The Present and Future of Parallelism on GPUs

Paulius Micikevicius
NVIDIA

Terminology

- **We'll define a "brick" for this talk**
  - Since all vendors use different names
  - GPUs are built by replicating bricks (10s)
    - Connected to memory via some network
- **Brick = minimal building block that contains own:**
  - Control unit(s) - decodes/issues instructions
  - Registers
  - Pipelines for instruction execution
  - Local cache(s)

HW Commonalities
Likely to remain in the future

- **Built by replicating 10s of "bricks"**
- **"Bricks" are vector processors**
  - Different execution paths within vectors are supported but degrade performance
  - Different execution paths in different vectors have no impact on performance
- **Vectors access memory in cache-lines**
  - Consecutive threads (vector elements) should access a contiguous memory region
  - Scatter-gather supported, but will fetch multiple lines, increasing latency and bandwidth-pressure
- **High GPU memory bandwidth (100+ GB/s)**
- **In-order instruction issue**

NVIDIA GPU Brick (Streaming Multiprocessor)

- **Up to 1536 threads**
- **Instructions are issued per 32 threads (*warp*)**
  - Think 32-way vector of threads
- **Source code is for a single thread and is scalar:**
  - No vector intrinsics a la SSE
  - HW handles grouping of threads into vectors, vector control flow
- **Dual-issue: instructions from different warps**
- **Shared memory, L1 cache**
- **Large register file, partitioned among threads**

AMD GPU Brick (SIMD Engine)

- **Up to ~1500 threads**
- **Instructions are issued per 64 threads**
- **VLIW instruction issue:**
  - HW designed for 5 “issue” slots (16x5 ‘cores’ per brick)
  - Combine up to 5 instructions from the same thread to maximize performance
- **Source code is for a single thread and is <u>scalar</u>:**
  - HW handles grouping threads into vectors and control flow
  - Compiler handles VLIW combining
- **Shared memory, L1 cache**
- **Large register file, partitioned among threads**

Intel Larrabee Brick (Core)

- **Up to 4 threads**
- **Scalar and vector (512-bit SIMD) units**
  - For example: 16-fp32 vector SIMD
- **Dual issue: scalar-vector, from the same thread**
- **Source code is for a single thread and is vector:**
  - Intrinsics for SIMD operations (a la SSE)
- **L1 and L2 caches**
  - Intrinsics for pre-fetching and prioritization of cache lines
  - No user-managed shared memory
- **Small register file (relies on caches)**

HW Commonalities
Likely to remain in the future

- **Built by replicating 10s of "bricks"**
- **"Bricks" are vector processors**
  - Different execution paths within vectors are supported but degrade performance
  - Different execution paths in different vectors have no impact on performance
- **Vectors access memory in cache-lines**
  - Consecutive threads (vector elements) should access a contiguous memory region
  - Scatter-gather supported, but will fetch multiple lines, increasing latency and bandwidth-pressure
- **High GPU memory bandwidth (100+ GB/s)**
- **In-order instruction issue**

Current Differences

- **Hardware:**
  - Vector width: 16, 32, 64
  - VLIW (AMD) vs dual-issue (NVIDIA, Intel)
  - Dual-issue: same thread (Intel) vs different threads (NVIDIA)
  - Large register file, small cache (NVIDIA,AMD) vs small register file, larger caches (Intel)
- **Programming model**
  - Intel: vector source code (SIMD intrinsics)
  - AMD,NVIDIA: scalar source code
    - hw aggregates threads into vectors and resolves control flow divergence

References

- **NVIDIA:**
  - http://www.nvidia.com/object/fermi_architecture.html
  - Fermi Compute Architecture Whitepaper (http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf)
- **AMD:**
  - Unleashing the Power of Parallel Compujte with Commodity ATI Radeon 5800 GPU, Siggraph Asia 2009 (http://sa09.idav.ucdavis.edu/docs/SA09_AMD_IHV.pdf)
- **Intel:**
  - L. Seiler et al. Larrabee: A Many Core X86 Architecture for Visual Computing. Siggraph 2008 (http://software.intel.com/file/18198/)
  - Tom Forsyth. The Challenge of Larrabee as GPU. Colloquium at Stanford, 2010 (http://www.stanford.edu/class/ee380/Abstracts/100106.html)

Simple 2D Stencil Code in CUDA (OpenCL equivalent in comments)

```
__global__ void stencil_2d( float *output, float *input,
                                    const int dimx, const int dimy, const int row_size )
{
   int ix = blockIdx.x*blockDim.x + threadIdx.x;  // ix = get_global_id(0);
   int iy = blockIdx.y*blockDim.y + threadIdx.y;  // iy = get_global_id(1);
   int idx = iy * row_size + ix;

   output[idx] = -4 * input[idx]
                    + input[idx+1]  + input[idx-1]
                    + input[idx + row_size] + input[idx – row_size];
}
```