HPC is used to derive answers to questions that cannot be adequately addressed alone through means of empiricism, theory, or even widely available or accessible commercial computers. Supercomputing is what HPC is used for. Scientific simulations, data analytics, machine learning AI, financial modelling, climate prediction, personalized medicine and drug discovery, fraud detection, natural disaster prediction, nuclear bomb study. Numerical Methods – Finite element method, monte carlo, fast fourier transformation, linear algebraic methods, optimization methods, euler method, runge kutta method. **SISD** simplest model of computation-single processor executes single instruction on single piece of data at a time-traditional model used in most personal computers. **SIMD**-single instruction is applied to multiple pieces of data in parallel-used in vector processors and graphics processing units (GPUs), which can perform operations on large amounts of data in parallel. **~~MISD~~**-multiple processors execute different instructions on the same piece of data simultaneously-not commonly used in practice, but it has been proposed for fault-tolerant computing. **MIMD**-multiple processors execute different instructions on different pieces of data simultaneously. This is the most common model used in parallel computing, where multiple processors work together to solve a problem by dividing it into smaller tasks and executing them concurrently. Epoch 3 instruction parallelism, vacuum tube with transistor, 4 75-80vector processing, increased clock rate, pipeline structures, 5 80-90 SIMD array, Parallel Vector Processing, Cray XMP, 6 90-05 communicating sequential processing (CSP), distributed memory, commodity clusters, MPPs, 7 5-20 multi0core, hybrid execution, fastest computers, improved energy efficiency, 8 3D die stacking.

**Peak floating point operations per second(flops),** peak instructions per second(ips),main memory capacity,, I/O bandwidth, secondary storage capacity, organization: class of system, # nodes, # processors per node, accelerators, network topology; file system, control strategy: MIMD, Vector, PVP, SPMD, SIMD; **sustained throughput(avg performance over a period of time(flops, Mflops, Gflops, ips, Mips, ops, Mops), Cycles per instruction(cpi), Memory access latency(cycles per second), memory access bandwidth(bytes per second), bi-section bandwidth(bytes/second). Hops-# of network devices through which data passes from source to destination, latency(time taken for a single task to be completed, memory access time), scalability (ability of system to handle increase in workload by adding more resources), power consumption, memory usage, CPU utilization, Cycle Times (memory cycle time, logic switching speed), granularity (logic density, memory density, task and packet size).** Performance factors parallelism: fully independent processing elements operating concurrently on separate tasks, instruction level parallelism(ILP)-fine grained, Pipelining, SIMD operations, overlapping of computation and communication(asynchronous), multithreading  
SLOW-starvation(Amdahl’s law), latency, overhead, waiting for contention

Table

Description automatically generated

Managed resources: computer nodes, processing cores, interconnect, permanent storage I/O options, Accelerators**:**  Availability of execution and auxiliary resources, priority, max # of jobs per user, software licenses, requested execution time, event occurrence, job dependencies**:** **SLURM**, PBS, OpenLava(LSF),Hadoop - Event triggered scheduling, backfill scheduler, gang scheduling, pre-emption, generic resources(GRES) driven scheduling, Trackable resources (TRES), elastic computing, high throughput computing. srun, salloc, sbatch, squeue, scancel, sacct, sinfo

OS controls the execution of application program. **Primary functionalit**y: exploit the hardware resources of one or more processors. Provides a set of services to system users. Manages secondary memory and IO devices.- **adv:** makes comp more convenient to use, allows comp system resources to be used in an efficient matter, permit effective dev, test, and introduction of new system functions without interfering with service**. Services provided by OS** – program development (editors and debuggers), program execution, access to IO devices, controlled access to files, system access, protection, error detection and response(internal and external hardware errors, software errors, OS can’t grant request of application), accounting. **Resources managed by OS**-Processor, main memory(volatile), IO modules(secondary memory devices, communication equipment, terminals), system bus(communication among processor, memory, and IO modules). **Process Mgmt.**- process states, process control block, process mgmt. activities(process context switching, pcb, context switching is an expensive operation in terms of performance, as it requires OS to save and restore the state of a process, switch the memory address space and update internal DS; OS supports synchronization mechanisms such as barriers, semaphores, mutexes(to avoid read write ops hazard); message passing b/w process is responsibility of OS. OS gives user interface for message passing such as sockets, pipes, fifios), scheduling (there are job queues that have sub queue that store all processes based on their state and features; there are job schedulers maintained by OS. Long term scheduler, short term scheduler, medium term scheduler) Threads provide medium grained parallelism which may give more parallelism, better scalability, and possibly shorter time to solution, **Memory Mgmt.**-(keep track of which parts of memory are currently being used and by whom, deciding which processes and data to move into and out of memory, allocation and deallocating memory space as needed, virtual to physical address translation)Virtual Memory(allows programmers to address memory from a logical point of view), Virtual Page address(Paging allows a process to store a fixed sized blocks. Each page has a virtual address), Virtual address translation (OS incorporates a page address which is central to the method for associating virtual page addresses to physical locations within main memory or secondary storages), **File Mgmt.**, **IO system Mgmt**, **secondary storage mgmt.**

Accelerators are designed to perform certain types of computations and specific tasks or set of tasks much faster than CPU. CPUS are designed to handle wide range of tasks. It is controlled and coordinated by main processor in the system, but they are capable of performing these computations independently. It uses thousands of lightweight cores whose instruction sets are optimized for dimensional matrix arithmetic and floating-point calculations. CPU has a few heavy weight cores with high clock speed. Accelerators often have their own specialized memory, while CPU has shared memory with other components. Accelerators are designed to move large amounts of data quickly while CPUs are more general purpose and not as optimized for data throughput. Task granularities are very fine for GPU and individual computations is broken down into small data parallel tasks that can be executed simultaneously on multiple GPU cores. In threads and processes, data transfer between different threads or processes is typically done through shared memory or IPCs. In GPUs, data transfer between the CPU and GPU and between different GPU cores is typically done through high-bandwidth memory transfers across the PCIe bus. Threads and processes typically use virtual memory mechanisms to manage memory allocation and protection. GPUs, on the other hand, use a combination of specialized GPU memory and system memory, and often require careful management of data placement and transfer to achieve optimal performance. synchronization is typically achieved through locks, semaphores, and other mechanisms that ensure mutual exclusion or orderly execution of shared resources. In GPUs, synchronization is achieved through specialized memory fences and barriers that ensure that memory operations are executed in a specific order across multiple cores. threads and processes are typically programmed using high-level languages like C, C++, and Java, with support for threads and processes provided through libraries like POSIX threads and OpenMP. GPUs, on the other hand, are typically programmed using specialized languages like CUDA and OpenCL that provide low-level control over the GPU hardware and memory. instead of breaking up a single task into smaller sub-tasks that can be executed independently, the task is broken down into many small, independent computations that can be executed in parallel across the cores of the accelerator. (assign each chunk to different GPU core)

Mass storage not concerned with volatile devices such as mainmemory or processor registers. It plays integral role in checkpoint and restart of compute applications, alleviating the impact of temporal ad system resource limits imposed on app execution. **HDD/ Platter**-info is stored on 1 or both sides of a disc shaped platter with base glass, AL, Ceramics. Used for storage devices in computers and servers. large files like multimedia content, documents db, **SSD**-faster and more reliable. Storge in laptops, tablets, phones. Ideal for high-speed data transferring and processing such as in gaming and video editing, **Magnetic Tape**-storage for backups and archives especially for large amounts of data that need to be stored for a long time. Suitable for data not accessed frequently such as historically records, scientific data, govt documents, **Optical Storage**-CD-ROMS,DVDs, Blu-ray disc, used for storing music, videos, and software applications, one-way messages. NOT Suitable for large data.

**A file system** is a method for storing and organizing computer files and data on storage devices. A **distributed file system (DFS)** is a file system that spans multiple devices and allows multiple users to access files and data. There are several differences between a cluster operating system and a desktop OS when it comes to implementing a DFS. **Fault tolerance**: A DFS implemented on a cluster operating system is designed to be fault tolerant. If one of the devices in the cluster fails, the DFS will automatically redirect requests to another device in the cluster. This helps ensure that the DFS remains available even if some of the devices fail. In contrast, a desktop OS may not be designed with the same level of fault tolerance, and a failure of a single device could result in the loss of data or the unavailability of the DFS. **Account separation**: In a DFS implemented on a cluster OS, user accounts are typically managed by a central authentication server. This allows users to access files and data from any device in the cluster using the same account credentials. In contrast, a desktop OS may not have the

same level of centralized account management, which could make it more difficult to manage access to the DFS. **Concurrent users**: A DFS implemented on a cluster OS is designed to handle multiple concurrent users accessing the same files and data. The DFS will typically employ mechanisms such as file locking to ensure that multiple users do not attempt to modify the same file at the same time. In contrast, a desktop OS may not be designed to handle multiple concurrent users accessing the same files and data, which could lead to issues with data consistency. -a cluster OS is better suited for implementing DFS because it is designed with fault tolerance, centralized account management, and support for multiple concurrent users.

Export OMP\_NUM\_THREADS=8, export OMP\_DYNAMIC=true, export OMP\_SCHEDULE static,N/P, export OMP\_NESTED FALSE

The input (sales.txt) is divided into two splits. Two map tasks running on two different nodes, Node A and Node B, extract product and quantity from the respective split’s records in parallel. The output from each map function is a key-value pair where product is the key while quantity is the value. The combiner then performs local summation of product quantities. As there is only one reduce task, no partitioning is performed. The output from the two map tasks is then copied to a third node, Node C, that runs the shuffle stage as part of the reduce task. The sort stage then groups all quantities of the same product together as a list. Like the combiner, the reduce function then sums up the quantities of each unique product in order to create the output. **Split>combine>partition>shuffle/sort>reduce**Begin with sample data and 2 initially supplied cluster points. Use Euclidean measure, each individual is assigned to the cluster nearest to the (x,y) pair. For a given (x,y) value pair the mapper iterates over each cluster’s mean value and finds the cluster with the nearest distance to the (x,y) pair. It returns as a key the cluster and as a value the (x,y) pair. The reducer receives a list of (x,y) value pairs for each cluster and computes the new cluster mean value. This MapReduce operation can be performed iteratively until no more updates occur or until a maximum number of iterations is reached.

Text

Description automatically generated Text

Description automatically generated

#include <omp.h> #define N 16 main (){ int i, chunk; float a[N], b[N], c[N]; **for (i=0; i < N; i++) a[i] = b[i] = i \* 1.0;** chunk = 4; printf("a[i] + b[i] = c[i] \n");  
**#pragma omp parallel shared(a,b,c,chunk) private(i){ #pragma omp for schedule(dynamic,chunk) nowait**   
**for (i=0; i < N; i++) c[i] = a[i] + b[i];** **#pragma omp sections{{ a=…; b=…;} }** for (i=0; i < N; i++)printf(" %f + %f = %f \n",a[i],b[i],c[i]); }  
**sum = 0.0   
#pragma omp parallel for reduction(+:sum) for (i=0; i < 20; i++) sum = sum + (a[i] \* b[i]);** At the end of a reduction, the shared variable contains the result obtained upon combination of the list of variables processed using the operator specified**. #pragma omp critical**