HP-SEE

Introduction to heterogeneous parallel programming

www.hp-see.eu

Petar Jovanović
Scientific Computing Laboratory
Institute of Physics Belgrade
petarj@ipb.ac.rs

HP-SEE
High-Performance Computing Infrastructure
for South East Europe's Research Communities

The HP-SEE initiative is co-funded by the European Commission under the FP7 Research Infrastructures contract no. 261499

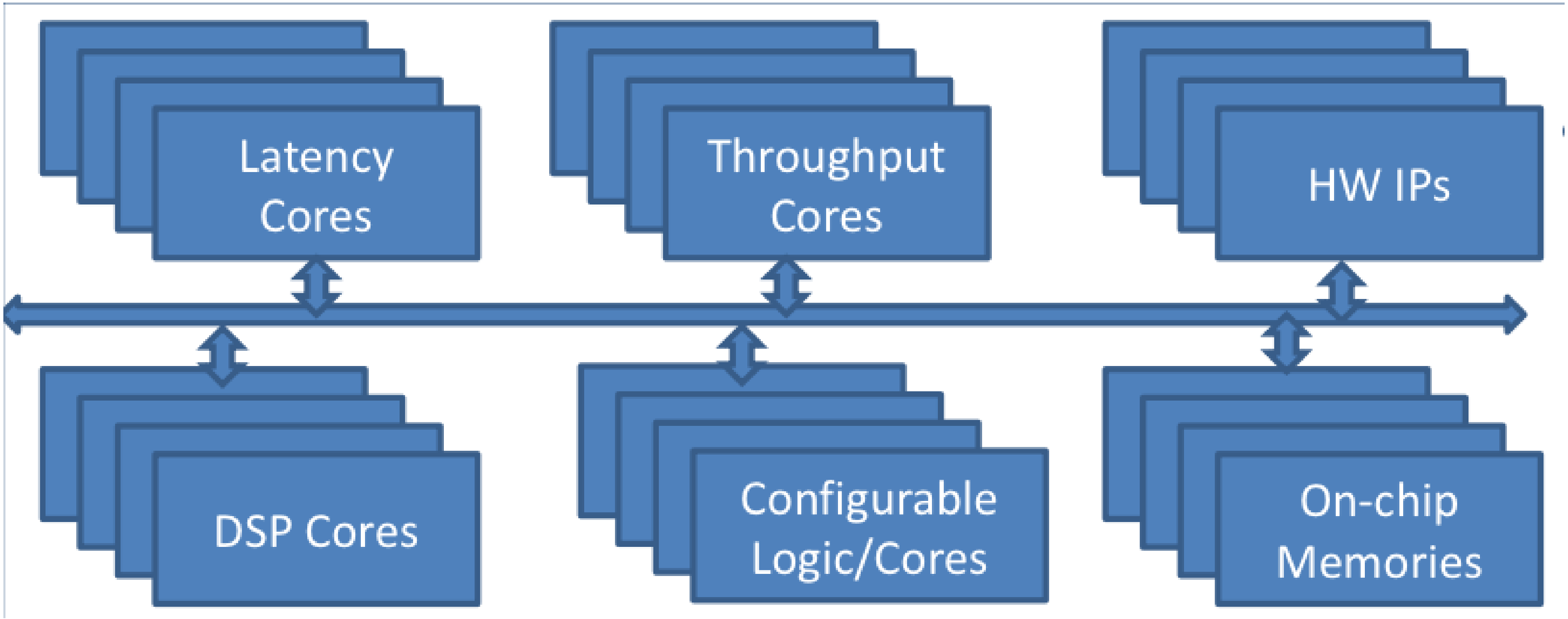
Agenda

HP-SEE
High-Performance Computing Infrastructure
for South East Europe's Research Communities

- Heterogeneous parallel system
  - Motivation
- CPU/GPU Architecture
  - Abstract view of a processor
  - Latency vs. Throughput
- CUDA Platform
  - Streaming Multiprocessor
  - Memory hierarchies
  - Scalable programming model

Heterogeneous Parallel Systems

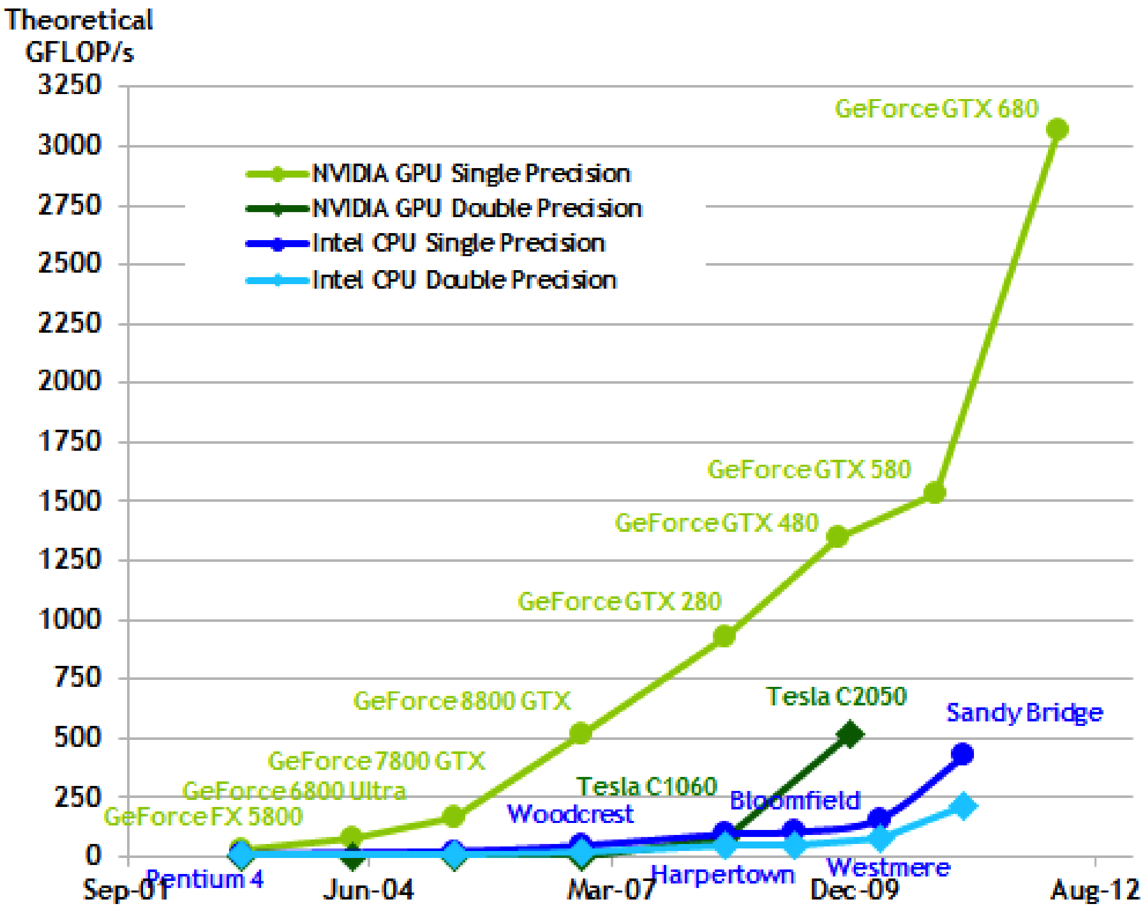
- Using the best match for the task at hand



Heterogeneous Parallel Systems: Motivation(1)

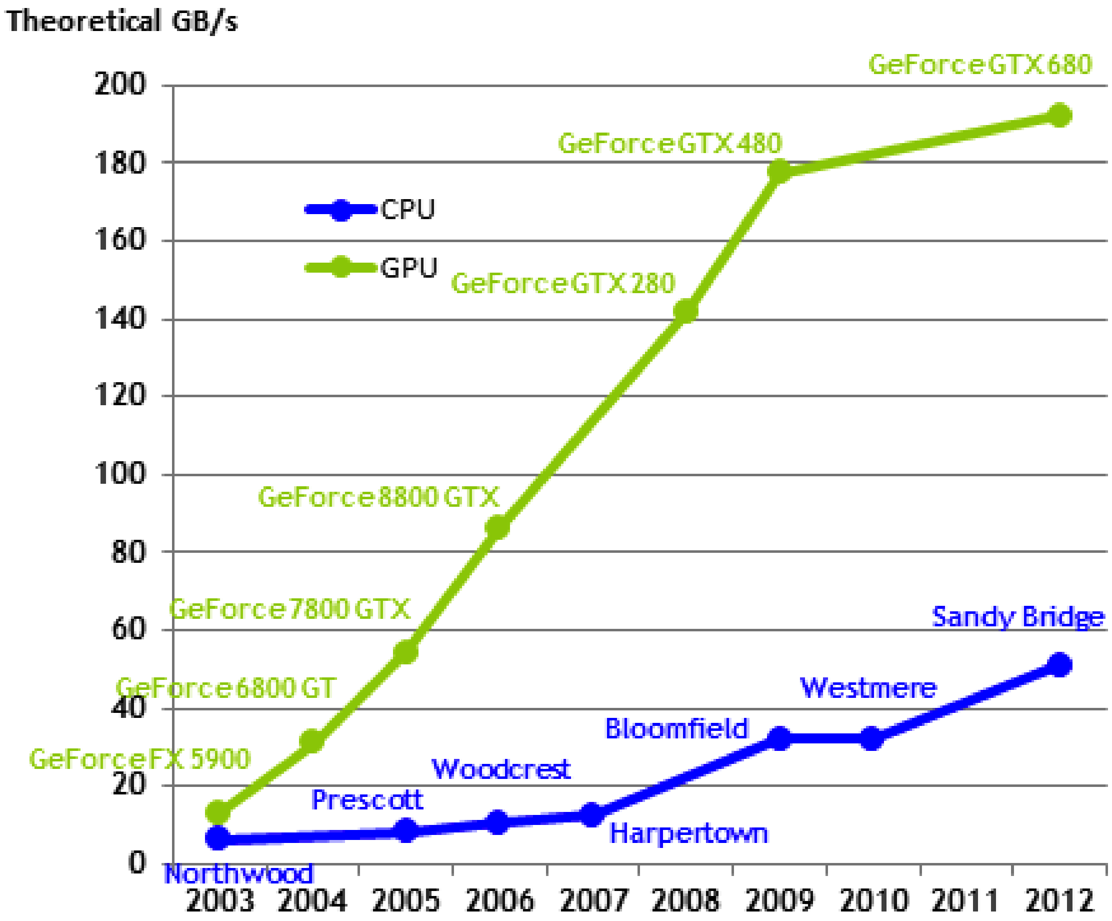
HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- Floating-Point Operations per Seconds for the CPU and GPU



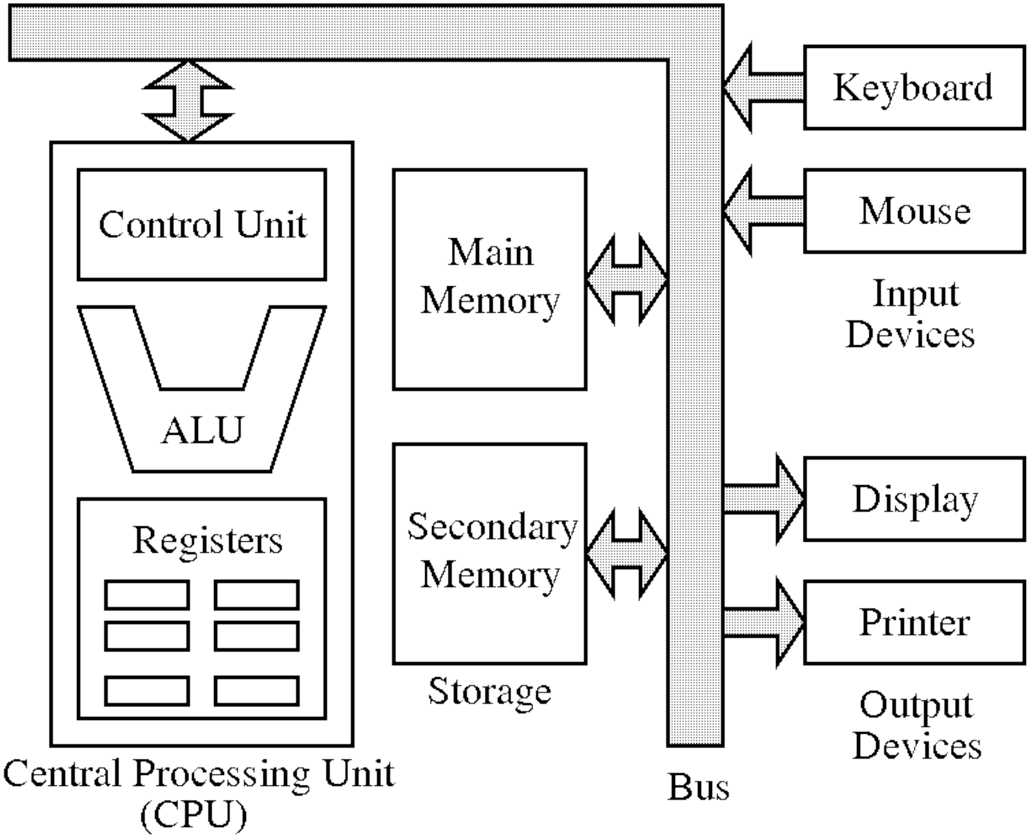
Heterogeneous Parallel Systems: Motivation(2)

- Memory Bandwidth for the CPU and GPU

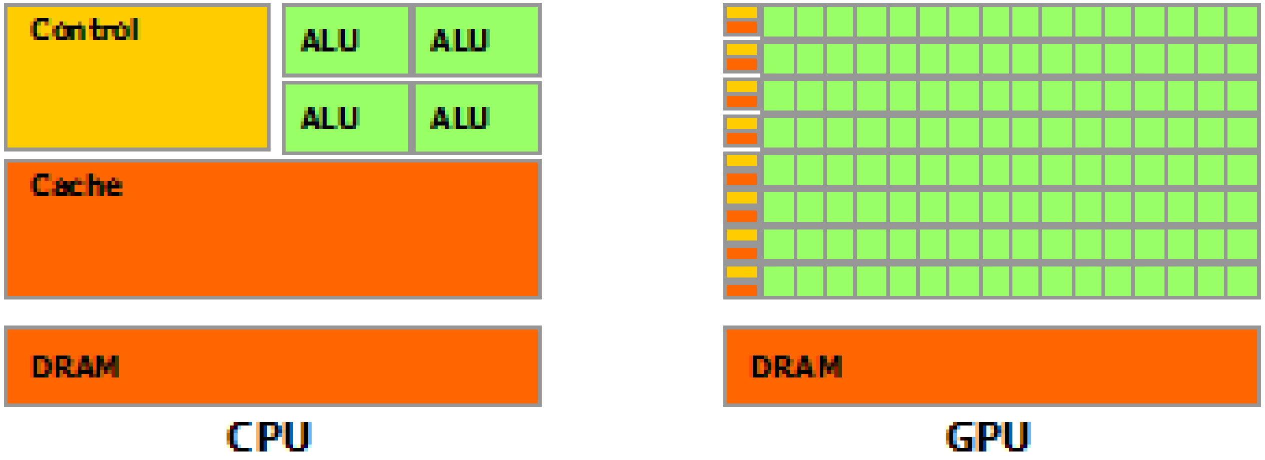


CPU/GPU Architecture: Abstract view of a processor

- The von Neumann architecture



CPU/GPU Architecture: Latency vs. throughput



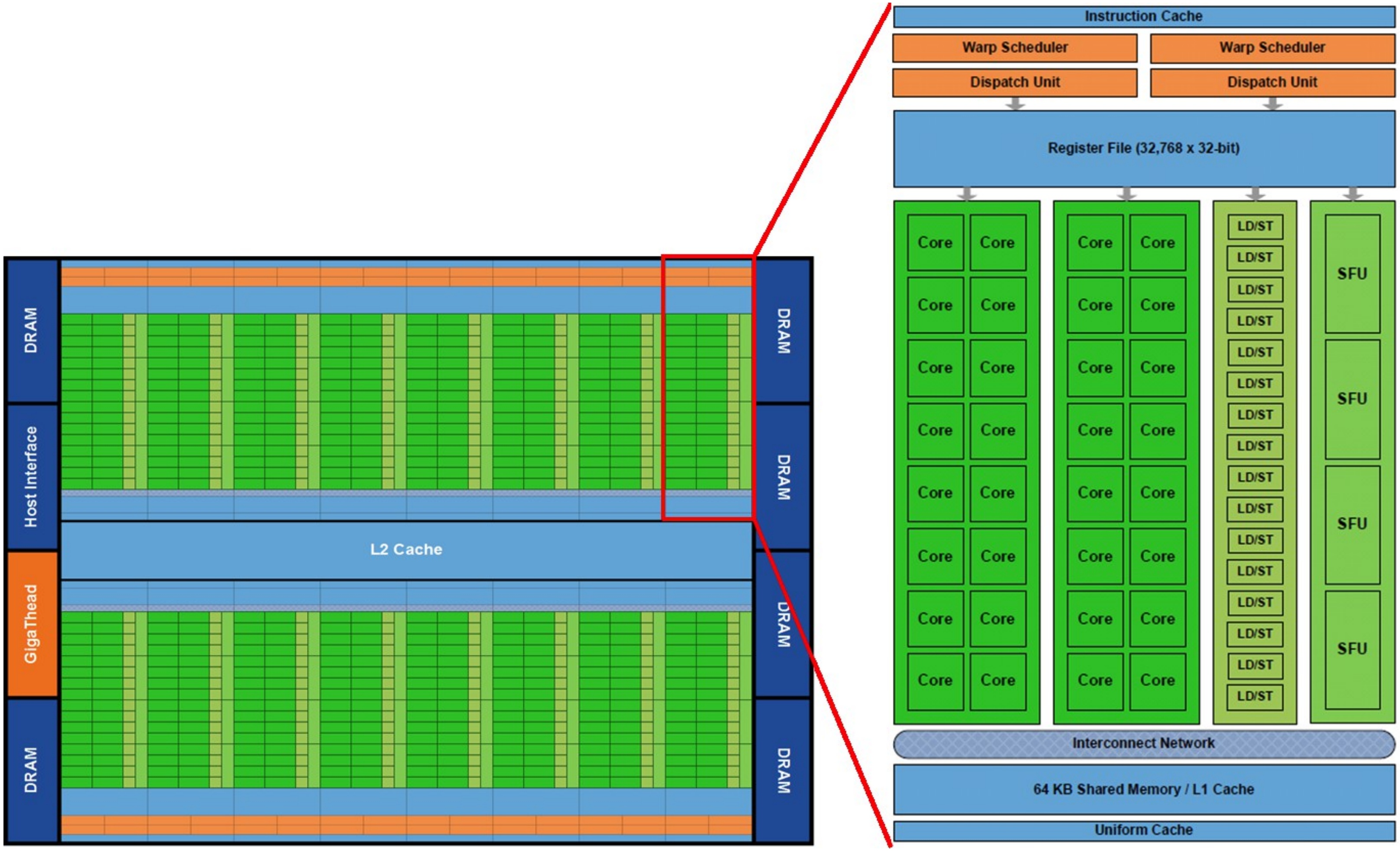
CPU (latency oriented design):

- Large caches
- Sophisticated control
- Powerful ALU

GPU (throughput oriented design):

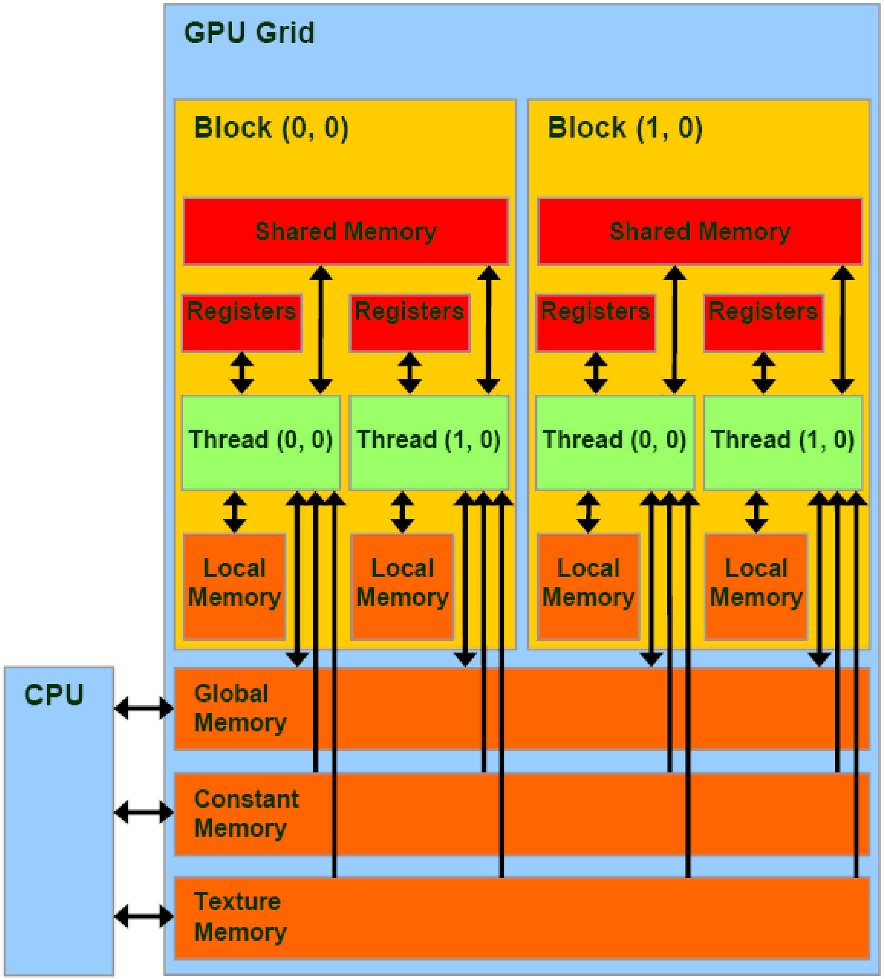
- Small caches
- Simple control
- Energy efficient ALUs
- Latencies compensated by large number of threads

CUDA: Streaming multiprocessor (SMP)

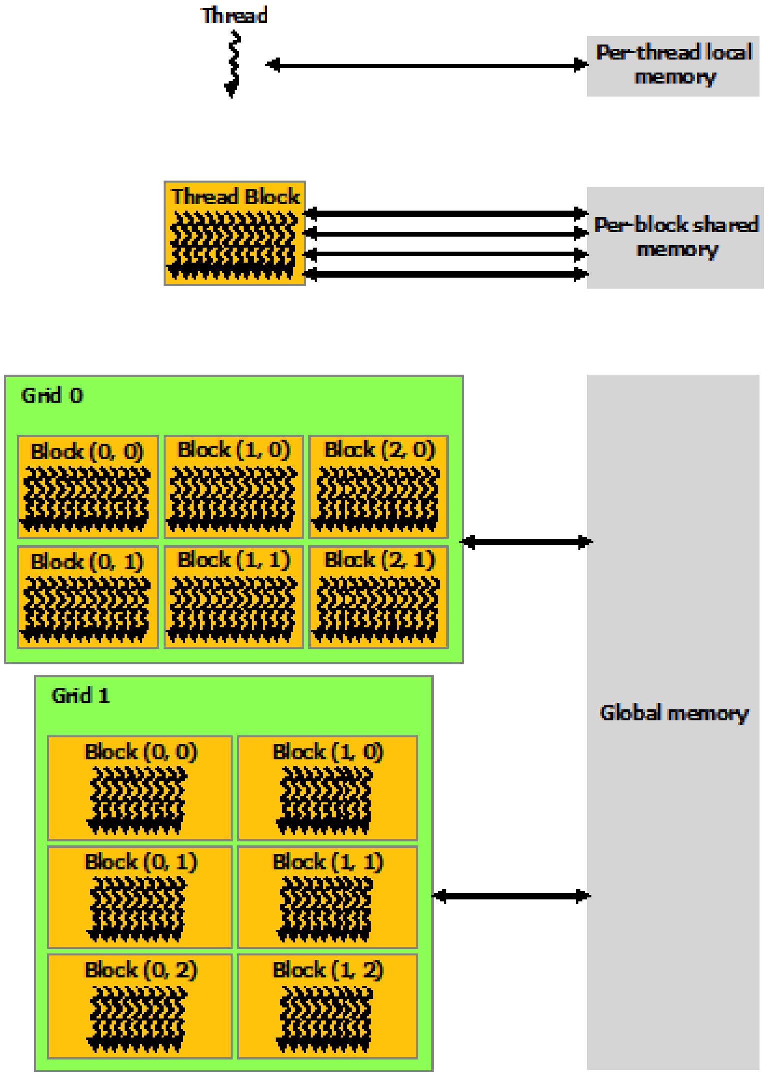


CUDA: Memory hierarchy(1)

- Local memory & registers
  - Small
  - Accessed by one core/thread
  - Low latency (~1 cycle)
- Shared memory
  - Not large (16 KB)
  - Low latency (~5 cycles)
  - Shared between cores/threads within a thread block
- Global memory
  - Large (256mb+)
  - High bandwidth (100 GB/s)
  - High latency (~500 cycles)
- Constant memory
  - Read only, low latency, shared by all threads



CUDA: Memory hierarchy(2)

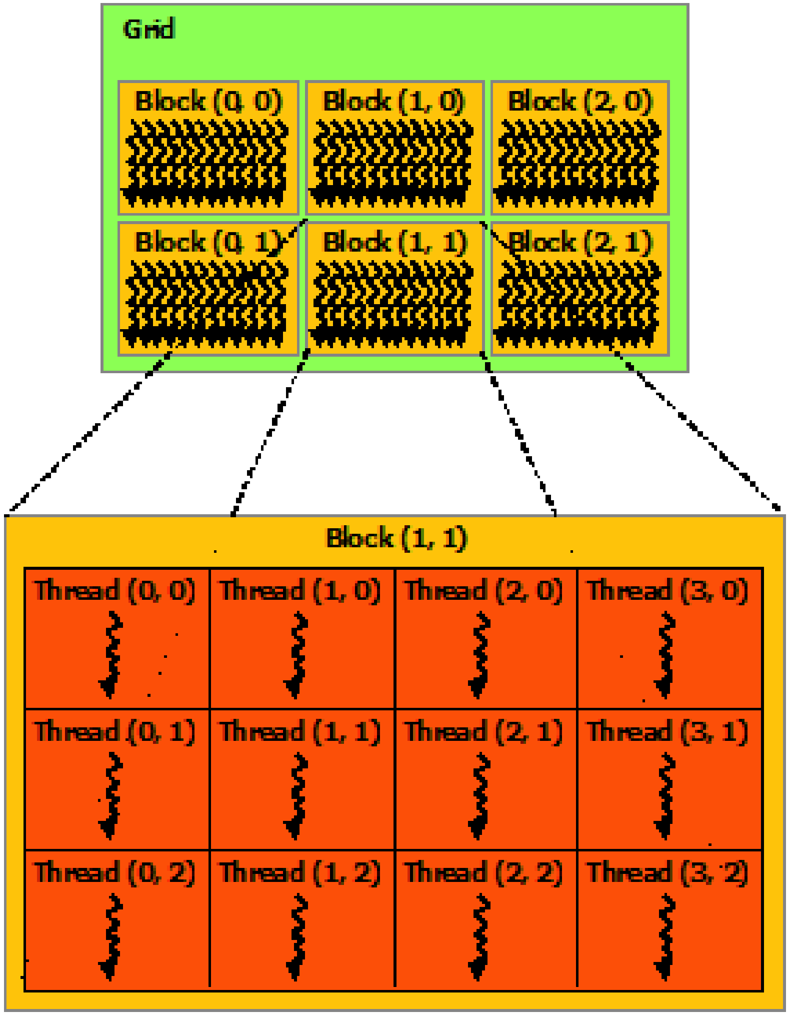


CUDA: Scalable Programming Model (1)

- Three key abstractions:
- Hierarchy of thread groups
  - Grid, thread blocks, warps, threads
- Shared memories
  - Global, shared, local, registers
- Barrier synchronization

CUDA: Scalable Programming Model (2)

- Grid
  - 3D array of thread blocks
- Thread blocks
  - 3D array of threads
  - Up to 1024 threads
- Thread warp
  - Consists of 32 threads which share a control unit.



Grid

Block (0, 0) Block (1, 0) Block (2, 0)

Block (0, 1) Block (1, 1) Block (2, 1)

Block (1, 1)

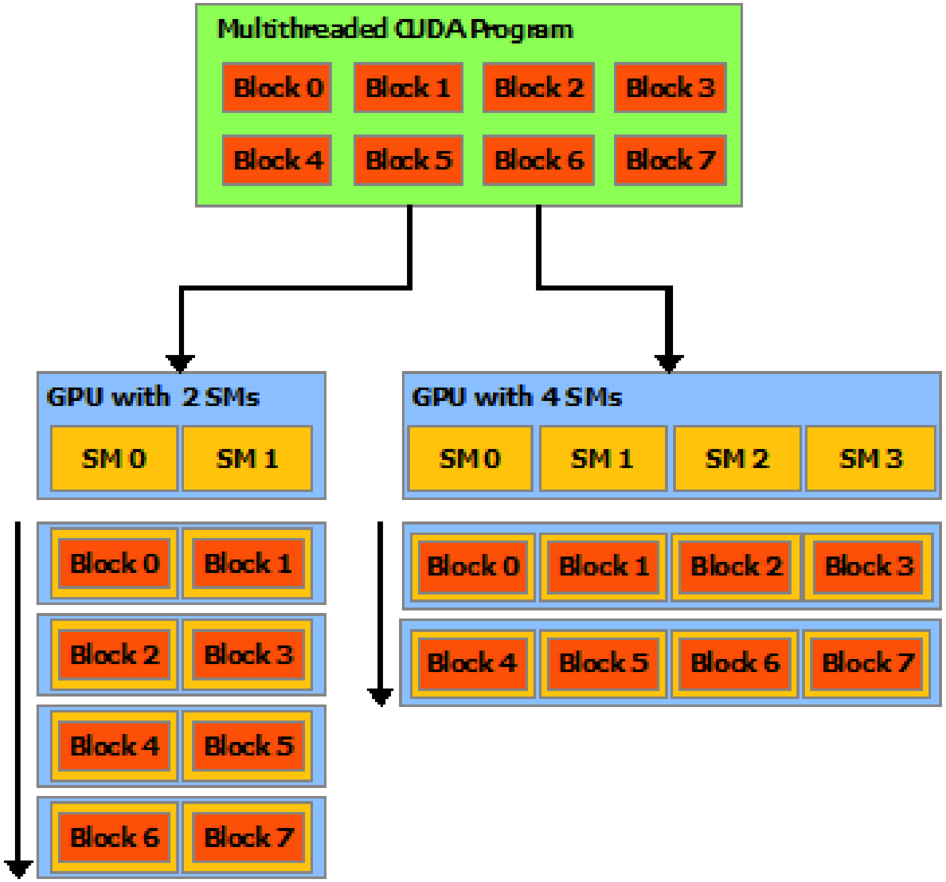
Thread (0, 0) Thread (1, 0) Thread (2, 0) Thread (3, 0)

Thread (0, 1) Thread (1, 1) Thread (2, 1) Thread (3, 1)

Thread (0, 2) Thread (1, 2) Thread (2, 2) Thread (3, 2)

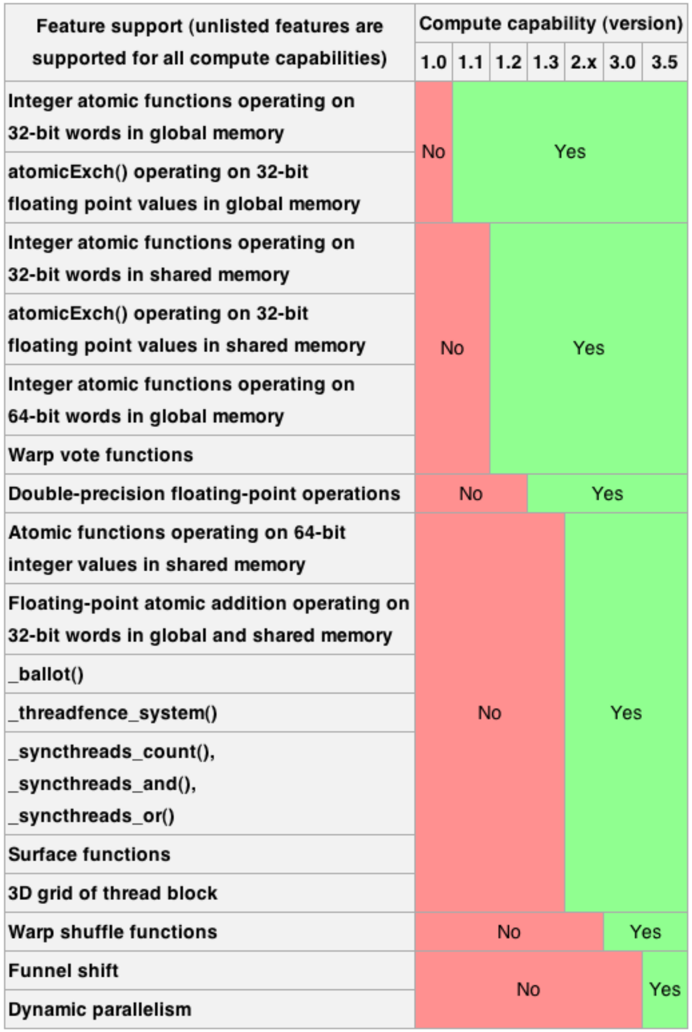
CUDA: Scalable Programming Model (4)

- Automatic program scalability
  - – Across cards of various sizes
  - – Across new core architectures
    - Subject to compute capabilities
- SMP is a basic unit of hardware components each GPU has.
- Better GPUs have more SMPs.
- Compute capabilities are backward compatible, so that older code can run on newer higher capability hardware.



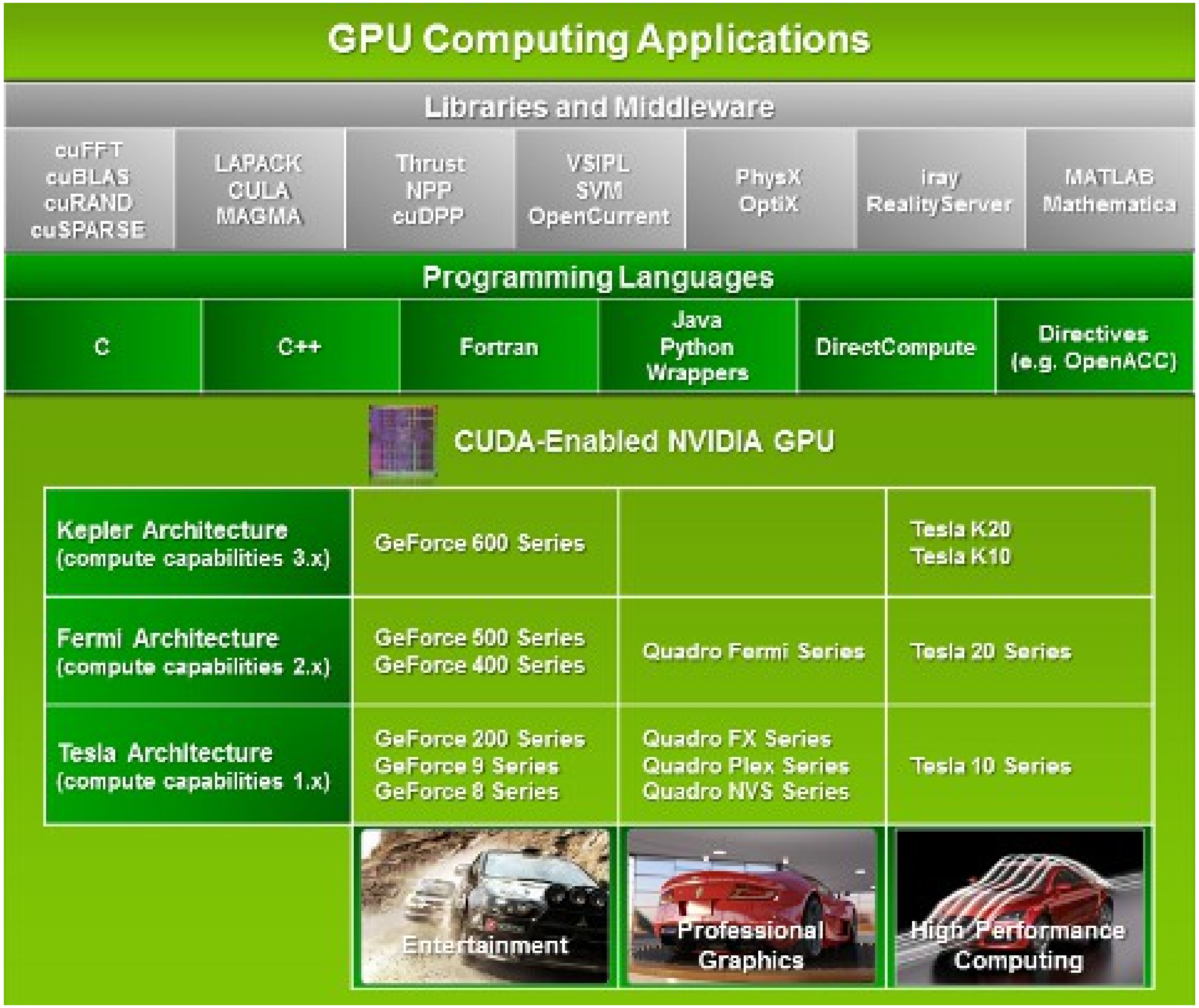
CUDA: Scalable Programming Model (5)

- Compute capabilities specify which features hardware can support.



Feature support (unlisted features are supported for all compute capabilities)	Compute capability (version)						
	1.0	1.1	1.2	1.3	2.x	3.0	3.5
Integer atomic functions operating on 32-bit words in global memory	No	Yes					
atomicExch() operating on 32-bit floating point values in global memory							
Integer atomic functions operating on 32-bit words in shared memory	No		Yes				
atomicExch() operating on 32-bit floating point values in shared memory							
Integer atomic functions operating on 64-bit words in global memory							
Warp vote functions							
Double-precision floating-point operations	No			Yes			
Atomic functions operating on 64-bit integer values in shared memory	No				Yes		
Floating-point atomic addition operating on 32-bit words in global and shared memory							
_ballot()							
_threadfence_system()							
_syncthreads_count(), _syncthreads_and(), _syncthreads_or()							
Surface functions							
3D grid of thread block							
Warp shuffle functions	No					Yes	
Funnel shift	No						Yes
Dynamic parallelism							

CUDA: Languages and APIs



GPU Computing Applications

Libraries and Middleware

cuFFT cuBLAS cuRAND cuSPARSE	LAPACK CULA MAGMA	Thrust NPP cuDPP	VSIPL SVM OpenCurrent	PhysX OptiX	iray RealityServer	MATLAB Mathematica

Programming Languages

C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)

CUDA-Enabled NVIDIA GPU

Kepler Architecture (compute capabilities 3.x)	GeForce 600 Series		Tesla K20 Tesla K10
Fermi Architecture (compute capabilities 2.x)	GeForce 500 Series GeForce 400 Series	Quadro Fermi Series	Tesla 20 Series
Tesla Architecture (compute capabilities 1.x)	GeForce 200 Series GeForce 9 Series GeForce 8 Series	Quadro FX Series Quadro Plex Series Quadro NVS Series	Tesla 10 Series
	Entertainment	Professional Graphics	High Performance Computing

Prerequisites

- CUDA Toolkit and developer driver
  - http://www.nvidia.com/getcuda
- CUDA capable hardware
  - http://www.nvidia.com/object/cuda_gpus.htm
- To test if the CUDA Toolkit is correctly installed:

```
$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2012 NVIDIA Corporation
Built on Fri_Sep_28_16:10:16_PDT_2012
Cuda compilation tools, release 5.0, V0.2.1221
```

References

Many graphics and materials in this presentation are borrowed from the following sources:

- NVidia CUDA Tookit documentation
  - http://docs.nvidia.com/cuda/index.html
- Slides by prof. Wen-mei W. Hwu, of University of Illinois at Urbana-Chapaign, from his online course in Heterogeneous Parallel programming at Coursera
  - https://www.coursera.org/course/hetero
- Stanford CS193G course material
  - http://see.stanford.edu/see/courses.aspx