NVIDIA’S VISION FOR EXASCALE
Cyril Zeller, Director, Developer Technology
EXASCALE COMPUTING

An industry target of 1 ExaFlops within 20 MW by 2020

- 1 ExaFlops: a necessity to advance science and technology
- Within 20 MW: to mitigate cost of ownership and power delivery infrastructure
- By 2020: based on extrapolation of historical trend
THE PATH TO EXASCALE

An energy efficiency challenge

2013

20 PFlops
10 MW
=
2 GFlops/W

2020
Exascale

50 GFlops/W

= 1000 PFlops (50x)
20 MW (2x)

25x
THE PATH TO EXASCALE

Process will only get us so far

2013
2 GFlops/W

2020
Exascale
50 GFlops/W

25x

2-5x from process
(28 nm to 7 nm)
THE PATH TO EXASCALE

We must harness energy-efficient architectures

2013

2 GFllops/W

2020

Exascale

50 GFlops/W

5-12x from architecture and circuit

2-5x from process (28 nm to 7 nm)

25x
GPU = AN ENERGY-EFFICIENT ARCHITECTURE

**CPU**

- Optimized for latency
  (fast processing of single thread)
- through speculative, out-of-order execution and large caches
- adding a lot of energy-hungry overheads per instruction
- \(~400 \text{ pJ/flop}\)
- Intel Ivy Bridge

**GPU**

- Optimized for throughput
  (fast processing of many threads in parallel)
- Latency hidden through fine-grain multithreading
- Low overhead per instruction
  (no need for speculative, out-of-order execution and large caches)
- \(~100 \text{ pJ/flop}\)
- NVIDIA Kepler
NVIDIA KEEPS IMPROVING ENERGY EFFICIENCY

flops/W speedups relative to Tesla’s flops/W for SGEMM

Fermi
Kepler
Maxwell
Pascal

WHERE IS THE ENERGY GOING?

Data movement costs more energy than data processing

Large disparity of cost between the various memory levels

Data movement energy ratios:

- **On-chip:** \( \frac{\text{Remote SRAM}}{\text{Local SRAM}} = 20x \)
- **Off-chip:** \( \frac{\text{Local DRAM}}{\text{Local SRAM}} = 320x \)
- **Off-node:** \( \frac{\text{Remote DRAM}}{\text{Local DRAM}} = 100x \)
STRATEGY FOR ENERGY REDUCTION

Improve physical locality to minimize data movement cost

- First step: stacked memory in 2016 (Pascal architecture)
  - DRAM integrated into same package as processor
  - Much shorter interconnect between DRAM and processor
  - Much wider interfaces to memory for increased DRAM bandwidth

- Pascal’s stacked DRAM vs Kepler’s off-package DRAM
  - 4x more energy efficient per bit
  - 4x higher bandwidth (~1 TB/s on Pascal)
  - 3x larger capacity
RESEARCH AREAS FOR ENERGY REDUCTION

Improve physical locality to minimize data movement cost

- Hierarchical register file
  - To exploit register reuse by caching registers in a small *active register file* that is closer to the data processing units

- Malleable memory
  - Configurable on-chip memory to allow for optimal matching to application needs (register vs cache, hardware-managed vs software-managed cache, shared vs private, configurable sizes)
RESEARCH AREAS FOR ENERGY REDUCTION

Optimize Single-Instruction Multiple-Threads execution model

- SIMT execution model = single instruction fetched, decoded, and scheduled for a group of threads (aka warp)
  - Spatial implementation (Kepler): threads execute in parallel across multiple lanes
    - Caveat: idle lanes when warp diverges (i.e., threads take different execution paths)
  - Temporal implementation: threads execute sequentially within a single lane
    - Benefits: no idle lane when warp diverges, scalarization (i.e., execute instruction only once for threads with same source and destination operands)
    - Caveat: does not amortize cost of fetch/decode/schedule as well
- Explore trade-off between spatial and temporal implementations
RESEARCH AREAS FOR ENERGY REDUCTION

Use energy-efficient circuit technologies

- Low-voltage on-chip memory
- Low-swing circuits for on-chip signaling
- Ground-referenced signaling for on-package and off-package communication
EXASCALE NODE

- Heterogenous
  - Still need a few cores optimized for fast serial work
- Three building blocks (GPU, CPU, NIC) that can be:
  - Multiple sockets
  - Or a Multiple Chip Module
  - Or one chip

- Throughput-Optimized Cores for parallel code
- Latency-Optimized Cores for serial code (e.g., OS)
- Built for 100K nodes

System Interconnect
ALL CPU PLATFORMS SUPPORTED TODAY

NVIDIA GPU

ARM64

x86

POWER
NVLink in 2016

Enables CPU↔GPU data transfer at speed of CPU memory

- High speed interconnect at 80 to 200 GB/s
- Planned support for POWER CPUs

Diagram:
- GPU
  - NVLink ~100 GB/s
  - HBM ~1 TB/s
- CPU
  - DDR4 ~100 GB/s
- Stacked Memory
- DDR Memory
EXASCALE SYSTEM

A resilience challenge

- 50000+ nodes with petabytes of memory capacity
  - More susceptible to intermittent failures due to large number of components
  - Compounded by circuits with narrower voltage/timing for power efficiency
- Areas of research for increasing mean time to failure
  - Circuits that are more tolerant to variations in voltage, temperature, etc.
  - More vigilant hardware error detection recovery
  - Application-level fault tolerance (distributed and programmer-guided checkpointing, error containment domains)
THE FUTURE OF HPC PROGRAMMING

- Massively parallel
  - Structure applications as throughput problems
  - Expose all parallelism
  - By exascale timeframe: $10^5$ threads per GPU, $10^9$ threads per system

- Hierarchical
  - Manage memory placement across the memory hierarchy to expose locality

- Heterogeneous
  - Manage separate memory spaces and concurrent execution of parallel and serial codes
THE PATH TO EXASCALE

A programming challenge

- Design a development platform that provides:
  - Portability: allowing programmers to express all available parallelism and locality independent of targeted hardware
  - Control: for maximum performance
  - Productivity: simple programming model, libraries, tools

- Find new parallel approaches/algorithms and/or refactor codes
  - E.g., for existing applications that do not expose enough parallelism
CUDA PROGRAMMING MODEL FOR GPUS

Parallel, hierarchical, heterogeneous

Host (CPU)

Device (GPU)

Grid of Blocks

Block 0

Shared Memory

Block 1

Shared Memory

Block N

Shared Memory

Host Memory

Device Memory
CUDA PROGRAMMING MODEL FOR GPUS

Implemented as simple extensions to mainstream languages (C, C++, Fortran, Python, and others)

```c
void sortfile(FILE *fp, int N) {
    char *data = (char*)malloc(N);
    char *sorted = (char*)malloc(N);
    fread(data, 1, N, fp);

    char *d_data, *d_sorted;
    cudaMalloc(&d_data, N);
    cudaMalloc(&d_sorted, N);
    cudaMemcpy(d_data, data, N, ...);

    parallel_sort<<< ... >>>(d_sorted, d_data, N);

    cudaMemcpy(sorted, d_sorted, N, ...);
    cudaFree(d_data);
    cudaFree(d_sorted);

    use_data(sorted);
    free(data); free(sorted);
}
```
CUDA DEVELOPMENT PLATFORM

Productive tools and higher-level programming approaches

Programming Approaches
- Libraries: “Drop-in” acceleration
- OpenACC Directives: Maximum portability
- Programming Languages: Maximum control

Development Environment
- Parallel Nsight IDE
  - Linux, Mac and Windows
  - GPU Debugging and Profiling
- CUDA-GDB debugger
- NVIDIA Visual Profiler

Open Compiler Tool Chain
- Enables compiling new languages to CUDA platform, and CUDA languages to other architectures
GPU-ACCELERATED LIBRARIES

“Drop-in” acceleration

NVIDIA cuBLAS
NVIDIA cuSPARSE
NVIDIA NPP
NVIDIA cuFFT

MAGMA
Matrix Algebra on GPU and Multicore

GPU Accelerated Linear Algebra

Vector Signal Image Processing

NVIDIA cuRAND

IMSL Library
CenterSpace NMath

ArrayFire
Building-block Algorithms

C++ Templated Parallel Algorithms
OPENACC

Open standard for GPU compiler directives
Simple and portable

main() {
    ...
    <serial code>
    ...
    #pragma acc kernels
    {
        <compute intensive code>
    }
    ...
}
FUTURE IMPROVEMENTS TO DEVELOPMENT PLATFORM

- Hardware support for CPU-GPU unified memory space
- Programming model research
- Standardization
Explicit CPU↔GPU memory copies are no longer required

Hardware support in Pascal (2016)

```c
void sortfile(FILE *fp, int N) {
    char *data = (char*)malloc(N);
    char *sorted = (char*)malloc(N);
    fread(data, 1, N, fp);

    char *d_data, *d_sorted;
    cudaMalloc(&d_data, N);
    cudaMalloc(&d_sorted, N);
    cudaMemcpy(d_data, data, N, ...);

    parallel_sort<<< ... >>>(d_sorted, d_data, N);
    cudaMemcpy(sorted, d_sorted, N, ...);
    cudaFree(d_data);
    cudaFree(d_sorted);

    use_data(sorted);
    free(data); free(sorted);
}
```

```c
void sortfile(FILE *fp, int N) {
    char *data = (char*)malloc(N);
    char *sorted = (char*)malloc(N);
    fread(data, 1, N, fp);

    parallel_sort<<< ... >>>( sorted, data, N);
    cudaDeviceSynchronize();

    use_data(sorted);
    free(data); free(sorted);
}
```
PROGRAMMING MODEL RESEARCH

How should mainstream languages evolve to embrace the trajectory of hardware?

- Unified model for heterogeneous parallel machines
  - Single notation for all processors
  - Language extensions for platform-specific code generation
- CUB: highly tuned low-level collective operations (reduce, scan, sort, etc.)
- Portable collection-oriented framework built on top of CUB
- Framework for automatically changing the layout of data structures
STANDARDIZATION EFFORTS
A standard C++ parallel library

std::vector<int> vec = ... 

// previous standard sequential loop
std::for_each(vec.begin(), vec.end(), f);

// explicitly sequential loop
std::for_each(std::seq, vec.begin(), vec.end(), f);

// permitting parallel execution
std::for_each(std::par, vec.begin(), vec.end(), f);

- Complete set of parallel primitives: for_each, sort, reduce, scan, etc.
- ISO C++ committee voted unanimously to accept as official technical specification working draft

Prototype: https://github.com/n3554/n3554
Incorporating OpenACC into GCC is an excellent example of open source and open standards working together to make accelerated computing broadly accessible to all Linux developers.

Oscar Hernandez
Oak Ridge National Laboratories
SPREAD PARALLEL PROGRAMMING AROUND THE WORLD

Foster research in parallel algorithms

- 14,000 Institutions with CUDA developers
- 2,000,000 CUDA downloads
- 487,000,000 CUDA GPUs shipped
- 700+ university courses in 62 countries
NVIDIA IN EUROPE

731 staff
580 engaged in engineering R&D
(USA: 4,478 total, 3,070 R&D)

12 offices and 3 HPC labs
(DE, FI, FR, IT, SE, UK, CH)

HPC applications and tools,
bioinformatics, scalable
visualization, mobile platform,
professional graphics

€370 million acquisition and
investment
Icera, Mental Images, Hybrid
NVIDIA IN EUROPE

EXAMPLE CUSTOMERS

Automotive: Audi, JLR, PSA
Medical: Siemens, GE Healthcare, Philips
Oil and Gas: ENI, Total, Schlumberger
HPC: CEA, CSCS, Max Planck Society
Finance: Deutsche Bank, UniCredit, BNP
NVIDIA IN EUROPE

SAMPLE PROJECTS

Square Kilometer Array Telescope
Human Brain Project
Member of ETP 4 HPC
NVIDIA IN EUROPE

CUDA ACADEMIC PROGRAM

4 CUDA Centers of Excellence
- University of Cambridge
- University of Oxford
- Technical University of Dresden
- Barcelona Supercomputing Center

45 CUDA Research Centers

42 CUDA Teaching Centers

3 CUDA Fellows

USA: 9 CCoE, 34 CRC, 70 CTC, 6 CUDA Fellows
NVIDIA IS LEADING THE PATH TO EXASCALE

- Leveraging 20 years of R&D in parallel computing and 1$B/year of investment in core GPU technology
  - Sustainable business model: HPC is an incremental investment
  - GPU = the most efficient HPC processor today with a large installed base
- Fully engaged to meet the energy efficiency, resilience, and programming challenges of exascale
  - Building the HPC ecosystem jointly with education, tool, system, and CPU providers
  - Partnering with developers to accelerate HPC applications
  - Advancing processor architecture and circuit technologies, programming models, and software development platforms through a robust HPC R&D program