Performance Computation of Multiple-Cores on Storing of Data Using HDF5

Balakrishna Vardhineni,
Advisor: Alfa Heryudono, PhD
Department of Mathematics,
University of Massachusetts Dartmouth, MA, USA

ABSTRACT

Data storage is essential to improve business operations, it's no secret that technology glitches can (and do) occur for both small businesses as well as international corporations. Protecting users' and companies' important files and historical archives of documents with storage solutions is a vital part of running a business; it will also help to ensure that files can be recovered if a major loss does occur. Reliability, cost of the storage system, and security offerings are the major considerations to keep in mind as you prepare to implement data storage, performance management, and disaster recovery plans.

As the first step in our study, we performed computation of the efficiency of multiple cores while storing and manipulating the data on HDF5 using MATLAB. We've used one of the most popular data storage systems called HDF (Hierarchical Data Format). The HDF5 format is designed to address some of the limitations of the HDF4 library and to address current and anticipated requirements of modern systems and applications. We used MATLAB's inbuilt library HDF5 operations to perform the storage and manipulation of data. We also tested the speedup of these operations while using embarrassingly parallel computations. Finally, we calculated the average efficiency of multiple CPU cores' performance when executing parallel processing.

Software Used: MATLAB, HDF5

Introduction:

High-Performance Computing (HPC) refers to the practice of taking complex computations and breaking them into smaller pieces. Using the software, each piece of the computation can be solved by CPU cores. It's almost like taking a supercomputer and breaking it down into smaller, more manageable building blocks that can then be used to solve complex scientific problems. HPC can also be very computationally and memory intensive.

Data storage essentially means that files and documents are recorded digitally and saved in a storage system for future use. Storage systems may rely on electromagnetic, optical, or other media to preserve and restore the data if needed. Data storage makes it easy to back up files for safekeeping and quick recovery in the event of an unexpected computing crash or cyberattack.

While Cloud storage is a cloud computing model that enables storing data and files on the internet through a cloud computing provider that you access either through the public internet or a dedicated private network connection. The provider securely stores, manages, and maintains the storage servers, infrastructure, and network to ensure you have access to the data when you need it at a virtually unlimited scale, and with elastic capacity.

As mentioned above, we're going to use both of these functionalities in our project to produce the results of the high-performance efficiency of multiple cores in a system.

HDF5:

HDF5 consists of a file format for storing HDF5 data, a data model for logically organizing and accessing HDF5 data from an application, and the software (libraries, language interfaces, and tools) for working with this format. The HDF5 Data Model, also known as the HDF5 Abstract (or Logical) Data Model consists of the building blocks for data organization and specification in HDF5. An HDF5 file (an object in itself) can be thought of as a container (or group) that holds a variety of heterogeneous data objects (or datasets). The datasets can be images, tables, graphs, and even documents, such as PDF or Excel.

The general paradigm for working with objects in HDF5 is to:

- Open the object.
- Access the object.
- Close the object.

The library imposes an order on the operations by argument dependencies. For example, a file must be opened before a dataset because the dataset open call requires a file handle as an argument. Objects can be closed in any order. However, once an object is closed it no longer can be accessed.

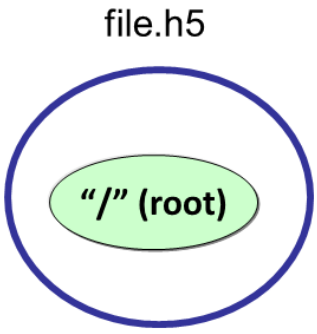

Fig 1: Ref. Image of an HDF5 file

HDF5 in MATLAB:

High-level access functions make it easy to read and view data in an HDF5 file or write a variable from the MATLAB® workspace into an HDF5 file. Low-level functions in the HDF5 library packages provide direct access to the more than 300 functions in the HDF5 C library of the HDF Group. To create or write to non-numeric datasets or attributes, you must use low-level functions.

To use the MATLAB low-level HDF5 functions, you must be familiar with HDF5 C API programming concepts, described on The HDF Group website. MATLAB supports HDF5 version 1.10.8.

Functions:

Read or Write HDF5 Files

h5create	Create HDF5 dataset
h5disp	Display contents of the HDF5 file
h5info	Information about the HDF5 file
h5read	Read data from the HDF5 dataset
h5readatt	Read attribute from the HDF5 file
h5write	Write to HDF5 dataset
h5writeatt	Write HDF5 attribute

Read Operations:

Reading the entire HDF5 Dataset:

To Get the metadata for a dataset from the HDF5 file and then read the dataset.

Displaying the metadata for a dataset '/g4/lat' from the HDF5 file example.h5

```
h5disp('example.h5','/g4/lat');
```

Output:

```
HDF5 example.h5
Dataset 'lat'
    Size:  19
    MaxSize:  19
    Datatype: H5T_IEEE_F64LE (double)
    ChunkSize: []
    Filters:  none
    FillValue:  0.000000
    Attributes:
        'units':  'degrees_north'
        'CLASS':  'DIMENSION_SCALE'
        'NAME':  'lat'
```

Reading the dataset,

```
data = h5read('example.h5','/g4/lat');
```

Output:

```
data = 19×1
   -90
   -80
   -70
   -60
   -50
   -40
   -30
   -20
   -10
     0
```

Write Operations:

Write to the Entire Dataset:

Creating a 10 X 20 dataset named DS1,

```
h5create("myfile.h5","/DS1",[10 20])
```

Write a 10-by-20 array of random numbers to the dataset. Since the dimensions of DS1 are fixed, the amount of data to be written to it must match its size.

```
mydata = rand (10,20);
h5write("myfile.h5","/DS1”, mydata)
```

Displaying the contents of the file,

```
h5disp("myfile.h5")
```

```
HDF5 myfile.h5
Group '/'
    Dataset 'DS1'
        Size:  10x20
        MaxSize:  10x20
```

```
        Datatype:  H5T_IEEE_F64LE (double)
        ChunkSize: []
        Filters:  none
        FillValue:  0.000000
```

Parallel Computing in MATLAB:

Use parallel for loops (parfor) to run independent iterations in parallel on multicore CPUs, for problems such as parameter sweeps, optimizations, and Monte Carlo simulations. parfor automates the creation of parallel pools and manages file dependencies so that users can focus on their work.

parfor Usage:

Syntax: parfor loopVar = initVal: endVal; statements; end

parfor executes for-loop iterations in parallel on workers in a parallel pool. MATLAB® executes the loop body commands in statements for values of loopVar between initVal and endVal. loopVar specifies a vector of integer values increasing by 1. If you have Parallel Computing Toolbox, the iterations of statements can execute on a parallel pool of workers on your multi-core computer or cluster. As with a for-loop, you can include a single line or multiple lines in statements.

By default, if you execute parfor, you automatically create a parallel pool of workers on the parallel environment defined by your default profile. The default parallel environment is Processes. By default, MATLAB uses the available workers in your parallel pool. You can override the default number of workers in a parallel pool by using parpool. When no workers are available in the pool or M is zero, MATLAB still executes the loop body in a nondeterministic order, but not in parallel.

Example of parfor usage:

```
tic
n = 200;
A = 500;
a = zeros(1, n);
parfor i = 1:n
  a(i) = max(abs(eig(rand(A))));
end
toc
```

Why Parallel Computing?

The main advantage of parallel computing is that computers can execute code more efficiently, which can save time and money by sorting through “big data” faster than ever. Parallel programming can also solve more complex problems, bringing more resources to the table. Parallel computing uses multiple computer cores to attack several operations at once. Unlike serial computing, parallel architecture can break down a job into its parts and multi-task them. Parallel computer systems are well suited to modeling and simulating real-world phenomena.

Performance Computation of Multiple Cores:

Amdahl's law states that a code with its parallelizable component comprising 90% of total computation time can at best achieve a 10 times speedup with multiples of workers. A code that is 50% parallelizable speeds up two-fold with lots of workers.

Parallel Efficiency can be calculated by E = S/N

Where S is the Speedup ratio and E is the Average Parallel Efficiency and N is the number of cores used. By generating a plot between time and the number of workers we can understand the behavior of the parallelized code.

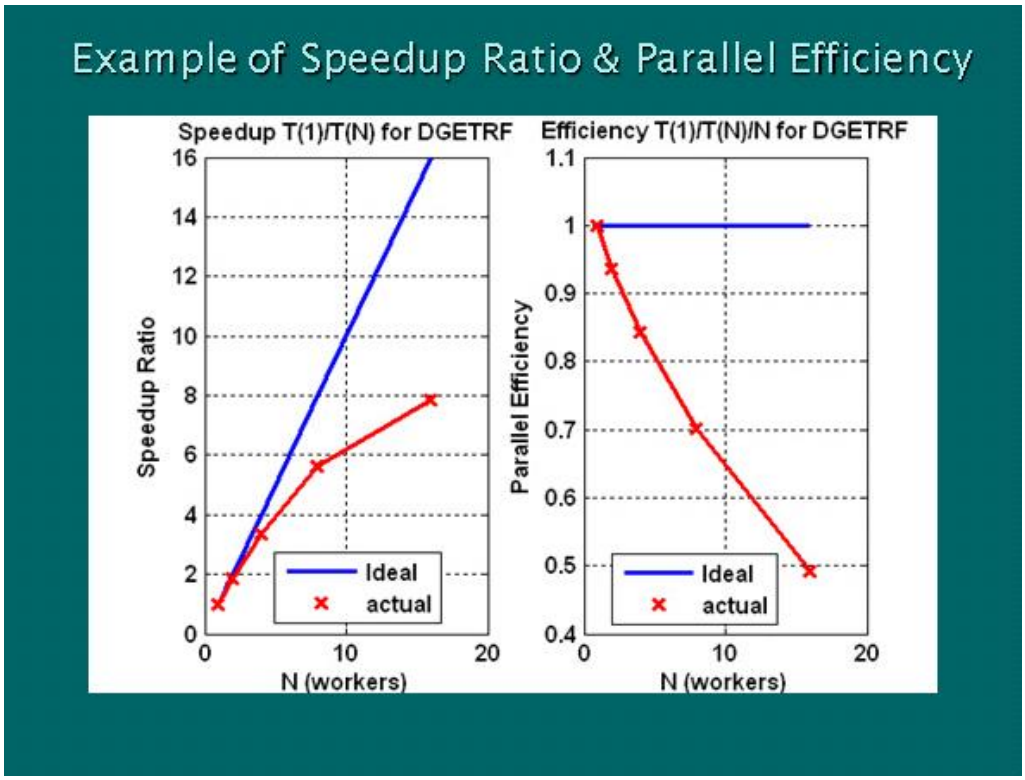

Fig 2: Example of Speedup ration & Parallel Efficiency

Results:

We performed computation of multi-core processing on save and manipulation of data using HDF5 storage file format in MATLAB. Created a function to perform read and write operations of HDRF5. Initially, an empty h5 file was created with an array size of 3X3. If required, the size of the file can be expanded with the write operation. The data is being stored under the "/Data" group. The metadata of the file can be displayed using the h5disp function.

We tried to insert an attribute of "att1" with vector data using h5writeatt and can read the attributes of the h5 file with the h5readatt operation. The h5read operation is used to fetch the records of data.

```
h5create("data.h5",'/Data',[3 3]);
function Hdf5Operations()

% Using class notes
matrix = magic(3);
str= '/Data';
h5write("data.h5",str,matrix);
h5disp("data.h5",str);

% Updating
v = 2*ones(2,1);
h5write("data.h5",str,v,[2 3],[2 1]);
readData = h5read("data.h5",str);
disp(readData);

lastColumn = h5read("data.h5",str,[1 3],[3 1]);
disp(lastColumn);

info = h5info('data.h5');
info.Datasets;

attData = [0 1 2 3];
h5writeatt('data.h5',str,'att1', attData);
att = h5readatt("data.h5",str,'att1');
disp(att);

end
```

Fig.3: Code ref. of HDF5 operations

Firstly, we performed normal computation of the h5 operations using for loop and then we did the same operations on multi-cores and recorded the elapsed time. The speedup can be calculated by

Speedup = Normal elapsed time / Embarrassingly Parallel elapsed time

Finally, the average efficiency of multi-cores is calculated by the speedup divided by the number of cores worked parallelly.

Average efficiency = Speedup / Number of cores

```
% Normal computation
tic
for i = 1:n
    Hdf5Operations;
end
tNormal = toc;
a = sprintf("Execution Time for Normal computation is: %f", tNormal);
disp(a)

% Utilize multiple cores to run embarrasingly parallel computations
tic
parfor i = 1:n
    Hdf5Operations;
end
tparallel = toc;

b = sprintf("Execution Time for Embarrasingly parallel computations is: %f", tparallel);
disp(b);

Sp = tNormal / tparallel;
x = sprintf("Speedup is:%f", Sp);
disp(x);
y = sprintf("Average Efficiency is: %f=%d%%", Sp / n, int8((Sp / n) * 100));
disp(y);
```

Fig.4: Code ref of Normal and Parallel execution

When we tested on our local system with 10 cores in embarrassingly parallel execution, we got the results as followed. For each iteration of loop a random matrix data inserted into data file and stored as a group of data.

The Execution Time for Normal computation is: 1.008709
The execution time for Embarrassingly parallel computation is: 0.914959
Speedup is: 1.102464
The average Efficiency is: 0.110246 = 11%

Conclusion and Discussions:

- After all the performance testing and analysis of multi-core processing we can conclude that high performance can be achieved using parallel computing as expected.
- We've achieved an average efficiency increase of 11% compared to normal computation.
- Using the HDF5 file storage system we were able to achieve the best performance compared to other storage systems.
- As discussed in the results above, the speedup achieved is 1.1 times of normal computation.
- Using CARNIE (A supercomputer of the University of Massachusetts Dartmouth), we can more cores to test the embarrassingly parallel computation more efficiently than local.
- As the number of cores increases there might be higher chances of performance increase in parallel computation.
- HDF5 can be the first choice of storage system selection if the data model looks like a hierarchical format and also if there is a tree-like structure.
- As HDF5 supports heterogenous data it can be used as a storage system for different aspects of software development and business applications.
- As discussed GDF5 is an open-source project it can be used freely to store a large set of data.
- By using low-level functions in the H5 library in MATLAB we can perform more functions along with the functions we discussed above.

References:

- HDRF5 files in MATLAB
  https://www.mathworks.com/help/matlab/hdf5-files.html?s_tid=CRUX_lftnav
- HDF5 library and file format
  https://www.hdfgroup.org/solutions/hdf5/
- HDF5 documentation API reference
  https://docs.hdfgroup.org/hdf5/develop/
- Efficiency of Parallel Algorithm
  https://www.sdsc.edu/~majumdar/thesis/node47.html
- HDF Wikipedia
  Hierarchical Data Format - Wikipedia
- Parallel computing in MATLAB
  https://www.mathworks.com/help/parallel-computing/parfor.html

Appendix

```
clear all; % Start fresh, you can comment this if it is too  annoying

% Create a parallel pool if none exists
n=feature('numcores');
if isempty(gcp())
parpool('local',n);
end
h5create("data.h5",'/Data',[3 3]);
% n=10;
% Normal computation
tic
for i = 1:n
    Hdf5Operations;
end
tNormal = toc;
a = sprintf("Execution Time for Normal computation is: %f" , tNormal);
disp(a)

% Utilize multiple cores to run embarrasingly parallel computations
tic
parfor i = 1:n
    Hdf5Operations;
end
tparallel = toc;

b = sprintf("Execution Time for Embarrasingly parallel computations is: %f" , ↙
tparallel);
disp(b);

Sp = tNormal / tparallel;
x = sprintf("Speedup is:%f", Sp);
disp(x);
y = sprintf("Average Efficiency is: %f=%d%%", Sp / n, int8((Sp / n) * 100));
disp(y);

delete(gcp('nocreate'));


function Hdf5Operations()

% Using class notes
matrix = magic(3);
str= '/Data';
h5write("data.h5",str,matrix);
h5disp("data.h5",str);

% Updating
v = 2*ones(2,1);
h5write("data.h5",str,v,[2 3],[2 1]);
```

```
readData = h5read("data.h5",str);
disp(readData);

lastColumn = h5read("data.h5",str,[1 3],[3 1]);
disp(lastColumn);

info = h5info('data.h5');
info.Datasets;

attData = [0 1 2 3];
h5writeatt('data.h5',str,'att1', attData);
att = h5readatt("data.h5",str,'att1');
disp(att);

end
```

```
Execution Time for Normal computation is: 1.008709
Execution Time for Embarrassingly parallel computation is: 0.914959
Speedup is: 1.102464
Average Efficiency is: 0.110246 = 11%
```