COMS3008A: Parallel Computing Lecture 1: Introduction

Hairong Bau

School of Computer Science & Applied Mathematics

Semester two 2021

WITS UNIVERSITY

Contents

WITS UNIVERSITY

Outline

Objectives

- Understand the basics of parallel computers, parallel computing, the motivation of parallel computing, and the classification of parallel computers.
- Understand and apply the simple quantitative modelling for parallel program performance.

Outline

WITS UNIVERSITY

Course admin

- Relevant course information is given in course outline (uploaded on ulwazi course site).
- Course related communications will be announced primarily through the announcement via ulwazi course site. It is important for you to check such announcements regularly to keep informed timely.

Outline

WITS UNIVERSITY

Outline

Parallel computing

- Parallel Computer: A parallel computer is a computer system that uses multiple processing elements simultaneously in a cooperative manner to solve a computational problem.
- Parallel computing (or processing): Parallel processing includes techniques and technologies that make it possible to compute in parallel.
    - Hardware, networks, operating systems, parallel libraries, languages, compilers, algorithms, tools etc.
- Parallel computing is an evolution of serial computing.
    - Parallelism is natural.
    - Computing problems differ in level or type of parallelism.

Serial computing

- A problem is broken into a discrete series of instructions;
- Instructions are executed sequentially one after another;
- Executed on a single processor;
- Only one instruction may execute at any moment in time.

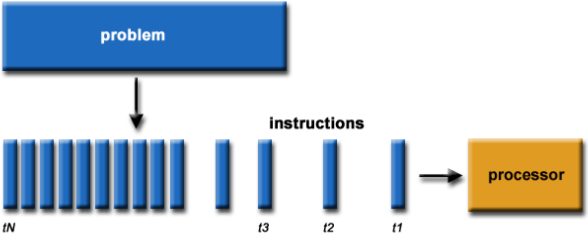


Figure: Serial computing

WITS UNIVERSITY

Parallel computing

- A problem is broken into discrete parts that can be solved concurrently;
- Each part is further broken down to a series of instructions;
- Instructions from each part execute simultaneously on different processors;
- An overall control/coordination mechanism is employed.

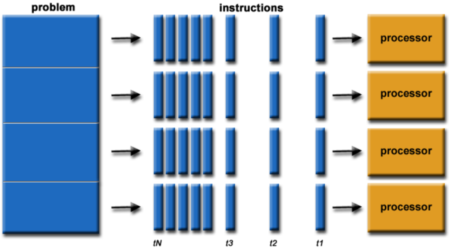


Figure: Parallel computing

WITS UNIVERSITY

Parallel computing cont.

An example of parallelizing an addition of two vectors is shown in Figure 3.

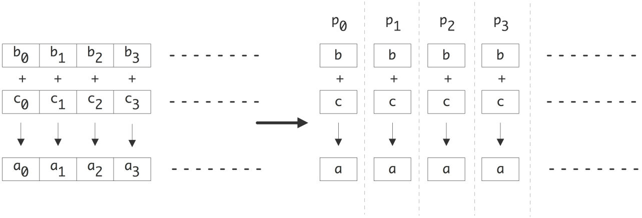


Figure: Parallelization of a vector addition

Outline

WITS UNIVERSITY

Why parallelism?

- In 2004, Intel changed its course from traditional chip design approach (single core processor) to embrace "dual core" processor structure.

The New York Times

TECHNOLOGY; Intel's Big Shift After Hitting Technical Wall

By John Markoff

May 17, 2004

Correction Appended

The warning came first from a group of hobbyists that tests the speeds of computer chips. This year, the group discovered that the Intel Corporation's newest microprocessor was running slower and hotter than its predecessor.

What they had stumbled upon was a major threat to Intel's longstanding approach to dominating the semiconductor industry -- relentlessly raising the clock speed of its chips.

Then two weeks ago, Intel, the world's largest chip maker, publicly acknowledged that it had hit a ''thermal wall'' on its microprocessor line. As a result, the company is changing its product strategy and disbanding one of its most advanced design groups. Intel also said that it would abandon two advanced chip development projects, code-named Tejas and Jayhawk.

Now, Intel is embarked on a course already adopted by some of its major rivals: obtaining more computing power by stamping multiple processors on a single chip rather than straining to increase the speed of a single processor.

Figure: Intel's shift in chip design to multi-core structure.

WITS UNIVERSITY

A brief history of processor performance

- What had been happening before multi-processor? – Single-processor machines: they had been getting exponentially faster.

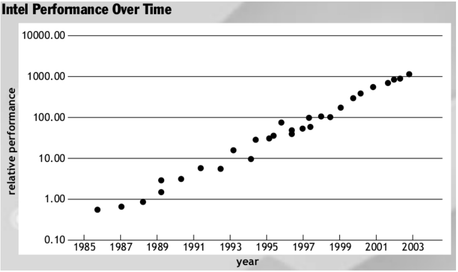


Figure: Intel single processor performance over time

A brief history of processor performance cont.

- Wider data paths
  - 4 bit $\rightarrow$ 8 bit $\rightarrow$ 16 bit $\rightarrow$ 32 bit $\rightarrow$ 64 bit
- More efficient pipelining
  - For example, from 3.5 cycles per instruction (CPI) $\rightarrow$ 1.1 CPI
- Exploiting instruction level parallelism (ILP)
  - " Superscale" processing: e.g., issues up to 4 instructions/cycle
- Faster clock rates
  - e.g., 10MHz $\rightarrow$ 100MHz $\rightarrow$ 1GHz $\rightarrow$ 3GHz

During 80s and 90s, computers had large exponential performance gains

WITS UNIVERSITY

A brief history of processor performance cont.

- “The number of transistors on an integrated circuit doubles every two years.” — Gordon E. Moore

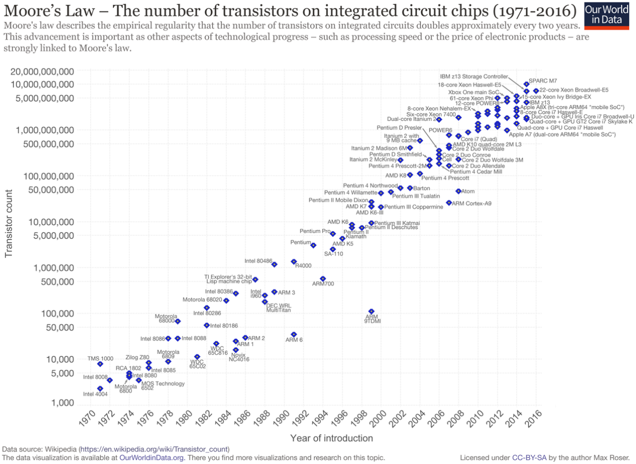
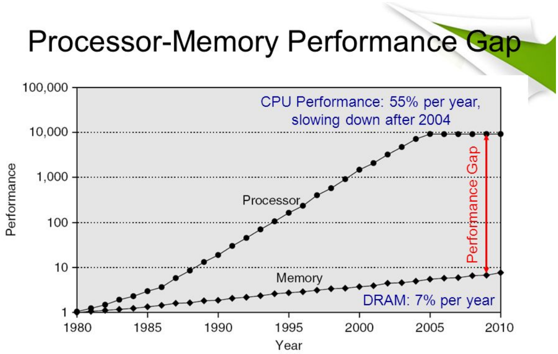


Figure: Moore’s Law – transistor count 1971 - 2016

WITS UNIVERSITY

A brief history of processor performance cont.



- 1980 – No cache in microprocessor
- 1995 – Two-level cache on microprocessor

Figure: Processor-memory performance gap

WITS UNIVERSITY

A brief history of processor performance cont.

- For example, Intel Itanium II
  - 6-way integer unit $<$ 2% die area;
  - Cache logic $>$ 50% die area.
- Most of chip there to keep these 6 integer units at ‘peak’ rate.
- Main issue is external DRAM latency (50ns) to internal clock rate (0.25ns) ratio is 200:1.

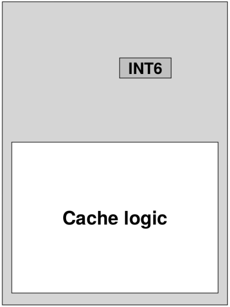


Figure: Illustration of the die area for integer unit and cache logic for Intel Itanium II.

A brief history of parallel computing

- Greater clock frequency, greater electrical power
- Intel VP Patrick Gelsinger (ISSCC 2001): “If scaling continues at present pace, by 2005, high speed processors would have power density of nuclear reactor, by 2010, a rocket nozzle, and by 2015, surface of sun.”

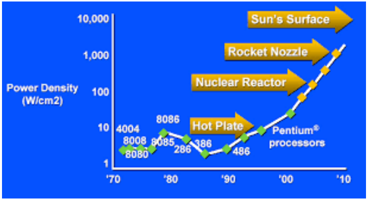
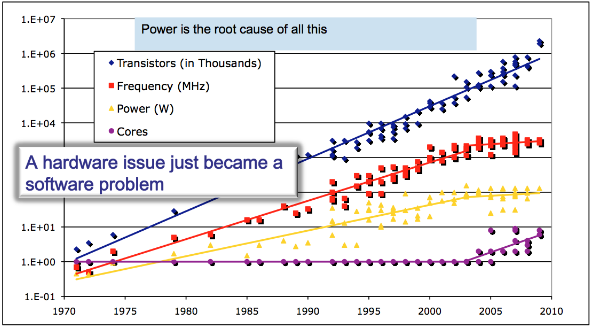


Figure: Intel VP Patrick Gelsinger (ISSCC 2001)

WITS UNIVERSITY

A brief history of parallel computing cont.

- Add multiple cores to add performance, keep clock frequency the same or reduced.



Data from Kunle Olukotun, Lance Hammond, Herb Sutter,
Burton Smith, Chris Batten, and Krste Asanoviç
Slide from Kathy Yelick

Figure: Growth of transistors, frequency, cores, and power consumption

A brief history of parallel computing cont.

- What we can do? We can still pack more and more transistors on to a die, but we cannot make the individual transistor faster and faster.
- We have to make better use of those more and more transistors to increase the performance instead of making only the clock speed faster. One solution is to use parallelism.

Why parallelism – summary

- Serial machines have inherent limitations:
  - Processor speed, memory bottlenecks, ...
- Parallelism has become the future of computing.
- Two primary benefits of parallel computing:
  - Solve fixed size problem in shorter time
  - Solve larger problems in a fixed time
- Other factors motivate parallel processing:
  - Effective use of machine resources;
  - Cost efficiencies;
  - Overcoming memory constraints.
- Performance is still the driving concern.
- Technology push
- Application pull
  - Application performance demands hardware advances;
  - Hardware advances generate new applications;
  - New applications have greater performance demands.

Outline

1. Objectives
2. Course admin
3. Parallel computers
   - Parallel Computing — What is it?
   - Why parallelism?
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A Quantitative Look at Parallel Computation

No 1 Supercomputer in 11/2020 — Fugaku

Site	RIKEN Center for Computational Science
Manufacturer	Fujitsu (Fugaku; Fugaku virtual tour)
Cores	7,630,848
Linpack Performance (Rmax)	442,010 TFlop/s
Theoretical Peak (Rpeak)	537,212 TFlop/s
HPCG	16,004.5 TFlop/s
Nmax	21,288,960
Power	29,899.23 kW (Submitted)
Memory	5,087,232 GB
Processor	A64FX 48C 2.2GHz
Interconnect	Tofu interconnect D
Operating System	RHEL
Compiler	Fujitsu software technical computing suite v4.0
Math library	Fujitsu software technical computing suite v4.0
MPI	Fujitsu software technical computing suite v4.0

No 1 Supercomputer in 11/2020 — Fugaku cont.

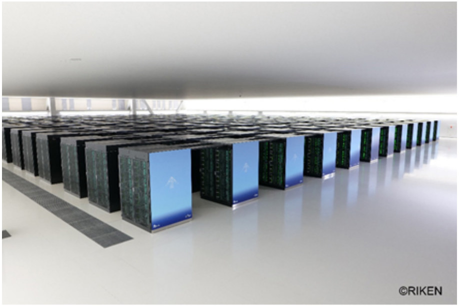


Figure: Fugaku, a supercomputer from RIKEN and Fujitsu Limited.

WITS UNIVERSITY

No 1 Supercomputer in 11/2019 — Summit

Site	DOE/SC/Oak Ridge National Laboratory
Manufacturer	IBM
Cores	2,414,592
Linpack Performance (Rmax)	148,600 TFlop/s
Theoretical Peak (Rpeak)	200,795 TFlop/s
HPCG	2,925.75 TFlop/s
Nmax	16,693,248
Power	9,783.00 kW (Submitted)
Memory	2,801,664 GB
Processor	IBM POWER9 AC9 22C 3.07GHz
GPU	Nvidia Volta GV100
Interconnect	Dual-rail Mellanox EDR Infiniband
Operating System	RHEL 7.4
Compiler	IBM XLC, nvcc
Math library	ESSL, CUBLAS 9.2
MPI	Spectrum MPI

No 1 Supercomputer in 11/2019 — Summit cont.

Figure: Summit, a supercomputer from IBM and the US Department of Energy's Oak Ridge National Laboratory (ORNL).

Lengau from CHPC SA — Rank 400 (11/2018)

Site	Centre for High Performance Computing
System URL	`http://www.chpc.ac.za/`
Manufacturer	Dell
Cores	32,856
Linpack Performance (Rmax)	1,029.3 TFlop/s
Theoretical Peak (Rpeak)	1,366.8 TFlop/s
Nmax	3,105,408
Power	685 kW (Submitted)
Memory	175,232 GB
Processor	Xeon E5-2690v3 12C 2.6GHz
Interconnect	Infiniband FDR
Operating System	CentOS
Compiler	Intel
Math Library	Intel MKL
MPI	Intel MPI

The same machine was ranked 161 and 165 in 11/2016 and 11/2017, respectively.

WITS UNIVERSITY

Lengau from CHPC SA

CHPC'S LENGAU HANGS-ON IN THE TOP500 LIST

The CHPC's Lengau supercomputer has placed 496th on the computing community's **Top500 List**. The list was announced at the International Supercomputing Conference in Frankfurt, Germany in June 2019.

For the first time, all 500 systems deliver a petaflop or more on the High Performance Linpack (HPL) benchmark, with the entry level to the list now at 1.022 petaflops. Lengau has appeared on the Top500 List since her launch in June 2016 and is currently at 1.029 petaflops. In her first appearance in June 2016, she was at number 121.

The Top 500 List lists computers ranked by their performance on the LINPACK benchmark (The LINPACK Benchmarks measure a system's floating point computing power. Introduced by Jack Dongarra, they measure how fast a computer solves a dense n by n system of linear equations, which is a common task in science and engineering). The list is announced in June and in November each year. With over 32000 cores, Lengau remains one of the fastest computers on the African continent, with a utilisation that averagares at 90%.

Lengau continues to put the country in the company of leading supercomputing nations. She has over 1500 registered users, 500 of which are actively engaged in over 200 research programmes.

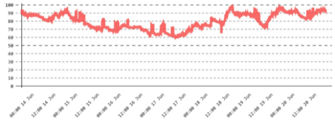


Figure 1: Lengau's utilisation from 14-20 June 2019 shows that usage of the cluster is averaging at 90%, very close to reaching full capacity. Full capacity will result in queues to access processing time on the machine. The dip in the figure can be attributed to the long weekend and public holiday of June 16 2019.

Figure: Lengau – CHPC

WITS UNIVERSITY

The Clusters at Mathematical Sciences Lab

- stampede: Up to sixteen nodes, each with two Xeon E5-2680 CPUs, two GTX1060 GPUs (6GB per GPU, 12GB per node), and 32GB of system AM. For general purpose use or jobs that can leverage InfiniBand, however, InfiniBand has not yet been enabled yet.
- batch: Up to sixteen nodes each with a single Intel Core i9-10940X CPU (14 cores), NVIDIA RTX3090 GPU (24GB), and 128GB of system RAM. For bigger jobs that can leverage a bigger GPU and additional system RAM.
- biggpu: Each node has two Intel Xeon Platinum 8280L CPUs (28 cores per CPU, 56 cores per node) with two NVIDIA Quadro RTX8000 GPUs (48GB per GPU, 96GB per node), and 1TB of system RAM. For mature code that can meaningfully leverage large amounts of GPU and system RAM. While the system will allow jobs to use all three nodes, please try to use only a single node.

WITS
UNIVERSITY

The Clusters at Mathematical Sciences Lab cont.

- ranger: Up to twelve nodes, each with two AMD CPUs, 32GB of RAM, and InfiniBand. No GPUs! For large CPU based jobs. Partition only available later in 2021.
- mia: For the exclusive use of researchers in MIA. Up to twelve nodes with InfiniBand.

Outline

Outline

WITS UNIVERSITY

Flynn's taxonomy

Flynn's taxonomy is widely used since 1966 for classification of parallel computers. The classification is based on two independent dimensions of *instruction stream* and *data stream* with two possible states: *single* or *multiple*.

- SISD: Single instruction stream single data stream. This is the traditional CPU architecture: at any one time only a single instruction is executed, operating on a single data item.

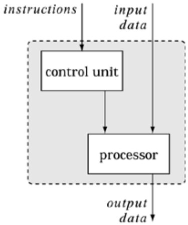


Figure: The SISD architecture

SIMD

- SIMD: Single instruction stream multiple data stream. In this computer type there can be multiple processors, each operating on its own data item, but they are all executing the same instruction on that data item. SIMD computers excel at operations on arrays, such as

$$for(i = 0; i < N; i++) \quad a[i] = b[i] + c[i];$$

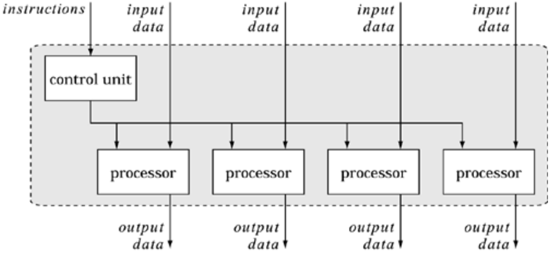


Figure: The SIMD architecture

MIMD

- MISD: Multiple instruction stream single data stream. Each processing unit executes different instruction streams on a single data stream. Very few computers are in this type.
- MIMD: Multiple instruction stream multiple data stream. Multiple processors operate on multiple data items, each executing independent, possibly different instructions. Most current parallel computers are of this type.

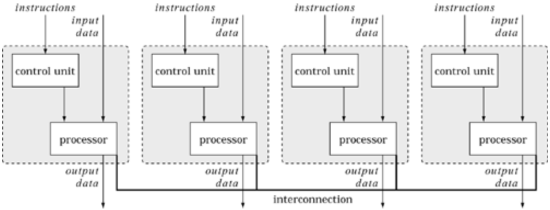


Figure: The MIMD architecture

WITS UNIVERSITY

MIMD cont.

- Most of MIMD machines operate in *single program multiple data* (SPMD) mode, where the programmers starts up the same executable on the parallel processors.

A Further Decomposition of MIMD

The MIMD category is typically further decomposed according to memory organization: shared memory and distributed memory.

- **Shared memory:** In a shared memory system, all processes share a single address space and communicate with each other by writing and reading shared variables.
- One class of shared-memory systems is called SMPs(symmetric multiprocessors).

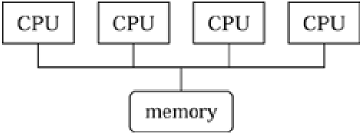


Figure: The SMP architecture

NUMA

- The other main class of shared-memory systems is called non-uniform memory access (NUMA). The memory is shared, it is uniformly addressable from all processors, but some blocks of memory may be physically more closely associated with some processors than others.
- To mitigate the effects of non-uniform access, each processor has a cache, along with a protocol to keep cache entries coherent — cache-coherent non-uniform memory access systems (ccNUMA).

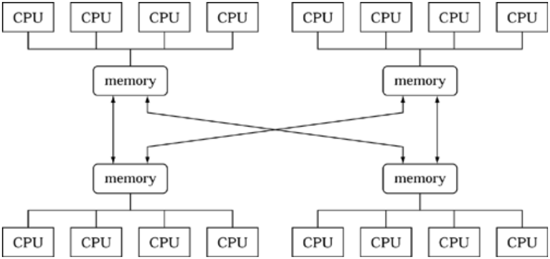


Figure: The NUMA architecture

Distributed memory systems

- Each process has its own address space and communicates with other processes by message passing (sending and receiving messages).

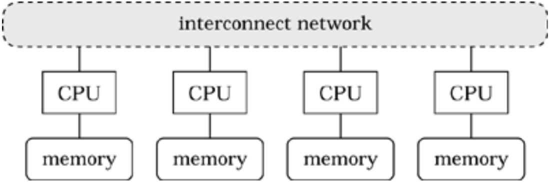


Figure: The distributed memory architecture

Clusters

- Clusters are distributed memory systems composed of off-the-shelf computers connected by an off-the-shelf network.

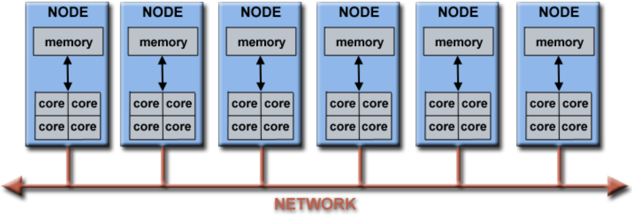


Figure: A cluster

Outline

WITS UNIVERSITY

A simple performance modelling

- Consider a computation consisting of three parts: a setup section, a computation section, and a finalization section.
- The total running time of this program on one processing element (PE) is given as:

$$T_{total}(1) = T_{setup} + T_{compute} + T_{finalization} \tag{1}$$

- What happens when we run this computation on a parallel computer with multiple PEs?

$$T_{total}(P) = T_{setup} + \frac{T_{compute}(1)}{P} + T_{finalization} \tag{2}$$

Speedup

- An important measure of how much additional PEs help is the relative speedup S, which describes how much faster a problem runs:

$$S(P) = \frac{T_{total}(1)}{T_{total}(P)} \tag{3}$$

- A related measure is the efficiency E, which is the speedup normalized by the number of PEs.

$$E(P) = \frac{S(P)}{P} = \frac{T_{total}(1)}{PT_{total}(P)} \tag{4}$$

- Ideally, we would want the speedup to be equal to P, the number of PEs. This is sometimes called perfect linear speedup.

WITS UNIVERSITY

Amdahl's Law

- The terms that cannot be run concurrently are called the serial terms. Their running times represent some fraction of the total, called the serial fraction, denoted γ.

$$\gamma = \frac{T_{setup} + T_{finalization}}{T_{total}(1)} \tag{5}$$

- The fraction of time for parallelizable part is then $1 - \gamma$. The total computation time with P PEs becomes

$$T_{total}(P) = \gamma T_{total}(1) + \frac{(1-\gamma)T_{total}(1)}{P} \tag{6}$$

WITS UNIVERSITY

Amdahl's Law cont.

- Then we obtain the well-known Amdahl's law:

$$S(P) = \frac{T_{total}(1)}{T_{total}(P)} = \frac{T_{total}(1)}{(\gamma + \frac{1-\gamma}{P})T_{total}(1)} = \frac{1}{\gamma + \frac{1-\gamma}{P}} \tag{7}$$

  Taking the limit as P goes to infinity in Eq. 7,

$$S(P) = \frac{1}{\gamma} \tag{8}$$

  Eq. 8 gives an upper bound on the speedup.

- Amdahl's Law says the speedup you can achieve is limited by the fraction for serial computation in your problem (i.e., the number of PEs does not determine the upper bound of speedup you can achieve).

WITS UNIVERSITY

Amdahl's Law cont.

Example 1

Suppose we are able to parallelize 90% of a serial program. Further suppose the speedup of this part is P, the number of processes we used (which is a perfect linear speedup). If the serial time is $T_{serial} = 20$s, then the runtime of the parallelized part is $(0.9 \times T_{serial})/P = 18/P$. The runtime of unparallelized part is $0.1 \times 20 = 2$s. The overall parallel runtime will be

$$T_{parallel} = 18/P + 2,$$

and the speedup will be

$$S = \frac{T_{serial}}{T_{parallel}} = \frac{20}{18/p + 2}.$$

As P gets larger, $18/P$ gets close to 0, and the denominator gets close to 2, in turn, S gets close to 10. That means $S \leq 10$, no matter how many number of processes you use in your program.

Gustafson's Law

- In contrast to Eq. 2, we obtain $T_{total}(1)$ from the serial and parallel parts when executed on P PEs.

$$T_{total}(1) = T_{setup} + PT_{compute}(P) + T_{finalization} \tag{9}$$

- Now, we define the so-called scaled serial fraction, denoted γ_{scaled}, as

$$\gamma_{scaled} = \frac{T_{setup} + T_{finalization}}{T_{total}(P)}, \tag{10}$$

and then

$$T_{total}(1) = \gamma_{scaled} T_{total}(P) + P(1 - \gamma_{scaled}) T_{total}(P). \tag{11}$$

Gustafson's Law cont.

- Using Eq. 11, we obtain the scaled speedup, sometimes known as Gustafson's law.

$$
\begin{aligned}
S(P) = \frac{T_{total}(1)}{T_{total}(P)} &= \frac{\gamma_{scaled} T_{total}(P) + P(1 - \gamma_{scaled}) T_{total}(P)}{T_{total}(P)} \\
&= P(1 - \gamma_{scaled}) + \gamma_{scaled} = P + (1 - P)\gamma_{scaled}.
\end{aligned} \tag{12}
$$

- Suppose we take the limit in P while holding $T_{compute}$ and γ_{scaled} constant. That is, we are increasing the size of the problem so that the total running time remains constant when more processors are added. In this case, the speedup is linear in P.

Gustafson's Law cont.

Example 2

Now, for the previous example, let's assume the scaled serial fraction of the same problem is 0.1, that is $\gamma_{scaled} = 0.1$, and $p = 16$. Then the scaled speedup is $S = 16(1 - 0.1) + 0.1 = 14.5$, which is greater than the upper bound determined by Amdahl's Law (10).

WITS UNIVERSITY

Efficiency

- There are another two performance metrics we frequently use in the process of the course: efficiency and scalability.
- Efficiency:

  $$E = \frac{S}{P}, \tag{13}$$

  where S is the speedup, and P is the number of processes used to achieve the speedup. Note $0 < E \leq 1$. Efficiency is better when E is closer to 1, which simply means you utilized the P processors efficiently.

Scalability

- Scalability: In general, a technology is scalable if it can handle ever-increasing problem size.
- In parallel program performance, scalability refers to the following measure. Suppose we run a parallel program using certain number of processors and with a certain problem size, and obtained an efficiency E. Further suppose that now we want to increase the number of processors. In this case, if we can find a rate at which the problem size increases so that we can still maintain the efficiency E, we say the parallel program scalable.

WITS UNIVERSITY

Scalability cont.

Example 3

Suppose $T_{serial} = n$, where n is also the problem size. Also suppose $T_{parallel} = n/p + 1$. Then

$$E = \frac{n}{p(n/p+1)} = \frac{n}{n+p}.$$

To see if the problem is scalable, we increase p by a factor of k, then we get

$$E = \frac{n}{n+p} = \frac{xn}{xn+kp}.$$

If $x = k$, then we have the same efficiency. That is, if we increase the problem size at the same rate that we increase the number of processors, then the efficiency remain constant, hence, the program is scalable.

Scalability cont.

- **Strong scalability:** When we increase the number of processes, we can keep the efficiency fixed without increasing the problem size, the program is strongly scalable.
- **Weak scalability:** If we can keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes, then the program is said to be weakly scalable.

Summary