GPUs: HMM: Heterogeneous Memory Management

Using HMM to Blur the Lines Between CPU and GPU Programming

John Hubbard (NVIDIA), Jerome Glisse (Red Hat)
May 4, 2017

Agenda: Part 1: HMM Overview

- Why heterogeneous programming matters
- Introduction to us
- GPUs: 10 second overview
- GPUs, CPUs, and Busses: speeds
- GPU programming: barriers to adoption
- HMM: your code, before and after
- CPU and GPU Page faults
- Profiler: see the page faults in action

Agenda Part 2: HMM in depth

- Hardware approaches: CCIX/CAPI
- Software approaches: HMM
- HMM: On the horizon
- HMM: Do’s and Don’ts
- Programmer-level APIs
- HMM: Distant future
- HMM: Under the hood

Agenda Part 3: Q&A session

- Questions and Answer session
- References

HMM Part 1: HMM Overview

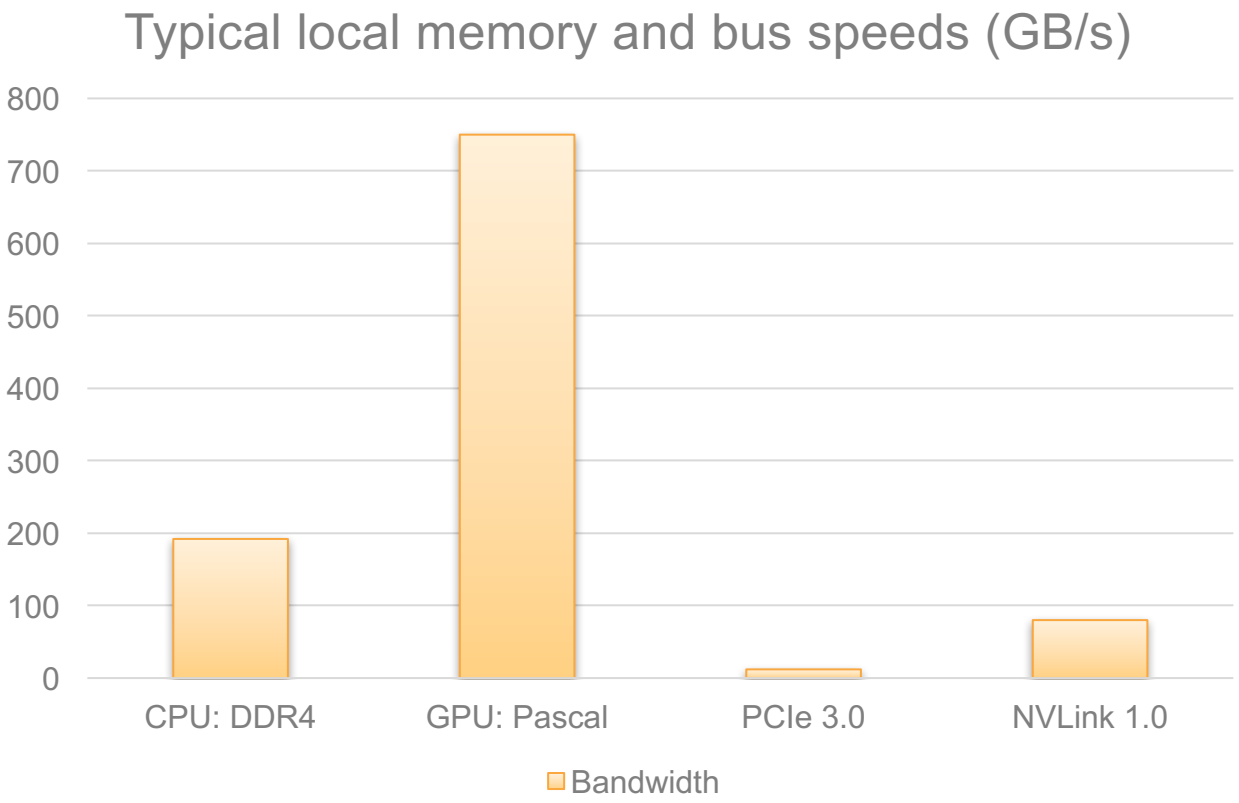
John Hubbard
Principal Software Engineer, NVIDIA

Agenda: Part 1

- Why heterogeneous programming matters
- Introduction to us
- GPUs: 10 second overview
- GPUs, CPUs, and Busses: speeds
- GPU programming: barriers to adoption
- HMM: your code, before and after
- CPU and GPU Page faults
- Profiler: see the page faults in action

GPU Programming: 10 second overview

- 10,000’s of threads
- Huge memory bandwidth
- Device driver required
- CPU cannot directly read from GPU memory
- Separate CPU and GPU memory address space


Typical local memory and bus speeds (GB/s)
800
700
600
500
400
300
200
100
0
CPU: DDR4
GPU: Pascal
PCIe 3.0
NVLink 1.0
Bandwidth

GPU Programming: Barriers to Adoption

- Languages
- Libraries
- Long Tails: lots of rarer applications
- Programming model: see the next slide

Standard Unified Memory (CUDA 8.0)

```
#include <stdio.h>
#define LEN sizeof(int)

__global__ void
compute_this(int *pDataFromCpu)
{
    atomicAdd(pDataFromCpu, 1);
}

int main(void)
{
    int *pData = NULL;
    cudaMallocManaged(&pData, LEN);
    *pData = 1;

    // Run on GPU:
    compute_this<<<512,1000>>>(pData);
    cudaDeviceSynchronize();

    printf("Results: %d\n", *pData);
    cudaFree(pData);
    return 0;
}
```

Unified Memory + HMM

```
#include <stdio.h>
#define LEN sizeof(int)

__global__ void
compute_this(int *pDataFromCpu)
{
    atomicAdd(pDataFromCpu, 1);
}

int main(void)
{
    int *pData = (int*)malloc(LEN);
    *pData = 1;

    // Run on GPU:
    compute_this<<<512,1000>>>(pData);
    cudaDeviceSynchronize();

    printf("Results: %d\n", *pData);
    free(pData);
    return 0;
}
```

Anatomy of a CPU page fault (with HMM)

- *p = 5; // page fault: page is not on CPU
- HMM sees that page is part of a mirrored address space
- HMM notifies the device driver about this matter
- Device driver copies page from GPU to CPU
- Device driver unmaps the page on the GPU (important)
- HMM maps the page on GPU
- Linux kernel replays the CPU page access:
- *p = 5; // continues running

Anatomy of a GPU page fault (with HMM)

- *q = 7; // In GPU code. Page fault because page is on CPU, not on GPU
- GPU raises a CPU interrupt, GPU device driver handles the interrupt
- GPU driver (UVM: Unified [Virtual] Memory driver) asks HMM for physical page
- HMM finds CPU physical page to match the virtual address that GPU faulted on
- HMM and GPU driver coordinate to copy page from CPU to GPU
- HMM unmaps page from CPU (important)
- GPU driver maps page on GPU, and replays the access
- *q = 7; // continues running on GPU

Performance visibility: profiler

```
$ /usr/local/cuda/bin/nvprof --unified-memory-profiling per-process-device ./hmm_app
==19835== NVPROF is profiling process 19835, command: ./hmm_app
Results: 512001
==19835== Profiling application: ./hmm_app
==19835== Profiling result:
Time(%) Time Calls Avg Min Max Name
100.00% 1.2904ms 1 1.2904ms 1.2904ms 1.2904ms compute_this(int*)

==19835== Unified Memory profiling result:
Device "GeForce GTX 1050 Ti (0)"
  Count Avg Size Min Size Max Size Total Size Total Time Name
  2 32.000KB 4.0000KB 60.000KB 64.00000KB 42.62400us Host To Device
  2 32.000KB 4.0000KB 60.000KB 64.00000KB 37.98400us Device To Host
  1 - - - - 1.179410ms GPU Page fault groups
Total CPU Page faults: 2

==19835== API calls:
Time(%) Time Calls Avg Min Max Name
 98.88% 388.41ms 1 388.41ms 388.41ms 388.41ms cudaMallocManaged
  0.39% 1.5479ms 190 8.1470us 768ns 408.58us cuDeviceGetAttribute
  0.33% 1.3125ms 1 1.3125ms 1.3125ms 1.3125ms cudaDeviceSynchronize
  0.19% 739.71us 2 369.86us 363.81us 375.90us cuDeviceTotalMem
  0.13% 524.45us 1 524.45us 524.45us 524.45us cudaFree
  0.04% 137.87us 1 137.87us 137.87us 137.87us cudaLaunch
  0.03% 126.84us 2 63.417us 58.109us 68.726us cuDeviceGetName
  0.00% 11.524us 1 11.524us 11.524us 11.524us cudaConfigureCall
  0.00% 6.4950us 1 6.4950us 6.4950us 6.4950us cudaSetupArgument
  0.00% 6.2160us 6 1.0360us 768ns 1.2570us cuDeviceGet
  0.00% 4.5400us 3 1.5130us 838ns 2.6540us cuDeviceGetCount
```

HMM Part 2: HMM in depth

Jerome Glisse
Linux Kernel Engineer, Red Hat

Hardware solution

- PowerPC architecture CAPI bus
- CCIX consortium (ARM, AMD, ...)

Heterogeneous Memory Management

HMM

- Software solution
- Duplicate CPU page table on the device
- Allow migration of process memory from main memory to device memory

On the horizon

- Duplicate memory
- Migration of file back page (mmap of file)
- Multi-GPU improvements
- Performance optimization

DO

- Prefer large allocation (multi mega bytes)
- Use aligned allocation (posix_memalign()) (2MB alignment or bigger)
- Work on anonymous memory (malloc/calloc/new/...)

DO NOT (better not to)

- fork
- Concurrent CPU and GPU access
- Small allocations
- Changing memory protection (mprotect())

API for share virtual address space

- CUDA
- OpenCL
- Others

Distant future

- More seamless integration with language like C++ (for (par), ...)
- OpenMP

Under the hood

- Add struct page for device memory
- Most kernel code unaware of device memory specificities
- Use mmu_notifier to track CPU page table update
- Grow migrate mechanism to support migration to device memory

HMM Part 3: Question & Answer session

HMM: References

- https://cgit.freedesktop.org/~glisse/linux/log/?h=hmm-v21 (Source code for the kernel patch)
- https://lkml.org/lkml/2017/4/24/747 HMM Patchset v21, in email discussion
- http://www.nvidia.com/object/unix.html UVM (Unified Virtual Memory) drivers for NVIDIA GPUs, that will use HMM kernel features
- https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf (How many ”CUDA cores” in a Pascal GPU? And more)
- http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz4g4BTGGsW NVIDIA CUDA C Language Programming Guide

RED HAT SUMMIT

THANK YOU


plus.google.com/+RedHat


facebook.com/redhatinc


linkedin.com/company/red-hat


twitter.com/RedHatNews


youtube.com/user/RedHatVideos

LEARN. NETWORK.
EXPERIENCE
OPEN SOURCE.