Monte Carlo Simulation of Photon Migration in a Cloud Computing Environment with MapReduce

Source code documentation for MC321-Cloud

Guillem Pratx

Stanford University

1 Introduction

This document provides basic documentation for MC321-Cloud, a MapReduce implementation of the MC321 photon migration Monte Carlo program. A detailed evaluation (performance and accuracy) is available in the journal article that describes this work [1]. The original MC321 program was written by Steven L. Jacques based on prior collaborative work with Lihong Wang, Scott Prahl, and Marleen Keijzer [2]. A good reference on MapReduce is the original article from Dean *et al.* [3].

MapReduce can be used to run long Monte Carlo simulations in massively-parallel cloud computing environment. As will be shown here, porting an existing Monte Carlo program to MapReduce can be done with little code change. We implemented MC321-Cloud using Hadoop, an open-source implementation of MapReduce which can easily be set-up on a workstation. Large job can then be submitted to a cloud computing service such as Amazon EC2. In this document, we will show how to implement a Monte-Carlo program using Hadoop, and how to run it in a cloud computing environment such as Amazon EC2.

2 File description

The MC321-Cloud package comprises the following files:

mc231.c: Single-threaded MC321 application, included as a reference.

mcmap.c: Map function.

mcred.c: Reduce function

mcconv.c: Utility, used to run the MapReduce application directly in the console.

makefile: Compilation parameters.

batch_ref.sh: Runs the reference MC321 program.

batch_sing.sh: Runs MC321-Cloud single-threaded in the console.

batch_loc.sh: Run MC321-Cloud locally using Hadoop.

batch_rem.sh: Starts a MC321-Cloud simulation on a remote EC2 Hadoop cluster.

seeds.txt: List of 10,000 random seeds.

compare.m: Matlab script to compare the output of the reference MC321 program with the MC321-Cloud program.

3 Single-threaded execution

All source files can be compiled by typing 'make all' in the console. The reference MC321 program can be run by calling the batch_ref.sh shell script. The MC321 code was modified to allow initialization of the random number generator using a list of seed file, fed directly to standard input (stdin). the output of the MC321 program is written directly to disk (filename: output_ref.dat) using a binary file format. Use compare.m in Matlab to load output_ref.dat.

MC321-Cloud can also be executed single-threaded in the console by calling batch_sing.sh. In this script, the seeds are fed to the Map application through a UNIX pipe. The output of the Map function is itself piped to the input of the Reduce application. Because the Reduce application output is formatted using TypedBytes, another utility is called to convert TypedBytes to ASCII (see mcconv.c). The output of the script is written as ASCII to output_sing.txt.

4 Pseudo-distributed execution using a local Hadoop system

It is highly recommended to establish a local Hadoop system to develop, debug and profile MapReduce jobs. Larger jobs can then be submitted to a larger scale cluster. MC321-Cloud was developed for Hadoop 0.21 and is not compatible with older versions of Hadoop. Hadoop can also be run on a single-node in a pseudo-distributed mode where each Hadoop daemon runs in a separate Java process. The instructions for setting up such a system are available on the Hadoop website:

```
http://hadoop.apache.org/common/docs/r0.21.0/single_node_setup.html#Local
```

The hadoop-0.21.0/bin directory in the Hadoop installation should be added to the shell path. For the bash shell, this is done by editing the .bashrc configuration file in the user's directory.

Before running a job, the input random seeds must be copied onto HDFS, Hadoop's distributed file system:

```
hadoop dfs -put  seeds.txt seeds.txt
```

Note it is not necessary to copy the binary files since these are automatically copied to all the nodes using the -file option. The script starts a job using Hadoop Streaming, and using the two compiled binaries as Map and Reduce functions. Communication between Hadoop and external binary files is handled using the TypedBytes format, which is significantly faster than the text format. The input of the Map task is text since the seed file is stored as ASCII. It is necessary to specify a long timeout parameter because the Map task may run a long time before completing a simulation. Note that the output directory must not be already existing in HDFS. After completion of the MapReduce task, the output can be retrieved from HDFS:

```
hadoop dfs -put  seeds.txt seeds.txt
```

5 Remote execution using Amazon EC2

Amazon Elastic Cloud Computing (EC2) allows anybody with a credit card to allocate hundreds of node to run computation. Amazon also provides a Hadoop 0.21 framework (called Elastic MapReduce–or EMR), pre-installed and integrated with Amazon other services. Jobs can be submitted to EMR using a web interface (`https://console.aws.amazon.com/elasticmapreduce`) or a Ruby command-line interface, which can be freely downloaded from Amazon. The command-line interface offers the convenience that jobs cam be saved as scripts and repeatedly submitted without having to enter the information through a web form. An example of such a script is given in batch_rem.sh. The ruby command-line interface must be configured by following the online instructions (`http://aws.amazon.com/developertools/2264`).

Once a job is submitted, it is possible to track its progress by logging in to the web-based AWS management console. If debugging is enabled, one can follow which tasks are currently active. It is also possible to debug tasks by examining the stderr output. The EC2 tab also allows monitoring of the individual nodes running the job. At any time, one can ssh onto a Hadoop node to troubleshoot a problem or monitor resource usage. Jobs can be killed via the web interface.

Tab. 1: TypedBytes used in MC321-Cloud

Code	Type
3	Integer
5	Float
8	Vector

6 Communicating with Hadoop

Map and Reduce tasks must exchange data with Hadoop. To do so, they can use the following formats: Text, RawBytes or TypedBytes. MC321-Cloud uses the latter option. A TypedByte is a byte sequence in which the first byte is a code indicating the type of the following bytes. The following types were used in MC321-Cloud:

Because Hadoop is implemented in JAVA, data must be reformatted as big-endian prior to being sent to Hadoop. While converting little-endian integer to big-endian format is straight-forward (see ByteSwap in mcmap.c), converting floating-point values requires special care. Simply swapping the floating-point bytes may result in an undefined floating-point value, which gets automatically converted to NaN. Therefore, it is necessary to cast the floating-point variable into an integer prior to swapping the bytes.

1. To send a uint32 integer to Hadoop:
   unsigned int a, a_;
   a_ = ByteSwap(a);
   fwrite(&a_, sizeof(unsigned int), 1, stdout);

2. To read a uint32 from Hadoop:
   fread(&a_, sizeof(unsigned int), 1, stdin);
   a = ByteSwap(a_);

3. To send a float to Hadoop:
   float f; unsigned int f_;
   f_ = ByteSwap(float_as_int(f));
   fwrite(&f_, sizeof(unsigned int), 1, stdout);

4. To read a float from Hadoop:
   fread(&f_, sizeof(unsigned int), 1, stdin);
   f = int_as_float(ByteSwap(f_));

References

[1] G. Pratx and L. Xing, "Monte carlo simulation of photon migration in a cloud computing environment with mapreduce," *J. Biomed. Opt.* in press.

[2] S. Jacques, "Light distributions from point, line and plane sources for photochemical reactions and fluorescence in turbid biological tissues," *Photochem. Photobiol.*, vol. 67, no. 1, pp. 23–32, 1998.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.