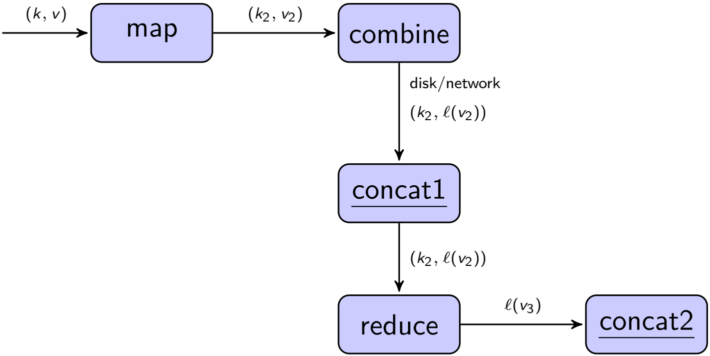
Lecture 22—Programming with OpenCL

ECE 459: Programming for Performance

April 2, 2013

MapReduce


(k, v)
map
(k_2, v_2)
combine
disk/network
$(k_2, \ell(v_2))$
concat1
$(k_2, \ell(v_2))$
reduce
$\ell(v_3)$
concat2

Introduction: MapReduce

Framework introduced by Google for large problems.
Consists of two functional operations: map and reduce.

```
>>> map(lambda x: x*x, [1, 2, 3, 4, 5])
[1, 4, 9, 16, 25]
```

- **map**: applies a function to an iterable data-set.

```
>>> reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])
15
```

- **reduce**: applies a function to an iterable data-set cumulatively.
  ((((1+2)+3)+4)+5) in this example.

MapReduce Intuition

In functional languages, the functions are "pure" (no side-effects). Since they are pure, and hence independent, it's always safe to parallelize them.

Note: functional languages, like Haskell, have their own parallel frameworks, which allow easy parallelization.

Many problems can be represented as a map operation followed by a reduce (for example, Assignment 1).

Hadoop

Apache Hadoop is a framework which implements MapReduce.

- The most widely used open source framework,
  used by Amazon's EC2 (elastic compute cloud).
- Allows work to be distributed across many different nodes
  (or re-tried if a node goes down).
- Includes HDFS (Hadoop distributed file system):
  distributes data across nodes and provides failure handling.
  (You can also use Amazon's S3 storage service).

Map (massive parallelism)

You split the input file into multiple pieces.

The pieces are then processed as (key, value) pairs.

Your **Mapper** function uses these (key, value) pairs and outputs another set of (key, value) pairs.

Reduce (not as parallel)

Collects the input files from the previous map
(which may be on different nodes, needing copying).

Merge-sorts the files,
so that the key-value pairs for a given key are contiguous.

Reads the file sequentially and
splits the values into lists of values with the same key.

Passes this data, consisting of keys and lists of values,
to your **reduce** method (in parallel), & concatenates results.

Combine (optional)

This step may be run right after map and before reduce.

Takes advantage of the fact that elements produced by the map operation are still available in memory.

Every so many elements, you can use your combine operation to take (key, value) outputs of the map and create new (key, value) inputs of the same types.

WordCount Example

Say we want to count the number of occurrences of words in some files.

Consider, for example, the following files:

```
Hello World Bye World
```

```
Hello Hadoop Goodbye Hadoop
```

We want the following output:

```
(Bye, 1)
(Goodbye, 1)
(Hadoop, 2)
(Hello, 2)
(World, 2)
```

WordCount: Example Operations

Mapper

- Split the input file into strings, representing words.
- For each word, output the following (key, value) pair: **(word, 1)**

Reduce

- Sum all values for each word (key) and output: **(word, sum)**

Combine

- We could do the reduce step for in-memory values while doing map.

  Note: here, the output of map and input/output of reduce are the same, but they don't have to be.

WordCount: Running the Example (File 1)

```
Hello World Bye World
```

After map:

```
(Hello, 1)
(World, 1)
(Bye, 1)
(World, 1)
```

After combine:

```
(Hello, 1)
(Bye, 1)
(World, 2)
```

WordCount: Running the Example (File 2)

```
Hello Hadoop Goodbye Hadoop
```

After map:

```
(Hello, 1)
(Hadoop, 1)
(Goodbye, 1)
(Hadoop, 1)
```

After combine:

```
(Hello, 1)
(Goodbye, 1)
(Hadoop, 2)
```

WordCount: Running the Example (Reduce)

After concatenation, sorting, and creating lists of values:

```
(Bye, [1])
(Goodbye, [1])
(Hadoop, [2])
(Hello, [1, 1])
(World, [2])
```

After the reduce, we get what we want:

```
(Bye, 1)
(Goodbye, 1)
(Hadoop, 2)
(Hello, 2)
(World, 2)
```

WordCount Example: C++ Code (1)

- APIs for Java/Python also exist.

```
#include "hadoop/Pipes.hh"
#include "hadoop/TemplateFactory.hh"
#include "hadoop/StringUtils.hh"

class WordCountMap: public HadoopPipes::Mapper {
public:
  WordCountMap(HadoopPipes::TaskContext& context){}
  void map(HadoopPipes::MapContext& context) {
    std::vector<std::string> words =
      HadoopUtils::splitString(context.getInputValue()," ");
    for(unsigned int i=0; i < words.size(); ++i) {
      context.emit(words[i], "1");
    }
  }
};
```

WordCount Example C++ Code (2)

```
class WordCountReduce: public HadoopPipes::Reducer {
public:
  WordCountReduce(HadoopPipes::TaskContext& context){}
  void reduce(HadoopPipes::ReduceContext& context) {
    int sum = 0;
    while (context.nextValue()) {
      sum += HadoopUtils::toInt(context.getInputValue());
    }
    context.emit(context.getInputKey(),
                 HadoopUtils::toString(sum));
  }
};

int main(int argc, char *argv[]) {
  return HadoopPipes::runTask(
    HadoopPipes::TemplateFactory<WordCountMap,
                                 WordCountReduce>());
}
```

Other Examples

- Distributed Grep.
- Count of URL Access Frequency.
- Reverse Web-Link Graph.
- Term-Vector per Host.
- Inverted Index:
  - **Map:** parses each document, and emits a sequence of (word, document ID) pairs.
  - **Reduce:** accepts all pairs for a given word, sorts the corresponding document IDs, and emits a (word, list(document ID)) pair.
  - **Output:** all of the output pairs from reducing form a simple inverted index.

Other Notes

Hive builds on top of Hadoop and allows you to use an SQL-like language to query outputs on HDFS; or you can provide custom mappers/reducers to get more information.

The cloud is a great way to start a new project: you can add or remove nodes easily as your problem changes size. (Hadoop or MPI are good examples).

References:

`http://wiki.apache.org/hadoop/`

`http://code.google.com/edu/parallel/mapreduce-tutorial.html`

Summary

MapReduce is an excellent framework
for dealing with massive data-sets.

Hadoop is a common implementation you can use
(on most cloud computing services, even!)

You just need 2 functions (optionally 3):
mapper, reducer and combiner.

Just remember:
output of the mapper/combiner is input to the reducer.