**Parallelizing a Program**

Robert Davis

School of Computer Science, Northeastern State University, OK, USA

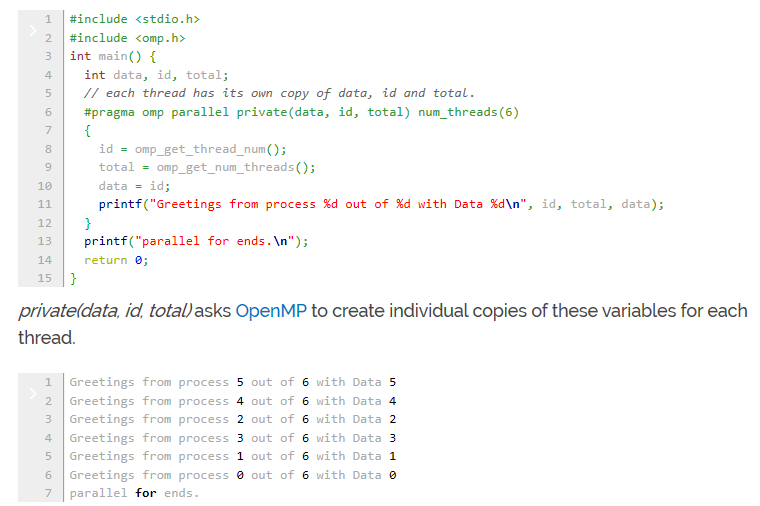
Email: davis178@nsuok.edu

**[Abstract](C:\\Users\\spitt\\appdata\\roaming\\qualcomm\\eudora\\attach\\Abstract)**: Parallelism is used for making the program get done faster and use all processors/cores in the process. Parallelism in computer science is growing and programmes, engineers, and computer scientists need to be able to use parallelism. Coordination language is design simultaneous activities and have them talk to each other. Parallel programs use concurrency. Three paradigms of parallelism are result parallelism, agenda parallelism, and specialty parallelism. Time and money will be saved is one advantage of parallelism computing. One disadvantage of parallelism is code tweaking must be modified for performance of different of architectures. OpenMP goals are standardization, lean and mean, ease of use and portability. Threads are executions of a process scheduled by the operating system. Thread based parallelism programs use up to four threads to complete the process. There more than two types of parallelism but found two that fit perfect for everyday use which are task parallelism and pipeline parallelism. Trying to write a program that can be parallelized is hard. Finishing the program does not mean it is perfect because it can still be designed better and perform faster and make it easier for the use to use.

***Key words***: algorithms, computing, parallel computing, OpenMp, processors, program, parallelism, coordination, computing language, parallelizing, C++, API, task parallelism, pipeline parallelism, data parallelism

**1. Introduction**

Parallel computing is for using multiple threads to use all the cores of the computer to complete a program to get it done faster and easier to use. Basically, breaking a problem up into smaller pieces to get done in the shortest amount of time. Throwing processors at one problem did not make sense and the processors were expensive too. It is hard to find programs to be parallelized. Algorithms cannot be parallelized are often replaced by better algorithms that suited for that type of program or programs. Processors are used simtaneously for execution for parallel programs. A multiprocessor or parallel computer or separate, autonomous machines connected by network. Examples of parallel machines are a cluster computer which contains multiple PC’s combined together with a high-speed network, shared multiprocessor by connecting multiple processors to a single memory system, and chip multi-processor contains multiple processors on a single chip. Massive parallelism is encompassing parallel machines with ten thousand powerful processors or more. Parallel programming uses computing language and coordination language. Computing language lets us computer values and manipulate local data objects. Coordination language must allow us to design simultaneous activities and require those activities to communicate with each other. Parallelism will become an essential part of every programmer’s repertoire. General phenomenon- coordination is example of parallelism will become widespread in computer science. Programmers, software engineers, and computer scientist will need to understand parallelism. Parallelism is a proven way to run program fast. Processor packed into a single box usually communicate with each other at much greater speed than processors connected in a local or wide-area network. The principle is still the same, take a hard problem, break it down and hand each piece to the number of processors whether the processors are in one box or out across the country. Not all parallel programs that run well on parallel computers, fail to perform well on networks. Coordination is an ensemble which a computer and its user are two active independent pieces if the computation is to have any meaning. Parallel programs use concurrency to run fast; a distributed system- a distributed databases, mail system, or file service, for example use concurrent processes because of the physical distribution of the machines it manages. Parallelism has three paradigms which are result parallelism, agenda parallelism, and specialist parallelism. To the three paradigms are the three parallel programming methods. Each method involves a different view of the role of processes and the distribution of data in a parallel program. Coordination framework organizes strategy for a parallel program- the part of the program for the programmer rely in a coordination language. A person’s goal to produce a series of values with predictable organization and interdependencies. Versatile approach adapts easily to many difference problems is the agenda parallelism. Programs conceived in terms of a logical network is special parallelism. Parallelism is important to AI because more tasks are done on multiprocessor cores and the AI needs to make decisions very quickly.

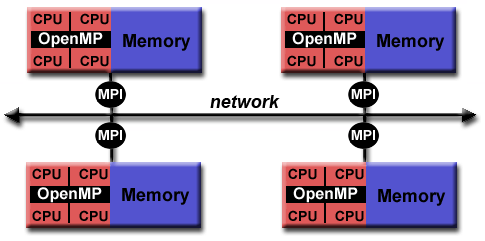
**Picture of A Complete Parallel Program**

**Advantages of parallelism**

It saves time and money as resources are working together will reduce the time and shave potential costs. Solving larger problems on serial computing is illogical. Non-local resources are used when local resources are finite. The potential computing power is best used with parallel computing so it can make better work of hardware than serial computing. Data bases, data mining, real time simulation of systems, science, engineering, advanced graphics, augmented reality and virtual reality are applications that use parallel computing.

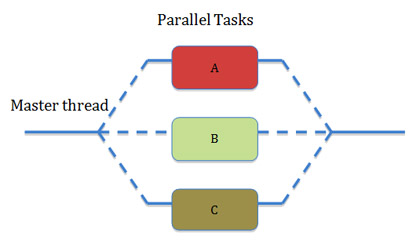
**Disadvantages of parallelism**

It is difficult to learn. Tweaking code must be modified for different architectures to properly improve performance. Estimating consistent results is hard because communication of results can be problematic for certain architectures. The power consumption is a problem for using a multitude of processors for various of architectures; cooling technologies will be required in order to cool the parallel clusters.

**Model of a Hybrid Parallel**

**2. OpenMp**

OpenMp is an application program interface (API) used to explicitly direct multi-threaded, shared memory parallelism. OpenMP is comprised of three API components- computer directives, runtime library routines, and environment variables. The goals of OpenMP is standardization: providing a standard among variety of shared memory architecture/platforms and defined and endorsed by a group of computer hardware and software vendors. Lean and mean setting a simple and limited set of directives for programming shared memory machines. Important parallelism be implemented by using 3 or 4 directives. Ease of use providing efficiency to increase parallelize a serial program, dissimilar message-passing libraries which typically an all or nothing approach. Providing the means to use both coarse-grain and fine-grain parallelism. Portability: API is stated for C/C++ and Fortran. A public forum for API and membership. Major platform has been using Unix/Linux and Windows. OpenMp was created for multi-processor/core, shared memory machines. OpenMP parallelism is limited to a single node by itself. High Performance Computing (HPC) applications, OpenMP is combined with MPI for the distributed memory parallelism. It is often called as Hybrid Parallel Programming. Computationally intensive work on each node is what OpenMp is used for. MPI is used to manage communications and data sharing between nodes. Parallelism to achieve to the full scale of a cluster.

**Figure of OpenMp Threads**

**3. Thread Based Parallelism**

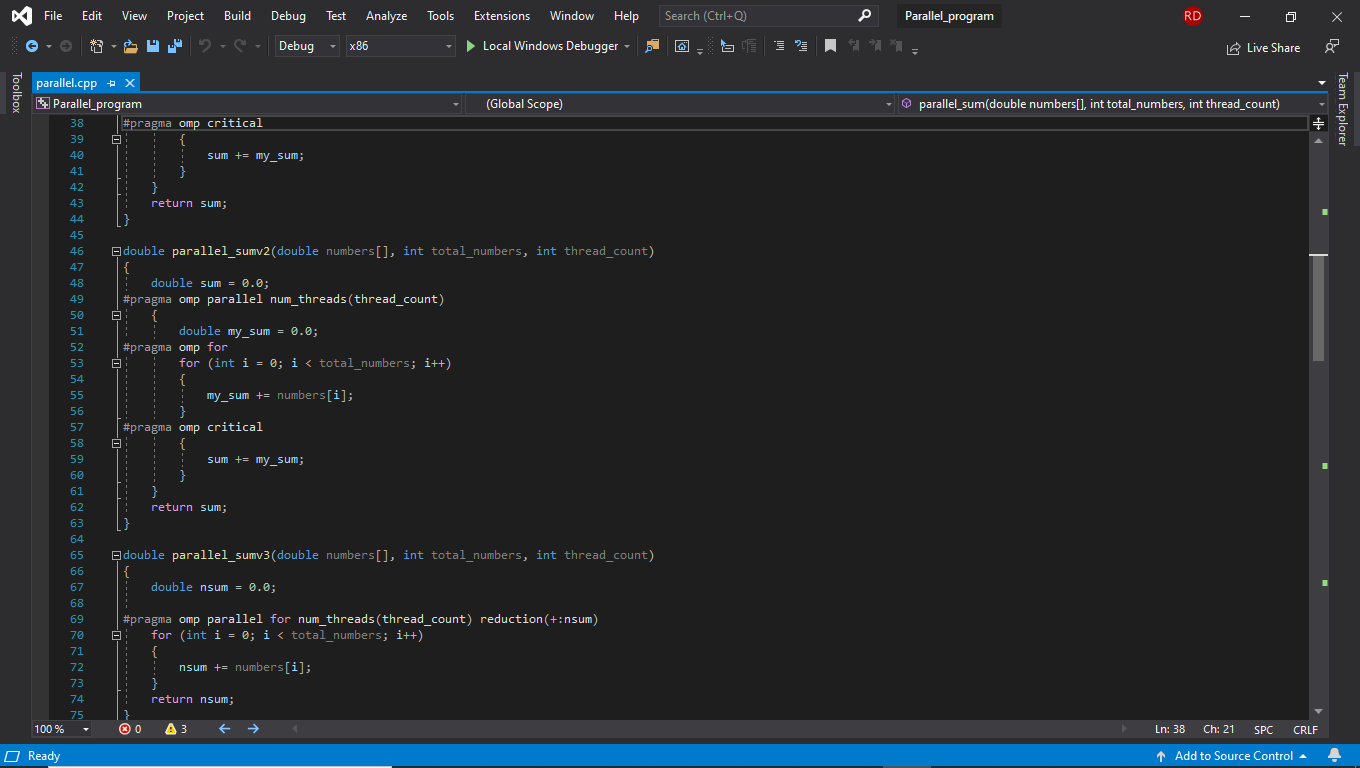
Threads are used for the completion of OpenMP programs. A thread of execution is the smallest unit of processing scheduled by an operating system. A single process resources are what threads stored in. Number of threads match the number of machine processors/cores. Actual use of threads is up to the application of in use. Thread base parallel programs usually use up to 4 threads to complete the process of the program. The “include <omp.h>” has to be included to be able to use threads. Directive #pragma omp parallel default(shared) private(beta,pi) is used to complete a task. There can only be one directive name specified per directive. Parallel for, section, sections, single, aster, critical, flush, ordered, and atomic are directives that either specify work-sharing or synchronization constructs. Some of the directives have clauses that are optional modifiers and they affect the behaviour of the directives. The use of “num\_threads()” is specify how many threads the programmer wants to use. Threads are executed at the same time to get the program finished faster. Each thread shows a different part of the program that it executed when the command prompt opens to show the results.

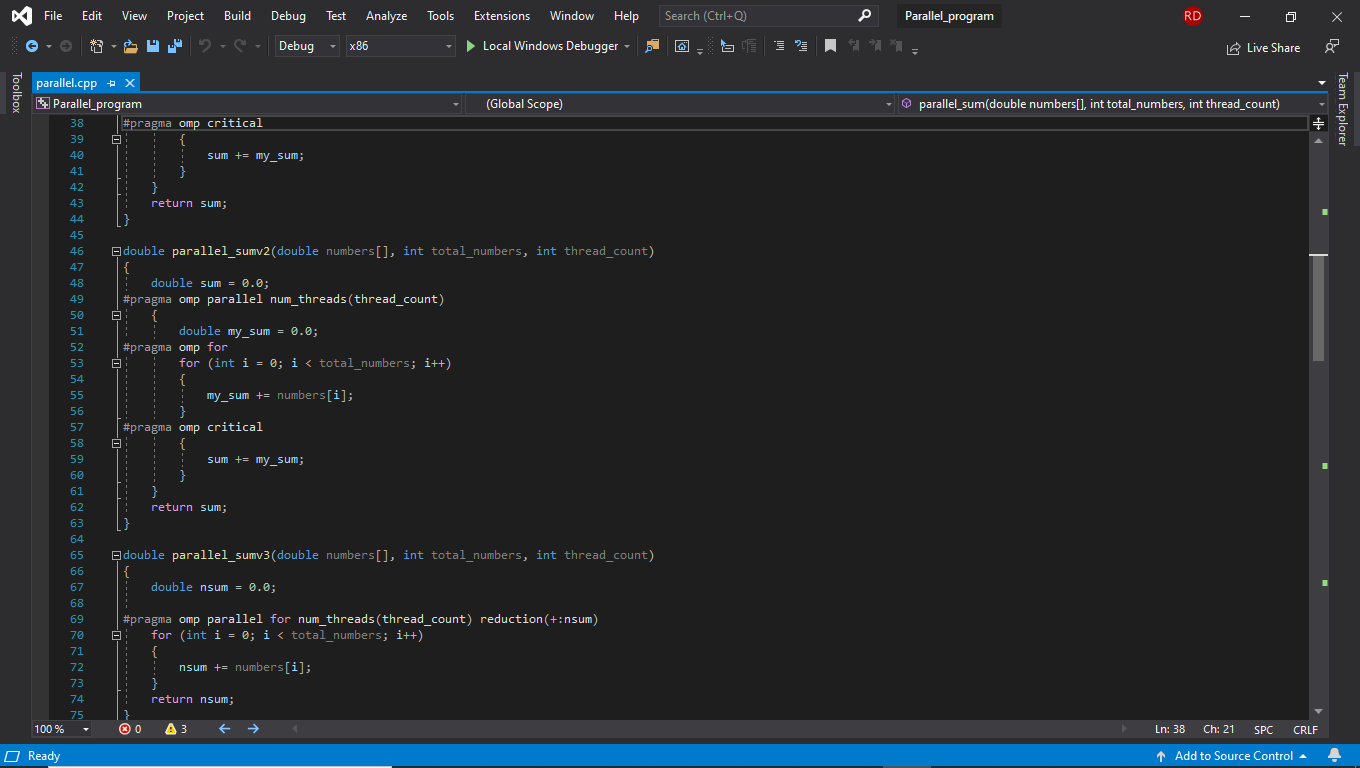
**4. Different Types of Parallelism**

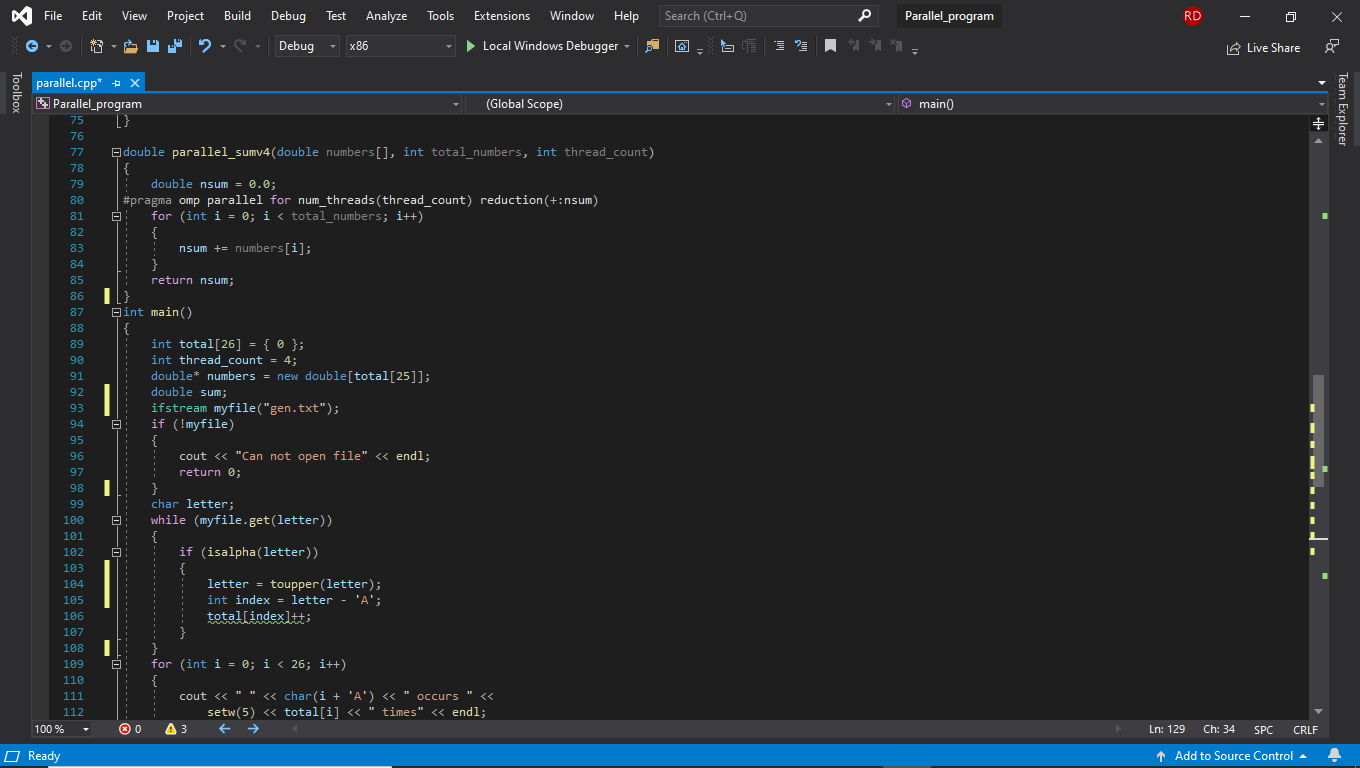
Task parallelism is each process performs the same function but do not communicate with each other, only with a master process. Are often called Embarrassingly parallel codes. Parallel for to avoid the need to serially spawn individual loop body instances and make it very clear that all iterations are ready to spawn concurrently. Examples of task parallelism are independence Monte Carlo Simulations and ATM transactions. Pipeline parallelism is each stage works on a part of a solution. Output of one stage is the input of the next stage. Example of pipeline parallelism is computing partial sums. Data parallelism is abstract enough that a programmer is not explicitly mapping work to SIMD instructions vs. multiple processor cores vs. attached computing. Tools should be mapping not the programming. Shift all the programming back to mathematical expressions, not intrinsic or explicitly parallel algorithm decompositions.

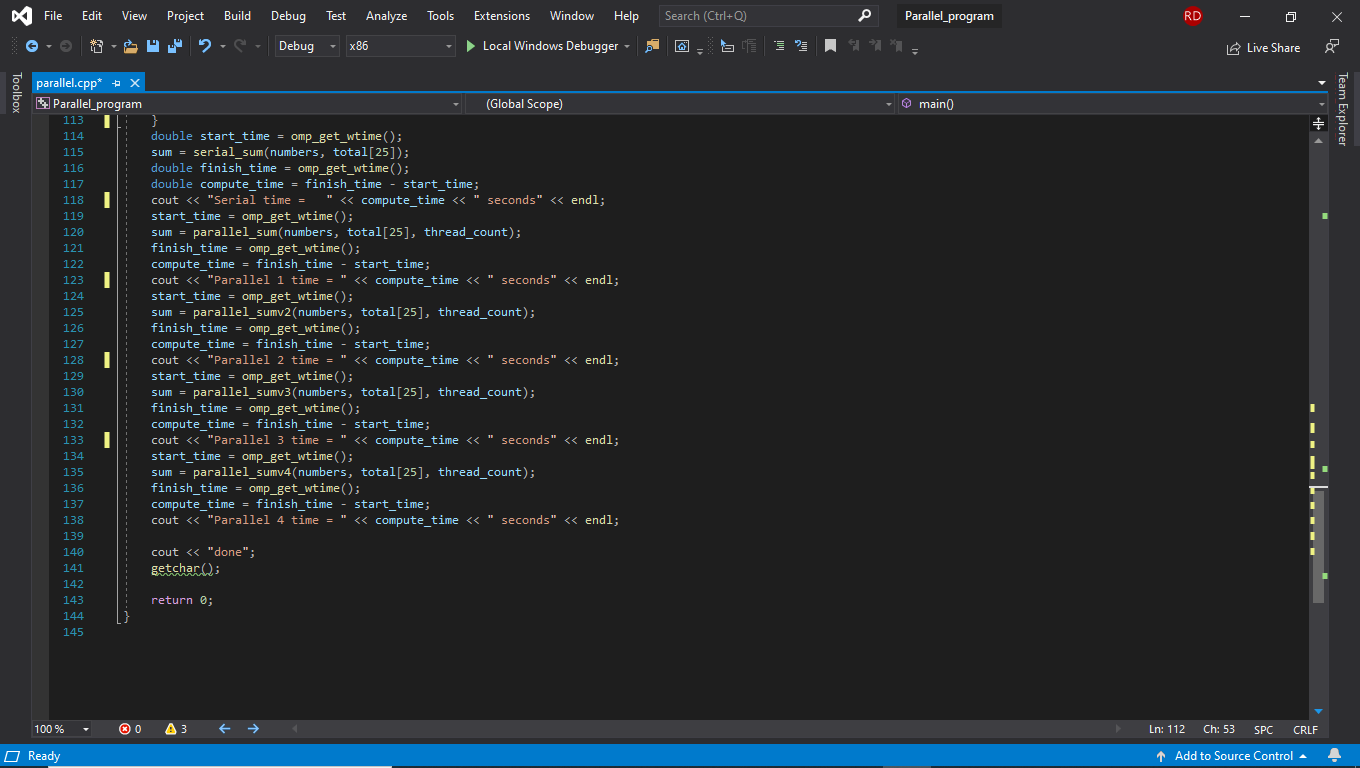
**5. Parallelizing a Program**

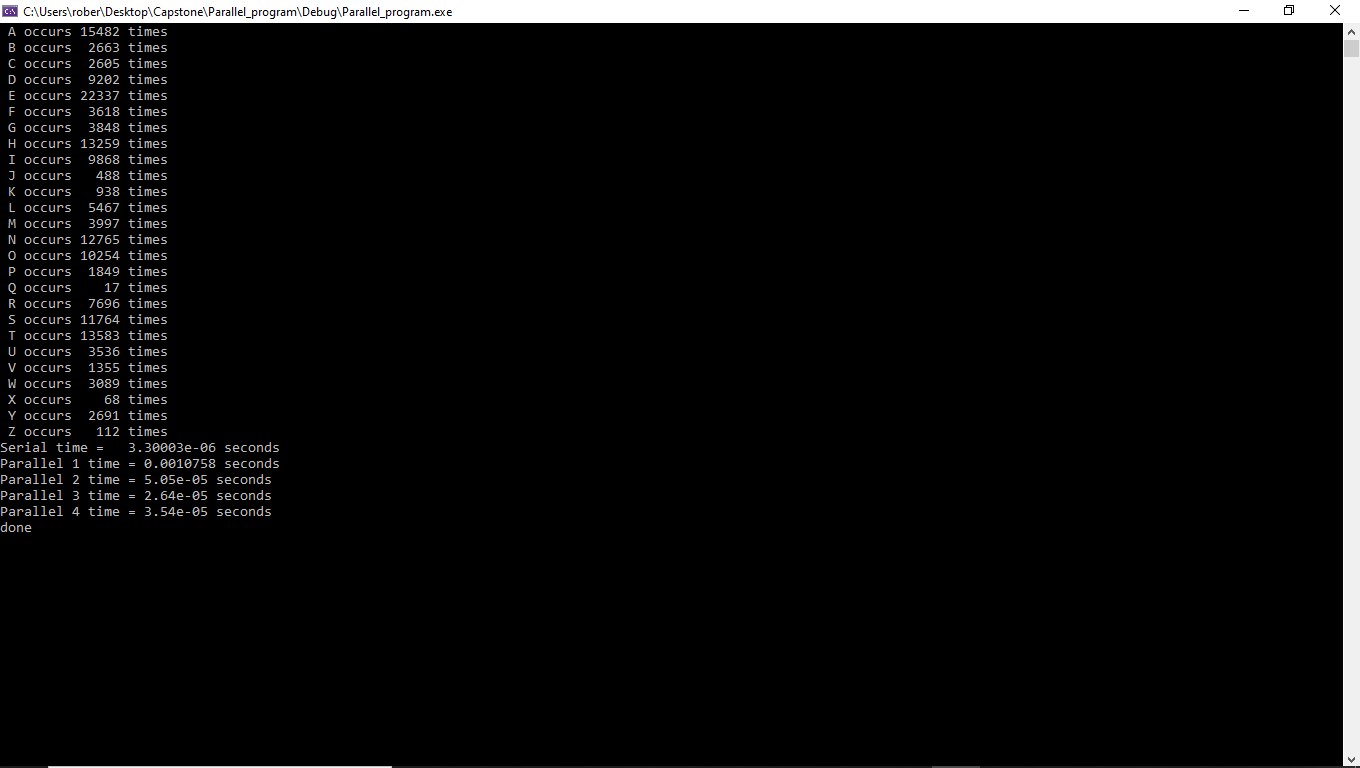
I know I am not supposed to use first person point of view, but I am going to and write about my experience of trying to develop a parallelized program. I had no idea of how to start the process of developing a program that can be parallelized, so I had to get help from a professor. The professor gave me materials on parallel programming to use on how to get started. So, I looked over the materials and still had to come up with a program that could be parallelized. The programming language I used is C++ and the application I used is Visual Studios Community 2019. With help from the professor, we came up with trying to count each individual letter from the bible in text document form into the program. For example, A’s 20, B’s 45, etc. I had little trouble of counting the letters from the bible into a char array, so I had to find a way to make it easier. Finally, I found a way and now the program can count each letter and how many times it occurred. The next part was trying to figure out how to parallelize it. I looked over the material again and use some of the materials to help me get finished with the program. Being finished with the program does not mean I will stop trying to make it better. Below this paragraph will be pictures of my program and the results of it.

**Picture 1 of My Parallel Program Code**

**Picture 2 of My Parallel Program Code**

**Picture 3 of My Parallel Program Code**

**Picture 4 of My Parallel Program Code**

**Picture of Result of My Program**

**6. Conclusion/Future Work**

There three to four ways on how to parallel a program. There are advantages and disadvantages in parallel programming. A person needs to use OpenMP to be able to use parallel programming or it will not work because I did it when I was trying to complete my project. Some parallel programs I looked at only had three to four threads in them so they can maximize the usage of the computer. Parallel programming is hard to do but I will need to learn more about how to do it in the future. In the future I will hopefully be creating more programs and get better at looking how programs work and function. I will probably post my works on github so everyone can see them and get help from them. I will also be trying to make my project better when I can. I will also be learning other programming languages so I can make my programs be understandable for people who do not like using C++. So I recommend to my fellow computer scientists to learn parallel programming.

**References**

[1] Carriero, N., & Gelenter, D. (n.d.). *How to Write a Parallel Programs: A First Course*. Retrieved from https://zoo.cs.yale.edu/classes/cs424/howto.pdf

[2] Acar, U. A., Chargueraud, A., & Rainey, M. (2016, March). An Introduction to Parallel Computing in C++ . Retrieved March 14, 2020, from https://www.cs.cmu.edu/~15210/pasl.html

[3] Simple Tutorial with OpenMP: How to Use Parallel Block in C/C using OpenMP? (2017, January 25). Retrieved March 14, 2020, from https://helloacm.com/simple-tutorial-with-openmp-how-to-use-parallel-block-in-cc-using-openmp/

[4] Barney, B. (n.d.). OpenMP. Retrieved March 14, 2020, from https://computing.llnl.gov/tutorials/openMP/

[5] Kexugit. (2019, October 18). OpenMP and C : Reap the Benefits of Multithreading without All the Work. Retrieved March 17, 2020, from https://docs.microsoft.com/en-us/archive/msdn-magazine/2005/october/openmp-and-c-reap-the-benefits-of-multithreading-without-all-the-work

[6] Universedecoder. (2018, December 14). Introduction to parallel computing. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-parallel-computing/

[7] *What is parallel programming?* (n.d.). Best Computer Science Degrees – Find and pick the best Computer Science Program now. https://www.bestcomputersciencedegrees.com/faq/what-is-parallel-programming/

[8] Belliars, R. (2019, April 10). *How to take advantage of Multithreading and parallel programming in C/C++*. Perforce Software. https://www.perforce.com/blog/qac/multithreading-parallel-programming-c-cpp

[9] R., J. (2011, August 9). *Parallelism as a first class citizen in C and C++, the time has come*. Intel® Software. https://software.intel.com/en-us/blogs/2011/08/09/parallelism-as-a-first-class-citizen-in-c-and-c-the-time-has-come