To GPU or Not to GPU

Sumit Jamgade
Cloud Developer
sjamgade@suse.com
Interesting Internals of GPU

- Streaming Multiprocessor
- Memory
  - Device
  - Shared
  - L3 Cache
  - Pinned Memory
- Registers
Interesting Internals of GPU...

- Streaming Multiprocessor (SM)
- Smallest units that work synchronously
- Juggler of warps (later…)
- Memory (shared)

- Should be always busy
- Fixed in hardware, so more is better
- Involved in a key metric – Occupancy
Interesting Internals of GPU...

Memory
- Completely software managed
- Shared (Fast)
- More than 25% per year growth
- Research data growth
  - More than 25% per year growth
- Up to the user to populate
- Use this like RAM
  - L1/2/3 Cache (Faster)
- No user-level control
  - Registers (Guess what...yup Fastest)
- Populated by execution unit
- Master it!
- Global Memory (super slow)
- Hard disk
Interesting Internals of GPU!
Annoying Internals of GPU

- `kernel`
- `threadIdx.{x,y,z}`
- `blockIdx.{x,y,z}`
- `gridIdx.{x,y,z}`

RealThreadID = block[x][y][z]

RealBlockID = grid[a][b][c]
A Typical Rxample: 5x5 Matrix Addition

grid = [1,1]
block = [5,5,1]

int i = threadIdx.x
int j = threadIdx.y
C[i][j] = A[i][j] + B[i][j]
Tools

Nvidia Visual Profiler and nvprof
– Search: cuda nvprof handy

Pycuda - https://github.com/inducer/pycuda
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule

CUDA compiler
Real World Example: Gnocchi

Time Series Database

Distributed

Built data Expiry

(Pre) Aggregate
• Function (sum, min, max)
• Resource Classes
• Interval

Computing Library (Carbonara)
__global__ void v1(float *list, int *i) {
  int perthread = i[0];
  int counter = i[0] - 1;
  int col = blockIdx.x * blockDim.x + threadIdx.x;
  int row = blockIdx.y * blockDim.y + threadIdx.y;
  int index = col * perthread + row;
  for(;counter; counter--)
    atomicAdd(list + index, list[index + counter]);
}

Continuous range
  Gpu: 17.53 msec
  Cpu: 56.29 msec

Just zeros
  Gpu: 16.68 msec
  Cpu: 56.09 msec
__global__ void v1(float *list, int *i) {
    int perthread = i[0];
    int counter = i[0] - 1;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int index = col * perthread + row;
    for(;counter;counter--)
        atomicAdd(list+index, list[index+counter]);
}
__global__ void v1(float *list, int *i) {
    int perthread=i[0];
    int counter = i[0]-1;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int index = col * perthread + row;
    for(;counter;counter--)
        atomicAdd(list+index, list[index+counter]);
}

Continuous range
Gpu: 17.53 msec
Cpu: 56.29 msec

Just zeros
Gpu: 16.68 msec
Cpu: 56.09 msec
__global__ void v3(float *a, int *i) {
    __shared__ float inter[6*1024/16];
    for (int blk=1;blk<blklimit;blk++) {
        counter = perthread-1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for(counter=0;counter<perthread;counter++){
            index = col + blockDim.x*counter + row;
            inter[blockDim.x * counter + tid] = a[index];
        }
        __syncthreads();
        counter = perthread-1;
        for(;counter;counter--){
            x += inter[blockDim.x * counter + tid];
        }
        a[index] = x;
        __syncthreads();
    }
}
```c
__global__ void v3(float *a, int *i) {
    __shared__ float inter[6*1024/16];
    for (int blk=1;blk<blklimit;blk++) {
        counter = perthread-1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for(counter=0;counter<perthread;counter++){
            index = col + blockDim.x*counter + row;
            inter[blockDim.x * counter + tid] = a[index];
        }
        __syncthreads();
        counter = perthread-1;
        for(;counter;counter--){
            x += inter[blockDim.x * counter + tid];
        }
        a[index] = x;
        __syncthreads();
    }
}
```

Continuous range
Gpu: 15.50 msec
Cpu: 75.13 msec

Just zeros
Gpu: 15.04 msec
Cpu: 75.82 msec
__global__ void v3(float *a, int *i) {
    __shared__ float inter[6*1024/16];
    for (int blk=1;blk<blklimit;blk++) {
        counter = perthread-1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for(counter=0;counter<perthread;counter++){
            index = col + blockDim.x*counter + row;
            inter[blockDim.x * counter + tid] = a[index];
        }
        __syncthreads();
        counter = perthread-1;
        for(;counter;counter--){
            x += inter[blockDim.x * counter + tid];
        }
        a[index] = x;
        __syncthreads();
    }
}
__global__ void v3(float *a, int *i) {
    __shared__ float inter[6*1024/16];
    for (int blk=1;blk<blklimit;blk++) {
        counter = perthread-1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for(counter=0;counter<perthread;counter++){
            index = col + blockDim.x*counter + row;
            inter[blockDim.x * counter + tid] = a[index];
        }
    }
    __syncthreads();
    counter = perthread-1;
    for(;counter;counter--){
        x += inter[blockDim.x * counter + tid];
    }
    a[index] = x;
    __syncthreads();
}
__global__ void v3(float *a, int *i) {
  __shared__ float inter[6*1024/16];
  for (int blk=1;blk<blklimit;blk++) {
    counter = perthread-1;
    col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
    row = blockIdx.y * blockDim.y + threadIdx.y;
    for(counter=0;counter<perthread;counter++){
      index = col + blockDim.x*counter + row;
      inter[blockDim.x * counter + tid] = a[index];
    }
    __syncthreads();
    counter = perthread-1;
    for(;counter;counter--){
      x += inter[blockDim.x * counter + tid];
    }
    a[index] = x;
    __syncthreads();
  }
}

Continuous range
Gpu: 15.50 msec
Cpu: 75.13 msec

Just zeros
Gpu: 15.04 msec
Cpu: 75.82 msec
__global__ void v3(float *a, int *i) {
    __shared__ float inter[6*1024/16];
    for (int blk=1;blk<blklimit;blk++) {
        counter = perthread-1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for(counter=0;counter<perthread;counter++){
            index = col + blockDim.x*counter + row;
            inter[blockDim.x * counter + tid] = a[index];
        }
    __syncthreads();
    counter = perthread-1;
    for(;counter;counter--){
        x += inter[blockDim.x * counter + tid];
    }
    a[index] = x;
    __syncthreads();
    }
}
__global__ void v3(float *a, int *i) {
    __shared__ float inter[6*1024/16];
    for (int blk=1; blk<blklimit; blk++) {
        counter = perthread - 1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x * blk * perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for (counter = 0; counter < perthread; counter++) {
            index = col + blockDim.x * counter + row;
            inter[blockDim.x * counter + tid] = a[index];
        }
        __syncthreads();
        counter = perthread - 1;
        for (; counter; counter--){
            x += inter[blockDim.x * counter + tid];
        }
        a[index] = x;
        __syncthreads();
    }
}
__global__ void v3(float *a, int *i) {
  __shared__ float inter[6*1024/16];
  for (int blk=1; blk<blklimit; blk++) {
    counter = perthread-1;
    col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
    row = blockIdx.y * blockDim.y + threadIdx.y;
    for(counter=0; counter<perthread; counter++) {
      index = col + blockDim.x*counter + row;
      inter[blockDim.x * counter + tid] = a[index];
    }
  }
  __syncthreads();
  counter = perthread-1;
  for(; counter; counter--) {
    x += inter[blockDim.x * counter + tid];
  }
  a[index] = x;
  __syncthreads();
}

Continuous range
Gpu: 15.50 msec
Cpu: 75.13 msec

Just zeros
Gpu: 15.04 msec
Cpu: 75.82 msec
__global__ void v3(float *a, int *i) {
    __shared__ float inter[6*1024/16];
    for (int blk=1; blk<blklimit; blk++) {
        counter = perthread-1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for(counter=0; counter<perthread; counter++) {
            index = col + blockDim.x*counter + row;
            inter[blockDim.x * counter + tid] = a[index];
        }
    __syncthreads();
    counter = perthread-1;
    for(; counter; counter--){
        x += inter[blockDim.x * counter + tid];
    }
    a[index] = x;
    __syncthreads();
    }
}
__global__ void v3(float *a, int *i) {
    __shared__ float inter[6*1024/16];
    for (int blk=1;blk<blklimit;blk++) {
        counter = perthread-1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for(counter=0;counter<perthread;counter++){
            index = col + blockDim.x*counter + row;
            inter[blockDim.x * counter + tid] = a[index];
        }
    }
    __syncthreads();
    counter = perthread-1;
    for(;counter;counter--){
        x += inter[blockDim.x * counter + tid];
    }
    __syncthreads();
    a[index] = x;
    __syncthreads();
}
__global__ void v3(float *a, int *i) {
  __shared__ float inter[(6*1024)/16];
  for (int blk=1;blk<blklimit;blk++) {
    counter = perthread - 1;
    col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
    row = blockIdx.y * blockDim.y + threadIdx.y;
    for(counter=0;counter<perthread;counter++){
      index = col + blockDim.x*counter + row;
      inter[blockDim.x * counter + tid] = a[index];
    }
    __syncthreads();
    counter = perthread - 1;
    for(;counter;counter--){
      x += inter[blockDim.x * counter + tid];
    }
    a[index] = x;
    __syncthreads();
  }
}

Continuous range
Gpu: 15.50 msec
Cpu: 75.13 msec

Just zeros
Gpu: 15.04 msec
Cpu: 75.82 msec
Continuous range
GPU: 17.53 msec
CPU: 56.29 msec

Just zeros
GPU: 16.68 msec
CPU: 56.09 msec

PREVIOUS

Continuous range
GPU: 15.50 msec
CPU: 75.13 msec

Just zeros
GPU: 15.04 msec
CPU: 75.82 msec

NOW
<table>
<thead>
<tr>
<th>Stall Reasons</th>
<th>NOW</th>
<th>PREVIOUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory dependency</td>
<td>Gpu: 15.50 msec</td>
<td>Gpu: 17.53 msec</td>
</tr>
<tr>
<td>execution dependency</td>
<td>Cpu: 75.13 msec</td>
<td>Cpu: 56.29 msec</td>
</tr>
<tr>
<td>texture</td>
<td>Just zeros</td>
<td>Just zeros</td>
</tr>
<tr>
<td>synchronization</td>
<td>Gpu: 16.68 msec</td>
<td>Gpu: 15.04 msec</td>
</tr>
<tr>
<td>instruction fetch</td>
<td>Cpu: 56.09 msec</td>
<td>Cpu: 75.82 msec</td>
</tr>
</tbody>
</table>
### Memory Allocation

<table>
<thead>
<tr>
<th>cuMemAlloc</th>
<th>cuMemcpyHtoD</th>
<th>cuLaunchKernel</th>
<th>cuMemcpyDtoH</th>
<th>cuMemFree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory HtoD [sync]

Memory DtoH [sync]

Memory HtoD [sync]

Memory DtoH [sync]
Memory Allocation

Real Execution
Memory Allocation

HtoD

DtoH

Real Execution

```
cuMemAlloc

| cuMemcpyHtoD |

| cuLaunchKernel |

| cuMemFree |

| cuMemcpyDtoH |

Memory HtoD [sync]

v3

Memory DtoH [sync]

v3

Memory HtoD [sync]

v3

Memory DtoH [sync]
```
C2050 Execution Time Lines

**Sequential Version**

- H2D Engine
- Kernel Engine
- D2H Engine

**Asynchronous Version 1**

- H2D Engine
- Kernel Engine
- D2H Engine

**Asynchronous Version 2**

- H2D Engine
- Kernel Engine
- D2H Engine
__global__ void v4(float *data, int *i, float *output) {
    extern __shared__ float inter[];
    int counter = 0;
    for (blk=0;blk<blklimit;blk++) {
        counter = perthread-1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for(counter=0;counter<perthread;counter++) {
            index = col + blockDim.x*counter + row;
            inter[blockDim.x * counter + tid] = data[index];
        }
        __syncthreads();
        for(counter=0;counter<perthread;counter++) {
            x += inter[blockDim.x * counter + tid];
        }
        output[tid] = x;
        __syncthreads();
    }
}
__global__ void v4(float *data, int *i, float *output) {
    extern __shared__ float inter[];
    int counter = 0;
    for (blk=0;blk<blklimit;blk++) {
        counter = perthread-1;
        col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x*blk*perthread;
        row = blockIdx.y * blockDim.y + threadIdx.y;
        for(counter=0;counter<perthread;counter++){
            index = col + blockDim.x*counter + row;
            inter[blockDim.x * counter + tid] = data[index];
        }
        __syncthreads();
        for(counter=0;counter<perthread;counter++){
            x += inter[blockDim.x * counter + tid];
        }
        output[tid] = x;
        __syncthreads();
    }
}
__global__ void v4(float *data, int *i, float *output) {
  extern __shared__ float inter[];
  int counter = 0;
  for (blk=0; blk<blklimit; blk++) {
    counter = perthread - 1;
    col = blockIdx.x * blockDim.x + threadIdx.x + blockDim.x * blk * perthread;
    row = blockIdx.y * blockDim.y + threadIdx.y;
    for (counter = 0; counter < perthread; counter++) {
      index = col + blockDim.x * counter + row;
      inter[blockDim.x * counter + tid] = data[index];
    }
    __syncthreads();
    for (counter = 0; counter < perthread; counter++) {
      x += inter[blockDim.x * counter + tid];
    }
    output[tid] = x;
    __syncthreads();
  }
}

Continuous range
Gpu: 10.41 msec
Cpu: 79.37 msec
Continuous range
Gpu: 10.41 msec
Cpu: 79.37 msec
Continuous range
Gpu: 10.41 msec
Cpu: 79.37 msec
Continuous range
Gpu: 10.41 msec
Cpu: 79.37 msec
## SPECIFICATIONS

<table>
<thead>
<tr>
<th>GPU Memory</th>
<th>2 GB DDR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Interface</td>
<td>128-bit</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>29.0 GB/s</td>
</tr>
<tr>
<td>NVIDIA CUDA® Cores</td>
<td>384</td>
</tr>
<tr>
<td>System Interface</td>
<td>PCI Express 2.0 x16</td>
</tr>
<tr>
<td>Max Power Consumption</td>
<td>45 W</td>
</tr>
<tr>
<td>Thermal Solution</td>
<td>Ultra-Quiet Active Finsink</td>
</tr>
<tr>
<td>Form Factor</td>
<td>2.713” H x 6.3” L, Single Slot, Low Profile</td>
</tr>
<tr>
<td>Display Connectors</td>
<td>DVI-I DL+ DP 1.2</td>
</tr>
<tr>
<td>Max Simultaneous Displays</td>
<td>2 direct, 4 DP 1.2 Multi-Stream</td>
</tr>
<tr>
<td>Max DP 1.2 Resolution</td>
<td>3840 x 2160 at 60 Hz</td>
</tr>
<tr>
<td>Max DVI-I DL Resolution</td>
<td>2560 x 1600 at 60 Hz</td>
</tr>
<tr>
<td>Max DVI-I SL Resolution</td>
<td>1920 x 1200 at 60 Hz</td>
</tr>
<tr>
<td>Max VGA Resolution</td>
<td>2048 x 1536 at 85 Hz</td>
</tr>
<tr>
<td>Graphics APIs</td>
<td>Shader Model 5.0, OpenGL 4.5, DirectX 11.2, Vulkan 1.0</td>
</tr>
<tr>
<td>Compute APIs</td>
<td>CUDA, DirectCompute, OpenCL</td>
</tr>
</tbody>
</table>

### PCI Express link performance[^33][^34]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>×1</td>
<td>×2</td>
</tr>
<tr>
<td>1.0</td>
<td>2003</td>
<td>8b/10b</td>
<td>2.5 GT/s</td>
<td>250 MB/s</td>
</tr>
<tr>
<td>2.0</td>
<td>2007</td>
<td>8b/10b</td>
<td>5.0 GT/s</td>
<td>500 MB/s</td>
</tr>
<tr>
<td>3.0</td>
<td>2010</td>
<td>128b/130b</td>
<td>8.0 GT/s</td>
<td>984.6 MB/s</td>
</tr>
<tr>
<td>4.0</td>
<td>2017</td>
<td>128b/130b</td>
<td>16.0 GT/s</td>
<td>1969 MB/s</td>
</tr>
<tr>
<td>5.0[^35][^36] expected in Q1 2019[^37]</td>
<td>128b/130b</td>
<td>32.0 GT/s[^2]</td>
<td>3938 MB/s</td>
<td>7.88 GB/s</td>
</tr>
</tbody>
</table>

[^1]: Using either ×2 or ×4 connection with the appropriate cable.

[^33]: PCI Express 1.0 Specification

[^34]: PCI Express 2.0 Specification

[^35]: PCI Express 3.0 Specification

[^36]: PCI Express 4.0 Specification

[^37]: PCI Express 5.0 Specification
Thank You!
SUSE
We adapt. You succeed.
Unpublished Work of SUSE LLC. All Rights Reserved.

This work is an unpublished work and contains confidential, proprietary and trade secret information of SUSE LLC. Access to this work is restricted to SUSE employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of SUSE. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of SUSE LLC. in the United States and other countries. All third-party trademarks are the property of their respective owners.