Green Computing: A Parallel Processing Approach on Data Encoding Problems

Team: LAHS-6

Steven Chen, Los Alamos High School

Final Report, April 5th, 2017

Executive Summary

Data coding techniques have been used in wired and wireless data transmission areas for many years. These techniques involve heavy mathematic computing operations. Usually, byte streams are used in traditional data transmission and normally embedded processors are used to handle efficient byte stream data coding processes. The "software data coding" method was not used in very large scale storage systems until five years ago. This is due to the fact that, back then the CPU computing power could not efficiently handle software coding processes. Today we see that advanced multicore computing systems are commonly used in many commercial and scientific applications. Improving the Energy Efficiency of Software Systems for Multi-Core Architectures is an important research topic. Energy efficient based green computing attracts a lot of attention these days. Power and energy efficiency are important challenges for the High-Performance Computing (HPC) community. This year I am focusing on the study of "Green Computing" and investigating how to utilize parallel processing to improve the energy efficiency of multicore computing systems.

In this project, I enhanced and implemented a parallel encoding software called PARC. The PARC software is an enhancement of my project from last year PAR-EC. I conducted various workload tests on a single compute node and studied the relation of parallel processing and power consumption. I demonstrated the advantage of using parallel processing on multicore computing systems. The PARC software showed significant reductions in total encoding time and energy cost. Furthermore, my results showed that I can obtain almost linear-scaling bandwidth on a single multicore machine in terms of processing time reduction and bandwidth improvement. Lastly, the energy cost savings provides strong proof of the benefit of adapting parallel programming in general applications such as storage systems.

1. Introduction – Investigating the Problem and Motivation

1.1 Investigating the Problem: Multicore programming and data encoding challenges in big data computing environments

Today, multicore processors (Intel or ARM based) are used in numerous commercial products such as smartphones, tablets, laptops, desktops, workstations, and servers. Almost all computers now are parallel computers. It is often heard that people ask "Why can't my applications run on multiple CPU cores?" In order to take advantage of multicore chips, applications and systems should be re-designed to fully utilize potential parallelism in applications and parallel computing capabilities [1]. Furthermore, to capitalize fully on the power of multicore systems, we need to adopt parallel programming models, parallel algorithms, and parallel programming libraries that are appropriate for these multicore computing systems.

Data coding techniques have been used in wired and wireless data transmission areas for many years. These techniques involve heavy mathematic computing operations. Usually, byte streams are used in traditional data transmission and normally embedded processors are used to handle efficient byte stream data coding processes. The "software data coding" method was not used in very large scale storage systems until ten or twenty years ago. This is due to the fact that, back then the CPU computing power could not efficiently handle software coding processes. Generally, the hardware-based technologies are used to support coding solutions, such as RAID (originally redundant array of inexpensive disks, now commonly redundant array of independent disks). RAID technologies were proposed, designed and have been used in storage systems since 1990 [2]. As hard disk capacity gets increasingly larger, the data rebuilding time gets dangerously long when a disk fails. It also increases the chance that another disk will fail before the rebuilding process is completed. Hardware based RAID technologies are running into limitation and cannot handle very large storage systems [2].

Software coding processes have become increasingly popular in recent years. Today with advanced hardware (HW) design technologies, such as multicore CPU and advanced vectoring processing techniques (SIMD: Single Instruction and Multiple Data), individuals and corporations can effectively apply software coding techniques on storage systems. Data storage systems can greatly benefit from software coding techniques due to improved reliability of the storage media [3][4][10][11][12].

1.2 Motivation

My main motivation behind this project was to answer the question of "How to efficiently utilize multicore computing systems in regard to the data coding problem?" By definition, a multi-core processor is a single computing component with two or more independent actual processing units called "cores". Cores are the units that read and execute program instructions. The improvement in performance gained by the use of a multi-core processor depends on the software algorithms used as well as their implementation. In particular, possible gains are limited by the portion of the software that can be run in parallel simultaneously on multiple cores. The challenge

of programming multi-core processors is legitimate with tangible applications. In this case, there is little or no hidden, parallelism to be found. Parallelism should be exposed to and managed by software. Concurrent computing and parallel computing are the two most common approaches to utilize multicore computing systems.

The current data coding software approach is done with a single process approach [3][10][11]. This approach cannot fully utilize a multicore computing system. Most applications are still using single CPU computing models. To improve this situation and to capitalize fully on the power of multicore systems, we need to adopt programming models, parallel algorithms, and programming languages that are appropriate for the multicore world. Furthermore, we need to integrate these ideas and tools into the courses that educate the next generation of computer scientists.

In this project, I investigated the potential task parallelism features of the data encoding problem on shared and non-shared data objects. I leveraged the open source erasure code software library, applied the MPI parallel programming library [6][7], and implemented a parallel erasure coding software (PARC) on multicore computing systems. The original parZFEC software [17] is implemented using the zfec open source python encoding function [10]. The PARC software, uses the open source Jerasure 1.2's C library's encoding function (mainly the reed_sol_vandermonde encoder) [18]. In this project, I chose the concurrent computing approach with the implementation of parallel coding software. The focus of this project was to study the impact of concurrent coding processes on multicore computing systems. This was modeled using concurrent data encoding software using MPI on multicore computing systems [8][9]. The results were evaluated by examining the parallel coding software in terms of coding bandwidth improvement and energy cost savings. I then drew conclusions based on the data collected.

2. Introduction of Data Encoding Problem and Encoding Mathematical Calculation,

In information theory, an erasure code is a forward error correction (FEC) code for the binary erasure channel, which transforms a message of N symbols into a longer message (code word) with n symbols such that the original message can be recovered from a subset of the M symbols. The fraction r = N/M is called the code rate. Reed–Solomon codes are a group of error-correcting codes that were introduced by Irving S. Reed and Gustave Solomon in 1960. In coding theory, the Reed–Solomon code belongs to the class of non-binary cyclic error-correcting codes. The Reed–Solomon code is based on univariate polynomials over finite fields. The Reed-Solomon codes are block-based error correcting codes with numerous applications in digital communication and storage. In Reed and Solomon code we normally use a pair (N, M) to illustrate the coding scheme. It can take a message, break it into N "data" pieces, add M "parity" pieces, and then reconstruct the original from N of the (N+M) pieces.

The Reed-Solomon algorithm creates a coding matrix in which you multiply your data matrix to create the coded data. The examples below (Figure-1A and Figure-1B) use a "9+3" coding system, where the original file is broken into 9 "data" pieces (D1 to D9), and then 3 "parity" pieces (C1, C2, C3) are added. The matrix is set up so that the first nine rows of the result are the

same as the first nine rows of the input. This means that the data is left intact, and all it is really doing is computing the parity [3]. We have the entire storage set be resilient up to three failures.

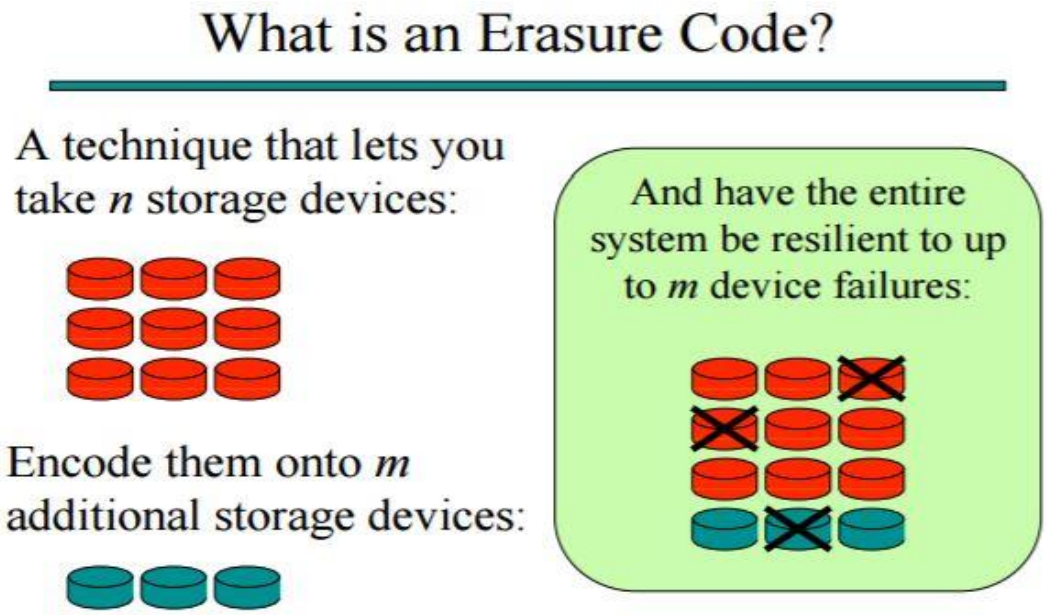

Figure-1A: A (9,3) or “9+3” Erasure Coding Example

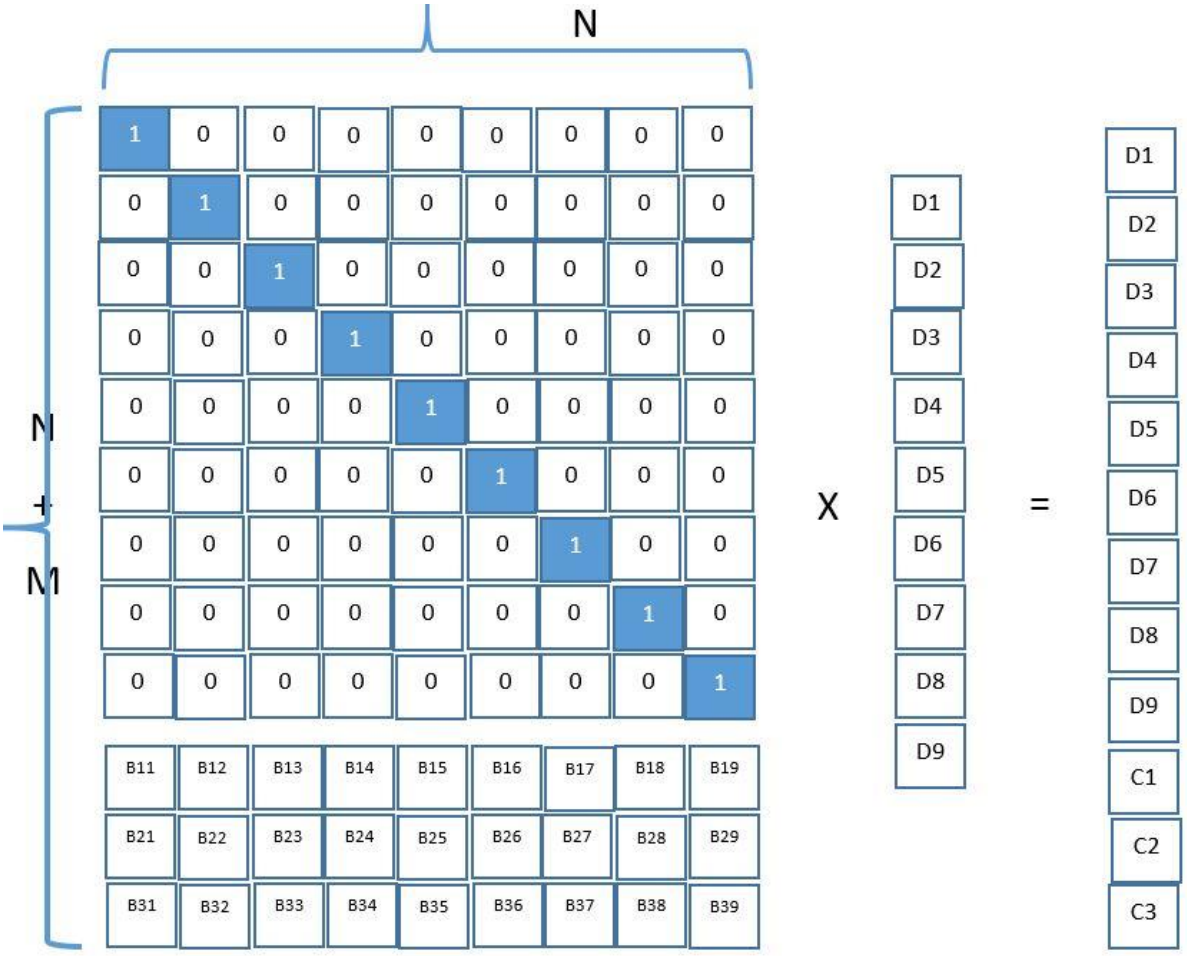

Figure-1B: A (9,3) or "9+3" Erasure Coding Encoding Mathematic Calculation

Here is a brief description of the basic encoding mathematical calculation (Figure-1C). Both Reed-Solomon encoding and decoding processes involve heavy array, matrix, and table look-up procedures [4]. They require powerful computing systems to handle these computing procedures. Here I provide a simplified description of the coding arithmetic operations. Matrix A, matrix E, data matrix D, matrix F, and identity matrix are defined here as:

A = I E = Data D = the source data

F Code

I = $\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$

F is used as the checksum calculation matrix. F is defined as

F = $\begin{bmatrix} f_{1,1} & f_{1,2} & \dots & f_{1,n} \\ f_{2,1} & f_{2,2} & \dots & f_{2,n} \\ \vdots & \vdots & & \vdots \\ f_{m,1} & f_{m,2} & \dots & f_{m,n} \end{bmatrix}$ or $\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 3 & \dots & n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \dots & n^{m-1} \end{bmatrix}$

If $f_{i,j} = j^{i-1}$

A simplified encoding process can be viewed as A * D = E

I applied the encoding process on the source data D, generated encoded data chunks and code chunks, and stored the generated chunks on storage devices.

{Data chunks} = Identity Matrix * D

{Code chunks} = F * D

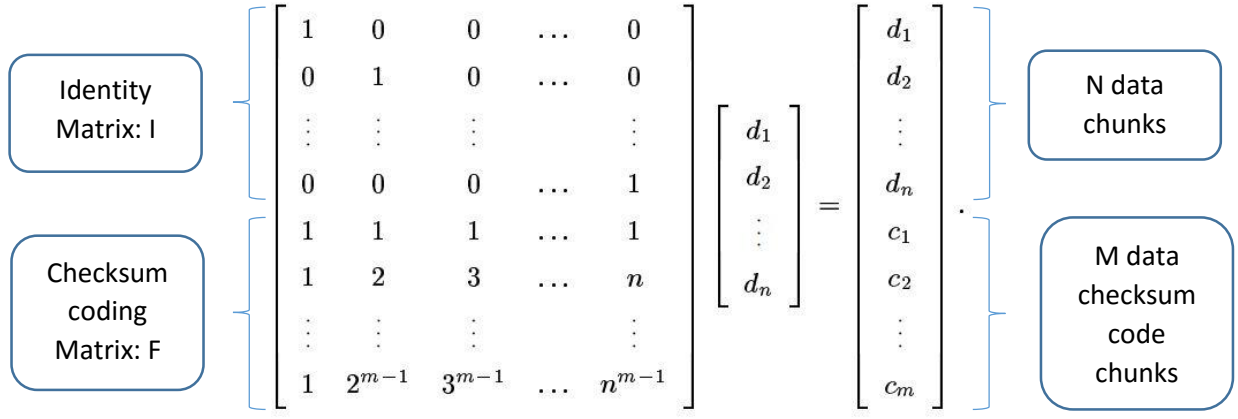

Figure-1C: A brief matrix calculation using Reed-Solomon encoding

3. The Proposed Software Architecture and Implementation (PARC)

The PARC can handle two data encoding process types: multiple no-shared object encoding (N-to-M parallel I/O model) and single shared-object encoding process for very large files (1-to-N parallel I/O model). I have designed four different MPI process types in parEC: Main Process (MPI process Rank 0), SourceDataExplorer Process (MPI process Rank 1), FinalReport Process (MPI process Rank 2), and PARC Process (MPI processes Rank 3 to N-1). The proposed PARC software system is shown in the Figure-2. The function of each type of MPI process is described as follows:

- Manager: This function of this process is to coordinate the other three MPI process types and handle the overall administration jobs. First, it sends the "INIT" message to all other MPI processes and starts the initialization process. It asks the SourceDataExplorer process to discover how many sources objects as passed to the PARC program and waiting to be processed. Then it checks the incoming message queue and determines which PARC process is asking for a source object. Furthermore, it also inspects the receiving message from a PARC process and updates the encoding result for each processed object data. It then picks an unprocessed source object data from the source data queue, packs the source data object's information into a message, and sends it back to the PARC process. A demand based workload assignment protocol is maintained between the Manager Process and parEC processes. When all source data objects are processed by PARC processes, the main manager process will issue an "ENDTAG" message to all PARC processes and finalize the encoding process.
- SourceDataExplorer: The SourcedataExplorer process is the READDIR/file tree walk function on "source object data" location. It explores each individual source object data. It records each object information. This includes: filename/objectname, object ID, size, encoding scheme ratio (K, M), and destination location. All source data objects will be put into a dataQueue for further workload assignment. The SourceDataExplorer process also handles multi- part-object data partitioning. If a file size is too big, the DataExplorer process invokes a partitioning process (called single shared object 1-to-N parallel I/O chunking process), dividing this file into N sub-chunks, and inserting information of each sub-chunk into the data queue.
- Finalreport: After all the source object data is encoded, the mainManager process will ask the FinalReport to prepare a final report. The final report incudes:
    - Each PARC process: number data assigned, each assigned data's starting encoding time and finished encoding time, the first assigned data's starting processing time, the last assigned data finished time, total encoding time, total encoding data size, average assigned data size, the minimal data size of assigned data objects, the maximal data size of assigned data, and encoding bandwidth
    - Accumulated encoding bandwidth from PARC processes
    - Parallel encoding time calculation from PARC processes

- PARC MPI encoding process: After receiving the "INIT" message from the main manager process, each PARC process sends a "WORKLOAD" tag message and asks for the next object data to be encoded. If a PARC process has finished the encoding process on an assigned data object, the PARC process stores the encoding results, such as starting time, finished time, and PARC process Rank information in a message. The PARC process is the core computing procedure to realize the coding task. In this project, I intend to take the task parallelism feature of a multicore computing system and study the benefit of applying parallel programming on multicore computing systems.

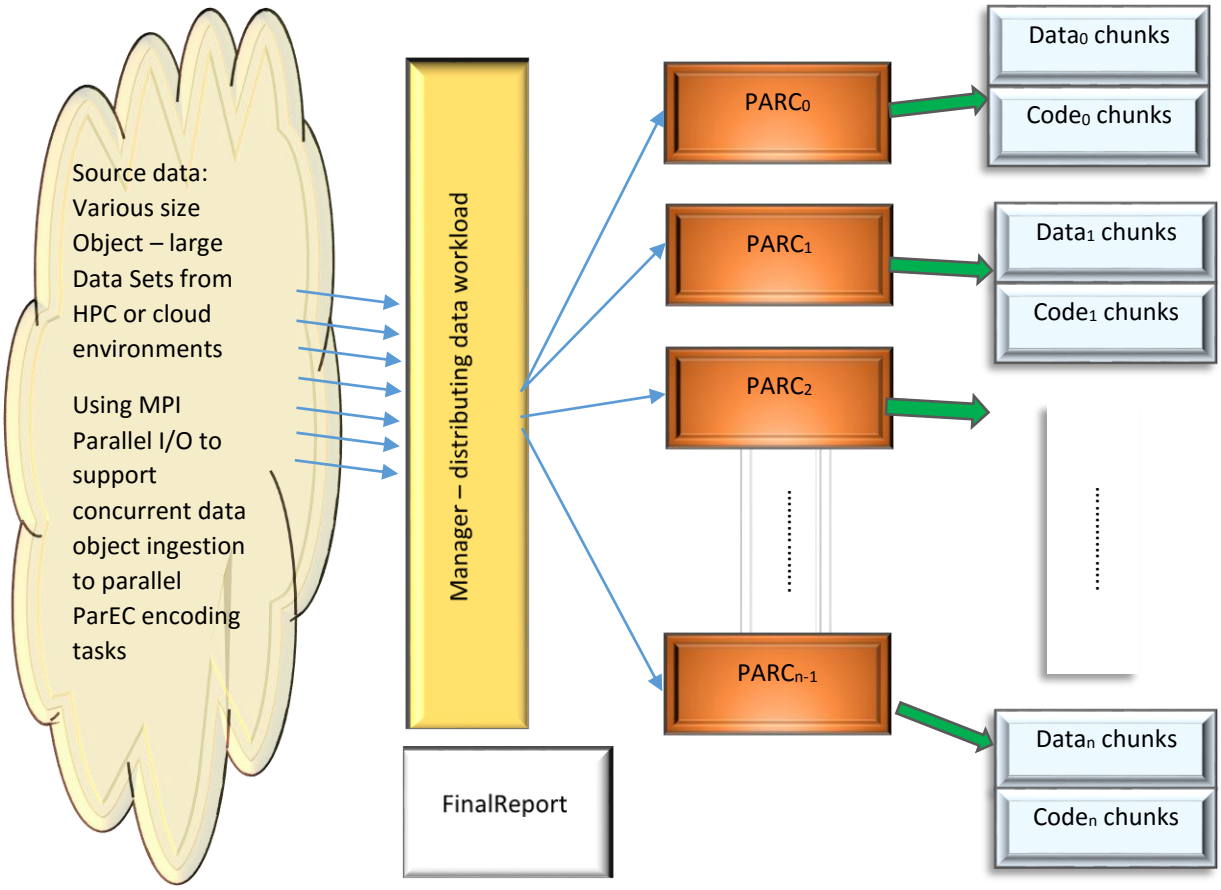

Figure-2: PARC parallel software system diagram

4 Software and Hardware used in Code Development and Testing

4.1 Software

- C programming language: gcc C compiler
- Centos 7.3 Linux Operating System
- MPI programming library: OPENMPI 1.10.1 release

- Jerasure 1.2 release [10]: Jerasure 1.2 encoding sample program is running as a single process program. Jerasure 1.2 is a C library released in 2007 that supports a wide variety of erasure codes, including Reed-Solomon (RS) coding, Cauchy-RS coding, general Generator matrix and bit-matrix coding, and Minimal Density RAID-6 coding [10]. Reed-Solomon coding may be based on Vandermonde or Cauchy matrices, and word-size may be 8, 16 or 32 bitwise. I adapted and converted the Jerasure's Reed-Sol-Van encoding process into a library function and applied it in the PARC software.

4.2 Hardware used in this project

Two HPZ620 workstations are used for performance testing and power studies.

- Computing node: One HP-Z620 workstation: Single Six-Core CPU/12 threads, 64 GB DDR3 memory, two Micro 2TB PCI-E NVMe Flash cards (installed BTRFS Raid-0, 2.2GB/sec write bandwidth)
  PARC MPI processes are launched on this compute nodes.
- Power measurement node: One HP-Z620 workstation: Single Six-Core CPU/12 threads, 64 GB DDR3 memory, two Micro 2TB NVMe Flash
- A WattsUP/.net power meter [16]: Features of this power meter are listed here: Simply plug any device into Watts Up and the meter instantaneously displays the wattage (power) being used, as well as the cost in dollars and cents. Watts Up provides lots of information yet it is simple to use. Only 6 values are displayed in the main modes. The other information is available in the detail mode. Clear English wording (not abbreviations) on the LCD is used to describe what is displayed. WattsUp/.net incorporates sophisticated digital electronics that enable precise and accurate measurements. State-of-the-art digital microprocessor design utilizes high-frequency sampling of both voltage and current measurements for true power. Power factor is captured so even phase-shifted loads such as motors are accurately measured. Fast, intuitive and easy-to-use, WattsUp/.net meter quickly and accurately measures any 120V AC appliance. This Watts Up Standard Monitor will provide the following information: current watts, minimum watts, maximum watts, power factor, cumulative watt hours, average monthly kilowatt hours, tier 2 kilowatt hour threshold (used to calculate secondary kWh rates), elapsed time, cumulative cost, average monthly cost, line volts, minimum volts, maximum volts, current amps, minimum amps, and maximum amps. The WattsUp/.net has memory and comes with Windows reporting software and communications cable to chart results and also logs a few more parameters than the standard.

In the Figure-3, the testing environment setup and electric power wiring diagram are shown.

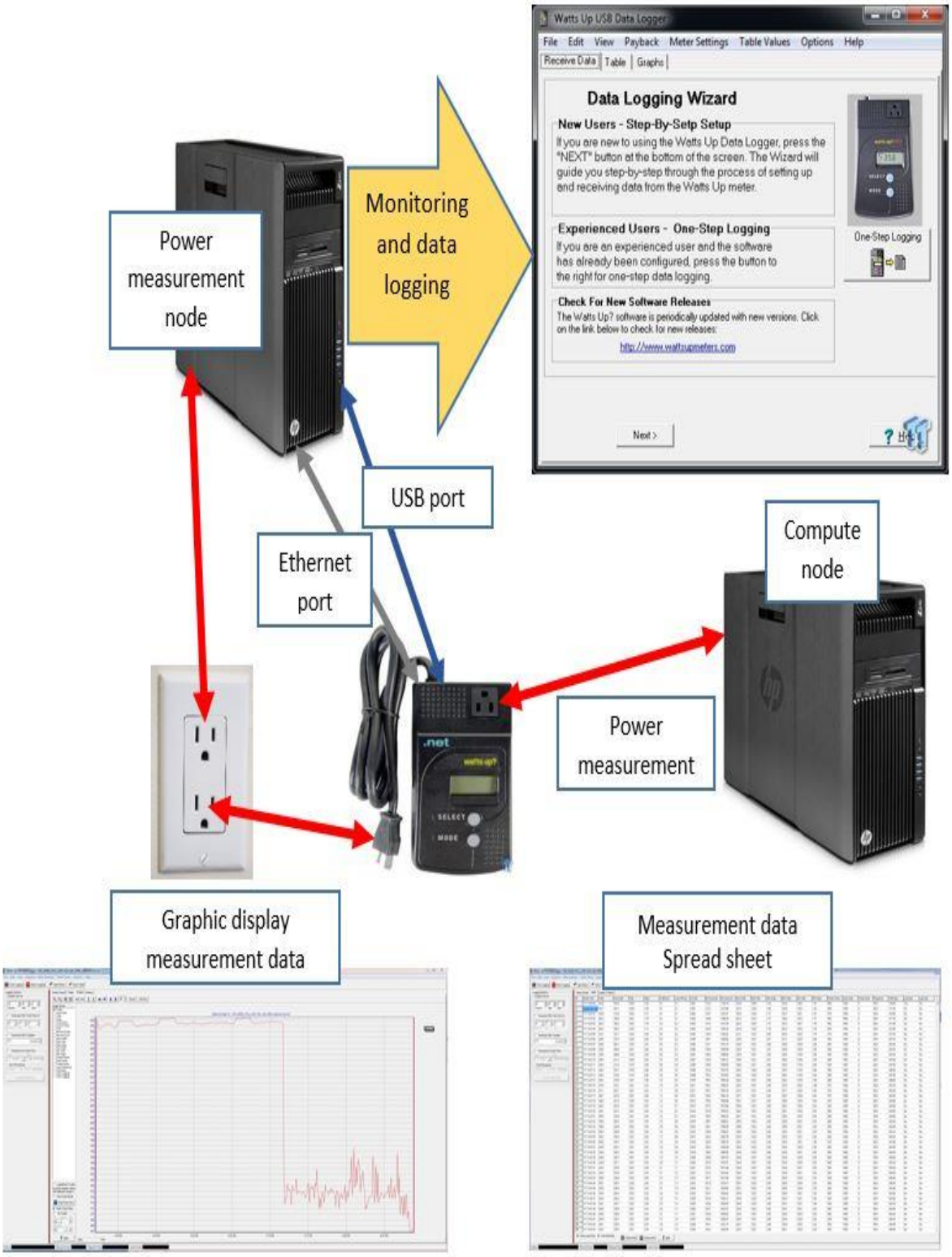

Figure-3: Testing Machines and WattsUp/.net Power Meter Setup

5 Performance Testing and Power Consumption Evaluation

I choose both strong scaling and weak scaling testing cases in this project.

- Weak Scaling Testing Cases: In the weak scaling case the problem size (workload) assigned to each PARC encoding stays constant and additional elements are used to solve a larger total problem. Therefore, this type of measurement is justification for programs that take a lot of memory or other system resources (CPU core). In the case of weak scaling, linear scaling is achieved if the run time stays constant while the workload is increased in direct proportion to the number of processors [19].
- Strong Scaling Testing Cases: In the case the problem size stays fixed but the number of processing elements are increased. This is used as justification for programs that take a long time to run (something that is cpu-bound). The goal in this case is to find a "sweet spot" that allows the computation to finish in a reasonable amount of time, but not waste too many cycles due to parallel overhead [19].

5.1 Encoding Bandwidth Evaluation

This illustrates the weak scaling test case. Each PARC process is handling a 34GB data set. I measured the processing time and calculated the encoding bandwidth for (1,2,3,4,5 and 6) parallel encoding test cases.

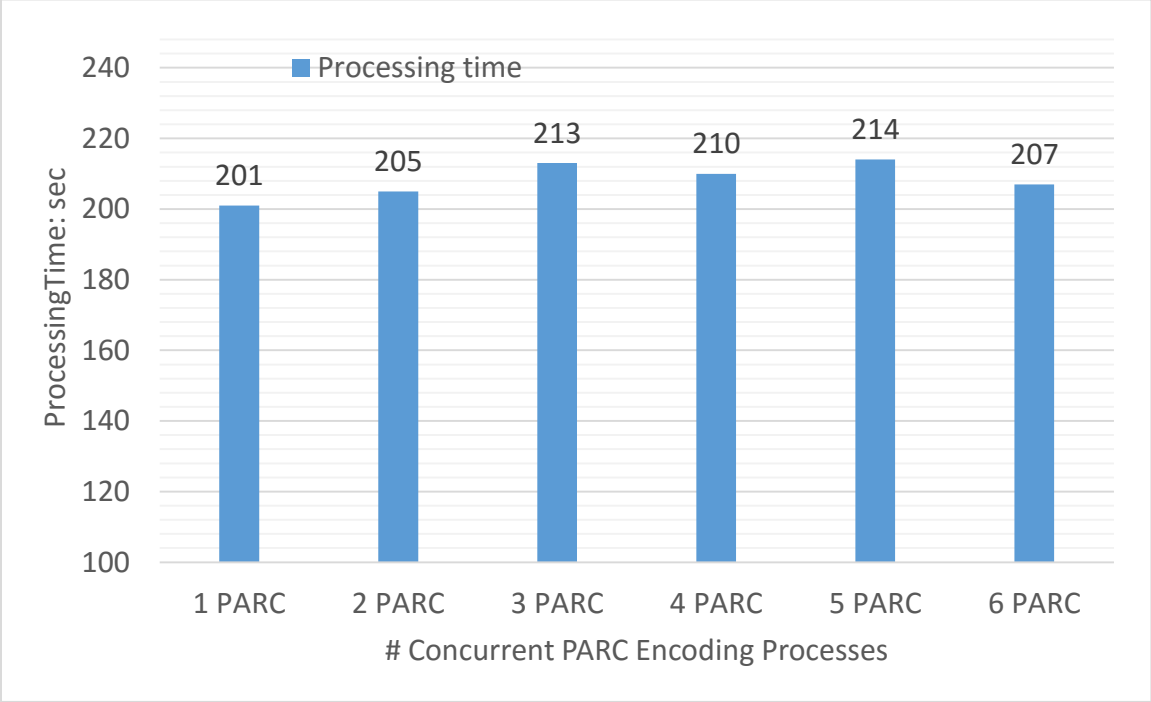

Figure 4A: Processing time for one to six parallelism tests (weak scaling)

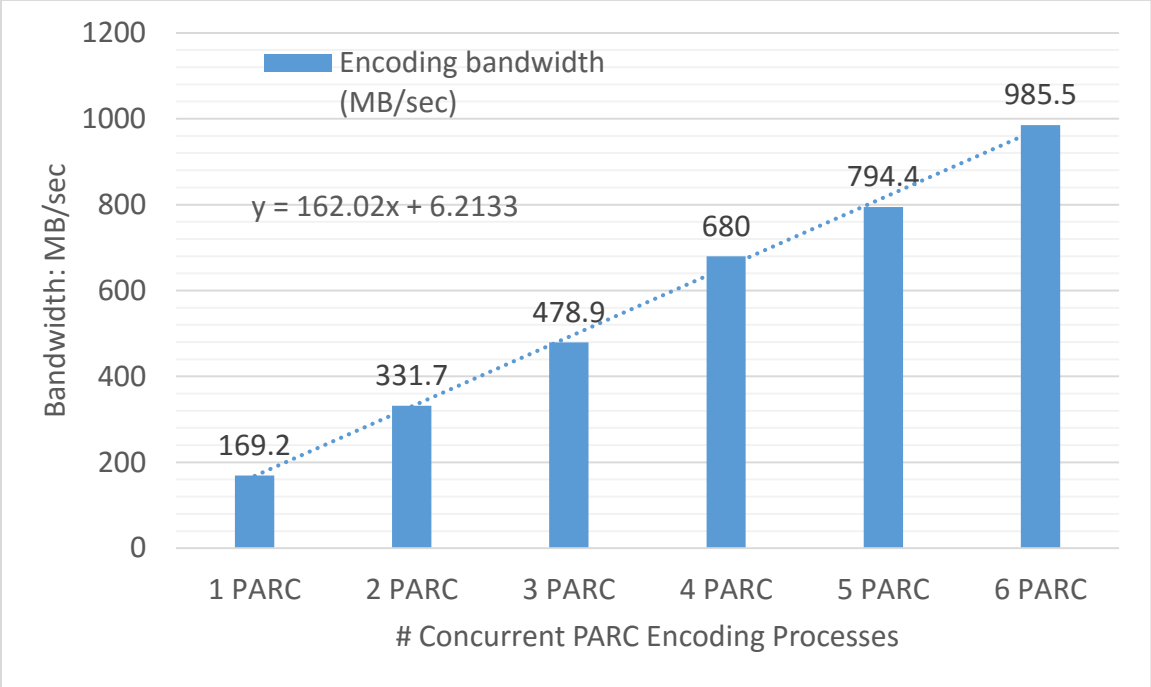

Figure 4B: Encoding bandwidth for one to six parallelism tests (weak scaling)

The bandwidth improvement trend line in Figure-4C shows an almost near linear scaling.

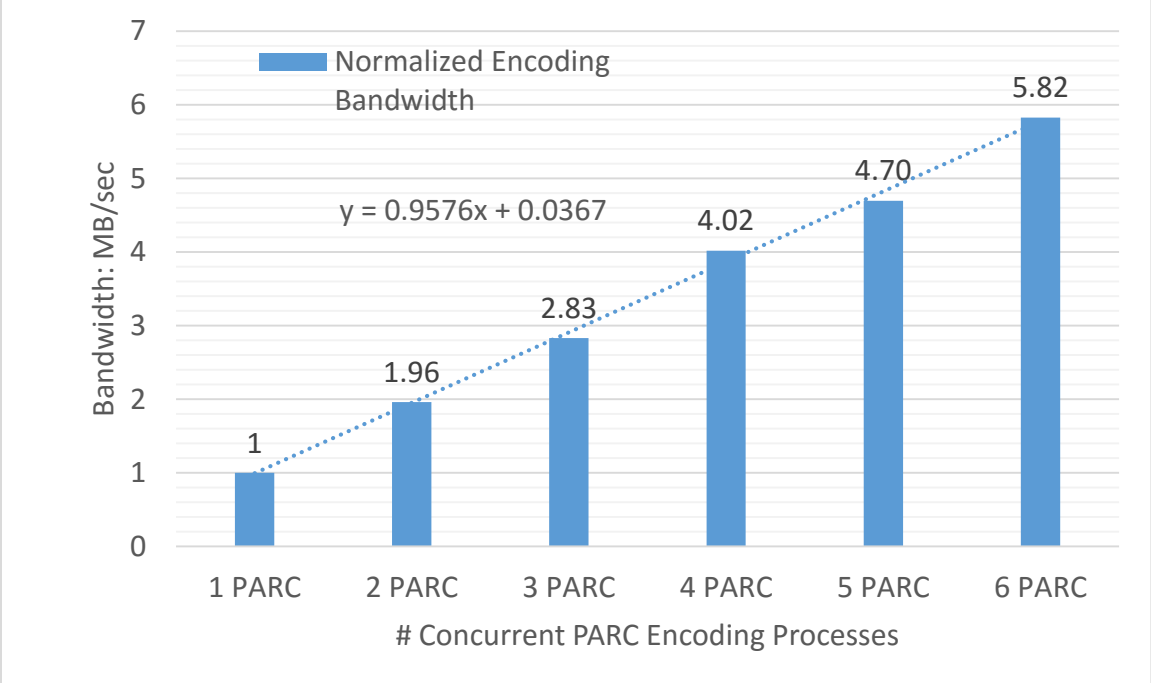

Figure 4C: Normalized encoding bandwidth for one to six parallelism tests (weak scaling)

5.2 Power Consumption Measurement and Energy Efficiency comparison

Energy consumption is a major concern with high-performance multicore systems. I also conducted power consumption tests on the proposed PARC software. I intended to study the energy consumption and performance (processing time) characteristics of parallel implementations of data encoding process vs. serial implementation of data encoding process.

Measuring the energy consumption of software components is a major building block for generating models that allow for energy-aware scheduling, accounting and budgeting. Current measurement techniques focus on coarse-grained measurements of application or system events. However, fine-grain adjustments in particular in the operating-system kernel and in application-level servers require power profiles at the level of a single software function.

Energy consumption constraints on computing systems are more important than ever. Maintenance costs for high performance systems are limiting the applicability of processing devices with large dissipation power. New solutions are needed to increase both the computation capability and the power efficiency. Moreover, energy efficient applications should balance performance vs. consumption. Therefore power data readings of components are important. This work presents the most remarkable alternatives to measure the power consumption of different types of computing systems, describing the advantages and limitations of available power measurement systems. Finally, a methodology is proposed to select the right power consumption measurement system taking into account precision of the measure, scalability and controllability of the acquisition system.

One “WattsUp/.net” [16] power meter equipment was used to collect power consumption sample during the erasure coding process. The “Watts Up/.net” power meter provides useful power consumption information such as current watts, minimum watts, maximum watts, power factor, cumulative watt hours, average monthly kilowatt hours, tier 2 kilowatt hour threshold, elapsed time, cumulative energy cost, average monthly cost, line volts, minimum volts, maximum volts, current amps, minimum amps, and maximum amps.

5.2.1 Power Consumption Measurement and Energy Efficiency Comparison of Strong Scaling Case

Table-1 shows the power consumption measuring items collected from the WattsUp meter. The Watts information and processing time is shown in Figure-5. I used the measuring results from the Table-1 and defined the performance metrics in Figure-6. I usde one 48GB file and six 8GB files on sequential and parallel encoding tests respectively. Parallel Read operation was used to support concurrent input when parallel encoding processing was applied.

Item Name	Item Description	Comment
W_i	Watts consumption on System Idle state	
W_s	Watts consumption on Sequential encoding	
W_p	Watts consumption on Parallel encoding	
T_0	Starting time for encoding process	
T_s	Finishing time for sequential encoding process	Encoding a 48 GB file using one sequential encoding process
T_p	Finishing time for parallel encoding process	Encoding a 6X8GB file set using 6 concurrent PARC process

Table-1: Power Consumption Metrics from the WattsUp power meter

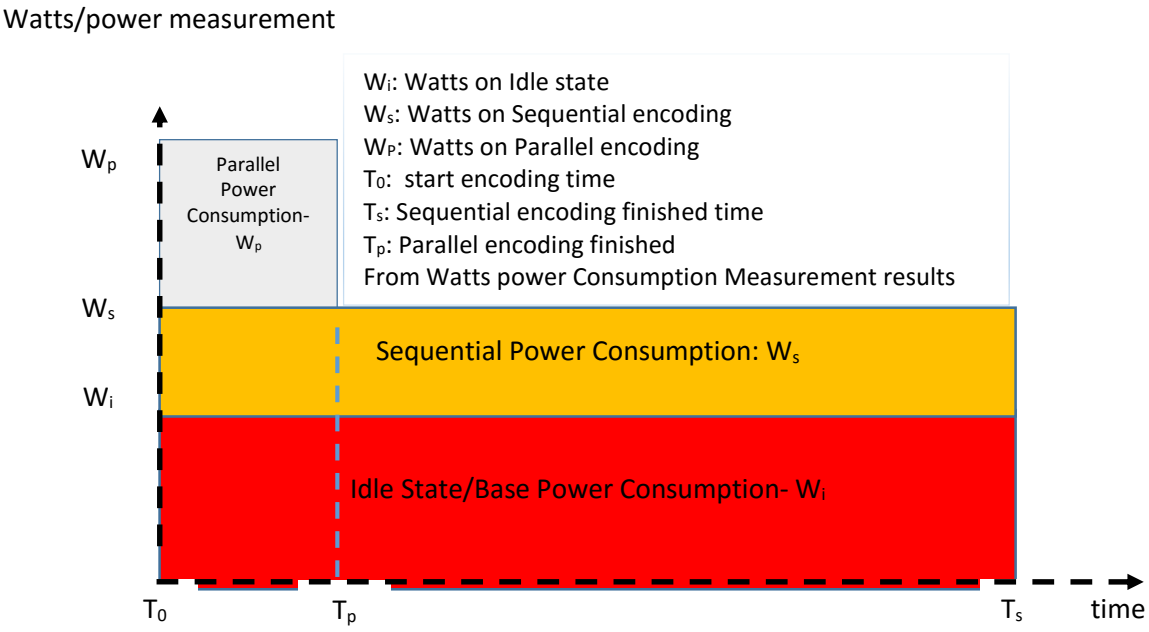

Figure-5: Power Measurement (Watts and processing time) for Sequential and Parallel Encoding

Data: encoding data size / 48GB data object

T: fixed processing time, one hour

$Data_{seq}$: total size of data processed by sequential encoding during the fixed processing time T

$Data_{par}$: total size of data processed by parallel encoding during the fixed processing time T

Energy cost calculation: total energy cost, kWh, baseline assumption is $0.01/kWh

$Energy_{par} = W_p * (T_p - T_0)$

$Energy_{seq} = W_s * (T_s - T_0)$

$Energy_{idle/par} = W_i * (T_p - T_0)$

$Energy_{idle/seq} = W_i * (T_s - T_0)$

Energy Cost: encoding only cost (sequential encoding and parallel encoding)

$EnergyDiff_{seq/idle} = (W_s - W_i) * (T_s - T_0)$

$EnergyDiff_{par/idle} = (W_p - W_i) * (T_p - T_0)$

Energy reduction ratio:

$EnergyReduction_{ratio} = (Energy_{seq}) / (Energy_{par})$

Idle State Energy ratio on Sequential and Parallel encoding

$IdleEnergy_{ratio/seq} = (Energy_{idle}) / (Energy_{seq})$

$IdleEnergy_{ratio/par} = (Energy_{idle}) / (Energy_{par})$

Encoding Unit Cost : baseline assumption is $0.01/kWh

$Cost_{seq} = (Energy_{seq}) / Data$

$Cost_{par} = (Energy_{par}) / Data$

Encoding bandwidth calculation

$BandWidth_{seq} = Data / (T_s - T_0)$

$BandWidth_{par} = Data / (T_p - T_0)$

$Data_{seq} = T * BandWidth_{seq}$

$Data_{par} = T * BandWidth_{par}$

Figure-6: Performance metric definition

Table-2 shows the energy efficiency comparison for sequential and parallel encoding processes. The PARC parallel encoding approach can significantly reduce about 81% energy cost when processing the same size of data. The PARC also can handle more data encoding workload during a fixed period of processing time. From encoding performance and energy efficiency studies, I strongly demonstrate the benefit of exploring potential task paralleling in data encoding problems and efficiently applying a parallel programming model on data encoding problem.

	Sequential Encoding	Parallel Encoding	Normalized Ratio
Energy consumption: 48GB data set	98.84	16.02	6.17: 1
Energy cost: 48 GB data set	0.9884 cents	0.1602 cents	6.17:1
Encoding Unit Cost: based on 48GB data	0.020591cents/GB	0.003337cents/GB	6.17:1
T=one hour, total encoded data size	297.22 GB	1803.60 GB	1: 6.07

Table-2: Energy cost and efficiency comparison – sequential vs. parallel encoding

5.2.2 Power Consumption Measurement and Energy Efficiency Comparison (One to Six Parallel Encoding Cases)

I then measured the weak scaling cases. For each N-way parallel coding test, I let each individual coding processes to handle 34GB of data. Figure-7 shows an example of Watts/Sec and time sequence information for 1,2,3,4,5, and 6 parallel coding test cases.

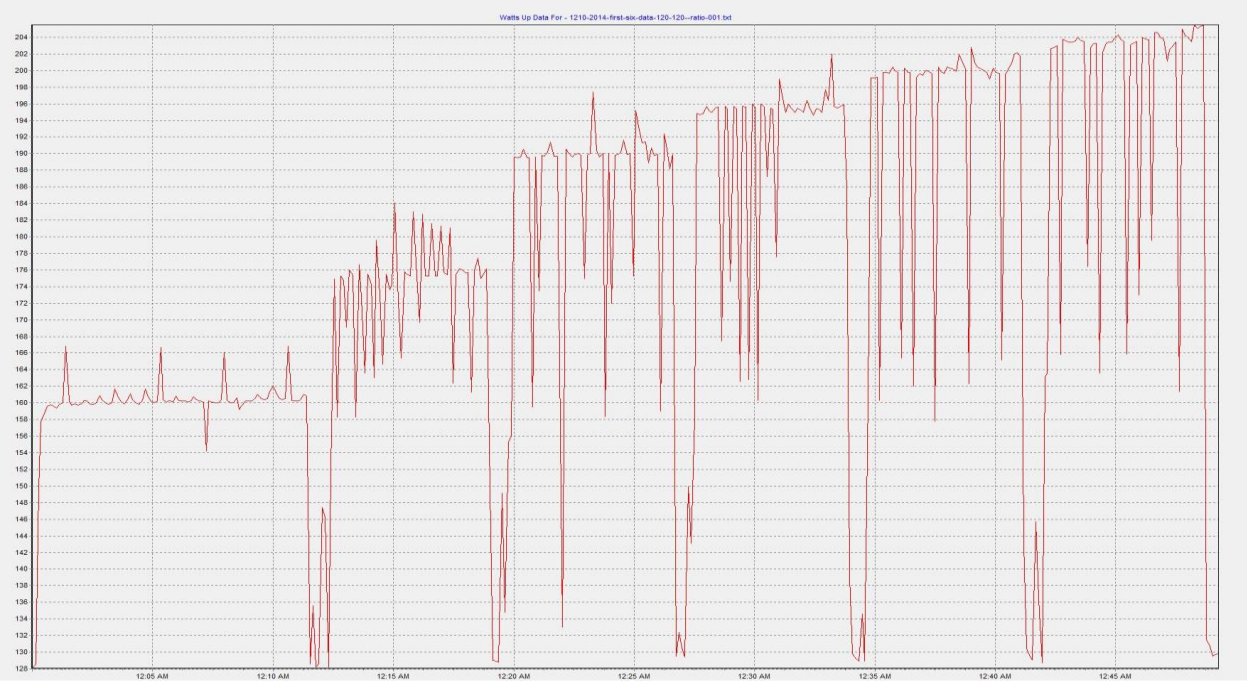

Figure-7: Graph of Watts Power Consumption vs. Time Sequence ((1,2,3,4,5,6) Parallel Encoding Processes)

I used the spread sheet data log from the WattsUP/.net power meter to create graphs displaying watts/sec and processing time sequence for each testing case (Figure-8a, Figure-8b, Figure-8c, Figure-8d, Figure-8e, Figure-8f).

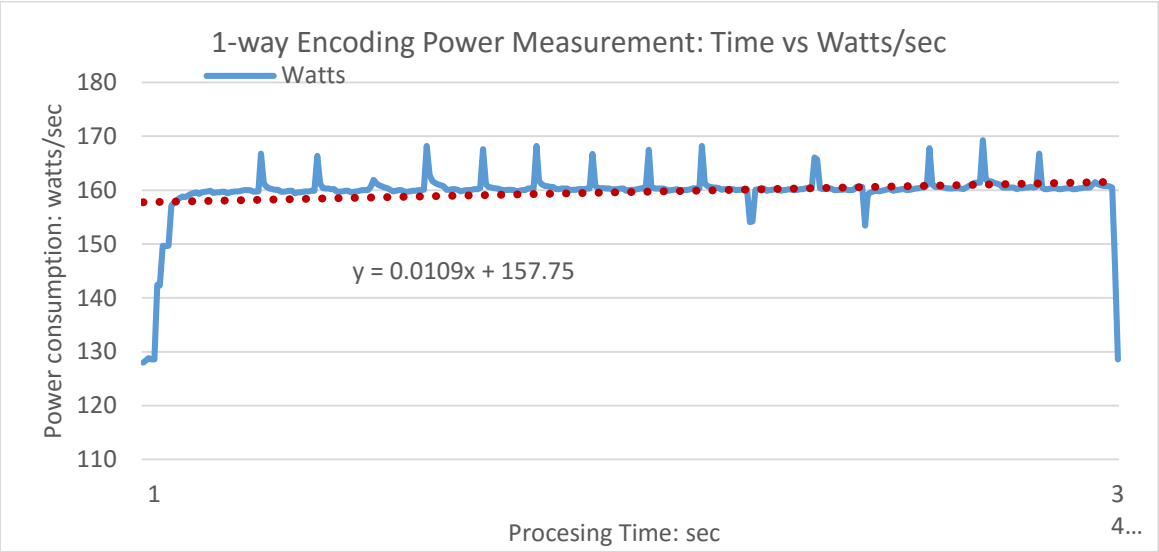

Figure-8a: 1 Way PARC Power Measurement

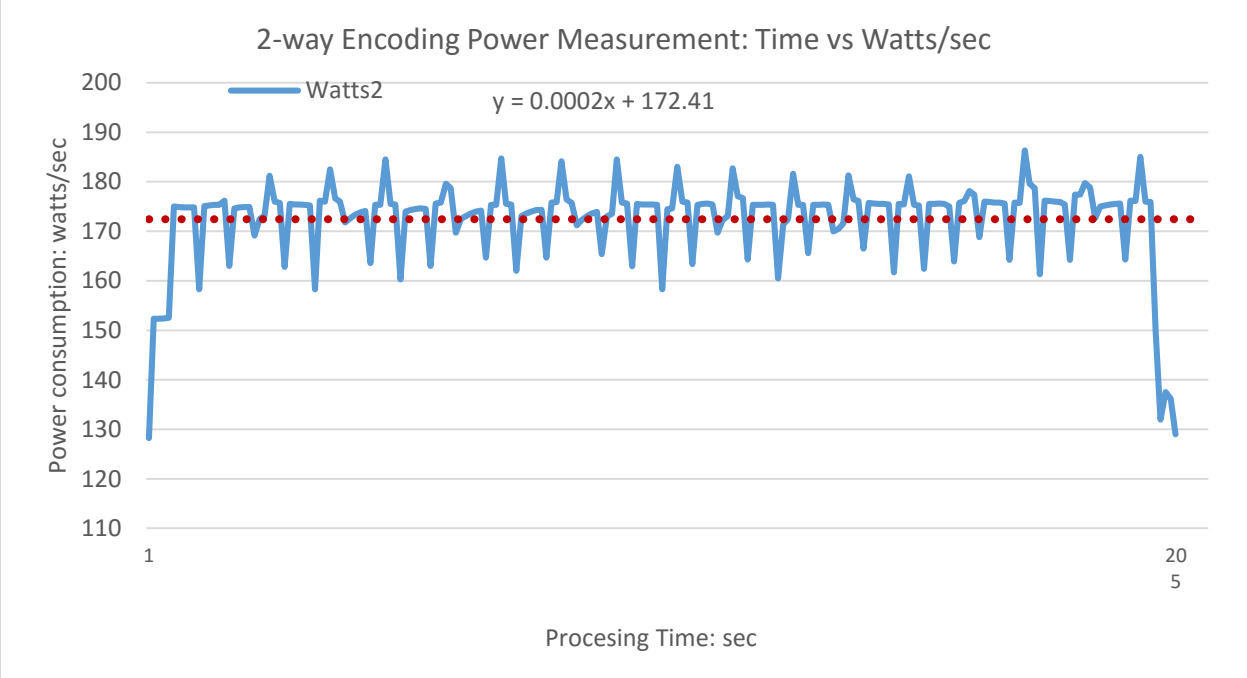

Figure-8b: 2 Way PARC Power Measurement

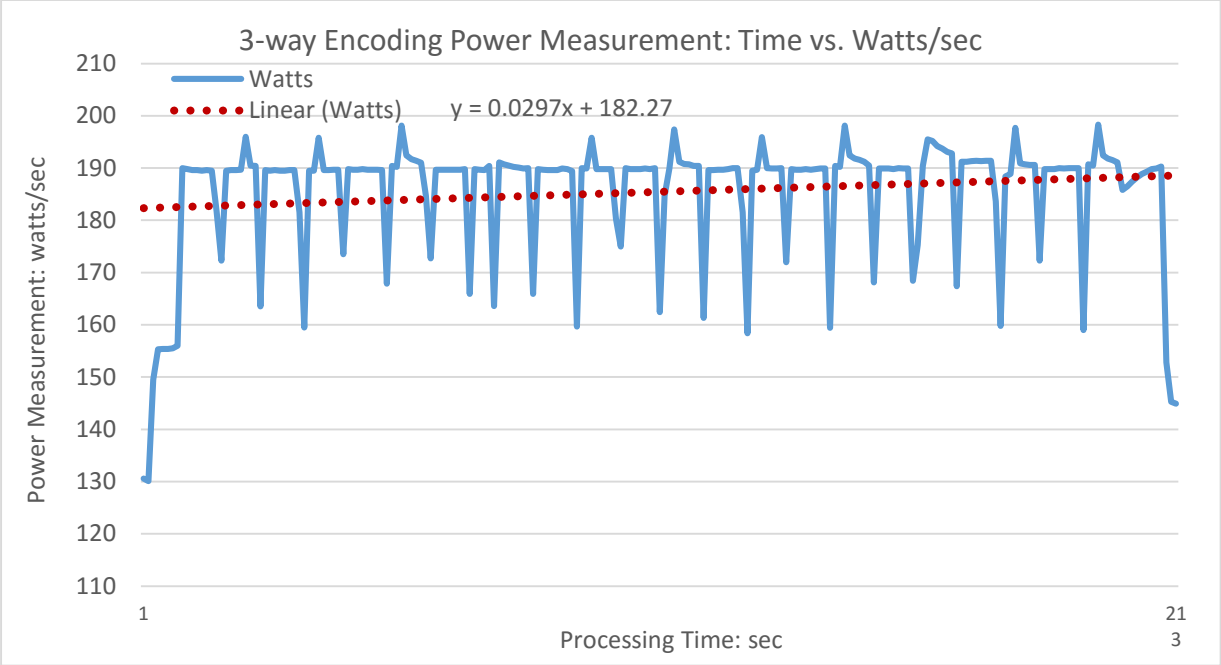

Figure-8c: 3 Way PARC Power Measurement

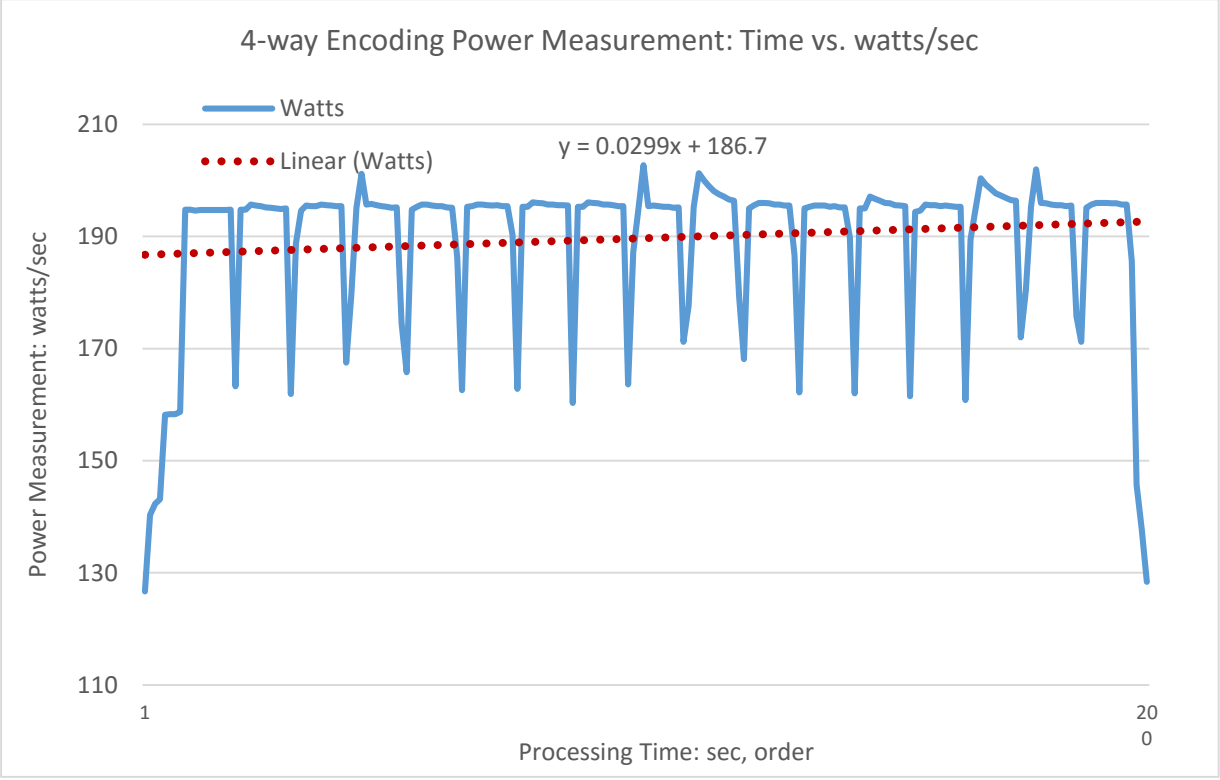

Figure-8d: 4 Way PARC Power Measurement

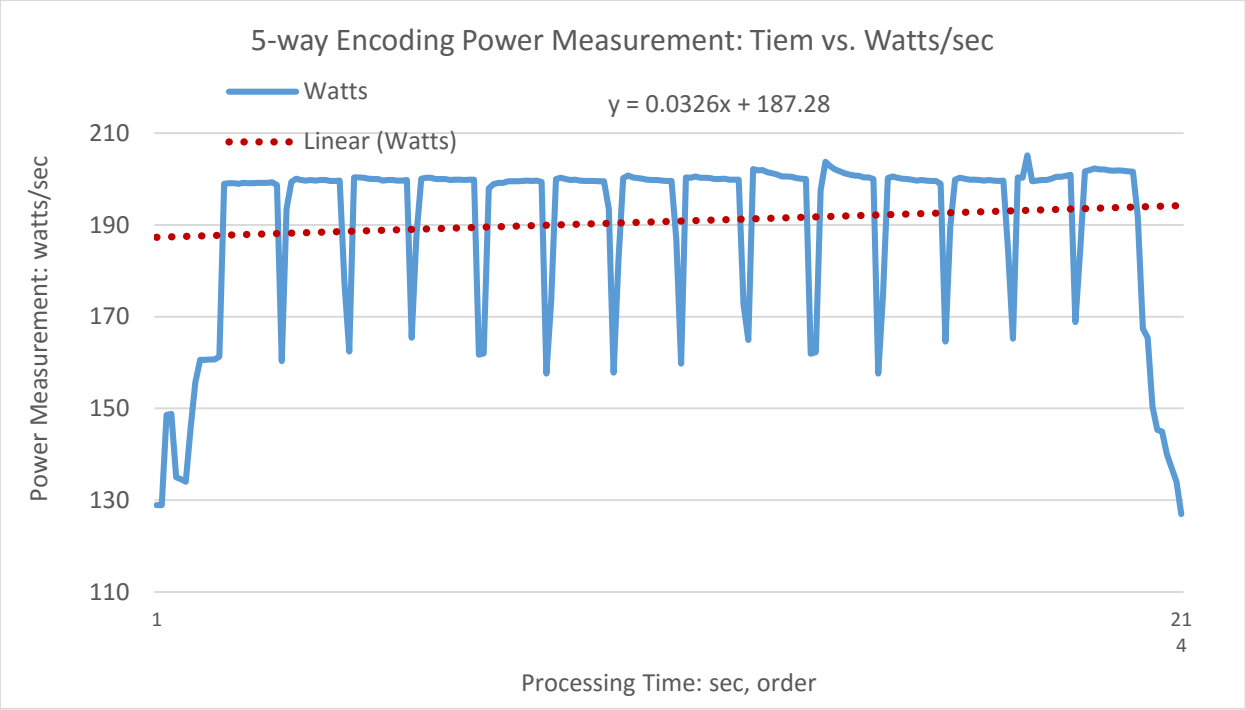

Figure-8e: 5 Way PARC Power Measurement

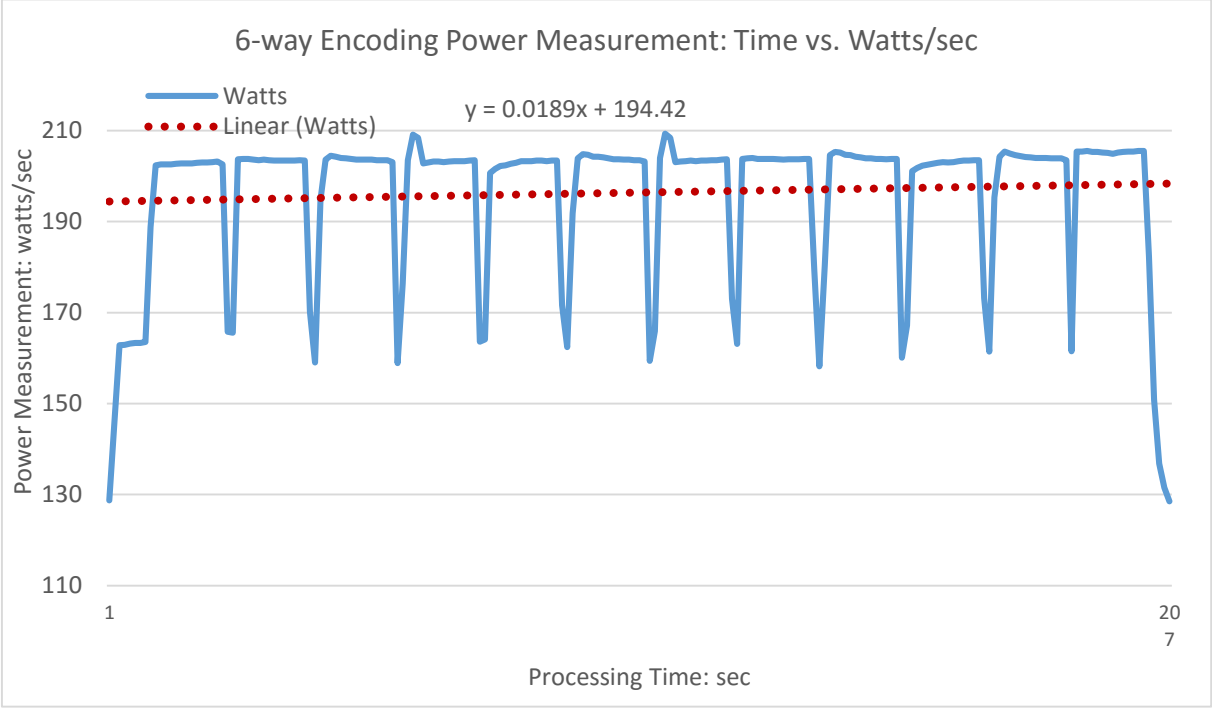

Figure-8f: 6 Way PARC Power Measurement

The power consumption trend line is shown for all graphs above.

- Definition of a Trend Line: A trend line, often referred to as a line of best fit, is a line that is used to represent the behavior of a set of data to determine if there is a certain pattern.

Using the Linear Trim line power consumption function for 1-way, 2-way, 3-way, 4-way, 5-way, and 6-way parallel stream encoding:

F_1-way(X) = 0.0109X + 157.75

F_2-way(X) = 0.0002X + 171.41

F_3-way(X) = 0.0297X + 182.27

F_4-way(X) = 0.0299X + 186.7

F_5-way(X) = 0.0326X + 187.28

F_6-way(X) = 0.0189X + 194.42

I used electricity consumption and energy bill calculation formula to calculate energy cost:

- Energy consumption calculation

The energy E in kilowatt-hours (kWh) per day is equal to the power P in watts (W) time number of usage hours per day t divided by 1000 watts per kilowatt:

$$E_{(kWh/day)} = P_{(W)} \times t_{(h/day)} / 1000_{(W/kW)}$$

- Electricity cost calculation

The electricity cost per day in dollars is equal to the energy consumption E in kWh per day times the energy cost of 1 kWh in cents/kWh divided by 100 cents per dollar:

$$Cost_{(\$/day)} = E_{(kWh/day)} \times Cost_{(cent/kWh)} / 100_{(cent/\$)}$$

In my testing cases, the WattsUP/.net meter generates power measurement data every second. There are two different ways to calculate energy consumption.

- Discrete event calculation: applied the measured watts/sec value for time t (Fm(t), m-way coding)

$EnergyConsumption_{discrete/m\text{-}way} = \sum_{i=t0}^{i=tn-1} Fm(time(i)) * time(i)$, the time(i) is set to one second. This is the minimal time result when using the Watts/.net power meter.

Fm(t) is the measured power consumption (from WattsUP/.net meter) at time t.

- Continuous event calculation: applied the trend line function (Ft)

$$\text{EnergyConsumption}_{\text{continue/x-way}} = \int_{t0}^{tn-1} Ft(t)dt$$

$\text{EnergyConsumption}_{\text{continue/1-way}}$ = X/20000 * (109X + 3155000)

$\text{EnergyConsumption}_{\text{continue/2-way}}$ = X/10000 *(X + 1714100)

$\text{EnergyConsumption}_{\text{continue/3-way}}$ = 11X/20000 * (27X + 331400)

$\text{EnergyConsumption}_{\text{continue/4-way}}$ = X/20000 * (299X +3734000)

$\text{EnergyConsumption}_{\text{continue/5-way}}$ = X/10000 * (163X + 1872800)

$\text{EnergyConsumption}_{\text{continue/6-way}}$ = X/20000 * (189X + 3888400)

Here I use 1-way, 2-way, 3-way, and 6-way coding test cases and compare the power consumption. The LCM of the number data sets for 1,2,3,6 way parallel coding is 6. I assume that I apply six "34GB" data sets on each coding approach.

Figure-9a, Figure9b, and Figure-9c show the Watts/sec vs time sequence data for the cases of "1-way vs. 2way", 1-way vs. 3-way" and 1-way vs. 6-way" respectively. The results have shown that parallel coding can significantly reduce the total processing time.

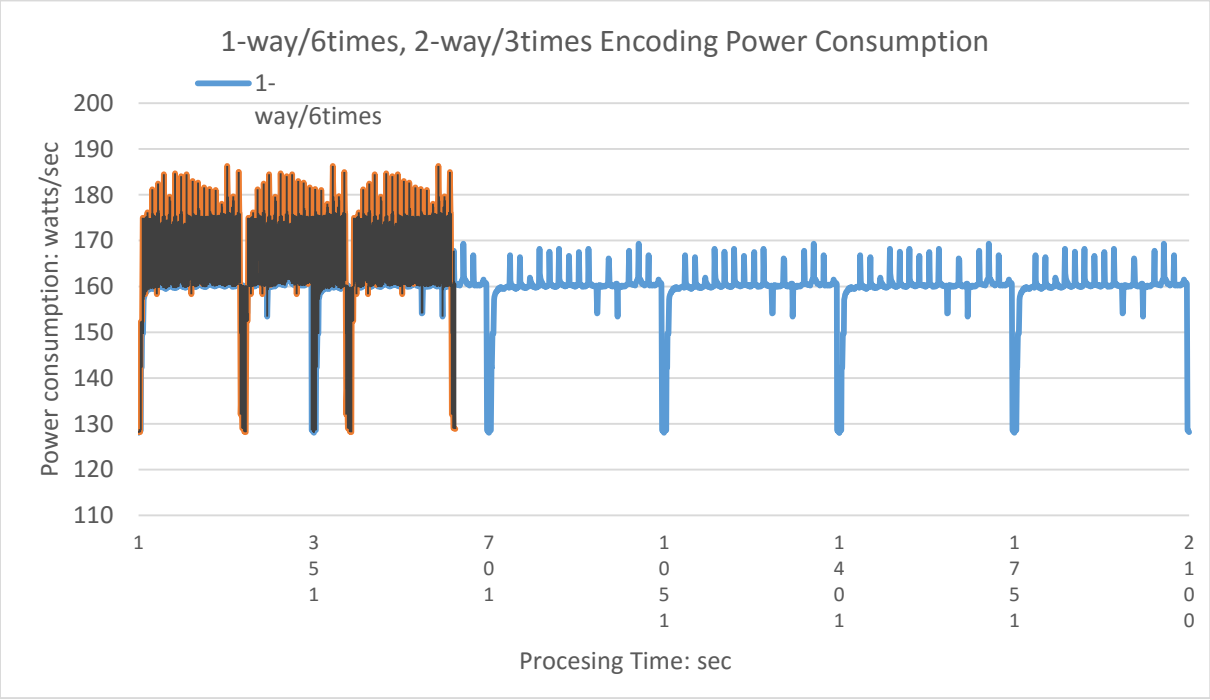

Figure-9a: Power consumption for processing six "34 GB" data sets -1-way vs. 2 way parallel coding

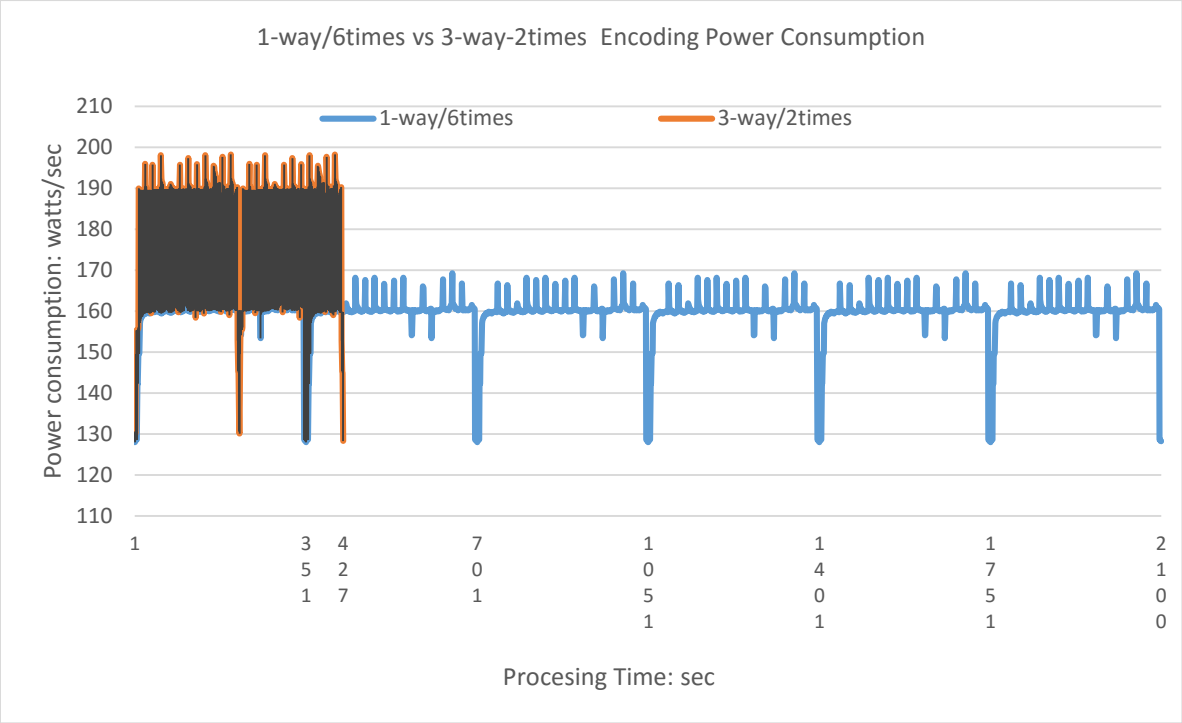

Figure-9b: Power consumption for processing six “34 GB” data sets -1 vs. 3 way parallel coding

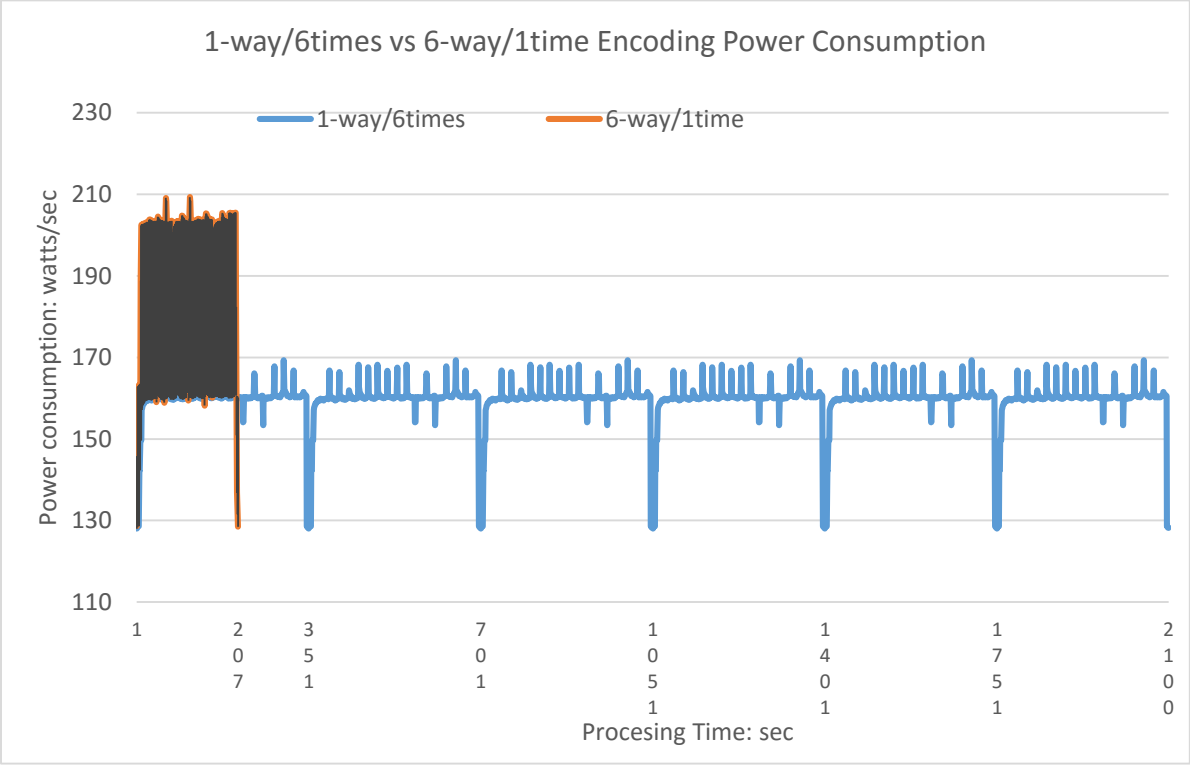

Figure-9c: Power consumption for processing six “34 GB” data sets -1 vs. 6 way parallel coding

Figure-10a shows the processing time for each coding method. Figure-10b shows the total power consumption for each coding method when using the continuous Energy Consumption formula and the discrete Energy Consumption formula. The results have shown that parallel coding can significantly reduce the total power consumption.

Three performance metrics are defined here. The base value is 1-Way coding's power consumption:

- ERI : Energy reduction index = Base/NWay
- ERP : Energy reduction % = 1 – (NWay/Base)
- PW = MB coding/Watts, performance per watts index [20]

Figure-10c shows the normalized Energy Prediction Index for each coding method. Figure-10d shows the normalized Energy Reduction Percentage for each coding method. EPI and ERP performance values have further demonstrated the benefit and advantage of applying parallel data processing multicore computing systems. Table-3 lists values of processing time, power consumption values, and ERI, and ERP performance values.

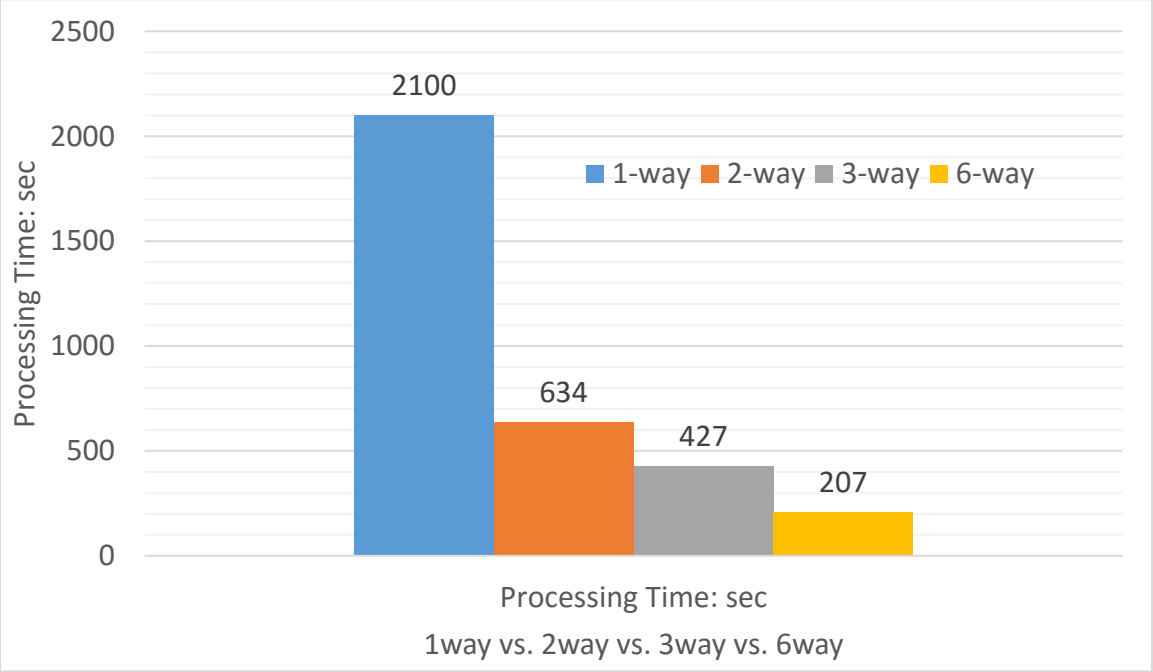

Figure-10a: Processing time comparison

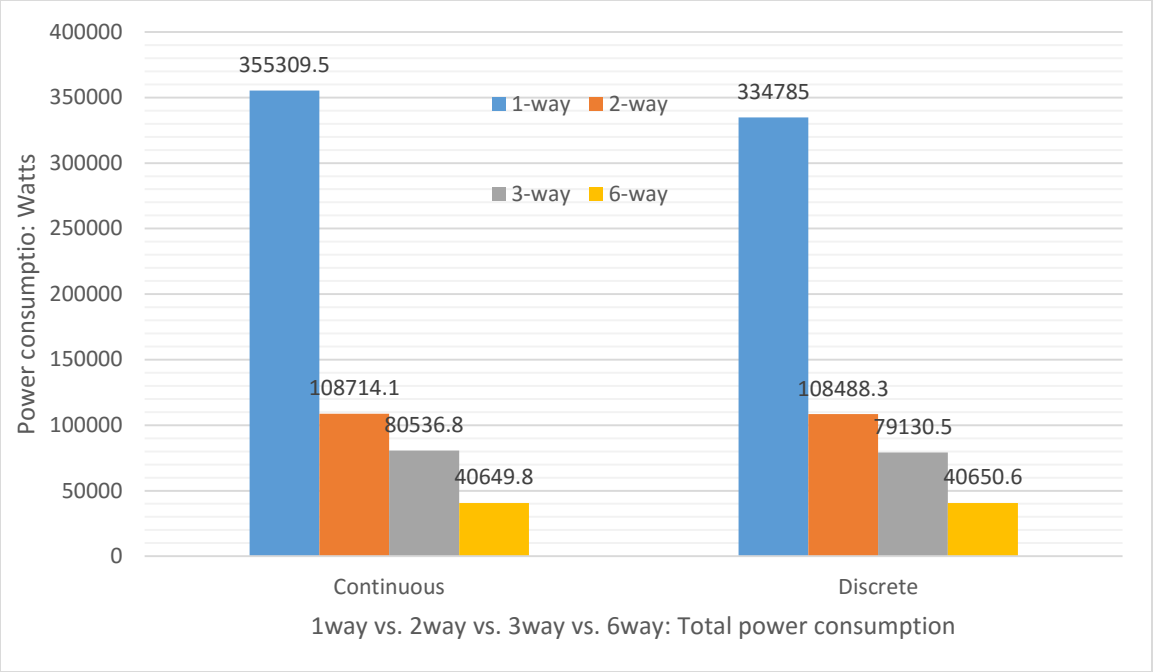

Figure-10b: Power consumption comparison

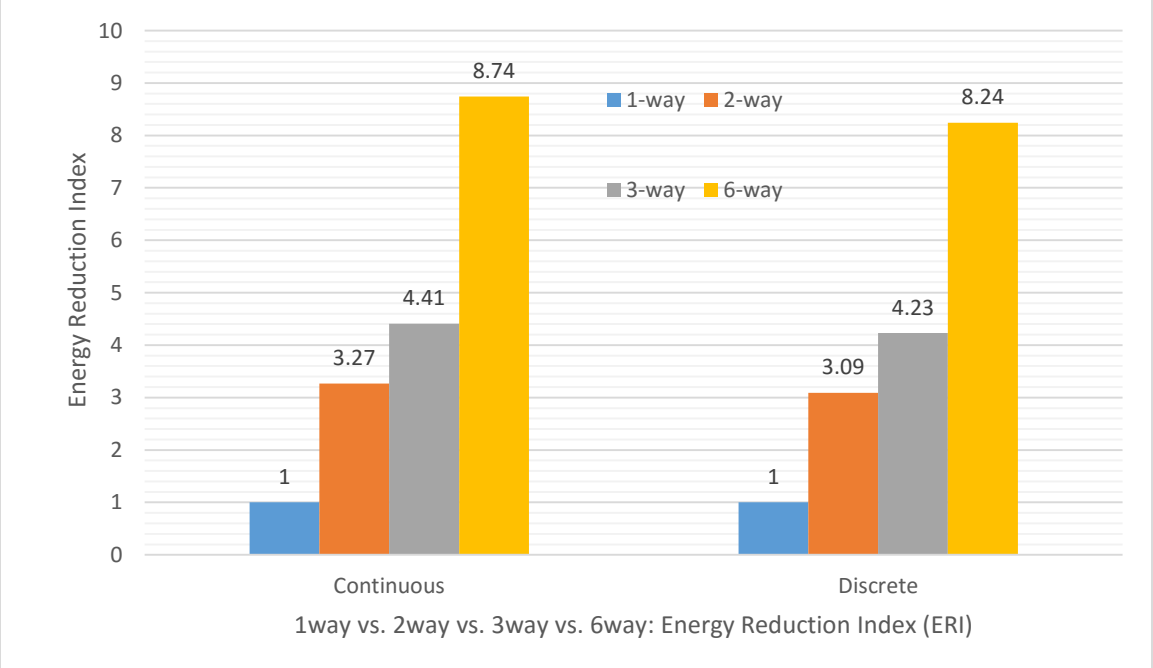

Figure-10c: EIR Energy Reduction Index Comparison

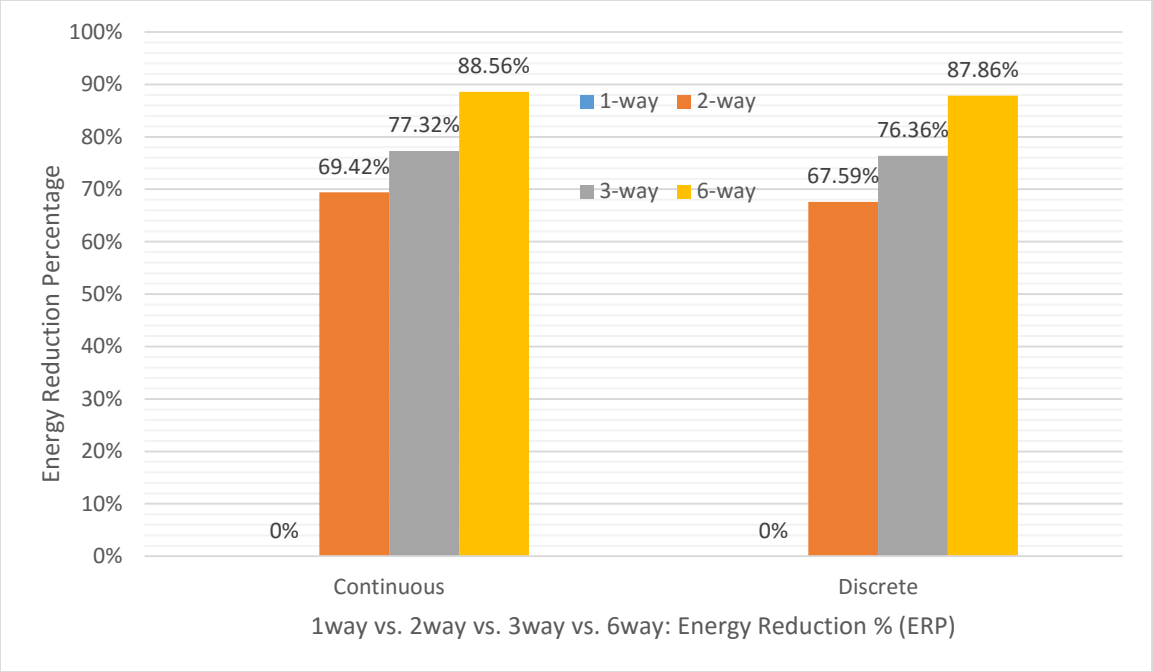

Figure-10d: ERP Energy Reduction Percentage Comparison

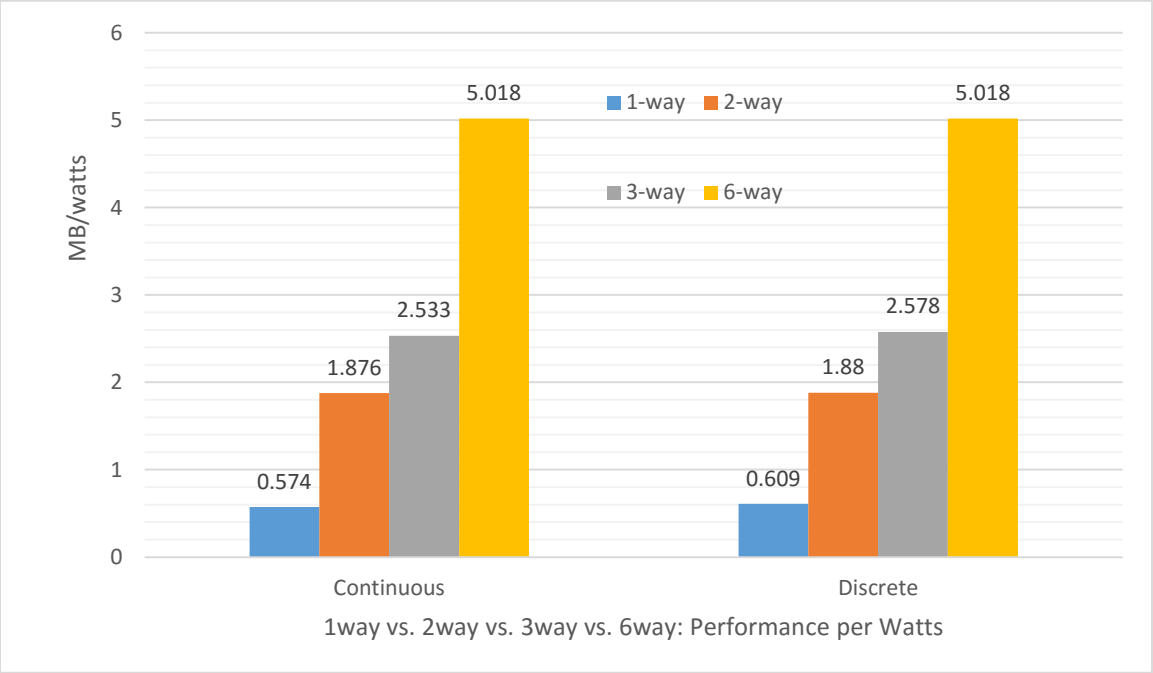

Figure-10e: PW Performance per watt comparison (MB/watt)

Power consumption (total watts)	Continuous	normalized	Discrete	normalized
$EnergyConsumption_{1\text{-}way}$	X/20000 * (109X + 3155000) X=2100, P=355309.5	ERI=1 ERP=0% PW=0.574	P=334785.0	ERI=1 ERP=0% PW=0.609
$EnergyConsumption_{2\text{-}way}$	X/10000 *(X + 1714100) X=634, P= 108714.1	ERI=3.27 ERP=69.42% PW=1.876	P=108488.3	ERI=3.09 ERP=67.59% PW=1.880
$EnergyConsumption_{3\text{-}way}$	11X/20000 * (27X + 331400) X=427, P= 80536.8	ERI=4.41 ERP=77.32% PW=2.533	P=79130.5	ERI=4.23 ERP=76.36% PW=2.578
$EnergyConsumption_{6\text{-}way}$	X/20000 * (189X + 3888400) X=207, P= 40649.8	ERI=8.74 ERP=88.56% PW=5.018	P=40650.6	ERI=8.24 ERP=87.86% PW=5.018

Table-3:

6 Conclusion and Future works

Multicore computing systems dominate the current commercial marketplace such as personal computers, enterprise computers, and supercomputers. Almost all computers have various parallel processing capabilities. To take maximum advantage of multicore chips, applications and systems should take advantage of built-in embedded parallelism. In this project, I studied parallel programming on multicore computing systems and investigated potential parallel processing capabilities using data encoding procedures. I implemented a parallel erasure coding software called PARC. I then conducted various workload tests on a single multicore compute node. I demonstrated the advantage of using parallel processing on multicore computing systems. The PARC software showed significant reductions in total encoding process time. Furthermore my results show that we can obtain almost linear-scaling bandwidth on single multicore machines and multiple multicore cluster based machines. I also demonstrated that the data coding problem on storage systems could be solved efficiently and effectively by task parallelism on multicore computing systems. My research project has shown that efficiently utilizing parallel computing capability of multicore system can improve performance per watts, reducing processing time, reduce poser consumption, and reduce energy cost. My performance testing cases have characterized the energy use of data encoding systems. This is the first step toward identifying opportunities for improvement.

Reference List

1. Keith D. Cooper, The Multicore Transformation: Making Effective Use of Multicore Systems: A Software Perspective, ACM Ubiquity, September 2014
2. Hardware RAID vs. Software RAID: Which Implementation is the Best for my Application? , Adaptec Storage Solutions White paper -
3. BackAbBlaze Reed-Solomon Code : https://www.backblaze.com/blog/reed-solomon/
4. Reed-Solomon-Code Code: https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction
5. NASUNI 2015 The State of Cloud Storage report, http://www6.nasuni.com/rs/nasuni/images/Nasuni-2015-State-of-Cloud-Storage-Report.pdf
6. MPI Programming Tutorial – Lawrence Livermore National Lab: https://computing.llnl.gov/tutorials/mpi/
7. An Introduction to MPI – Parallel Programming with Message Passing Interface, William Gropp, Ewing Lusk, Argonne National Lab http://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiintro/ppframe.htm
8. Rauber, Thomas, Rünger, Gudula, Parallel Programming for Multicore and Cluster Systems
9. Yuval Cassuto, Coding Techniques for Data-Storage Systems, PhD Thesis, California Institute of Technology, 2008
10. ZFEC - a fast erasure code which can be used with the command-line, C, Python, or Haskell, https://pypi.python.org/pypi/zfec
11. James Plank, "Erasure Codes for Storage Systems A Brief Primer", Usenix, Login 2013
12. Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) - https://software.intel.com/en-us/storage/ISA-L
13. Idilio Drago, Marco Mellia, Maurizio M. Munafò, Anna Sperotto, Ramin Sadre, and Aiko Pras, Inside Dropbox:Understanding Personal Cloud Storage Services, 2012 ACM Internet Management Conference
14. Hsing-bung Chen, An Innovative Parallel Cloud Storage System using OpenStack's Swift Object Store and Transformative Parallel I/O Approach, Los Alamos National Lab, 2013 LA-UR 13-20839
15. Tao Wang, Shihong Yao, Zhengquan Xu, Lian Xiong, Xin Gu, and Xiping Yang, An Effective Strategy for Improving Small File Problem in Distributed File System, 2015 2nd International Conference on Information Science and Control Engineering (ICISCE)
16. WattsUP .net A Power Measurement and Power Analyzer Meter, https://www.wattsupmeters.com
17. Steven Chen, 2016 Supercomputing Challenge Team #51 Final Report
18. Jame Plank, Jerasure: A Library in C/C++ Facilitating Erasure Coding for Storage Applications Version 1.2, Jerasure 1.2, opensource Erasure coding library, https://web.eecs.utk.edu/~plank/plank/papers/CS-08-627.html
19. Measuring_Parallel_Scaling_Performance, https://www.sharcnet.ca/help/index.php/Measuring_Parallel_Scaling_Performance

20. Performance per Watts: https://en.wikipedia.org/wiki/Performance_per_watt