Parallel accelerators

Přemysl Šůcha, Jan Dvořák and Libor Bukata

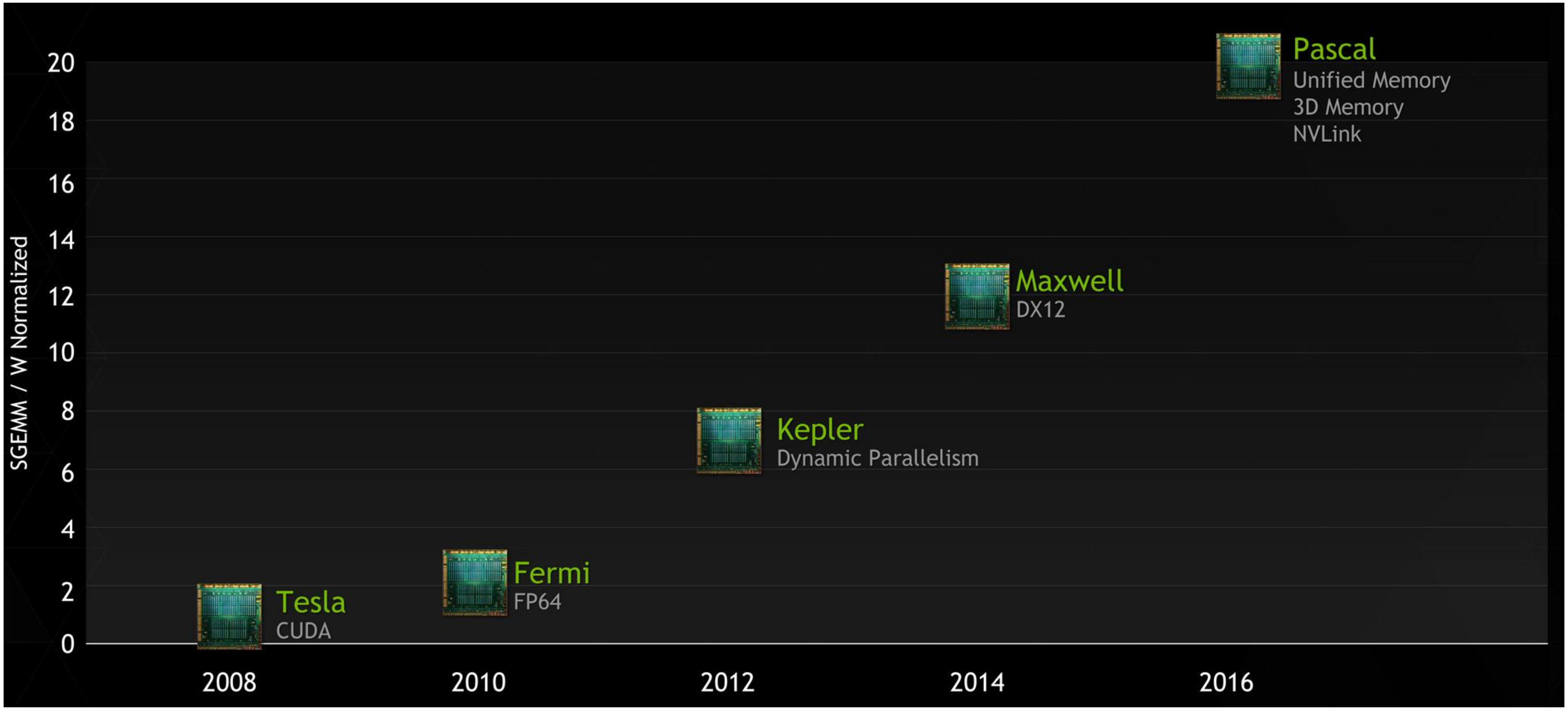
``Parallel algorithms'', 2016/2017
CTU/FEL

Topic Overview

- Graphical Processing Units (GPU) and CUDA
- Matrix transpose on CUDA
- Intel Xeon Phi
- Matrix equations on Xeon Phi

Graphical Processing Units

Graphical Processing Units



SGEMM - Single precision floating General Matrix Multiply

Graphical Processing Units

- GPU is especially well-suited to address problems that can be expressed as **data-parallel computations**.
- The same program is executed on many data elements in parallel - with high **arithmetic intensity** - the ratio of arithmetic operations to memory operations.
- Applications that process **large data sets** can use a data-parallel programming model to speed up the computations (3D rendering, image processing, video encoding, …)
- Many algorithms **outside the field of image rendering and processing** are accelerated by data-parallel processing too (general signal processing, physics simulation, finance, computational biology).

Streaming Multiprocessor (SM)

Streaming Multiprocessor (SM) - Pascal

- **Pascal** SM incorporates 64 single-precision CUDA Cores (Core), 32 double precision CUDA Cores (DP Unit), 16 **SFUs** (accelerate transcendental functions) and 16 **LD/ST** (Load / Store) units.
- SM is partitioned into two processing block.
- Each **warp scheduler** (one per processing block) is capable of dispatching two warp instructions per clock (1 warp = 32 threads).
- SMM has a dedicated **register file** and **shared memory**.

GPU Architecture - Pascal

GPU Architecture

Tesla Products	Tesla K40	Tesla M40	Tesla P100
GPU	GK110 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)
SMs	15	24	56
FP32 CUDA Cores / SM	192	128	64
FP32 CUDA Cores / GPU	2880	3072	3584
Base Clock	745 MHz	948 MHz	1328 MHz
Peak FP32 GFLOPs	5040	6840	10600
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB
Register File Size / SM	256 KB	256 KB	256 KB
TDP	235 Watts	250 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion

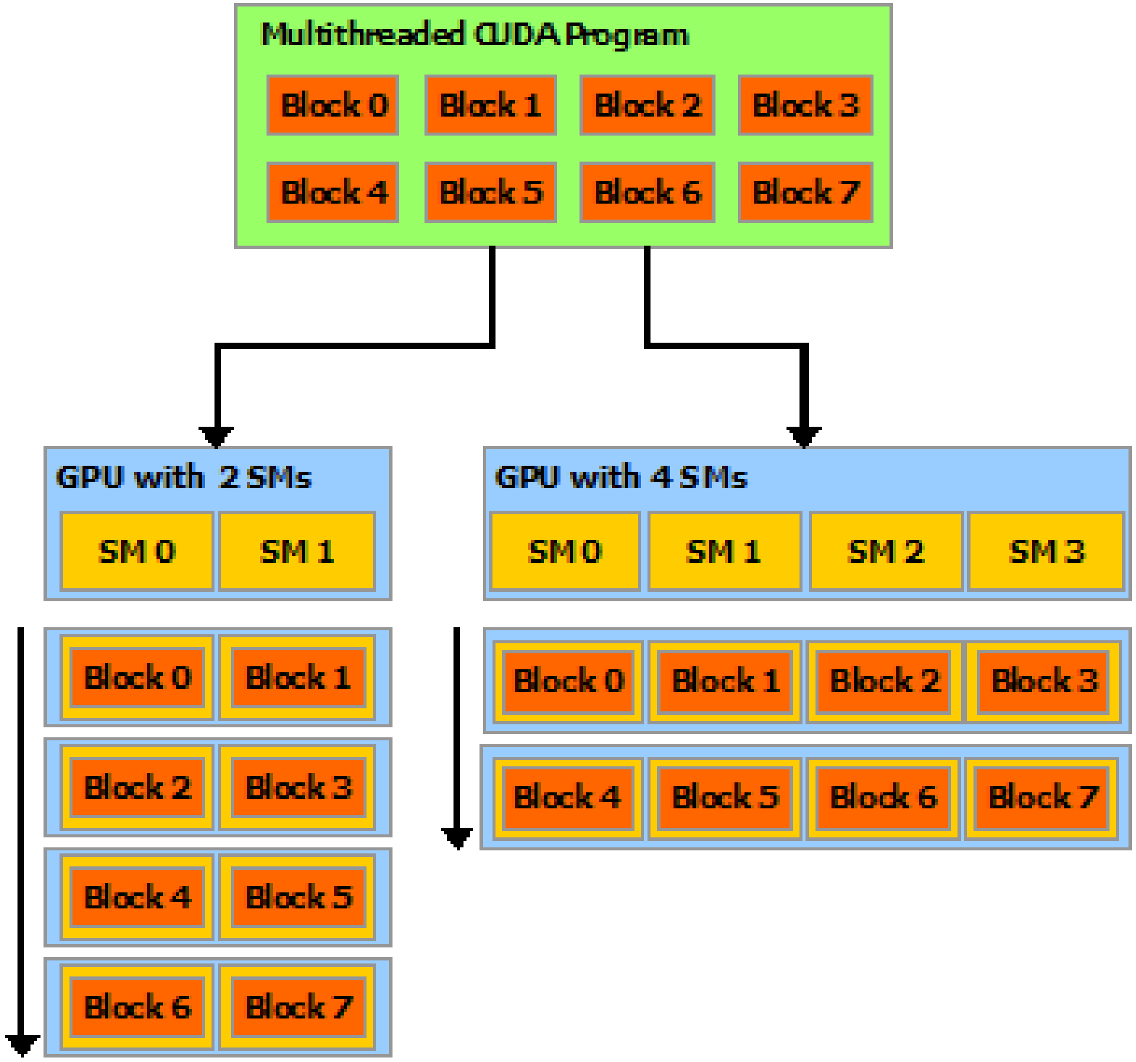
Single-Instruction, Multiple-Thread

- SIMT is an execution model where single instruction, multiple data (**SIMD**) **is combined with multithreading**.
- The SM creates, manages, schedules, and executes threads in groups of 32 parallel threads called **warps**.
- A warp start together at the same program address, but they have their **own instruction address counter** and **register state** and are therefore **free to branch** and **execute independently**.

CUDA

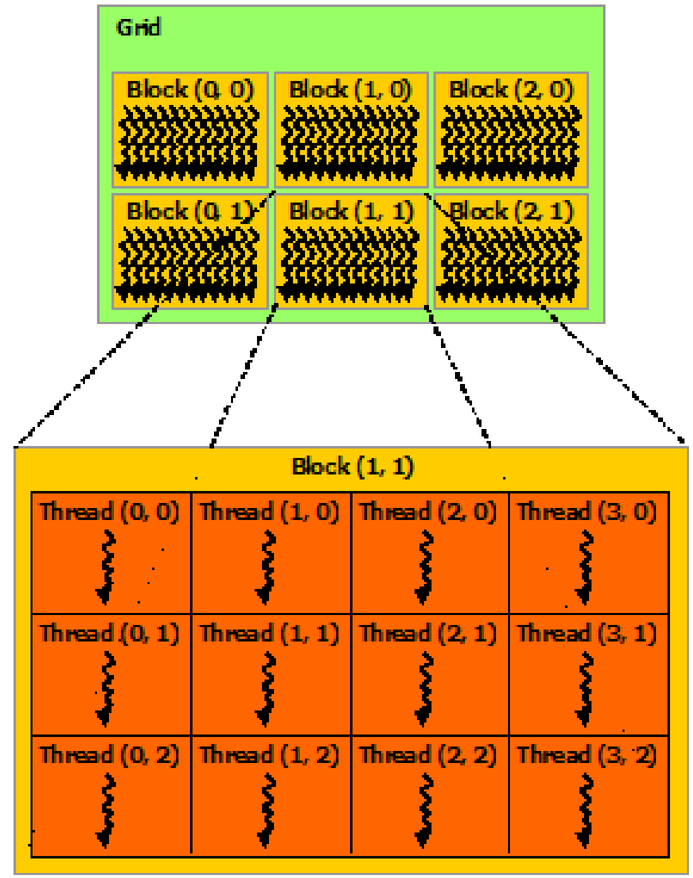
- The NVIDIA GPU architecture is built around a **scalable** array of multithreaded Streaming Multiprocessors (SMs).
- CUDA (Compute Unified Device Architecture) provides a way how a CUDA **program can be executed on any number of SMs**.
- A multithreaded program is partitioned into **blocks** of threads that execute independently from each other, so that a GPU with more multiprocessors will automatically execute the program in less time than a GPU with fewer multiprocessors.

CUDA


Multithreaded CUDA Program
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
GPU with 2 SMs
SM 0
SM 1
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
GPU with 4 SMs
SM 0
SM 1
SM 2
SM 3
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7

Grid/Block/Thread

- threads can be identified using a 1-D, 2-D, or 3-D thread index, forming a 1-D, 2-D, or 3-D block of threads, called a **thread block**.
- Blocks are organized into a 1-D, 2-D, or 3-D **grid** of **thread blocks**.



2-D grid with 2-D thread blocks

Kernel

- CUDA C extends C by allowing the programmer to define C functions, called **kernels**.

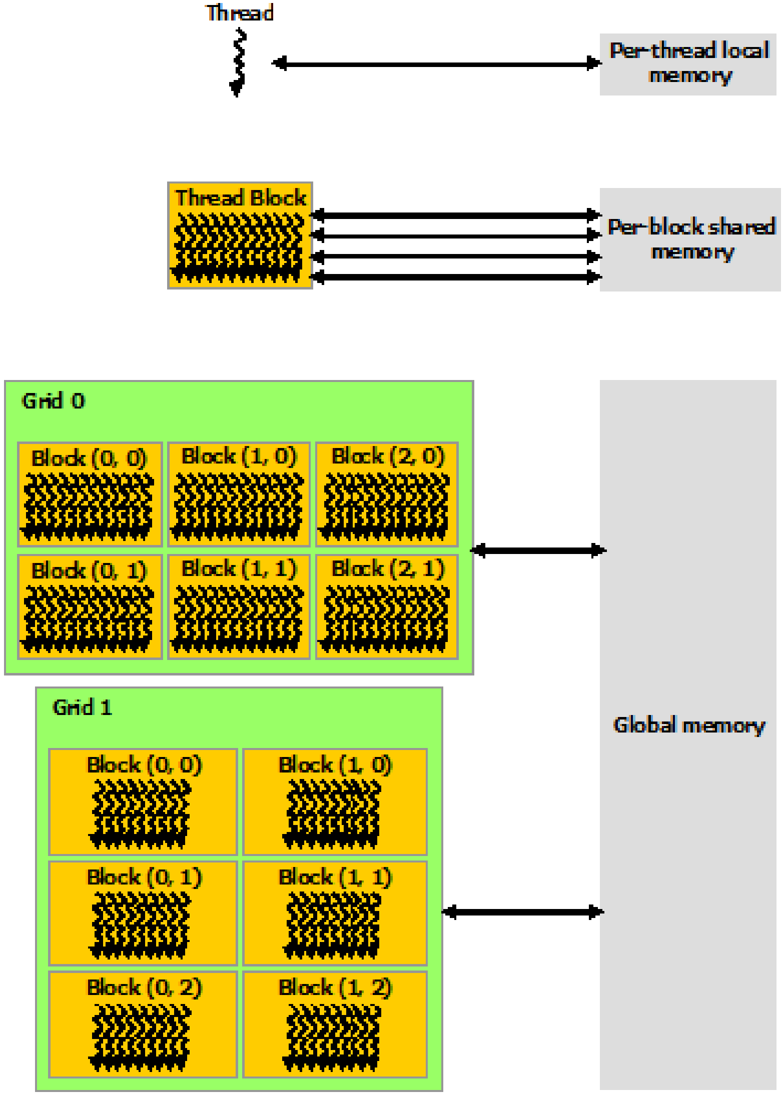
```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
        int i = threadIdx.x;
        C[i] = A[i] + B[i];
}

int main()
{ ...
        // Kernel invocation with N threads inside 1 thread block
        VecAdd<<<1, N>>>(A, B, C);
}
```

- *threadIdx* is a 3-component vector, so that threads can be identified using a 1-D, 2-D, or 3-D **thread index**.

Memory Hierarchy

- Each thread has private set of **registers** and **local memory**.
- Each thread block has **shared memory** visible to all threads of the block.
- All threads have access to the same **global memory**.
- There are also two additional read-only memory spaces accessible by all threads (**constant** and **texture memory**).

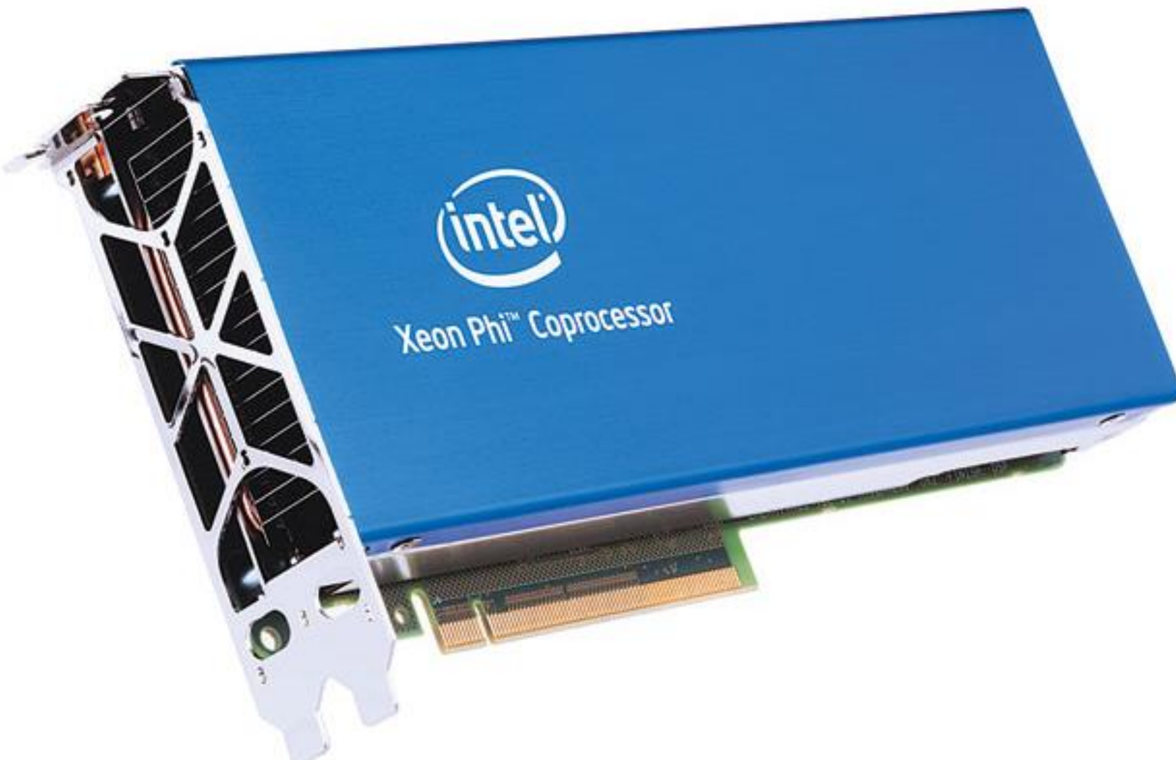


GPU Programming - Demo

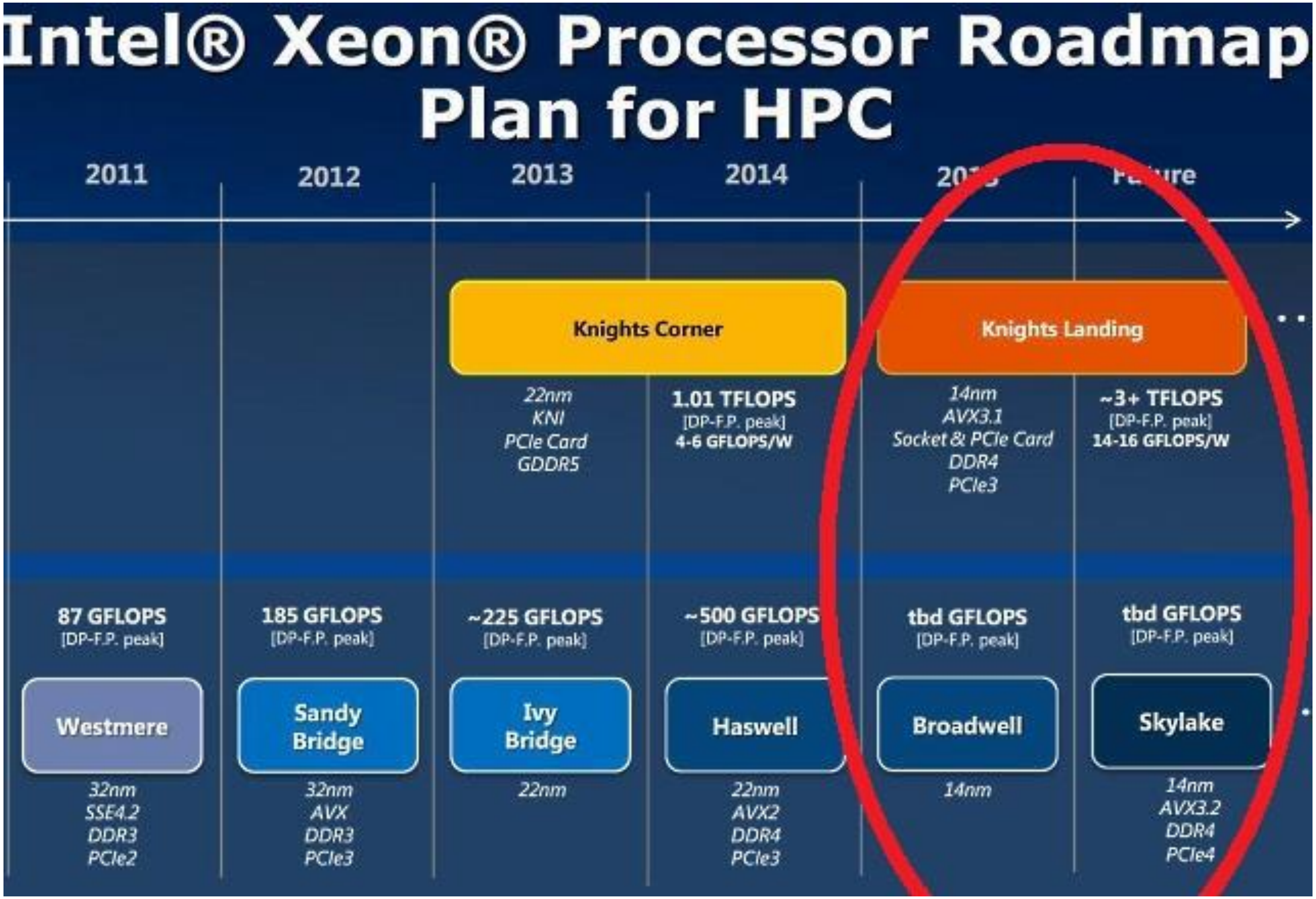
- Matrix transposition.

[1] Greg Ruetsch, *Optimizing Matrix Transpose in CUDA*, NVIDIA, 2009, http://www.cs.colostate.edu/~cs675/MatrixTranspose.pdf.

Intel Xeon Phi



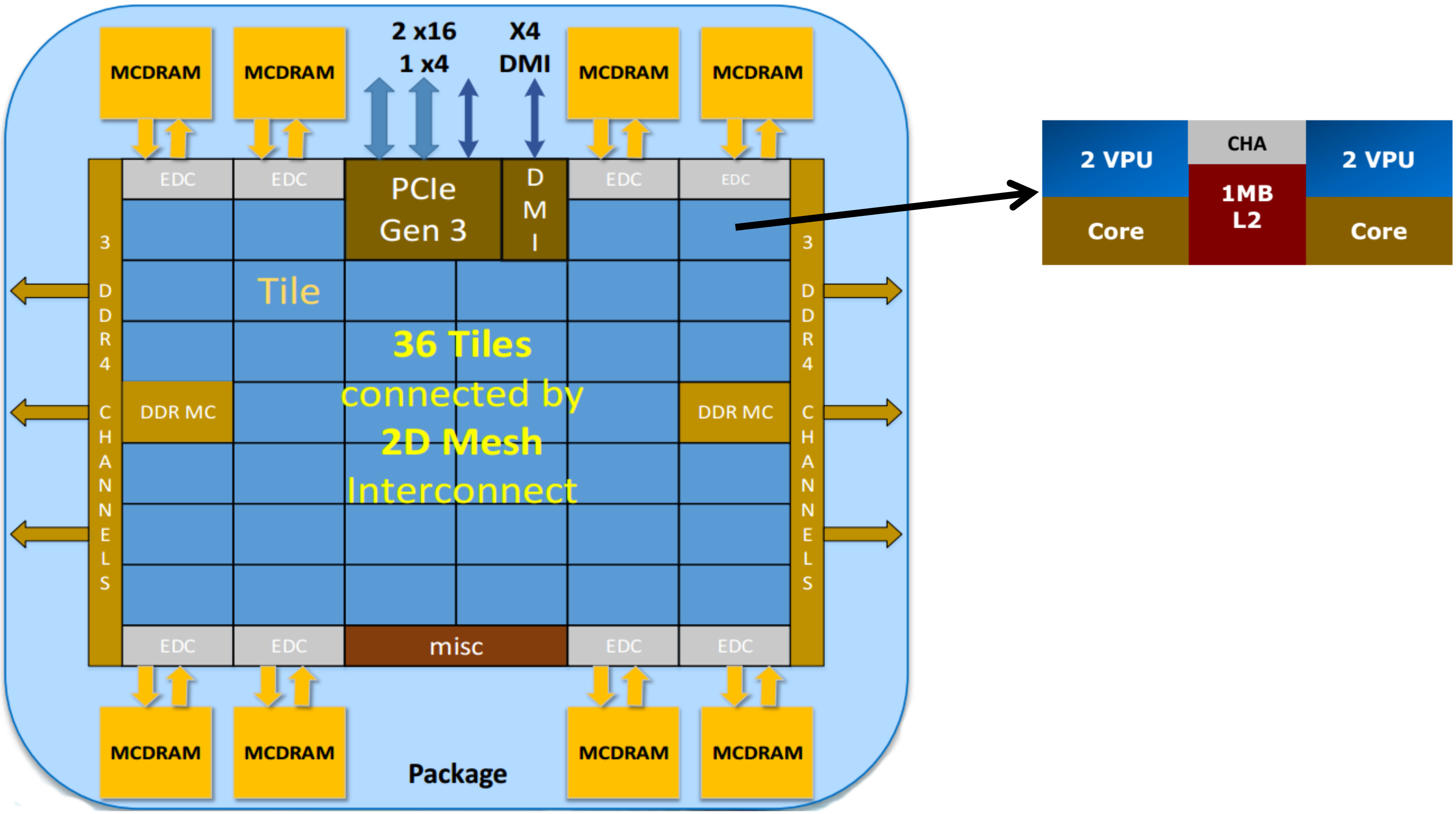
Intel Xeon Phi



Intel Xeon Phi

- Intel Xeon Phi coprocessors are designed to **extend the reach** of applications that have demonstrated the ability to fully utilize the scaling capabilities of **Intel Xeon processor-based systems**.
- Code compiled for Xeon processors can be run on an Xeon Phi (Knights Landing).
- For successful parallelization it requires a program with **lots of threads** and operations with **vectors**.

Knights Landing Architecture



Knights Landing Architecture

- The chip is constituted by **36 tiles** interconnected by 2D mesh.
- The tile has **two Cores** (Atom Silvermont architecture), **two vector processing units** (VPU) and 1M **L2 cache**.
- A tile can execute concurrently 4 threads.
- The tiles are interconnected a **cache-coherent 2D mesh**; which provides a higher bandwidth and lower latency compare to the 1D ring interconnect on Knights corner.
- The mesh enforces **XY routing** rule.

Knights Landing Architecture

- Xeon Phi has 2 types of memory: (i) **MCDRAM** (Multi-channel DRAM) and (ii) **DDR**.
- MCDRAM is a **high-bandwidth memory** integrated on the package. There are 8 of them 2 GB each.
- MCDRAM can be configured at boot time into one of **three modes**:
  - Cache mode – MCDRAM is a cache for DDR,
  - Flat mode – MCDRAM is a standard memory in the same address space as DDR,
  - Hybrid – a combination
- DDR is a **high-capacity memory** which is external to the Knight Landing package.

Vectorization

- Each tile has **two VPUs** (Vector Processing Unit).
- It is the heard of computation. It processes all floating point computations using SSE, AVX, AVX2, …, **AVX-512**.
- Thus each tile can **execute two 512-bit vector multiple-add instructions per cycle**, i.e. compute 32 double precision resp. 64 single precision floating point operation in each cycle.

Knights Corner vs. Knights Landing

Product Name	Intel® Xeon Phi™ Coprocessor 7120X (16GB, 1.238 GHz, 61 core)	Intel® Xeon Phi™ Processor 7290F (16GB, 1.50 GHz, 72 core)
Code Name	Knights Corner	Knights Landing
Lithography	22 nm	14 nm
Recommended Customer Price	N/A	$6401.00
# of Cores	61	72
Processor Base Frequency	1.24 GHz	1.50 GHz
Cache	30.5 MB L2	36 MB L2
TDP	300 W	260 W
Max Memory Size (dependent on memory type)	16 GB	384 GB
Max Memory Bandwidth	352 GB/s	490 GB/s

Offloading

- Choose **highly-parallel sections** of code to run on the coprocessor. Serial code offloaded to the coprocessor will run much slower than on the CPU.

```
int x, y[100];
void f()
{
    x = 55;
    // x sent from CPU, y computed on coprocessor
    ...
#pragma offload target(mic:0) in(x) nocopy(y)
{ y[50] = 66; }
…
#pragma offload target(mic:0) nocopy(x,y)
{ // x and y retain previous values }
}
```

Xeon Phi Programming - Demo

- Simple matrix equation.

[2] James Jeffers and James Reinders, *Intel Xeon Phi Coprocessor High-Performance Programming*, Morgan Kaufmann, 2013.

References

[1] Greg Ruetsch, *Optimizing Matrix Transpose in CUDA*, NVIDIA, 2009.

[2] James Jeffers and James Reinders, *Intel Xeon Phi Coprocessor High-Performance Programming*, Morgan Kaufmann, 2013.

[3] NVIDIA, *CUDA Toolkit Documentation v8.0*, 2016. (http://docs.nvidia.com/cuda/index.html)

[4] Avinash Sodani, *Knights Landing (KNL): 2nd Generation Intel® Xeon Phi™ Processor*, Intel, 2016. ()

[5] James Jeffers and James Reinders and Avinash Sodani, Intel Xeon Phi Processor High Performance Programming, 2nd Edition, Morgan Kaufmann, 2016.