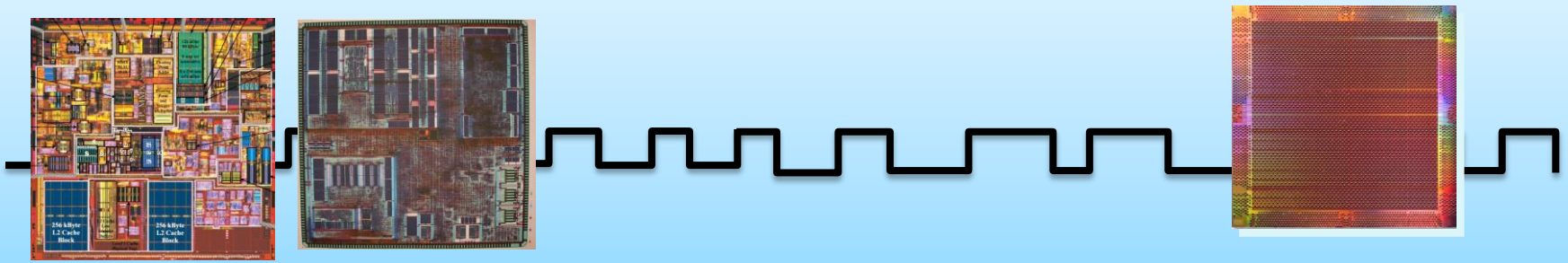
Compiling OpenCL to FPGAs : A Standard and Portable Software Abstraction for System Design

Deshanand Singh
Supervising Principal Engineer
Altera Toronto Technology Center

ALTERA®

Programmable Solutions: 1985-2002

- Technology scaling favors programmability



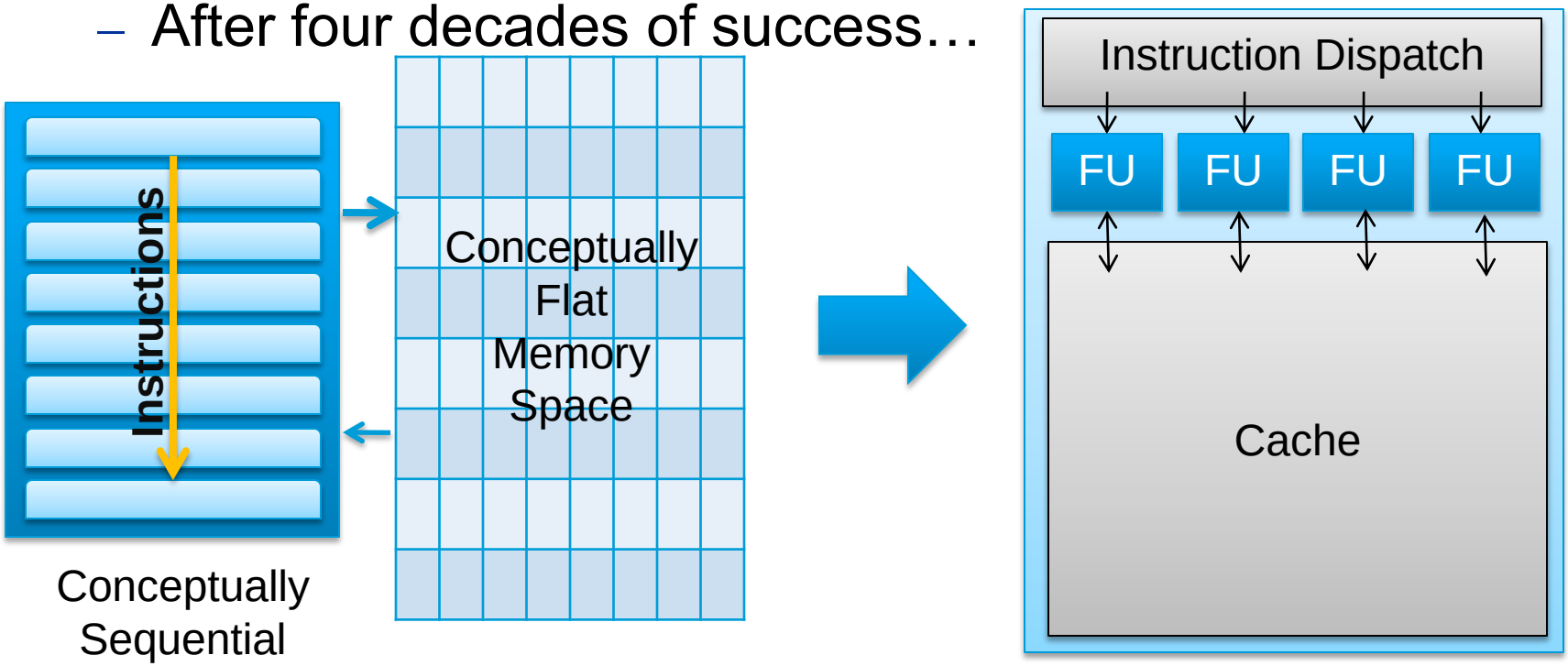
CPUs DSPs FPGAs

Single Cores

Fine-Grained Arrays

Programmable and *Sequential*

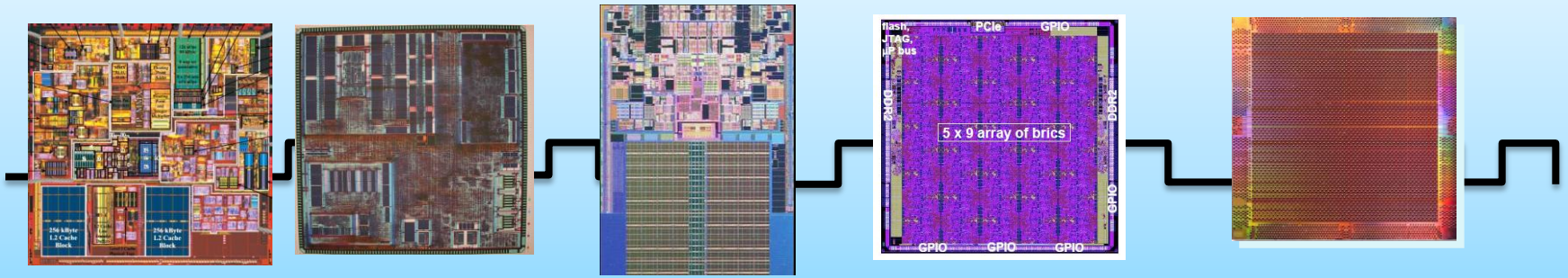
- Single core microprocessor
- Reaching the limit
  - After four decades of success…



Large, Power Hungry hardware is necessary for the comforts of sequential programming models

Programmable Solutions: 2002-20XX

- Technology scaling favors programmability and parallelism

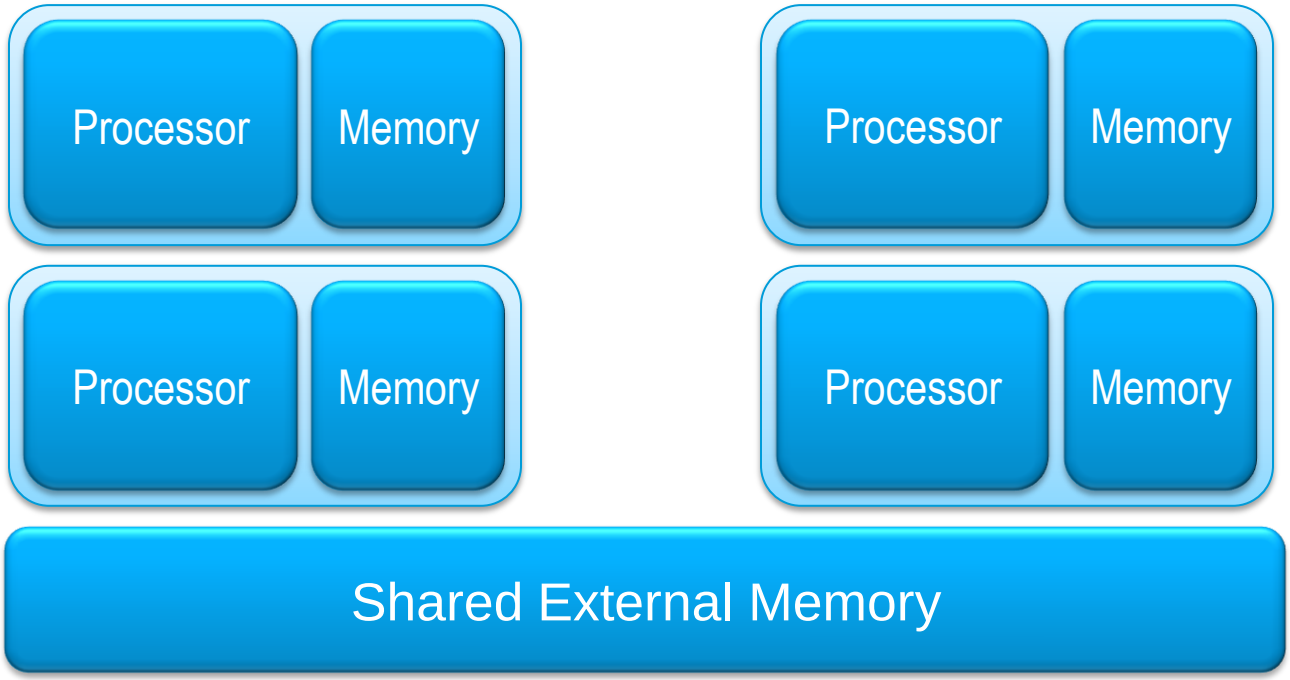


CPUs → DSPs → Multi-Cores → Arrays → FPGAs

Single Cores	Multi-Cores Coarse-Grained CPUs and DSPs	Coarse-Grained Massively Parallel Processor Arrays	Fine-Grained Massively Parallel Arrays

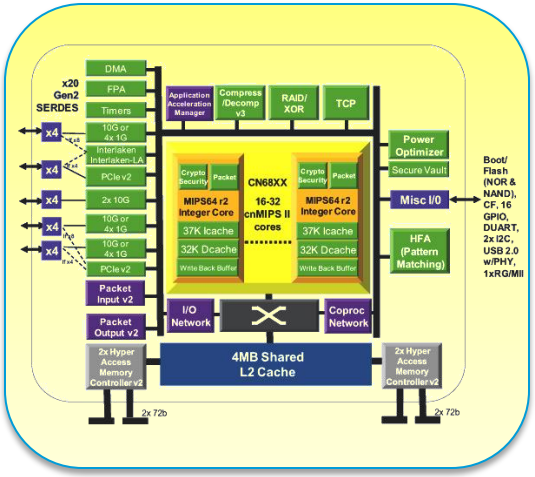
Programmable and *Parallel*

- Exploit parallelism on a chip
  - Take advantage of Moore's law
    - Processors not getting faster, just wider
  - Keep the power consumption down
- Use more transistors for information processing
  - ***Programmers required to code in an explicitly parallel fashion***

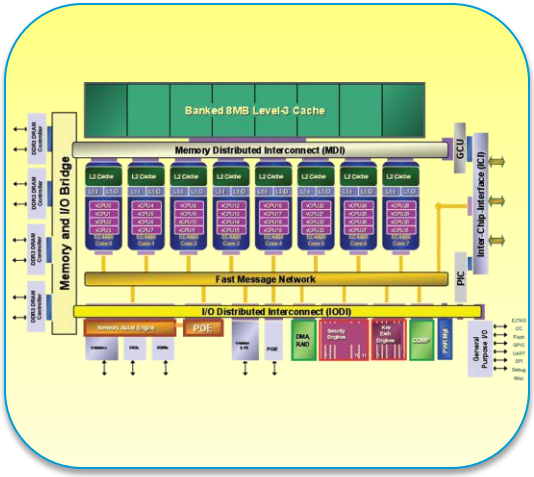


Multicore Devices

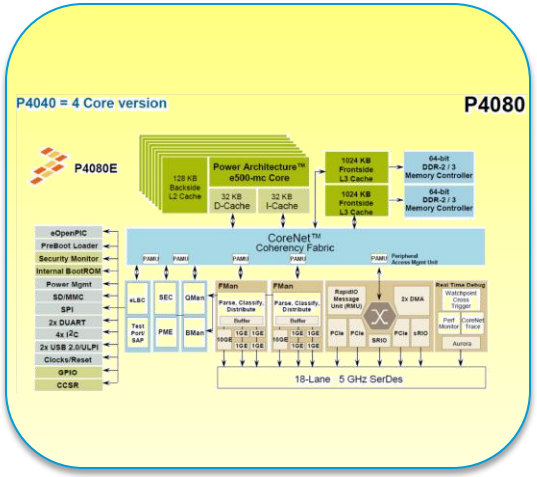
Cavium



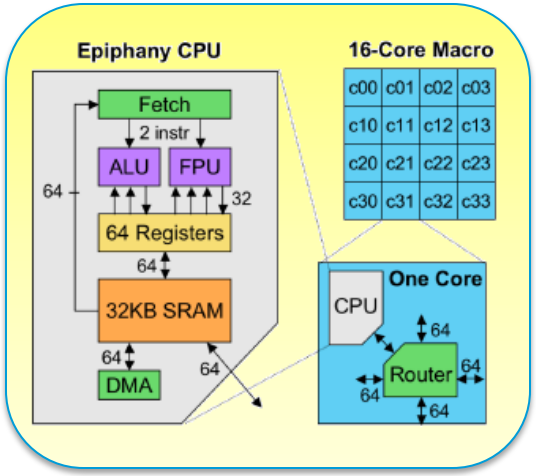
NetLogic



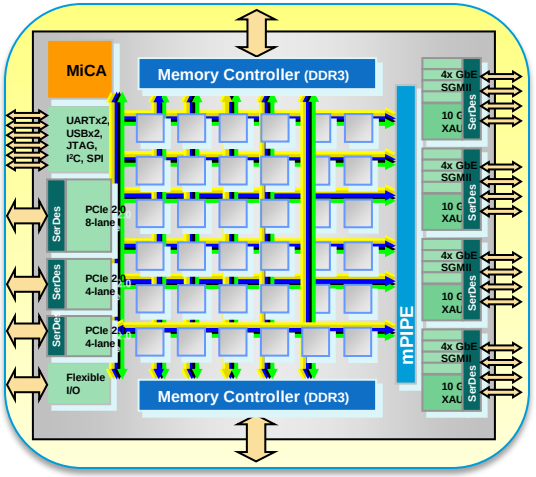
Freescale



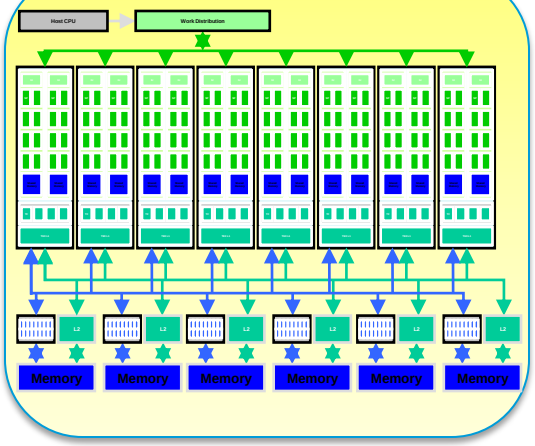
Adapteva



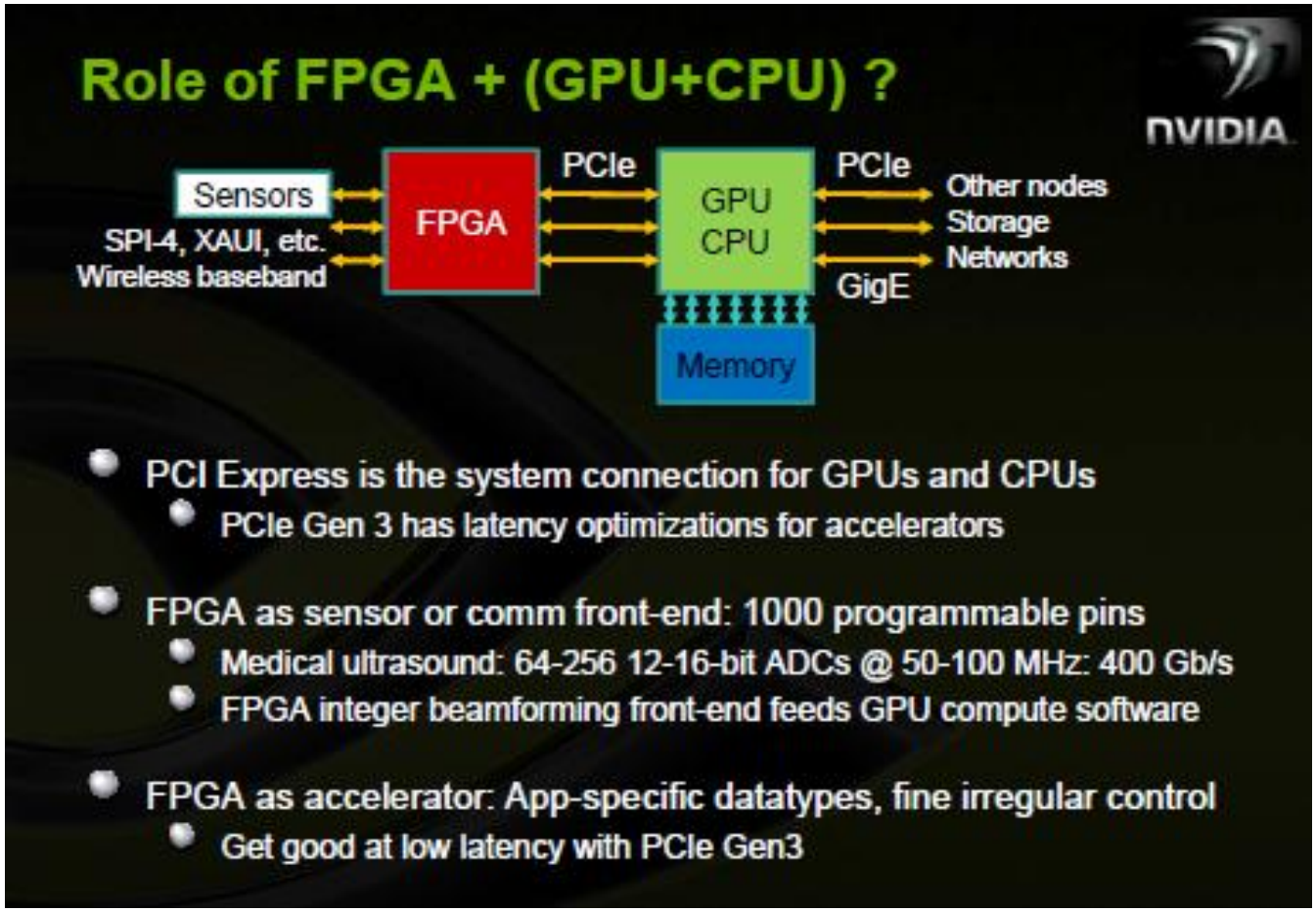
Tilera



NVIDIA



The Role of FPGAs [FPGA'2011]



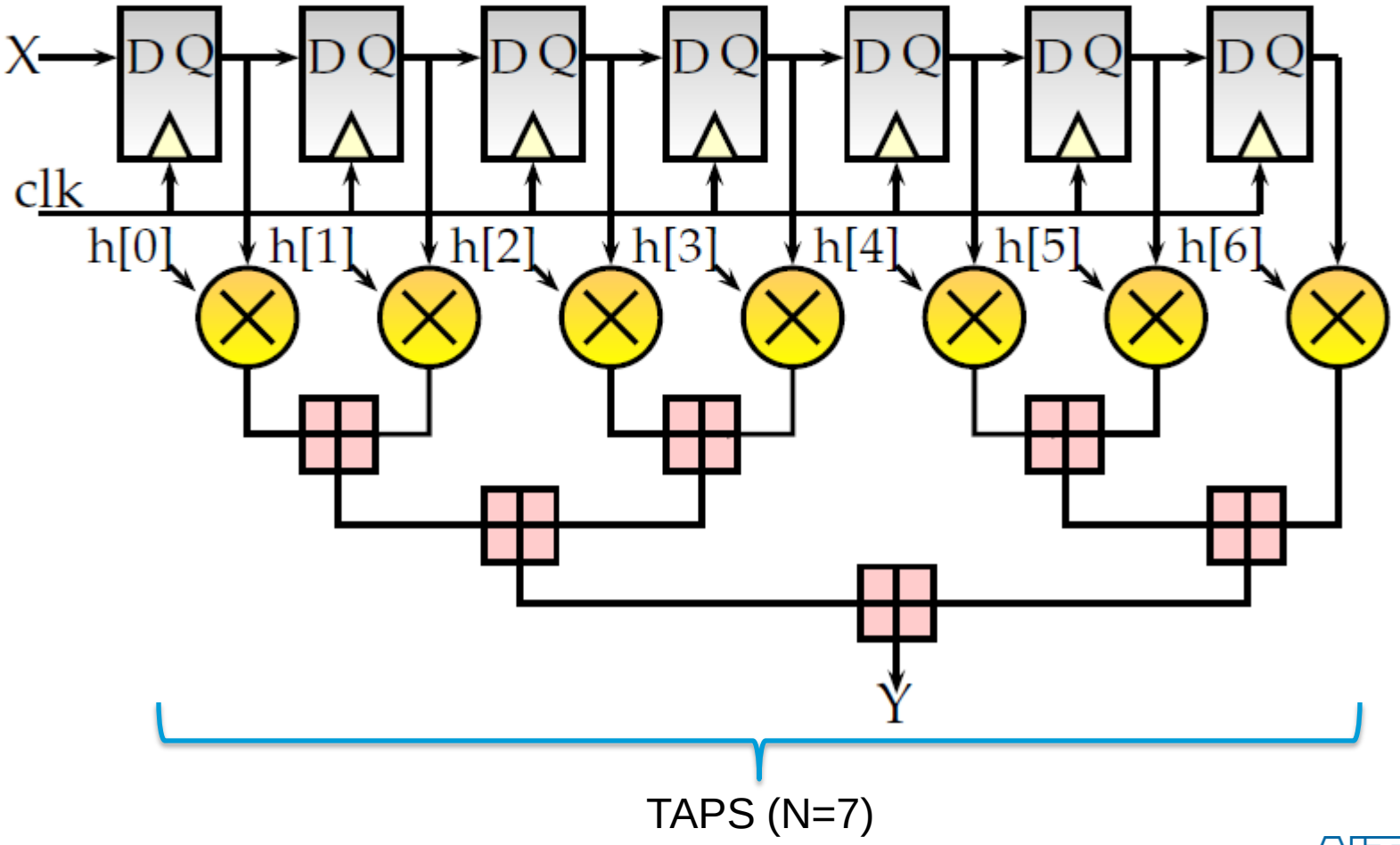
- ***FPGA Flexible IO used to bring data into GPUs / CPUs for algorithmic processing***

Source: http://www.eecg.toronto.edu/~jayar/fpga11/FPGA2011PreConfWorkshop.htm

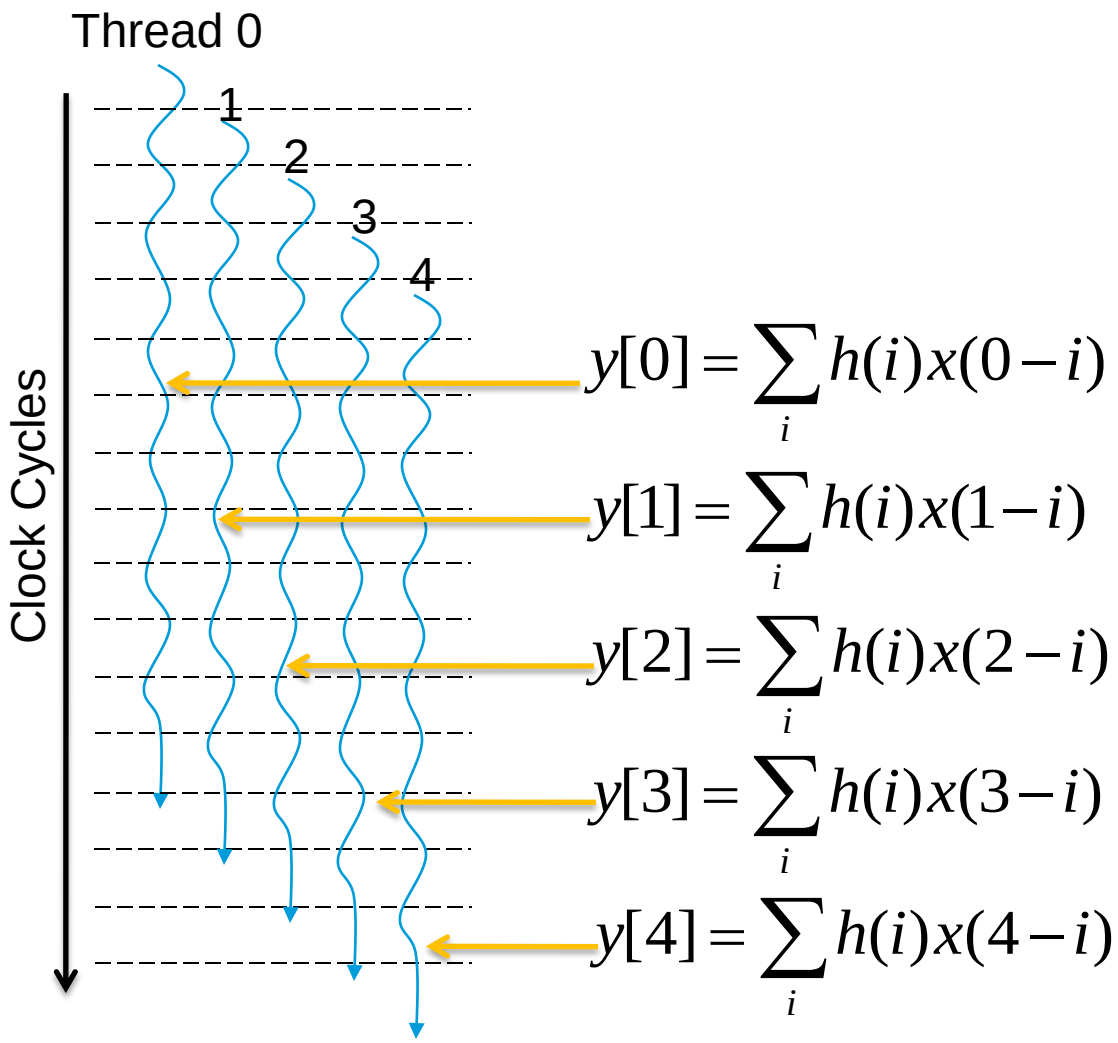
MOTIVATING CASE STUDY :
FINITE IMPULSE RESPONSE (FIR) FILTER

FIR Filter Example

$$y(n) = \sum_{i=0}^{N-1} h(i)x(n-i)$$

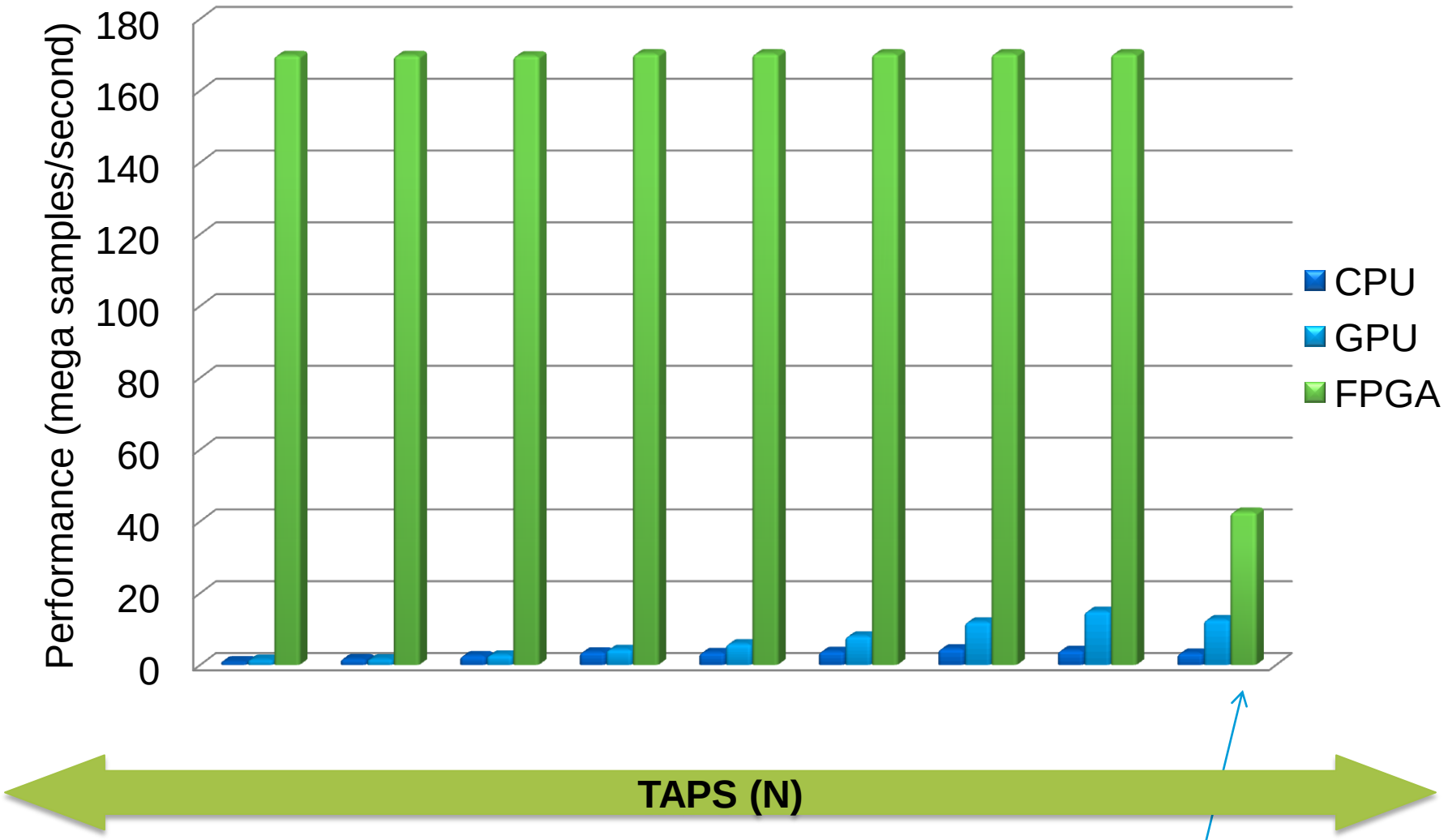


Custom Multithreaded Pipeline

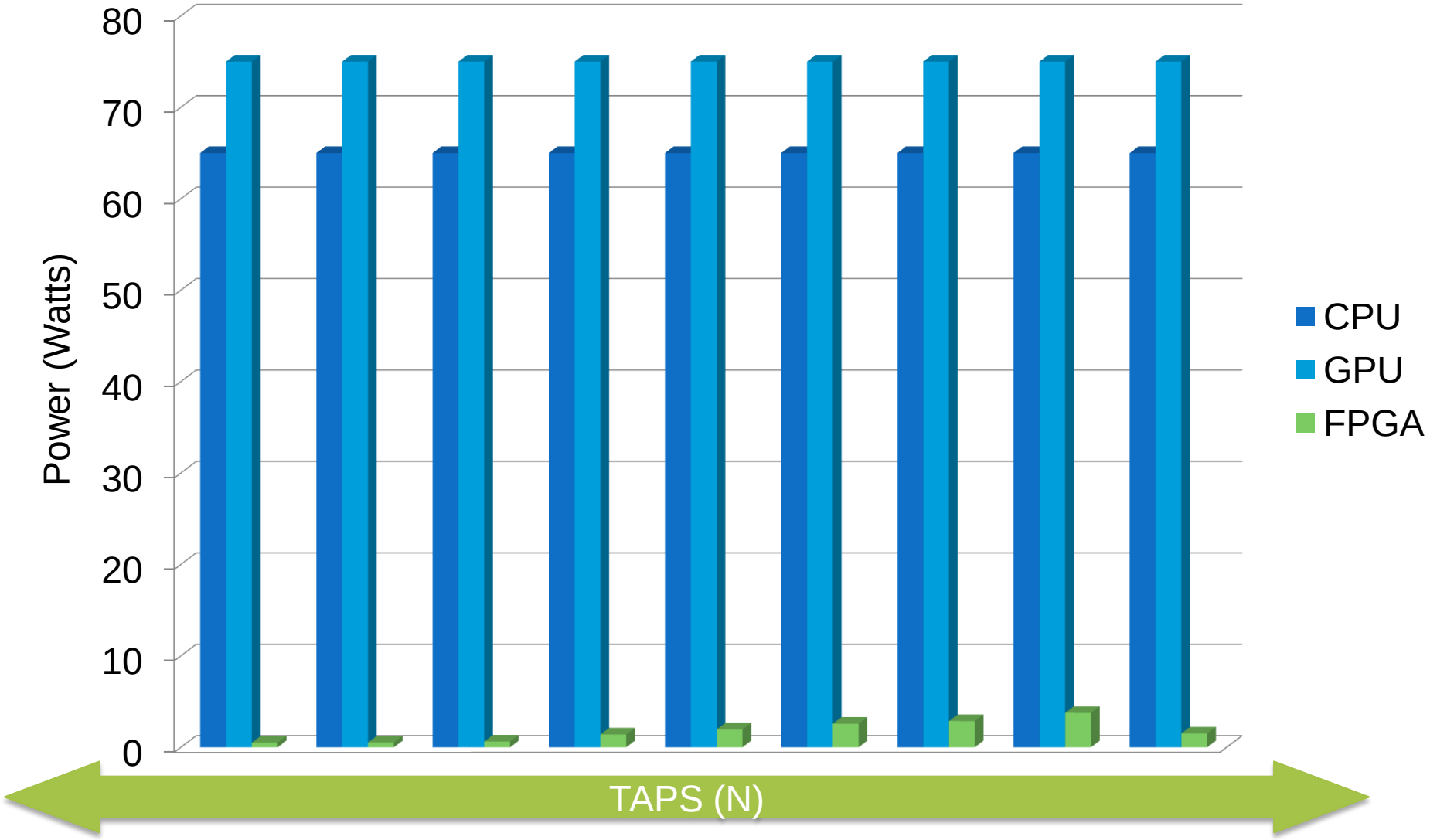


- Throughput of 1 thread per cycle is possible using a direct HW implementation
- FPGA offers custom pipeline parallelism which can be perfectly tailored to the FIR filter

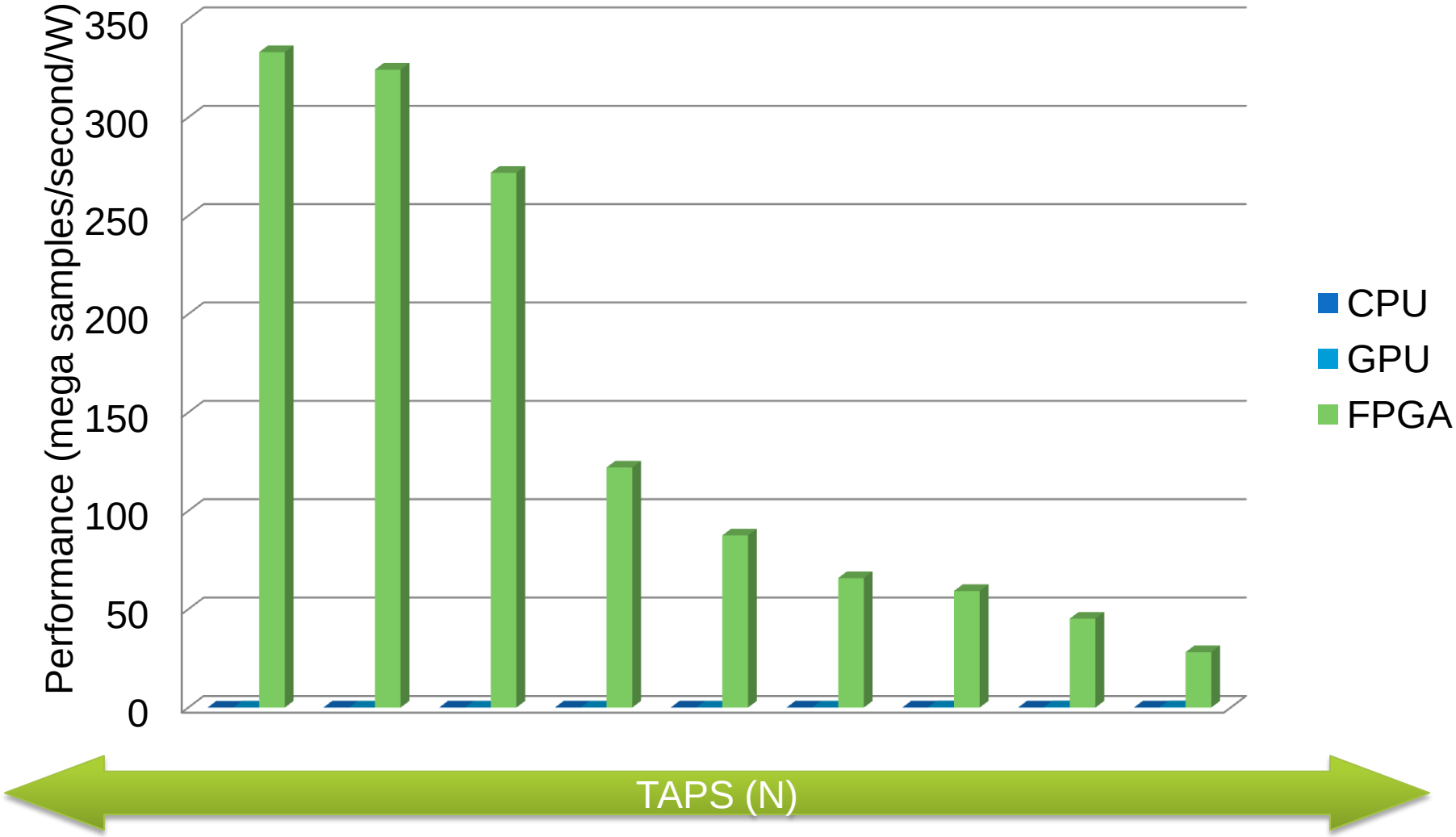
Absolute Performance Comparison (FIR)



Power Comparison (FIR)



Performance-to-Power Ratio (FIR)



FPGAs for Computation

- Although the FIR filter is a simple example, it is representative of a large class of applications
  - Large amounts of spatial locality
  - Computation can be expressed as a feed forward pipeline
- The fine grained parallelism of the FPGA can be used to create custom “processors” that are orders of magnitude more efficient than CPUs or GPUs

Future Forecast

Incorporation or impact of new technologies on HPC in next two years	Users Said (N = 25) (1 = very unlikely; 5 = very likely						Vendors Said (N = 30-33) (1 = very unlikely; 5 = very likely					
Component	1	2	3	4	5	Avg.	1	2	3	4	5	Avg.
GPU based acceleration	8%	16%	4%	32%	40%	3.80	0%	6%	9%	28%	56%	4.34
FPGU based acceleration	32%	16%	32%	16%	4%	2.44	13%	37%	20%	20%	10%	2.77
Solid State Disks	4%	20%	28%	32%	16%	3.36	3%	3%	31%	31%	31%	3.84
New processor architectures	8%	12%	16%	36%	28%	3.64	3%	16%	22%	34%	25%	3.63
New programming models	4%	4%	28%	48%	16%	3.68	6%	13%	25%	31%	25%	3.56
Optical interconnect technology	8%	12%	28%	40%	12%	3.36	6%	18%	24%	39%	12%	3.33
Optical networking technology	4%	24%	20%	44%	8%	3.28	6%	26%	26%	26%	16%	3.19
Optical processors	16%	36%	24%	16%	8%	2.64	34%	19%	28%	9%	9%	2.41
Next generation networking technology	4%	12%	44%	20%	20%	3.40	6%	9%	27%	42%	15%	3.52
Cloud computing	20%	12%	28%	24%	16%	3.04	0%	19%	19%	47%	16%	3.59

Source: Intersect360 Research Report, 2010

- Vendors see GPU acceleration as having a dramatic impact on HPC in the next two years
- *FPGAs could not even be spelled correctly*

Why not FPGAs?

- David Mayhew, AMD fellow [IMA: High Performance Computing and Emerging Architectures]
  - "*Availability of cost-effective, massively-parallel, floating-point and scalar accelerators crucial to many HPC workloads*
    - *I apologize profusely to the FPGAs crowd who believe that FPGAs are going to somehow become relevant in this space, but I believe that it is GPUs or nothing (nothing meaning that general-purpose processors do everything).*
      - ***FPGAs have been 3 years away from being standard system components for the last 10 years and will be for the next 10 years***
      - *Die-stacking may affect this bit of cynicism/pessimism*
        - *A layer of FPGA in a standard, vertical processor/memory stack may make FPGAs inexpensive enough and generally useful enough to achieve general system integration*"

Source: http://www.ima.umn.edu/2010-2011/W1.10-14.11/activities/Mayhew-David/Minn-Jan-11%5B2%5D.pptx

THE COMPETITION

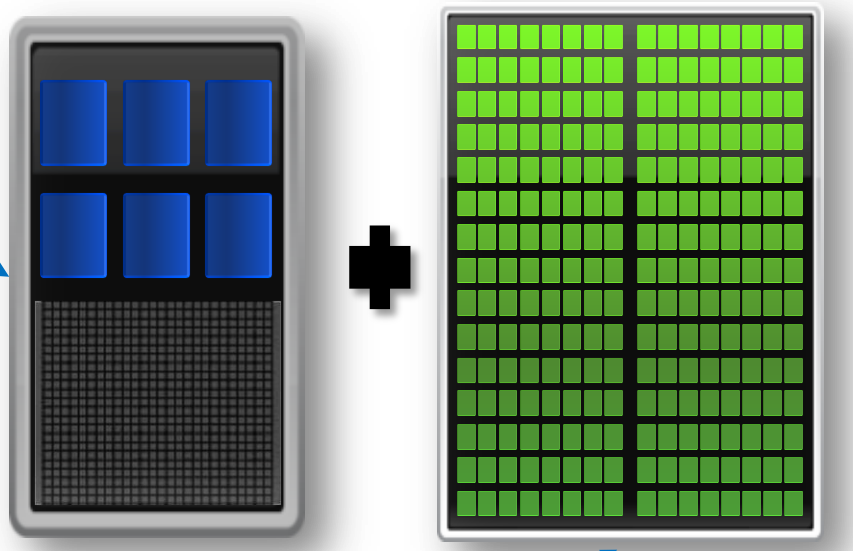
CPUs & GPUs

OpenCL/CUDA Application Code

```
void coarse_grained_serial (… ) {
  ...
}
```

```
void parallel_kernel(float ... )
{
 ...
}
```

```
void main( ) {
  ...
  parallel_function<<...>>(..);
  serial_function(..);
  ...
}
```

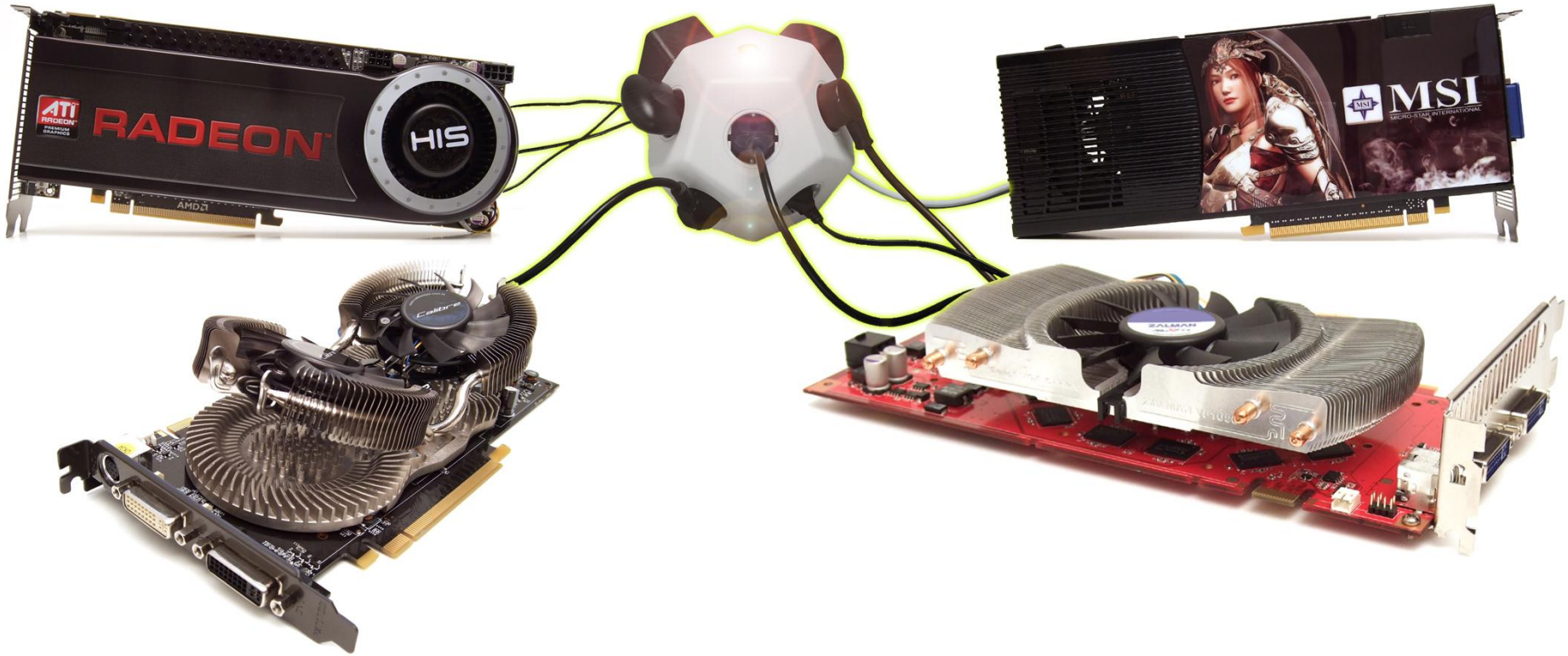


Heavy parallel workload on the GPU

Serial routines on the CPU

GPU Power Consumption

- High-end GPU cards may exceed 250W power requirements



Datacenter Applications

- A facility used to house a large farm of servers and associated components
  - Connectivity, storage and cooling
- FPGAs & GPUs can only reasonably address a specific class of datacenters:
  - High Performance Technical Computing (HPTC)
    - Scientific or Engineering Computations
  - High Performance Business Computing (HPBC)
    - Financial Calculations, Analytics
  - Almost 20% of the entire server market

Power & Cooling

- Power and cooling has become a top concern among HPC data centers
  - Energy prices have been hovering at near historic levels
  - Processor based design has increasingly come up against the power wall
    - More challenging to obtain higher single-core performance while maintaining reasonable power
  - Companies are increasingly sensitive about reducing their carbon footprint
    - The “Green Movement”

Source: IDC, 2010

Power & Cooling (2)

- HPC data centers' average per site:
  - Available floor space over 26,000 ft
  - Used floor space about 17,000 ft
  - Annual power consumption 6.3 MW
- Data centers costs
  - Annual power cost was $2.9 million or $456 per KW
  - Ten sites provided the percentage of their budget spent on power
    - Average was 23%
- Cooling Upgrades
    - Average amount budgeted is $6.87 million

Source: IDC, 2010

Power & Cooling (3)

- GPUs have massive power requirements
- Power and Cooling are one of the key datacenter drivers
- GPUs are seen as having a bright future in HPC servers
- *It doesn't add up*

CHALLENGES

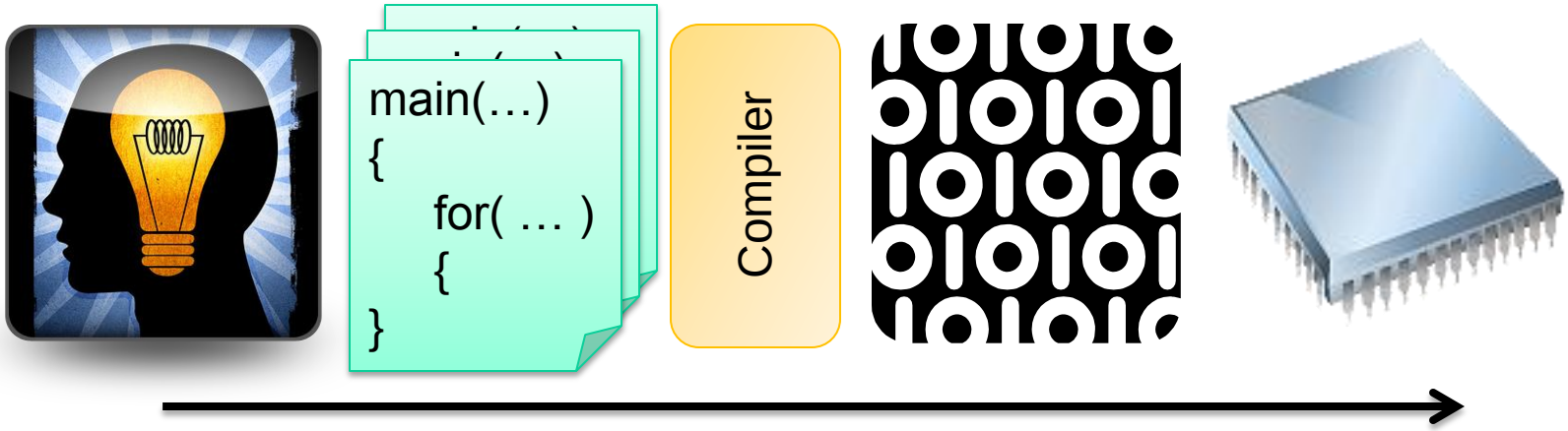
HPC GAPs

SATISFACTION GAP (Satisfaction score - Importance score) x 100	Users Said			Vendors Said		
	Sat.	Imp.	GAP	Sat.	Imp.	GAP
Processor core performance	3.75	4.19	-44	3.84	4.24	-40
Overall processor performance	3.79	4.54	-75	3.77	4.61	-84
Number of Processors per node	3.63	3.85	-22	3.97	3.91	6
Memory capacity (configurable memory per node)	3.58	4.04	-46	3.87	4.12	-25
Memory performance (latency and/or bandwidth)	3.17	4.58	-141	3.26	4.45	-119
Cluster interconnect bandwidth	3.71	4.23	-52	3.53	4.21	-68
Cluster interconnect latency	3.42	4.35	-93	3.56	4.56	-100
LAN networking bandwidth	3.65	3.27	38	3.63	3.09	54
LAN networking latency	3.65	3.00	65	3.59	3.03	56
Storage system capacity	3.58	3.73	-15	3.65	3.42	23
Storage system management software	3.00	3.16	-16	3.17	3.15	2
Parallel programming models	2.71	4.04	-133	2.94	4.30	-136
Parallel programming tools/environment	2.75	4.19	-144	2.87	4.03	-116
Average:	**3.41**	**3.94**	**-52**	**3.51**	**3.93**	**-42**

Source: Intersect360 Research Report, 2010

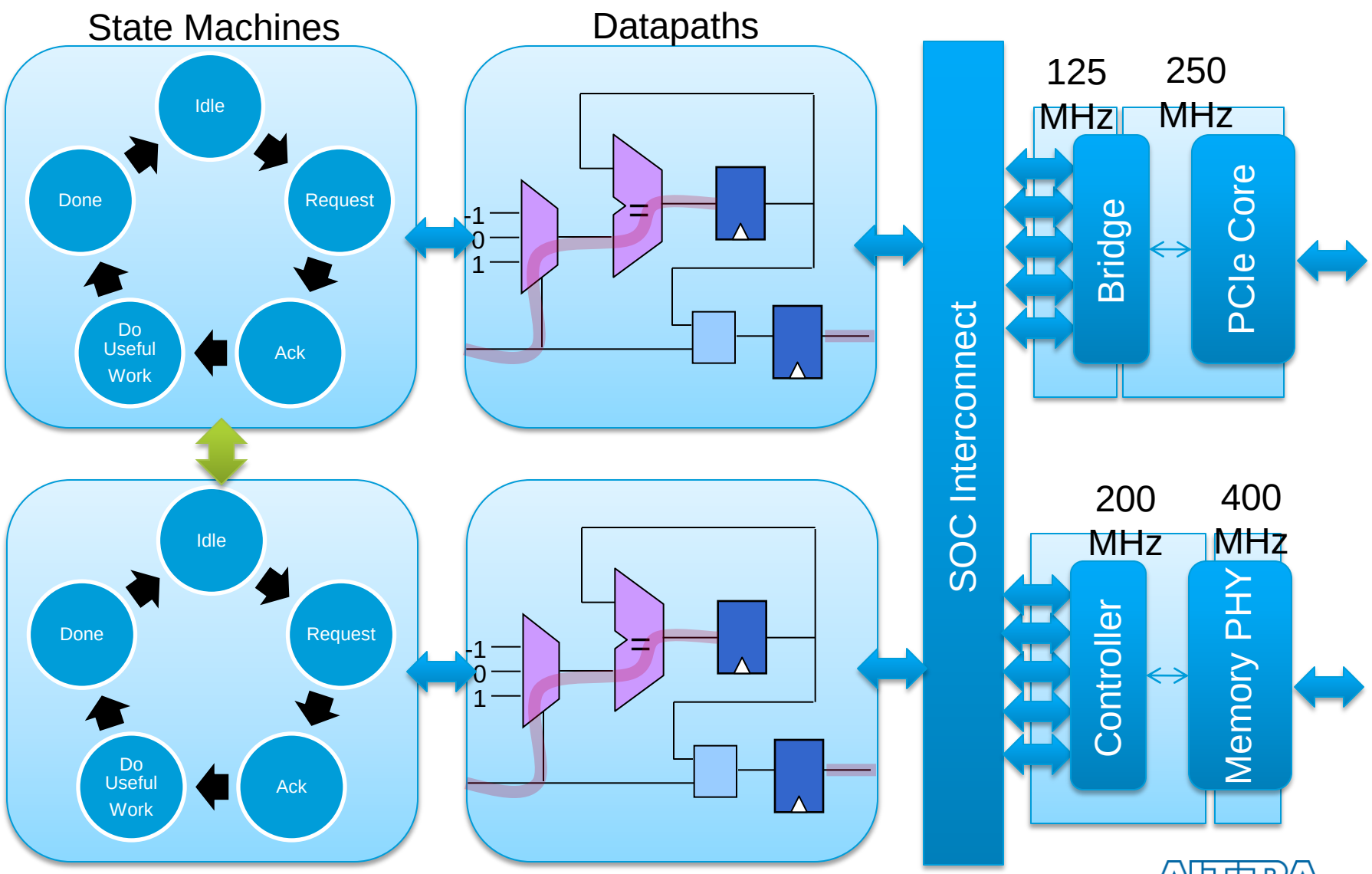
- Programming Models & Memory Performance highlighted as the largest GAPs

Software Programmer's View



- Programmers are used to software-like environments
  - Ideas can easily be expressed in languages such as 'C'
    - Typically start with simple sequential program
    - Use parallel APIs / language extensions to exploit multi core for additional performance.
  - Compilation times are almost instantaneous
  - Immediate feedback and rich debugging tools

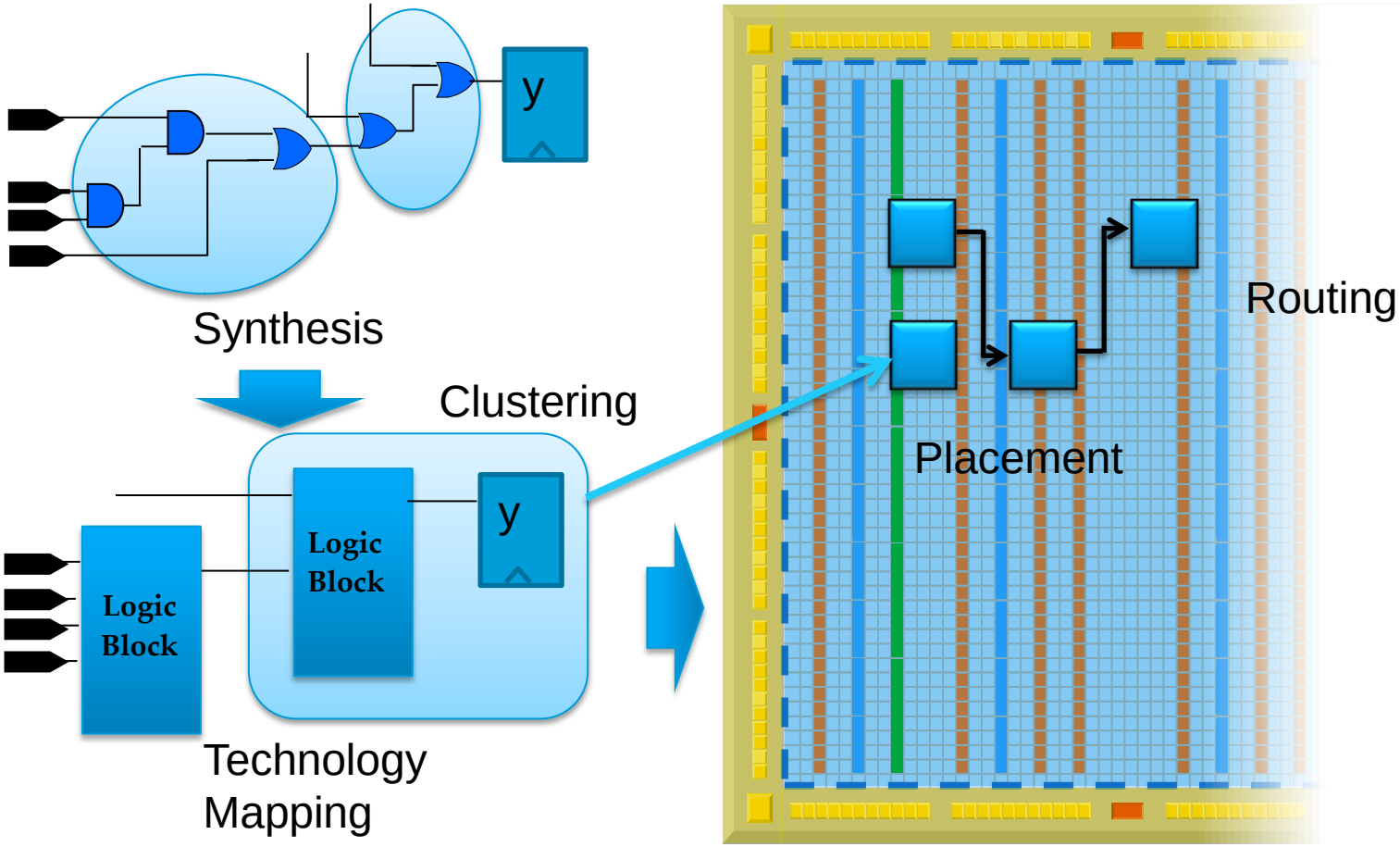
FPGA Hardware Design



Design Entry Complexity

- Description of these circuits is done through Hardware Design Languages such as VHDL or Verilog
- Incredibly detailed design must be done before a first working version is possible
  - Cycle by cycle behavior must be specified for every register in the design
  - The complete flexibility of the FPGA means that the designer needs to specify all aspects of the hardware circuit
    - Buffering, Arbitration, IP Core interfacing, etc

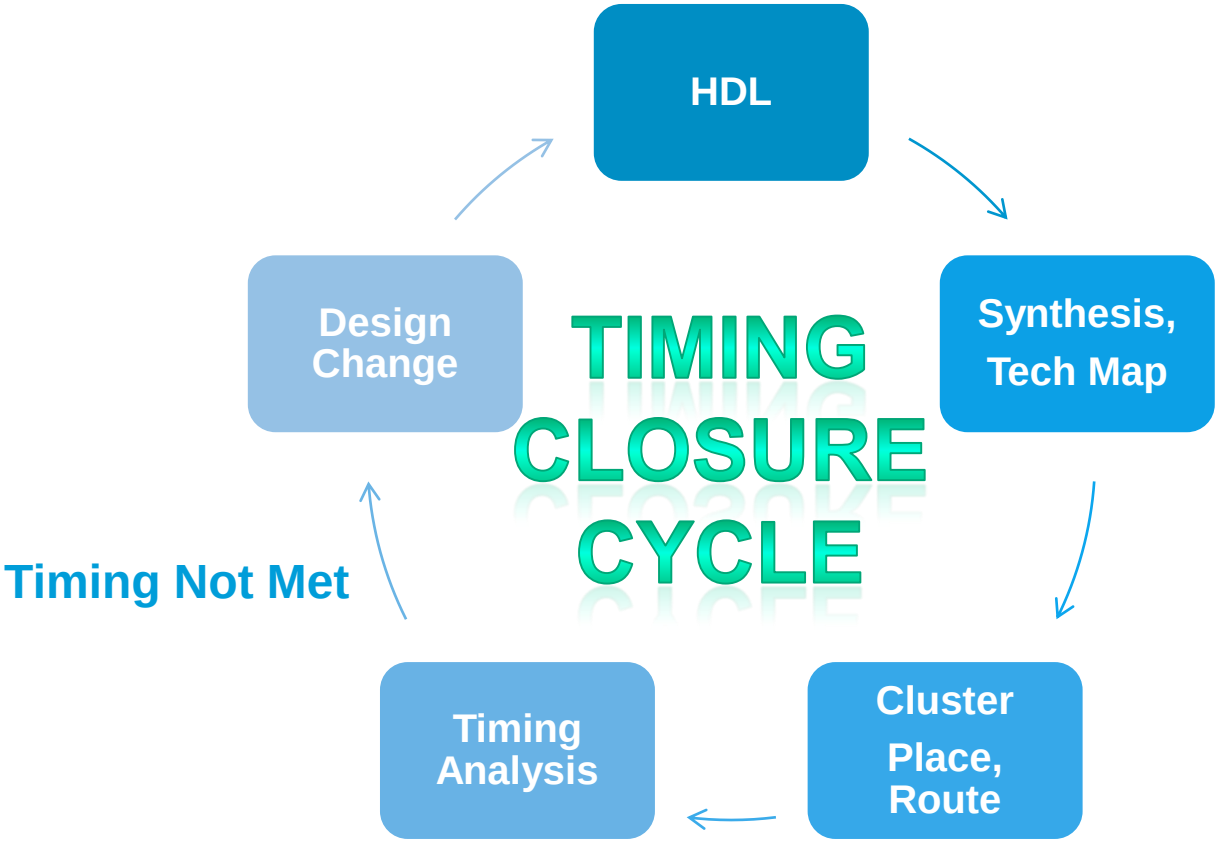
FPGA CAD / Compilation is Complex



- Sophisticated optimization algorithms are used in each step and lead to significantly longer runtimes than a software compile (hours vs. minutes)

Timing Closure Problems

- Designers will often have to go through numerous iterations to meet timing requirements

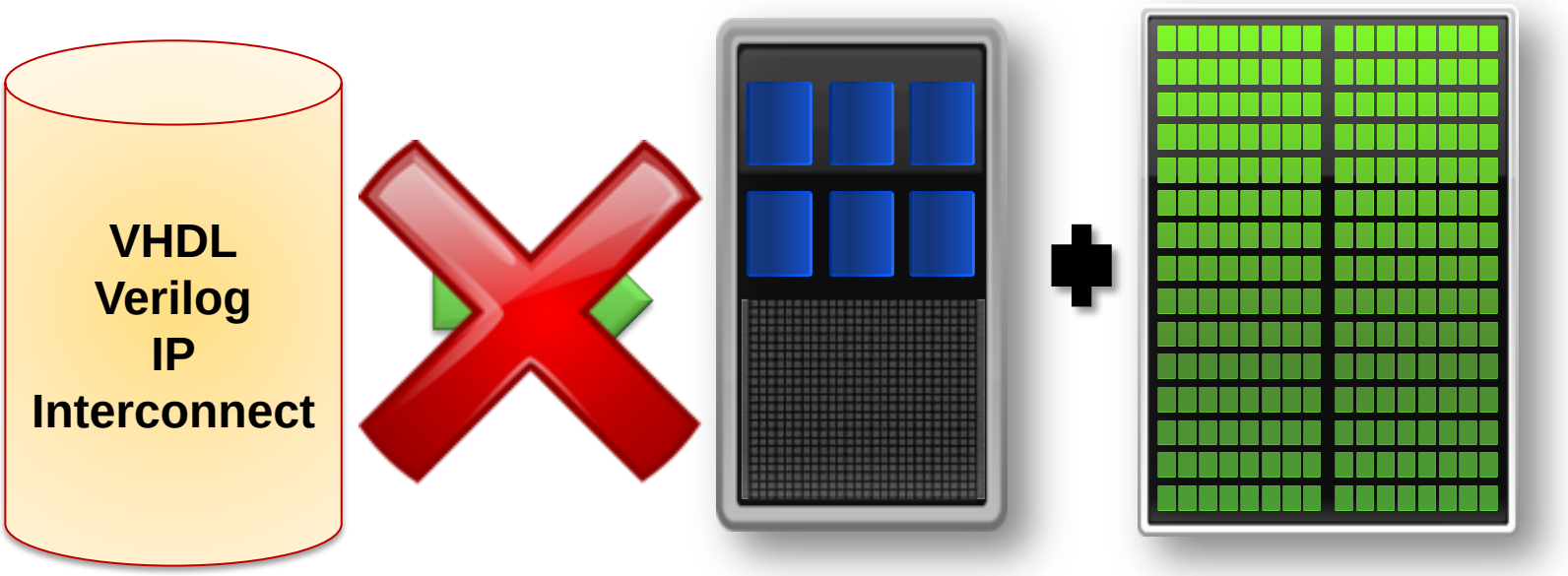


Design Scalability

- Using RTL design entry, there is significant work in porting applications from generation to generation of FPGA technology
- Ideally a 2x improvement in logic capacity should translate into 2x performance
- In addition to doubling the datapath, control logic and SOC interconnect need to change as well

Portability

- What happens when a designer wants to try their algorithm on another platform?
  - Would it be better on a CPU, GPU or DSP processor?



Fundamental challenges

- Implementing an algorithm on an FPGA is done by designing hardware
  - Difficult to design, verify and code for scalable performance
- Generally, software programmers will have difficulty using FPGAs as massive multi-core devices to accelerate parallel applications
- Need a programming model that allows the designer to think about the FPGA as a configurable multi-core device

An ideal programming environment …

- Has the following characteristics:
  - Based on a standard multicore programming model rather than something which is FPGA-specific
  - Abstracts away the underlying details of the hardware
    - VHDL / Verilog are similar to “assembly language” programming
      - Useful in rare circumstances where the highest possible efficiency is needed
  - The price of abstraction is not too high
    - Still need to efficiently use the FPGA’s resources to achieve high throughput / low area
  - Allows for software-like compilation & debug cycles
    - Faster compile times
    - Profiling & user feedback

OPENCL : THE ANSWER ?

The BIG Idea behind OpenCL

- OpenCL execution model …
  - Define N-dimensional computation domain
  - Execute a kernel at each point in computation domain

Traditional loops

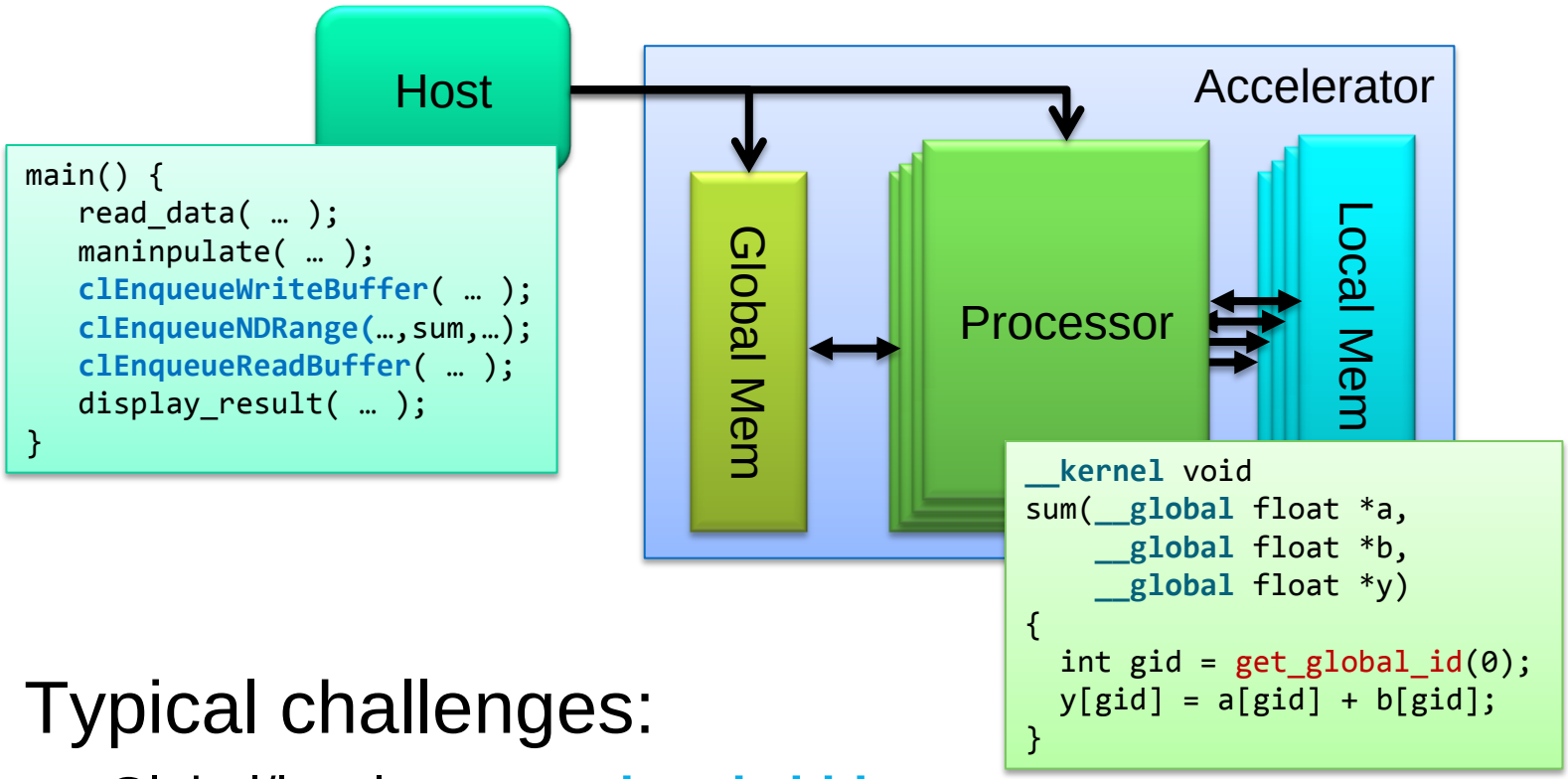
```
void
trad_mul(int n,
         const float *a,
         const float *b,
         float *c)
{
  int i;
  for (i=0; i<n; i++)
    c[i] = a[i] * b[i];
}
```

Data Parallel OpenCL

```
kernel void
dp_mul(global const float *a,
       global const float *b,
       global float *c)
{
  int id = get_global_id(0);

  c[id] = a[id] * b[id];

} // execute over "n" work-items
```

Parallelism is Explicit

OpenCL Programming Model



- Typical challenges:
  - Global/local **memory bandwidth**
  - Limited **floating point** cores
  - Thread **occupancy**

OpenCL Host Program

- Pure software written in standard ‘C’
- Communicates with the Accelerator Device via a set of library routines which abstract the communication between the host processor and the kernels

Copy data from Host to FPGA

Ask the FPGA to run a particular kernel

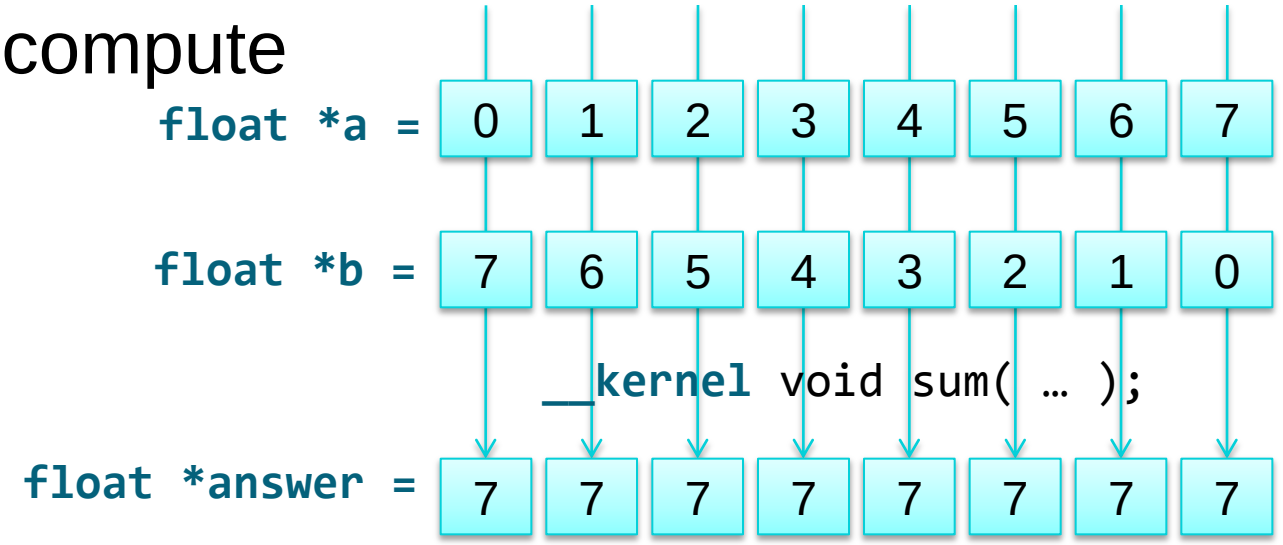
Copy data from FPGA to Host

```
main()
{
   read_data_from_file( … );
   maninpulate_data( … );

   clEnqueueWriteBuffer( … );
   clEnqueueTask(…, my_kernel, …);
   clEnqueueReadBuffer( … );

   display_result_to_user( … );
}
```

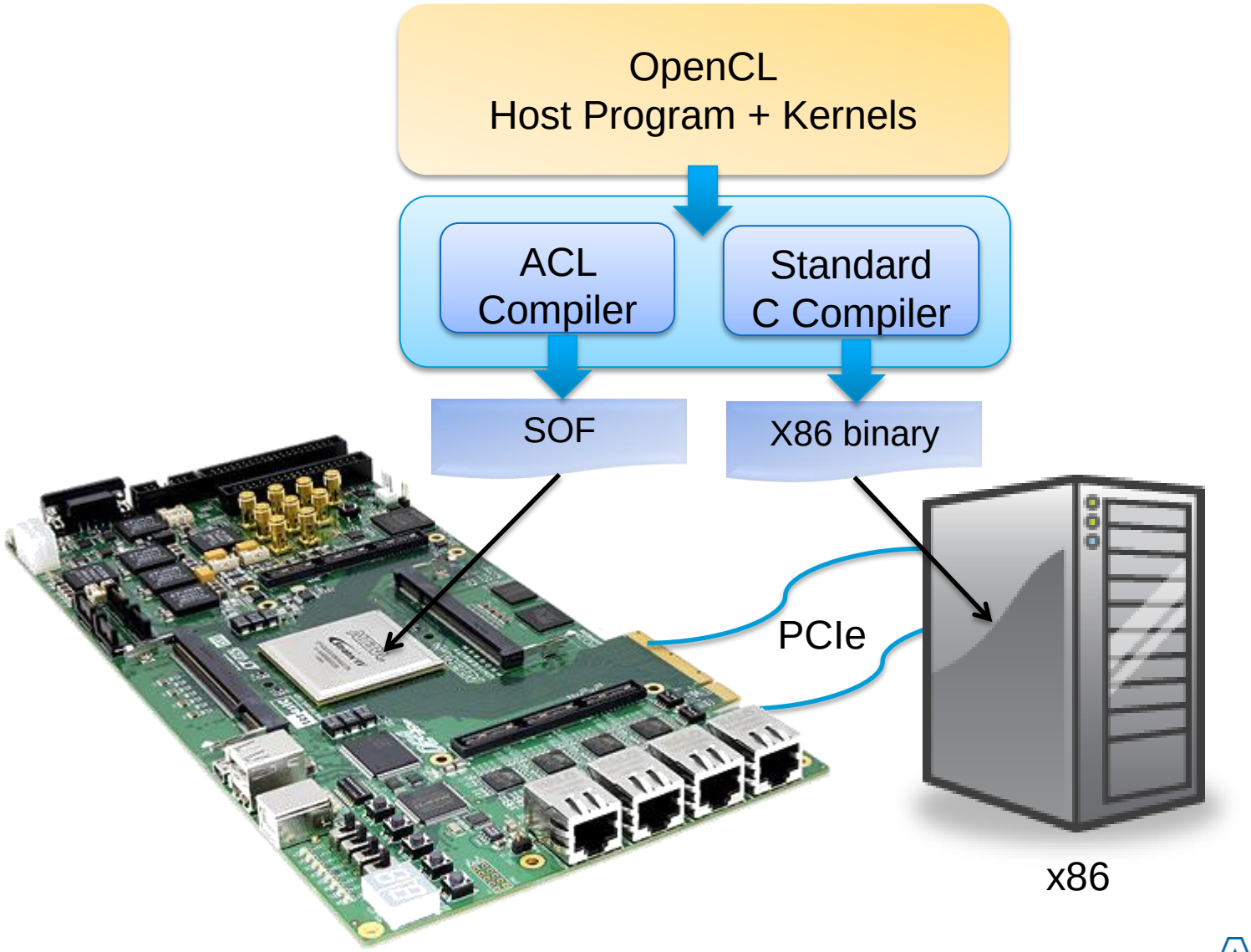
OpenCL Kernels

- Data-parallel function
  - Defines many parallel threads of execution
  - Each thread has an identifier specified by "**get_global_id**"
  - Contains keyword extensions to specify parallelism and memory hierarchy
- Executed by compute object
  - CPU
  - GPU
  - Accelerator

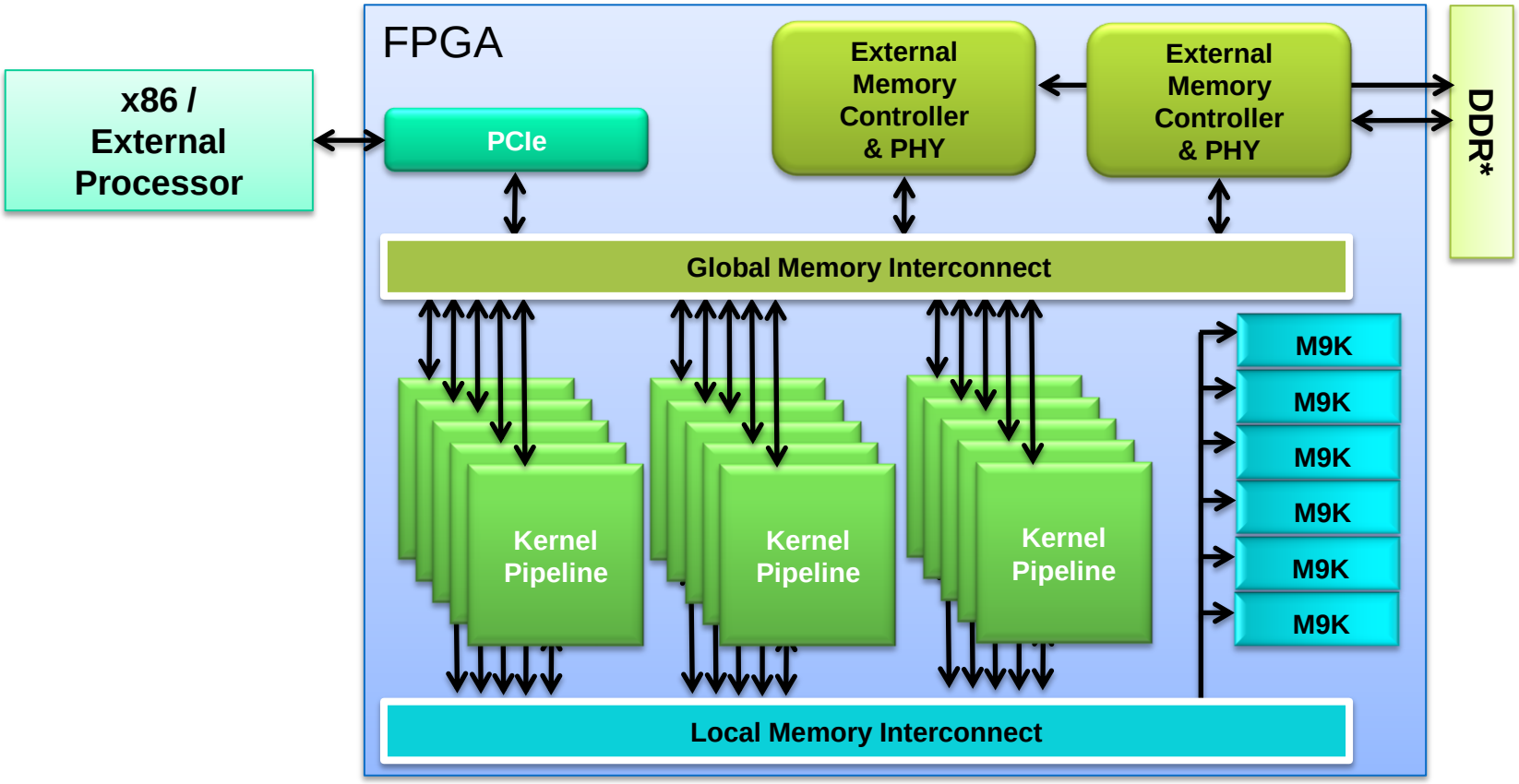
```
__kernel void
sum(__global const float *a,
__global const float *b,
__global float *answer)
{
int xid = get_global_id(0);
answer[xid] = a[xid] + b[xid];
}
```



Mapping OpenCL Programs



FPGA OpenCL Architecture



Modest external memory bandwidth

Extremely high internal memory bandwidth

Highly customizable compute cores

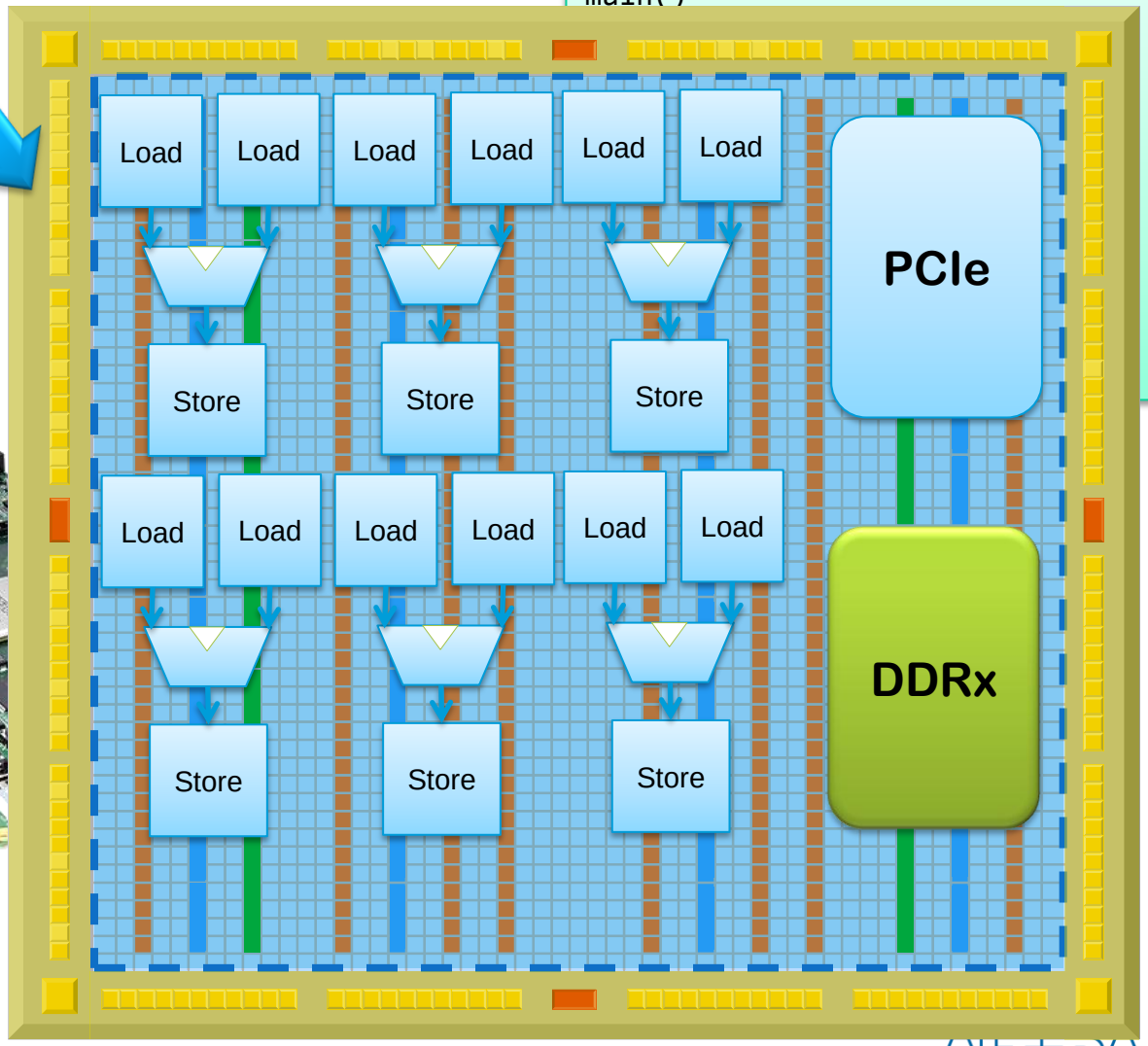
Compiling OpenCL to FPGAs

```
__kernel void
sum(__global const float *a,
__global const float *b,
__global float *answer)
{
int xid = get_global_id(0);
answer[xid] = a[xid] + b[xid];
}
```

```
{
int xid = get_global_id(0);
answer[xid] = a[xid] + b[xid];
}
```

Host Program

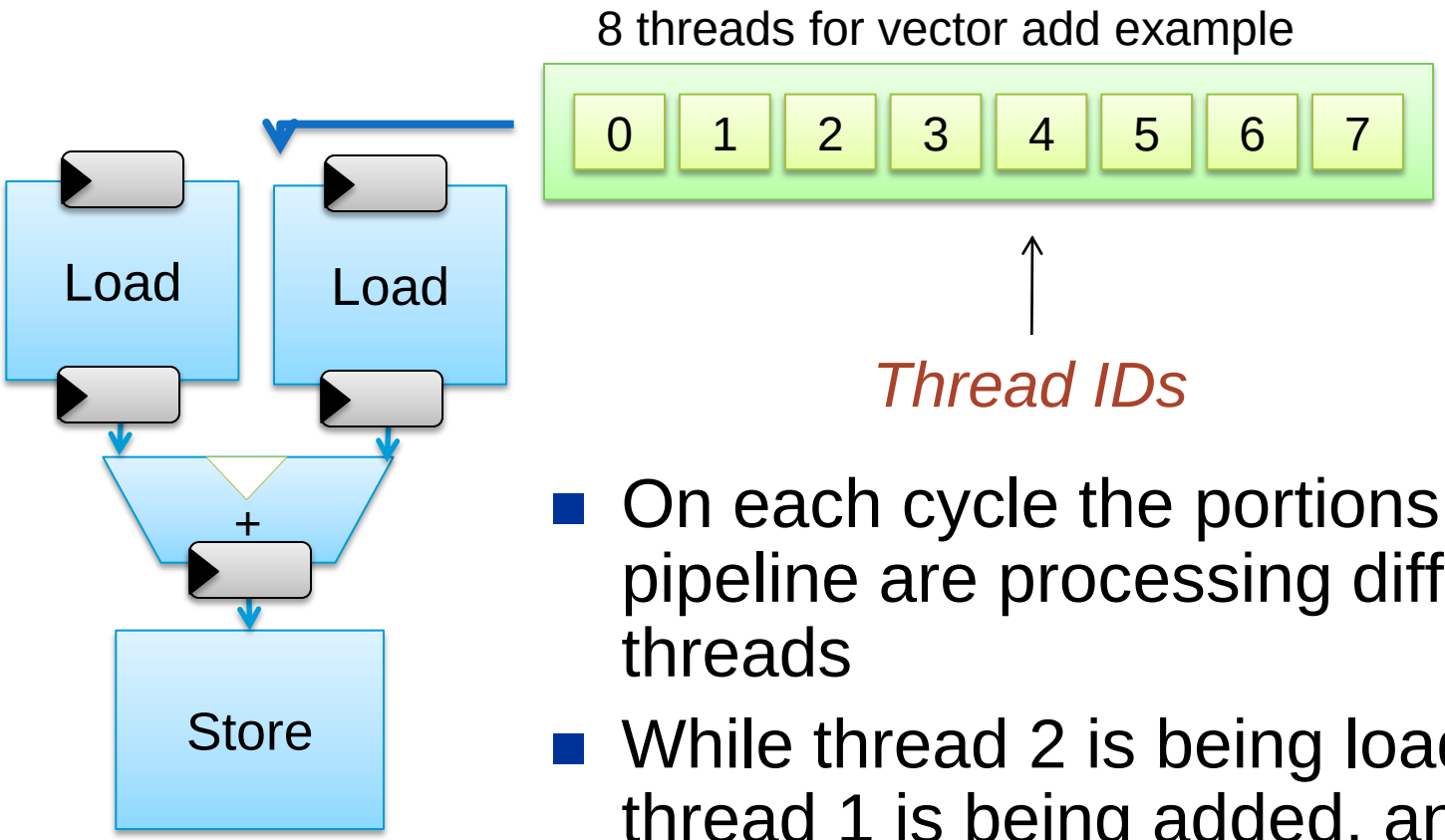
```
main()
```



Mapping Multithreaded Kernels to FPGAs

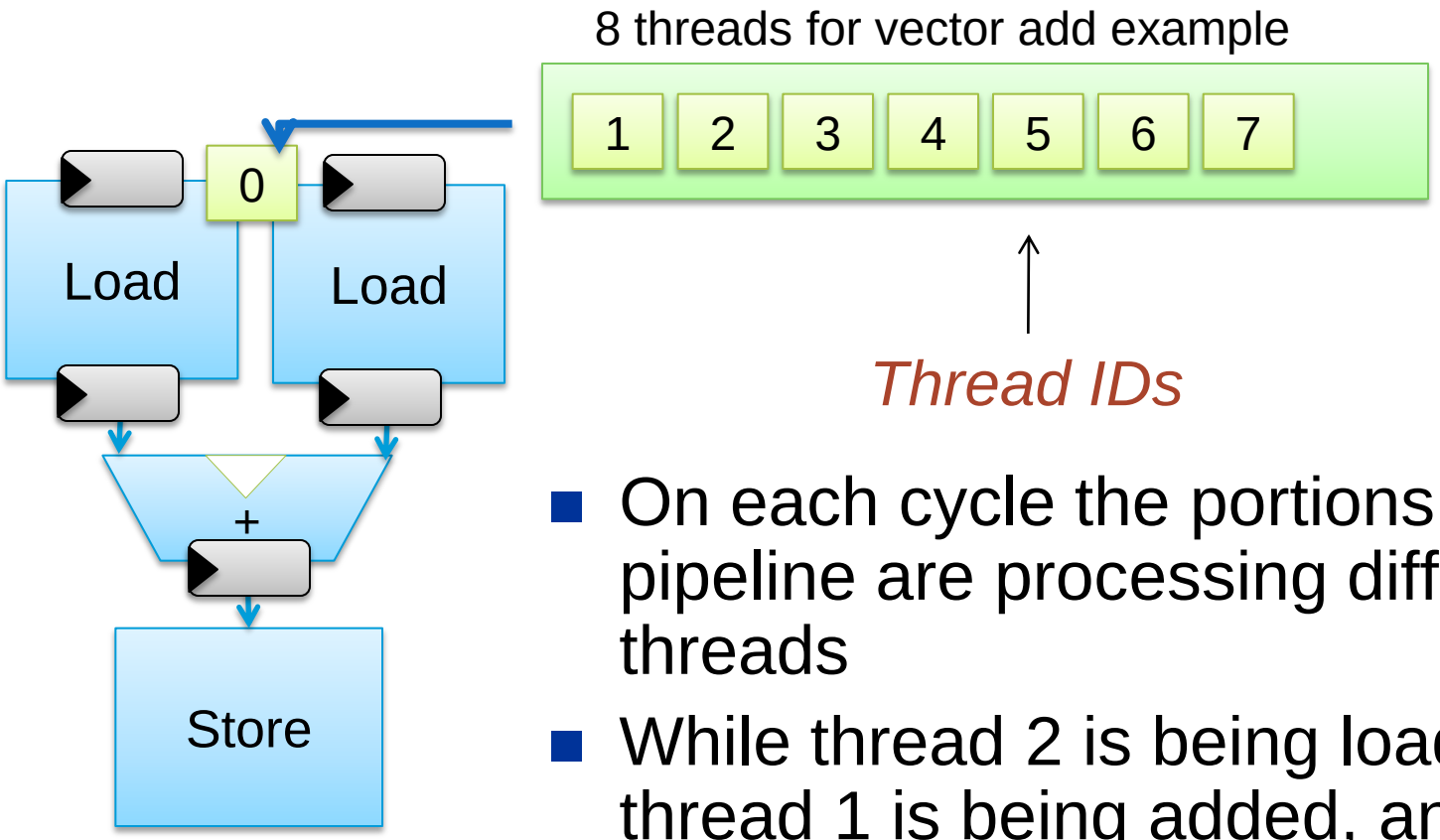
- The most simple way of mapping kernel functions to FPGAs is to replicate hardware for each thread
  - Inefficient and wasteful
- Better method involves taking advantage of *pipeline parallelism*
  - Attempt to create a deeply pipelined representation of a kernel
  - On each clock cycle, we attempt to send in input data for a new thread
  - Method of mapping coarse grained thread parallelism to fine-grained FPGA parallelism

Example Pipeline for Vector Add



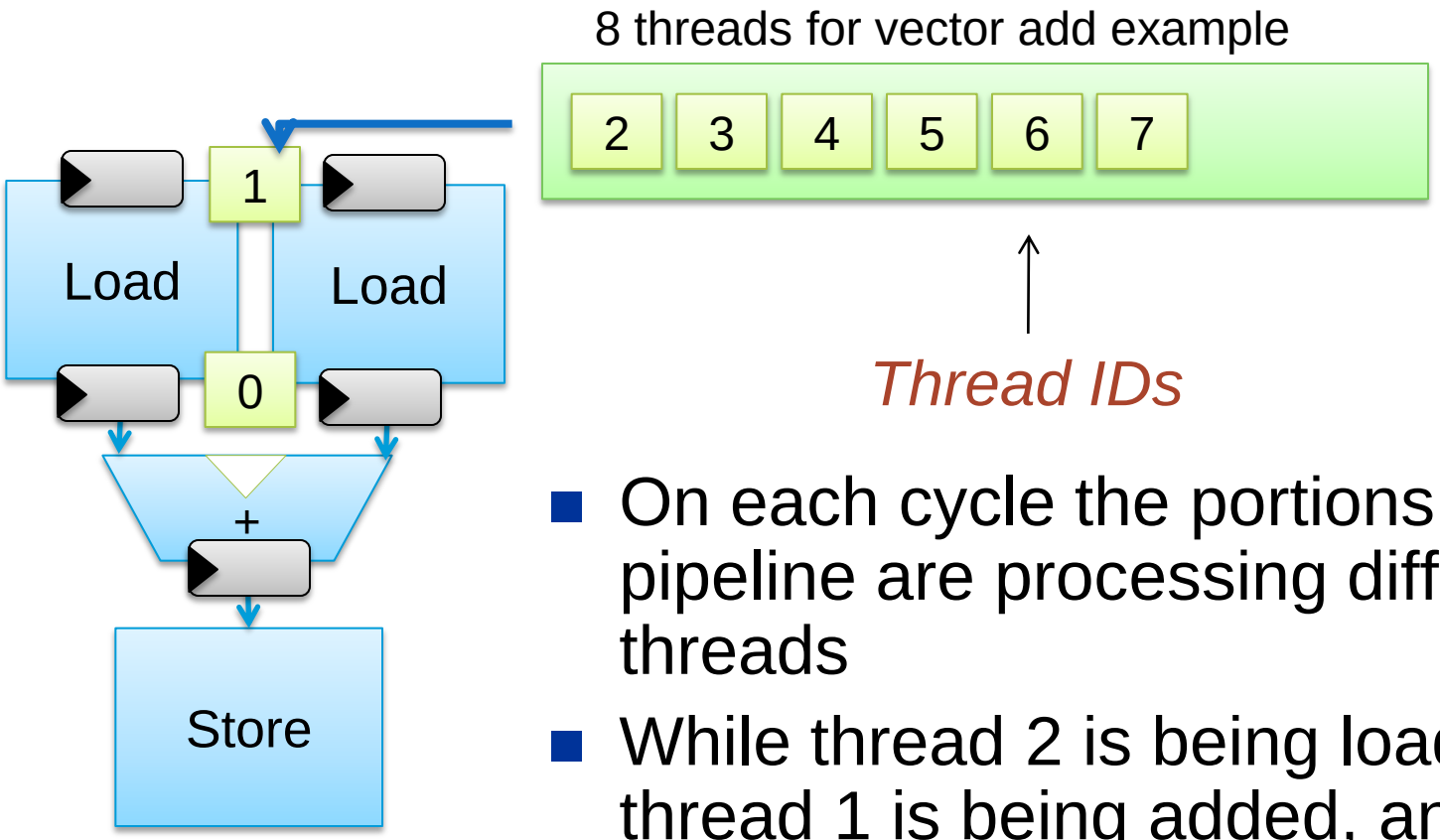
- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

Example Pipeline for Vector Add



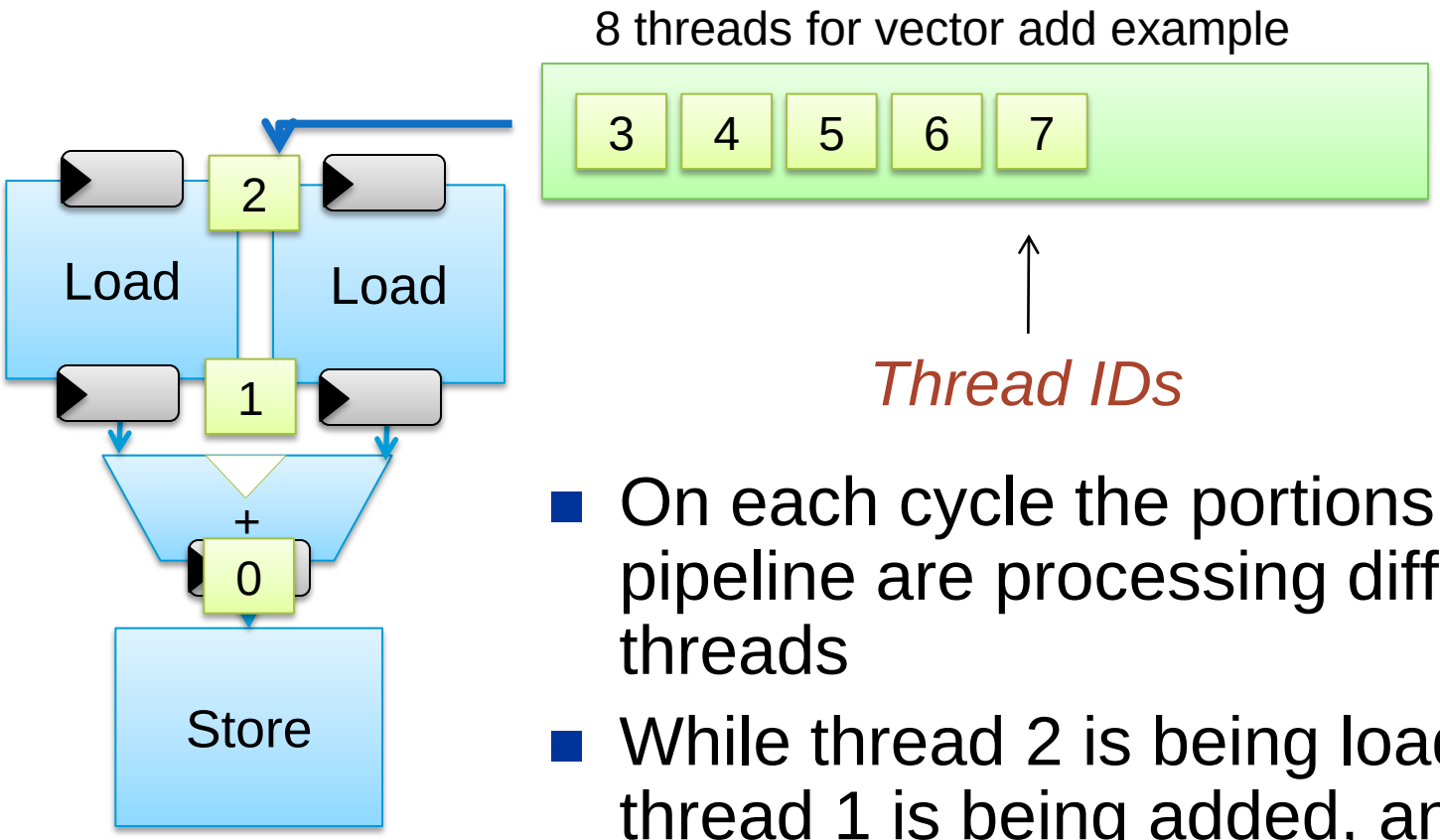
- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

Example Pipeline for Vector Add



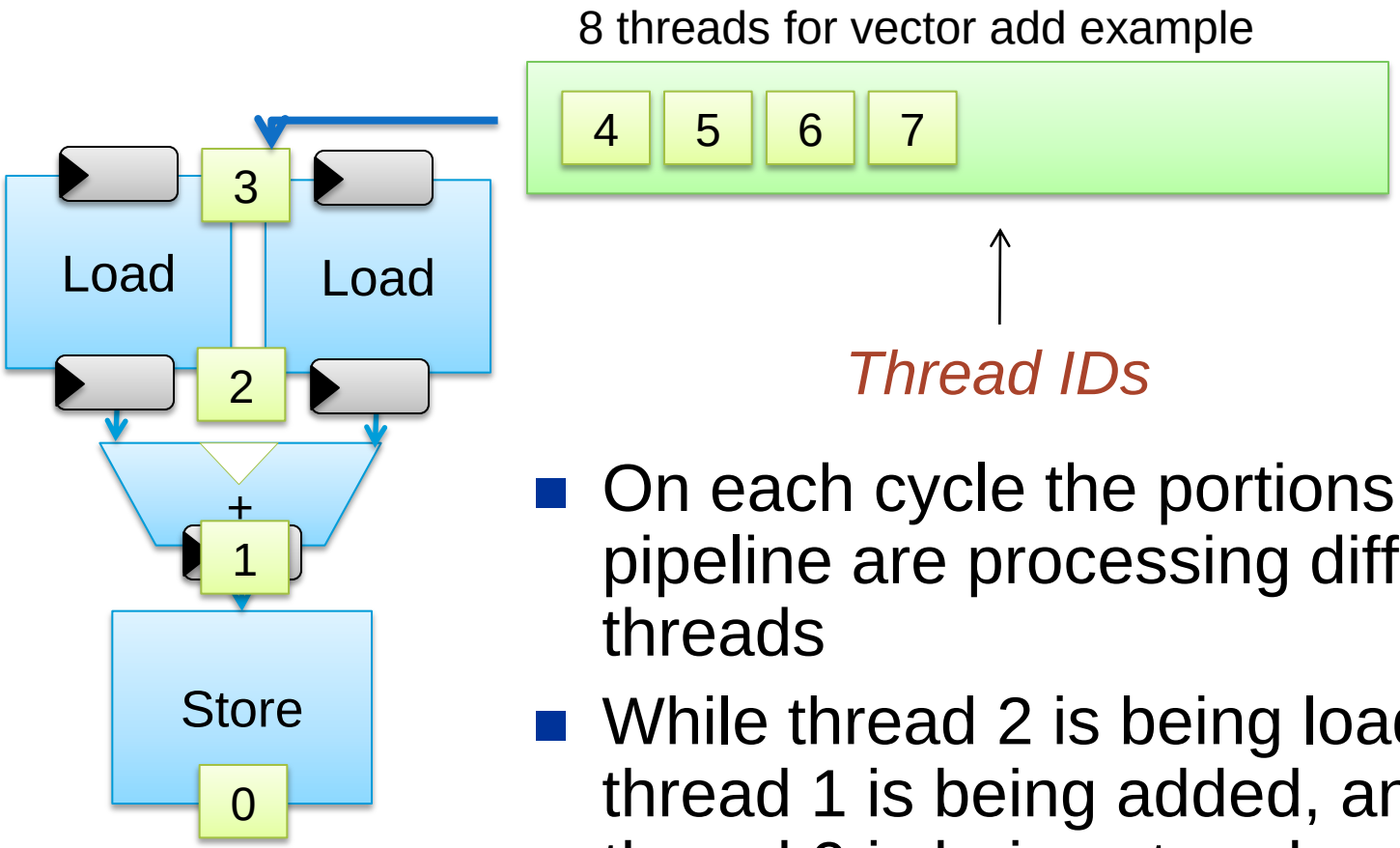
- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

Example Pipeline for Vector Add



- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

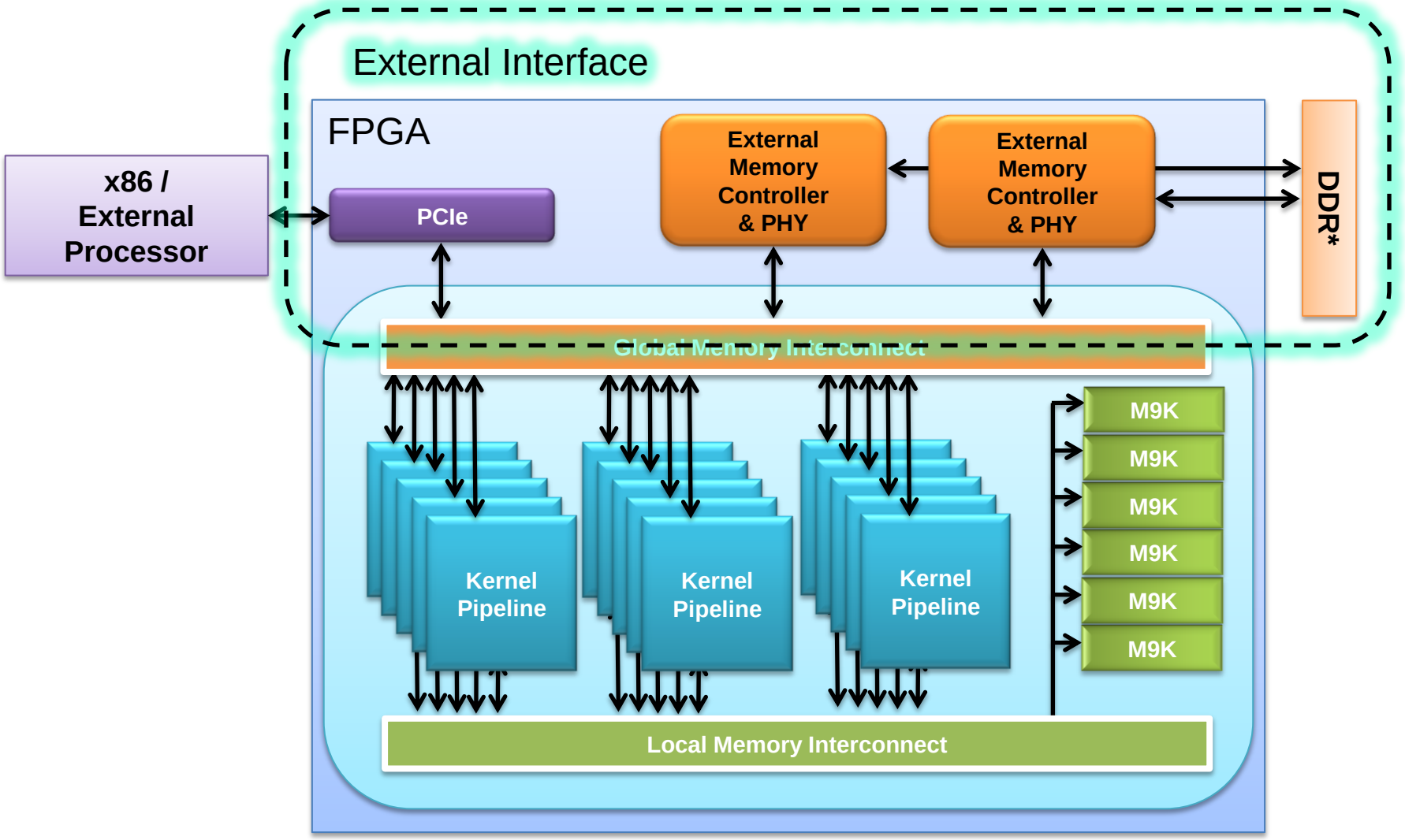
Example Pipeline for Vector Add



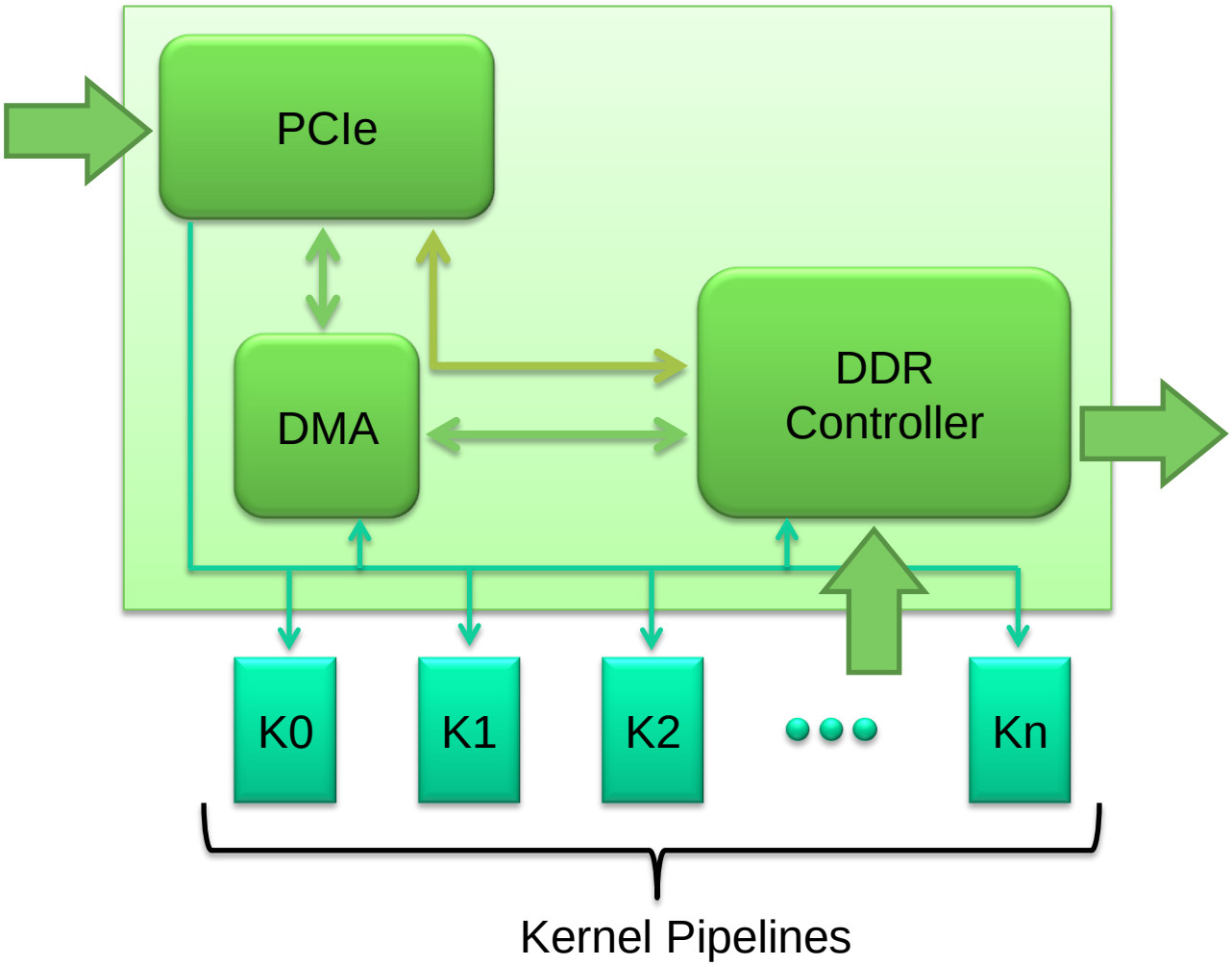
- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

ALTERA OPENCL SYSTEM ARCHITECTURE

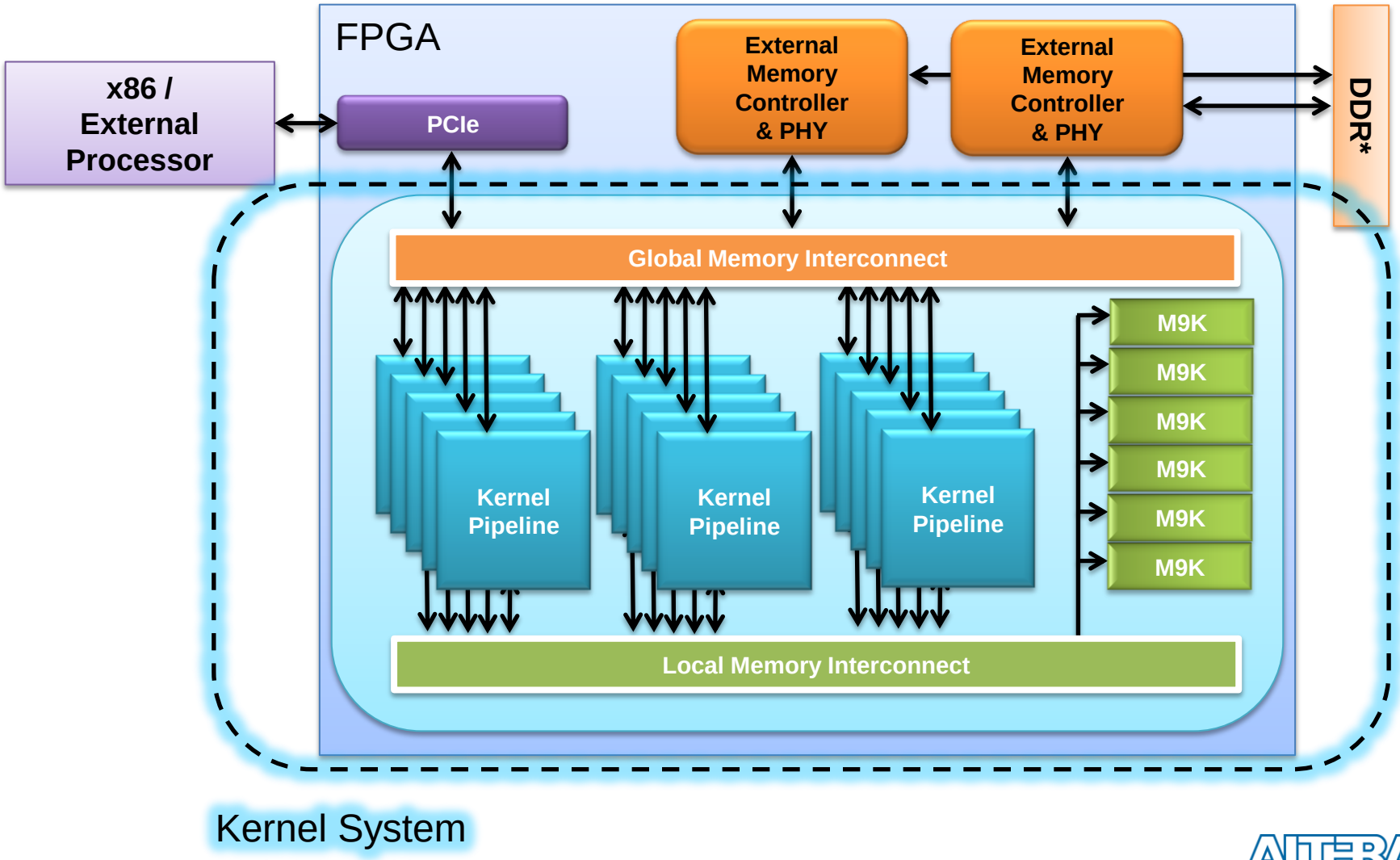
OpenCL System Architecture



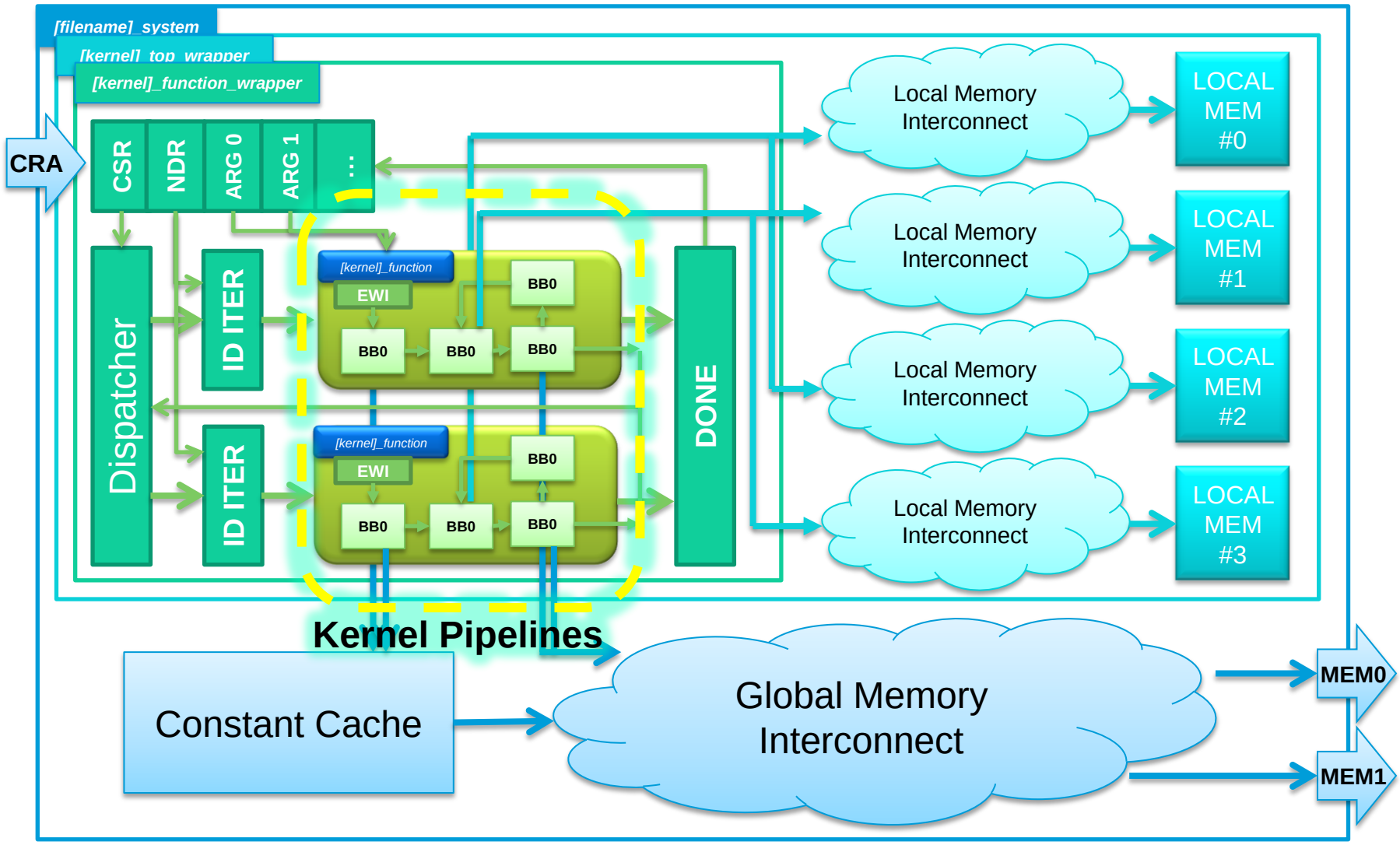
External Interface : High Level



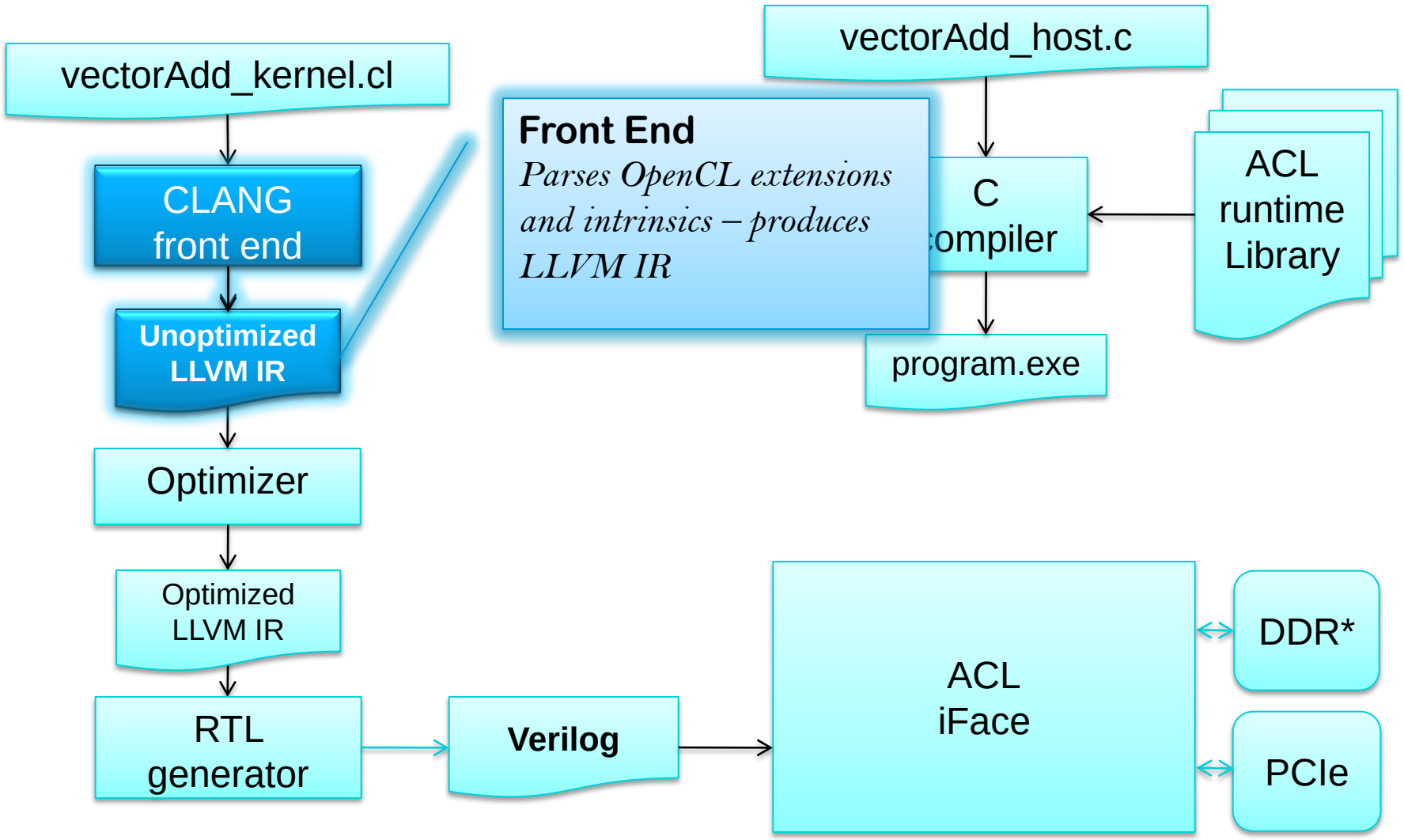
OpenCL System Architecture



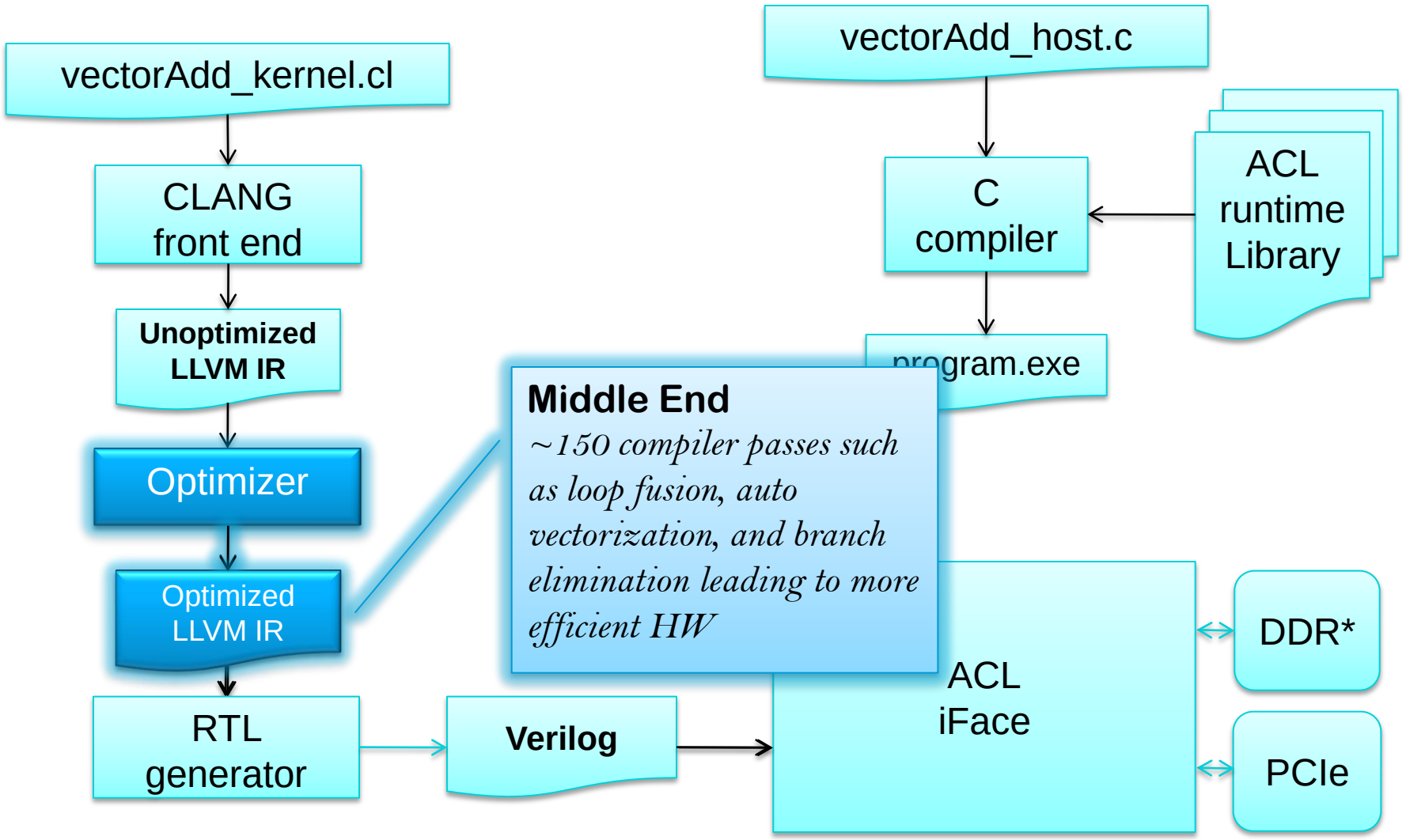
Altera OpenCL Kernel Architecture



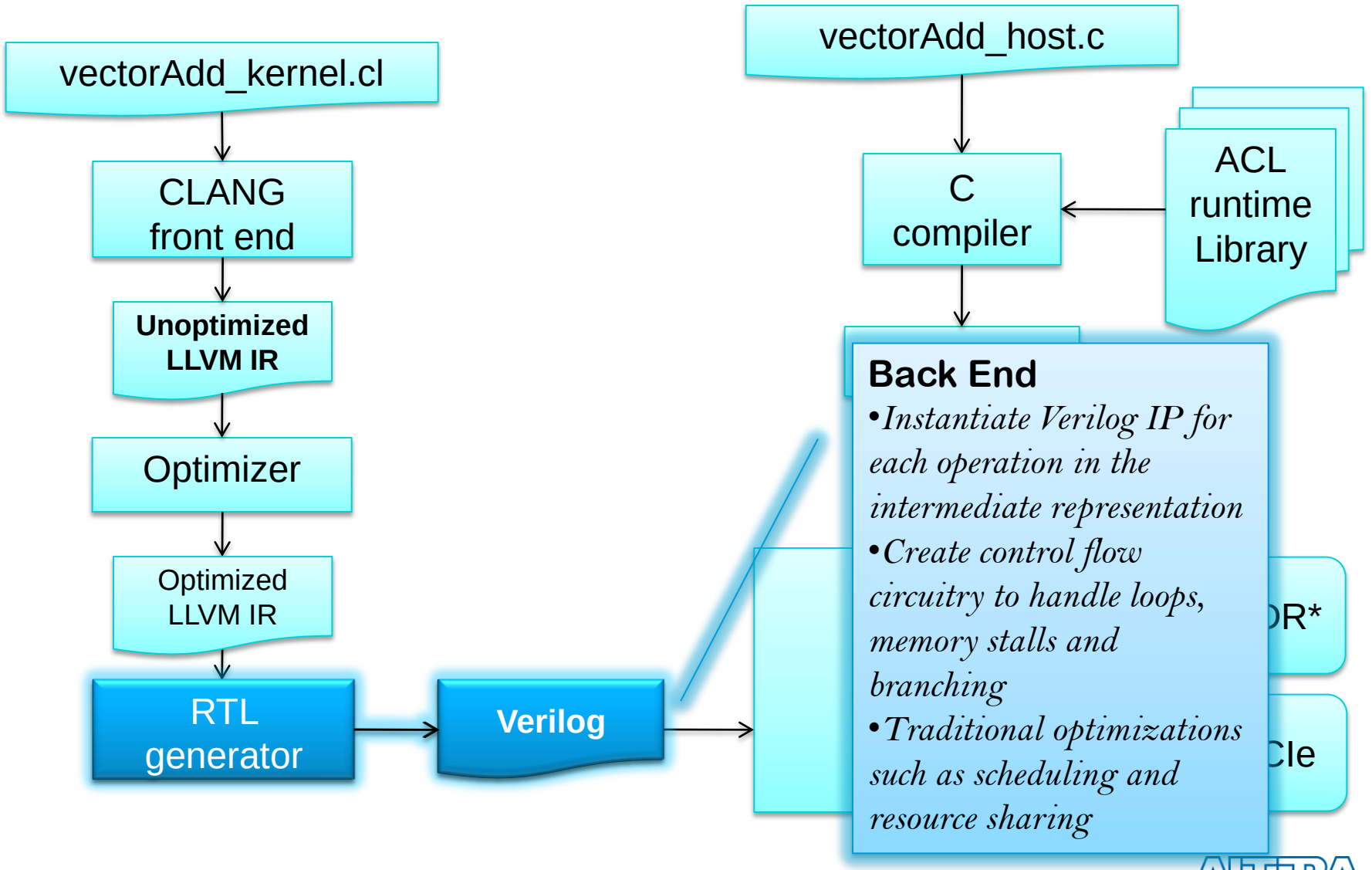
OpenCL CAD Flow



OpenCL CAD Flow



OpenCL CAD Flow



CASE STUDIES

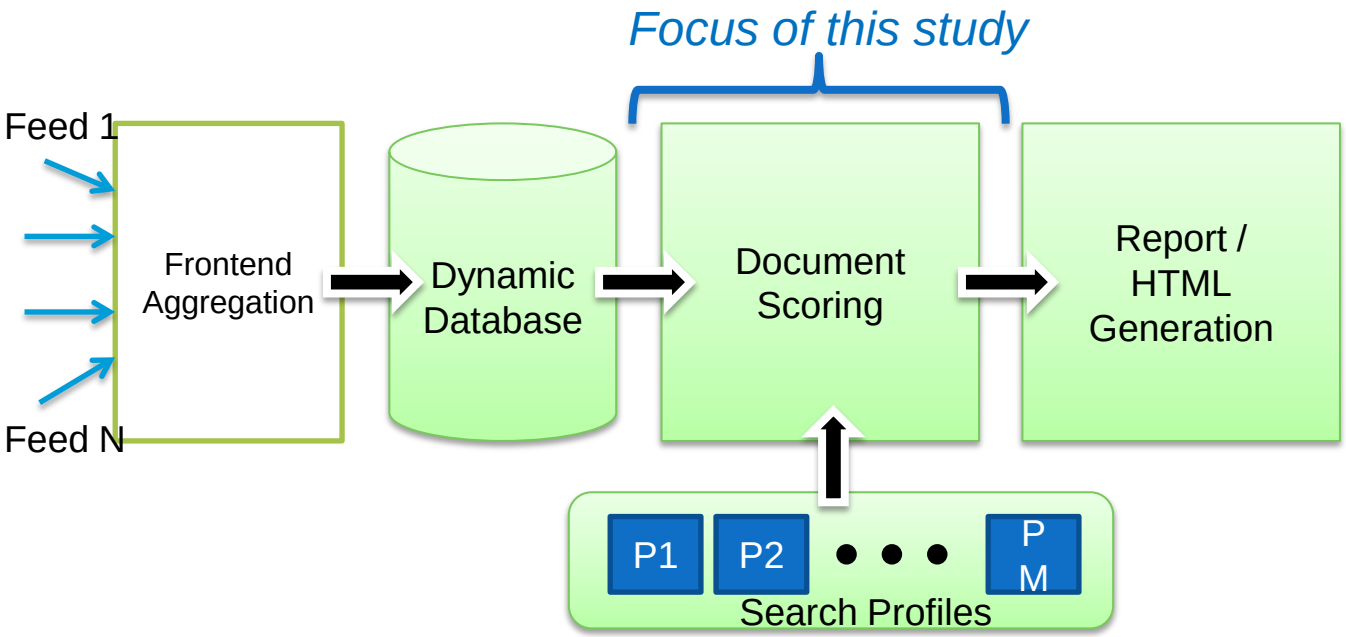
Applications from Different Domains

- Document Filtering
  - Simplified information filtering benchmark
  - High on BW, low on compute
    - All about moving the data efficiently
  - Detailed study in:
  - Doris Chen, Deshanand Singh, "*Invited Paper: Using OpenCL to evaluate the efficiency of CPUs, GPUs, and FPGAs for Information Filtering*", FPL'2012.

- Monte Carlo Black Scholes Computation
  - Financial Benchmark to compute derivative prices
  - Heavy on compute, low on BW requirements
    - The vast majority of data and communications are kept onchip

EXAMPLE: INFORMATION FILTERING

Information Filtering

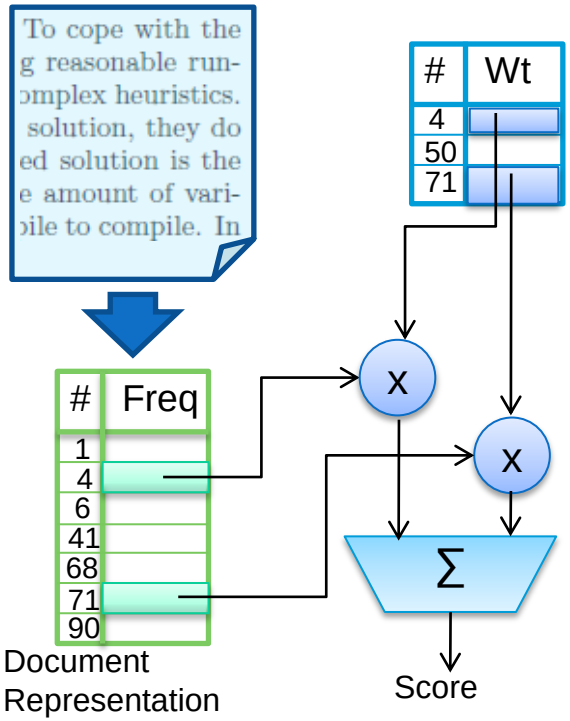


- Filter document feeds for content which matches particular user search profiles
- Examples:
  - News articles which match a particular interest list
  - Newly published conference or journal papers which are in your research area

General Idea

- Documents are converted into bag of words format:
  - (t_1,f_1), (t_2, f_2), (t_n, f_n)
  - 8 bits for the frequency
  - 24 bits for the term ID
- Search profiles have the format:
  - (t_1,w_1), (t_2, w_2), (t_n, w_n)
  - 64 bit fixed point representations of the weights
- Documents are scored using the following:

$$Score(Doc, P) = \sum_{t_i \in Doc} f_i w(t_i)$$



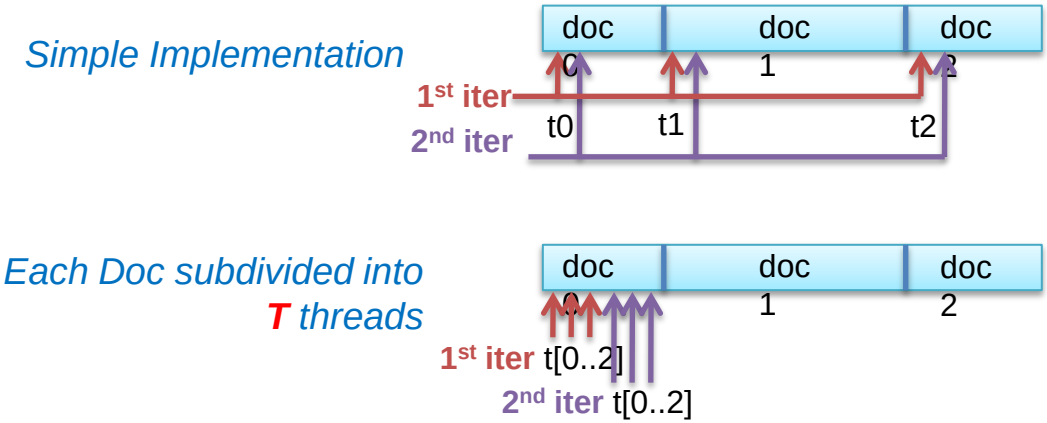
Simple Initial OpenCL Implementation

```
__kernel void computeScore(
  unsigned* termFreq, ulong* profileWts,
  unsigned* docStart,
  unsigned* docNumTerms, ulong* profileScore) {
  unsigned first_term = docBegins[get_global_id(0)];
  unsigned num_terms = docNumTerms[get_global_id(0)];
  ulong my_score = 0;

  for (unsigned i = 0; i < num_terms; i++) {
    unsigned curr_entry = wordFreq[first_term + i];
    unsigned term_id = curr_entry >> 8;
    unsigned frequency = curr_entry & 0x00ff;

    my_score += profileWts[term_id] * (ulong)frequency;
  }
  profileScore[get_global_id(0)] = my_score;
}
```

- Process each document as one parallel thread

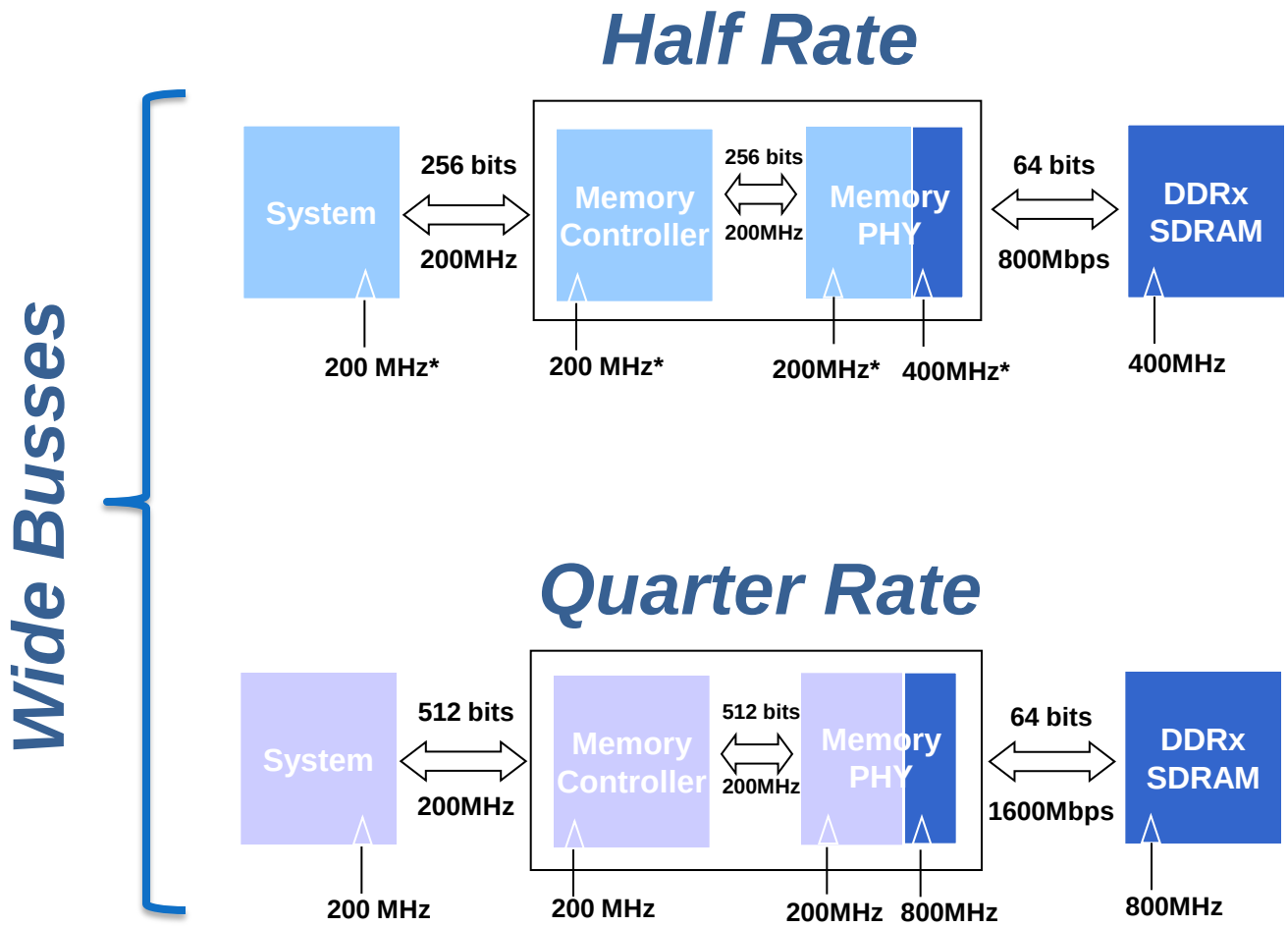
Parameterizing the code

- Some architectures (FPGAs, GPUs) make use of *memory coalescing* optimizations where efficient requests are made in the case where:
  - Consecutive threads access consecutive memory locations



  - Note that *T=1* is the same as having one document processed per parallel thread

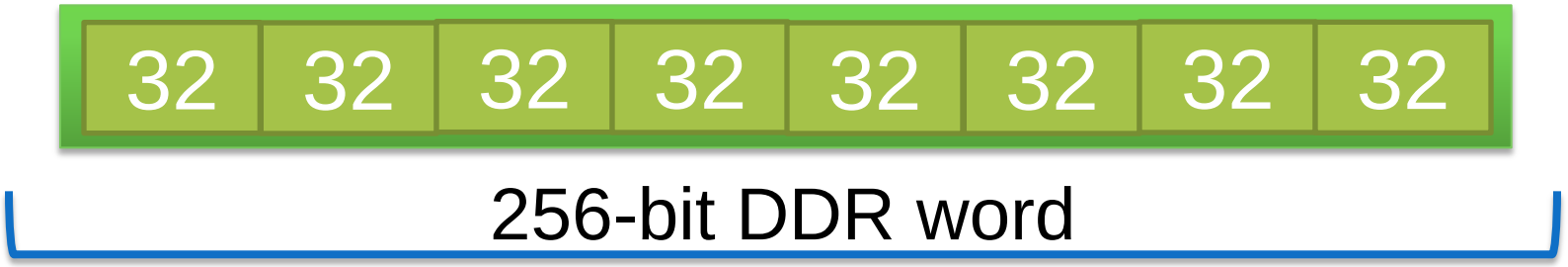
DDRx Configuration



* This frequency target is an example only, it does not reflect the actual frequency configuration that is supported by DDRx controller

Load/Store Memory Coalescing

- External memory has wide words (256 bits)
- Loads/stores typically access narrower words (32 or 128 bits)

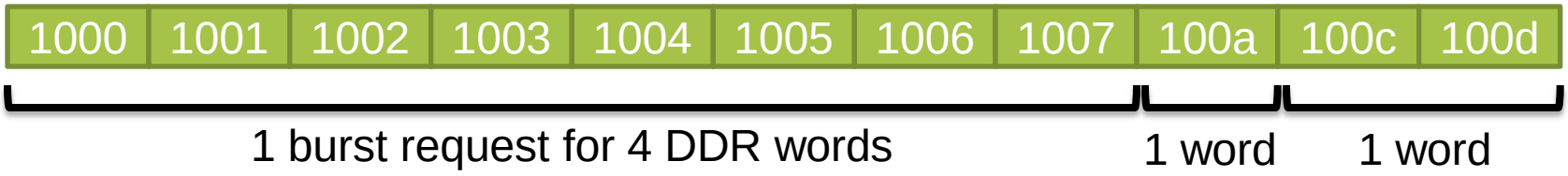


- Given a sequence of loads/stores, we want to make as few external memory read/write requests as possible

Load/Store Memory Coalescing

- **Coalescing is important for good performance**
  - Combine loads/stores that access the same DDR word **or the one ahead of the previously-accessed DDR word**
  - Make one big multi-word burst request to external memory whenever possible
  - ➢ Fewer requests → less contention to global memory
  - ➢ Contiguous bursts → less external memory overhead

Load/Store Addresses (128-bit words):



→ 3 requests in total

Bloom Filtering

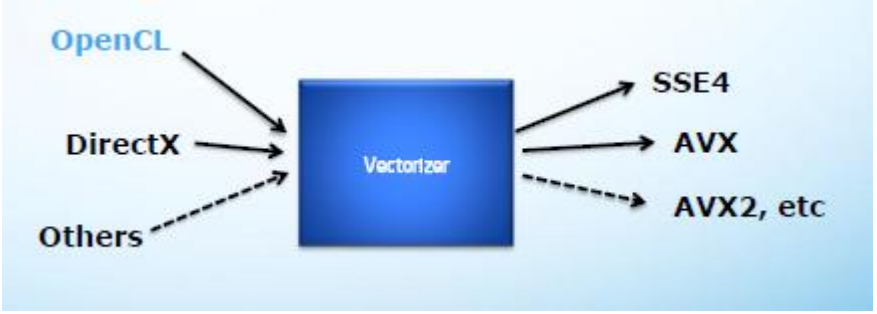
- *Most entries in the search profile are ZERO*
  - User typically only cares about some subset of the terms present in all documents
- A simple hashing strategy can be used to filter out unnecessary requests to external memory
- Bloom filters are generalizations where multiple hash functions can be used

- The size of the bloom filter directly impacts the number of false positives leading to external memory accesses

Test Platforms

Test Platform	Specs	Process	External Memory BW	Cache Size	Board Power
Multi-Core CPU	Intel Xeon W3690	32nm	32 GB/s	12 MB	130W
GPU	NVIDIA Tesla C2075	40nm	144 GB/s	768 MB	215W
FPGA	Altera Stratix-IV 530 → DE4	40nm	12.8 GB/s	No hardened cache	21W

OpenCL on CPU



Source: http://llvm.org/devmtg/2011-11/Rotem_IntelOpenCLSDKVectorizer.pdf

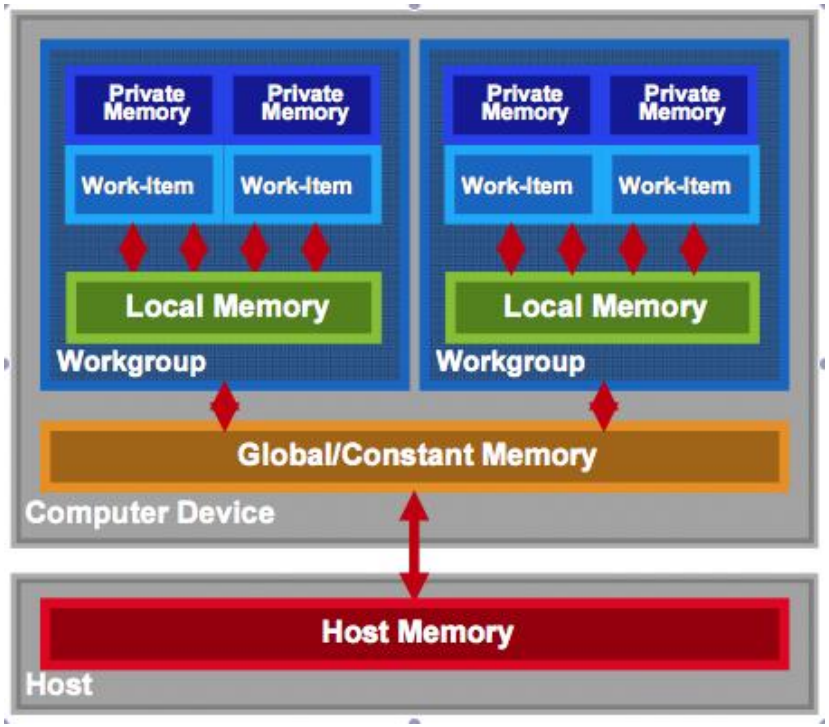
- Version 1.5 of the Intel OpenCL SDK has Autovectorizing capabilities that allow the Kernel to take advantage of SSE* instructions.
  - Need to use these to get a fair comparison of the CPU baseline

CPU Results

Configuration	MT / s	MT / J
T=1, no bloom filter	196	1.5
T=1, bloom filter (32k)	1614	12.4
T=1, bloom filter (64k)	**2070**	**15.9**
T=1, bloom filter (128k)	1717	13.2
T=2, bloom filter (64k)	1949	15.0
T=4, bloom filter (64k)	442	3.4

- T=1 leads to the best performance
  - Entire documents can be fetched into local caches
- Large bloom filters can be used and entirely kept in the cache

OpenCL on GPU



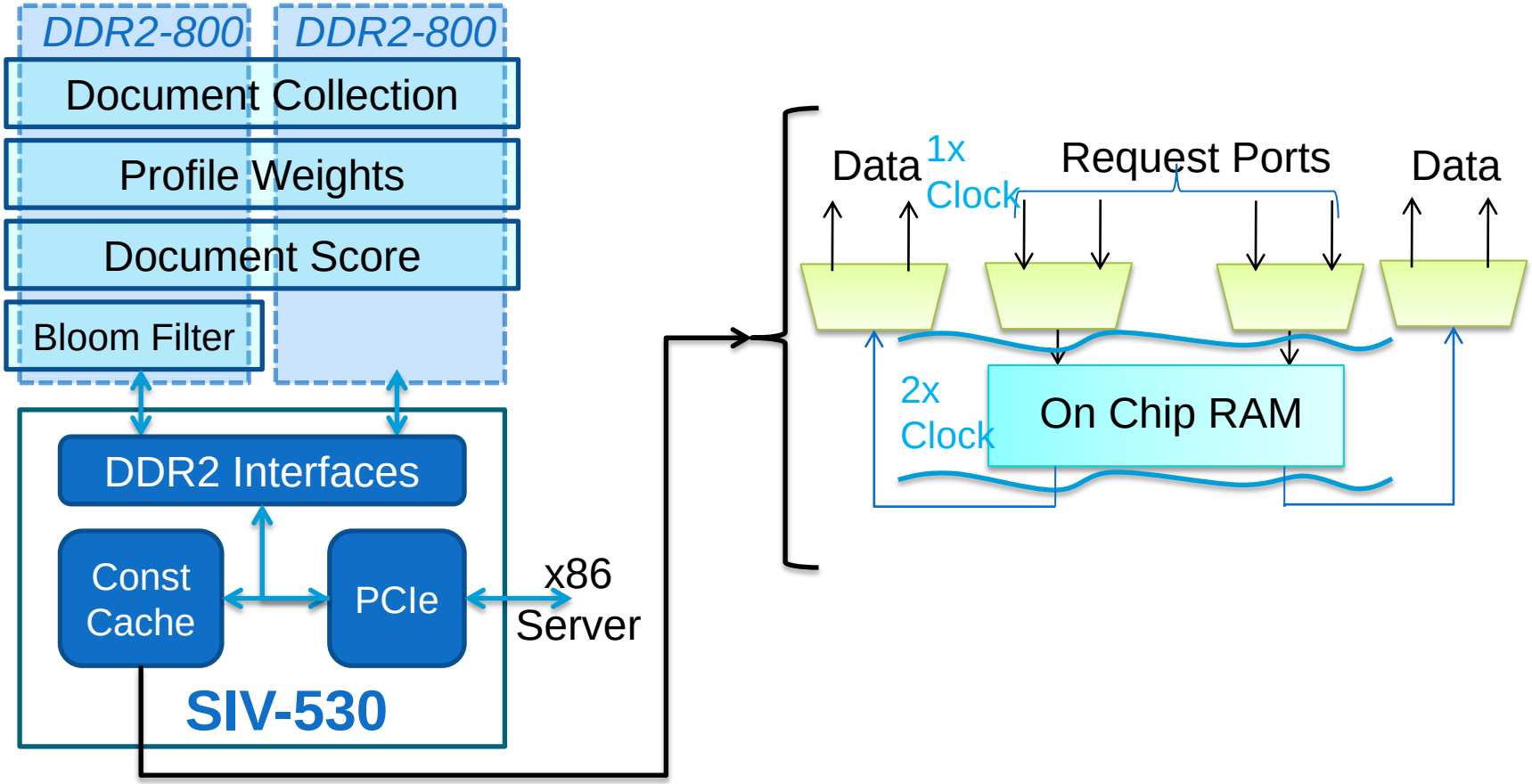
- Hierarchical Memory Model
  - Constant → used to hold lookup table data that is unchanging during the run of a program
  - Local Memory → Scratchpad space where threads can share information / intermediate results
  - Global → Off chip DDR memoy

GPU Results

Configuration	MT / s	MT / J
T=128, no bloom filter	671	3.1
T=128, bloom filter (32k, constant)	1138	5.3
T=128, bloom filter (32k, local)	501	2.3
T=128, bloom filter (32k, global)	2499	11.6
T=64, bloom filter (32k, global)	2196	10.2
T=256, bloom filter (32k, global)	2381	11.1
T=512, bloom filter (32k, global)	1695	7.9
T=128, bloom filter (64k, global)	1923	8.9
T=128, bloom filter (16k, global)	**3240**	**15.1**
T=128, bloom filter (8k, global)	2798	13.0
T=128, simple hash (8k, global)	2515	11.7

- Small bloom filter (16K) which is stored in “global” memory performs the best
  - Tesla C2075 has a cache which likely holds the entire bloom filter

OpenCL on FPGA



- One of the key innovations in the Altera OpenCL compiler is the ability to create a “soft logic” cache for *constant* data

FPGA Results

Configuration	MT / s	MT / J
T=64, no bloom filter	50	2.4
T=64, bloom filter (32k, constant)	1637	77.9
T=64, bloom filter (64k, constant)	**1755**	**83.6**
T=32, bloom filter (64k, constant)	1535	73.1
Extrapolated: FPGA + Double Bandwidth	2925	117
Hand coded FPGA solution [1]	772	N/A

- FPGA solution is completely limited by external memory bandwidth
- Notice the tremendous impact of the bloom filter on ensuring that the bandwidth is not wasted

[1] Sai Rahul Chalamalasetti, Martin Margala, Wim Vanderbauwhede, Mitch Wright, Parthasarathy Ranganathan: Evaluating FPGA-acceleration for real-time unstructured search. ISPASS 2012: 200-209

Overall Results

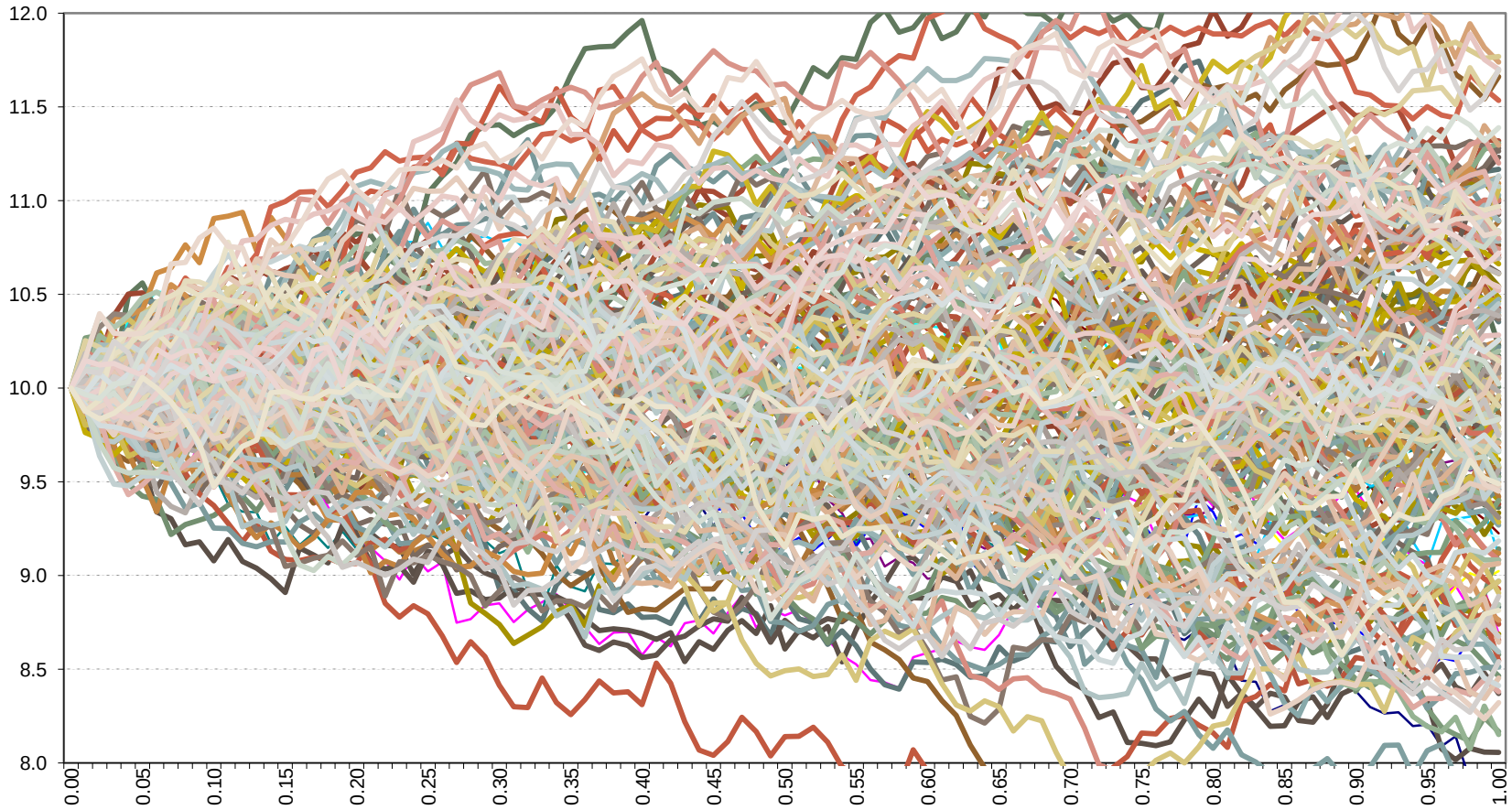
- Kernel is able to filter documents at a rate of 11.7 GBYTES / second
- Achieves much better performance / watt than GPU or CPU

Platform	Perf/Watt (MT / J)
Statix IV-530	**83.6 → 117 (extrapolated)**
Xeon W3690	15.9
Tesla C2075 GPU	15.1

- With a slightly better board design, the FPGA can almost match the GPU in terms of pure performance while consuming 190W less
  - *200W equates to approx $300 / year in power costs*

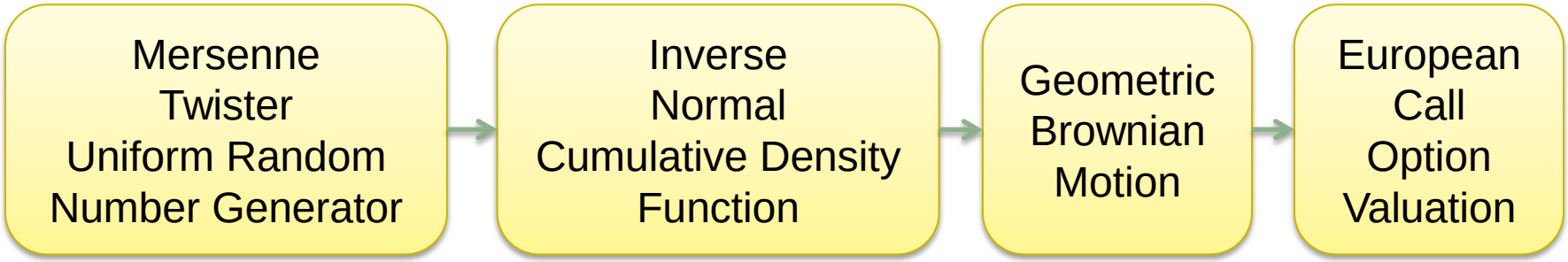
EXAMPLE: MONTE CARLO BLACK SCHOLES

Finance : Equity Derivative Pricing 200 simulations



~ 100,000 simulations may be required to achieve a result that is accurate enough

Overall Algorithm Architecture

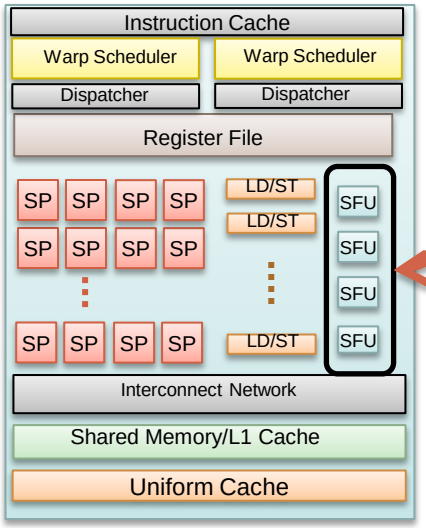


- Approximately 300 lines of OpenCL code can be used to describe this entire application
- Portable from CPU to GPU to FPGA with no changes
- Currently implemented with IEEE 754 single precision
  - Possible extension to extended single precision (36 bit mantissa)

Example : Inverse Normal CDF

```
float ltqnorm(float p)
{
        floatq, r;

        errno = 0;

        if (p < 0 || p > 1)
        {
                return 0.0;
        }
        else if (p == 0)
        {
                return -HUGE_VAL /* minus "infinity" */;
        }
        else if (p == 1)
        {
                return HUGE_VAL /* "infinity" */;
        }
        else if (p < LOW)
        {
                /* Rational approximation for lower region */
                q = sqrt(-2*log(p));
                return (((((c[0]*q+c[1])*q+c[2])*q+c[3])*q+c[4])*q+c[5]) /
                        ((((d[0]*q+d[1])*q+d[2])*q+d[3])*q+1);
        }
        else if (p > HIGH)
        {
                /* Rational approximation for upper region */
                q  = sqrt(-2*log(1-p));
                return -(((((c[0]*q+c[1])*q+c[2])*q+c[3])*q+c[4])*q+c[5]) /
                        ((((d[0]*q+d[1])*q+d[2])*q+d[3])*q+1);
        }
        else
        {
                /* Rational approximation for central region */
                q = p - 0.5;
                r = q*q;
                return (((((a[0]*r+a[1])*r+a[2])*r+a[3])*r+a[4])*r+a[5])*q /
                        ((((b[0]*r+b[1])*r+b[2])*r+b[3])*r+b[4])*r+1);
        }
}
```

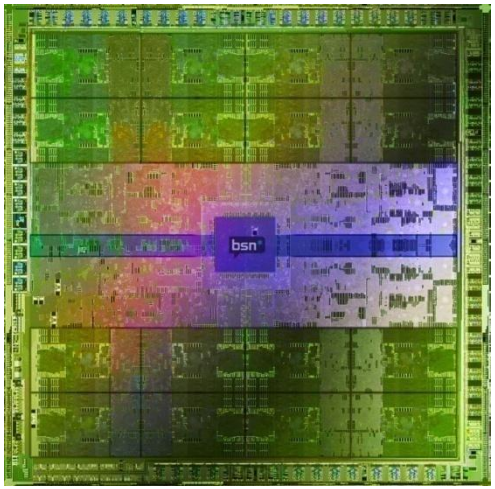
- This computation contains complex functions such as sqrt and log
- *GPUs cannot execute this code as efficiently as FP Adds and Mults*



Throughput & Power Comparison

	Power	Throughput
C2075 (GPU)	215W	2098 MSims/second
SIV530 (DE4)	21W	2181 MSims/second

- C2075 : NVIDIA's accelerator card
  - Based on the "Fermi" architecture
    - 40nm – 570 mm^2
  - 1.15 GHZ
  - 448 Cores
  - 1.03 Teraflops of single-precision performance
  - 144 GB/second global memory bandwidth
  - PCIe Gen2 x 16



RESEARCH AND RECOMMENDATIONS

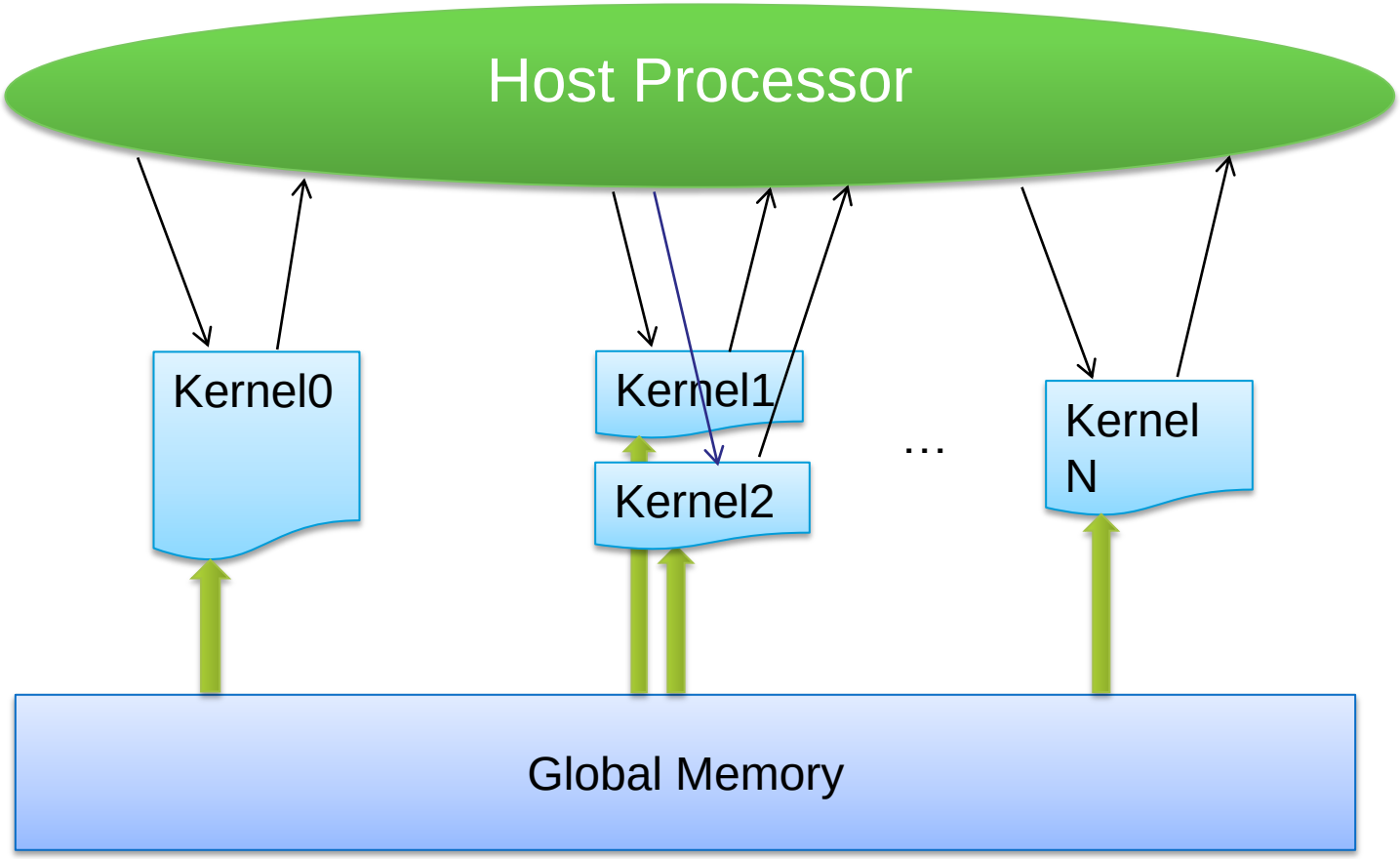
Recommendations

- Fundamentally new areas of FPGA research need to be undertaken
  - Standard programming models
  - Tool usability (debugging, profiling)
- Develop boards that are meant for algorithm acceleration
- Have the entire software stack ready to support this board
  - A high level language compile (OpenCL or others)
  - Debugging & Profiling
  - Libraries that are pre-optimized for best possible implementation on the FPGA

Recommendations (2)

- Don’t underestimate the GPU
- Potential to infiltrate the “traditional server” market
  - Eg. Speed up database queries for companies like Amazon & Ebay
  - IEEE Spectrum Article : "Why Graphics Processors will Transform Database Processing", Blas and Kladeway, Oracle Corporation
- FPGA hardware is ideally suited to these kinds of matching algorithms
  - Need tools for people to harness the power

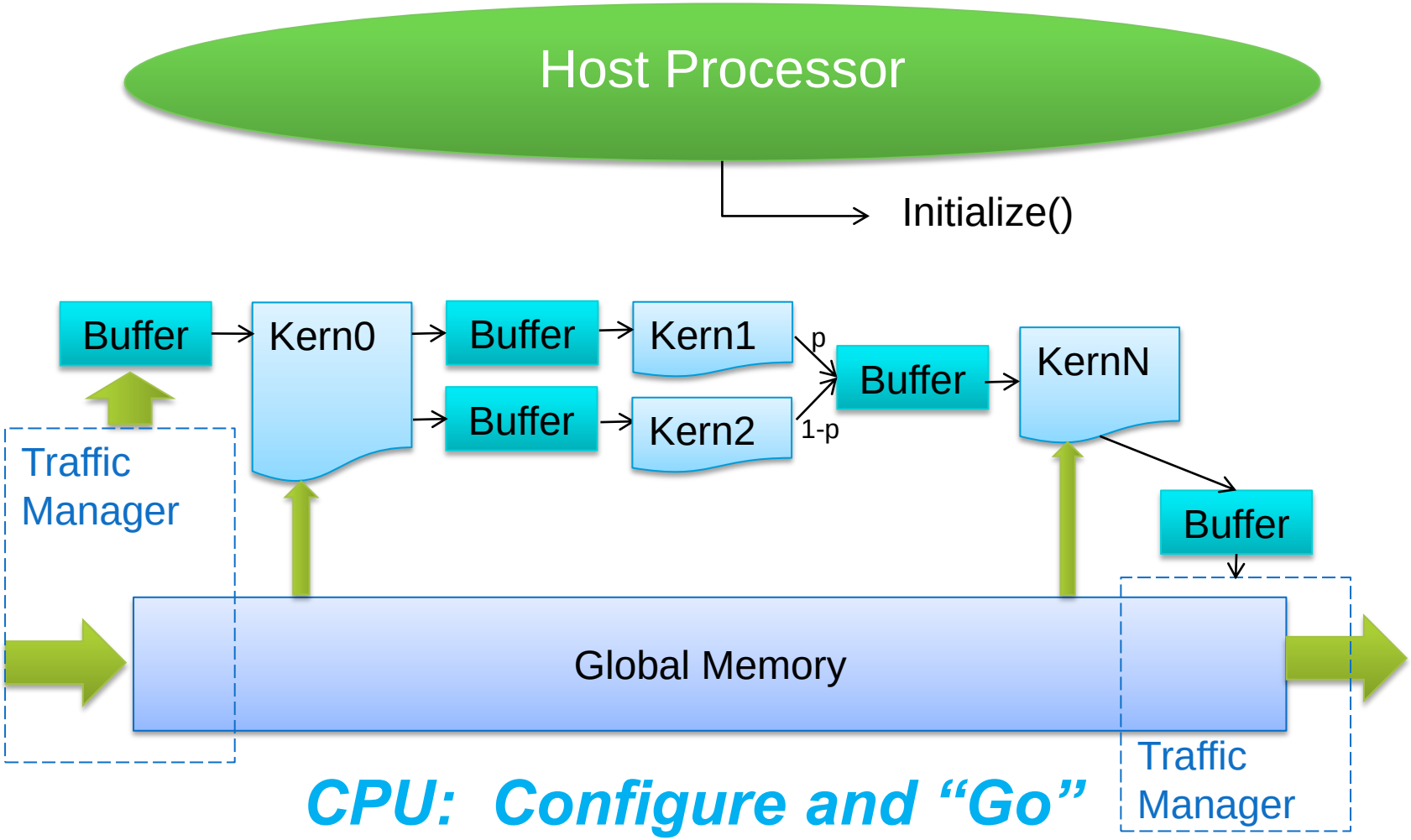
Current OpenCL System Architecture



High demand on CPU

Memory-to-memory paradigm

Desired Architecture (OpenCL Pipes)



CPU: Configure and "Go"

Stream orientation when needed