BIG DATA SEMINAR

GPU and Xeon Phi

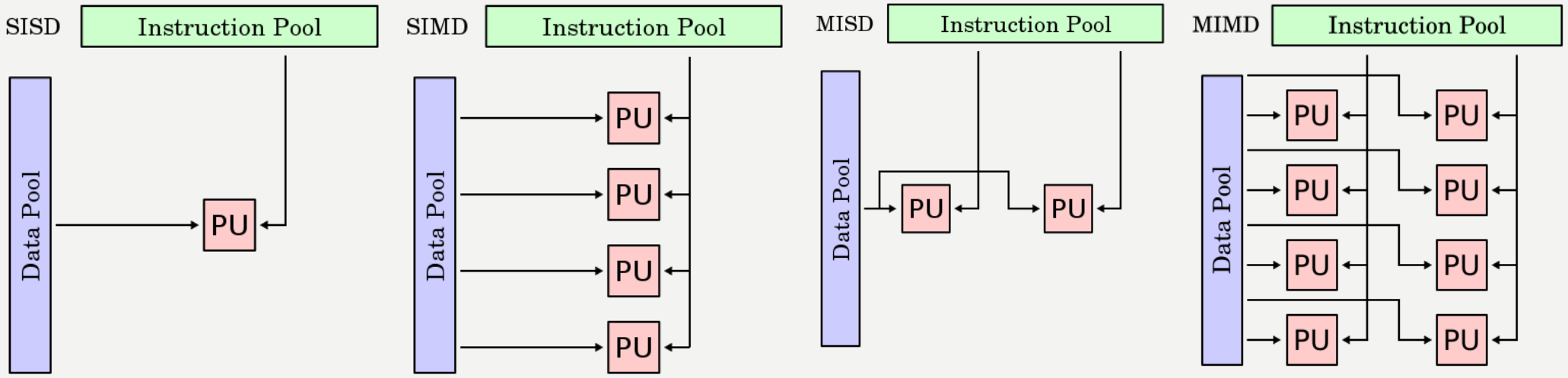
1. LE TRUNG HIEU
2. HUYNH QUANG LAN
3. HOANG VAN CONG
4. DUONG TAN THANH

Layout

- High Performance Computing and Big Data
- Graphics Processing Unit (GPU)
- Xeon Phi
- GPU vs Xeon Phi

High Performance Computing and Big Data

Flynn's taxonomy



https://en.wikipedia.org/wiki/Flynn's_taxonomy

HPC and Big Data

HPC systems are purpose built to parallelize and process complex computational algorithms.

Big Data system are purpose built to handle data intensive application. The major issues here were scalability, reliability and availability, not so much computational power.

https://www2.wwt.com/all-blog/how-gpus-and-high-performance-computing-can-augment-big-data/

HPC and Big Data (cont.)

With the grow of the volume of the data, more computational power is needed in Big Data systems.

A future of heterogeneous system of CPUs and GPUs

- GPUs implementation reduce the running time of the time series classification on large dataset from half a day to 2 minutes.
- Baidu and Google research groups using GPUs on their deep learning system and reduce the training time from weeks or months to days or even hours.

GPU

Graphics Processing Unit

GPUs are "designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device", wrote Wikipedia.

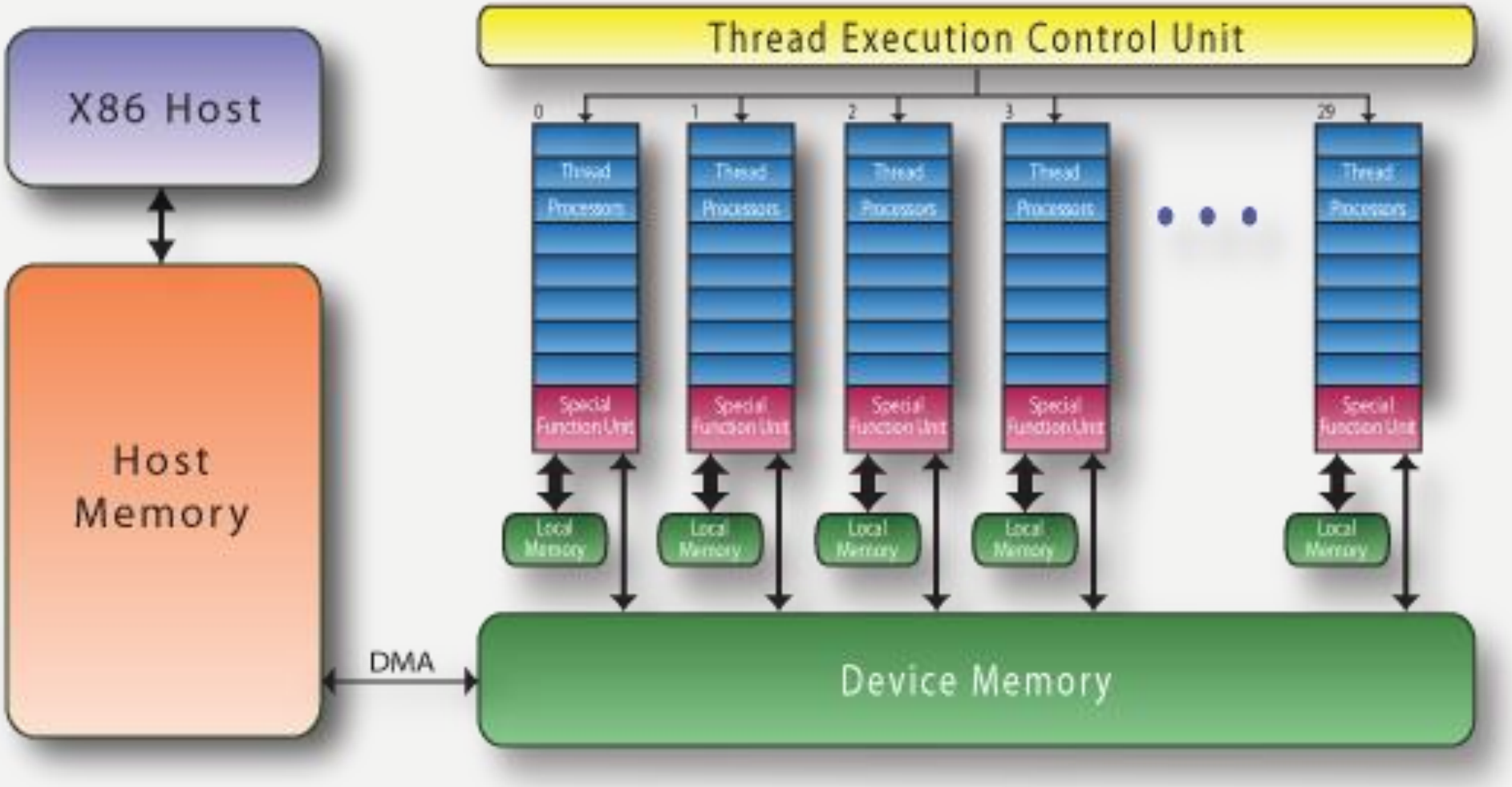
Applications Characteristics

Computational requirements are large.

Parallelism is substantial.

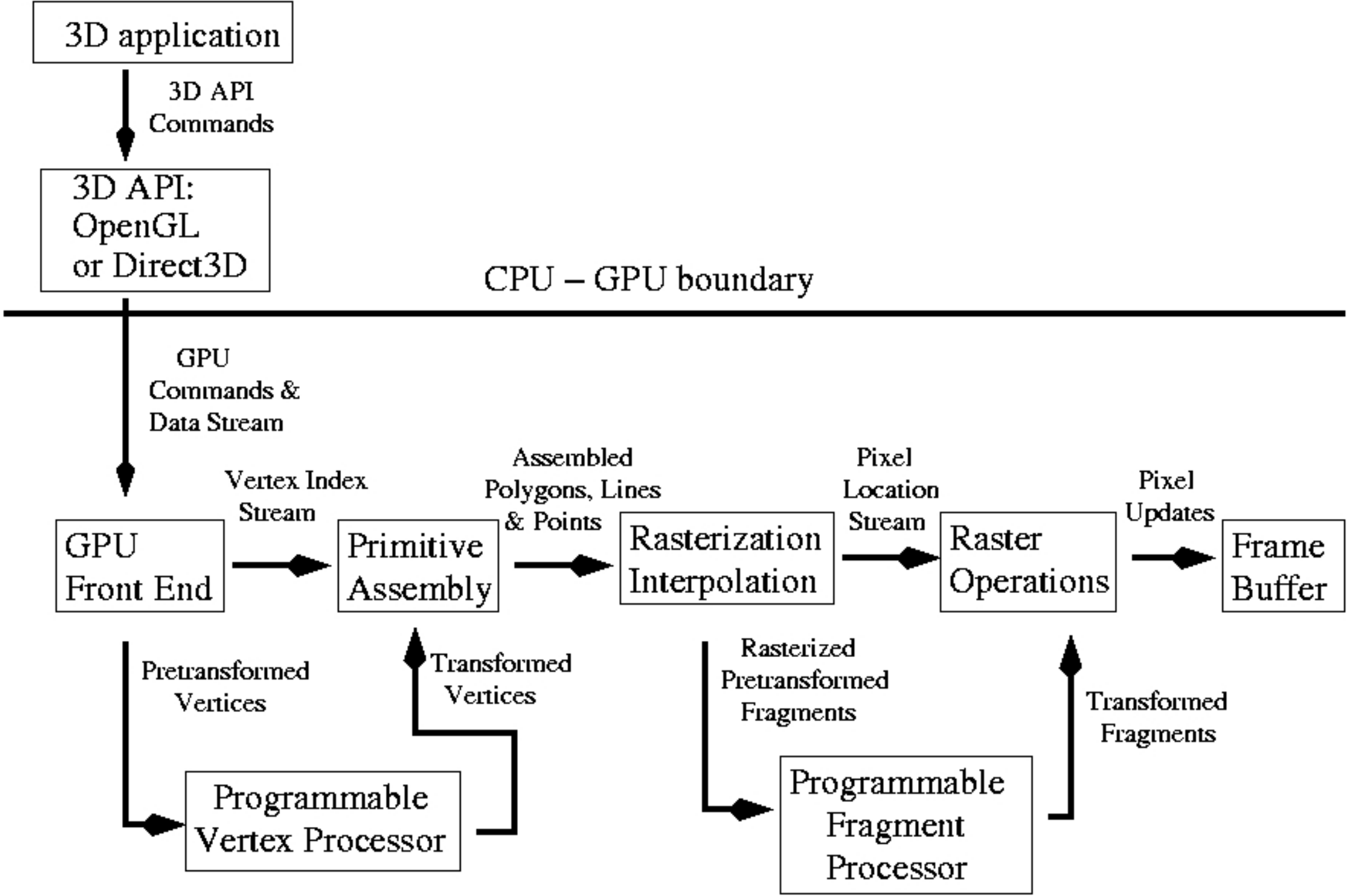
Throughput is more important than latency.

GPU Architecture



The Graphics Pipeline

- Vertex Operations
- Primitive Assembly
- Rasterization
- Fragment Operations
- Composition


3D application
3D API Commands
3D API: OpenGL or Direct3D
CPU – GPU boundary
GPU Commands & Data Stream
GPU Front End
Vertex Index Stream
Primitive Assembly
Assembled Polygons, Lines & Points
Rasterization Interpolation
Pixel Location Stream
Raster Operations
Pixel Updates
Frame Buffer
Pretransformed Vertices
Transformed Vertices
Programmable Vertex Processor
Rasterized Pretransformed Fragments
Transformed Fragments
Programmable Fragment Processor

Evolution of GPU Architecture

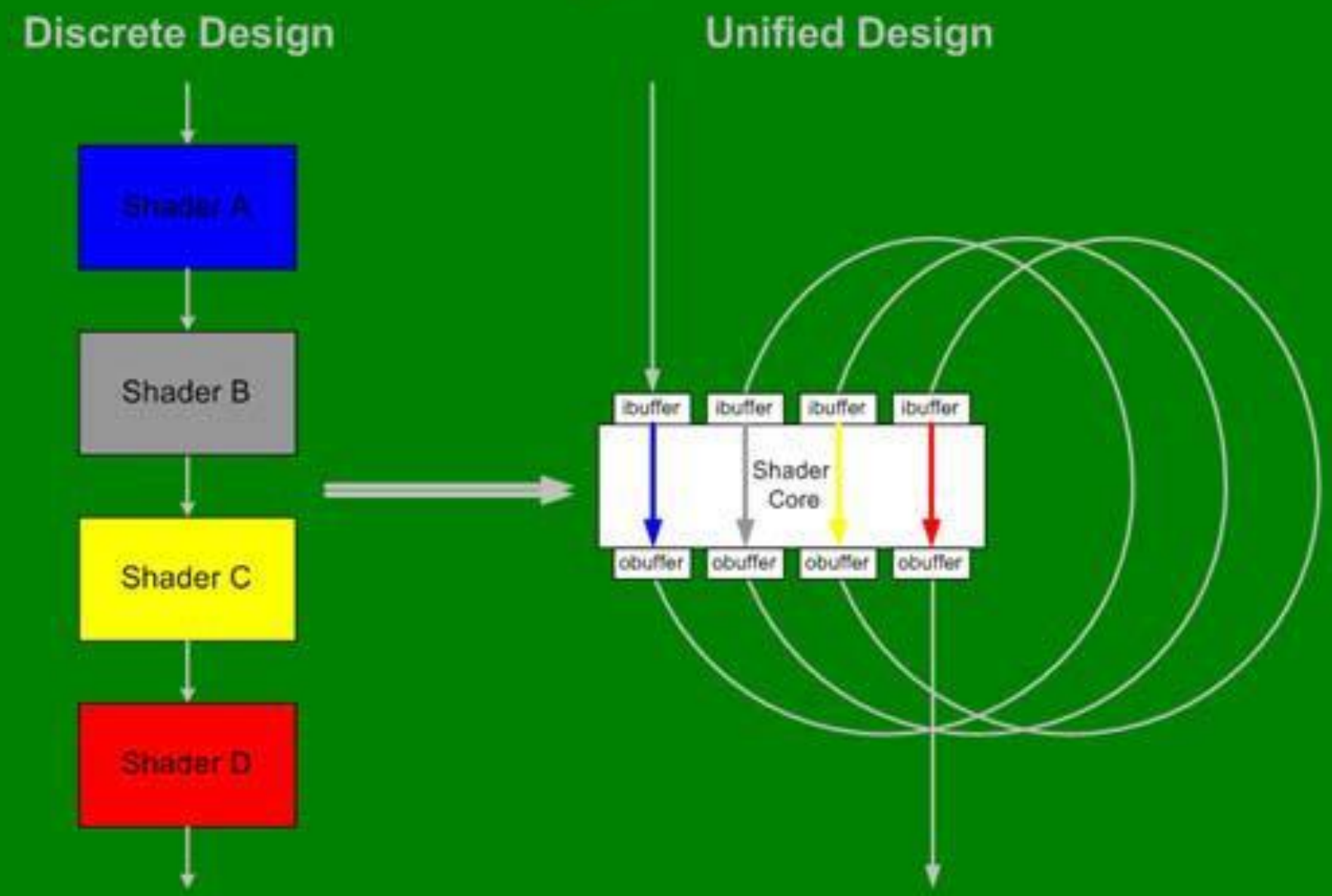
Disadvantage of the GPU task-parallel pipeline is load balancing.

Replacing the fixed-function per-vertex and per-fragment operations with user-specified programs run on each vertex and fragment

GPGPU programmers can now target that programmable unit directly, rather than the previous approach of dividing work across multiple hardware units.

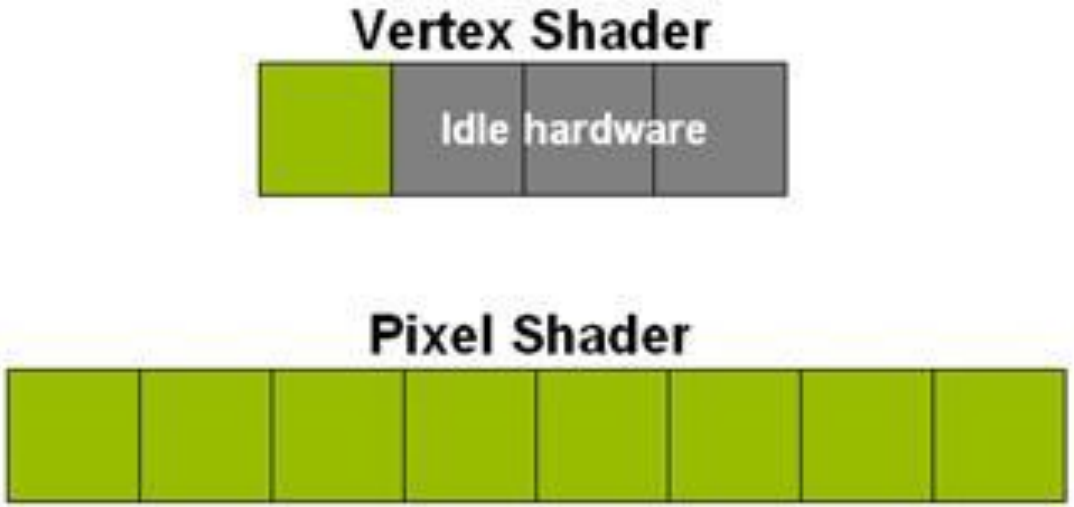
=> GPUs support the unified Shader Model. (Xenos GPU)

Unified Design



Why unify?

Heavy Geometry
Workload Perf = 4



Heavy Pixel
Workload Perf = 8

Why unify?

Unified Shader

Heavy Geometry
Workload Perf =12

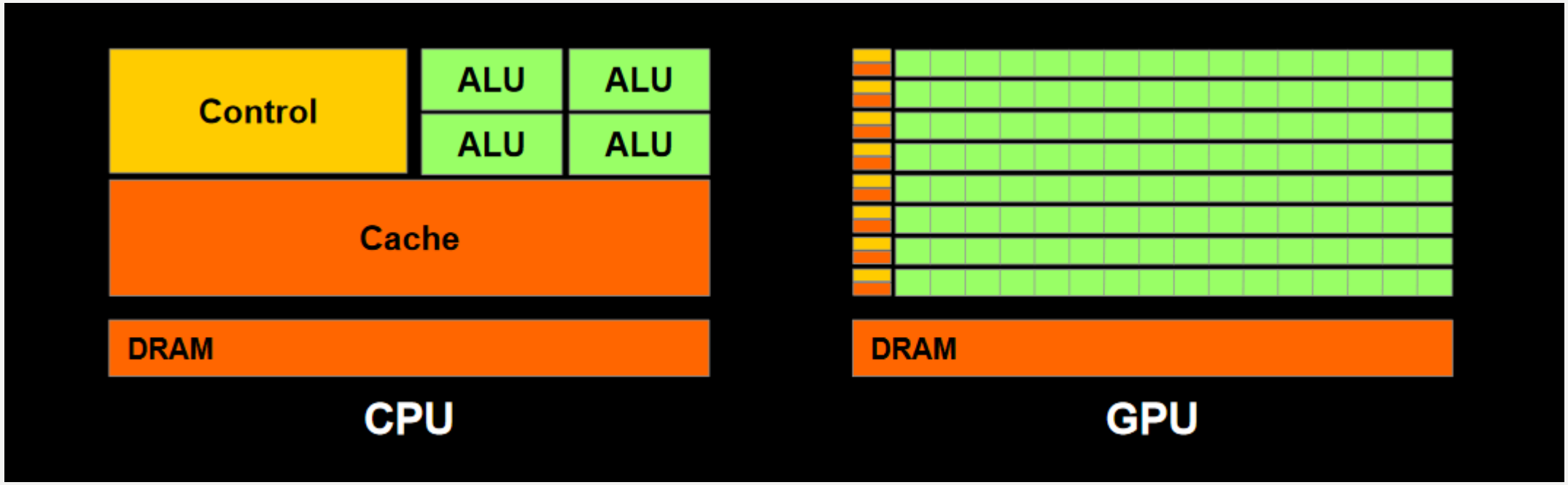
Unified Shader

Heavy Pixel
Workload Perf = 12

GPUs and CPUs

CPU	GPU
Low latency low throughput	High latency high throughput
Need a large cache	Do not need a large cache
Task parallelism	Data parallelism
Multi-threaded cores	Single Instruction Multiple Thread cores
10s of threads	10,000s of threads

GPUs and CPUs (cont.)



Rege, A. (2009). An Introduction to Modern GPU Architecture. *NVIDIA Presentation*.

The GPU Programming Model

Singleprogram multiple-data (SPMD) programming model.

- Many parallel elements, each processed in parallel by a single program.
- Elements cannot communicate with each other.

Operate on 32-bit integer or floatingpoint data.

Read data from a shared global memory (a “gather” operation).

Write back to arbitrary locations in shared global memory (“scatter”).

Elements are grouped together into blocks, and blocks are processed in parallel.

General-Purpose Computing on the GPU

Mapping general-purpose computation onto the GPU uses the graphics hardware in much the same way as any standard graphics application.

GPGPU is today's GPU computing applications.

General Purpose Computing on Graphics Processing Units

General Purpose Computing on Graphics Processing Units (GPGPU) is the use of GPUs to perform computation that is traditionally handled by CPU.

Many GPUs can be connected together provides advantages that multiple CPUs do not offer.

CUDA

It's a parallel computing platform and API model created by NVIDIA.

CUDA was an acronym for **Compute Unified Device Architecture** but NVIDIA dropped the use of the acronym.

Programming a GPU for Graphics

1. The programmer specifies geometry that covers a region on the screen. The rasterizer generates a fragment at each pixel location covered by that geometry.
2. Each fragment is shaded by the fragment program.
3. The fragment program computes the value of the fragment by a combination of math operations and global memory reads from a global "texture" memory.
4. The resulting image can then be used as texture on future passes through the graphics pipeline.

Programming a GPU for General-Purpose Programs (Old)

1. The programmer specifies a geometric primitive that covers a computation domain of interest. The rasterizer generates a fragment at each pixel location covered by that geometry.
2. Each fragment is shaded by an SPMD generalpurpose fragment program.
3. The fragment program computes the value of the fragment by a combination of math operations and Bgather[accesses from global memory.
4. The resulting buffer in global memory can then be used as an input on future passes.

Programming a GPU for General-Purpose Programs (New)

1. The programmer directly defines the computation domain of interest as a structured grid of threads.
2. An SPMD general-purpose program computes the value of each thread.
3. The value for each thread is computed by a combination of math operations and both "gather" (read) accesses from and "scatter" (write) accesses to global memory. Unlike in the previous two methods, the same buffer can be used for both reading and writing, allowing more flexible algorithms.
4. The resulting buffer in global memory can then be used as an input in future computation

Programming a GPU for General-Purpose Programs (New)

1. Programs are more often expressed in a familiar programming language.
   - NVIDIA's C-like syntax in CUDA programming
2. simpler and easier to build and debug.
3. The result is a programming model that allows its users to take full advantage of the GPU's powerful hardware but also permits an increasingly high-level programming model that enables productive authoring of complex applications

XEON Phi

What is Xeon Phi?

Massively-parallel many-cores processor

Target market

- Supercomputer
- Enterprise
- High-end workstation

Competitors

NVIDIA TESLA

History

2006

The **Larrabee** microarchitecture was introduced as the main architecture for an Intel **GPGPU** chip

History

2009

Single Chip Cloud Computer (SCC) was introduced. It has **48** distinct physical cores that communicate through architecture similar to that of a cloud computer data center.

History

2010

The **Knights Ferry** MIC (Many Integrated Cores) prototype board derived the **Larrabee** project was offered as a PCIe card with **32** cores and 4 threads per cores.

History

2011

Intel showed an early silicon version of **Knights Corner** processor, the first Intel's many-cores commercial product.

History

2012

Xeon Phi was decided to be the brand name of all Intel product based on MIC architecture. (around 1.01 teraFLOPS of double floating point instructions with 320GB/s memory bandwidth at 300W) (22nm)

History

2013

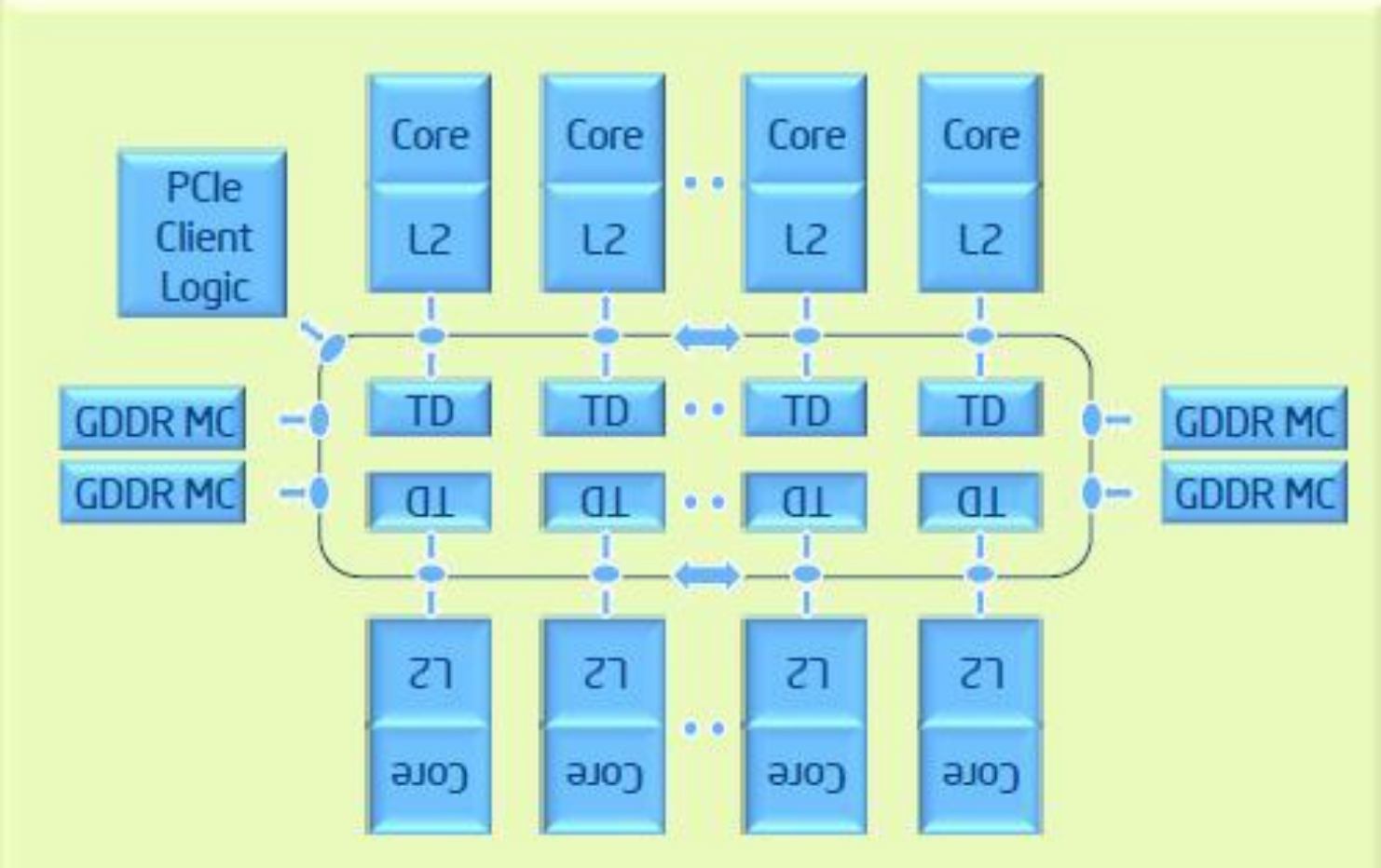
Intel introduced the second generation of Xeon Phi product lines with the code name of **Knights Landing**

History

2016

Intel launched Xeon Phi product family **x200** based on Knights Landing architecture (14nm) (close to 3.5 teraFLOPS at peak double precision performance)

Xeon Phi architecture



GPU vs. Xeon Phi

Xeon Phi 7290

3.45 TFLOPS (FP64)

72 cores (1.5 GHz)

245 W Thermal Design Power

$ 6,500

NVIDIA Tesla P100

5.3 TFLOPS (FP64)

1792 CUDA cores (FP64)

300 W Thermal Design Power

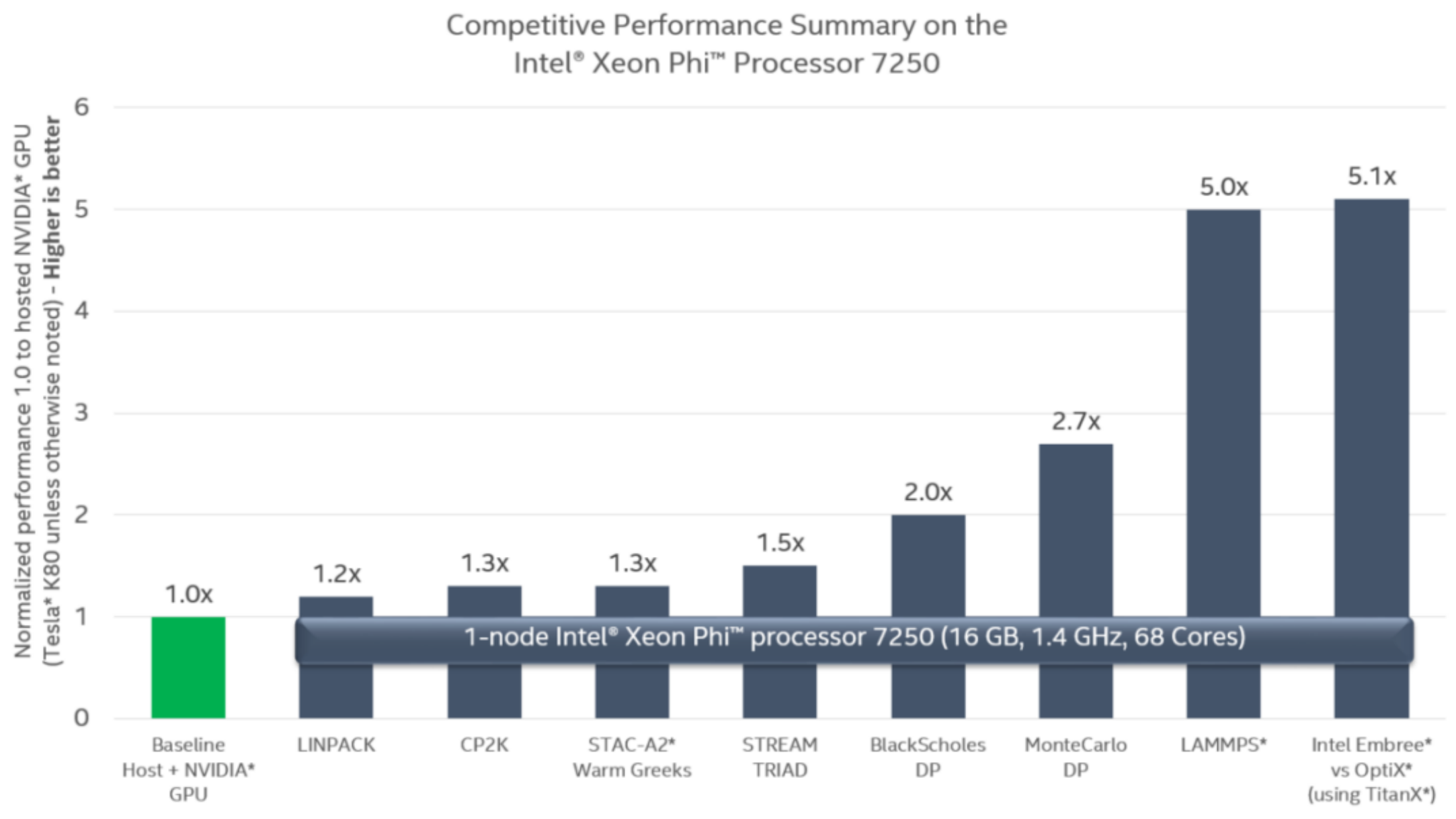
$ 7,000

“

Moving a code to MIC might involve sitting down and adding a couple of lines of directives that takes a few minutes, moving a code to a GPU is a project.

Stanzione

https://www.hpcwire.com/2011/04/21/tacc_steps_up_to_the_mic/



http://www.intel.com/content/www/us/en/benchmarks/server/xeon-phi/xeon-phi-competitive-performance.html

References

- El-Khamra, Y., Gaffney, N., Walling, D., Wernert, E., Xu, W., & Zhang, H. (2013). Performance evaluation of R with Intel Xeon Phi coprocessor. In *2013 IEEE International Conference on Big Data* (pp. 23–30). IEEE. http://doi.org/10.1109/BigData.2013.6691695
- Coates, A., Huval, B., Wang, T., Wu, D., & Ng, A. Y. (2013). Deep learning with COTS HPC systems. *Proceedings of The 30th International Conference on Machine Learning*, 1337–1345. http://doi.org/10.1177/0278364910371999
- Chang, K. W., Deka, B., Hwu, W. M. W., & Roth, D. (2012). Efficient pattern-based time series classification on GPU. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 131–140. http://doi.org/10.1109/ICDM.2012.132

References

- DeanV, J. (2015). Large Scale Deep Learning. In *GPU Technology Conference*.
- Owens, J., & Houston, M. (2008). GPU computing. *Proceedings of the IEEE*, *96*(5), 879–899. http://doi.org/10.1109/JPROC.2008.917757
- Rege, A. (2009). An Introduction to Modern GPU Architecture. *NVIDIA Presentation*.
- NVIDIA. (2016). NVIDIA Tesla P100 Whitepaper, 45.
- NVIDIA. (2008). Parallel computing with CUDA. *NVIDIA Presentation*.
- Intel. (2013). Intel ® Xeon Phi ™ Coprocessor Architecture Overview. *Intel Presentation*.
- Graphics processing unit – Wikipedia (https://en.wikipedia.org/wiki/Graphics_processing_unit)

References (cont.)

- Xeon Phi – Wikipedia (https://en.wikipedia.org/wiki/Xeon_Phi)
- Flynn's taxonomy – Wikipedia (https://en.wikipedia.org/wiki/Flynn's_taxonomy)
- Intel® Xeon Phi™ Processor Competitive Performance (http://www.intel.com/content/www/us/en/benchmarks/server/xeon-phi/xeon-phi-competitive-performance.html)
- TACC Steps Up to the MIC (https://www.hpcwire.com/2011/04/21/tacc_steps_up_to_the_mic/)
- How GPUs and High Performance Computing can augment Big Data (https://www2.wwt.com/all-blog/how-gpus-and-high-performance-computing-can-augment-big-data)