Programming GPGPU with MapReduce

Wenguang CHEN
Tsinghua University
2010/1/15
Program GPU with CUDA

• CUDA is a revolutionary step in GPU programming
  – Much easier than OpenGL, or other shader languages, such as Cg

• However, it is still not very easy to program with CUDA
  – It is not easy to do any parallel programming
What makes parallel computing so difficult

• Identifying and expressing parallelism
  – Autoparallelizing has been failed so far
• Complex synchronization may be required
  – Data races and deadlocks which are difficult to debug
• Load balance...
• Locality optimization
An ideal parallel programming model

- Portable
- Fast
- Easy to program
- Scalable
- Fault-tolerant
- Automatic load balancing
- Versatile
- ...

No Silver Bullet

• If we can not solve the parallel programming in general
  – Can we solve that for a specific domain?
  – Domain specific language (DSL)
    • SQL
    • MAP-Reduce
MapReduce Programming Model

• Borrows from functional programming

• Users implement interface of two functions:

  \[
  \text{map} \ (\text{in\_key}, \text{in\_value}) \\
  \quad \to \ (\text{out\_key}, \text{intermediate\_value}) \ \text{list} \\
  \text{reduce} \ (\text{out\_key}, \text{intermediate\_value \ list}) \\
  \quad \to \ \text{out\_value \ list}
  \]

Jeffrey Dean and Sanjay Ghemawat,
Map-Reduce Architecture

Input key-value pairs

Data store 1 → map → Data store n

(key 1, values...) → (key 2, values...) → (key 3, values...)

== Barrier ==: Aggregates intermediate values by output key

reduce
final key 1 values

reduce
final key 2 values

reduce
final key 3 values
WordCount Program in Map-Reduce

map(string key, string val):
    // key: document name
    // value: document content
    for each word w in value:
        emit_intermediate(w, "1");

reduce(string key, iterator values):
    // key: a word
    // values: a list of counts
    int result=0;
    for each v in values:
        result+=ParseInt(v);
    emit(AsString(result));
MapReduce is promising

• Easy to use
  – Programmers only need to write sequential code
  – Deal with fault tolerance and load balance automatically which is a very desired feature for large scale computing

• Dominated programming paradigm in Internet companies

• Originally support distributed systems, now ported to GPU, CELL, multi-core
  – Phoenix, Mars, Merge etc.
However, there are many dialects of MapReduce

• Because of limited features provided by different architectures

• nVidia GPU as an example
  – No “host” function support
  – No dynamic memory allocation
  – Complex memory hierarchy make it difficult to tune performance
MapReduce on Multi-core CPU (Phoenix [HPCA'07])

- Input
  - Split
  - Map
  - Partition
  - Reduce
  - Merge
  - Output
MapReduce on GPU (Mars[PACT‘08])

Input

MapCount

Prefixsum

Allocate intermediate buffer on GPU

Map

Sort and Group

ReduceCount

Prefixsum

Allocate output buffer on GPU

Reduce

Output

Extra phase to overcome the lack of dynamic memory allocation

Extra phase to overcome the lack of dynamic memory allocation
Program Example

• Word Count (Phoenix Implementation)

```c
...
for (i = 0; i < args->length; i++)
{
    curr_ltr = toupper(data[i]);
    switch (state)
    {
        case IN_WORD:
            data[i] = curr_ltr;
            if ((curr_ltr < 'A' || curr_ltr > 'Z') && curr_ltr != '\')
            {
                data[i] = 0;
                emit_intermediate(curr_start, (void *)1, &data[i] - curr_start + 1);
                state = NOT_IN_WORD;
            }
            break;
    }
...```
Program Example

• Word Count (Mars Implementation)

```c
__device__ void GPU_MAP_COUNT_FUNC
  //(void *key, void *val, int keySize, int valSize)
{
    ....
    do {
      ....
      if (*line != ' ') line++;
      else {
        line++;
        GPU_EMIT_INTER_COUNT_FUNC(
          wordSize-1, sizeof(int));
        while (*line == ' ') {
          line++;
        }
        wordSize = 0;
      }
    } while (*line != '\n');
    ... 
}
```

```c
__device__ void GPU_MAP_FUNC
  //(void *key, void *val, int keySize, int valSize)
{
  ....
  do {
    ....
    if (*line != ' ') line++;
    else {
      line++;
      GPU_EMIT_INTER_FUNC(word,
                          &wordSize, wordSize-1, sizeof(int));
      while (*line == ' ') {
        line++;
      }
      wordSize = 0;
    }
  } while (*line != '\n');
  ...
}
```
Mars Speedup over CPU

- Speedup of Mars vs. Phoenix, according to Mars paper
Grouping Performance is Important

Execution time of WordCount using Mars
Mars: Using Bitonic Sort

- High complexity
  - $O(n(\log n)^2)$
- Unnecessary:
  - The order of the intermediate key/value pairs is usually not desired, as in the case of WordCount, PageViewCount, etc...
HMM: Efficient MapReduce Framework for GPU

- **Goals**
  - Better portability of Map-Reduce on GPU
  - Better Performance

- **Two key ideas:**
  - Lightweight memory allocator
  - Uses hash table to group key/values instead of sorting
Dynamic Memory Allocation on GPU

• Observations:
  – No need for free()
    • Memory can be freed at once at the end of a phase
  – Small, regular size allocations
    • Key/value lists are usually small (<=32byte)
Dynamic Memory Allocation -- Problems

- **Correctness**
  - Two threads should not get the same memory address

- **Performance**
  - Contention
    - Thousands, even millions of thread allocating simultaneously (HMM uses 256*256 threads by default)
  - Long latency
    - The latency of accessing a global address takes 400 cycles
Dynamic Memory Allocation
-- Solutions

• Correctness
  – use atomic operation
    \[
    \text{unsigned atomicAdd(} \text{unsigned *} \text{addr, unsigned inc)}
    \]
    char * memory_ptr;  // the pointer to the free memory space
    void * malloc_from_memory_pool ( int size ){
      return (char *)atomicAdd( (int*)&memory_ptr, size );
    }

• Performance
  – take advantage of the fast \textit{shared memory}
Grouping with Hash Table

- Low complexity:
  - \( O(n) \)
- Closed hashing, enabled with dynamic memory allocator
Problems

• Data race when different threads try to insert into the same value list
  – need a thread-safe list implementation
• The value lists and the value list nodes should be dynamically allocated
  – fortunately, we have the memory allocator
Dealing with Data Races

• No lock on GPU, but there is atomic CAS provided:
  – atomicCAS(addr, old_val, new_val)
    • If *addr==old_val, then *addr=new_val. The compare and store are executed atomically. The return value is *addr before the operation.
  – atomicCAS can be used to implement efficient lock-free data structures, such as linked list

// inserting a new node into a list
Node * new_node=get_new_node();
While(1){
    Node * local_copy_of_list_head=list_head;
    new_node->next=local_copy_of_list_head;
    if(atomicCAS(&list_head, local_copy_of_list_head, new_node)==local_copy_of_list_head)
        break;
}
# Benchmark Applications

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hist</strong></td>
<td>Histogram: generates the histogram of frequencies of pixel values in the red, green and blue channels of a bitmap picture</td>
<td>9MB file, 10 pixels/map</td>
</tr>
<tr>
<td><strong>MM</strong></td>
<td>Dense Matrix Multiplication: $A<em>B=C$, where $A, B$ are N</em>N matrices</td>
<td>N=2048</td>
</tr>
<tr>
<td><strong>SS</strong></td>
<td>Similarity Score: calculates the similarity between a set of documents, given their vector representation</td>
<td>2048 documents, dim=128</td>
</tr>
<tr>
<td><strong>WC</strong></td>
<td>Word Count: count the number of times each word occurs in a file</td>
<td>40MB</td>
</tr>
<tr>
<td><strong>Kmeans</strong></td>
<td>K-means: K-means clustering algorithm</td>
<td>8000 points, 200 clusters, vector dim=40</td>
</tr>
</tbody>
</table>
Benchmarks – under construction

- PLSA: Probabilistic Latent Semantic Analysis
- N-Body: N-body simulation
- Inverted Index
- Linear Regression
Speedup of HMM vs. Mars on:
Histogram, MatrixMultiplication, Kmeans, WordCount, and SimilarityScore
Open Source

• You are welcome to try HMM and give us your feedback!
  – http://sourceforge.net/projects/mapp/
Conclusion

- Map-Reduce is a promising high level parallel programming model for GPU
  - Easy to write
  - Fault-tolerant and load balancing
  - Scalable to multiple cards
  - Portable (ongoing)
Thanks!