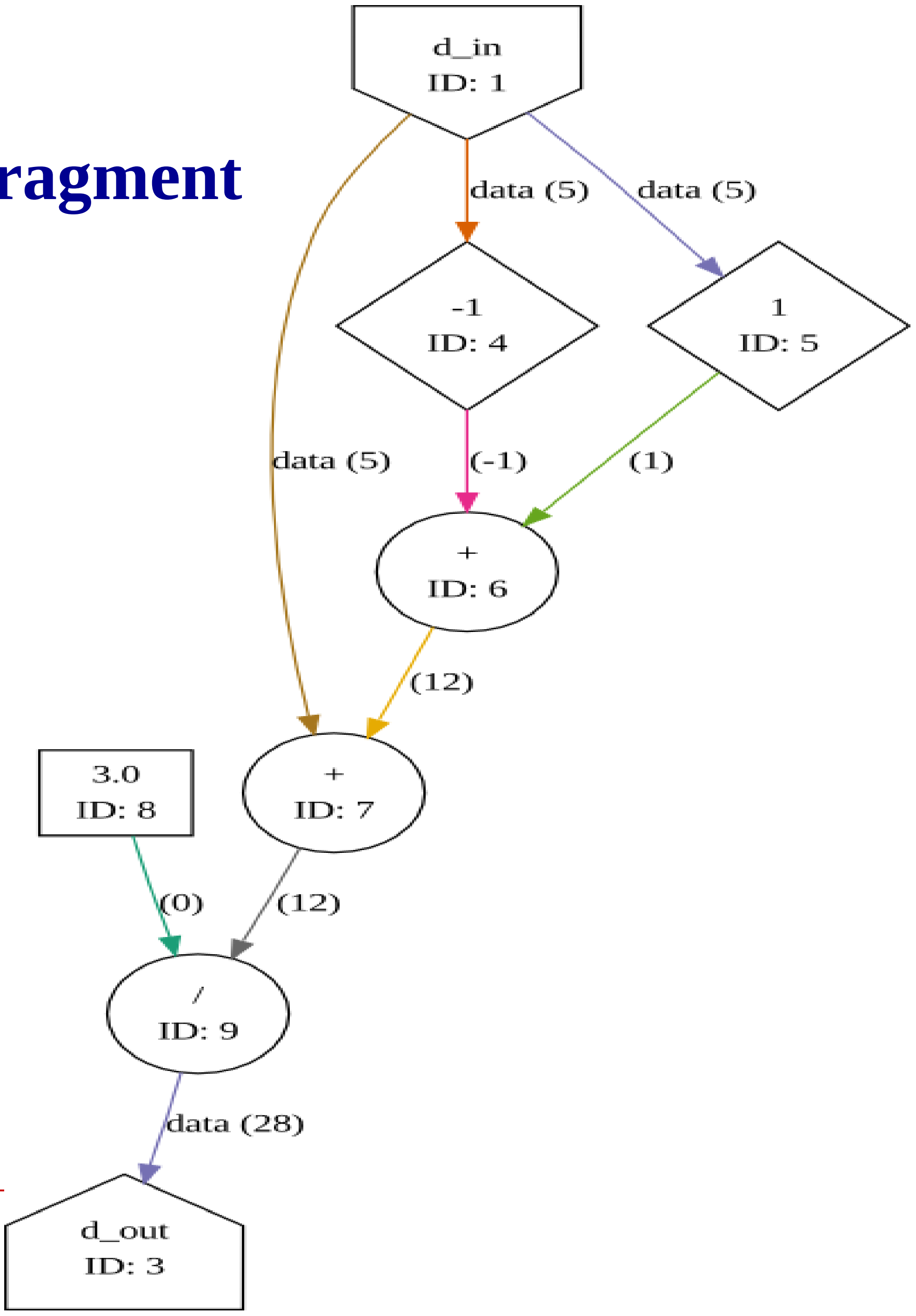


- First cut / accelerate a small vertical kernel / cylinder
- Later extend kernel size to achieve full application speedup

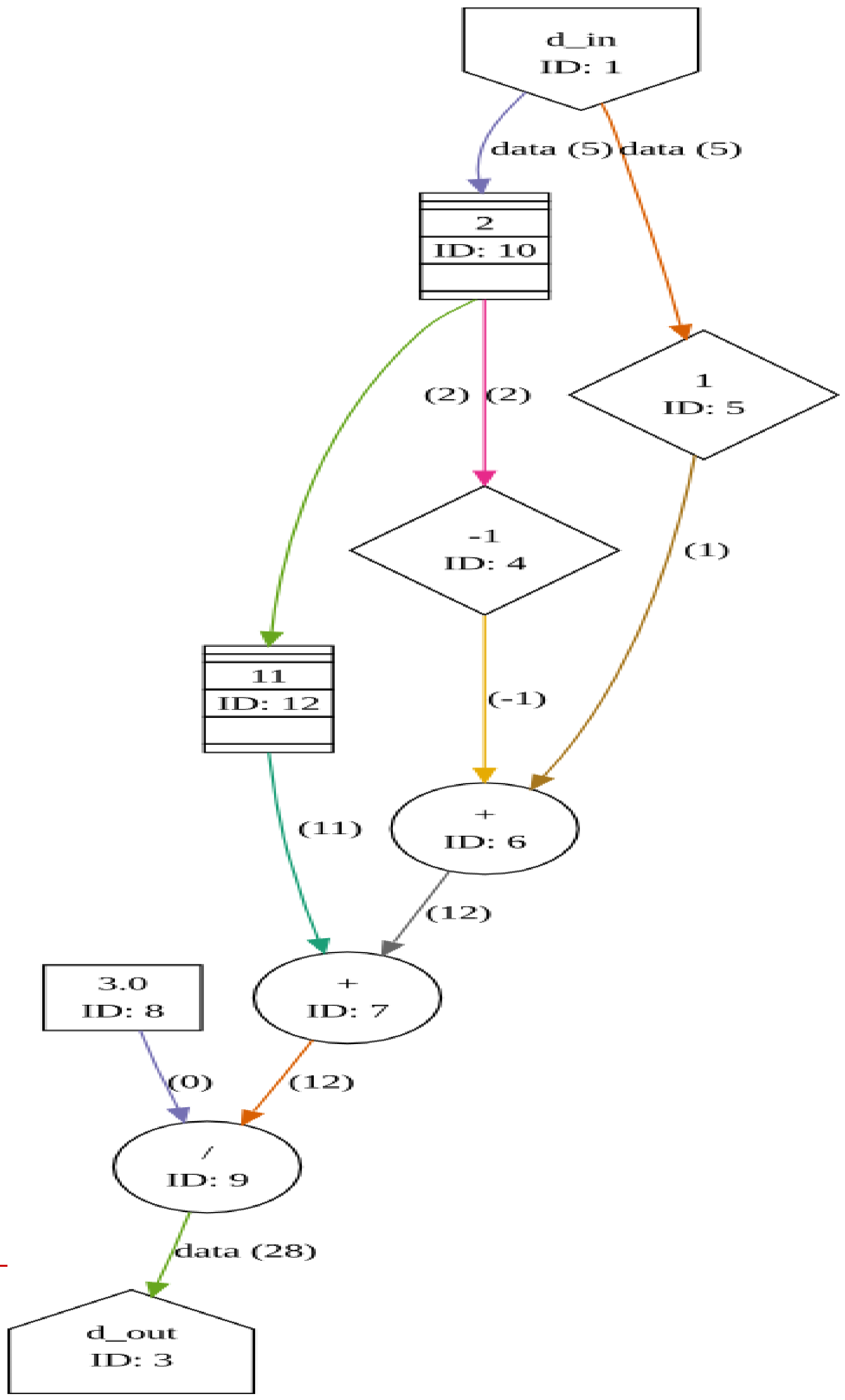
Speedup with the Cylindrical Model

- Use JAVA to create the DFG.
- Transform application to execute multiple simultaneous DFMs using DRAM “pipes”
- Stream computations through each pipe using memory choreography
- DFM size limited by FPGA area and DRAM (and FPGA pin) bandwidth
  - Application specific data precision
    - multiplies FPGA area
    - multiplies DRAM bandwidth

Data flow graph fragment

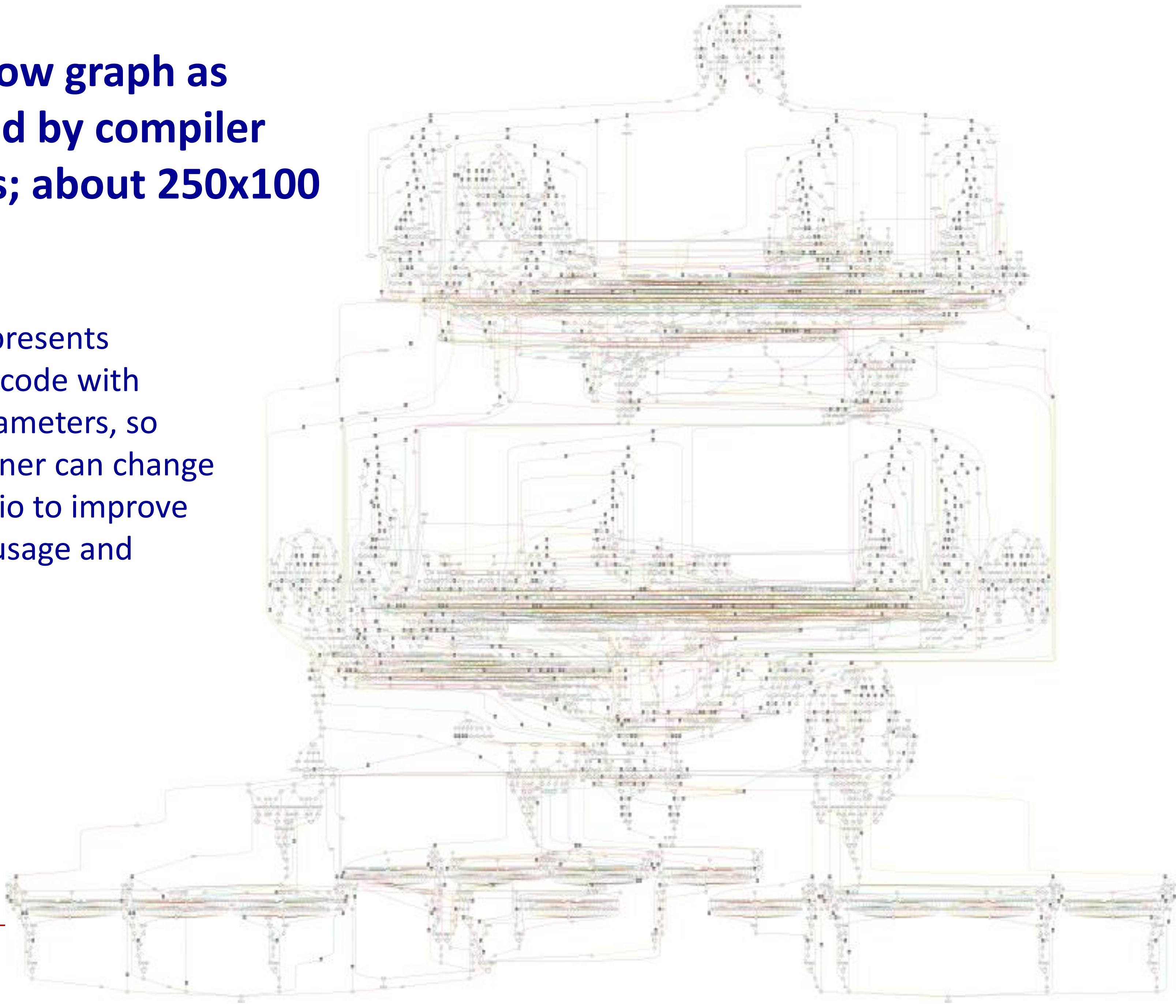


JAVA based DF graph description.
Automatic generation /
compilation creating DFM
buffer synchronized



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

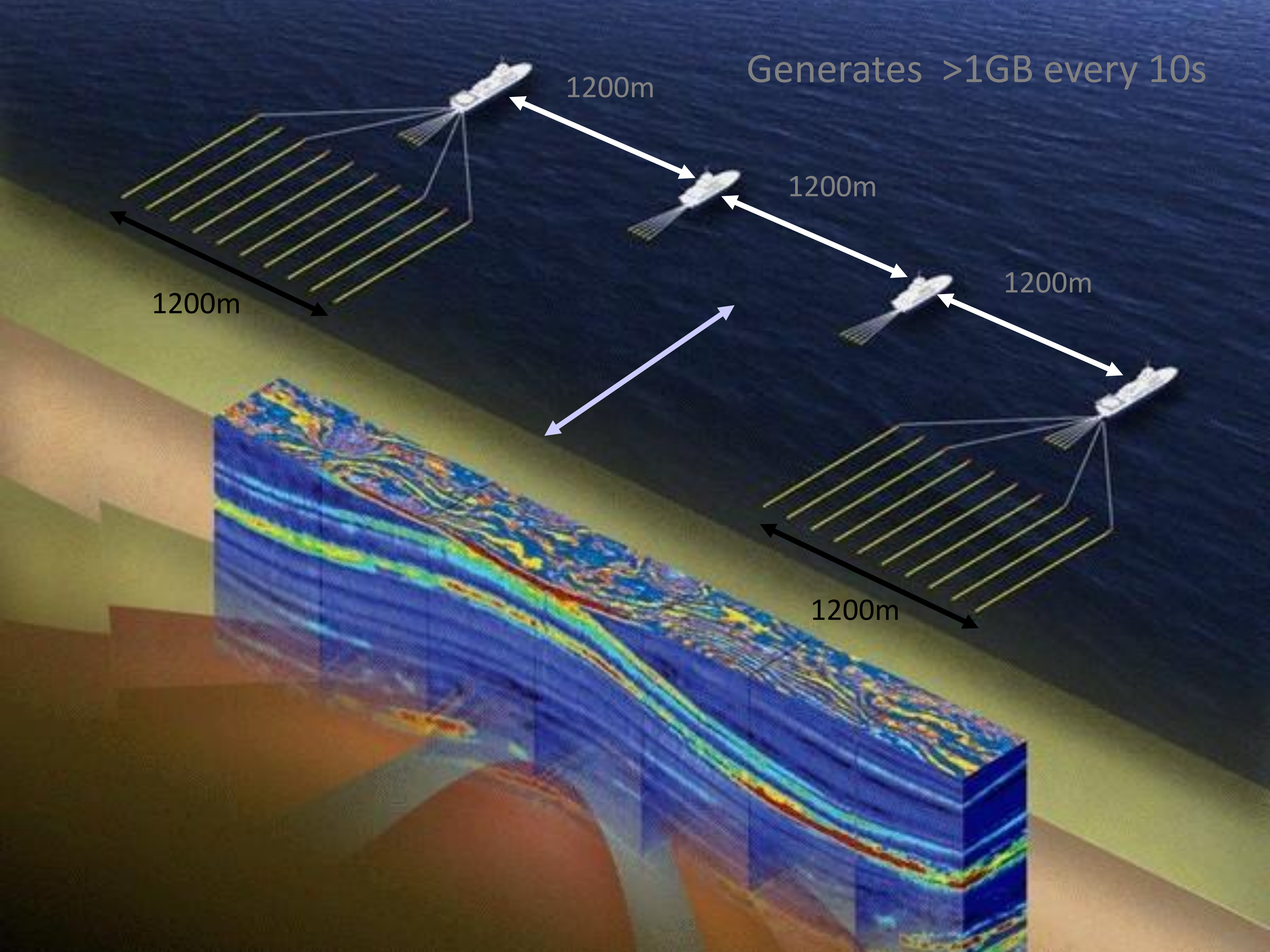


Example: Seismic Data Processing

- For Oil & Gas exploration:
  distribute grid of sensors over large area
- Sonic impulse the area and record reflections:
  frequency, amplitude, delay at each sensor
- Sea based surveys use 30,000 sensors to record data (120 db range) each sampled at more than 2kbps with new sonic impulse every 10 seconds

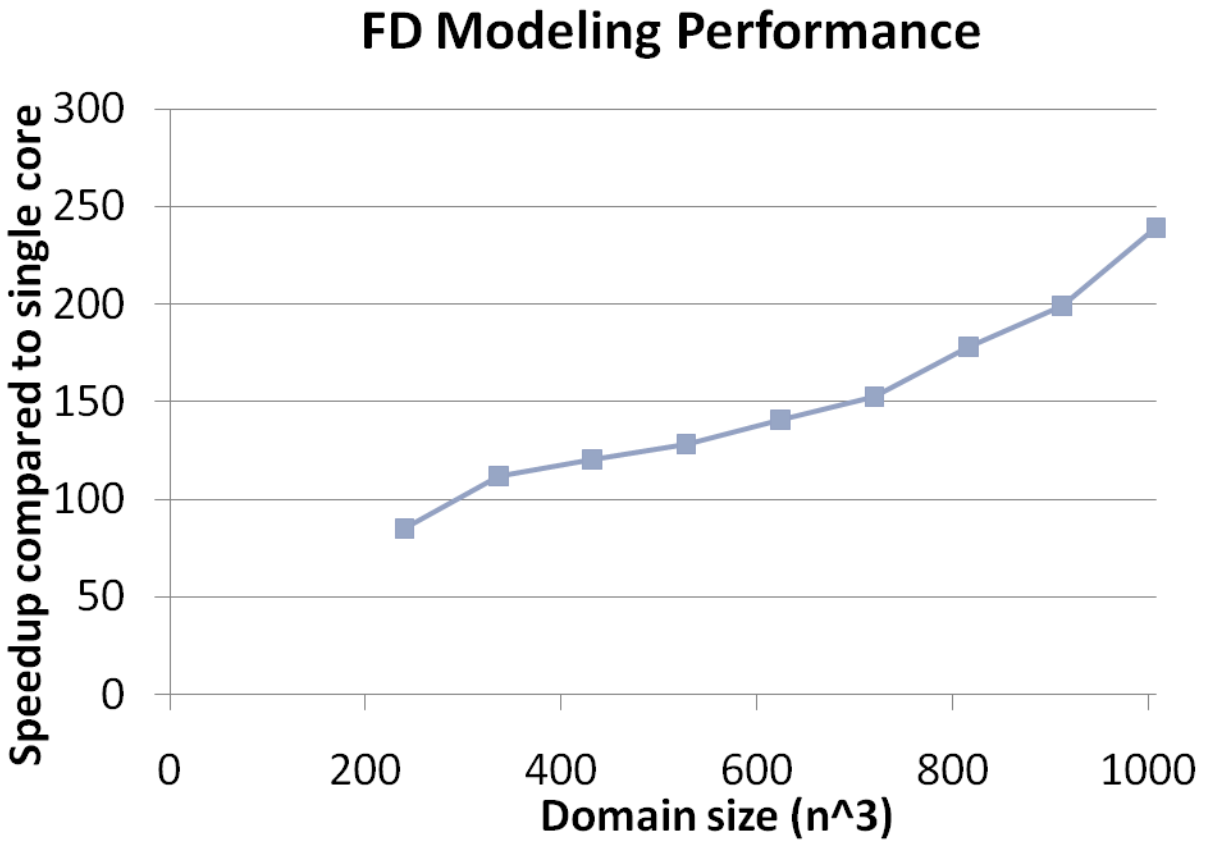
⇩

Order of terabytes of data each day

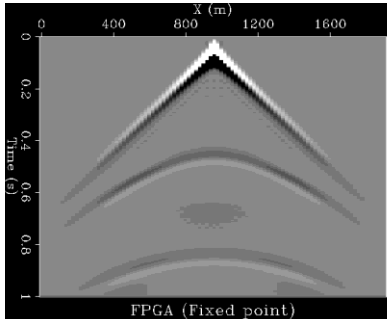

Generates >1GB every 10s
1200m
1200m
1200m
1200m
1200m

Modelling Results

- Up to 240x speedup for 1 MAX2 card compared to single CPU core
- Speedup increases with cube size
- 1 billion point modelling domain using single FPGA card

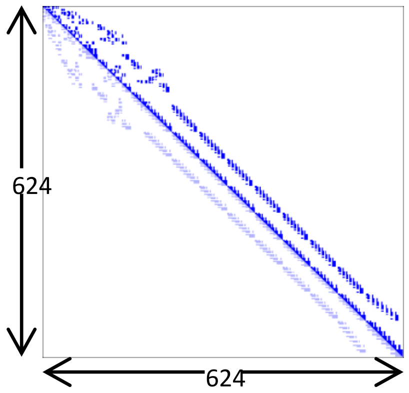


Achieved Computational Speedup for the entire application (not just the kernel) compared to Intel server



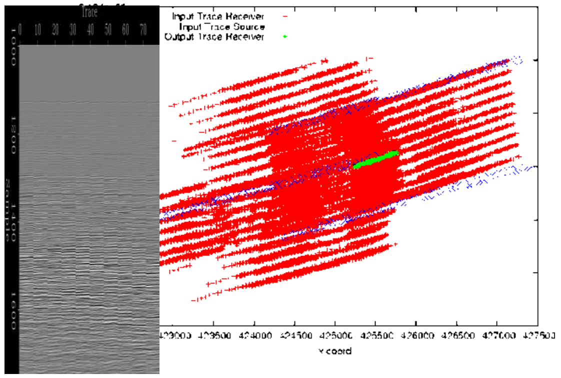
RTM with Chevron

VTI 19x and TTI 25x



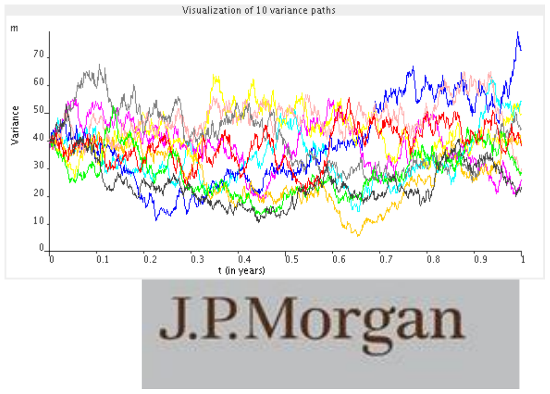
Sparse Matrix

20-40x

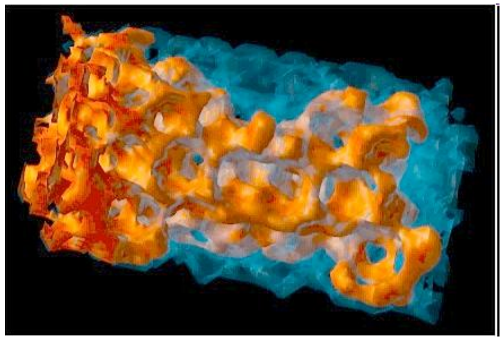


Seismic Trace Processing

24x

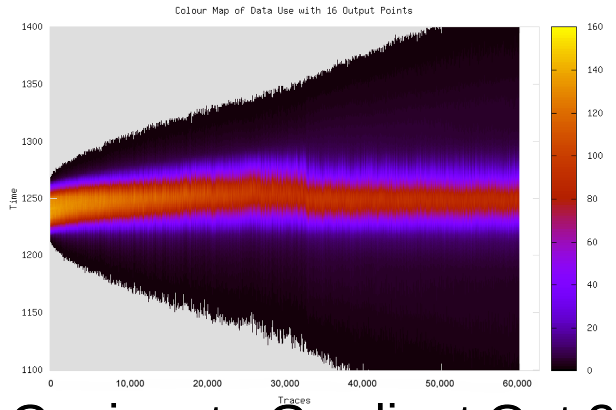


Credit 32x and Rates 26x



Lattice Boltzman

Fluid Flow 30x



Conjugate Gradient Opt 26x

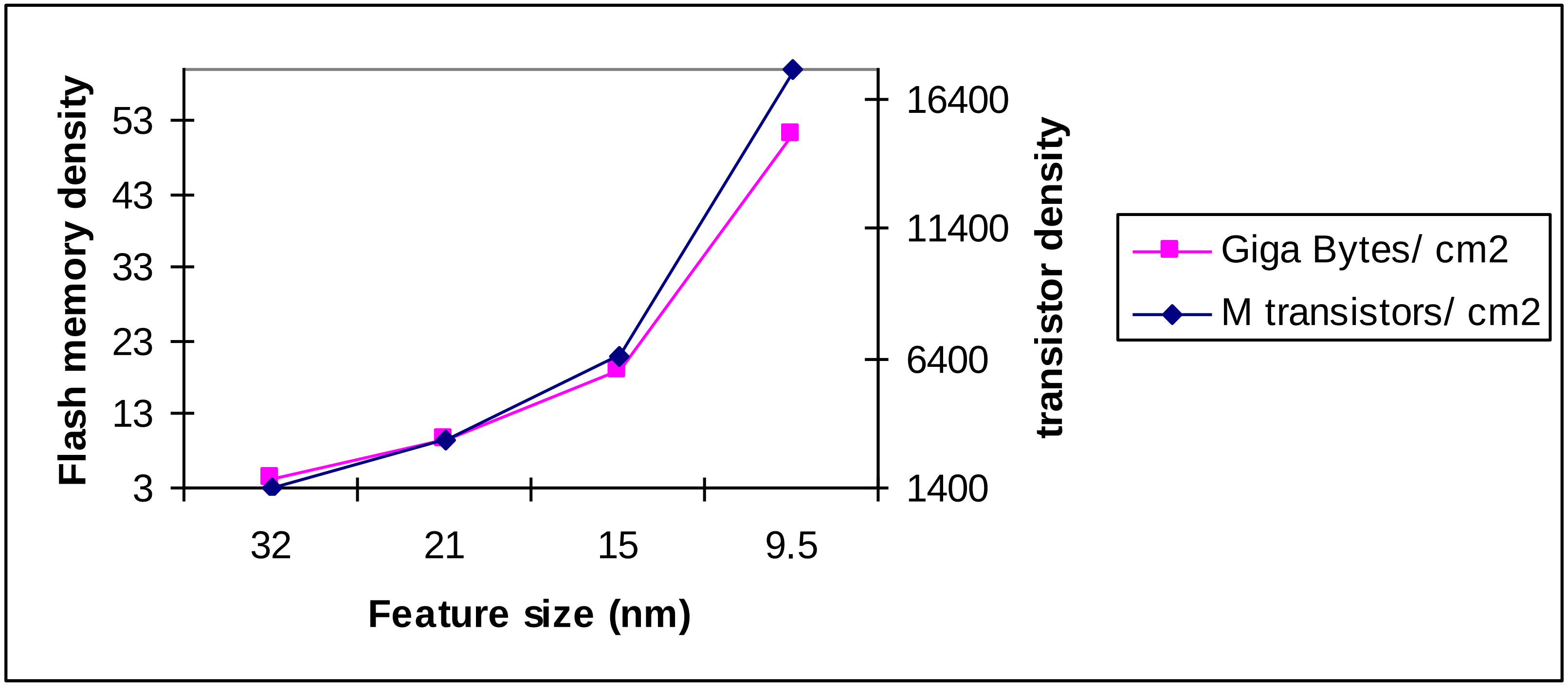
So for HPC, how can emulation (FPGA) be better than high performance x86 processor(s)?

- Multi core approach lacks robustness in streaming hardware (spanning area, time, power)
- Multi core lacks robust parallel software methodology and tools
- FPGAs emulate the ideal data flow machine
- 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 provide significant application speedup

And what's ahead for FPGAs and parallel processors?

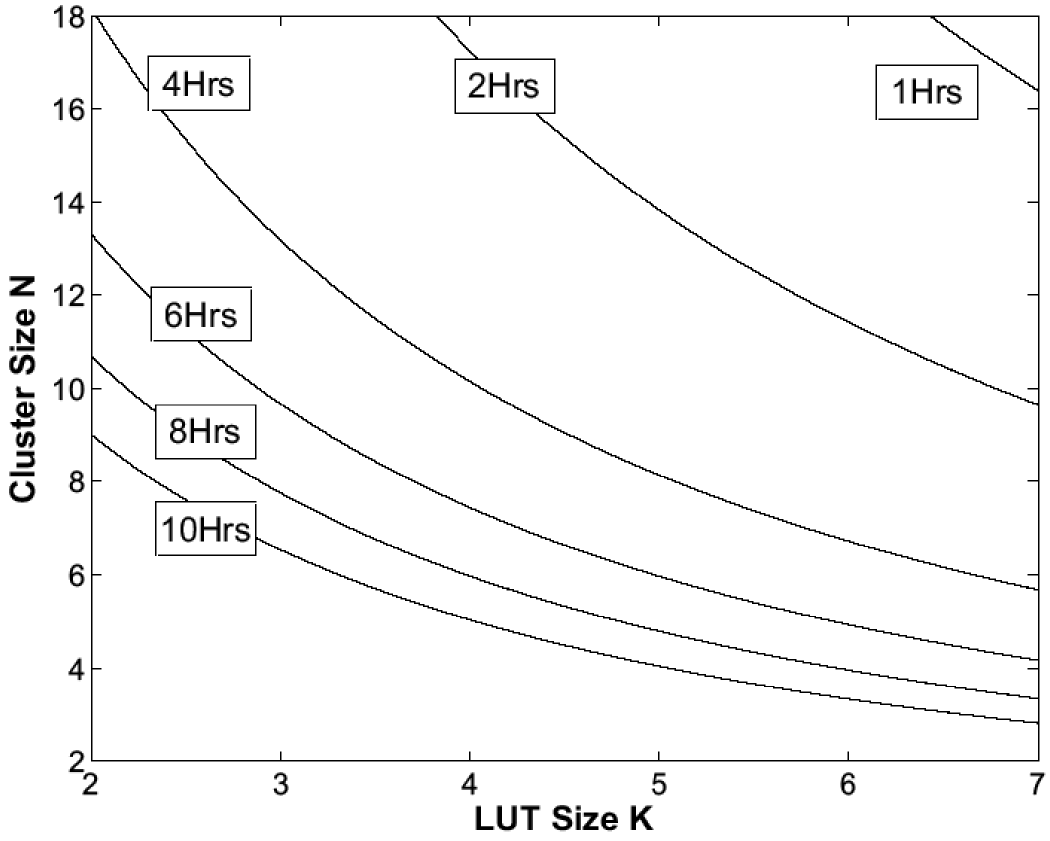
- Moore's Law for FPGA hardware
- May's Law for software
- Achieving convergence

Silicon device density scaling (ITRS 10 year projections)



Net: there's either 20 billion transistors or 50 Giga Bytes of Flash on a 1cm^2 die

BUT, what about place and route?



The more clusters to place and route, the more the runtime

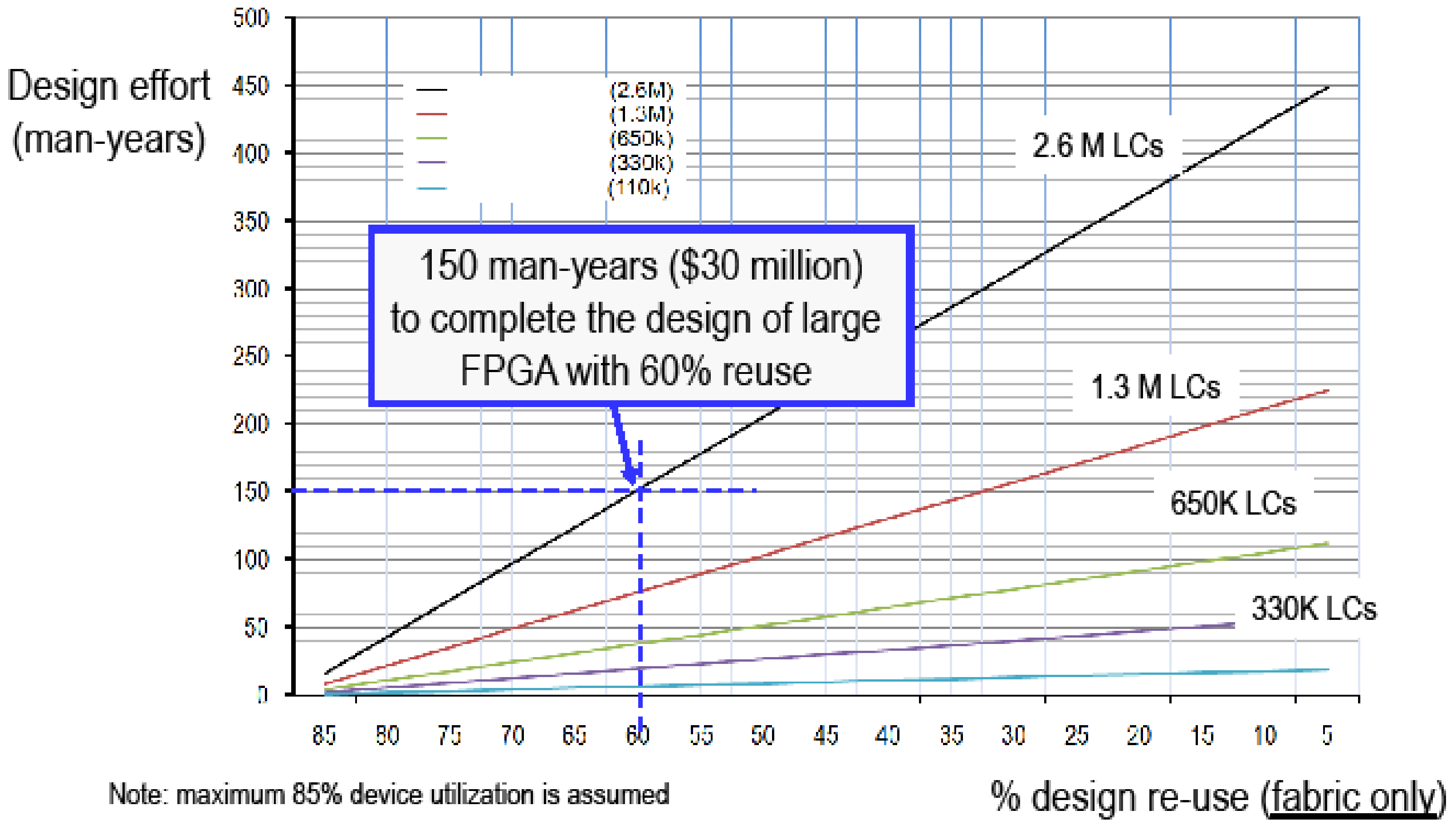
Runtime constraints for P&R time

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

And what about FPGA programming?

- Using Xilinx's model of effort for current design practice, effort depends on % design reuse and design size.
- *Data & next slide from presentation of Ivo Bolsens at FPGA 2012 symposium*

Model of FPGA HW customer design effort



Note: maximum 85% device utilization is assumed

XILINX

And then there's 3 problems: programming, programming & programming

- Efficient place and route
- Efficient FPGA implementation models
- New thinking about parallel systems:
  - Algorithms
  - Analytics (data structures, etc)
  - Program model

May's laws (for software)

- May's first law: *Software efficiency halves every 18 months, exactly compensating for Moore's Law*

- May's second law: *Compiler technology doubles efficiency no faster than once a decade*

David May, 2005

Moore’s vs May’s laws

- Proactive vs. reactive
- Imagine if we had proactive software law
- “The compiler will double speedup every 18 months.”
- And speedup is not even a well defined metric

Convergent software

- Need for a transparent framework to integrate disparate resources with different programming models.

  - sequential model

  - multi core shared memory

  - accelerator based (vector, etc)

  - dataflow based

With a definite attention to speedup!

Conclusions 1

- Parallel Processing demands rethinking algorithms, programming approach and environment and hardware.
- 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 some applications
- The automation of acceleration is still early on; still required: tools, methodology for writing apps., analysis methodology and (maybe) a new hardware basis
- For FPGA success software is key: VHDL, inefficient place and route, SW are big limitations

Conclusions 2

- In parallel processing: to find success, start with the problem not the solution.
- There's a lot of research ahead to effectively create parallel translation technology.

MAX-UP

MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

University Program

Giving universities and research labs access to state of the art dataflow computing technology with free software and subsidised hardware.

Visit our stand at FPL for more information

Or attend our talk: Dataflow Programming with MaxCompiler
Industry Session 2, 11:15 on Thursday August 30th

Thank you