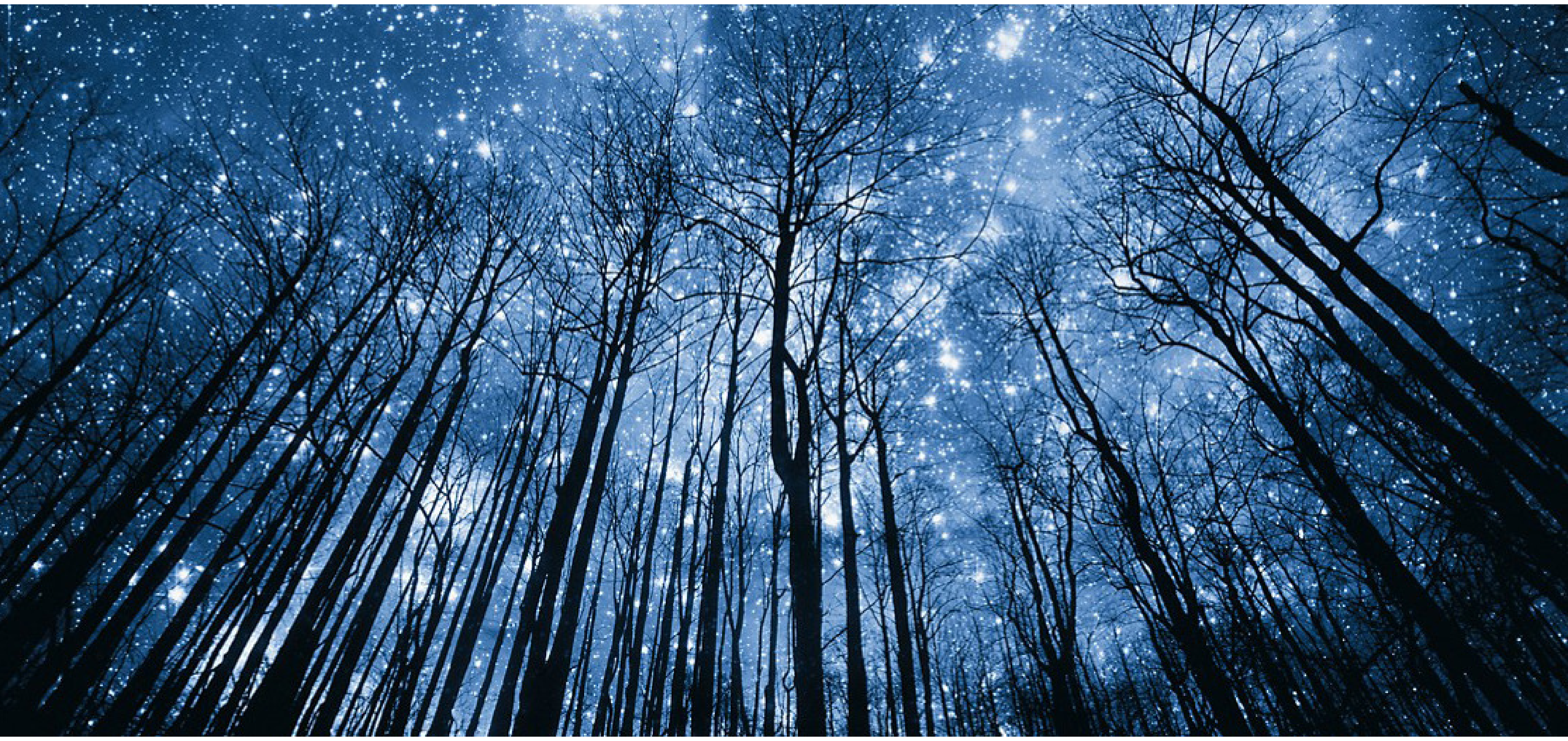
# HPC FOR BIO-INSPIRED OPTIMIZATION ALGORITHMS



Rajasekhar Nannapaneni
Senior Principal Site Reliability Engineer
Dell Technologies

Proven Professional

The Dell Technologies Proven Professional Certification program validates a wide range of skills and competencies across multiple technologies and products.

From Associate, entry-level courses to Expert-level, experience-based exams, all professionals in or looking to begin a career in IT benefit from industry-leading training and certification paths from one of the world's most trusted technology partners.

Proven Professional certifications include:

- Cloud
- Converged and Hyperconverged Infrastructure
- Data Protection
- Data Science
- Networking
- Security
- Servers
- Storage

Courses are offered to meet different learning styles and schedules, including self-paced On Demand, remote-based Virtual Instructor-Led and in-person classrooms.

Whether you are an experienced IT professional or just getting started, Dell Technologies Proven Professional certifications are designed to clearly signal proficiency to colleagues and employers. Learn more at www.dell.com/certification

## Table of Contents

## Introduction:

A supercomputer is generally considered to be a massive computational device that consists of high specifications of processing power packed into a single machine. While there are physical and technological constraints on scaling a single machine, evolution in computing led to enhancement towards grouping multiple massive computational machines that can work together to solve a problem and this scalable architecture is called high performance computing which can solve advanced computational problems.

In the initial sections of this article, a detailed coverage on computing and parallel computing is discussed. This also leads to the details of necessity that led to high performance computing.

Whenever there is an application on a computer, there is usually a numerical algorithm behind it or at least we restrict ourselves to numerical algorithms in computational science. It turns out that in computational science there are many standard algorithms. As per one of the research projects from Berkley [2], most of computational science codes comprise one or more of the following seven basic families or basic motives of computing which were termed as seven dwarfs.

- Dense Linear Algebra - deals with matrices or multi-dimensional structures
- Sparse Linear Algebra - in which most entries are zero
- Spectral Methods - Fourier transformations for example to solve differential equations
- N-body methods
- Structured Grids
- Unstructured grids
- Monte Carlo Methods

In the later sections of this article, one of the bio-inspired optimization algorithms called particle swarm optimization is discussed to detail the computational needs of meta-heuristic algorithms. Details are also discussed on how parallel computing using graphical processing units and high-performance computing can improve efficiency in executing these nature inspired algorithms.

**Computing:**

At the heart of every computer, we are using the stored program computer concept which is shown in Figure 1. It was invented in the 30s of the 20th Century and it comprises a collection of components. There is a control unit and arithmetic logic unit which does the actual work. It is the purpose of the control unit to read instructions from memory and these instructions are encoded as simple numbers usually organized in bytes or words or quad words. After interpreting those instructions, the arithmetic logic unit obtains data from memory and performs operations on that data. More data is then written back to memory and of course we need some I/O hardware to get the results.

Out of the system, nobody has produced a more powerful and more flexible concept. Starting from the first computers that used the stored program computer concept like the EDSAC in 1949 to typical home computer and business computer systems in the 70s - 80s and even super computers, iPhone all of those use the stored program computer concept.

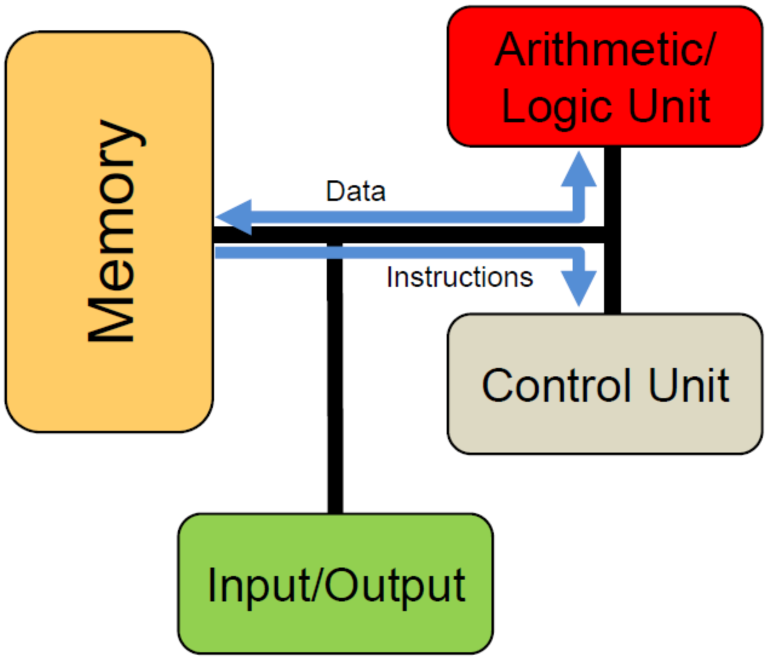


*Figure 1: Stored Program Computer*

The main performance limitation, the thing that usually limits the performance of executing programs on such a computer is memory access. Memory is usually something that can be obtained or can be built in very large quantities, however compared to anything that happens within the core, within the arithmetic logic unit and the control unit, memory tends to be slow. This is because it is connected via wires and wires are much slower than connections on the chip.

The main purpose of a computer is instruction execution. Instructions are the primary resource of the processor. When a computer architect designs a new processor, their main goal is to make instruction execution as fast as possible.

So, processor designers use instructions as their main metric of work, however programmers as developers in scientific computing do not care about instructions. They care about actual work and often actual work is FLOPS or any other metric that is relevant in their context of algorithm.

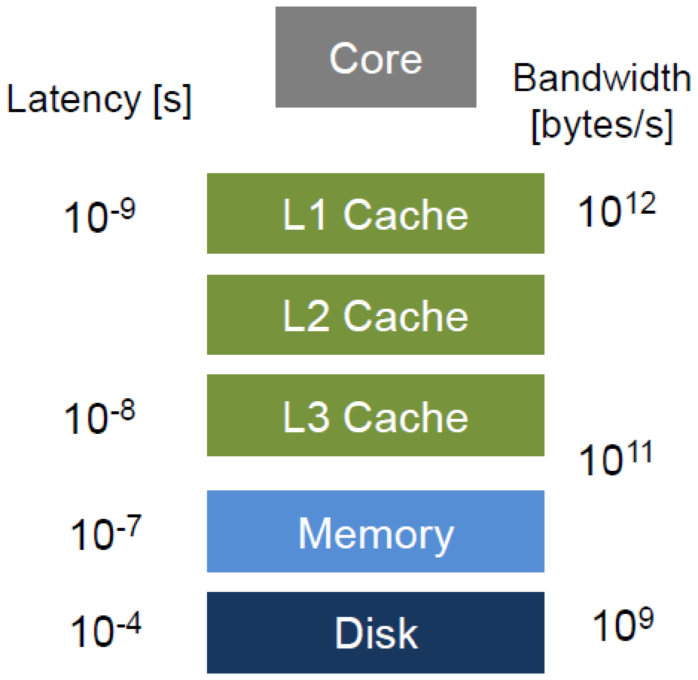


*Figure 2: Cache Levels*

Of course, executing instructions alone does not help to get problem done, it is the secondary resource that makes the computer work, and the secondary resource is a consequence of instruction execution, and this is data transfer. How fast data can be transferred is determined by the maximum bandwidth the system can provide.

As data transfers are the number one limiting factor in computing, if anything can be done to make program move less data than the chances are that it will get faster. The main insight is that main memory is too slow to keep up with the CPU's hunger for data. To mitigate this, caches have been invented. We can either build a small (with the order of tens of kilobytes) and fast memory or a large (with the order of tens or hundreds of gigabytes) and slow memory.

So, the caches are a compromise and usually we have several levels of those in modern server CPUs as shown in figure 2. They are three levels normally and they are used to hold data which is often used as close as possible to the CPU. The smaller the cache, the faster it is. The outermost L3 cache is often of the order of tens of megabytes, L2 cache is a couple of hundred kilobytes or a megabyte and L1 cache is 32 or 64 kilobytes.

**Parallel Computing:**

Parallel computing means that we find a way to map a numerical algorithm to the hardware of a parallel computer.

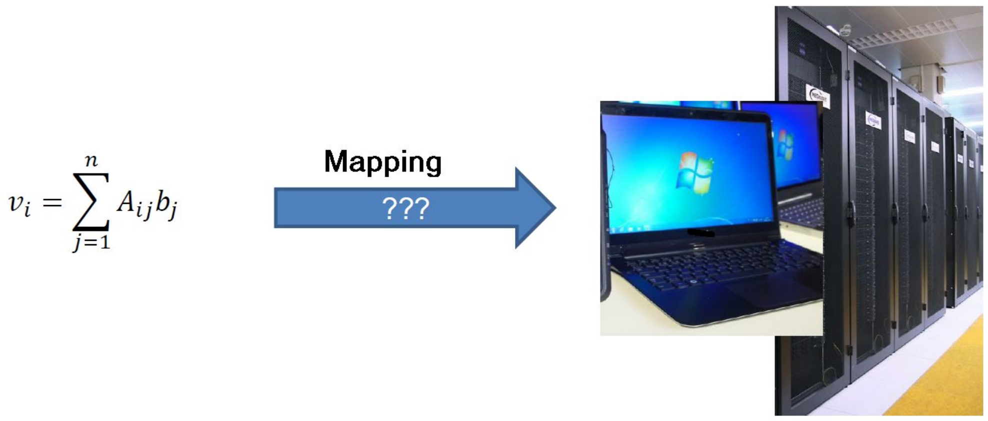


*Figure 3: Mapping numerical algorithm to parallel computer*

A simple numerical algorithm is shown in the figure 3. The multiplication of a dense matrix or some matrix with a vector which yields another vector seems simple enough, but it turns out that this seemingly simple operation has a surprisingly rich phenomenology in terms of performance.

Every laptop now a days is a parallel computer with at least two or four cores, and we could have a supercomputer with thousands or hundreds of thousands of cores.

The algorithm shown in figure 3 can run in parallel in all these systems so it is users' task to find out how to do that and how to do that fast and effective. Doing something fast and doing something effectively may not be the same thing in parallel computing.

It is difficult to perform parallel computing because parallel computers nowadays have a very hierarchical structure with layers upon layers of hardware units that needs to be known and addressed when doing parallel computing.

To start with the place where the actual work is done, if it needs multiplying a matrix with a vector as shown in figure 3, then a part of the problem requires multiplications and the accumulation. Another part of the problem requires the work with the arithmetic, and this is done in the executional units of each computational core as shown in figure 4.

So, there is one core which has execution units that could do single and double precision multiply, add, subtract, and divide. There are registers which are fast places to hold data and there may be multiple levels of cache in which the hardware keeps often used data for future reference.

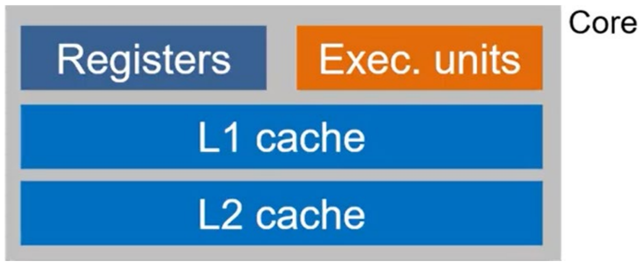


*Figure 4: Single CPU Core*

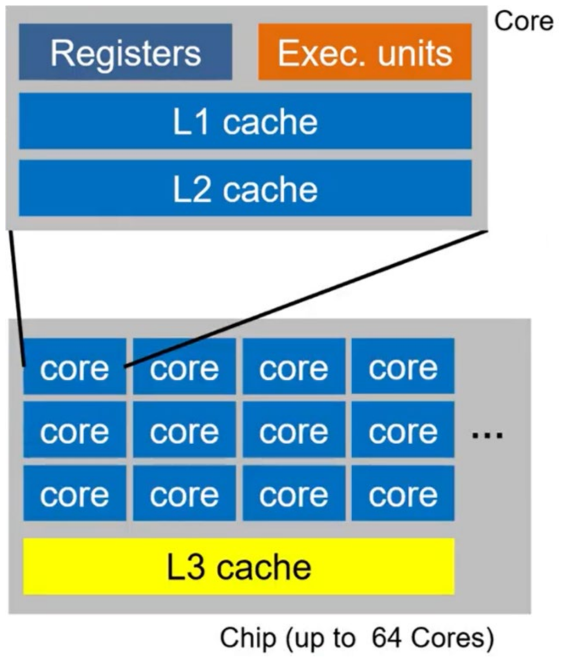


*Figure 5: Socket and cores*

This is a what is called as a core, and this is the basic unit computer is made of. Several of such cores are typically put onto a die or a chip and nowadays in the commodity server market in HPC we have up to 64 cores on such a chip or socket. These are usually put together not by themselves but also using a shared L3 cache by which those cores can communicate with each other.

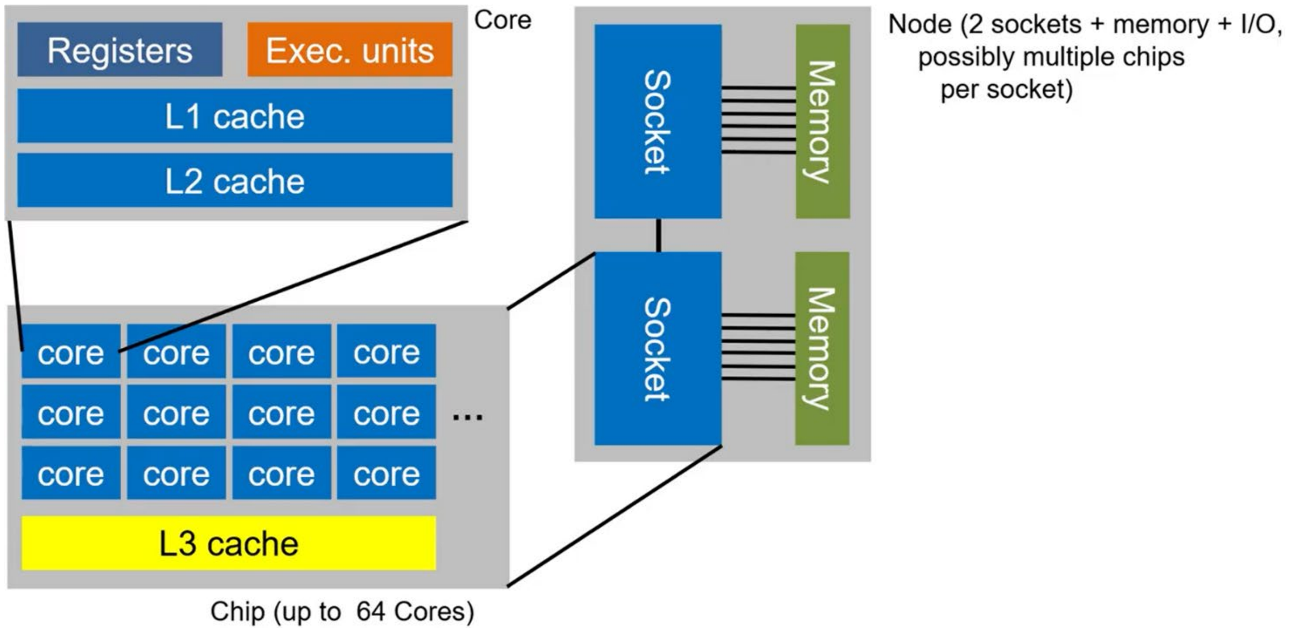


*Figure 6: Node, Socket and Core*

Each chip has a couple of billion transistors nowadays and it needs some infrastructure to run like memory interface, memory DIMM, some I/O and this infrastructure we call a node.

In high performance computing the price performance sweet spot is usually a node with two such chips or as we call them sockets. The socket or the package is the thing or the processor we get from Intel or AMD. The socket is attached on the motherboard along with and in high performance computing typically there are two sockets on the board, and this is called a node. It is called a node because one operating system instance is running on such a node.

Typical in high performance computing, a node is quite space efficient with two CPUs along with their cooling infrastructure and the memory DIMMs are at the side of the CPU's. There are also fans to blow cool air through the system and some network infrastructure is available to provide connectivity.

It is this part of the structure that users address when they do Open MP programming. Open MP cannot leave the node level because it requires shared memory and that is what is available at this level. If more compute power is required, then many of such nodes are put together using a high-performance network and this is then called a parallel computer or supercomputer.

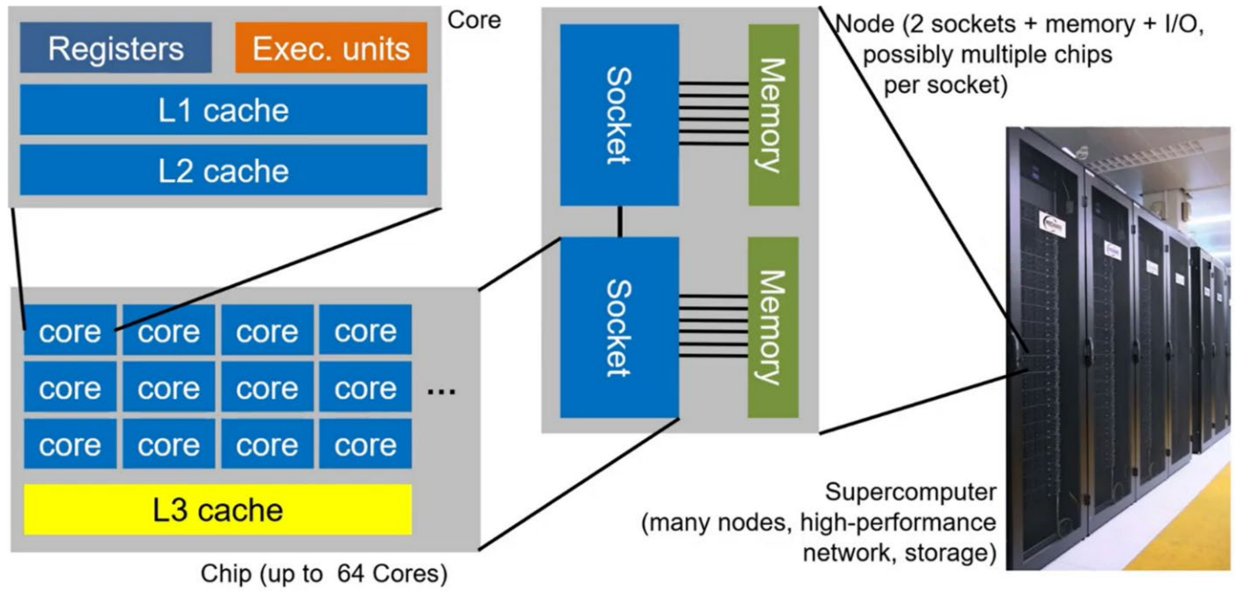

Registers
Exec. units
Core
L1 cache
L2 cache
core
...
L3 cache
Chip (up to 64 Cores)
Socket
Memory
Node (2 sockets + memory + I/O, possibly multiple chips per socket)
Supercomputer (many nodes, high-performance network, storage)

*Figure 7: HPC, Node, Socket & Core*

There will be many nodes which are connected using a high-performance network and of course they need some storage because usually in high performance computers there are minimal hard disks used on the nodes. A hard disk is a major point of failure and the fewer hard disks you have in the compute system the better.

**HPC and its applications:**

High-performance computing can process data and perform complex calculations at high speeds. To put it into perspective a laptop or desktop with a 3 gigahertz processor can perform around 3 billion calculations per second. While that is much faster than any human can achieve, it pales in comparison to HPC solutions that can perform quadrillions of calculations per second.

HPC uses parallel processing for running advanced application programs efficiently reliably and quickly.

Here is a short survey about applications that high performance computing is used for:

- **Weather and climate prediction:**
    - There is a huge amount of computing power used every day to compute tomorrow's weather and it is successful

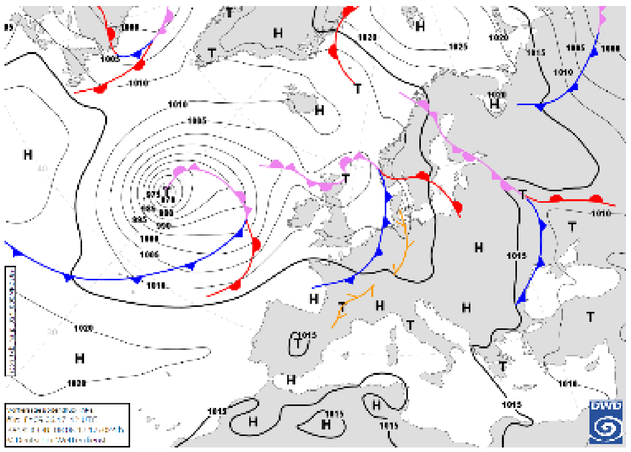


*Figure 8: Weather Prediction*

- **Drug design:**
    - New medications especially in the current situation, a lot of cycles are spent designing or trying to design vaccines or drugs that help against the pandemic
- **Simulation of biochemical reactions:**
    - A major part of compute cycles in some of the supercomputers around the globe are spent with molecular dynamics codes and quantum simulation codes that try to figure out reactions between complex molecules

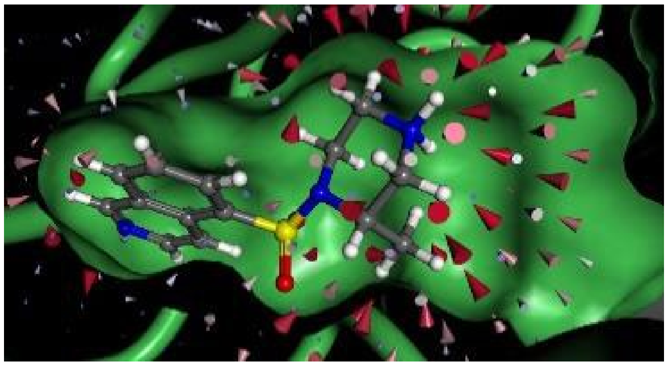


*Figure 9: Simulation of Biochemical Reaction*

- **Processing analysis for measurement data:**
    - By detectors or by measurement devices in orbit
- **Properties of condensed matter:**
    - Some of the compute cycles of supercomputers along with quantum systems are used to study condensed matter
- **Fundamental interactions and structure of matter:**
    - Lattice Quantum Chromo Dynamics to explore the fundamental structure of matter also requires a lot of computer power

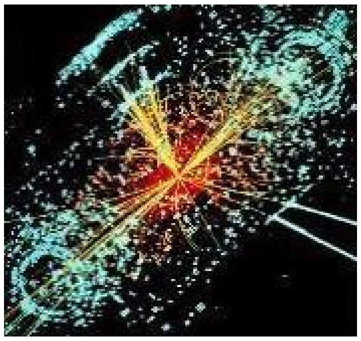


*Figure 10: Structure of Matter*

- **Fluid simulations, structural analysis, fluid-structure interaction:**
    - Requires high compute power
- **Mechanical properties of materials:**
    - This can also be using simulations requiring high computational power
- **Rendering of 3d images and movies:**
    - In 1997, Titanic was the first movie in which the CGI scenes were rendered on Linux cluster systems

*Figure 11: 3d image rendering*

- **Medical image deconstruction:**
    - Generally, a CT scan or an MRI scan requires some high-performance hardware involved that converts the raw measurement data from x-ray detectors and the MRT device to an actual image

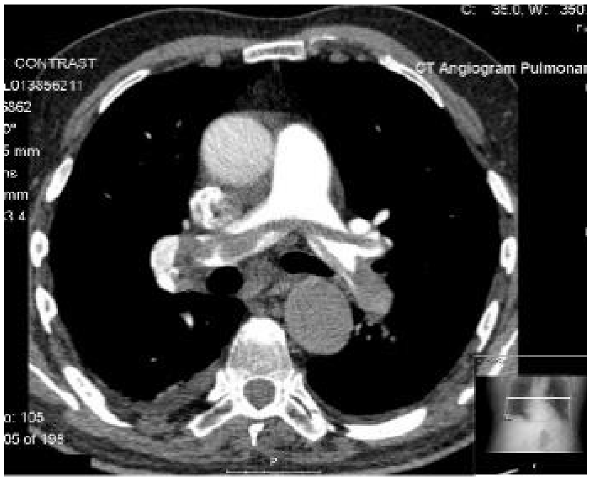


*Figure 12: Medical Image reconstruction*

## Performance of HPC:

Linpack is a well-defined performance benchmark that is used to rank supercomputers worldwide. Linpack solves a dense system of linear equation shown in figure 3, solves this for x. The size of the system to solve is unspecified.

So, in practice to get best performance, the nodes of the supercomputer are filled with as much data as possible. This means the matrix whose dimension is the dimension of the system is made large enough so that almost all the memory is filled, and one can show mathematically that in this limit the communication overhead between the parts of your system plays a minimum role.

This linpack is run and its output is $R_{max}$ which is the achieved floating-point performance of the linpack algorithm. It is measured in FLOPS which is floating point operations per second. This Linpack is used to rank top 500 supercomputers twice every year. To put these things into perspective, in the first list which was published in 1993 the number one was a connection machine CM5 with 1024 processors.

At that time there was no multi-core thing so, one processor was one core, one chip and one socket. This machine had a performance $R_{max}$ for linpack of 60 GigaFlops which is 60 billion floating point operations per second.

In the current list Nov 2022, the current number one is the Frontier system which has 8.7 million processors or to be exact 8.7 million cores and $R_{max}$ performance of 1102 PetaFlops.

**First: 1993 (#1: CM5 / 1024 procs.): 60 Gflops/s**

**Nov 2022 (#1: Frontier / 8.7 mi procs.): 1102 Pflops/s**

Based on these numbers, it quite some ratio that there is a performance increase over those 30 years of approximately 79 percent per year from 1993 to 2022.

According to Moore's law the complexity of the number of transistors on a chip double about every 24 months. If Moore's law is compared with this observed increase in performance of HPC, we can notice that HPC performance grows faster than Moore's law and this is because over the years parallelism has caused this acceleration.

So, Moore's law was well and alive up to now at least but the increase in parallelism added another factor on top of it so the list performance grows faster than Moore's law.

So far, the performance of HPC is discussed as performance metric for linpack but in general terms performance is the ratio of work divided by time.

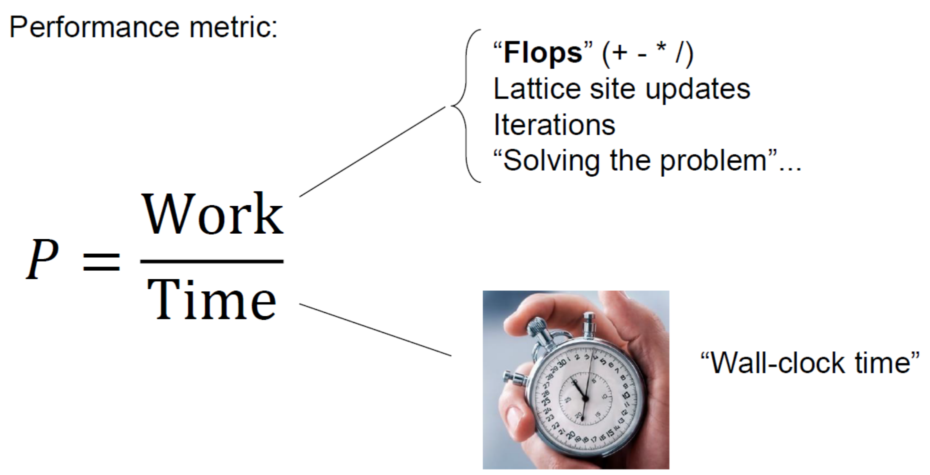


*Figure 13: Performance Metric of HPC*

Here time is simple, it means the wall clock time, but work is a little bit more complicated and diverse. Work can mean FLOPS which are operations such as multiply, add, subtract, divide and probably more complicated stuff. It can also mean other things depending on what was being done for example if the task is on structured or unstructured grids then the concept of work may be a lattice site updates as here data is being updated on lattice sites. It could be an inner iteration of a specific problem solving. So, performance metric is just one over time all these are viable options for work.

**Flop == Floating-point operation (add, subtract, multiply, divide)**

**Flops/s == "how many flops can be done per second?"**

Flops per second metric can mean two concepts, one you can use it to characterize a computer on how fast a computer can possibly compute? That is, how many flops per second can it do, and it is called the peak performance and the other meaning are how fast does it solves a given problem using a given program?

The peak performance depends on many things and especially depends nowadays on the accuracy of the input operands as there's double, float and half precision. This is because the float performance which is the single position performance is twice as high as the double position performance in modern processors. When looking at peak performance of processors then often the divides are neglected because divides are very slow.

Following are some double-precision peak numbers for perspective:

- Top 500 Supercomputers range: 2.3 Pflops/s – 1685.65 Pflops/s
- Modern multicore server CPU: 2.3 Tflops/s
- Personal Computer: 100 – 500 Gflops/s
- Cellphone: 5 – 50 Gflops/s

## Power Consumption of HPC:

HPC systems consume a lot of power and if linpack is ran for peak performance then it is a hot code drawing a lot of power. The cost of electrical energy is significant in some countries and if you compare the cost of electrical power over lifetime (5-6 years) of a supercomputer will approximately be equal to the investment that is spent for procuring the system in the first place. This is a major insight and it is important to know that that the running system costs are about as much as buying the system.

These numbers do not even include the cost for cooling. This cooling overhead will be about another 5 - 150 percent.

To a give a perspective, the top three of the current top 500 lists consumes power about 21 Mega Watt, 29 Mega Watt and 6 Mega Watt, respectively. Power is a major insight and should be considered because it is also an environmental issue which depends on the amount of electrical energy we burn in computers. For production usages there are some power efficient systems that can be leveraged for computing.

## Emergent Complexity:

Emergent complexity is a theory that describes how some complex systems arise. It is "a phenomenon whereby larger entities arise through interactions among smaller or simpler entities". Essentially emergent complexity requires two things:

- Fundamental units in a system that have some simple behavior
- An external force that drives those things to cooperate in some way

This results in some complex behavior from an optimization process.

A few examples could be hexagonal bonding of ice, ice bonds into hexagonal. Here dendritic growth happens which allows the ice crystals to move along preferential axes in the crystal, so they grow along preferential crystal axes and get developed into complex and interesting structures that we call snowflakes. Another example is an ant bridge where ants just can link together to allow them to get food from areas.

This natural selection is interesting because it generates algorithms that are good for computer science. Life itself is just a series of emergent complexities layered on top of each other as formation of proteins, formation of tissues and formation organs all appear to be naturally happening in a layered fashion. Emergent complexity is not fundamentally natural phenomenon.

Complex systems are not necessarily useful but there is something interesting about the way complexity arises in natural systems. Complexity in nature is expensive from an evolutionary level. If there is a complex system in nature and a simple system in nature that are both solving the same problem with the same effectiveness, the simple one will win purely because it uses less energy to achieve that. Natural selection will dictate that the simpler one wins.

Swarm intelligence is a useful subset of systems that exhibit emergent complexity. Natural systems are good candidates for this because they must solve some sort of problems. Another reason natural systems have potential because problems in computer science sometimes line up incredibly well with things that we must solve in real life. An example is path finding which is like some natural systems trying to find food in some unknown field is the same as trying to find the minimum value for function in some unknown search space.

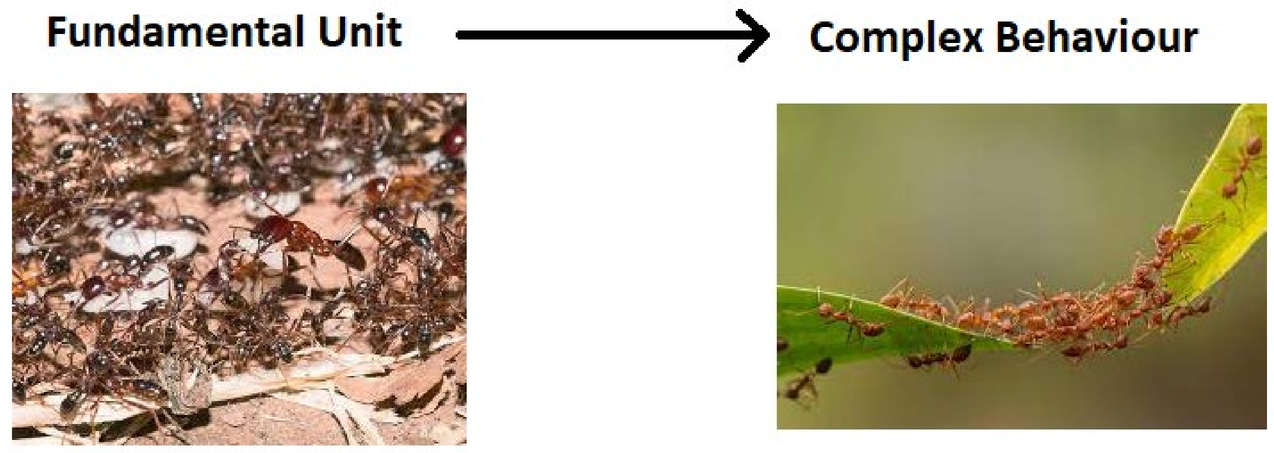


*Figure 14: Ant Army and Ant Bridge*

Lastly nature has inherently high distributivity which can be observed when a series of organisms move together. Ants in large groups do not become more intelligent as they increase in size of the group. Each ant only has some finite capacity to think about its surroundings and so any algorithm that ants must produce to solve their problems is something that we could use in a distributed system to achieve as shown in figure 14.

Not all problems can be solved by something in nature and not all things in nature can map one to in computer science.

**Natural Phenomenon: Bio-Inspired:**

Bird Flocking can be observed during sunsets. As shown in figure15, thousands of starlings moving through space. They are solving an optimization problem which is they are trying to minimize their energy use in traveling through space and it is quite a hard optimization problem. There's dynamic flow of air going, the starlings themselves are simple and not so intelligent but are trying to solve a problem with limited resources.

This bird flocking triggers a question, can we achieve something like this with a simple computer system to solve a problem by accurately simulating this behavior. Instead of having a complicated system, solving a complex problem with simpler emergent way seems to be right approach.

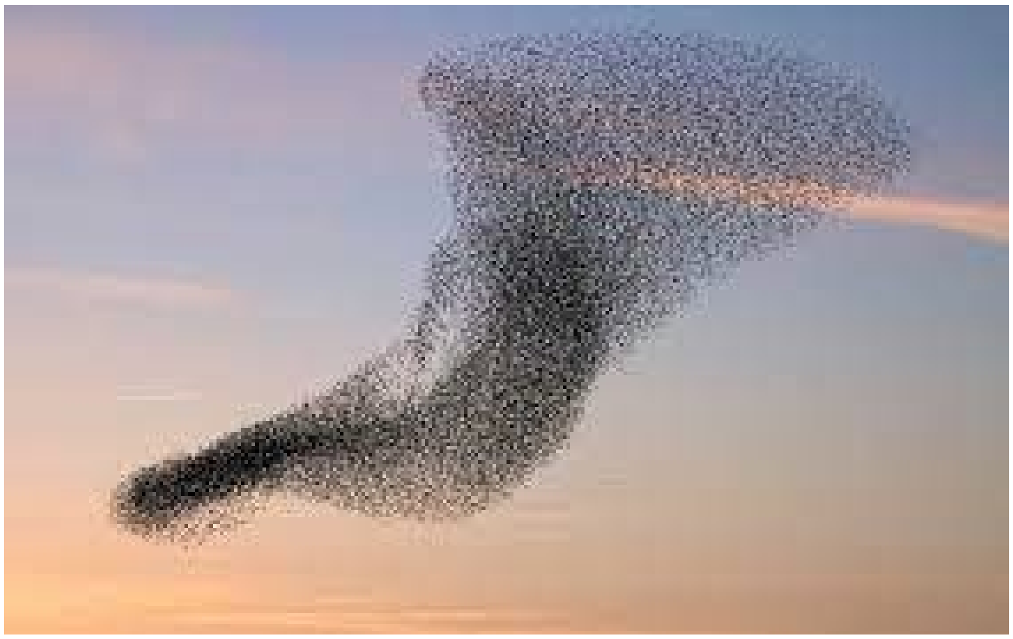


*Figure 15: Bird Flocking*

One of the ways could be to develop a simple model of bird-like objects called BOIDS where each BOID is defined by its position and its velocity. Each BOID also has some total awareness of everything that is happening around it in a specified sphere where it can see everything that is happening but can see absolutely nothing outside that sphere.

The way the simulation can work is that at each point a time step is evolved where each part moves by some amount corresponding to its velocity vector, then it looks around in the vicinity around it within the sphere defined and it considers changing its velocity vector by some amount. The time steps keep evolving and at each simulation each bird is going to change its velocity according to behaviors that are defined.

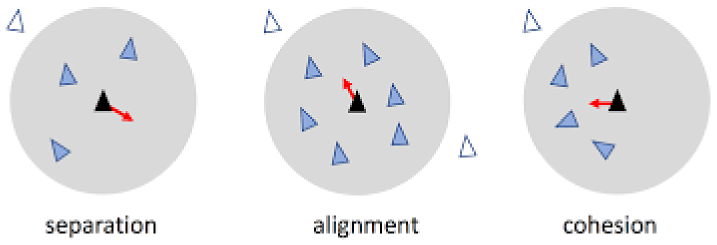


*Figure 16: Flock Behaviors*

Based on how we want our system to behave, we will try to derive some behaviors shown in figure 16:

- **Separation**: In real-life, BOIDS cannot fly through each other, nor do they very often hit each other because that is expensive and from an evolutionary point of view. Each bird will try to fly away from any birds that are very close to it within its own sphere, it's kind of a smaller sphere that it's better cares.

- **Alignment**: If closely observed, birds tend to agree about the direction that the flock is moving in at that local level. However, at a high-level observation, the birds are all moving in complex manner and kind of a very different but subtly different direction. So, a bird cares about moving in the average direction of the birds in its vicinity to some extent.

- **Cohesion**: The flock moves together as a group and in real life any stragglers in a group suffer massively more drag as they move through space. So, the birds have some incentive to keep together and so a bird has some extent to which it wants to move towards the center of gravity of the flock of the birds that it sees in its vicinity.

## Optimization Problem:

The classical solution to an optimization problem is to take approximations to the way the function is evolving within a local vicinity and move in the direction that corresponds to the way gradient descent moves.

Another solution that is kind of canonical form of a class of algorithms that solve this problem is simulated annealing. This is a more stochastic method that tries to randomly converge on the true solution.

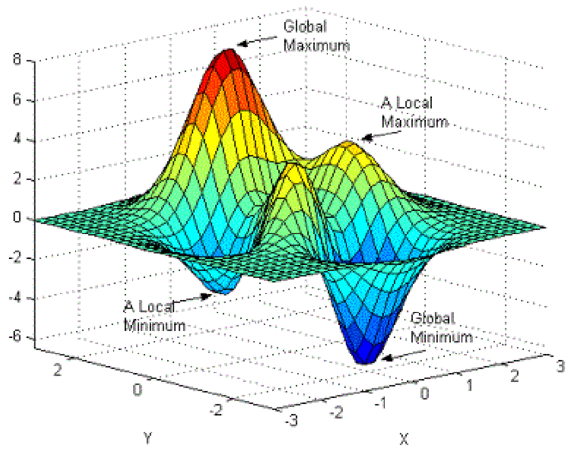


*Figure 17: Local Minima and Maxima*

Both solutions suffer from local minima in that they will return local minima as if they were a global minimum as shown in figure 17 and so they have pathological cases that can be constructed. Since the algorithm that we are deriving is a more kind of random inspired process that is inspired from nature, we are expected to avoid some of those pathological cases.

## Particle Swarm Optimization:

Particle swarm optimization or PSO is a population-based stochastic optimization algorithm motivated by intelligent collective behavior of some animals such as flocks of birds or schools of fish.

Particle swarm optimization is a simple idea where flock of birds or BOIDS are considered and are moved into search space for the function. Some new behaviors are provided to BOIDS that corresponds to them wanting to move in the direction that minimizes the function.

Behaviors that are instilled into BOIDS are:

- **Convergence**: steer towards the best of the vicinity
- **Separation**: delegate the search space among the BOIDS
- **Alignment**: local agreement about the best movement
- **Cohesion**: prevent the flock from scattering

## Mathematical Form of PSO:

The mathematical form is formulated to describes behaviors created. Vectors of length n are considered where n is the dimensionality of the search space. The BOIDS are initialized with random uniformly distributed random components from 0 to alpha where alpha is some parameter that is picked. The introduction of random numbers here helps to do the magic of avoid pathological cases.

$$\Delta v_{pso} = \beta\, u_1(b_{LOCAL} - x) + \gamma u_2(b_{VICINITY} - x) \tag{1}$$

v: velocity

x: position vector

$u_1$ & $u_2$: $vectors\ of\ samples$

$b_{LOCAL}$: best fitness achieved by this particle

$b_{VICINITY}$: best fitness achieved by particles in the vicinity

$\alpha, \beta$ & $\gamma$: are the hyper parameters

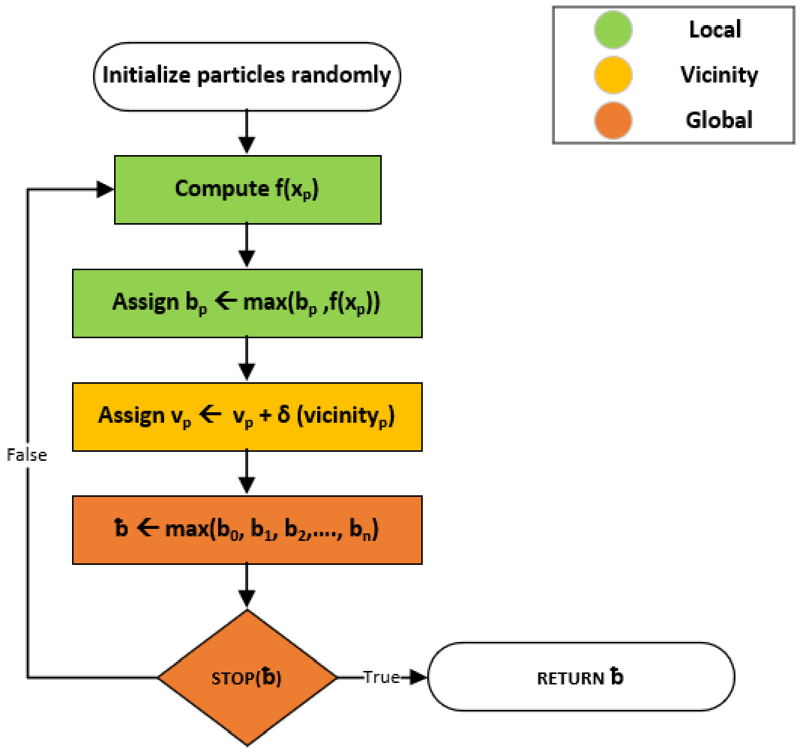
The effect of the new convergence behavior is defined in equation 1 and it has two terms. Each bird is now going to be tracking over time what the best position it is ever found was. As it moved through space it will keep track of its best position. At each point, it is going to want an extent to move towards that best position.

It is also going to ask all the birds in its vicinity what their best positions are and then it will contribute some factor towards moving towards that position as well. There are three parameters in equation 1 where alpha describes the amount of randomness, beta and gamma define weightages of local best found as far as opposed to its neighbors, respectively.

**Algorithm of PSO:**

At each time step, each particle computes the value of the fitness function at its current position and then it compares that value to its previous best and if it is greater, updates its current best as shown in flowchart 1. Then it changes its velocity in some way and in equation 1, the Delta function is just considering the sum of all the behaviors that are captured prior as per the model.

Once the velocity is updated, the system says asks each bird what its best is and takes the maximum of those best values. At this point, the best value is evaluated against our precision requirement and accordingly stop or repeat the process.


Local
Vicinity
Global
Initialize particles randomly
Compute $f(x_p)$
Assign $b_p \leftarrow \max(b_p, f(x_p))$
Assign $v_p \leftarrow v_p + \delta(\text{vicinity}_p)$
$ƀ \leftarrow \max(b_0, b_1, b_2, \ldots, b_n)$
STOP(ƀ)
True
False
RETURN ƀ

*Flowchart 1: Flowchart of PSO*

**PSO as a Distributed System:**

So far, the work that's being done here is being done at a local level where the birds are evaluating their own fitness functions and comparing that to their previous best, that's entirely local and then it must ask its vicinity which is some small subset of the birds and then this global behavior of calculating the maximum is performed.

Instead of doing the computation this way, we could calculate several iterations (say five) at every time step and then take the maximum.

Let us have several threads each running these processes defining over a shared data structure that allows them to access birds in their vicinity and then we can just perform this over a single thread that is performing this process. Then we will have a concurrent algorithm that works very well using this technique. Similarly, we can change things slightly to make a good, distributed algorithm that would have a lot of massage passing.

## Computational Limitations of Bio-Inspired Optimizations:

Bio-Inspired optimization algorithms are frequently used as one of the applications or tools of artificial intelligence as these optimization algorithms such as swarm intelligence have overcome conventional arithmetic and numerical methods which fail when the solutions settle in local minima or maxima.

The main drawback of bio-inspired algorithms is their hunger for intensive computational power as this significantly time consuming when solving some of the complex functions during optimization.

## Parallel Computation methods for Bio-Inspired Optimizations:

Bio-inspired optimization algorithms require high computational power and the traditional computation via single system, or a CPU is very time consuming to solve some of the complex functions. In this section, two parallel computing approaches are detailed that can enhance the performance of these swarm intelligence algorithms.

### Parallel Computation using GPUs:

A processor or a central processing unit also known as CPU oversees all the mathematical and logical calculations of our computer, so its main purpose is to run instructions which we commonly known as code. So, every time we interact with a program, every time we copy, delete files or even when we type a single letter inside a text document all these tasks are performed by the CPU.

Additionally, all the communication between the different computer components also relies on the CPU. So, the components do not communicate to each other directly, but they have this CPU middleman in between.

So, CPUs are good at multitasking because we are always able to run a bunch of applications at once. Though we tend to consider the CPUs are multi-tasking, surprisingly this is not exactly the case as the core of our CPU can only handle one task at a time.

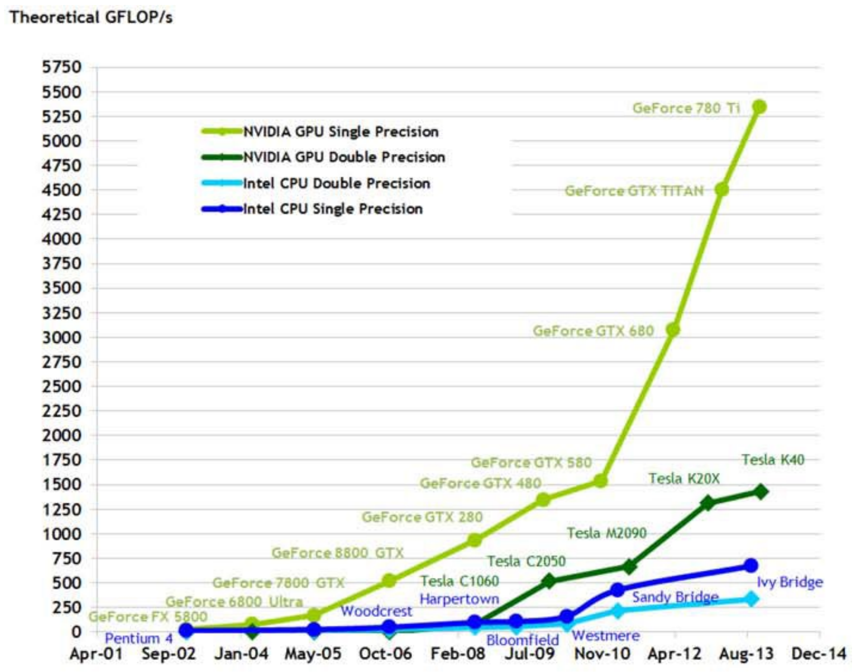


*Figure 18: CPU Vs GPU*

This core is an independent processing unit, and our multitasking abilities directly depend on how many cores our hardware has. The most advanced and newest CPU on the market only has 32 cores. So, CPUs are incredibly fast, but they cannot do multi-tasking.

Graphical processing units also known as GPUs oversees displaying images on your screen. Modern GPUs are extremely powerful because often they come with their very own memory and their very own processor. GPU's help execute code in parallel as well as offload processing on the CPU. GPUs in contrast to CPUs use thousands of ALU's to execute huge computations in parallel such as matrix multiplication.

When running a complex parallel algorithm, instead of sending every little task to CPU, they are sent to GPU so that those tasks can be executed in parallel. GPUs can skip this unnecessary communication and provide you with a much faster outcome. A modern GPU has approximately 10496 cores and thus helps making the computations parallel and faster.

Cuda is a powerful software platform that helps computer programs run faster. We often use it to solve performance intensive problems such as cryptocurrency mining, video rendering and of course machine learning problems. Cuda is not just software, but it is also embedded in hardware.

Cuda runs code in parallel and it allows you to switch between CPU processing and GPU processing for very specific tasks.

Using CUDA and the GPUs, the PSO optimization algorithm can be computed in parallel and faster. One of the ways to do so is by dissecting the algorithm into parallel and serial tasks. As shown in equation 1, the fitness function has matrix multiplications which is a compute intensive parallel task. This fitness evaluation step can be executed on GPU while all other steps are executed on CPU and Cuda will allow to switch information between CPU and GPU.

The figure 19 shows the schematic of the PSO algorithm before and after dissecting. Implementing this new approach by leveraging GPUs reduces the computational time of the optimization. However, all the algorithm steps still take time as they are handled by the CPU on single machine.

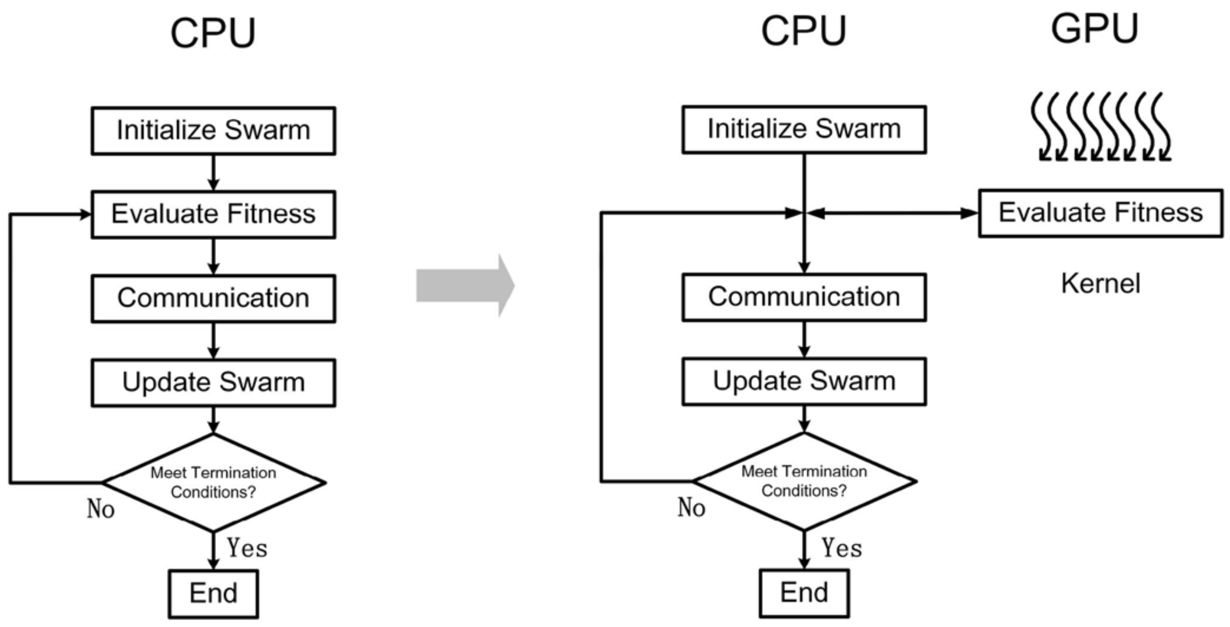


*Figure 19: PSO on GPU*

**Parallel Computation using HPC:**

In large scale and or high dimensional optimization problems, the compute power in a single compute machine is not sufficient and thus requires multiple compute nodes clustered together to solve the problem. This clustered multiple distributed compute nodes are nothing but HPC cluster.

OpenMP is a shared-memory programming standard that can provide compiler directives based on threads within a single compute node. Thus, using OpenMP provides us a way to parallelize the tasks on a multi-core CPU within a single compute node. OpenMP allows to create multiple threads to get processed on multiple cores.

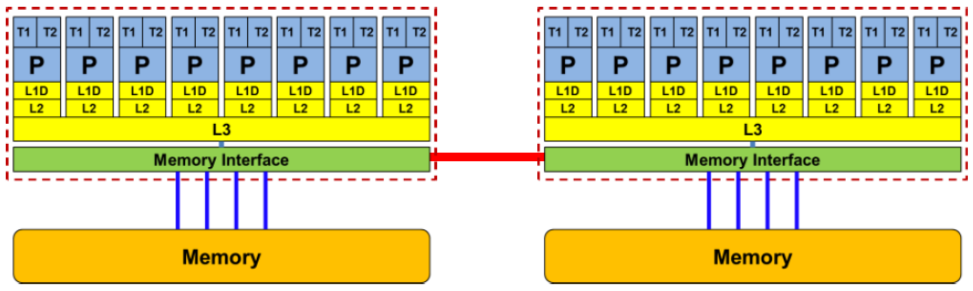


*Figure 20: Shared Memory within a node*

Message Passing Interface is a distributed-memory programming standard which uses library calls to initiate multiple processes across multiple distributed compute nodes. These processes execute tasks on different compute nodes and interacts using the MPI communication network for information sharing.

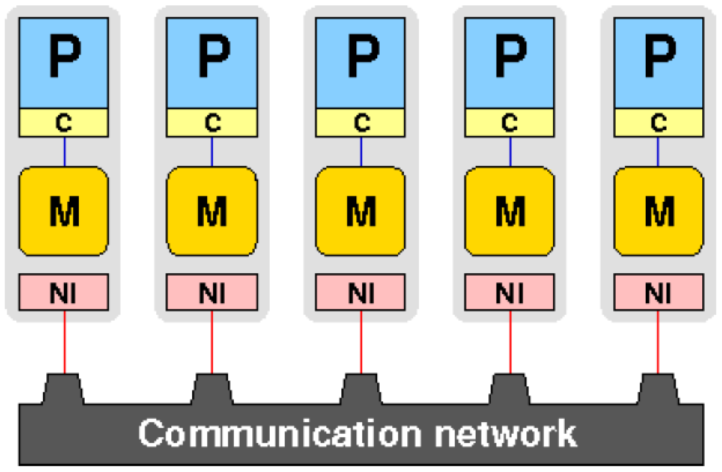


*Figure 21: Distributed Memory across nodes*

If you look at modern cluster topology, then part of it namely the part inside a node or a shared memory node is accessible to shape memory parallelization, and it seems to be the natural way to program this subsystem to use threading model. In high performance computing and scientific computing, OpenMP is the typical choice for that.

So, the basic idea is to combine the threading on the node level with MPI between nodes so that message passing is done with MPI where it is necessary and everywhere we have shared memory, we can use OpenMP.

This hybrid approach to combine OpenMP at shared memory level within node and MPI at distributed memory level across nodes on a HPC can improve efficiency of the large scale high dimensional bio-inspired optimization algorithms such as PSO.

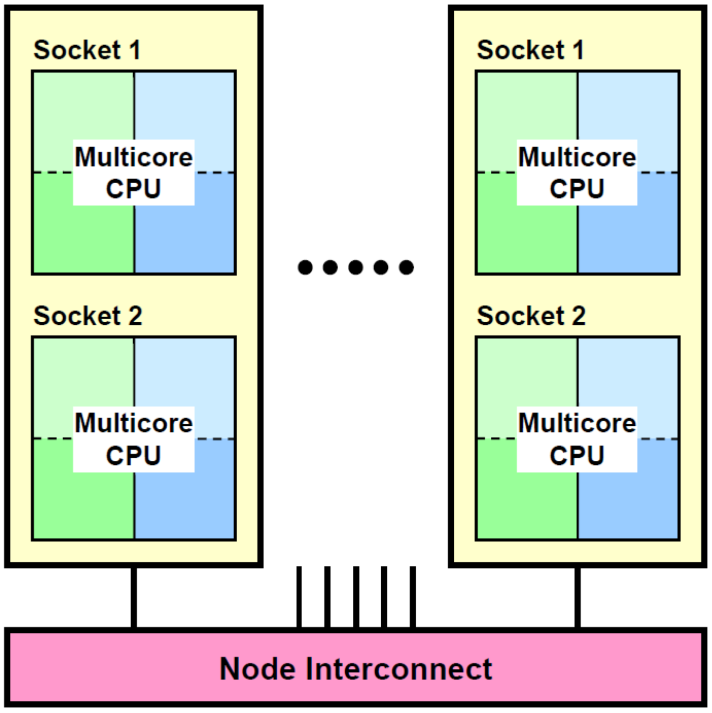
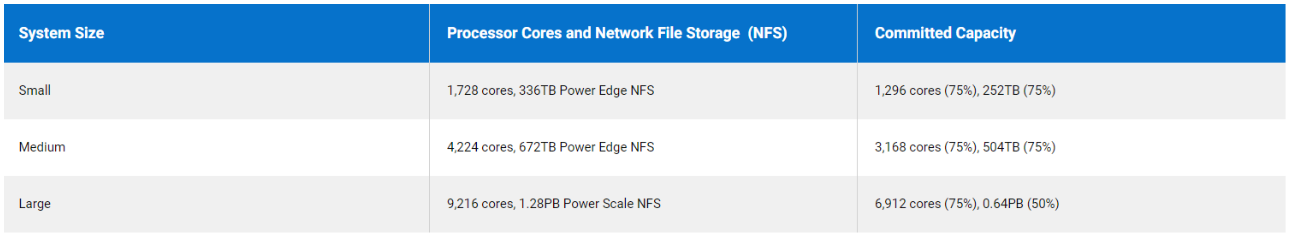


*Figure 22: Hybrid Programming using OpenMP and MPI*

**Dell Apex HPCaaS:**

Dell Technologies Apex High Performance Computing As A Service provides HPC systems as a fully managed solution that comes in various configurations. It is a scalable subscription-based solution that can scale based on seasonal demands in the computational requirements. It comes with low latency interconnect across nodes to provide highest throughput. Optimization algorithms can be implemented using the hybrid approach of programming shared memory using OpenMP and distributed memory using MPI to solve high dimensional large scale optimization problems.



| System Size | Processor Cores and Network File Storage (NFS) | Committed Capacity |
|---|---|---|
| Small | 1,728 cores, 336TB Power Edge NFS | 1,296 cores (75%), 252TB (75%) |
| Medium | 4,224 cores, 672TB Power Edge NFS | 3,168 cores (75%), 504TB (75%) |
| Large | 9,216 cores, 1.28PB Power Scale NFS | 6,912 cores (75%), 0.64PB (50%) |

*Figure 23: Dell Apex HPCaaS Offers*

## References:

1) Hager, G., & Wellein, G. (2010). Introduction to high performance computing for scientists and engineers. CRC Press.

2) Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., ... & Yelick, K. A. (2006). The landscape of parallel computing research: A view from Berkeley.

3) de Melo Menezes, B. A., Kuchen, H., & Buarque de Lima Neto, F. (2022). Parallelization of Swarm Intelligence Algorithms: Literature Review. *International Journal of Parallel Programming*, 1-29.

4) Tan, Y., & Ding, K. (2015). A survey on GPU-based implementation of swarm intelligence algorithms. *IEEE transactions on cybernetics*, *46*(9), 2028-2041.

5) Stojanovic, N., & Stojanovic, D. (2013). High–performance computing in GIS: Techniques and applications. *International Journal of Reasoning-based Intelligent Systems*, *5*(1), 42-49.

6) Lalwani, S., Sharma, H., Satapathy, S. C., Deep, K., & Bansal, J. C. (2019). A survey on parallel particle swarm optimization algorithms. *Arabian Journal for Science and Engineering*, *44*(4), 2899-2923.

7) Krömer, P., Platoš, J., & Snášel, V. (2013, August). A brief survey of advances in particle swarm optimization on graphic processing units. In *2013 World Congress on Nature and Biologically Inspired Computing* (pp. 182-188). IEEE.

8) https://www.dell.com/en-us/dt/apex/compute-hci/high-performance-computing.htm#accordion0

9) https://www.top500.org/

10) https://www.mpi-forum.org/