Homework 02: Matrix Multiplication

Overview and Objectives

It's very common in High Performance Computing that programmers often leverage existing and extremely well-tuned libraries. It's also important that the code we are parallelizing is tuned as efficiently as possible in serial first. The matrix multiplication is an excellent example problem because a naively written program can be several magnitudes slower than a highly optimized library (as you will see). There is no reason to parallelize non-optimal code.

In this assignment, you will link two commonly used HPC libraries: The Intel MKL BLAS library and the HDF5 file IO library. You will use these and compare their performance with code you write that serves a similar purpose.

In the first part, you will write code that reads and writes a matrix in a text format and in the HDF5 format. This will familiarize you with the HDF5 library even though we are using it in a serial way.

In the second part, you will write and compare the results of three matrix multiplication routines:

1. Naive Multiply: this is the straight-forward, no tricks implementation.
2. Block Multiply: this approach divides the each matrix into smaller components, or blocks, that ideally fit into cache.
3. Intel MKL Multiply: you will link your code to the Intel MLK BLAS library.

In addition to code that you submit, you will also write a short report comparing the three matrix multiplication implementations as well as the two IO strategies.

Any code you turn in must include a Makefile that builds your code on Janus with all the optimization flags you specify. The Makefile can be hand-coded or come from build system. In either case, we will want to type "make clean", and not see any executables or object files, and then type "make" to build your executables.

Please be creative. In one of the labs following the assignment, we will run and compare some of the better submission. Your challenge: write a multiply algorithm that is half as slow as the MKL!

Part A: File IO

Each matrix will be described in a file that is one of two formats: either a flat text file or an HDF5 file. You will need to be able to read and write both file types. Write a driver program that accepts two arguments and call this program read_write_matrix. We will pass two filenames to your program. The first file contains a matrix in one of the two formats (*.txt or *.hdf5). Read this matrix and write it out to the second file. We will mix and match formats. For example:

read_write_matrix m100.txt results.hdf5

read_write_matrix m200.txt results.txt

read_write_matrix m10.hdf5 results.txt

The first two numbers of the text file correspond to the shape of the matrix. The first number specifies the number of rows and the second number specifies the number of columns. The next numbers will be the matrix in row_major order. In other words, the second line of the file is the first row of the matrix. The third line of the file is the second row of the matrix. The dimensions of the HDF5 file can be determined from the H5Sget_simple_extent_dims function.

Sample files are located in at learn.colorado.edu Lab 2 Documents folder. All data will be in **positive integer format**.

Resources:

The general HDF5 website. There are several examples in this site, please use them.
http://www.hdfgroup.org/

HDF5 View is a GUI that you can use to verify you are reading (and writing) the files correctly.
http://www.hdfgroup.org/downloads/index.html#hdfview

Linking:

On Janus, you can access the HDF5 library by typing

```
use .hdf5-1.8.7
```

and include the following when compiling your code:

```
HDF5_INCLUDE=-I/curc/tools/free/redhat_5_x86_64/hdf5-1.8.7/
include
HDF5_LIB_DIR=-L/curc/tools/free/redhat_5_x86_64/hdf5-1.8.7/lib
HDF5_LIBS=-lhdf5
```

Part B: Matrix Multiply

You will generate three executables that all take three arguments. The first two arguments are matrices that you will multiply together. The third will be the name of the file that you will write your results to. The block_multiply takes an optional third argument, the block size. For example:

naive_multiply <matrix1>.<ext> <matrix2>.<ext> <result>.<ext>
block_multiply <matrix1>.<ext> <matrix2>.<ext> <result>.<ext>
intel_multiply <matrix1>.<ext> <matrix2>.<ext> <result>.<ext>
block_multiply <matrix1>.<ext> <matrix2>.<ext> <result>.<ext> <block_size>

You should handle non-square matrix cases, although don't worry about error checking. We will give you the correct dimensions for file one and two.

On Janus you can access the intel MKL library by typing

```
use .ics-2012.0.032
```

and linking the following libraries.

```
MKL_INCLUDE=-I/curc/tools/nonfree/redhat_5_x86_64/
ics_2012.0.032/composer_xe_2011_sp1.6.233/mkl/include
MKL_LIB_DIR=-L/curc/tools/nonfree/redhat_5_x86_64/
ics_2012.0.032/composer_xe_2011_sp1.6.233/mkl/lib/intel64
MKL_LIBS=-lmkl_intel_lp64 -lmkl_sequential -lmkl_core
```

Resources:

http://software.intel.com/en-us/articles/intel-mkl/

Part C: The Report

We will provide you with five sample sample input files: m10.*, m100.*, m200.*, m500.*, and m1000.*. Each file will be given in both the text and HDF5 format.

You will need to put a timer in your code to measure the CUP time for each method you write in each executable you build. How you do this is your choice. Please use your implementations to answer the following questions:

1. How do the two file IO methods compare in read and write time as a function of file size? Please graph your results and briefly explain them.
2. How do the matrix multiply methods compare in execution time as a function of the matrix size? Again, please graph your results and explain them.
3. How does block size impact your algorithms for a fixed size matrix (e.g. m1000.txt)? Please plot some of the block sizes you tested and explain the results.

Summary

Your code will build four executables:

1. read_write_matrix
2. naive_multiply
3. block_multiply

4. intel_multiply

Your report should contain three graphs that are based on five different matrices we provide.

1. The time it takes to read and write a matrix as a function of matrix size for each method.
2. The time it takes to multiply to matrices as a function of matrix size for each of the implementations.
3. The time it takes your block multiplication method to compute as a function of block size for a fixed matrix size (use the large file).

You should briefly discuss your results by explaining your data.

Please start early and ask questions when you get stuck. Your grade will be based on the correctness of your implementations and partly on your report. Here is a breakdown:

1. Part A: 30%
2. Part B: 45%
3. Part C: 25%

Appendix: Sample Code

Hdf5 example

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <vector>
#include "hdf5.h"

namespace hdf5
{

void read(const std::string &filename, std::vector<int> &data)
{
	hid_t file_id, dataset_id, space_id;
	herr_t status;

	file_id = H5Fopen(filename.c_str(), H5F_ACC_RDONLY, H5P_DEFAULT);
	dataset_id = H5Dopen(file_id, "DATASET", H5P_DEFAULT);
	space_id = H5Dget_space(dataset_id);

	int length = H5Sget_simple_extent_npoints(space_id);
	int  * image = new int[length];
	status = H5Dread(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
image);

	// Copy back to vector
	data.resize(length);
```

```
	for(int i=0; i<length; ++i){
		data[i] = image[i];
	}
	delete [] image;

	status = H5Sclose(space_id);
	status = H5Dclose(dataset_id);
	status = H5Fclose(file_id);
}
void write(const std::string &filename, std::vector<int> &data)
{
	// HDF5 handles
	hid_t file_id, dataset_id, space_id, property_id;
    herr_t status;

	hsize_t  dims[1] = {data.size()};

    //Create a new file using the default properties.
    file_id = H5Fcreate (filename.c_str(), H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

    //Create dataspace.  Setting maximum size to NULL sets the maximum
    //size to be the current size.
    space_id = H5Screate_simple (1, dims, NULL);

    //Create the dataset creation property list, set the layout to compact.
    property_id = H5Pcreate (H5P_DATASET_CREATE);
    status = H5Pset_layout (property_id, H5D_COMPACT);

    // Create the dataset.
    dataset_id = H5Dcreate (file_id, "DATASET", H5T_STD_I32LE, space_id, H5P_DEFAULT,
property_id, H5P_DEFAULT);

    //Write the data to the dataset.
    status = H5Dwrite (dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
&data[0]);

	status = H5Sclose(space_id);
	status = H5Dclose(dataset_id);
	status = H5Fclose(file_id);
	status = H5Pclose(property_id);
}

};

int main(int argc, char ** argv)
{

	std::vector<int> values(30,10);
	std::fill (values.begin(),values.begin()+10,5);

	std::string filename = "data-1d.hdf5";

	hdf5::write(filename,values);

	std::vector<int> results;
	hdf5::read(filename, results);

	std::cout << "results" << std::endl;
	for(int i=0; i<results.size(); ++i)
	{
		std::cout << results[i] << " ";
	}
	std::cout << std::endl;
```

```
    return 0;
}
```

Makefile

```
CC=/usr/bin/g++

HDF5_INCLUDE=-I/curc/tools/free/redhat_5_x86_64/hdf5-1.8.7/include
HDF5_LIB_DIR=-L/curc/tools/free/redhat_5_x86_64/hdf5-1.8.7/lib
HDF5_LIBS=-lhdf5

INCLUDE=$(HDF5_INCLUDE)
LIB_DIR=$(HDF5_LIB_DIR)
LIBS=$(HDF5_LIBS)

all:
	rm -f example_io
	make example_io

example_io:
	$(CC) example_io.cpp $(INCLUDE) $(LIB_DIR) $(LIBS) -o example_io

clean:
	rm example_io data-1d.hdf5
```

MKL Example

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <vector>

#include "mkl_cblas.h"


void print(double *d1, const int n1)
{
	for(int i=0; i<n1; ++i)
		std::cout << d1[i] << " ";
	std::cout << std::endl;

}

int main(int argc, char ** argv)
{

	const int N = 5;

	double * vector_1 = new double[N];
	for(int i=0; i<N; ++i)
		vector_1[i] = 10;

	double * vector_2 = new double[N];
	for(int i=0; i<N; ++i)
		vector_2[i] = 5;

	// Print
	print(vector_1, N);
	print(vector_2, N);
```

```
        // The number of elements in the vectors
        // 1 = stride (use every element)
        double value = cblas_ddot(N, vector_1, 1, vector_2, 1);

        std::cout << value << std::endl;


        delete [] vector_1;
        delete [] vector_2;

    return 0;
}
```

Makefile

```
CXX=/curc/tools/nonfree/redhat_5_x86_64/ics_2012.0.032/composer_xe_2011_sp1.6.233/bin/
intel64/icpc


MKL_INCLUDE=-I/curc/tools/nonfree/redhat_5_x86_64/ics_2012.0.032/
composer_xe_2011_sp1.6.233/mkl/include
MKL_LIB_DIR=-L/curc/tools/nonfree/redhat_5_x86_64/ics_2012.0.032/
composer_xe_2011_sp1.6.233/mkl/lib/intel64
MKL_LIBS=-lmkl_intel_lp64 -lmkl_sequential -lmkl_core

INCLUDE=$(MKL_INCLUDE)
LIB_DIR=$(MKL_LIB_DIR)
LIBS=$(MKL_LIBS)

all:
        rm -f example_mkl
        make example_mkl

example_mkl:
        $(CXX) example_mkl.cpp $(INCLUDE) $(LIB_DIR) $(LIBS) -o example_mkl
```