Chapter 1
Introduction to parallel programming

Getahun F. (MSc)

getahun.fikadu@bhu.edu.et

The slide is based on ‘introduction to parallel computing book(peter pacheco)’

Outline

- **Parallel computing overview, and the need of parallelism**
- Concurrent, parallel and distributed computing
- Parallel hard ware and parallel software
- Parallel programming

Why parallel computing?

❖The vast increases in computational power provides todays enjoyable advanced technology.

Accurate medical imaging's

Internet

Advanced web search and etc…

All have been impossible without increasing computers computational power

❖ ***Even needs extra increment in feature***

As our computational power increases, the number of problems that we can seriously consider solving also increases

Eg. *Climate modeling, Protein folding*, *Drug discovery*, *Data analysis, energy research*

Con…

Climate modelling:

- For modeling the interactions between the atmosphere, the oceans, solid land, and the ice caps at the poles

Energy research:

- For modelling accurate wind turbines, solar cells, and batteries.

Drug discovery:

- *For alternative treatments by careful analysis of the genomes of the individuals for whom the known treatment is ineffective*

Protein folding:

- *Our ability to study configurations of complex molecules such as proteins is severely limited by our current computational power*

Data analysis:

- *Data generated doubles every two years*
- *E.g.* Web search engines

WHY WE'RE BUILDING PARALLEL SYSTEMS?

- ❖ By the ever-increasing density of transistors, the scientist get tremendous increase in single processor performance
  - ➢ But it needs decreasing the size of transistor
    - ▪ Their power consumption also increases and Most of this power is dissipated as heat
      - ✓ Causes integrated circuit gets too hot, the system becomes unreliable
- ▪ Therefore, it is becoming impossible to continue to increase the speed of integrated circuits
  - ✓ but there is moral imperative to continue to increase computational power
    - ✓ Surprisingly, if the integrated circuit industry doesn't continue provide solution, ….
- ❖ Can the scientist exploit the continuing increase in transistor density in other method?
  - ❖ Yes, by *parallelism*

parallelism

- Rather than building ever-faster, more complex, monolithic processors, the industry has decided to put multiple, relatively simple, complete processors on a single chip.
  - Such integrated circuits are called multicore processors
    - Core has become synonymous with CPU
    - In this setting a conventional processor with one CPU is often called a single-core system
    - Scientist tried to provide parallel computation by writing parallel program which is run on parallel processor
- WHY WE NEED TO WRITE PARALLEL PROGRAMS?
  - To exploit the presence of multiple cores system
  - To provide realistic parallel computations
    - But it needs exact parallel program or finding method which convert serial program to parallel program

The bad news is that researchers have had very limited success writing programs that convert serial programs to parallel.

HOW DO WE WRITE PARALLEL PROGRAMS?

- Mostly depend on the basic idea of *partitioning the work* to be done among the cores
  - Two widely used approaches: task-parallelism and data-parallelism

In task-parallelism:

we partition the various tasks carried out in solving the problem among the cores.

E.g. *executing different instructions*

In data-parallelism:

we *partition the data used in solving the problem* among the cores, and each core carries out more or less similar operations on its part of the data

- ✓ *each core carries out roughly the same operations on its assigned data elements*

Con…

- Though it is difficult to standardize parallel programming language, currently, the most powerful parallel programs are written using extensions to languages such as C and C++
  - These programs include explicit instructions for parallelism
  - We'll be focusing on learning to write programs on c language that are explicitly parallel.
  - C language has explicit extension to execute the program parallel, those are like
    - *Message-Passing Interface or MPI,*
    - *POSIX Threads or Pthreads, and*

are libraries of type definitions, functions, and macros that can be used in C programs

    - *OpenMP*

consists of a library and some modifications to the C compiler

Con…

- When we write parallel programs,. we usually need to *coordinate* the work of the cores
- This can involve:
  - Communication among the cores,
  - load balancing, and
  - synchronization of the cores.

Types of parallelism

- In parallel programming, it is normal if each core do their own work independently.
  - Difficulty is when they share some resource to each other
    - Structuring may be simple but converting to program is very complex
- For these reason scientist divides parallelism mechanism in to two, shared memory based parallelism and distributed memory based parallelism.

- In a shared-memory system, the cores can share access to the computer's memory; in principle, each core can read and write each memory location
  - *Pthreads and OpenMP were designed for programming shared-memory systems*
    - ***They provide mechanisms for accessing shared-memory locations***

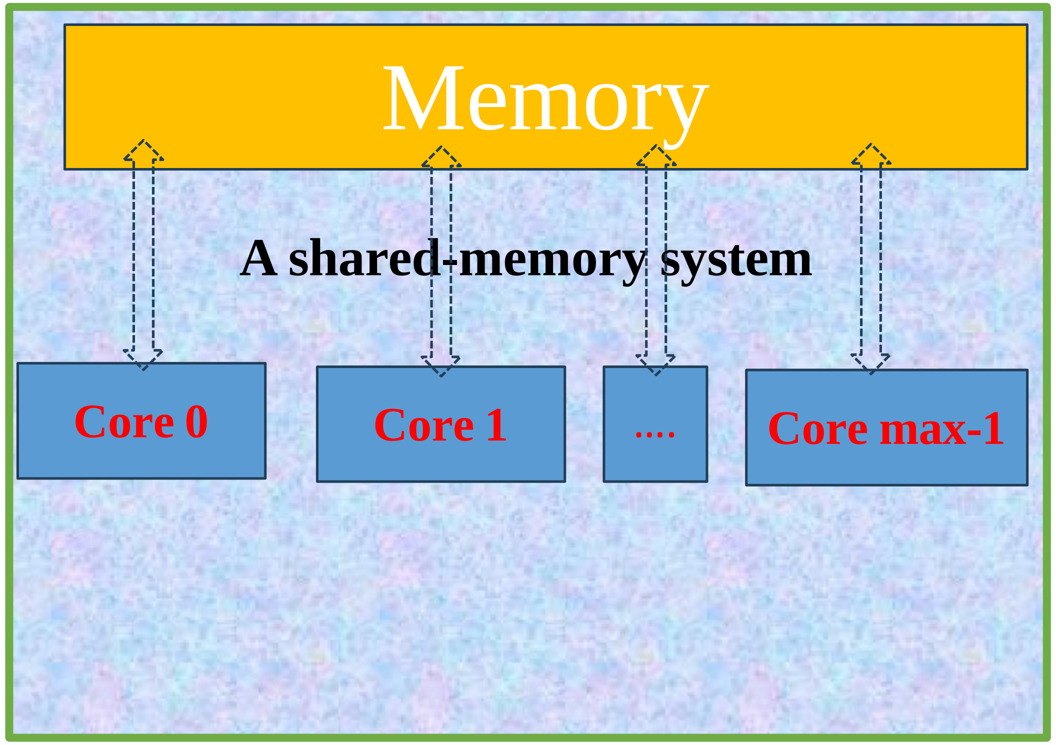
In a distributed memory system, each core has its own, private memory, and the cores must communicate explicitly by doing something like sending messages across a network.

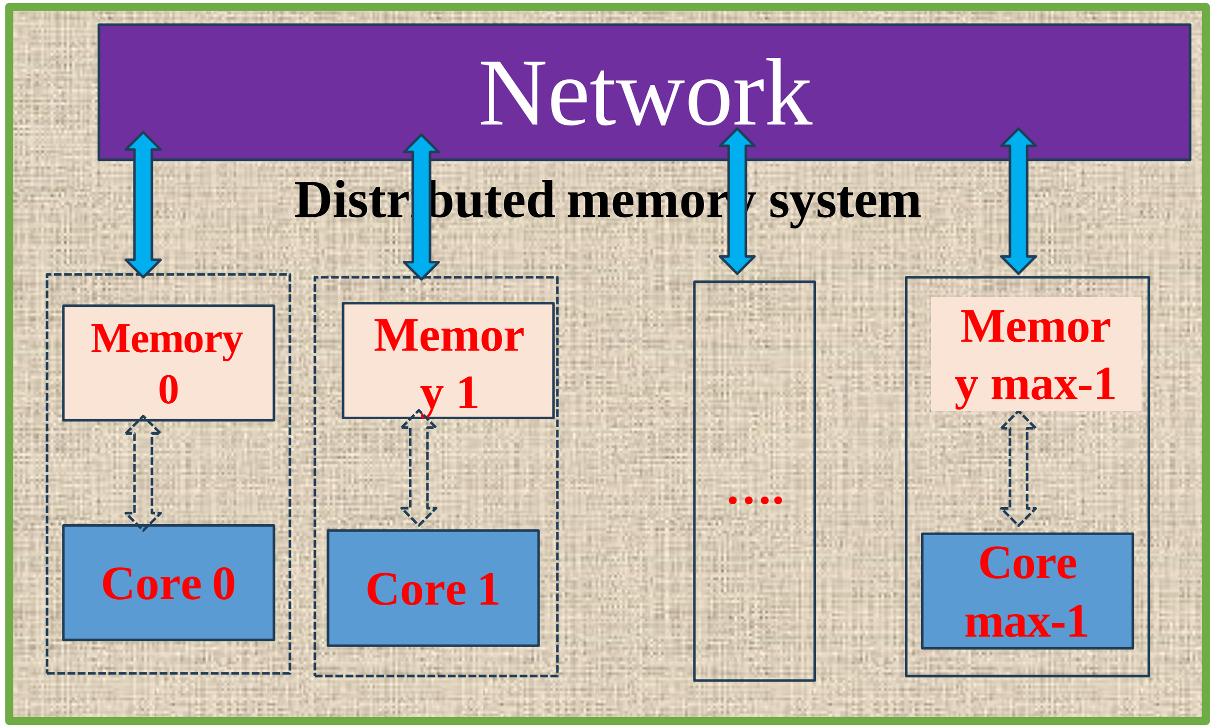
- ✓ MPI was designed for programming distributed-memory systems. It provides mechanisms for sending messages

Con…

❖In shared memory, OpenMP allows us to parallelize many programs with relative ease, while Pthreads provides us with some constructs that make other programs easier to parallelize.

❖OpenMP+ Pthreads





CONCURRENT, PARALLEL, DISTRIBUTED

CONCURRENT COMPUTING :

- ✓ Multiple tasks can be ***in progress*** at any instant.

PARALLEL COMPUTING :

- ✓ multiple tasks ***cooperate closely*** to solve a problem
- ✓ They are concurrent and Runs multiple tasks ***simultaneously*** on cores that are physically close to each other

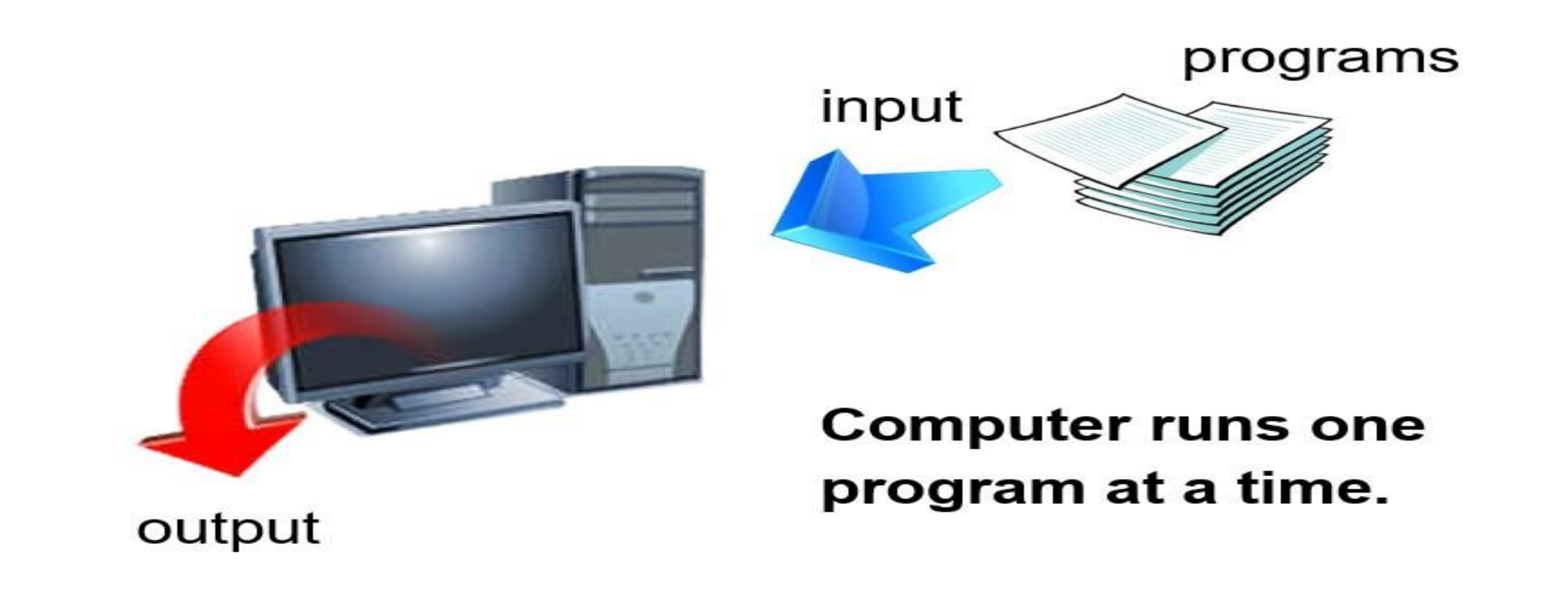
DISTRIBUTED COMPUTING : program may ***need to cooperate with other programs*** to solve a problem

- ✓ They are concurrent and The tasks may be executed by multiple computers that are may be separated by large distances.

- ✓ But beware, there isn't general agreement on these terms
- ✓ For example, many authors consider shared-memory programs to be "parallel" and distributed-memory programs to be "distributed."

Parallel software and parallel hard ware

- ❖ Serial hardware runs one program at a time



- ❖ Parallel hardware and software have grown out of conventional serial hardware and Software.
  - ✓ PS designed to run multiple instruction simultaneously at a time.

parallel hardware

- Hard ware is parallel if its functional units are replicated and perform several tasks simultaneously
- Parallel hardware is often classified using Flynn’s taxonomy, which distinguishes between the *number of instruction streams* and the *number of data streams* a system can handle.
- A von Neumann system has a single instruction stream and a single data stream so it is classified as a single instruction, single data, or SISD, system.
- A single instruction, multiple data, or SIMD, system executes a single instruction at a time, but the instruction can operate on multiple data items
    - ✓ *Called data parallel programs*

Con…

- *Parallel data programs* are programs in which the data are divided among the processors and each data item is subjected to more or less the same sequence of instructions.

- Vector processors and graphics processing units are often classified as SIMD systems, although current generation GPUs also have characteristics of multiple instruction, multiple data stream systems (MIMD).

Flynn's taxonomy

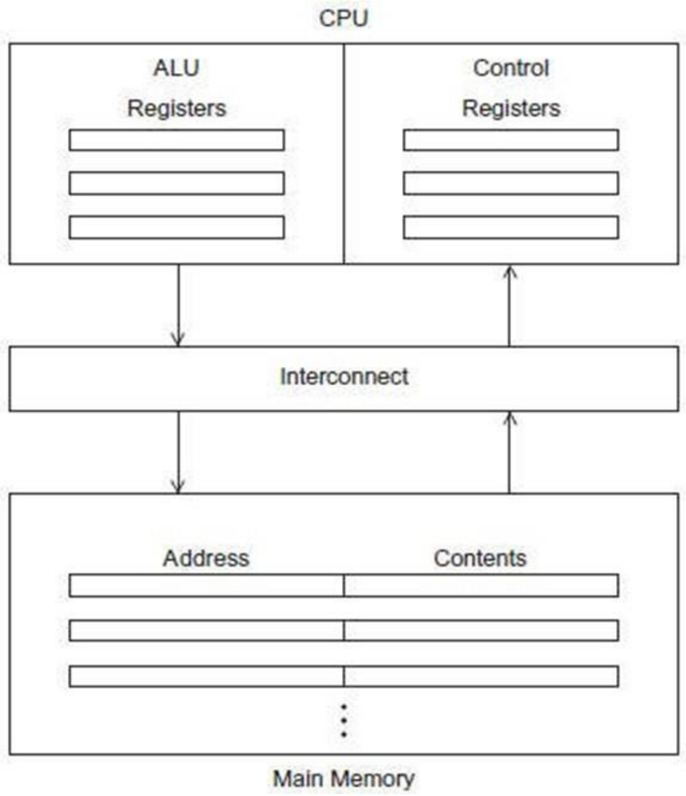
- Flynn's taxonomy refers to a classification computer system in architecture that categorizes computer processors based on the number of concurrent instruction streams they can handle and the level of data parallelism they support

1, SISD(Single Instruction stream, Single Data stream): Traditional sequential processing, where a single instruction stream operates on a single data stream. This represents the classical von Neumann architecture.

2,SIMD (Single Instruction stream, Multiple Data streams): This architecture involves a single control unit that executes the same instruction on multiple pieces of data simultaneously.

- Parallelism achieved by dividing data among the processors
- SIMD processors are capable of performing parallel operations on elements of vectors or arrays.
- Called data parallelism
- Example: GPU's, converts the internal representation into an array of pixels that can be sent to a computer screen

SISD: The von Neumann Architecture



Cont…

3, MISD (Multiple Instruction, Single Data):

- MISD architecture involves multiple instruction streams acting on a single stream of data.

Example: Leiserson

Ex. Leiserson are an example of MISD architecture. In which parallel input data flows through a network of hard-wired processor nodes, combined, processed, merged or sort the input data into a derived result.

4, MIMD (Multiple Instruction, Multiple Data):

- MIMD architecture comprises multiple instruction streams operating on multiple data streams concurrently.
- Typically consist of a collection of fully independent processing units or cores, each of which has its own control unit and its own ALU
- **Example:** Multi-core processors, clusters, or supercomputers that run several independent tasks or instructions on various pieces of data at the same time,
- providing true parallel processing.

PARALLEL SOFTWARE

- Refers to programs or applications designed to execute tasks simultaneously, taking advantage of multicore processors HW or distributed computing systems.
- The primary objective of parallel software is to divide a large task into smaller sub-tasks that can be processed concurrently, thereby improving performance, speeding up execution, and handling more extensive workloads.
- There are various forms of parallelism in software:
  1) *Instruction-Level Parallelism (ILP)*
  2) *Task Parallelism*
  3) *Data Parallelism*

- **Instruction-Level Parallelism (ILP):** executing multiple instructions at the same time within a single processor core.
  - ✓ Techniques like pipelining increase ILP.
    - ✓ *Pipelining breaking down the execution of instructions into several stages.*
    - ✓ *Each stage completes a part of the instruction's execution, and as one stage finishes, it passes the partially processed data to the next stage.*

For instance, dependencies between instructions might cause problems in the pipeline,

- Task Parallelism: This approach involves breaking down a program into smaller tasks that can be executed concurrently.
  - ✓ *These tasks might be independent or have dependencies.*
- *Data Parallelism*: In this form, large datasets are divided, and different parts of the dataset are processed simultaneously by multiple processors.

Processes and threads

- In parallel computing, processes and threads are fundamental concepts that enable concurrent execution, but they differ in their characteristics and how they function.

What is a process?

A process is an independent entity in an operating system that executes a program. It comprises its memory space, resources, and system state. Each process has its address space, its own set of registers, and other attributes, and operates independently of other processes.

Processes are isolated from one another and generally do not share memory so that require a significant amount of system resource.

What is a thread?

A thread is a component within a process that can be scheduled for execution.

Threads share the same memory space and resources within a process.

- ✓ *lighter in terms of resource consumption*
- ✓ *Switching between threads is faster than switching between processes*

speed up

main purpose in writing parallel programs is usually increasing performance.

- It can be evaluated in terms of Speedup and efficiency
- Increasing number of cores, and equally divide the work among the cores.
- If we call the serial run-time Tserial, our parallel run-time Tparallel, and *p* as the computers core, then the best we can hope for is:

$$T_{\text{parallel}} = T_{\text{serial}}/p.$$

- When this happens, we say that our parallel program has linear speedup.
- It is impossible to get linear speedup because the use of multiple *processes/threads* almost invariably introduces some overhead.
- Thread share processes memory which has always critical sections,
  - Calling *mutex function* to avoid dead lock-which causes overhead

cont..

If we define the speedup S of a parallel program to be

$S = T_{serial}/T_{parallel}$, since $T_{parallel} = T_{serial}/p$ then

$S = T_{serial}/T_{serial}/p$, so $S = p$, which is unusual

Efficiency(E)

- ❖ The ratio between speedup S of a processor or core p.
  - ➤ E = S/P, which becomes

$$E = \frac{S}{p} = \frac{\left(\frac{T_{\text{serial}}}{T_{\text{parallel}}}\right)}{p} = \frac{T_{\text{serial}}}{p \cdot T_{\text{parallel}}}.$$

Table 2.4 Speedups and Efficiencies of a Parallel Program

p	1	2	4	8	16
S	1.0	1.9	3.6	6.5	10.8
E = S/p	1.0	0.95	0.90	0.81	0.68

Amdahl's law

"Unless virtually all of a serial program is parallelized, the possible speedup is going to be very limited—regardless of the number of cores available."

- Amdahl's Law *gives the maximum theoretical speedup that can be achieved when a portion of a program is parallelized*, while *some part remains sequential*.

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

Where:

S(N) = Speedup using N processors

P = Fraction of the program that can be parallelized ($0 \leq P \leq 1$)

N = Number of processors

Amdahl's law

- Even if you increase the number of processors, the sequential portion (1 - P) limits the overall speedup.
- In other words, parallelism is limited by the serial bottleneck.

Example:If 80% of a program can be parallelized (P=0.8) and we use 4 processors

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \Rightarrow S(4) = \frac{1}{(1-0.8) + \frac{0.8}{4}} = \frac{1}{0.2 + 0.2} = 2.5$$

So, even with 4 processors, the speedup is only 2.5×, not 4×.

Efficiency

Efficiency : Is the measure of how well parallel resources (processors) are being utilized

$$E = \frac{S(N)}{N}$$

- It tells us *what fraction of each processor's time is actually being used for productive work* (not wasted on waiting or coordination)
- In above example, Even though you have 4 processors, so, $\frac{2.5}{4} = 0.625$ only about 62.5% of their potential is being effectively used because of the *20% sequential portion that cannot be parallelized.*
- *That means each processor is effectively contributing 62.5% of its capacity to the total computation*

Efficiency

If E=1(or 100%), it means perfect efficiency, all processors are working at full capacity with no overhead or idle time.

If E<1, some time is being lost due to communication, synchronization, or sequential portions of the program.

The bottle neck here is considering fixed problem size➔ what if number of problem size increases?

Gustafson's provide solution for this problem➔ Gustafson's law

Gustafson’s Law

Gustafson’s Law argues that *as we increase the number of processors, we can scale the problem size, so the parallel portion dominates* and we can achieve better speedup.

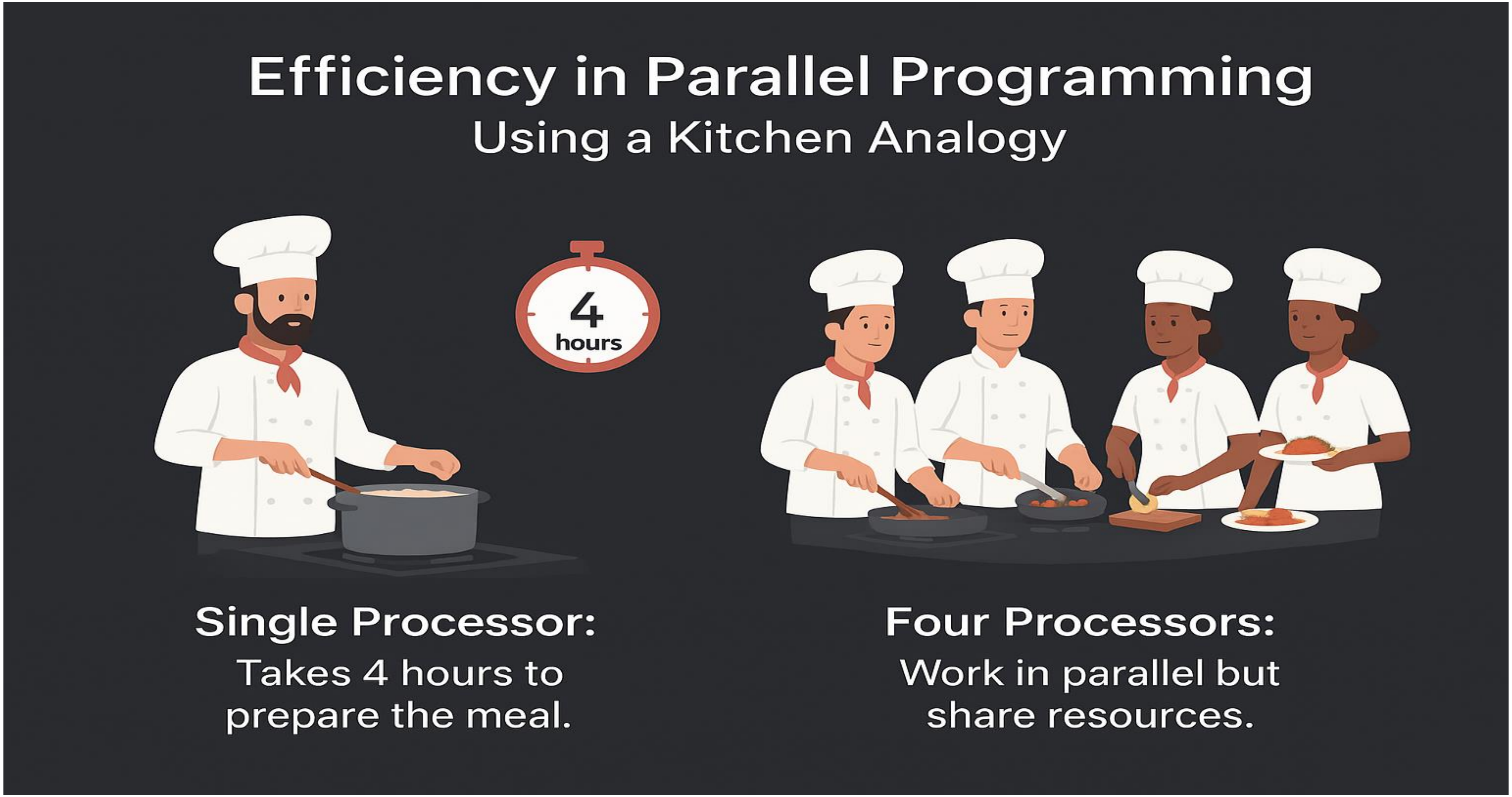
Formula: $$S(N) = N - (1 - P)(N - 1)$$

Where: S(N) = Scaled speedup using N processors
P = Fraction of parallelizable code
N = Number of processors

Unlike Amdahl's Law (which fixes the total work), Gustafson's Law assumes that larger problems can be solved in the same time by using more processors a more realistic model for modern parallel computing.

In above example If P=0.8 and N=4 so $S(N) = N - (1 - P)(N - 1)$

S(4)=4−(1−0.8)(4−1)=4−0.2×3=4−0.6=3.4x which is better than Hamdals


Efficiency in Parallel Programming
Using a Kitchen Analogy
4
hours
Single Processor:
Takes 4 hours to
prepare the meal.
Four Processors:
Work in parallel but
share resources.