Parallel Programming Notes

Ayush Raina

February 1, 2025

Lecture 1: Architectures 1

Classification of Architectures - Flynns Classification

- SISD: Single Instruction Single Data, for example, serial computers.
- SIMD: Single Instruction Multiple Data, for example, vector processors and processor arrays.
- MISD: Multiple Instruction Single Data, for example, trying different ways to decrypt a message.
- MIMD: Multiple Instruction Multiple Data, for example, multi-core processors.

Classification based on Memory

- **Shared Memory:** All processors share a common memory. Communication is done using this shared address space. This is further classified into UMA (Uniform Memory Access) and NUMA (Non-Uniform Memory Access).
  - UMA: All processors have equal access time to all memory locations. Memory access is slow and has limited bandwidth. UMA has single memory controller.
  - NUMA: Processors have variable access to memory locations. Memory access is faster and has higher bandwidth than UMA. NUMA has multiple memory controllers.
- **Distributed Memory:** Each processor has its own memory.

Shared memory could itself be distributed among processor nodes. Each processor might have some portion of the memory that is physically close to it and therefore accessible in less time.

Cache Coherence Problem

If each processor in a shared memory multiprocessor machine has a data cache, then the problem of cache coherence arises. Our objective is that processes should not read **stale** data.

Cache Coherence Protocols

- **Write Invalidate:** When a processor writes to its cache, it sends invalidate signal to all other caches that have a copy of that memory location. This means all other cache locations must discard their copy of the memory location and fetch it again from the main memory if needed.
- **Write Update:** When a processor writes to its cache, it sends the updated data to other caches that have a copy of that memory location. This keeps all caches up to date.

False Sharing

If two processors access different variables that are located within same cache line but are not related to each other. In this case any write to one variable will cause cache line to be invalidated and other processor might have to reload the data from main memory. This is called false sharing.

In next lecture we will discuss about interconnecting networks.

Lecture 2: Architectures 2

Interconnection Networks

They are used in both shared memory to connect processors to memory and in distributed memory to connect different processors to each other. Components for interconenction networks are interfaces such as Peripheral Component Interconnect (PCI) or PCI - Express used for connecting processors to network and a network link which connected to communication network.

Communication Network

It consists of switching elements to which processors are connected through ports. **Switching elements** receive data from one point and send it to another point. **Switch** refers to collection of these switching elements. **Network topology** is specific pattern of connections in which these switching elements are connected.
In shared memory systems processors as well as memory units are connected to communication network.

Different Kinds of Network Topologies

1. **Bus:** All processors are connected to a single bus. It is simple and cheap but has limited bandwidth.
2. **Crossbar Switch:** It consists of 2D grid of switching elements, where each switching element consists of two input and two output ports. Input Ports are connected to output ports through a switching logic.

In previous case of Crossbar Switch, we require nm switching elements where n is number of processors and m is number of memory units in case of shared memory architecture or m is the number of processors in distributed memory architectures. To reduce switching complexity, we can use **Multistage Network - Omega Network**.

3. **Omega Network:** It consists of $logP$ stages each consisting of $P/2$ switching elements.

Consider Distributed Memory Architecture. In Crossbar switch, there is dedicated path for any processor to communicate with any other processor without contention but in Omega Network, there is contention if 2 processors wants to communicate to 2 different processors, they might have to take same path through some stage of the network.

If P_i and P_j wants to communicate in Omega Network, first convert $ID(P_i)$ and $ID(P_j)$ to binary and then keep comparing most significant bits and follow **cut through routing** or **pass through routing** to reach destination.

Some commonly used network topologies used in distributed memory architectures are Mesh, Torus, hypercubes and Fat tree.

1. **Mesh:** Grid of switching elements where each switching element is connected to 4 directional neighbours.
2. **Torus:** Mesh with wrap around connections. In this $T(i,1)$ is connected to $T(i,n) \forall i$. Similarly $T(1,j)$ is connected to $T(m,j) \forall j$.
3. **Hypercube:** Keep Joining binary n cubes to get hypercube. In hypercubes, distance between any two processors is bit difference between their binary representation.
4. **Fat Tree:** It is a tree like structure where each node is a switch and each switch has multiple ports. Leaves are processors. As we go up the tree, number of ports in switch increases. This is Non Blocking Network means no contention because there is unique path between any two processors. Fat Tree has a special property which makes all this happen and that is **Number of Links from node to a children = Number of Links from node to parent**.

bandwidth: maximum amount of data that can be transferred over the network in given time, usually measured in bits per second. (bps)

Evaluating Interconnection Topologies

1. **Diameter:** Maximum distance between any two processing nodes.Smaller diameter means lower latencies.
2. **Connectivity:** Number of Paths between two nodes. It can also be defined as minimum number of links that need to be removed to disconnect the network. Higher connectivity improves fault tolerance.
3. **Fault Tolerance:** Ability of network to operate correctly even if some links or switches fail.
4. **Bisection Width:** Minimum number of links that need to be removed to divide the network into two equal halves.
5. **Channel Width:** Number of bits that can be simultaneously communicated over a link i.e number of physical wires between two nodes. Higher Channel width increases data transfer capacity.
6. **Channel Rate:** Performance of single physical wire i.e The speed at which single physical wire transmits the data and is generally measured in bits per second(bps).
7. **Channel bandwidth:** The total data transfer capacity of a link. It is calculated as Channel Width * Channel Rate. Higher Channel bandwidth allows faster communication.
8. **Bisection bandwidth:** This is defined as maximum volume of communication between two halves of network or in other words maximum data transfer capacity between two halves of network and is given by bisection width * channel bandwidth. Higher bisection bandwidth means better performance under heavy traffic.

Questions / Doubts

1. How fat tree network has constant bisection bandwidth?

Lecture 3: Parallelization Principles

We have seen several advantages of parallelism, but there are also some overheads which are not seen in sequential programs like **communication delay**, **synchronization** and **idling**.

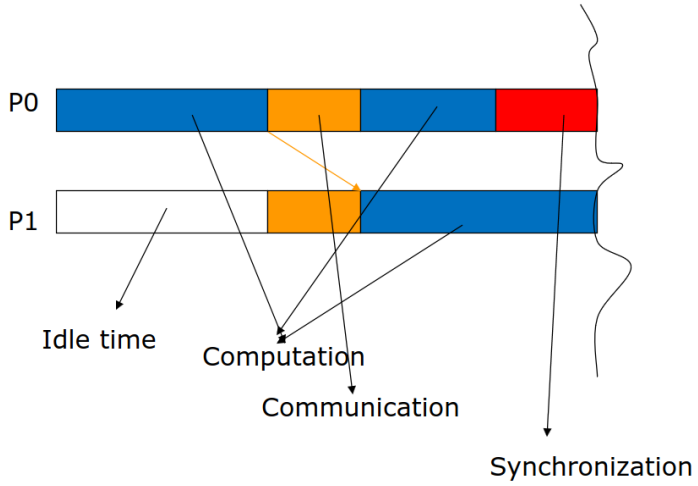

Figure 1: Overheads

Evaluation of Parallel Program

Let us denote execution time by T_p for parallel program using p processors.

1. **Speedup:** $S(p,n) = T(1,n)/T(p,n)$ where $T(1,n)$ is execution time of sequential program. Usually $S(p,n) \leq p$ as we expect program to get p times faster in idead case but sometimes $S(p,n) > p$ which is called **superlinear speedup**. Ideally we want $S(p,n) = p$.

2. **Efficiency:** $E(p,n) = S(p,n)/p$. Efficiency is a measure of how well the parallel program is utilizing the resources. Generally $E(p,n) \leq 1$, but in case of superlinear speedup, $E(p,n) > 1$. Ideally we want $E(p,n) = 1$.

3. **Cost:** $C(p,n) = T(p,n) * p$. It is similar to Efficiency but relates the runtime to number of utilized processors.

4. **Scalability:** We want to measure the efficieny of parallel program for variable number of processors. This is called scalability analysis. There are two types of scalability - **Strong Scalability** and **Weak Scalability**. In strong scalability, we measure the efficiencies for varying number of processors while keeping indut data size fixed. In weak scalability, we measure efficiency of parallel code by varying both the number of processors and input data size.

We will now derive an upper bound on achievable speedup when parallelizing a given sequential program. Let T_{seq} and T_{par} denote the parts whihc cannot benefit from parallelization and can benefit from parallelization respectively. Further assume we can get ideal linear speed up and super linear speedup is also possible. Then we have $T(p,n) \geq T_{seq} + T_{par}/p$.

The Speedup is given by

$$\boxed{S(p,n) = \frac{T(1,n)}{T(p,n)} \leq \frac{T_{seq} + T_{par}}{T_{seq} + T_{par}/p}}$$

Instead of using the actual runtimes, now use fraction of the total runtime. Let f be such that $T_{seq} = f * T(1,n)$ and $T_{par} = (1-f) * T(1,n)$. Then we have

$$\boxed{S(p,n) \leq \frac{1}{f + \frac{1-f}{p}}}$$

The above equation is known as Amdahl's Law. Now we can see some examples:

1. Suppose 95% of a rogram's execution time occurs inside a loop that we want to parallelize. What is the maximum speedup we can expect from a parallel version of our program executed on six processors?

$$\boxed{S(6,n) \leq \frac{1}{0.05 + \frac{0.95}{6}} = 4.8}$$

2. 10% of the program's execution time is spent within sequential code. What is limit to speedup achievable by parallel version ?

Since we are not given how many processors are used, we assume ideal case of unbounded processors.

$$\boxed{S(\infty,n) \leq \lim_{p\to\infty} \frac{1}{0.1 + \frac{0.9}{p}} = 10}$$

We can see the limitation to Amdahl's Law is that it only applies in situation where problem size is fixed. We will now derive a more general upper bound on speedup. Let α be the scaling function for the sequential part of program and β be the scaling function for the parallel part of program. Both α and β are w.r.t to complexity of problem size. This means T_{seq} grows as $\alpha * T(1,n)$ as problem size grows and T_{par} grows as $\beta * T(1,n)$ as problem size grows.

$$T_{\alpha\beta}(1,n) = \alpha * T_{seq} + \beta * T_{par} = \alpha * f * T(1,n) + \beta * (1-f) * T(1,n)$$
$$T_{\alpha\beta}(p,n) = \alpha * T_{seq} + \frac{\beta * T_{par}}{p} = \alpha * f * T(1,n) + \frac{\beta * (1-f) * T(1,n)}{p}$$

Then we have

$$\boxed{S_{\alpha\beta}(p,n) \leq \frac{\alpha * f + \beta * (1-f)}{\alpha * f + \frac{\beta*(1-f)}{p}}}$$

Let $\gamma = \frac{\beta}{\alpha}$, then we have

$$\boxed{S_{\gamma}(p,n) \leq \frac{f + \gamma * (1-f)}{f + \frac{\gamma*(1-f)}{p}}}$$

Now depending on the value of γ, we can have different upper bounds on speedup.

1. Amdahl's Law: $\gamma = 1$ which means $\alpha = \beta$
2. Gustafson's Law: $\gamma = p$ which means $\alpha = 1$ and $\beta = p$. This means parallel part of program grows linearly in p as problem size grows. In this case the formula becomes

$$\boxed{S(p,n) \leq f + (1-f) * p}$$

This law can also be thought as by knowing f , we can use it to predict the theoretically achievable speedup using multiple processors when parallelizable part scales linearly with the problem size while the serial part remains constant

let us discuss some examples:

1. Suppose we have a parallel program that is 15% serial and 85% linearly parallelizable for a given problem size. Assume that the (absolute) serial time does not grow as the problem size is scaled.

(i) How much speedup can we achieve if we use 50 processors without scaling the problem - $S_{\gamma=1}(50,n)$ (ii) Suppose we scale up the problem size by a factor of 100. How much speedup could we achieve with 50 processors - $S_{\gamma=100}(50,n)$

Roofline Performance Model

This gives a bound on the performance of an application on a particular architecture and it also helps to categorize the code's performance as memory bound or performance bound. It depends on three parameters - Peak Performance of the machine P_{peak} $(FLOP/s)$, Memory Bandwidth B_{peak} $(Bytes/s)$ and Computational Intensity I_{op} $(FLOP/Byte)$.

Questions / Doubts

1. Discuss this Roofline Model again with Professor and work out some examples.

Lecture 4.1: Parallel Programming Models

Some parallel programming models include Single Program Multiple Data (SPMD) and Multiple Program Multiple Data (MPMD). We will focus on SPMD model.

Programming Paradigms

1. Shared Memory Model: Threads, Open MP, CUDA.
2. Distributed Memory Model: MPI - Message Passing Interface.

Data Parallelism and Domain Decomposition

Given data is divided into processing entities. Each process owns and computes a portion of the data.
Multidimensional Data in simulations is divided into subdomains and each subdomain is assigned to a processing entity. This is called **Domain Decomposition**.

Process Grids

Given P processors are arranged in multi dimensions forming a process grid. Once this is done then domain of problem is divided in process grid.

Distribution of Data

There are various ways of distributing data like block distribution, cyclic distribution, block cyclic distribution etc. Check Parallelization Principles pdf in good notes for visualization.