GPU Computing Workshop

CSU 2013

Background and motivation

Garland Durham
Quantos Analytics

—

Outline

(1) Background and motivation

(2) Configuring and using an Amazon EC2 GPU server

(3) Getting started with CUDA

(4) Advanced topics

(5) Higher level APIs and applications

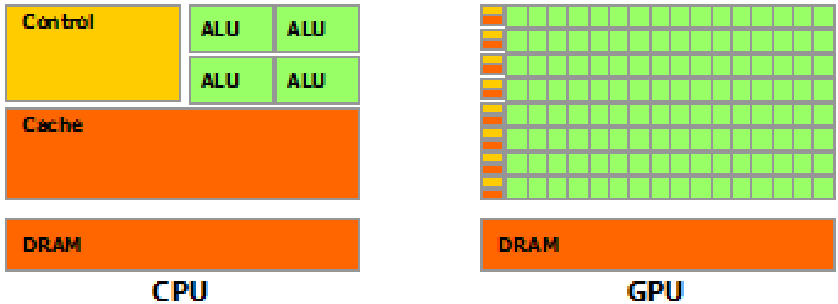
—

Background and motivation

What is GPU computing and why should we be interested in it?

- GPU can be thought of as a "math coprocessor", optimized for performance on specialized computations
  - *motivated* by the needs of graphics applications
  - but, also *useful* for general computations...
- CPU's are optimized for **serial** computations
  - hardware implications (lots of silicone devoted to cache and control logic ...)
  - inherent limits to this approach (diminishing returns...)
  - therefore moving toward mainstream use of multicore CPUs (phones,...)
- GPUs are optimized for **total throughput** on **highly parallel** calculations
  - a typical GPU may have 100s, or even 1000s of cores (much higher density of computational units)

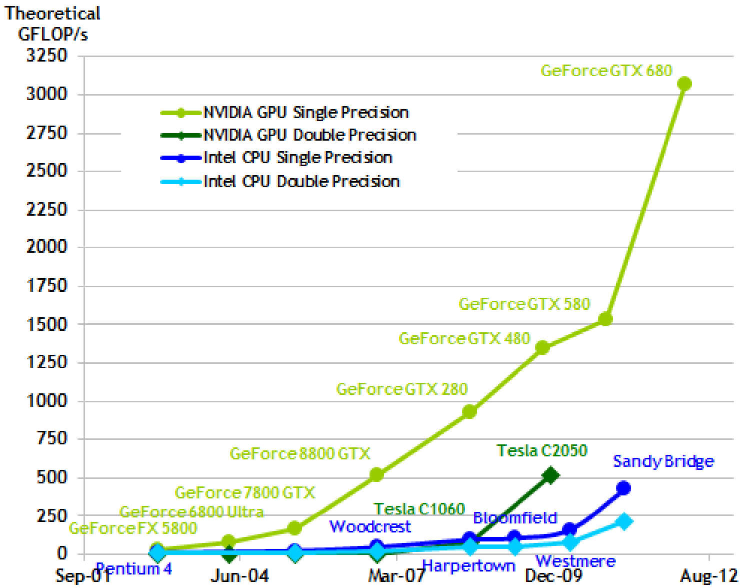
Figure — The GPU devotes more transistors to data processing



Source: Nvidia C Programming Guide

—

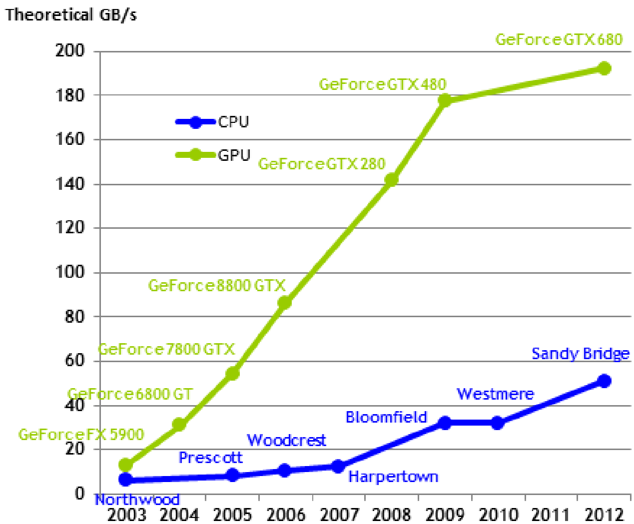
Figure — Floating-Point Operations per Second for the CPU and GPU



Source: Nvidia C Programming Guide

—

Figure — Memory Bandwidth for the CPU and GPU



Source: Nvidia C Programming Guide

—

Discussion

- If you want performance now or in the future, you have to go parallel
  - CPUs (several cores)
  - Clusters (network latency)
  - GPU's (thousands of cores)
- How much faster?
  - From an academic perspective... (10 years ahead of the curve...)
- Research implications
  - new applications
  - new algorithms
- Alternative hardware
  - FPGA (field programmable gate arrays)
  - Playstation (cell)
  - Intel
  - ???
- Supercomputing

—

Discussion — continued

- Why not just develop hardware specialized for the needs of scientific computing?

- Amdahl's Law

—

Hardware

Understanding the hardware is critical to making efficient use of it.

- For future reference, we refer to
  - host: CPU and system memory
  - device: GPU and GPU memory
- A GPU is comprised of
  - global memory (accessible from host and all GPU cores)
  - SMs (stream multiprocessors)
- Each SM has
  - registers
  - thread specific memory
  - "shared" memory (shared between threads)
  - processing units
- SIMT architecture
  - Each SM operates on groups of threads in SIMT fashion.
  - Different SM's can work independently of each other.

—

Hardware — continued

- A function run on the device is referred to as a **kernel** and operates on **blocks** of threads.

- Each thread block executes on a single SM and can access a common block of **shared memory** (threads also have private memory).

- Communication and synchronization
  - Threads in the same block can communicate using **shared memory**.
  - Threads in different blocks can only communicate via **global memory**.
  - In either case, **synchronization barriers** must be used (e.g., to ensure that all threads have written data before reads are attempted by different threads).
  - Synchronization is always **costly**. But especially when it involves synchronizing across SMs.

- Memory on the SM itself is **nonpersistent** (memory associated with a thread block is no longer accessible after a kernel is finished executing).

- Memory hierarchy:
  - Reads/writes to memory on the SM (including shared memory) are fast.
  - Reads/writes to global memory are costly.
  - Transfers between host and device are yet more costly.

—

Hardware — continued

- An SM can have several blocks resident at one time, subject to the following constraints:
  - 1024 threads maximum
  - resource constraints
    * each thread has resource requirements(registers, memory, etc)
    * resources are very limited
  - a single block is never split across SMs
- The collection of blocks is referred to as the "**grid**". The number of blocks (gridsize) is user determined.
  - There can be more blocks than it is possible to have resident on the available SMs at one time.

—

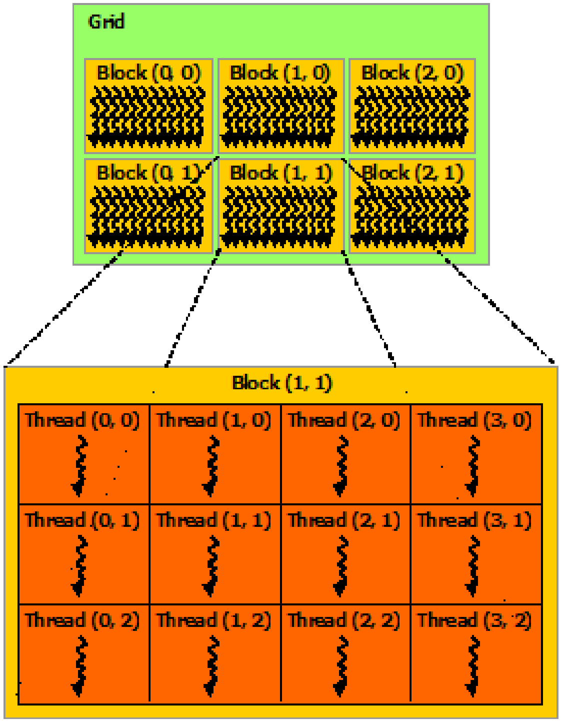
Hardware — continued

- In practice, the SM operates on groups of threads ("**warps**"*) in parallel
  - warp size is hardware determined (32 threads on recent hardware)
  - A block can consist of several warps (blocksize is user determined).
  - Processing switches from one warp to the other whenever the execution is waiting on data.
- Having lots of threads resident on the SM is good (latency hiding)
  - referred to as **occupancy**.
  - implications for block size and software design (how much work is done by a single thread and how work is organized)
- **From the user perspective, this is all transparent**
  - the user selects the grid size and block size
  - the compiler takes care of all scheduling.

* In recognition of weaving, the first massively parallel threaded technology (Nvidia User's Guide).

—

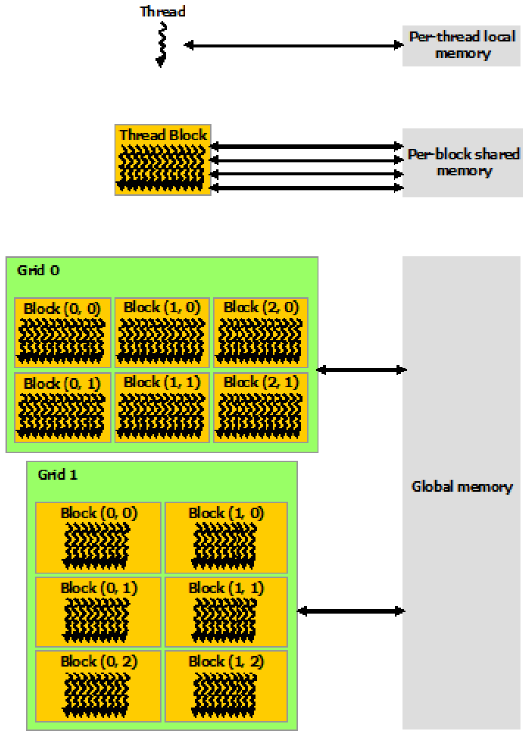
Figure — grid of thread blocks



Source: Nvidia C Programming Guide

—

Figure — memory hierarchy



Source: Nvidia C Programming Guide

—

Links

- https://developer.nvidia.com/category/zone/cuda-zone
- https://developer.nvidia.com/cuda-downloads

—

Nvidia GPUs

For details on all available GPUs see `https://developer.nvidia.com/cuda-gpus`.

- Important specs are
  - Number of cores
  - Amount of memory
  - "Compute capability" (see Users Guide for details)

—

Nvidia GPUs — continued

```
Tesla (scientific computing)
  *** Tesla K20                  (about \$3000)
      Tesla K10                  (optimized for single precision)
      Tesla C2050 C2070 C2075    (previous generation)


Quadro
==========


GTX (consumer cards)
=============
   **Titan                   (about \$1000)
         *780 *770 *760      (double precision crippled; \$650 - 400 - 250)
     690  680  670            GTX 690 is basically 2 580's on one card
     590  580  570            GTX 590 is basically 2 580's on one card

      The 500 series is better than 600 and 700 series for double precision.
      Double precision speed as fraction of single precision:
         Titan:               1/3
         600 and 700 series:  1/24
         500 series:          1/8


Mobile
==============


anandtech.com is a good source of information and benchmarks
```

—

Remarks

- The Amazon EC2 GPU servers have 2 Tesla C2050 cards.

- If you are building your own GPU server, the more powerful cards draw lots of power. Get a BIG power supply!

—

Documentation

Installed at /usr/local/cuda/doc/pdf. Also available online.

```
 ** CUDA_C_Best_Practices_Guide.pdf
  * CUDA_Compiler_Driver_NVCC.pdf
*** CUDA_C_Programming_Guide.pdf
  * CUDA_CUBLAS_Users_Guide.pdf
    CUDA_CUFFT_Users_Guide.pdf
    CUDA_CUSPARSE_Users_Guide.pdf
    CUDA_Debugger_API.pdf
    CUDA_Developer_Guide_for_Optimus_Platforms.pdf
    CUDA_Dynamic_Parallelism_Programming_Guide.pdf
    CUDA_GDB.pdf
  * CUDA_Getting_Started_Guide_For_Linux.pdf
    CUDA_Getting_Started_Guide_For_Mac_OS_X.pdf
    CUDA_Getting_Started_Guide_For_Microsoft_Windows.pdf
    CUDA_Memcheck.pdf
    CUDA_Profiler_Users_Guide.pdf
    CUDA_Samples_Guide_To_New_Features.pdf
  * CUDA_Samples.pdf
    CUDA_Samples_Release_Notes.pdf
  * CUDA_Toolkit_Reference_Manual.pdf
    CUDA_Toolkit_Release_Notes.pdf
    CUDA_VideoDecoder_Library.pdf
    cuobjdump.pdf
    CUPTI_User_Guide.pdf
  * CURAND_Library.pdf
    Floating_Point_on_NVIDIA_GPU_White_Paper.pdf
  * Getting_Started_With_CUDA_Samples.pdf
    GPUDirect_RDMA.pdf
    Kepler_Compatibility_Guide.pdf
    Kepler_Tuning_Guide.pdf
    NPP_Library.pdf
    Nsight_Eclipse_Edition_Getting_Started.pdf
    Preconditioned_Iterative_Methods_White_Paper.pdf
    ptx_isa_3.1.pdf
    qwcode.highlight.css
  * Thrust_Quick_Start_Guide.pdf
    Using_Inline_PTX_Assembly_In_CUDA.pdf
```

—

Samples

Installed at `/usr/local/cuda/samples`. Also available online.

`0_Simple`

asyncAPI	cdpSimplePrint	cdpSimpleQuicksort	clock
cppIntegration	cudaOpenMP inlinePTX	matrixMul	matrixMulCUBLAS
matrixMulDrv	matrixMulDynlinkJIT	simpleAssert	simpleAtomicIntrinsics
simpleCallback	simpleCubemapTexture	simpleIPC	simpleLayeredTexture
simpleMPI	simpleMultiCopy	simpleMultiGPU	simpleP2P
simplePitchLinearTexture	simplePrintf	simpleSeparateCompilation	simpleStreams
simpleSurfaceWrite	simpleTemplates	simpleTexture	simpleTextureDrv
simpleVoteIntrinsics	simpleZeroCopy	template	template_runtime
vectorAdd	vectorAddDrv		

`1_Utilities`

`bandwidthTest  deviceQuery  deviceQueryDrv`

`2_Graphics`

`bindlessTexture  Mandelbrot  marchingCubes  simpleGL  simpleTexture3D  volumeFiltering volumeRender`

`3_Imaging`

bicubicTexture	bilateralFilter	boxFilter	convolutionFFT2D
convolutionSeparable	convolutionTexture	dct8x8	dwtHaar1D
dxtc	histogram	HSOpticalFlow	imageDenoising
postProcessGL	recursiveGaussian	SobelFilter	stereoDisparity

`4_Finance`

`binomialOptions  BlackScholes  MonteCarloMultiGPU  quasirandomGenerator  SobolQRNG`

—

Samples — continued

5_Simulations

fluidsGL nbody oceanFFT particles smokeParticles

6_Advanced

alignedTypes	cdpAdvancedQuicksort	cdpLUDecomposition	cdpQuadtree
concurrentKernels	eigenvalues	fastWalshTransform	FDTD3d
FunctionPointers	interval	lineOfSight	mergeSort
newdelete	ptxjit	radixSortThrust	reduction
scalarProd	scan	segmentationTreeThrust	shfl_scan
simpleHyperQ	sortingNetworks	threadFenceReduction	threadMigration
transpose			

7_CUDALibraries

batchCUBLAS	boxFilterNPP	common	conjugateGradient
conjugateGradientPrecond	freeImageInteropNPP	grabcutNPP	histEqualizationNPP
imageSegmentationNPP	MC_EstimatePiInlineP	MC_EstimatePiInlineQ	MC_EstimatePiP
MC_EstimatePiQ	MC_SingleAsianOptionP	MersenneTwisterGP11213	randomFog
simpleCUBLAS	simpleCUFFT	simpleDevLibCUBLAS	

—

Other resources

- “Cuda By Example” (an excellent book for learning about CUDA at the introductory level)

- “CUDA Application Design and Development”

- “Programming Massively Parallel Processors: A Hands On Approach”

- ”GPU Gems,” Volumes 1-3 (These are available online at Nvidia.com).
  - Volume 1 is mostly outdated.
  - Volume 2, see Part IV: “General-Purpose Computation on GPUS: A Primer” (chapters 29-36)
  - Volume 3, Part VI: ”GPU Computing”, especially chapter 37 (random number generation) and chapter 39 (prefix scan).

See workshop home page for more links.

—