Implications of Combination of Operating Systems and Programming Languages on Parallel Programming Efficiency

THOMAS F. FREEMAN HONORS COLLEGE

SENIOR THESIS

BY

Isaac Cerda

Electrical & Computer Engineering

College of Science, Engineering & Technology

MAY 2019

APPROVED BY

Mury S Desai 4-18-2019

HONORS FACULTY FELLOW MENTOR DATE

Dr. Dianne Jenkard 4-18-2019

DEAN, HONORS COLLEGE DATE

TSU

TEXAS SOUTHERN UNIVERSITY

Thomas F. Freeman Honors College

Table of Contents

VITA

March, 2nd ..... Born – Monterrey, Mexico

2014 ..... Member of Houston Archeological Club

2014 – Present ..... Honors Scholar;
Thomas F. Freeman Honors College
Houston, Texas.

2015 ..... Mr. Honors Sophomore
Thomas F. Freeman Honors College

2019 .....Honors College Senior Thesis:
Implications of Combinations of
Operating Systems and
Programming Languages on
Parallel Programming Efficiency.

May, 2019 ..... B.A Electrical & Computer Engineering
Texas Southern University
Houston, Texas.

Major Field ..... Electrical and Computer Engineering.

Acknowledgements

I would like to thank the staff and students I have encountered during my time in Texas Southern University. They have all shaped me and helped me see the world differently, and now I do not simply see the world with pure academics. The Honors College Faculty, Dr. Dianne Jemison Pollard, Ms. Renuka Nair, Mr. Shandon Neal, and Dr. Hector Miranda, have all given reason to keep pursuing education. Through the tough times they have listened and have given me assistance to how to keep going. A great debt of acknowledgement to Dr. Mayur Desai, who has also given me a great chance to help me improve. All the great mentors I have received and had given me their time, I would like to give my thanks and that I will continue to improve while keeping your guidance in mind. Thank you, Thomas F. Freeman Honors College for giving me a great opportunity to do this research and having faith in me.

ABSTRACT

There are a number of programming languages and operating systems, and the appropriate selection of the two depends on the type of application. The effective application of parallel programming depends on a number of factors such as hardware and software, the application type, and the cost. At the micro level, understanding the concepts of threading, processors, and memory are also critical factors in deciding the type of parallel programming to use. The purpose of this paper is to discuss the impact of different combinations of operating systems and programming languages on the efficiency of parallel programming.

Chapter 1 - Introduction to Parallel Programming

Parallel Programming is a method to use algorithms within programs in multiple sequences all in the same time in order to complete certain task. Sharing tasks in order to complete different implementations, and even different ideas to fully make the system make the most out its memory. Within programming, specifically C++, it is necessary to manage the use of memory. In addition the execution from the compiler will become easier. While libraries can be imported in different programming languages, it can easily add a new ideas and make parallel programming easier to use. In order to achieve these new type of programming ideas, computers must be built in order to withstand the demanding usage of parallel computation.

Processor Development

As time goes on the processors are upgraded with the idea that it will reach its capabilities to find the fastest way of execution. Over the years the processors were skyrocketing the amount of processes they were running either in a series of executions or in parallel (Ray Kurzweil, 2005). As of present we are reaching the peak of power we can gain from computers and handheld devices. With parallelism we can easily make it more fluent and improve the results. Not only is this a bit difficult to do since threads that use one core of a processor can be changed at any given time and maintaining the code that is being passed or written can be tedious. Since the speed is not advancing as much as before, there will be an instance where improvement may not be achieved. Things that can be done with parallel execution that can bring a lot of new technology not only to have a new framework but improve efficiency. New

applications can be made such as an advancement in banking system, medical equipment, and economical databases. Key thing is to maintain the processor to always be busy. Since the processor is constantly managing all the data that is being passed it will rarely ever have any downtime. Having the processor to constantly be on uptime, the processor will be close to being 100% efficient. Having an ideal processor will always produce ideal positive results. The more power the processor can produce the better the results can be. This also causes the execution time to diminish so that other task can be done afterwards. When processors were being worked on such as the 1950's on the development of the IBM 650 machine, it is clear that it lacked processor power. At that time period, companies had machines that can only do executions such numerical calculations. When the first computers were introduced they were stored in labs. For instance IBM 650 was stored in universities labs (Frank da Cruz, 1955). As of now the smaller handheld devices such as a smart watch have more processing power than the IBM 650. An example would be how wireless systems work nowadays, where before it was typically a single telephone being able to make calls.

Concurrent vs. Parallel Programming

Each operating system has its own method of putting these POSIX threads in use however, there is a difference in parallel programming and concurrent programming. Concurrent programming specifies and focuses on tasks being defined in any order randomly. One thread can occur before or after any other thread. In example there is no control since they can run and stop whenever they choose. Concurrent programming benefits from threading since it can get information quicker and complete task in a timely manner in comparison to finishing one by one a single line of execution. Each processor can do more if it is more gifted with cores which will

improve the execution time. It can be useful but in parallel programming the key thing is to do tasks simultaneously. Being able to achieve this will not only benefit the execution time that happens, but also uses less resources. If more things can be done within a time period then the more things that it will be accomplished overall.

Threading

A thread is simply a term used for a processor register that has values saved. This thread is then saved into a single core of the processor where it can then be used to execute instructions. Threading usually are associated with kernels in the processor. Multiple thread strings can be used simultaneously to create something visual or mathematical by using software. Using the idea of shared memory where not just one processor will be forced to do one simple task, it can easily share tasks. Issues arise such as deadlock, where a thread will have to wait for another to finish its action so it constantly cycles wasting valuable memory. Each language has its own flaws and advantages which can lead to some being more useful than others. Needless to say all languages follow around the same pattern when using threading, multiprocessing and memory when creating threads. Since there are many different operating system, there is also different methods of parallel programming that goes on inside the system. Some work on Linux, MacOX, Windows, and many other operating systems that are well known. Each having a specific way to managing this parallel programming. These types of threads are special since they all work in a similar way, not all work exactly in the same manner. In Linux some functions are only design to work properly on this platform. The reason why is since the compiler is specific to the operating system that it is in. Since you can use C++ inside Linux by using the program called Code::Blocks IDE. You are able to write a simple program with basic knowledge

of coding. Since you always have to provide libraries to use in code, then the library for threads will always be used with the line of, "#include <pthread>." To create a single thread you will need to use the code of line, "pthread_t threadname1; This will cause the programming language to create a single thread. Since the idea of parallel programming is the same throughout all of the operating systems, the idea and structure at times is a bit different. With that idea, maybe one certain language with a certain operating system may provide the most benefits and give the best function to the basics and advance ideas of parallel programming. With so many operating systems and languages there has to be some perks of certain ones depending on the type of work that is supposed to be done. Again, it is so vague on which is better to be use for certain kinds of projects, but as basics go when learning programming, the focus will be towards which is better suited for parallel programming. So that when the user decides to learn more about the subject, then they will understand the path they should take. Either a rigorous one or something straight forward. Also to not waste valuable time learning a language for parallel programming if it does not suit the design that the programmer is looking for.

A Few Issues Due to Parallel Programming

Information can be deleted or corrupted at any given time making parallelism hard to implement but not impossible. The more information and experience that is gained about parallel programming, the more it will benefit future devices to do task more efficiently. The main issue that happens often is having multiple threads collide together. This will cause the system to have a memory lock until the processor fixes the issue. With this in mind the primary focus will be on which kind of language will give more benefits to create parallel programs. With more benefits the less drawbacks a system will have throughout the execution of the program. Deadlock is a

very important factor to take into account since it will actually slow down the system significantly. For instance, when a program that is opened and there is barely any RAM, random access memory, to allocate for the given execution, the program will simply wait and end up "freezing." It will wait for the processor to decide what to do and give an interrupt signal so that the issue is looked upon. As many know the constant rolling circle of the mouse courser, it is a significant visual representation of the issue that the processor is currently having. This representation will indicate that the program is in a deadlock state which is waiting for the processor to take action so that it can become free. In most cases when it occurs, other functions are also not permitted to happen because it is trying to find a solution to fix it. When doing concurrent programming these issues are needed to focus on so that it will not happen. Parallelism has things that it wishes not to do while trying to achieve the most efficient method to execute its methods and to provide a result quicker.

Chapter 2 – Techniques & Languages

Since the languages become updated every once in a while, new techniques of parallel programming becomes available. The languages become updated with new libraries. Libraries are additional software information that can be added so that the programming language can use new ideas created by users. This enhances the way the language works, which can also improve functionality. As devices become more complex, the languages that are used to program them also increase in libraries. With every code that is made, there is a main thread. In threading, there is a mother and child concept. This concept is where the mother thread is the parent, and can create more tasks which become the child thread. The "mother" thread holds information of the child thread within memory state. So once a thread is created it is simply sitting there waiting to join the main thread. The main thread is already built in and does not need to be called. This main thread has a purpose to run items that are called on periodically and constantly until all of the tasks are finished. This can take a while and having it run constantly can decrease your RAM and lower the performance of the system. This is due to the idea that programs that are ran have to finish one by one. This is somewhat fixed by the priority they have. Priority is simply how important a certain execution is compared to another execution. Each having an execution time, delay time, and period. Depending on the values that each specific variable has will decide the priority of each process. These are ideas of how processes inside the processor are ran. Since the main thread job is to run these processes, it is established by certain execution methods such as LIFO, a processes where the last one in is the first one out. This certain method enables the processes that are brought last to finish first, and then continues until the first process is finished. You can think in a form that the last process will be the first one to execute, leaving the first on

of the bunch to wait until the rest are executed. Another idea would be periodically execution, where the period is defined per each process and the deadline are taken into account. Deadline specifies how long the process needs to take in order to finish. The one with the higher priority will be ran until the next priority process is given. Many different methods are used to help the main thread know what it should do. As there are many different execution methods, it will also translate to how the parallel program will be ran. This either will change how the program should be written or simply how it should look at it. As the processor runs differently depending on the model, it is recommended to consider using different languages depending on that specific processor. As some have more cores, so they are more capable to do LIFO and FIFO, first in – first out, faster than others, then the execution time will decrease.

Techniques

Parallel programming can be done with multiple languages and the key difference is how it is written. Each language has its updates on its libraries which ends up with improving or adding new techniques. As stated before LIFO and FIFO are techniques, but deal with more of the execution of data. Techniques are not always methods to how to parallel program, but also on how to oversee the code. Being able to keep track of variables, how to use implemented data types such as Pthreads, and being able to keep track on tasks are also techniques that are used when parallel programming.

Keeping Track of Variable Values

When there are many variables, it is necessary to maintain an idea of where certain values are at during the program at any given time. Keeping tack will also help know when variables

change, which in turn can help solve bugs, issues that the program may have during development. By using message passing it is capable of keeping track of variables and objects of the program. It also exchanges data to other areas, which helps both receivers and the senders. This also get processor to be ready for any given task that. As the concept of parallel techniques need certain ways for implementation, there are different conceptual requirements to keep in mind. As languages have new techniques, such as libraries, are created, it further complications some aspects of parallel programming but also sometimes solves some issues. One technique, threading, from C++, is a specific implementation, where instructions are given to where memory stacking are used. These memory stacking also includes local variables, functions, and other implementations that calls functions to be used or shared.

Some Pthreads Usages

Operating systems such as Windows, Linux, and Mac OS have their own methods and coding to create threads for their programming. For instance Pthreads is generally what is used to create, execute, and change threads in programming within the operating system. A Pthread is simply an instance of a thread within the POSIX programming interface. They have their own libraries which can incorporate community libraries to further develop the interface. Pthreads cannot be used in every single operating system. Some require special installation of software to transcribe it into readable code for that specific operating system. Linux is the main operating system that takes advantage of threads but it isn't the only one capable of using it. Other operating systems such as Solaris, Mac OSX, Tru64, and certain shareware that is used in Windows can use these kind of libraries. Key thing is that the global variables are usually shared by all the threads. This allows every single thread to have access to every information of every

thread in the system. With each thread having the power of changing and sharing its data with other threads. One thread should have to wait for others to rejoin before they continue or else it will get cluttered. An issue will arise that will cause many executions to be done which can also change variable values at any given time. This is an issue since both are trying to change a variable at the same time since they are running inside the processor concurrently. They will both run simultaneously and will end up changing the variable to whatever is ran last. The issue is what if the variable is supposed to change by the second thread after the first thread is finished. Then the result will be a proper change of the first thread. Issues with threads in any language or operating system, as stated before, is the global variable predicament. For instance, when a global variable is changed it changes for the rest of the threads at that exact moment. This may cause issues due to results changing at any time so parallel programming itself has some flaws. Some simply do not use global variables for that reason alone. It is a hassle to maintain a check of what the value is at any given time. Threads are used for parallel programming but issues can occur when the global variables and information is not kept protected. When variables are not protected information may be lost or tampered with. A global variable is where a variable is used to where each function or section of a code can have access and manipulation of that specific data. Without protection the threads will continue to run and produce an unappealing result for the user. If protected, the user will obtain the results that are sought for.

Keeping Track of Task with the Use of Pthread

As memory is shared and is being used by multiple programs it is necessary to maintain them so that they cannot interfere with one another. At times where multiple processes are being called upon and working at the same time to make it seem like it is working in one CPU (Central

Processing Unit) core, but it is actually working with multiple. Multiple examples of this is used daily as you run your computer, for example having the Windows or Mac OS systems to do multitasking. Having multiple programs to be execution as long as there is free memory to be used. This allows multitasking which is always used to accomplish multiple task at the same time. While being able to run multiple applications there should be an option to close them out to create new memory. Pthread is typically used in Windows with the use of C++ which is often called POSIX, which is one of the main ones that programmers tend to use. To get familiar with it, it is necessary to understand that the language is also provided with an extended library which will help if the methods are known. In the concept of threading the possibility of exiting a thread is made simple with the use of one line, pthread_exit (thread_Name). This line only exits the thread being passed to it, all while the rest, if there are more, continue to be executed. As there are multiple ways to use threading, it is recommended to understand the equipment that the program is being used to work on. Some having less memory will cause the threading idea to not be used as much since the system cannot take control of the parallel programming capabilities. For instance when there is a parent and child threads, each will have their own addresses with its own memory, holding them active always will clog up processor. When it becomes clogged the system will slow down in speed and cause the rest of the processes to wait. Fork is a method that simply splits off parent processors and creates child ones with its own address and memory space. Each being independent of one another which causes separate addresses to be taken into account by the processor. This ends up becoming more complex as they split into more. At times using the method can be useful but at other times it can create issues such as having to maintain a different set of local variables that take up memory space. Since threads are also very efficient, they can also be difficult to maintain. Yet, once it is done correctly it can have many benefits.

They can be switched between each other at any points as long as the system allows it to. Not having to make different addresses or process memory space, description for files or even variables can save a lot of time for the CPU. With the Pthread library, there is a function that can easily delete a thread by exiting it in comparison a process takes longer to take away. When processes need to share information it becomes difficult. This is due to having pointers which essentially uses variables to point a specific place that another variable is currently in. With each pointer variable having a specific memory location then manipulating that pointer will change the actual value of the other local variable at any scope of the program. Implementing a pointer can have its drawbacks in a process because when another threads needs a previous value that was changed by another thread, it might have already been change. Threads fixed this issue by sharing all the memory together so the space they are all sharing makes it easier to use variables without having to change it. Once a data is produced by a single thread every other thread can access it with ease. Only downside in threads, in any operating system or programming language, would be the sharing idea. Once a thread itself becomes an error, it will affect all the others due to the fact that they are all sharing a memory space. One suffers and then the rest do fail as well. Keeping track of a thread to not fail is a key thing to do. Within C++ there are also other libraries that can be added, it only enhances its ability to be capable with other operating systems as well.

Multiprocessing

Multiprocessing is by having two processes or more that can communicate with each other. While multithreading is an idea that all threads can share information and communicate with one another. In multiprocessing, the idea is sometimes difficult to maintain due to the

amount of communications, since it has to be extremely careful for one process to not enter another by mistake. Only sharing information if any, when necessary. When sharing memory it is sometimes easier to handle threads so that they can have less issues. Once a thread is joined in the main function, then it cannot be joined again after it has been detached, stopped running. If the programmer does so then the program will crash, yet there is a method to check if the thread is joinable again in order to be used. A function using an if statement, if (Thread_Name.Joinable()) Thread_Name.join();. Allows it to check if it is being used again without crashing the whole program. If a thread is called and does not have a chance to be joined or detached, then the program will terminate, therefore you will have to use join or detach before the thread becomes out of scope, unusable. Making sure which threads are being used and which should be discarded due to the fact that memory is so important. The more memory that is filled up without organization will cause issues to arise. This in turn goes against the idea of what parallel programming does. Being able to keep track of which thread to delete is always a good thing to take into account. The more structured the program is, the easier it is to execute and to achieve true parallel programming. Without achieving proper parallel programming and not using the PThread library fixed methods will only causes problems. Windows itself makes it easier to understand with the use of Pthread because it has already been settled universally, It has been given out to the public to read and understand. This is an open source material that will benefit multiple programmers to understand when starting to look into parallel computing.

When a thread is called and then another program is called afterwards, the thread will be lost due to it finishing first. A method to maintain this thread would be to use the try-catch-throw methods used in C++. This eliminates that the thread would be destroyed instead it is "catch," so

whenever the program finishes then it will throw the thread back out. Then either it will be detached or joined afterward depending on what the user will be doing. Another method would be using RAII, Resource Acquisition is Initialization, so when call upon Wrapper w(Thread_Name) will be the new scope for the thread. Once it is out of scope then it would automatically join the thread causing the thread to remain. Passing reference using std::ref(Variable_Name), can be used to change certain thread. The thread is being called from outside the main function and can only be changed by passing references to it, which in turn creates different outcomes depending on the references that are passed. What this means is that if a class, for instance main, and thread are created to print out something in particular, they can be manipulated to print out something else depending on the references used, This is only to maintain a structure change or passing of arguments. Without causing issues since at times when the main function is finished, it will just delete all of the other threads, but with using the references it will make sure to focus on certain threads.

Thread Locking and Identification

Each thread has a certain identification number associated to it and in order to receive or print out where the ID is at, the user would have to use, "Cout << std::this_thread::get_id() << endl;". This is allowed when the programming language is C++, and every other programming language may have a similar way to finding out the thread's identification number. Running multiple threads to the point where CPU cores are outnumbered will cause the system to decrease in speed, and times simply shut down. This is called Oversubscription. Using a function to avoid this would be std::thread::hardware_concurrency(); which will show you how much threads can be used concurrently without causing issues to the sub hardware and memory.

Whenever two threads are running at the same time they are constantly trying to fight over each other for the common resource. Most common resource would be using the, "cout," which would be necessary to print out anything that threads would like. To avoid such an issue a function can be called in order to share the print together. In order to accomplish this a function would have to have two parameters. A new class function has to be called. It will be, class LogFile{ std::mutex m_mutex; ofstream f; public: LogFile() { f.open("log.text"); } void shared_print(string id, int value) { std::lock_guard<mutex> locker(m_mutex); f << "from " << id << ": " << value << endl;. After this class is created then continuing to the next step should be easier. The calling of std::lock_<std::mutex> guard (mu); should be called in order to lock in the cout so that when one thread is called upon, the other thread cannot use the cout until it is finished. This will in turn look something similar to; void shared_print(string any_name, int id){ cout << msg << id << endl; }. When this is called then the program will know to print out one thread without having issues and unlocking it. This will create the capability to reuse this mutex without any issue, and print out in a protected manner. Some things should not be done, such as returning the f variable, ofstream& getstream(){return f;}, which would allow the user to access f, without using the mutex as a security reason. Also to never allow the f variable to be used as an argument that the user can enter as an argument in a new function. This will cause the usage of "f" in any way and change it and will cause it to be out of control of the program. This will cause it to be leaked and cause security issues in the code. It is important to use the MUTEX idea since it is a way for it maintain security so that no other outside code can change it. It is the idea of when there are multiple threads and when the user wishes to only focus on one in particular. When focused on one it will also help to not cause issues or changes to the rest of the threads. This is when mutex becomes more apparent because it is simple like putting a cage around one thread. All the

changes will occur within the cage. This will in turn only affect the single thread that is in the cage without causing issues or changing any of the other variables throughout the program. Now, if this is not done then possibly when something is changed within a thread, it will also change the variables and outcomes of the rest of the threads. This is something to be cautious about since when changing one thread the user does not wish to make any sudden changes to the rest. Mutex is a great way to import so that a mistake such as this, is not easily made.

Even though threading can be safe, it can also leak out data if not used properly. Different issues with threading can be found but in the end it can be used in a protected manner and can cause different solutions. Data race, which is one issue to be fixed in threading is to use certain methods such as mutex to synchronize data access. This never leaks out any information about what the thread has publically, but it helps to code carefully when create a class to accomplish something with the thread. Threading is very useful to allow different codes and programs to work together with the same data or even with different data to create a huge projects. Even to the extent that they can be done simultaneously without issues and using less memory space. Being able to be more fluent with threading can lead to bigger projects, or even easier solutions to issues or task that are currently in process. While it creates a quicker and easier form of coding multiple things in one, it can also be one of the riskiest if coded wrong which could possibly leak information to the public that should be kept private. Not just in parallel programming is safety and memory leak a big issue, but it is valid for any language and any programming model. The reason why it is so crucial so that the users are always wary about the safety and public data that can be seen, is so that data will not be leaked. When data is leaked it can cause issues since the data may be so important that it should not been seen through public eyes. This usually matters

due to the idea that anyone truthfully does not want their information to be seen by others that they do not tRust. When it fails it can also cost a company millions of dollars at times. Parallel programming has its own methods of keeping things private which often helps with maintaining a proper safe environment.

Now the main thread is a thread that is always there. If you create a thread and let it join the main thread then the number of threads are increased. For instance if you create two threads inside the main function, then it will have a total of 3 threads. This is due to the idea that the main thread is already created initially. Now to join the threads you will have to use the line, "pthread _join(threadname1, NULL);". The join line lets the thread that is created join the main thread. Having multiple threads is quite important if you wish to run multiple things simultaneously. Now the methods discussed previously can be used to run many execution processes at the same time without having to wait for the main thread to decide when it should be executed. Now if one of the two threads is open then it won't have to wait as long. This can cause the files to be executed even quicker and helping the processor as well as increasing the response time of the system. This improves it many ways and it is the point on why the system should have parallel programming for most cases.

Languages

There are multiple languages that can be used to do parallel programming. Each having their own way of using the techniques discussed previously. C++ is also a language that can keep track of multiple task and do parallel programming with efficiency. C++ will also be the language that the

following languages will be compared to. The languages that will be discussed will be Rust, Java, and Python to see how each can provide parallel programming efficiency.

Rust Language Usage with Parallel Programming

Rust is another language that is still in development. What this means is that it is open to the community where many can provide solutions or new libraries to it. With this in mind the Rust team implemented some parallelism. Since this language focuses on individuals who were too afraid to even step foot into parallelism it simplifies it a bit more. This language is also a bit more of a difficult to comprehend. Rust compiler like all other languages checks if it is unable to work and if there will be issues. The only difference is that Rust compiler checks statically for the programmer and makes sure the memory is safe. Rust's compiler will also figure out in depth issues that are harder to see especially when debugging. When a code is created in Rust, it will create a channel where it will be a sender and a receiver. Something in the sense of data communications. The compiler will send a pointer from one thread to another since it focuses on thread isolation. Where one thread is by itself until it is dismiss. It is dismiss when the code is done or when a receiver is not present and after it will stop the iteration. This is better already since the threads want to be isolated so that another cannot change its information if not needed. A lock is also present where the compiler will focus on not letting the data leaked. A comparison is that other compilers from other languages tend to focuses on the code and not the data that is being passed. As stated before that threads that take information from another thread can cause issues but if they all access certain variables it may help the overlapping. Maintaining this is difficult but with Rust it does enforce the usage of these variables in a safe manner. They can easily access without any issues of deadlock happening. Sharing information between each

thread even after the thread has finish, it will save the state it was without causing any memory issues or variable confusion. These are some of the basic information that can be used in Rust that comes inside the language itself. Now as time goes by the community will hopefully provide more libraries that will enable the language to use the advantages it has to the full extent. Using temporary variables also helps with threads since it focuses on grabbing information from another thread without causing confusing. Instead of implementing this on other language, Rust itself allows this to happen without having to worry about it. Providing this is such a relief since it does it by itself which makes the program a bit easier to tend to. Taking out an issue and only solving it so that the programmer does not need to worry about it. This is important since Rust itself will help the programmer to focus on program. Focusing more on that end will reduce the attention of figuring out what to do with the variables themselves since the language handles it all, making it easier overall.

Since there is also no overhead runtime it is a lot more secure and more viable. If there is an error where a vector is constantly grabbing different size information that is large than the vector size, the compiler will let the programmer know there is an issue. This helps to figure out an issue before the code itself gets too complicated. Solving issues before and not having to use a debugger to figure out the problem of the program. It also utilizes the message passing approach where they do not share the memory as other parallel programming languages do it simply communicates the information that is needed. This helps confusion and deadlock that may occur when a variable is being used by different threads. Also it fixes the issue of share-state of threads where locks are used. Locks are typically used when you want to isolate a thread by itself so another cannot interfere or cause issues. Mutex as stated before is when a thread is locked and

can only be worked on within the lock. The compiler in Rust enforces this rule so one misstep of accessing data that should not will be declined automatically by Rust. Within Rust they let two versions of accessing the threads one that uses less memory while the other is more secure. This can help improve how fast a program can run and how safe certain information can be. "Knowing that if they ever do accidentally try to send one to another thread, the Rust compiler will say: 'Rc <<Vec<i32>> cannot be sent between threads safely'" (Aaron Turon in Fearless Concurrency with Rust, 2015). This is better since it will help understand the issues before they are done and ran by the program. Maintaining safety and no data to be leaked among the code that is being made. In some cases where a parent creates a separate thread called a child, since the child depends on the parent sometimes scopes are an issue. Sometimes the parent thread finishes its execution first while the child is still running. Once the parent exits the child will be left unable to finish since the information that was being shared from the parent is now gone. Rust itself makes sure as well to not let this happen since it will force the parent thread to stay active until the child exits as well. If it does not and is not implement it will let the programmer know that the child thread cannot finish due to the issue of scoping. Data races are also an issue when it comes to parallelism as explained before but there are ways around it. Since Rust itself tries to prevent these from happening by not having any garbage collection and having concurrent programming with using data races it helps data to be more linear without having data corrupted.

Rust

As time goes by the Rust programming language may get better when it comes to parallel programming only improving on everything that already focuses. The main thing Rust focuses

will be making a processor and program more efficient without the headache of issues that most programmers come to face when making these codes. Rust will only improve as more users create more libraries that can be used, fixing bugs, and making new classes that become standard. So far Rust itself as a programming language seems very efficient in parallelism. An ideal language for the most efficient and safest way to create parallel programs with. Only issue is since it is still being worked on and still is far away in comparison to other such as C++ and Java. On the bright side even with the amount of work that is done already it still seems more viable to work on, but for more advance coding possibly the other programming languages will be more viable. It still has the safest and most convenient methods of making parallel programming work since it constantly checks for any issues throughout the code that is usually difficult to notice and having data leak checked constantly. As for now it is somewhat ideal to use this programming language in major projects, but not fully. Since it is still being developed and improved upon then a suggestion would be to keep an open mind with Rust, but do not pick it as a preferred language for concurrent programming.

Java Programming Language in Parallel Programming

From operating systems create their own methods to maintain threading and parallel programming in their systems, programming languages have their own take on it. Java, another computer language, can also be used to create threads. This leads to believe that any programming language can support threads as long as there is a library that can be added. All Java programs use threads and they even support singles threads. Which is in the same principle like the main thread. This language can make threads but with the help with creating objects. Which leads to the idea of object oriented programming within programming languages. The

objects have to create a runnable interface. These threads are supported by creating a different approach for them. All in the same sense that all Java programs can be compiled in every operating system that can support Java. When the thread is created as an object and is runnable, it resembles a function. For example, "public void run()", the run method allows it to be compiled and to see results from the function. Just creating this function provides a starting point of the thread but it does not run. Just like in C and C++, Java has its own methods in which threads can start and end. To start the operating of start() has to called. Since the start() is an easier tool to create most programming do not use the run() function since it can be ran indirectly. The difference between these languages is that Java can create threads and they do not recommend the usage of global variables. This will cause them to change information by themselves. Not being able to go to the critical region of their cothreads. This solves an issue that was in the C and C++ languages. Yet, when something is solved another problem is presented. In this case the issue of not being able to use global variables. To solve this, another class will be created and this will be the sum of the integers within the threads. In Java the class sum will be created but with private variables and public variables. This in turn solves the issue that was presented. Since it will get the private variable from another thread in which no other thread can change, it will return it as a public integer. With this the function with the class Sum will be created. With having it start with the public int Functionname(){ return sum;}. These threads have to pass this reference so that the shared object can be passed to other threads. Having a more complex approach can either be more challenging due to how the interface works. In some cases Java can be used to do parallel computing, but in regards to the comparison to other languages it is pretty clear that it is good, but not the best.

Python - Synchronous vs. Asynchronous

Another programming language that can be used would be Python, its approach is a bit different than the others. It has two ways to do programming in parallel by optimizing the CPU cores, again it will signify that it would be parallel processing or multi-processing. The key difference between each would be using threads in comparison to processes. Not only is that but there also a difference in synchronous and asynchronous programming. Since synchronous focuses that everything happens one at a time. Each function takes a while for it to complete and produce a result which causes the program to stop every time there is an action to take place. Since each task needs to finish before another task is started, it will take longer than expected. Having threads makes it harder to debug afterwards if there is an issue because when a result is needed, figuring out where the result is current at any point will be challenging. Causing an issue in the end and makes the management of the code to be strenuous. When a thread is used to do this, when one function needs a local variable from another function to perform its task, it will wait until it is provided. Now, if more than one functions need results to continue then they will result in a deadlock. Where they will remain waiting, consuming RAM until it gets information of another function to continue its work. Resulting in a loop of waiting until one is finished. It is a common issue with not only Python, but in any language that uses multiprocessing or parallel programming. This problem is not only just simply in a specific operating system, but it occurs in all. A simple reason is how Python works with using parallel programming. It has its benefits but its drawbacks do cause an issue no matter the operating system.

On the other hand asynchronous models allows multiple things to be running at the same time. Once an action is finished it sends the result to one of programs that is being accessed to.

Both have its drawbacks as well. Synchronous allows multiple threads to be used by having a multi-core processor. This helps by achieving different task at the same time. The only thing is that it still needs to wait for both threads to be finished then it can use the core to run another thread. Unlike, asynchronous where it just uses one line, it can do things at the same time, while at the end of finishing an action it needs to notify the program that it is done. All of this is done in one section, which at times causes an issue if a thread is not finished with its execution. At times using this method also creates issues with interfaces such as when a program is requiring data form something else. The program will remain frozen until it has obtain the information resulting in time being wasted. Asynchronous programming for Python maybe not the correct way to go since it will require more time to work and obtain information for the remaining code. Other errors are visible when you need certain functions to provide an actual result and that result is needed. So running asynchronous can cause some issues.

Chapter 3 – Operating Systems

There are multiple operating systems, which can range from handheld devices such as android, to computers, such as Windows. Since each operating systems have their own way of working with threads, which will also decide on how to parallel program, understanding the difference will also impact the efficiency of parallel programming. Some operating systems give less cost of the hardware used, while others may provide efficiency. The operating systems that will be discussed will be Linux and Windows, and how Java and Python can be implemented to them.

Linux

While Linux can run both C and C++ compiler, it can have both threads working properly. The only difference would be in C you would start the line using cc – and in the C++, the start of the line would be g++. Without these small difference the code will not run properly and will cause issues to be encountered with both threading and the build of the code. While other specific codes such as exiting the thread can used with the line, pthread_exit(); this will cause all the threads to exit, even the main thread. As stated in the above paragraphs when using threads you will use a thing called mutex. This is a critical thing to use since it will lock the specific thread that is running, allowing change to occur without another thread to come into play and causing a change. The importance of this mutex would be to allow a certain code to run with a specific thread all while the rest of the threads are focusing on their own execution and priorities. The code does get a lot longer and a bit more difficult to read but it is beneficial to maintain information protection. Once the thread is locked by the mutex no other thread can change any information in it other than the code that is design for that specific thread. Once that specific thread gets unlocked by the mutex, then the new variables and the new executions that

are within that thread can be used by all the others. Linux is also a very cheap operating system. Most of the software and hardware that is compatible with the operating system is usually fairly cheap compared to other operating systems. This is an advantage to provide parallel programming applications all while maintaining a low budget. Linux can also run MPI, and most intel processors which can work together to create a very high performance application.

Windows Operating System

Now for Windows has different methods that are used to achieve the same amount of executions as Linux. You will need to know what pointers and arrays are. Pointers are typically only used to get the addresses of certain files or items in the register. These are given as addresses. Once the pointer is called, it literally points towards the address that certain variable is being stored. This in turn will cause the pointer to have the addresses and records that address to a new variable. This is simply used to either copy or keep data protected from being changed by an outside source. In Windows they have different codes that are used in which pointers come to play for threading. For instance a new include file will be called upon. This file is received from the registry of the Windows and allows the usage of threads. The line of code will start in the very start of the program which will read, "#include <Windows.h>". DWORD is what is used to have an unassigned int. This is only used in Windows since it is defined in Windows only. Winapi is used to clean out the stack when it is ran as a function. This prevents any corruption to occur and just focuses on a new function that is clean and clear of processes. In Windows the main difference between the pthreads in this operating system would be the syntactic and nomenclature of the coding. In other words how you call on certain functions, and how they are created. These are the major differences all while the basic understanding of the threads remain

the same. Within C++ or C computer language most of the basic functions remain the same. Integrating the same functions in the Windows operating system just requires a bit of a change but nothing to major.

POSIX Usage with Java in Most Operating Systems

Another key point in POSIX would be Fork/Join framework as stated before. It utilizes the objects that was made where it will divide into smaller instances of objects by creating its own local variables. As each level goes down, it waits for the results to be done and then it adds it to the next child. Requiring the parent level to finish it code first and passes its results to the following child. After this is all done it will add it all together form the child classes and merges it into one final child class that will produce the final result. Once finish the child threads will exit until all is done, then the parent exits afterwards. This helps by checking how much work there needs to be done. Such as in parallel programming it key point is to separate the work flow while simultaneously finding out new results as other threads are doing the same thing. If the work is small then there is no need of parallel programming by utilizing the Fork/Join methods. If it is a larger result to calculate then Java would be better. In comparison threads in C++ will need to be created individually while in Java the function Fork/Join makes it easier to create a parallel programming environment. While it is more object orientated, it may make it a bit difficult at times in comparison to C++. The difference between these two languages is very key to remember due to the idea of how the code should be programmed. As a specific function of a language may be used differently in another, it is important to note when and how it affects the code. Not only that, but also the type of operating system changes due to how threads are used. Some do not simply do a single pipeline to send data to be used. In some cases they prefer to use

threads differently and enforce their own rules. With parallel programming become more used in society with different forms of systems it is important to note how they system works. Not simply what kind of language is it running but how does it run with that kind of operating system. What is really intriguing is how some forms of languages also try to incorporate the same ideas even if the operating system uses threads differently. They tend to try to go around it to provide a universal feel to it so that it can help with coding. Which is what sometimes makes them universal to use and why it is necessary to understand the parallel programming concepts in depth.

Asynchronous Usage within Operating Systems with Python

Throughout most cases it seems that asynchronous is more beneficial in most cases. Since the ability to do certain things and multitask with another action without having to wait for other executions to finish. As CPU performance goes up having multiple threads giving requests and responses back will make it more efficient and produce more results faster. Since we are focusing on threading using cores of the CPU, it seems like it Python may be better of using asynchronous method instead of threading. The reason being that the GIL, global interpreter lock, allows only one thread to be ran in Python at a given time. Since it prefers multi-processing multi-core handling is more difficult to do since it does not allow you to do multi-threading due to its single core usage. This does not change depending on the operating system that it is being used in. It is the basic threading idea that the Python language has implemented. So in general in the context of using Python as a language to do parallel programming it is not the best. Possible refraining using it other than a singular program execution, since simplicity is the main focus when it comes to Python. One way to get over this issue would be by using sub – processes along

with each memory space when creating threads. Python integrated its own class to follow such as a library. It creates a single memory space while using as many cores as the user would like, depending on how many threads are created. It simply makes Python compatible with using threads. Explained before the GIL does not allow multiple threads running at the same time, running a multiprocessing can fork each process into having multiple child threads. This bypasses the issue that Python has, all while each thread holds a portion of the programs memory. In regards to either to use Python in the sense of multithreading would not be too beneficial in comparison the languages beforehand. Possibly will be viable in certain situations that would need a bit less memory usage since Python threading system is to focus one thread at a time.

The idea that Python uses is that it will switch between threads at any time to compute the necessary information. It makes it seem like it is doing parallel programming but in reality it is not. It does this by using the CPython interpreter to do this method. Along with this having another external interpreter such as having another file that has access to the network can run as proper parallel. Concurrent programming is what is mainly happening with this language. While Python can work its way with some parallel computing, it is very minor. As most of the parallel computing belongs to mainly computation of either numbers or string variables, it also only has a sense of one pipeline which can only do some types of execution. Python has its benefits and is very useful for many things, but the sense that it cannot compete with the other programming languages within the scope of parallel computing. Python does have a method to how to fix its parallel programming. It is a bit more difficult to execute but the idea is still there. It will require the idea of multiprocessing where it will create multiple interpreters which in turn will act like

processor cores where it will run code independently. The processes will not be able to share data, but if there is a need to share data then there will need to be a shared state to occur throughout the code. It will execute all the process in parallel with the idea that the each will run independently. Now this can only be done if the hardware and software are capable of working in that manner. In some cases it cannot so a higher end computer or device is recommended if it is going to Python concurrent programming. If the CPU has multiple cores then it can run the Python easier with the idea of parallel execution. The module for multiprocessing contains its own classes in which are needed in order to create and synchronize the processes. There will be issues when it comes to shared state as well, mainly dealing with the idea of when adding a section of a shared memory it will cause issues if a variable is changed. For example if there is an increment that is needed to happen, such as a counter variable, there might be an inaccurate change in it. The reason being is that when there are multiple threads and all share the same variables. So when a variable value is change the rest of the threads that use the previous value will be affected. This causes an issue since at times the programmer wishes to do threads in separate memory allocations so that one does not interfere with another calculations and executions. Python itself can be work in parallel to some extent. When compared to other languages no matter on the operating system it is easily noted that it can have its disadvantages to the rest. With that being said about Python, it is best to use it for minor programs that require a not so powerful device. It will work but not efficiently, but will cause issues not only with share memory but also at times with synchronization.

Chapter 4 – Speed of Processors, Threads, and Operating Systems

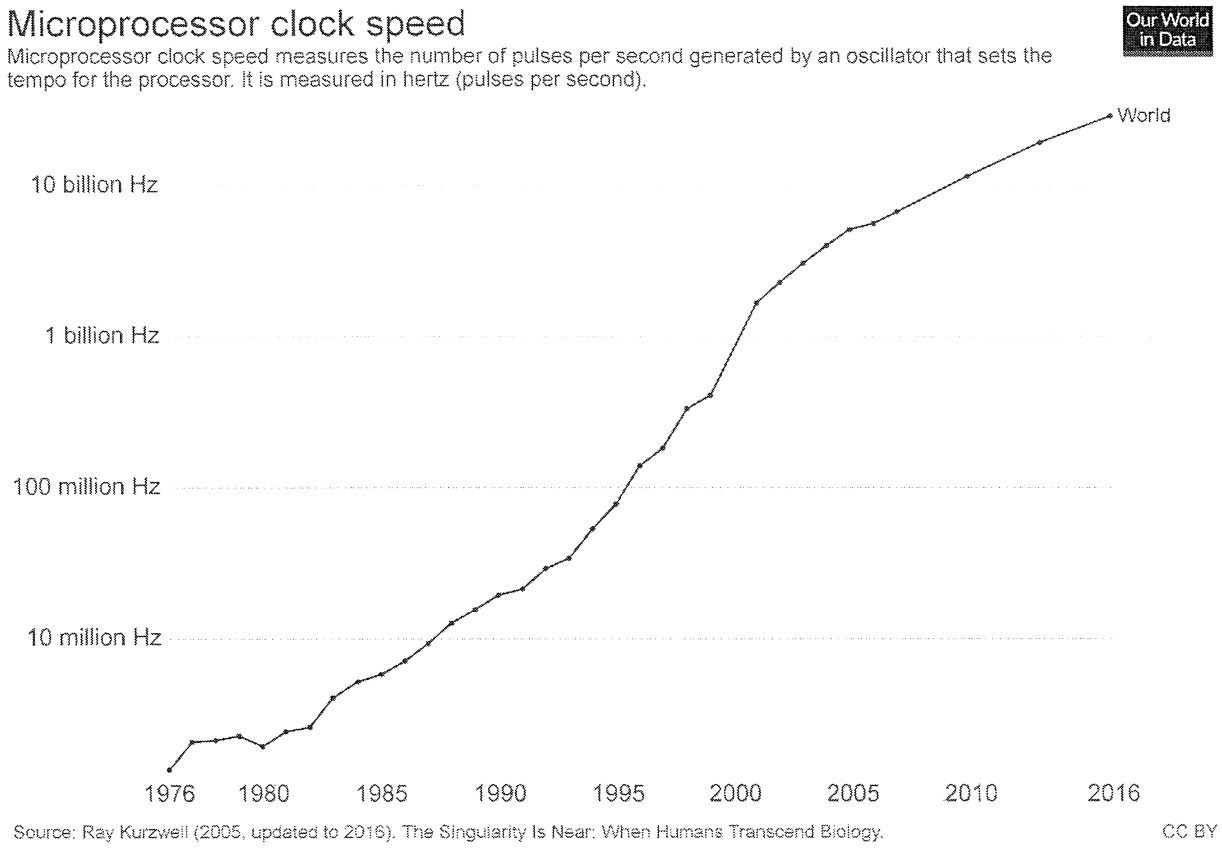

Figure 1: Note: Data 1976 - 1999 (E .R. Berndt, E.R. Dulberger, & N..J. Rappaport), Data 2001 - 2016 (ITRS 2002 Update)

As stated before the speed of devices have risen quite fast over the years, but have seem to slowly start to reach its end point of speed. Processor speed was improving rapidly as seen on figure 1. As time went by, during the years 2010 – 2016, not much a change in speed has occur. Most processors are reaching its high clock speed capacity that in turn will reach its peak of performance. So for instance speed can be calculated as, $speed \leq 1/(S + \frac{1-S}{N})$. Here, S, is for serial computation and N would be the number of processing cores. From the equation we can see that as the number of cores approaches to infinity the speed will end up being 1/S. This will

reach the limit of the max amount of speed. From this instance it is assumed that 75% of it is run as parallel while the 25% is serial. Due to this, many CPUs have limited themselves to 16 or 8 cores overall to maintain a good balance of speed and processor power. By using all the above languages talked about previously, it can be seen that the coding aspect may bring a difference in performance as of now. As coding gets faster and more reliable without that many bugs, they are still limited due to the hardware. This causes a great issue between both hardware and software since they both need each other to work properly. If a code is implemented but the hardware is incapable of keeping up, then the code itself is useless. The same thing vice versa can happen causing a great ordeal of issues that are hard to fix. From the equation we stumble into a hardship of finding out how we can improve. Well through parallel programming in any language it can help improve performance through software mainly since hardware is limited. Programs are typically presented to the public and are usually compatible with most of the systems. Keeping that in mind will show how a programmer needs to code their project so that it can be able to execute properly on most devices.

Models of Parallel Programming

Many models of threading are placed in different operating systems such as, many-to-one, one-to-one, and many-to-many. Some operating systems prefer certain models, which can have advantages and disadvantages. Learning how each model uses the idea of threading is beneficial to know how parallel programming is being used in certain programs or even operating systems. For instance in many-to-one, there is a single kernel which is what manages the hardware and software to work correctly together. Since many threads are tied in to a single kernel, sometimes it can cause issues such as a threading thread becoming unusable. Another

issue would be multithreading since only one kernel can be used at a time. This is an issue because since only one kernel is running no other threads can be ran simultaneously. For instance if there were more than one kernel then there can be more things that can be ran, but in this model none are able to work in that manner. For this reason not many use this model since it is very limited. One-to-one is a bit more user friendly and causes less issues. Of course that would mean more hardware to be used. This model ties a single kernel to a single thread. Yet, there are many kernels that causes parallel programming to take advantage. For instance where it can freely use different threads at the same time to finish executions. A lot of operating systems use this such as Windows and Linux. Another would be many-to-many, where multiple threads can access other threads at any time. Some use this model but it is a bit more complex. In most cases everyone usually follows one to one due to its simplicity and less issues. To pick a certain model, it is necessary to figure out which will benefit the code the most, and which will it cause less issues to happen. The lower the amount of issues the more free the model can work with and not cause any processor performance drops. To properly use a model it is best to understand how it is used in both kernels and threads. Knowing how each thread is being used by the hardware can give the programmer a better sense of how to write the code. Not specifically in a certain language but the aspect of the hardware and how to use it to the best of its capabilities. At times when a programmer is trying to use threads, they become only focus on the idea of the coding of the certain language they are programming with. Doing so sometimes makes the programmer forget the hardware capabilities. With only understanding the aspect of it, it will diminish how the code should be working. Knowing how the operating system works is a key factor of parallel programming because it gives boundaries of what can be done. A crucial mistake is not understanding this and just coding on demand then running into issues because the hardware that

is currently being used. Not only hardware but also the operating system might be inadequate. This brings an important thing to the table so that when learning parallel computing, the user should have the knowledge of the programming language, the individual hardware, and the type of operating system.

Languages and Its Benefits Due to Performance in Operating Systems

Now that there is basic knowledge about programming languages and what they benefit along with the hardware and models that are proposed, learning how they can all work together is key. Making them all work together is where parallel programming can create more processing improvement. Since there is a limitation about how fast a processor can work. Making threads work and interact better might be the next thing to focus to further improve system performance. Using different languages to implement the coding for this idea is where the most beneficial language will have the most use. From what was gathered it seems that Rust and C++, will be the best ones for this job. Java itself can produce more object oriented programming where it focuses on items having parents and children classes. The focus is a bit different from Java and the rest. C++ focuses mainly on the idea of programming different things inside the main function as well as organizing all the pointer variables if used to be stored in data. The same thing with Rust, where ideas can be stored and not accessed because of its compiler security. Where users may have issues fixing or figuring out but with the help of the compiler. Rust seems the best but as time goes on and the community grows then this programming language might be the most idea for parallel programming to be the safest and the easiest to work with. Now itself, the ideas and the learning of Rust is a bit more difficult compared to other languages since it focuses on users that already have programming backgrounds. Alongside with focusing on Rust, Java and C++,

can have its benefits to a certain extent. As stated previously both Rust and Java has its issues due to either not having enough libraries or the ability to create easier threads compared to other languages out there. C++ holds a great value of incorporating both the hardware and software together not just simply because of the basics of its language but also the extended information that can be added to it. For instance the POSIX, OPENMP, and OPENCL all have its own addition to the C++ language. This further enhances the ability to use it and create many different programs all while using the same basic language foundation. Yes, when adding the extended libraries the way the code will be programmed to work with parallel is a bit different, but the difference is what makes it a good idea to decide which addition should be used. Since this language works with almost any kind of operating system, the results are endless. As most cases the main operating systems that are used is Linux and Windows. Through the context that how threads are being used by its models within operating systems, the addition to the libraries only make it work easier without all the complications that it had previously. When the idea of all of the core aspects are working together the code is easier to work with and debug in case an issue comes. When knowing the kind of operating system and hardware, the programmer can pinpoint the issue about a certain language a lot easier speeding up the process of coding. When a process of coding is sped up and the user becomes more adequate to program in parallel the outputs are easier to achieve. Parallel computing is focused mainly with using different task to finish simultaneously and figuring out how it does will help achieve the results that are needed. As the operating system and language becomes faster and more convenient the more parallel computing programs are made. Which in turn will speed up the system creating multitasking to occur, thus improving performance of the system.

Yet, that does not mean that Rust or C++ should be the ones that should be used often when doing parallel programming. The main thing that was found out during the research of this paper would be that every language has benefits to use in its libraries that are made. Some better than others, but not all have a total advantage compared to others. Since each language has certain libraries that may be more beneficial to the idea or project that is going to be done, then every project should be carefully thought about which language to be used. Meaning for certain projects one language may be better. Also what computation requirements are needed, what device it is going to, and what is it going to accomplish are big questions to think about before just choosing the most familiar language. If thought properly then choosing a language that gives the best benefits will always end up being better and providing a more ease implementation. Since most languages do provide a similar amount of thread implementation then saying one is better than the other is not fully correct. All languages are also safe to some extent, while mostly all have to take precaution of memory leak or deadlock issues. While others offer more secure methods it is pretty much all the same. A programmer should always be weary of how much information is seen and hidden away from users. Some languages copy some of the basics of main languages such as C++ and Java, but uses them only to further enhance its capabilities. Scala specializes in parallel specifications within C and C++ that mainly uses the libraries of MPI and CUDA to be able to do parallelism accordingly. Also OPENMP and OPENCL are other instances where it uses the C and C++ approach to complete parallel programming. It typically also depends on preference if the user prefers a self-made library that can be used throughout the project, which in the end is limited to only that library. Something in the likes of MPI where it uses other additional libraries that works with threads and further enhances the code. Now both are good depending on what it is going to be used. For instance in a mobile device having only a

tight end programming language program will only let specific devices to be able to run it. Manufacturers use this so that their software cannot be used by another. While pretty much all operating system use parallel programming, figuring out the best language to use for a project is the hardest to decide. Like stated before if the designer wishes to only make a program or an operating system to work only for a singular device then that is what is going to be used for the project. No addition libraries can be used. Along with arm processors for small handheld devices it maintains it and deters it from being copied. Other operating systems such as Windows allow addition libraries to be added. This is when it becomes a bit more universal where communities can make programs with the basic framework from Windows.

Operating Systems Performance Due to Hardware

Most operating systems already adapt and welcome parallel programming. May be due to different models but it is still compatible. All languages can be used and all can give similar results. Since most operating systems already are programmed to work with multiple cores, and having a certain percentage of parallel programming, as discussed with the equation of speed, then applying a code to it is easier. As there are different methods to do parallel programming, nowadays it is easier to implement coding to make it work with the many different operating systems. Different languages are also compatible but some aren't such as PUNIX, which is Windows and Linux. They only work for those two operating systems and other methods to program parallel with using C++, Java, Rust, and other languages that branch from them are universal. From that idea it is best to program in parallel using kernels and threads with the one on one model as a base for a majority of programs. This is due to the fact that since it is universal, it will be easier to import the programs to other operating systems, or mirroring it so

that the compiler works correctly. The speed in which processors can do also depends on the clock that it has. Depends on the cache where it transfers data to processor, memory and other components of the computer. The faster it is, the faster the instructions can be executed. The memory as well of RAM, where now they are making DDR4 which is an upgraded and faster version of DDR3. The BUS speed as well is how fast the motherboard can fluctuate by going fast or slow on certain data transfers. The hard drive, HDD or SSD where data can also be transferred. The GPU, graphics processing unit, also known as a video card can help visually and maintain the performance of the system by rendering 3D graphics and overall improving the system. Software updates and also having the latest operating system for that certain machine. All of these are key factors of what can increase or slow down the speed of processor unit since it is all tied together. This is crucial do to the fact that parallel computing is not just simply left to the coding aspect but also the hardware of the device. As some processors have a higher speeds, the ability to increase it by parallel threads improves as well. This results not only in a better and more fluent execution but the overall performance of the device. As stated previously the speed of the processors and its cores has a limit to where it barely improves but with the help of the other hardware it can see improvement. Even now the video cards of the devices become more efficient and can render 3D so much quicker, which can also help with threading. Speed is the main focus when it comes to parallel computation but the issue of how hard it is to achieve it. As some portability of certain codes such as C++ cannot be ran automatically, since the compiler from different operating systems do not allow transfer. It creates an issue where if a code works perfectly in one system it might not work in another. Where if a thread has to wait for another thread to finish in order to use the information of the finish product, then it will need to wait. It is not deadlock, it is simply waiting, which coins the phrase weakest link. This phrase deals with

how if a group of workers are all working on a certain project, it will need to wait on the weakest link, individual, to finish. Causing to waste some time that something else can be done. Does not have to be weak, it's more of the sense of either difficult or harder to compile information or a lack of memory allocated to it to ease the execution.

Chapter 5 - Conclusions

Understanding the goal of parallel computation is to finish multiple task simultaneously without interfering with one another. Many devices do this already where it manages to multitask itself, such as being able to play a multimedia file all while being able to take a photo, or surf the internet. Think of it as where each core can focus on one or two things as long as there is enough RAM and processing capabilities. Some RAM either a single or a dual channel memory so that the performance is higher, which affects the speed of the device. For smaller devices that do not require a lot of power it is recommended to stay with a somewhat minimal memory, due to how the user will use the device. When a user is more in the need of higher performance it is recommended to upgrade to a higher channel memory and higher cores. As there is more demand for power, the users are able to do many things simultaneously without an issue of lagging, and of memory deadlock. For instance a new phone during the year of 2019 can give you a maximum of 12GB of RAM in an average selling point for the market at that time. This is how slowly handheld devices are becoming more relevant to being semi-computers. "Some smartphones now have 8 cores (processors), you can buy CPU chips with 12 cores, and this number will soon increase" (Quentin F. Stout, n.d.). Most speed tends to be done when increasing the clock frequency, which is the rate at which the processor can perform the functions that are asked to do. As there are different models for how parallel programming can use threads, new hardware can improve the execution times of processes. This is due to the idea of shared or mutable states where it can either have the information shared between threads or only at certain times. Programs now can improve its performance by taking advantage to how much parallelism can be taking advantage for the computer that it is being worked on.

Due to speed of hardware reaching its limit, figuring out which type of language will be most beneficial with the hardware available is the problem. From the previous information above speed, multiple languages, and different operating systems all have a play together. In fact, to be cautious and figuring out the best overall combination for a certain project will be recommended, so that the user or group can get the most out of their set up. By using all the different languages, even the ones that were not discussed, there is always a better language to use for a certain project. Figuring out which is the most efficient and will cause the project to go on ease, is usually what would be recommended when it comes to parallel programming. As most will typically use C++ since it is more known and is used overall in the industry. Thankfully the language itself has such a high volume of libraries that are beneficial which makes the language great to work with. When it comes to smaller projects in the sense of figuring out which one to use, then it might be easier to choose one that is more abstract. A lot of users who are in the professional field prefer using C++ either with the approach of OPENCL, for shared memory, or MPI, for distributed memory. The reason being is that it is a library that is added on top of the C++ framework. It is two different methods to do parallel programming while the only difference is how the parallelism works depending on the library added. To reiterate from above statements, to pit simply that OPENCL will do parallelism where every thread that is being used for it will have access to all data. MPI is used where each parallel thread is isolated to each own memory space where no other thread can have access to its data. The two reasons that there are two is that either may provide more efficiency or security. In C++, the programmer has to deal with having different variables that are either hidden or can be seen by any other user. Since both are different, the coding for them are also different. While typically using the same basics from

C++ it is a bit more difficult to code in MPI due to its idea of message passing that requires more knowledge. Meaning it has more of a network to deal with instead of individual objects. Both support C, C++, and FORTRAN which are the main ones used in parallel. Rust itself is a great language that is becoming available to multiple operating systems but since it is still in its developing stage where community authors are creating more libraries for it, it may be better to refrain from it until further development. Many other languages work with parallel programming, and sometimes some compilers will not allow other programs that were made to run in other operating systems. Java itself will, but as stated before parallel in Java is available but it is not as common in comparison to C & C++. From what was gathered there are many different methods to achieve parallel programming with what is available. To choose the best, will be difficult. Needless to say, each has its own boundaries and perks so choosing one is typically based upon the project itself. With ideas that are difficult to do, or do not have a proper map, then the best language to do it on will be C++ on Linux to test major programs. In most cases then the operating system of Windows is still considered as one of the best operating system to work on. Since it has multiple communities and upgrades from Microsoft itself, and that most hardware is compatible with Windows, it just makes it easier to plan and execute programs with that operating system. From the basics of knowing the type of threading models each operating system has, assuming the user uses the most common ones, it gives a sense of understanding what threads really do. Since most threads tend to go for a single pipeline, to get things done and once it is done it will meet up with the rest to either terminate or execute another task. Different languages and operating systems has its own ways to work with parallel programming. Each having its own advantages and disadvantages. It is also helped by the hardware that the device has. This all works together to improve parallel programming and to

know which the superior combination of all, is difficult to decide. The best way to focus on what combination is to decide what the goal of the project is. If something more complex it is more towards the recommendation of Windows with OPENCL or OPENMP. Parallel programming is becoming more common and is becoming an everyday thing to use, if more is learned there will be improvement in parallel programming. Thus, the speed of a lot of devices will increase. This can both turn to higher and faster calculations in banks, better devices for hospitals, and of course make everyday life easier to manage. The sole purpose of this concurrent programming is to improve devices all while giving benefits to daily life as well. When programming it is best to have a decent or a high end computer with a proper operating system to do parallel programming with a language that will bring the most benefits to the project. Yet, from the gathering of details towards parallel programming, it is said that it is best to use C++ and FORTRAN with either Windows or Linux. It has the most benefits and is widely used by most commercial companies.

References

Bell, D. T. (n.d.). Threads. Retrieved March 3, 2019, from https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html

Chapter 11Asynchronous Programming. (n.d.). Retrieved February 18, 2019, from https://eloquentJavascript.net/11_async.html

Cruz, F. D. (n.d.). The IBM 650 Magnetic Drum Calculator. Retrieved April 11, 2019, from http://www.columbia.edu/cu/computinghistory/650.html
Last Updated March 26, 2019.

Damian, M. (n.d.). POSIX Threads. Retrieved December 18, 2018, from http://www.csc.villanova.edu/~mdamian/threads/posixthreads.html

Data from 1976–1999: E. R. Berndt, E. R. Dulberger, and N. J. Rappaport, "Price and Quality of Desktop and Mobile Personal Computers: A Quarter Century of History," July 17, 2000, http://www.nber.org/~confer/2000/si2000/berndt.pdf. (Accessed on April 9, 2019)

Data from 2001–2016: ITRS, 2002 Update, On-Chip Local Clock in Table 4c: Performance and Package Chips: Frequency On-Chip Wiring Levels—Near-Term Years, p.167. (Accessed on April 9, 2019)

McCurdy, M. (n.d.). Python Multithreading and Multiprocessing Tutorial. Retrieved February 25, 2019, from https://www.toptal.com/Python/beginners-guide-to-concurrency-and-parallelism-in-Python

Ray Kurzweil (2005). The Singularity Is Near: When Humans Transcend Biology, from http://www.singularity.com/charts/page61.html (Accessed on April 9, 2019)

Roser, M., & Ritchie, H. (2013, May 11). Technological Progress. Retrieved April 10, 2019, from https://ourworldindata.org/technological-progress

Stout, Q. F. (n.d.). Retrieved February 18, 2019, from http://web.eecs.umich.edu/~qstout/parallel.html

Turon, A. (2015, April 15). Fearless Concurrency with Rust | Rust Blog. Retrieved January 5, 2019, from https://blog.Rust-lang.org/2015/04/10/Fearless-Concurrency.html