Program Language Design of MapReduce and its influence on Supercomputing

Oliver Collins
University of Colorado, Boulder
oliver.collins@colorado.edu

1. Abstract

Over the years, principles in supercomputing have revolutionized everyday tasks, from powering powerful AI algorithms that drive our day-to-day lives, to solving and proving complex mathematical theories and proofs. While the concept supercomputing has stayed relatively the same, the backbone of supercomputing has evolved substantially over the past few decades.

In this paper, I discuss how new programming paradigms, specifically Hadoop MapReduce, push forward innovations into supercomputing. The motivation for this paper is to have a greater understanding of the evolution of supercomputing principles and the ways we have advanced into more reliable and dependable technologies. Hopefully, we will develop more suitable solutions to advance large scale computing further, but learning about current tech is quintessential to progressing into the future.

2. Introduction

Supercomputing is becoming more of a day-to-day phenomenon. Companies and governments around the world are competing to have the latest and fastest supercomputer. Nonetheless, what is supercomputing, how are these companies devising higher-speed computers, and why does this all matter? These are some of the questions I am hoping to answer in this paper, but before we dive in, it is a good idea to get a solid understanding of the key topics at hand.

2.1 Supercomputing

The term "supercomputer" refers to computing systems (hardware, systems software, and applications software) that provide close to the best currently achievable sustained performance on demanding computational problems [1]. In essence, as computers become more powerful and sophisticated, only the latest and most recently powerful machines can be classified as "supercomputers." All older "supercomputers" lose their designation and become conventional, yet comprehensive, computers.

What is essential to know about supercomputing is the capability it has to compute many of our complex algorithms in a relatively short time. For example, many of the weather forecasting systems use complex and power-intensive algorithms which conventional computers do not have the necessary hardware, or even software as I have investigated, to compute. Supercomputers have, in turn, substantially more hardware capabilities and can, therefore, run these algorithms in a relatively small timeframe.

However, one of the main drawbacks of supercomputers is their sheer size. Many of these supercomputers range from a couple of refrigerator-sized computers while some are the size of buildings. The size alone is enough to prevent the majority of households from obtaining these machines. Another drawback is that these top-of-the-line machines are costly and out of reach for many people and even companies. However, as technology progresses, the size-to-performance ratio decreases, and the cost-per-performance decreases over time. So what might have been an extremely expensive supercomputer back in the day, the phones in our pocket, which cost a fraction of the price of these older supercomputers also has significantly more capabilities and performance.

2.2 MapReduce

Before the advent of MapReduce, we used to store terabytes of data on single servers. Accessing this quantity of data was remarkably slow and wildly expensive; additionally, these machines lacked the resilience that we take for granted today. This lack of resilience was in part because it had not been seen as a critical issue, but also because no one could adequately address the issues.

Then came around Jeffrey Dean and Sanjay Ghemawat in 2004 whom both developed MapReduce; a programming paradigm they hoped would come about to solve these issues stated above. MapReduce enables massive scalability across hundreds or thousands of servers in a Hadoop cluster, a particular type of cluster explicitly used for storing and analyzing large amounts of unstructured data[2]. The key benefits of MapReduce are:

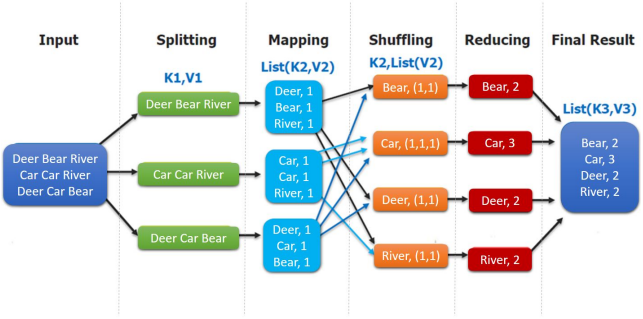
1. *Scalability*: Businesses can process petabytes of data stored in the Hadoop Distributed File System (HDFS).

2. *Flexibility*: Hadoop enables easier access to multiple sources of data and multiple types of data.

3. *Speed:* With parallel processing and minimal data movement, Hadoop offers fast processing of massive amounts of data.

4. *Simple:* Developers can write code in a choice of languages, including Java, C++ and Python.

MapReduce works in a two-step process. The first step is mapping a set of input data into another set of data. This process is done by breaking apart the data into individual elements and then grouping them into tuples of key-value pairs. The next step is reducing the tuples into smaller subsets of tuples bringing together a collection of mapped values. This mapped group of values is what the user, in the end, sees and care about, an organized subset of data points that represent what the data means. A diagram below shows a good representation of the MapReduce process.



MapReduce Word Count Process

2.3 Parallelism

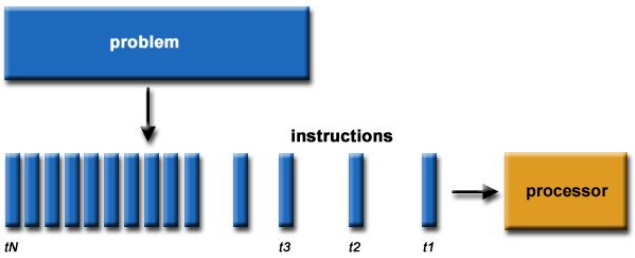
Parallel computing is one of the most important innovations in computer science as it allows computers to process and handle computations in parallel and small pieces rather than serially, in order. Parallelism is essential in computing, especially in large-scale computing because it allows computers to split tasks, which in turn, speeds up the process of computing tasks. While on smaller tasks, such as running small programs on your local machine parallelism might not have as much of an impact, but with applications of massive scale, parallelism can save from a couple of hundred dollars to several million depending on the capacity of the company in terms of cloud-compute time as well as valuable developer time.

Concerning supercomputing, most supercomputers are parallelized clusters and networks of computers which use shared memory space. Many tasks in the world, including many complex events, are coinciding. A few examples of natural events that happen in parallel include weather patterns, biological evolution, and the expansion of the universe. Parallelism works most efficiently in a similar manner, mostly for modeling, and very resource-intensive computations [7].

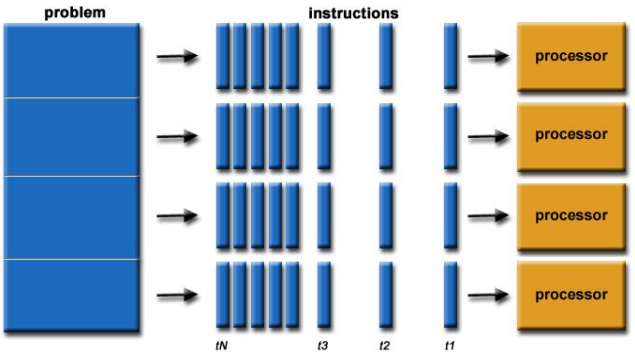
However, while parallelism might seem favorable over serialized coding, it is crucial to recognize that parallelism does have its flaws. Some of the drawbacks to using parallelism is that it is inherently non-deterministic, introduces new problems such as deadlock and race conditions, it is challenging to

debug, as well as it is hard to port between parallel architectures. Nevertheless, the pros heavily outweigh the cons, which is why the majority of enterprise and even small scale computing shops are adopting the likes of parallelism to help in their day-to-day activity.

Below is an example of how serial computation works:



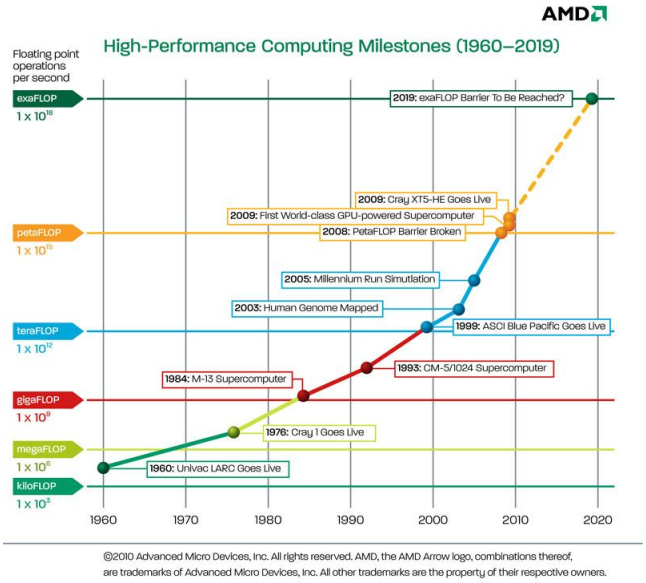
With the below example presenting how parallel processing works.



3. Rise of Supercomputing

In 1962, Seymour Cray created the Control Data Corporation (CDC) 6600, the world's first supercomputer. The CDC 6600 had only one CPU, operated at 40MHz, and peaked at about 3 million floating-point operations per second [3]. Since then, we are now reaching the point where a supercomputer can perform in the range of exaflops.

Below is a graph that illustrates the exponential growth of speeds of supercomputers.



Innovations in chip design and new and faster algorithms help spur this exponential growth, but also from programming paradigms such as MapReduce. However, the reason the race for supercomputers is strong is because of the amount we rely on these machines. Bitcoin mining is becoming more prevalent than ever, with miners needing faster and more efficient computers. Self-driving cars which process millions of events per second need an array of computers and even small versions of supercomputers mounted on their cars. Simulations of the aerodynamics of cars and planes during the manufacturing process are also powered by extraordinarily powerful machines which can be improved every day. Thus the societal need for faster computers is stronger than ever, which is why supercomputers show no sign of slowing down in terms of research.

MapReduce is one of the leading innovations that pushed not only supercomputing but other fields in computer science one step further. One of the main reasons MapReduce differentiates itself from other software solutions is that it is automatically parallelized and executed on a large cluster of commodity machines [4]. Also, since the run-time system handles machine failures, inter-machine communication, and partitioning the input data, it makes ease of use, especially for programmers

inexperienced with parallel and distributed systems a breeze to manage large-scale distributed systems. MapReduce is also a relatively easy concept to understand with only a few underlying concepts needed for someone to thoroughly understand the paradigm.

3.1 Prior to MapReduce

When the supercomputer first came out, its specs were paltry to what we see today. Supercomputing was more of a hobby rather than a means to compute vast amounts of calculations and run algorithms. It wasn't until the mid-90s and early 2000s where supercomputers become vital to our day-to-day lives. When supercomputers started to become necessary, the need to research and create new technologies was much needed.

Before MapReduce, computer scientists had to live with the fact that indexing data or computing calculations would take massive amounts of time and adding n amount more data would require a linear increase in performance. It was not out of the ordinary to see terabytes of data stored on a single server. Analyzing data in this fashion today seems unimaginably time-consuming and unreasonable as well. However, for people back then, that was the only solution they had, and so they managed to deal with what they had.

3.2 MapReduce Program Design

Though MapReduce is often confused with being a programming language, it is a programming model explicitly used to perform distributed and parallel processing. Apache Hadoop, which is often the tool used most often to develop with MapReduce, is written in Java; however, you can write MapReduce in any language, be it Java, C++, or Python. However, merely because MapReduce is not a programming language does not mean that it does not have a language design. Since the majority of the intermediary steps of MapReduce are customizable, it is best to follow best practices of the paradigm.

3.2.1 Mapping

The first significant principle in MapReduce is the concept of mapping and filter data into smaller subsets of data. Functional programming mapping and MapReduce mapping are two entirely different concepts. Inside MapReduce's mapping, the mapping serves as the argument for the map rather than the application of the map combinator or the combinator itself [8]. Within mapping, several tasks take place to ensure consistency regardless of programming language or system structure. These tasks consist of a record reading step, the use of a mapper, a combiner, and a partitioner.

The record reading phase parses the input data into separate records. It then forwards the data to the mapper in the form of records.

Once the record reader passes on the data, the mapper, which is user-provided code, is used to produce intermediate pairs or new key/value pairs. The mapper is essential to understand when it comes to computing these key/value pairs as the semantics of these pairs are what differentiate MapReduce from other programming paradigms.

Within the mapper is an optional combiner which aggregates and groups data. The main reason developers use combiners is that it reduces the total amount of data sent over the network, which therefore speeds up the entire process. This reduction in data transfer is due to the reduction in repetition and uses the value within a key/value pair to track necessary values.

The last step within mapping is partitioning the data. The partitioner is also a straightforward concept which takes the intermediate pairs which the mapper produced and splits them into shards, one per reducer.

The main reason these mapping phases occur is to load, parse, transform, and then filter the data into subsets of smaller reduce data. Once we complete these tasks, the data is grouped further by passing onto the reducing phases.

3.2.2 Reducing

The second primary programming principle under the MapReduce paradigm is to reduce and split the data into cleanly formatted output. Another confusion with MapReduce's reduce methodology is that it works the same as reduce in functional programming. This belief is not the case as reduce in MapReduce does not act as the reduce combinator or as the argument of reduce. However, MapReduce's reduce does serve as the argument for map, similarly to how MapReduce's map works, and also typically acts as an application of the reduce combinator. Naturally, both map and reduce functions are applied iteratively to each key/value pair, which is often why both functions get confused with the map and reduce functional programming languages utilize. This reduction task is again broken up into a set of phases. The four phases include shuffling the mapped data values, sorting the sharded values, running a reduction function, and finally formatting the output.

Shuffle and sorting are the first of the steps within the reduction tasks. Within shuffle and sorting, all the output files from the mapping tasks are downloaded onto the Hadoop instance where a reducer function is run to break up the individual data pieces based on their key value onto a more extensive list of data. The reason we sort the data in this specific order is so that when we access the data later, their values can be quickly iterated over.

Following this shuffle and sort step, the reducer runs a reduce function grabs the key from the set of values and uses it as an iterator to keep track of all the data values found within the passed set of values. This step is one of the more customizable pieces of the MapReduce paradigm as you're allowed to aggregate, filter, and combine the data in several ways, based on the developer's needs. This reduce step is also one of the most critical elements of the MapReduce paradigm as this step differentiates in terms of speed and efficiency over other software solutions.

The final step of MapReduce is to format the results through a process called output format. The output format phase separates the key and value based on the rules of the programmer and can be customized, similarly to the sort step, based on the developer's preferences.

3.2.3 MapReduce Abstraction

Abstraction is one of the core principles of Object-Oriented Programming, and so while you do not have to write MapReduce in an Object-Oriented Programming language, it is best thought to have some layer of abstraction to simplify communication between developers.

When any programming language deals levels of abstraction, the developer has to be cautious to write code that is neither too abstract and broad nor too complex. The best practices of MapReduce tells us to practice a level of abstraction where many standard key components and many programmers are familiar with are concealed behind a level of abstraction. The developer also has to be careful that the code they develop is not so broad that implementing code is too vague given the current codebase. This hiding of complexity helps transfer knowledge between developers much more effectively rather than having to explain every bit and piece of code. It is easier to tell someone that this code should make use of a reduce-side join rather than a map-side join is much easier than explaining the exact code changes that need to be made to that particular service [5].

However, because MapReduce practices a level abstraction, that does not mean it generalizes every construct of the paradigm. This concept is crucial to understand since code written too broad would negatively affect team performance. The main reason MapReduce has this high level of abstraction is that there is a fast adoption of this new style of programming and adding higher levels of abstraction makes it more attractive and more manageable for new developers to understand and grasp the language.

3.2.4 Fault Tolerance

Fault tolerance or resilence is the property of a program or a software system to continue operation even in the event of a failure. Fault tolerance within MapReduce was introduced with Apache Hadoop

version 0.21 and is now fully included with every version since.

Hadoop enables resilience by using the JobTracker to read off heartbeats from the TaskTracker. The heartbeat is a signal that is generated regularly to indicate that something is operating correctly. In the case of MapReduce, for example, whenever the JobTracker does not receive a heartbeat from a TaskTracker within a set amount of time (default is set to 10 minutes), the JobTracker assumes the worker for the TaskTracker has failed. When this occurs, the JobTracker reschedules all related tasks, both pending and in-progress tasks, to another TaskTracker.

This checking between the two systems is not uncommon within MapReduce, and the same procedure happens between the NameNode and the Data Node to make sure data leakage is kept to a minimum. Resilience is crucial as it allows damage from failovers to be as small as possible, and the process to fix any damages to be reduced to a minimum.

3.2.5 MapReduce and Hadoop

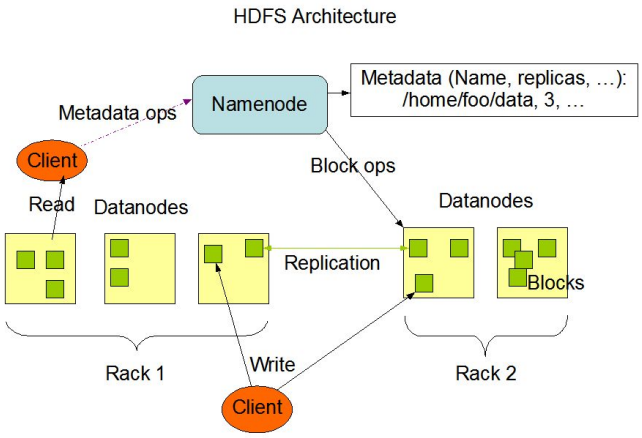
Apache Hadoop is a set of open-source software utilities that utilize networks of computers to solve complex big data problems. Often called Hadoop, it provides a software framework that derives from the principles of MapReduce. Hadoop has similar advantages to MapReduce in that it can store large amounts of data quickly (usually in the terabytes, petabytes, or higher), resilience, low cost, scalability, and computing power. Hadoop is one of the fastest-growing skills in the workforce and is becoming an even more significant role in companies and governments where data is far exceeding petabytes.

3.2.5.1 HDFS

Within Hadoop, it is crucial also to understand the Hadoop distributed file system (HDFS), which is used for storing data that MapReduce later processes. There are five essential services that HDFS provides:

1. Name Node
2. Secondary Name Node
3. Data Node
4. Job Tracker
5. Task Tracker

Similarly to MapReduce, the Job Tracker and the Task Tracker are responsible for managing and processing incoming data requests. Specifically, the Job Tracker's main job is to farm out MapReduce tasks to specific nodes in the cluster while the Task Tracker tracks the execution of MapReduce workloads happening on its slave node sends the status to the Job Tracker. The NameNode and Secondary NameNode are both responsible for tracking and checkpointing different points of the file system as well as track and manage the file system. The NameNode focuses more on storing the metadata of HDFS and tracks all files across the cluster, while the Secondary Node focuses just on checkpointing files. Lastly, the Data Node acts as a slave node that both sends a heartbeat back to the NameNode to notify that it is still alive as well as storing the actual data in blocks for the client to read and write to later.
The diagram below shows a good representation of the entire HDFS architecture.



While HDFS is optimized for extremely large files, streaming data access, and commodity hardware it fails for a lot of very small files, multiple writes or arbitrary file modifications, and with low latency access [9].

4. MapReduce's Influence on Computing

As the amount of data the world processes increases exponentially over the past several decades, the push for better technologies to handle big data increases. MapReduce helped fill a void within supercomputing. MapReduce reintroduced parallelism into the game of supercomputing, which massively increased the efficiency of these machines.

For example, due to the increasing performance of supercomputers, from innovations such as MapReduce and hardware changes, the office of Energy Efficiency & Renewable Energy believes that supercomputing is one of the most critical U.S. strategic assets and will forever change the way we analyze weather patterns, studying the evolution of the universe, and determining optimal strategies for deploying and managing energy [14]. More importantly, the Secretary of Energy, Rick Perry, states that supercomputers, especially recent and the most powerful supercomputers are essential to asserting the dominance of high-power computing in the U.S. as well as helping in the fields of medicine and health [15].

5. Analysis on MapReduce (Stretch Goal)

For the stretch goal, I decided to compare MapReduce to some prior technology that was used to compute highly complex tasks to simulate supercomputing speeds. I decided to use AWS EMR cloud-native application to quickly deploy clusters and not have to worry about provisioning nodes, infrastructure setup, Hadoop configurations, or cluster tuning.

The process for using AWS EMR is relatively simple; the diagram below illustrates the entire process.

The dataset I decided to use is large enough that it took over 10 minutes to download on my local machine (52943 rows), yet wasn't small enough where the discrepancy between execution time wasn't noticeable.

For the "prior technology," I decided to compare MapReduce against an SQLite instance on my local machine and run a simple query to select all rows and columns from the dataset. I performed the query several times to make sure the findings were as accurate as possible.

I decided to use 1 master m3.xlarge node with 2 m3.xlarge core nodes with the Amazon 2.8.5 Hadoop distribution installed. I also included Ganglia 3.7.2, Hive 2.3.5, Hue 4.4.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.2 with the entire cluster hosted on AWS's us-west-1 region. The m3.xlarge nodes came out-of-the-box with a 64-bit processor arch, 4 vCPUs, 15 GiB of memory, 2x40Gb of instance storage, with high network performance. I chose this architecture through AWS both because it was the cheapest available ($0.07/hr), but also because those most closely match that of my local machine which kept the test results as accurate as possible.

Once I created the EMR instance, I would then download the dataset onto the cluster and then move the data into my the HDFS. From there, I would run my query and test the results there.

In conclusion, after downloading both the exact dataset into HDFS and onto my local instance, running hive to perform the query took approximately 7x quicker than it did to perform the same operation on my local machine. Here is a sample result from my local instance.

```
Run Time: real 0.830 user 0.298686 sys 0.106727
```

And here is a result from my AWS EMR instance:

```
Time taken: 0.179 seconds, Fetched: 53944 row(s)
```

Using Hive, a subdomain of Apache Hadoop, took an average of 0.167 seconds per query execution, while with a local solution it would take roughly 1.229 seconds.

MapReduce is the apparent winner which explains the quick adoption for the technology and the need for different solutions. It there was a 7x speedup every time a new technology was created to solve these issues, the issues revolving around big data would no longer be an issue.

6. Conclusion

As the demand for faster and more efficient supercomputers and computers, in general, is strong, the research into designing new programming patterns and hardware breakthroughs will continue to be more impressive than ever.

New features will heavily influence future design patterns for MapReduce in Hive and Pig, tools used primarily for mangling with the HDFS. Hopefully, though, new technologies arise that are better designed and adopted to MapReduce, which will further improve large data computations and take supercomputing to a new level.

It is entirely unclear when supercomputing will reach a performance peak, but hopefully, with this intense research taking place around the world, there will be a fairly substantial time before we see any slowdown in the performance and efficiency of these complex machines.

7. References

[1] "Explanation of Supercomputing." The National Academies of Sciences, Engineering, Medicine, https://www.nap.edu/read/11148/chapter/4

[2] "What is Apache MapReduce." IBM, https://www.ibm.com/analytics/hadoop/mapreduce

[3] "The history of supercomputers." ExtremeTech, https://www.extremetech.com/extreme/125271-the-history-of-supercomputers

[4] "MapReduce: Simplified Data Processing on Large Clusters." Google AI, https://ai.google/research/pubs/pub62

[5] "Design Patterns and MapReduce", O'Reilly, https://www.oreilly.com/library/view/mapreduce-design-patterns/9781449341954/ch01.html

[6] "MapReduce Tutorial", The Apache Software Foundation, http://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[7] "Introduction to Parallel Computer", Blaise Barney, https://computing.llnl.gov/tutorials/parallel_comp/

[8] "Google's MapReduce programming model -- Revisited" Science of Computer Programming, https://www.sciencedirect.com/science/article/pii/S0167642307001281

[9] "Special Topics Course (Supercomputing) - MapReduce and Hadoop", Department of Computer Science SUNY Stony Brook https://www3.cs.stonybrook.edu/~rezaul/Spring-2016/CSE590/CSE590-lecture-10.pdf

[10] "Mimir: Memory-Efficient and Scalable MapReduce for Large Supercomputing Systems" Tao Gao, Yanfei Guo, Boyu Zhang, Pietro Cicotti, Yutong Lu, Pavan Balaji, Michela Taufer https://www.mcs.anl.gov/~balaji/pubs/2017/ipdps/ipdps17.mimir.pdf

[11] "Parallel Hardware and Parallel Software: a Reconciliation" University of Kent at Canterbury https://pdfs.semanticscholar.org/5a6f/211594a123f4dbc57d22a9f85d1d4fd80c92.pdf

[12] "Introduction to Parallel Computing" Blaise Barney, Lawrence Livermore National Laboratory https://computing.llnl.gov/tutorials/parallel_comp/

[13] "Getting Started: Analyzing Big Data with Amazon EMR" Amazon Web Services https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html

[14] “3 Ways Supercomputers are Impacting your Life Today” Office of Energy Efficiency & Renewable Energy https://www.energy.gov/eere/articles/3-ways-supercomputers-are-impacting-your-life-today

[15] “The Future Is in Supercomputers” James Richard “Rick” Perry https://www.whitehouse.gov/articles/the-future-is-in-supercomputers/