CSYE7374 HW4 Instructor: Dr. Handan Liu 2020-08-07

Give several reasons to use parallel computing?

Many naturally occurring events occur in parallel and cannot be captured with serial computation. These complex real-world phenomena often have a temporal sequence that is best suited for parallel computation. Without parallel computing, it would be very difficult to model simultaneous variables seen in things like weather patterns and rush hour traffic simulations. The huge number of variables in such models demand huge processing and memory capabilities and trying to compute them in serial would simply require unreasonable amounts of time and money. Sometimes big data may be geographically dispersed and computing resources would require communication over distributed networking and memory, something that cannot be achieved without utilizing parallel hardware.

On the CPU architecture, how many parallel programming models are there based on memory access methods? List them and explain briefly.

Two main parallel programming models are based on memory access methods and historically these have been classified as Uniform Memory Access (UMA) and Non-Uniform Memory Access (NUMA).

- Uniform Memory Access (UMA) consists of primarily symmetric multiprocessor (SMP) machines which are inherently composed of identical processors with equal access and access times to memory. Whenever anyone processor updates a location in the shared memory space, all the other processors are cognizant about the update. This behaviour is known as ‘cache coherency’ and is designed natively at the hardware level.

- Non-Uniform Memory Access (NUMA) is designed to be physically interlinked between numerous symmetric multiprocessor (SMP) machines where one can directly access the memory of another. Due to this interlinking, memory access is often slower and not all processors have equal access time to all memories. Variations, where cache coherency is enforced by design, are usually known as cache-coherent NUMAs (CC-NUMA).

What is Flynn’s Taxonomy? Please explain it.

Flynn's taxonomy is a way to distinguish various multi-processor computer architectures based on two independent dimensions, namely the data stream and the instruction stream. The two dimensions can only be in one of two states: multiple or single. These result in the four following permutations:

- Single Instruction Single Data (SISD) is where a serial computer operates with one central process unit (CPU) on an individual instruction set, to deterministically execute during one clock cycle.

- Single Instruction Multiple Data (SIMD) is a parallel computer where multiple processing units (CPUs) execute a given instruction set synchronously on different data elements, to deterministically execute during a given clock cycle.

- Multiple Instruction Single Data (MISD) is a parallel computer where multiple processing units execute distinct instruction sets independently on a set of shared data elements.

- Multiple Instruction Multiple Data (MIMD) is a parallel computer where multiple processors execute distinct instruction sets on different data elements, to execute asynchronously, the result of which may be non-deterministic.

What kind of classification in Flynn classification does GPU computing belong to? Please explain.

According to Flynn’s classification, GPU computing best fits into Multiple Instruction Multiple Data (MIMD). This is because GPUs can do multiple things on multiple sets of data independently. GPU’s are known to have shader units that are task agnostic when doing geometry work, pixel colouring or physics.

What is “embarrassingly parallel”?

“Embarrassingly parallel” solves problems which are composed of independent tasks where there is no coordination or communication required. These tasks can be executed in parallel as each task can complete their portion of the work with no data sharing.

How can we evaluate the speedup of parallel computing comparing with serial computing? Please explain in detail.

Amdahl's Law states that potential program speedup is defined by the fraction of code (P) that can be parallelized. If none of the code can be parallelized, P = 0 and the speedup = 1 (no speedup). If all of the code is parallelized, P = 1 and the speedup is infinite in theory. If 50% of the code can be parallelized, the maximum theoretical speedup = 2, meaning the code will run twice as fast. The simplest method to compare speedup is to compare the elapsed time of the common code segment using different numbers of CPUs and observing how the elapsed runtime changes. From this, we can plot the change curve. We can even combine parallel mechanisms like OpenMP (shared memory) and MPI (distributed memory) to evaluate whether that are any advantages of adopting a hybrid model, in terms of reducing runtime.

Give 3 popular math libraries for high performance computing.

1. The BLAS (Basic Linear Algebra Subprograms) routine is probably the most popular high-performance computing maths library. Intel’s MKL is one variation of BLAS.

2. The second most popular maths library is probably the AMD Core Math Library (ACML) due to the rise of cheap AMD GPU chips.

3. The third most popular maths library is probably The Engineering Scientific Subroutine Library (ESSL) by IBM which provides a variety of optimized complex mathematical functions.

Provide 6 MPI necessary functions for MPI parallel computing. Only write the function name, no parameters. C is recommended here.

1. MPI_Barrier()
2. MPI_Bcast()
3. MPI_Scatter()
4. MPI_Gather()
5. MPI_Allgather()
6. MPI_Reduce()

What are strong scaling and weak scaling? Or what difference between them.

Weak and strong scaling are two scaling goals that summarize the duration to reach a solution as you scale up.

- Weak scaling describes situations where the “problem size per processor” remain fixed as more processors are added so that the total problem size is proportional to the number of processors used. The goal with weak scaling is to run larger and larger problems without increasing the amount of time (in the same duration). In this regard, perfect scaling would mean N number of processers would still have the same runtime as a single processor (unrealistic).

- Strong scaling describes situations where the “total problem size” remains fixed, with the goal of being able to run the same problem size, faster and faster, as more processors are added. In this pursuit, perfect scaling would mean, with N processors being added, the time to solve the problem would proportionally be cut short by 1/N time when compared to serial (not practical).

Give several possible reasons why sometimes the computation time on multi-CPU on multiple nodes is slower than that of multi-CPU on a single node.

Depending on a program’s hotspots and bottlenecks there may be hidden inhibitors to parallelism, and forcing a multi-node setup may cause a slowdown. When adding a second node, the data from the first node requires to be migrated and rebalanced across both nodes and all writes will now have to do twice as much work (master and replica). Inter-node communication happens over a network, which may be congested, causing additional overhead.

Please list the general idea of a CUDA workflow. (hint: 6 steps)

1. Allocate memory on both the device and the host
2. Initialize variables on the host
3. Copy data from the host to the device
4. Perform parallel computation on the device
5. Copy data back from the device to the host
6. Free memory on both the device and the host

Please indicate which types of memory are on-chip and off-chip in GPU memory?

There are four types of off-chip GPU memory. In order of speed of access times, they are the following:

1) **Constant memory** is read-only and used for data that doesn't change over the course of kernel' execution. Using constant memory can reduce the required memory bandwidth but performance gains can only be realized when a warp of threads can read the same location

2) **Texture** memory is also read-only memory on the device but can only be accessed by physically adjacent warps. It also reduces memory traffic for slight performance gains.

3) **Local** memory is a slower variant of register memory.

4) **Global** memory allows data to be visible to all threads within the application, including the host and lasts for the duration of the host's allocation.

There are two types of on-chip GPU memory. In order of speed of access times, they are the following:

1. **Register** File memory is ephemeral, so data is only visible to the thread that writes it and lasts only for the lifetime of the thread.
2. **D-cache** or **Shared** memory lasts for the duration of a block and is visible to all threads within it, allowing for threads to intercommunicate and pass data amongst one another.

Information Systems

About Northeastern University

The objective of the course is to understand the principles of high-performance computing and the practice of the emerging parallelism-based machine learning paradigm. We will learn high-performance parallel architectures and parallel programming models. And we will explore the parallelism of machine learning and deep learning to achieve high speedup and high performance on heterogenous cluster architectures, as well as the applications to a variety of domains, including image classification, speech recognition, and natural language processing, etc. This course is mainly composed of four parts: Parallel architecture and programming; Parallel machine learning implementation; GPU architecture and CUDA programming; deep learning parallel implementation (TensorFlow, PyTorch etc.). Every student in this course will get permission to access northeastern high-performance computing cluster and practice many hands-on labs on Discovery CPU and GPU nodes. The homework explores parallel algorithms and simple applications, and the final project allows an in-depth exploration of a particular application area. From the project, students will also learn best practices to structure a model and manage research experiments.