Module 3 Part 3
Application Design for Manycore
Data Parallel Algorithms

Carnegie Mellon University
18-645

Jike Chong
Ian Lane

What we discussed last time:

Fast Platforms + **Good Techniques**

- **Multicore platforms**
- **Manycore platforms**
- **Cloud platforms**

- **Data structures**
- **Algorithms**
- **Software Architecture**

- Highlighted the difference between multicore and manycore platforms
- **Exposing** concurrency in k-means, **Exploiting** parallelism by **exploring** mappings from application to platform
- OpenMP and CUDA for **abstraction for multicore or manycore parallel programming**
- Algorithm design for Parallel Programming

Answers you should know after this…

- What are the important properties of a Map function?
- What are the important properties of a Reduce function?
- What are the important properties of a Scan function?

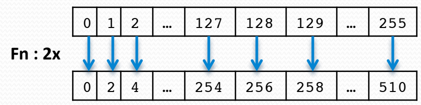
- How to compact an array in a data-parallel way?
- How to find unique elements in an array in a data-parallel way?

Outline

- Algorithm Optimization
  - Data Parallel Algorithms
    - Map, Reduce, and Scan
  - Common Compositions of Data Parallel Algorithms
    - Compact, Find Unique, Building a Flag Array

Data Parallel Algorithms - Map

- **Map** :
  A function that applies a given function to each element of a list, and returning a list of results



- Two important properties:
  - **Side-effect free:**
    Only returning a value, no modifications of state with the rest of the application
  - **Independent**:
    Has an independent piece of work, where its input does not depend on another function

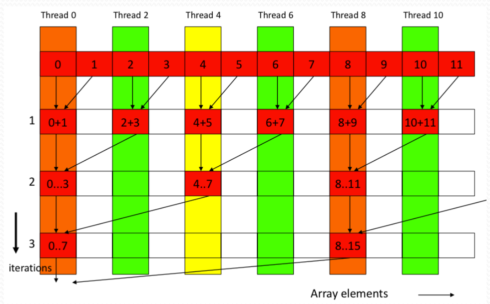
Data Parallel Algorithms - Reduce

- **Reduce**:
  A function that takes in a list of objects and builds up a return value
- Important properties for parallel reduction:
  - Associativity: a+(b+c) == (a+b)+c
  - Allows elements to be reduced in parallel in a “tree”
  - In CUDA, the synchronization has to be managed by the programmer

a+b+c+d+e+f+g+h
= ((a+b)+(c+d))+((e+f)+(g+h))
= (a+b+c+d) + (e+f+g+h)

How Best to Implement Reduce?

- What is an issue with this approach?



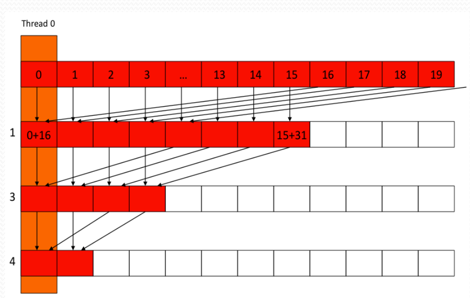

```
unsigned int t = threadIdx.x;
for (unsigned int stride = 1;
     stride<blockDim.x; stride*=2)
{
  __syncthreads();
  if (t % (2*stride) == 0)
    partialSum[t]+=partialSum[t+stride];
}
```

- No more than half of threads will be executing at any time

18-645 – How to Write Fast Code? *Module 3 Part 3* *Carnegie Mellon University (c) 2011-2014* 7

How Best to Implement Reduce?

- What about this approach?



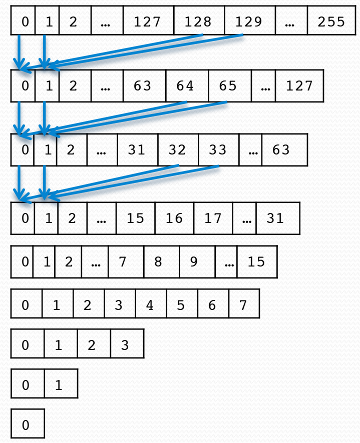

```
unsigned int t = threadIdx.x;
for (unsigned int stride = blockDim.x;
     stride > 1; stride >> 1)
{
  __syncthreads();
  if (t < stride)
    partialSum[t] += partialSum[t+stride];
}
```

- Minimize branch divergence

18-645 – How to Write Fast Code? *Module 3 Part 3* *Carnegie Mellon University (c) 2011-2014* 8

Elimination of __syncthreads()

```
__device__ void sum(float* g_idata, float* g_odata){
    unsigned int tid = threadIdx.x;
    extern __shared__ float s_data[];
    // Assign initial value
    s_data[tid] = g_idata[...];
    __syncthreads();
    if (tid < 128)
        s_data[tid] += s_data[tid + 128];
    __syncthreads();
    if (tid < 64)
        s_data[tid] += s_data[tid + 64];
    __syncthreads();
    if (tid < 32) {
        // No __syncthreads() 32 threads in each
        // warp execute in lock-step with each other
        volatile float* s_ptr = s_data;
        s_ptr[tid] += s_ptr[tid + 32];
        s_ptr[tid] += s_ptr[tid + 16];
        s_ptr[tid] += s_ptr[tid + 8];
        s_ptr[tid] += s_ptr[tid + 4];
        s_ptr[tid] += s_ptr[tid + 2];
        s_ptr[tid] += s_ptr[tid + 1];
    }
    // Write result for this thread block to global memory
    if (tid == 0)
        g_odata[blockIdx.x] = s_data[0];
}
```



18-645 – *How to Write Fast Code?* ***Module 3 Part 3*** ***Carnegie Mellon University (c) 2011-2014*** 9

Data Parallel Algorithms - Scan

- **Scan** (prefix-sum):
  Takes a binary associative operator ⊕ with identity I, and an array of n elements

  [a0, a1, ..., an-1]

  and returns the ordered set

  [I, a0, (a0 ⊕ a1), ..., (a0 ⊕ a1 ⊕ ... ⊕ an-2)].

- **Example:**
  if ⊕ is addition, then scan on the set

  [3 1 7 0 4 1 6 3]

  returns the set

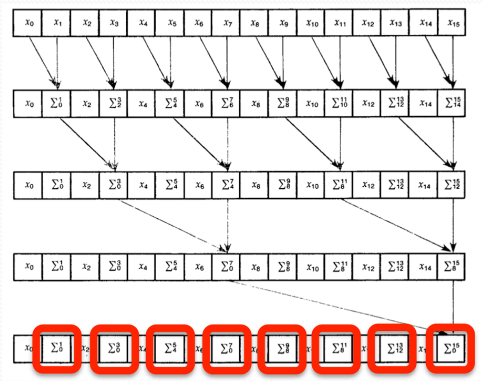
  [0 3 4 11 11 15 16 22]

- **How fast can we do that?**

18-645 – *How to Write Fast Code?* ***Module 3 Part 3*** ***Carnegie Mellon University (c) 2011-2014*** 10

To Implement Scan – Revisit Reduce

- Any techniques for creating for an O(logN)?



Sequential Reduction

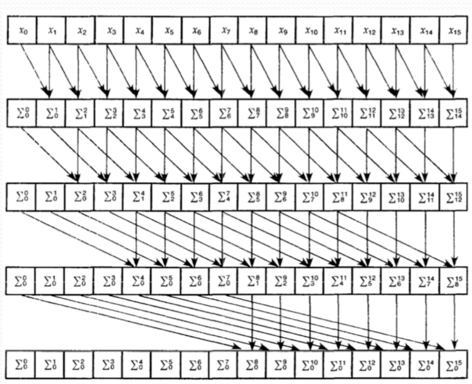
sum[1] = v[1]
For k = 2 to N
    sum[k] = sum[k-1] + v[k]

Parallel Reduction

How do you do a scan?

Scan Techniques

- Note that in the vector reduction, at least half of the processors were idle in any step. Let's have them compute something!



Parallel Reduction

```
for j = 1 to log2(n),
  for all k in parallel,
    if ((k+1) mod 2^j) == 0,
      x[k] = x[k - 2^(j-1)] + x[k]
```

Parallel Prefix Sum

```
for j = 1 to log2(n),
  for all k in parallel,
    if k >= 2^j,
      x[k] = x[k - 2^(j-1)] + x[k]
```

Scan Libraries

- Like **sort**, there exist many optimizations for **scan**
- Expert in the area: Professor Guy Blelloch
  - 15-499: Parallel Algorithms
- For writing fast scan implementations – use the **Thrust library**
  - Thrust library: C++ template library for CUDA
  - Now part of CUDA 4.0

```
#include <thrust/scan.h>

int data[6] = {1, 0, 2, 2, 1, 3};
thrust::exclusive_scan(data, data + 6, data); // in-place scan

// data is now {0, 1, 1, 3, 5, 6}
```

http://code.google.com/p/thrust/wiki/QuickStartGuide#Prefix-Sums

Outline

- Algorithm Optimization
  - Data Parallel Algorithms
    - Map, Reduce, and Scan
  - Common Compositions of Data Parallel Algorithms
    - Compact, Find Unique, Building a Flag Array

Data Parallel Algorithms - Compact

- **Compaction:**
  Removing elements from an array - take in an array, and produce an shorter array

Removing:
[1] [5] [6] [8]

A	B	C	D	E	F	G	H	I	J

A	C	D	E	H	J

- How do we perform removal in parallel?

Data Parallel Algorithms - Compact

- **Compaction:**
  Removing elements from an array - take in an array, and produce an shorter array

- How do we perform removal in parallel?
  - **Map** – create flags (“1” keep, “0” remove)
  - **Scan** – compute index
  - **Map** – copy to new array

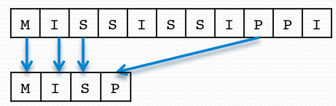
```
if (flag[i] == 1){
    dst[ scanIdx[i] ] = src[i];
}
```

Removing: [1] [5] [6] [8]

	A	B	C	D	E	F	G	H	I	J
Flags	1	0	1	1	1	0	0	1	0	1
scanIdx	0	1	1	2	3	3	3	4	4	5
src	A	B	C	D	E	F	G	H	I	J
dest	A	C	D	E	H	J				

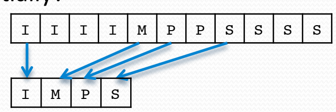
Data Parallel Algorithms - FindUniq

- **FindUniq:**
  Removing duplicates from an array – take in an set, produces a equal or smaller set of unique values



- How do we perform "find unique" in parallel?

- How do we "find unique" sequentially?
  - Sort
  - Iterate through and copy



Data Parallel Algorithms - FindUniq

- **FindUniq:**
  Removing duplicates from an array – take in an array, produces a equal or shorter array

M	I	S	S	I	S	S	I	P	P	I

- How do we perform "find unique" in parallel?
  - **Sort**

I	I	I	I	M	P	P	S	S	S	S

  - **Map** – flag when i^{th} and $(i\text{-}1)^{th}$ element differ
    [0] = 1

1	0	0	0	1	1	0	1	0	0	0

  - **Scan** – create compaction index

0	1	1	1	1	2	2	3	3	3	3

  - **Map** – copy to new array

I	M	P	S

Find Unique – Special case

Traditional Approach

List Sorting
- Sort (**0.310**)

Duplicate Removal
- Cluster-boundary Detection (**0.007**)
- Unique-index Prefix-scan (**0.025**)
- Unique-list Gathering (**0.007**)

0.349 seconds

- Special case:
  What if we know all the possible values the elements can take – such as the 26 letters of the alphabet.
- Sorting is the most expensive step
- How can we avoid sorting?

Find Unique – Special case

Traditional Approach

List Sorting
- Sort (**0.310**)

Duplicate Removal
- Cluster-boundary Detection (**0.007**)
- Unique-index Prefix-scan (**0.025**)
- Unique-list Gathering (**0.007**)

0.349 seconds

- Special case:
  What if we know all the possible values the elements can take – such as the 26 letters of the alphabet.
- Sorting is the most expensive step
- How can we avoid sorting?

Setup a hash table, or lookup table

A	B	C	…	I	J	…	M	…	P	…	S
0	0	0	…	1	0	…	1	…	1	…	1

Alternative Approach

Hash insertion
- Hash write (**0.030**)

Duplicate Removal
- Unique-index Prefix-scan (**0.020**)
- Unique-list Gathering (**0.005**)

0.055 seconds

Jike Chong, Ekaterina Gonina, Kurt Keutzer, "Efficient Automatic Speech Recognition on the GPU", Chapter in GPU Computing Gems Emerald Edition, Morgan Kaufmann, Vol. 1, February 9, 2011.

Populating a Hash Flag Array

- Hash insertion:
  - Leverage the semantics of conflicting writes for non-atomic memory accesses
  - At least one conflicting write to a device memory location is guaranteed to succeed
    - Order of execution is undefined

NVIDIA CUDA C Programming Guide Version 4.0, Chapter 4.2 (page 86)

- For setting flags, map the hash insertion to threads
  - success of insertion in any order can achieve the goal

M	I	S	S	I	S	S	I	P	P	I

A	B	C	…	I	J	…	M	…	P	…	S
0	0	0	…	1	0	…	1	…	1	…	1

Alternative Approach

Hash insertion

Hash write (**0.030**)

Duplicate Removal

Unique-index Prefix-scan (**0.020**)

Unique-list Gathering (**0.005**)

0.055 seconds

How is this relevant to writing fast code?

Fast Platforms + **Good Techniques**

- **Multicore platforms**
- **Manycore platforms**
- **Cloud platforms**

- **Data structures**
- **Algorithms**
- **Software Architecture**

- Introduced the manycore platform HW and SW mental models
- Introduced the terminologies for you to start FLIRTing with the technology
- Introduced design trade-offs in data structures
- Introduced parallel algorithms

- Next lectures: Focus on software architecture