Introduction to Parallel Programming

Ritu Arora, Texas Advanced Computing Center (TACC)

Abstract: This chapter presents a brief overview of parallelism. The chapter provides an introduction to what parallel programming is, and why you should care about it. The chapter also includes a short overview of the different parallel programming platforms and models.

Acknowledgement: *The content presented in this chapter was supported through the National Science Foundation Award #1642396: "SI2-SSE: An Interactive Parallelization Tool".* **Carlos Redondo,** from University of Texas at Austin, reviewed/edited the content.

1.1. Parallelism is All Around Us

There are multiple simultaneous events occurring around us at every instant in time. The events may be functionally independent or interrelated but their simultaneous occurrence and progress makes them parallel to each other. Let us consider some examples of parallelism around us.

1.1.1. Multiple cash registers at departmental stores

Departmental stores with large inflow of customers often have multiple cash registers to reduce the checkout time of the customers. The customers are helped with their checkout in the order of their arrival at the queues for the registers. As illustrated in Figure 1, but for the multiple queues served or processed in *parallel*, the time that the customers would be spending in a single queue for checkout would be significantly higher.

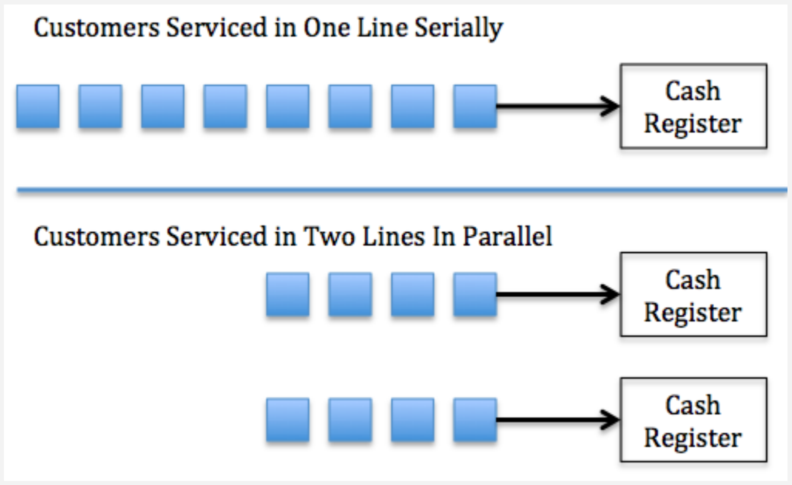
Customers Serviced in One Line Serially
Cash Register
Customers Serviced in Two Lines In Parallel
Cash Register
Cash Register

Figure 1. A single cash register versus multiple cash registers at a store

1.1.2. Commercial laundry

At a commercial laundry, there are multiple washers and dryers available. For washing three loads, one can conveniently use three washers in *parallel*. Once the clothes are washed, they can be dried using three dryers in *parallel*. Overall, by using three washers and dryers in parallel, the time taken to wash and dry the clothes is one-third of the time it would have taken to do the laundry using only one washer and dryer.

Parallelism – in nature or in software - can help in reducing the time-to-results (or time-to-solutions) by using the divide-and-conquer strategy.

1.2. What is Parallel Programming?

Parallel programming refers to writing software applications that have large computations decomposed into smaller units of computation, such that, these smaller units can be processed simultaneously using multiple processing elements. The processing elements can be multiple cores on a single processor or can be multiple independent processors. The results obtained from executing the small units of computations independently on different processing elements are combined meaningfully to achieve the final (or global) result. As shown in Figure 2, this process of parallelization can be neatly broken down into three main steps: (1) decompose, (2) distribute & execute, and (3) collect.

While the “decompose” and “distribute and compute” steps for parallelization are explained above, the step named “collect” warrants a little more explanation at this point. In the “collect” step, the results from the calculations on multiple processing elements are combined to produce a global result as per the computational algorithm. As shown in Figure 3, in some cases, the process of combining the results from the partial computations on different processing elements may merely involve the concatenation of the results, while in some other cases, some mathematical operation (*such as*, *sum*, *difference*, *multiplication*, or *division*) might also be required at the time of combining the results.

It is important that the final result from the “collect” step should *closely match* with the result of running the large computation serially using a single core of a processor (without parallelization). An *exact match* of results (from serial and parallel versions) is not expected with double or float values due to rounding off errors.

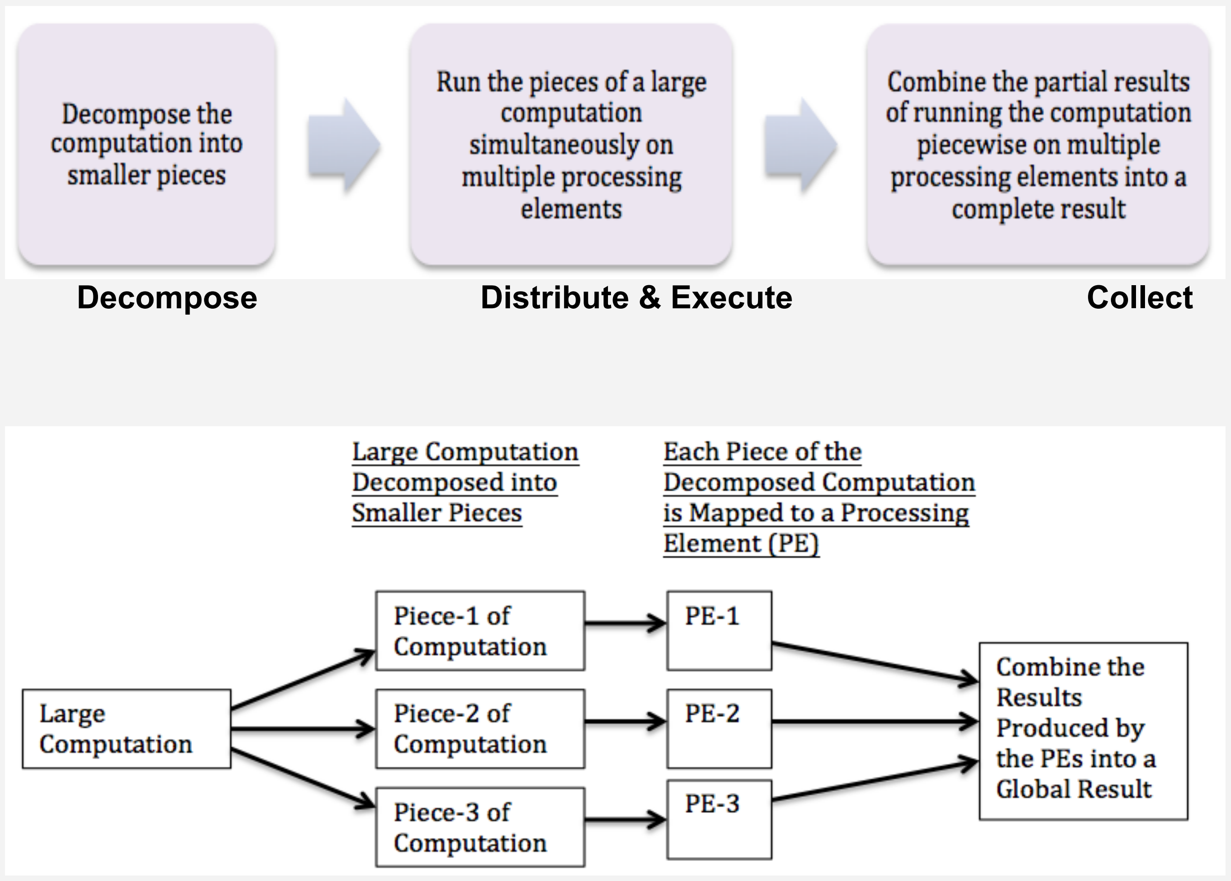
Decompose the computation into smaller pieces
Run the pieces of a large computation simultaneously on multiple processing elements
Combine the partial results of running the computation piecewise on multiple processing elements into a complete result
Decompose
Distribute & Execute
Collect
Large Computation Decomposed into Smaller Pieces
Each Piece of the Decomposed Computation is Mapped to a Processing Element (PE)
Large Computation
Piece-1 of Computation
Piece-2 of Computation
Piece-3 of Computation
PE-1
PE-2
PE-3
Combine the Results Produced by the PEs into a Global Result

Figure 2. The process of parallelizing a large computation

1.3. Parallel Programming in Science, Business, and Humanities

Today, parallel programming is being used to develop software applications catering to a wide variety of domains in science, business, and humanities. It is used for developing applications for *modeling and simulating* complex real-world phenomena like formation of ocean currents and galaxies. Parallel programming is also being used for *exploring*, *processing,* and *analyzing* very large and complex datasets. It is used in *solving* large mathematical problems and in *visualizing* large data collections. The sample applications included in this section demonstrate the growing demand for parallel programming solutions in various domains.

1.3.1. Parallel programming in science

There are several challenging scientific problems - such as simulating gravitational waves, sequencing genomes, and designing aircraft - that cannot be solved in a finite amount of time without using multiple processing elements in parallel.

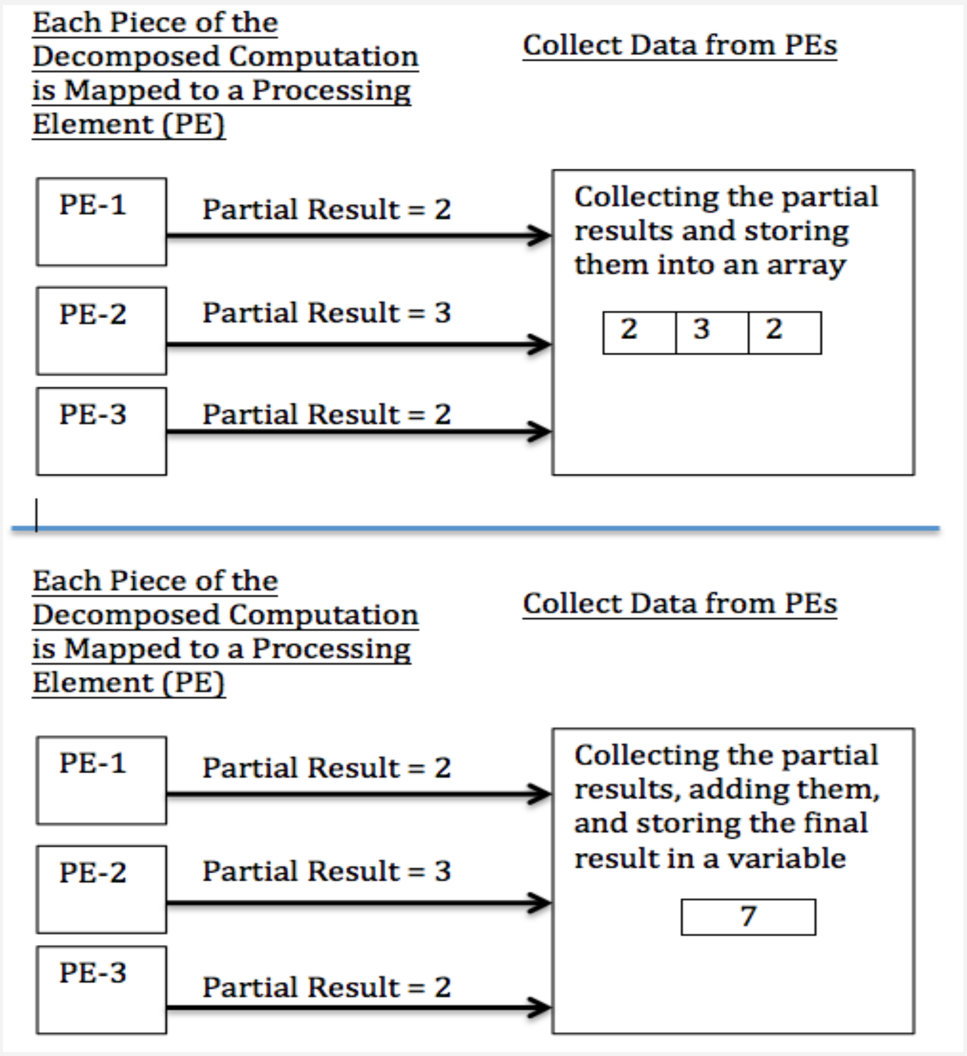
Each Piece of the Decomposed Computation is Mapped to a Processing Element (PE)
Collect Data from PEs
PE-1
Partial Result = 2
PE-2
Partial Result = 3
PE-3
Partial Result = 2
Collecting the partial results and storing them into an array
2
3
2
Each Piece of the Decomposed Computation is Mapped to a Processing Element (PE)
Collect Data from PEs
PE-1
Partial Result = 2
PE-2
Partial Result = 3
PE-3
Partial Result = 2
Collecting the partial results, adding them, and storing the final result in a variable
7

Figure 3. Some variations in the "collect" step - combining results by storing them in a single array, and by adding them together before storing in a variable

1.3.1.1 Studying thermonuclear explosions in astrophysics

A wide variety of topics in astrophysics, such as X-ray bursts, galaxy clusters, and stellar structure, are difficult to study and validate experimentally. Therefore, they are studied using computationally intensive simulations that use and produce large datasets. Due to the large-scale nature of the simulation in terms of the amount of memory and the computations needed, the simulation software cannot be run efficiently using a single processing element. Hence, the simulation software is parallelized to do the computation and the required I/O using multiple processing elements. An example of this would be the *Flash Code* developed by the Flash Center for Computational Science at The University of Chicago, which incorporates several solvers for magnetohydrodynamics, cosmology, etc. [1].

1.3.1.2 Genome sequence search

Genomic sequence search applications are often used in biology for understanding or identifying the functionality of newly discovered sequences, and to establish relationships across the various species having common ancestors. Typically, a newly discovered sequence is compared against a database of known nucleotide or amino acid sequences. Similarities between the new sequence and a known sequence in the database help in understanding the functionality of the new sequence. Because the sequence databases can be very large, the genome sequence search can be a compute-intensive and I/O-intensive activity. Hence, parallel applications for sequence search are developed. These applications use hundreds to thousands of processing elements and perform I/O in parallel to achieve the desired outcomes in a finite amount of time, *e.g.*, MPI BLAST [2].

1.3.1.3 Aircraft design

Fluid flow simulations are critical in the process of aerodynamic design. They help in determining the flight characteristics of an aircraft and in simulating critical conditions in which the aircraft can be flown. Using simulations for aircraft design can help in reducing the risk and cost associated with building and testing the aircraft. There can be several avenues for improving an aircraft's design, and hence, several hundred iterations of simulations are run to find the best combination of the various components required for building the aircraft. Parallel programming can help in simultaneously running multiple iterations for exploring the different aircraft designs, and hence, in reducing the time-to-solution.

1.3.2. Parallel programming in business

Parallel programming is used in the design, development, and testing of consumer products. It is critical for the success of organizations in the financial sector, where quick analysis of the current market trends and historical data can provide one with a competitive edge over others.

1.3.2.1 Modeling the flow of shampoo from bottles

The branch of physics that covers the flow of matter is called rheology, and it impacts the design of consumer products like shampoos, detergents, and body washes. The rate of flow of shampoo from the bottle and its dissolution during use can influence the future buying decision of the customer. Hence, manufacturers develop formulations of their products that have favorable characteristics in the context of rheology. Molecular simulation software is used to understand the mechanisms leading to different rheological behavior. This software runs ensembles of computations using parallel programming.

1.3.2.2 Stock trading

Stock trading software analyzes stock market data, applies trading rules, and recommends transactions. Financial institutions continuously optimize their trading software to improve the quality of recommendations by enhancing the trading rules and improving the speed of the analysis. The trading software is parallelized to compute more results (or recommendations on transactions) in the same time-frame, or spend less time than before to compute a result, so that they can stay competitive [3].

1.3.2.3 Crash test of cars

Structural analysis of cars or the crash testing of cars is critical for meeting the safety requirements for cars. Crash testing of cars in reality is a costly and risky activity. Hence, software simulations are used for this purpose to cut down on cost and risk. However, these simulations are complex and computationally intensive. Hence, parallel programming is used to distribute the computation required for the simulations across multiple processing elements [4].

1.3.3. Parallel programming in humanities

Parallel programming is being used for software development in the digital humanities and social sciences communities too. Scholars from these domains often work with very large sets of unstructured or semi-structured data, containing elements such as, textual documents, drawings, audio-visual content, and images. Parallel programming helps these scholars in exploring, mining, analyzing, and visualizing their datasets in a short time-span.

1.3.3.1 Image processing in archaeology

Parallel programming can enable archaeologists in processing and analyzing archaeological data worth several decades in a short time-span [5]. Specifically, the steps for cleaning, managing, and classifying archaeological datasets are already being carried out using parallel programs. The task of finding duplicate and contextually similar images amongst millions of image files of archaeological artifacts can be accomplished in a matter of days by using computer vision algorithms and parallel programming. Doing this task manually in the same time-frame would be beyond the capacity of a human being.

1.3.3.2 Sentiment analysis from Twitter data and emails

Sentiment analysis means analyzing textual data for understanding the attitude or emotion of the person who wrote it or reads it. The emotions of the people associated with the emails and Twitter feeds to be analyzed can be classified under positive, negative or neutral categories. The main computational tasks in this problem include

extraction and analysis of the keywords and emoticons embedded in textual content in various formats (*e.g.*, PDF, TXT, and HTML).
Parallel programming is used in conjunction with natural language processing, text mining and image processing techniques to reduce the time taken for sentiment analysis of a large set of emails or Twitter feeds.

1.3.3.3 Analyzing census data

Parallel programming can help social science scholars in processing, analyzing, and matching individual records in the census data collected over the last several decades in a finite amount of time. The studies using census data help in tracking population change across several years, and inform the agencies responsible for policy-making. With the help of parallel programming, manually infeasible tasks such as cleansing the data for a particular year and comparing the cleansed data with the data from last several years becomes possible in a finite amount of time.

1.4. Why Learn Parallel Programming?

As evidenced by the sample applications presented in this chapter, parallel programming is gradually becoming indispensable for developing high performance applications that solve large-scale problems in science, business, and humanities. Some of the reasons for learning parallel programming are enumerated further in this section.

1.4.1. Optimally using the latest hardware

The computer architecture discipline has tremendously evolved in recent years. The clock speed of modern processors is not increasing any further because it can result in (1) the increase in the temperature of the processors to unmanageable levels, and (2) the increase in gap between the speed of the processor and the memory, thereby, stalling the execution of the code until the required data is fetched from the memory. Therefore, manufacturers have been increasing the computing density per processor such that today we have multicore and manycore processors equipped with on-chip high-speed memory. Such multicore processors are common in desktop computers and laptops, and the manycore processors are becoming common in servers. While the number of cores in a processor has increased, their clock frequency has decreased. Therefore, to optimally use modern hardware, it is imperative to develop high performance software using parallel programming.

1.4.2. Reducing the time-to-results and cost

In today's competitive world, many organizations need time-sensitive data for their day-to-day operations. By using parallel software that can make optimal use of the underlying hardware, the time-to-results (or the time-to-solutions) can be drastically reduced. As an example, an image processing application for finding similar and duplicate images would take several months to process 400,000 images using a single core in a processor. However, the parallel version of the software can reduce the processing time to a few days by simultaneously using several hundred cores of processors. Hence, using a greater number of processing elements to solve a problem that can be parallelized will shorten the time-to-results, which can further save costs.

1.4.3. Solving large-scale and complex problems

Several computational problems are difficult to solve using a single core on a laptop or a personal computer due to the slow clock speed of the core and the limited amount of memory available on the computer. Parallel programming can be used to solve such large-scale and complex problems by employing multiple independent processing elements having their own independent memory.

1.4.4. Mitigating risks

Parallel software, like the one for simulating crash tests of cars, helps in saving time, money, and lives by conducting the tests virtually in a reasonable time-frame. There are several other complex scenarios similar to the crash testing of cars in which using physical resources for testing is costly and dangerous and hence, simulations are preferred.

1.4.5. Shortfall of people with parallel computing skills

As we get ready for the exascale computing era, and the innovations in the computer architecture discipline continue, parallel programming is the only way to do fast calculations with a short turnaround time. Hence, nations around the globe are ramping up their investments in advanced computing (or supercomputing) resources. For optimally using such advanced computing resources, parallel programming skills are required. However, there is a paucity of people equipped with parallel programming skills, and hence, parallel programming is a highly valued skill.

1.5. Parallel Programming Platforms

Any computer or laptop equipped with multicore processors can be turned into a parallel computing platform. However, for doing serious real-world parallel computations, High Performance Computing (HPC) platforms are used. HPC platforms are also referred to as supercomputers. A photo of a supercomputer is shown in Figure 4.

1.5.1. General overview of parallel programming platforms

An HPC platform consists of multiple nodes (or servers) that are interconnected with a high-speed network. Each node comprises of multiple components like processors, memory, and network interfaces.

Figure 4. Photo of a supercomputer

Photo courtesy: Texas Advanced Computing Center

Depending upon the selected hardware, a node can have multiple processors. The processors in turn can have multiple cores (ranging from two to seventy-two), and each core can be considered as an independent processing element. A high-level overview of the general architecture of an HPC platform is shown in Figure 5.

An HPC system can have shared memory, or distributed memory, or distributed-shared memory. **Shared memory** is the memory that is common to all the cores on a node and can be accessed as a global address space - that is, the changes made in the memory by the code running on one core are visible to the code running on other cores of a node. The architecture of shared memory can be further classified as Uniform Memory Access (UMA) or Non-Uniform Memory Access (NUMA). As shown in Figure 6, with UMA architecture, all cores have the same memory access time whereas with NUMA, different cores have different memory access times depending upon their proximity to the memory regions.

Distributed memory refers to the interprocessor memory that can be accessed through a communication network that is common to all processors. As shown in Figure 7, processors have their own local memory and address space. Hence, each processor can work independently of other processors in the HPC system. **Processors can access data in the memory of other processors with the help of specific function calls written by the programmer.**

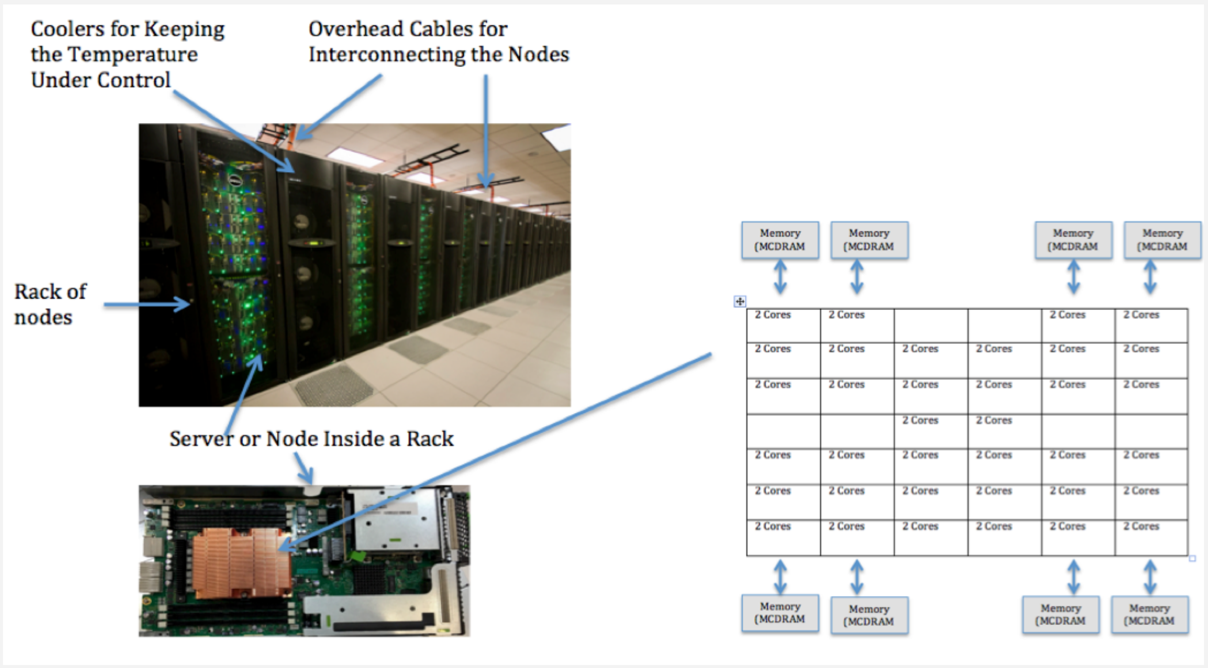
Coolers for Keeping the Temperature Under Control
Overhead Cables for Interconnecting the Nodes
Rack of nodes
Server or Node Inside a Rack
Memory (MCDRAM
2 Cores

Figure 5. What is inside an HPC system?

Distributed-shared memory refers to the memory architecture that is a hybrid of shared memory and distributed memory, and is the most common type of memory architecture in modern HPC platforms. It is built by connecting shared memory nodes with a high-speed connection network.

HPC systems are made usable with the help of a supporting software stack that consists of the filesystems, job schedulers, and parallel programming libraries.

1.5.2. Accessing HPC resources

Although, for the purpose of learning parallel programming, one can install the required libraries and compilers on a laptop or a desktop computer, the actual performance gain by parallelizing an application will only be visible by running it on a large-scale system. Amazon Web Services (AWS), and Google Cloud Platform can be used for accessing HPC resources including GPUs in the cloud from anywhere in the world at a cost. The researchers and scholars based in the United States, Europe, Canada, and Japan can get free access to supercomputing resources by requesting their respective national open-science data centers [7].

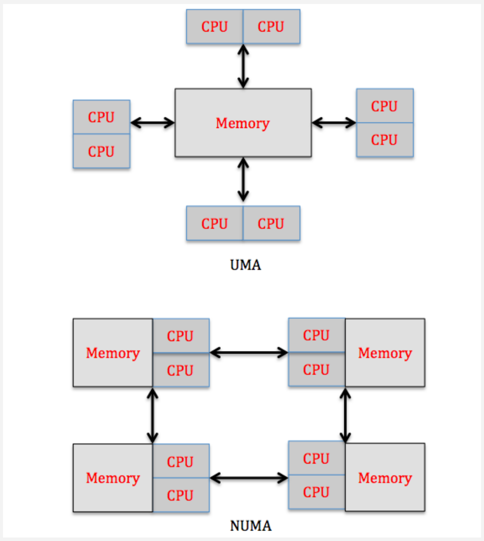

Figure 6. Shared memory architecture - UMA and NUMA

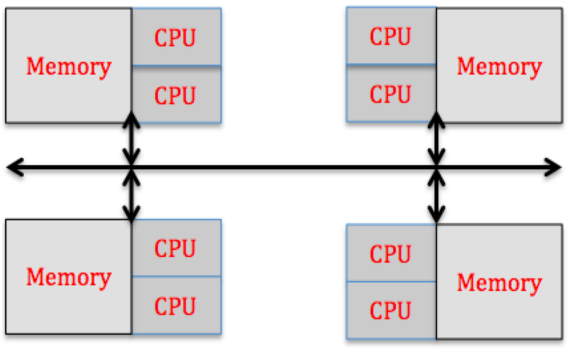

Figure 7. Distributed memory architecture

1.6. Common parallel programming models for leveraging HPC platforms

There are two modes of developing parallel programs - implicit and explicit. In the implicit parallelization mode, a programmer selects a parallel programming language and writes code from scratch. Parallelism is inherent in the programming construct used and the low-level details of how the parallelism is actually implemented in the programming language are hidden from the user. Programming languages like Fortress and X10 are used for implicit parallelization.

In explicit parallelization mode, programmers reengineer their existing serial applications, typically written in C/C++/Fortran and lately Python, to insert specific library calls or API for parallelizing their code. The user has to do a lot more work in explicit parallelization (*viz.*, decomposing the problem domain, and synchronizing the calculations across different processing elements) as compared to the implicit parallelization, but has more control on the way the parallelization is implemented in their code as compared to implicit parallelization. **The three most popular paradigms for explicit parallelization are OpenMP, Message Passing Interface (MPI), and CUDA.**

OpenMP [8] is a multi-threaded parallel programming paradigm that is used for developing parallel programs for shared memory architectures. **MPI** [9] enables message passing for interprocess communication and is used for developing programs for distributed and distributed-shared memory architectures. Using OpenMP and MPI together results in a **hybrid program** that can take advantage of distributed-shared memory architectures, such that the cores inside a node communicate with each other via reading and writing to a shared memory, and they communicate with the cores on other nodes via message passing. **CUDA** [10] is also a multi-threaded parallel programming paradigm that can be used for developing programs that can run on Graphical Processing Units (GPUs).

1.7. Summary

Software applications can be parallelized to reduce the time-to-results. Writing software applications that have large computations decomposed into smaller units of computation, such that, these smaller units can be processed simultaneously using multiple processing elements is called parallel programming.

Examples of Parallel Applications	***Science***: astrophysics, genome sequencing, aircraft design ***Business***: flow of shampoo from bottles, stock trading, crash test of cars ***Humanities***: image processing, sentiment analysis, census data analysis
Types of Parallel Programming Platforms	***AKA***: HPC platforms or supercomputers ***Memory architecture***: shared, distributed, distributed-shared ***Publicly accessible***: AWS, Google Cloud Platform, supercomputers at open-science data centers
Types of Parallel Programming Models	***Implicit***: Fortress, X10 ***Explicit***: OpenMP, MPI, CUDA

REFERENCES

1. FLASH astrophysics code: http://flash.uchicago.edu/site/flashcode
2. MPI BLAST: http://www.mpiblast.org/downloads/pubs/sc2008.pdf
3. Janko Strasburga, Christian Gonzalez-Martelb, Vassil Alexandrov, "Parallel genetic algorithms for stock market trading rules", *Procedia Computer Science*, vol. 9, pp. 1306-1313, 2012.
4. HPCWire article: https://www.hpcwire.com/2014/01/14/crash-testing-scale/
5. Ritu Arora, Jessica Trelogan, Trung Nguyen Ba, "Using High Performance Computing for Detecting Duplicate, Similar and Related Images in a Large Data Collection", Conquering Big Data with High Performance Computing, ISBN : 978-3-319-33740-1, 2016.
6. Ritu Arora, Maria Esteva, Jessica Trelogan, "Leveraging High Performance Computing for Managing Large and Evolving Data Collections", IJDC 9(2): 17-27 (2014).
7. XSEDE: https://www.xsede.org/
8. OpenMP: https://www.openmp.org/
9. MPI: https://www.mpi-forum.org/
10. CUDA: https://developer.nvidia.com/cuda-zone