CSE5351: Parallel Processing

Part III

Performance Metrics and Benchmarks

- **How should one characterize the performance of applications and systems?**
- **What are user's requirements in performance and cost?**
- **How should one measure the performance of application programs?**
- **How should one characterize the system performance when a parallel program is executed on a parallel computer?**
- **What are the factors (parameters) affecting the performance?**
- **What are the typical parameter values in current systems?**
- **How should one quantify and analyze system scalability?**
- **How can one determine the scalability of a parallel computer executing a given application?**

System and Application Benchmarks

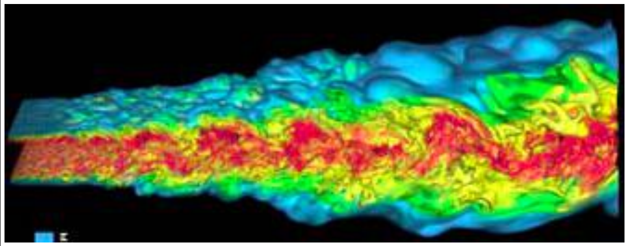
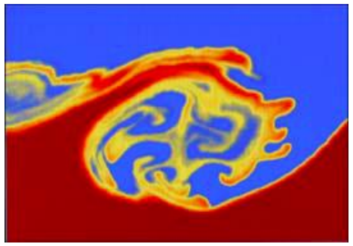
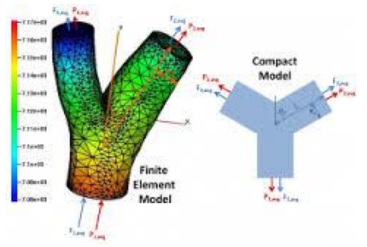
- **A *benchmark* is a performance testing program that is supposed to capture the processing and data movement characteristics of a class of applications.**
- **Benchmarks are used to measure and to predict the performance of computer systems, and to reveal their architectural weakness and strong points.**
- **A *benchmark suite* is a set of benchmark programs.**
- **A *benchmark family* is a set of benchmark suites.**
- **Benchmarks are classified as *scientific computing*, *commercial applications*, *network services*, *multimedia applications*, *signal processing*.**
- **Macro benchmarks and micro benchmarks.**
- **Applications or just *kernels.***
- ***Real work or a synthetic* program.**

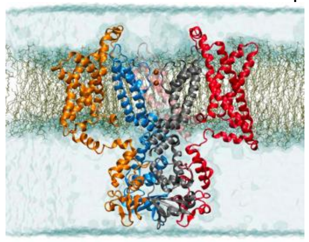
Many Other Benchmarks

- Top 500
- Green 500
- Sustained Petascale Performance
- HPC Challenge
- Perfect
- ParkBench
- SPEC-hpc
- Livermore Loops
- EuroBen
- NAS Parallel Benchmarks
- Genesis
- RAPS
- SHOC
- LAMMPS
- Dhrystone
- Whetstone

Goals for New Benchmark

- Augment the TOP500 listing with a benchmark that correlates with important scientific and technical apps not well represented by HPL



- Encourage vendors to focus on architecture features needed for high performance on those important scientific and technical apps.
  - Stress a balance of floating point and communication bandwidth and latency
  - Reward investment in high performance collective ops
  - Reward investment in high performance point-to-point messages of various sizes
  - Reward investment in local memory system performance
  - Reward investment in parallel runtimes that facilitate intra-node parallelism
- Provide an outreach/communication tool
  - Easy to understand
  - Easy to optimize
  - Easy to implement, run, and check results
- Provide a historical database of performance information
  - The new benchmark should have longevity

The NPB Suite

The *NAS Parallel Benchmarks* (NPB) is developed by the Numerical Aerodynamic Simulation (NAS) program at NASA Ames Research Center for the performance evaluation of parallel super-computers for large-scale *computational fluid dynamics* (CFD) applications.

The NPB suite consists of five kernels (EP, MG, CG, FT, IS) and three simulated applications (LU, SP, BT) programs:

- **EP (*Embarrassingly Parallel*)**
- **IS (*Integer Sorting*)**
- **The MG (*MultiGrid* method)**
- **The CG (*Conjugate Gradient* method)**
- **The FT (Fast Fourier Transform)**
- **BT (*Block Tri-diagonal*),**
- **LU (*block lower triangular, block upper triangular*)**
- **SP (*Scalar Penta-diagonal*)**

The PARKBENCH (*PARallel Kernels and BENCHmarks*)

There are four classes of benchmarks:

- ***Low-level benchmarks***
- ***Kernel benchmarks***
- ***Compact application benchmarks***
- ***HPF compiler benchmarks***

The Parallel STAP Suite

- The *Space-Time Adaptive Processing* (STAP) benchmark suite is a set of real-time, radar signal processing benchmark programs, originally developed at MIT Lincoln Laboratory.
- The STAP benchmark suite consists of five programs:

- *Adaptive Processing Testbed* (APT)

- *High-Order Post-Doppler* (HO-PD)

- *Beam Space PRI-Staggered Post Doppler* (BM-Stag)

- *Element Space PRI-Staggered Post Doppler* (EL-Stag)

- *General* (GEN)

Basic Performance Metrics

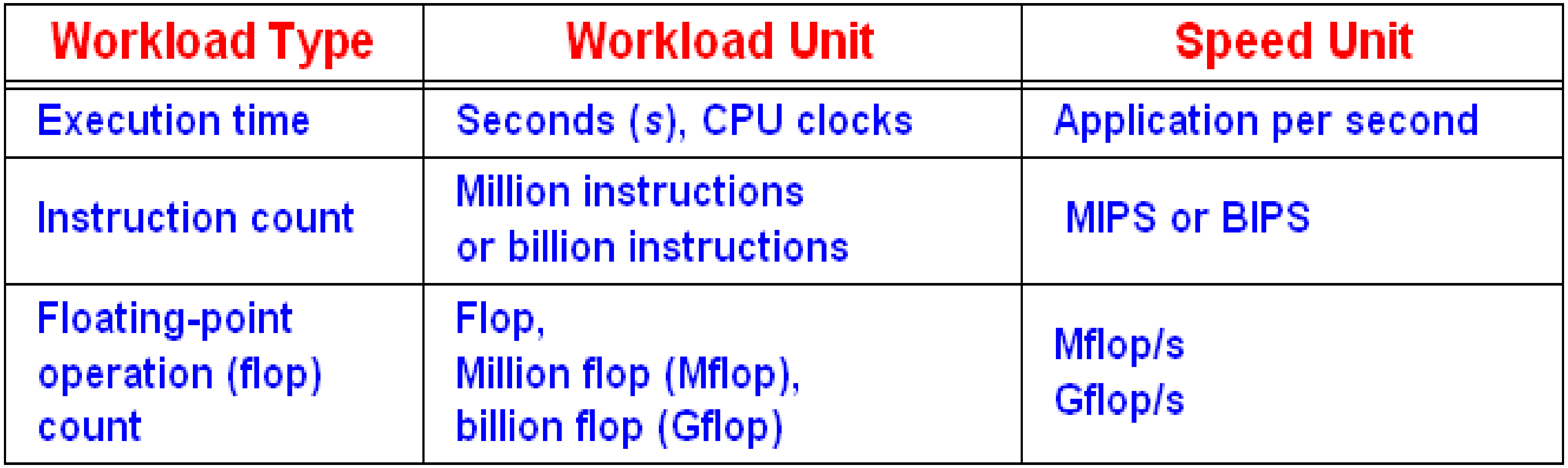
Workload and Speed Metrics

Three metrics are frequently used to measure the computational workload of a program:

- Execution time

- Number of instructions executed

- Number of floating-point operations executed



Workload Type	Workload Unit	Speed Unit
Execution time	Seconds (*s*), CPU clocks	Application per second
Instruction count	Million instructions or billion instructions	MIPS or BIPS
Floating-point operation (flop) count	Flop, Million flop (Mflop), billion flop (Gflop)	Mflop/s Gflop/s

Instruction Count

The workload is the instructions that the machine executed (i.e., *dynamic instruction count*), not just the number of instructions in the assembly program text (called *static program count*).

Instruction count may depend on:

- the input data values.

- machines.

- compilers or optimization options are used.

- A larger instruction count does not necessarily mean the program needs more time to execute.

Execution Time

- **For a given program on a specific computer system, one can define the workload as the total time taken to execute the program.**
- **This execution time should be measured in terms of *wallclock time*, also known as the *elapsed time*, obtained by, for instance, the Unix function gettimeofday().**

- **It should not be just the *CPU time*, which can be measured by the Unix function times().**
- **The basic unit of workload is seconds, although one can use minutes, hours, milliseconds (ms), or microseconds (μs) as well.**
- **The execution time depends on many factors:**

- Algorithm

- Data Structure

- Input Data

- Platform

- Language

Floating-Point Count

- **For scientific/engineering computing and signal processing applications where numerical calculation dominates.**

Parallelism and Interaction Overheads

The time to execute a parallel program is

$$T = T_{comp} + T_{par} + T_{interact}$$

where T_{comp}, T_{par}, and T_{interact} are the times needed to execute the computation, the parallelism, and the interaction operations, respectively.

- The overheads in a parallel program can be divided into three classes:

  - load-imbalance overhead

  - parallelism overhead

  - interaction overhead (including synchronization, communication, and aggregation)

Parallelism overhead

***Process management*, such as creation, termination, context switching, etc.**

***Grouping* operations, such as creation or destruction of a process group.**

***Process inquiry* operations, such as asking for process identification, rank, group identification, group size, etc.**

Interaction overhead

***Synchronization*, such as barrier, locks, critical regions, and events.**

***Aggregation*, such as reduction and scan.**

***Communication*, such as point-to-point and collective communication and reading/writing of shared variables.**

The ping-pong scheme to measure latency

```
for (i=0; i < *Runs*; i++)
 if (my_node_id ==0) {                          /* sender */
     tmp = Second();
     start_time = Second();
     send an *m*-byte message to node 1;
     receive an *m*-byte message from node 1;
     end_time = Second();
     timer_overhead = start_time - tmp ;
     total_time = end_time - start_time - timer_overhead ;
     communication_time[i] = total_time / 2 ;
   } else if (my_node_id ==1) {  /*receiver */
     receive an *m*-byte message from node 0;
     send an *m*-byte message to node 0;
   }
 }
```

Hot-potato (fire-brigade) method

- Instead of two nodes (the sender and the receiver), *n* nodes are involved.
- Node 0 sends a message of *m* bytes to node 1, which immediately sends the same message to node 2, and so on.
- Finally, node *n*–1 sends the message back to node 0.
- The total time is divided by *n* to get the point-to-point communication time.

Measuring collective communication performance

```
for (i=0; i < Runs; i++) {
        for (j=0; j< Iterations; j++)
        {
            barrier synchronization;
            tmp = Second();
            start_time = Second();
               the_collective_routine_being_measured;
            end_time = Second();
            timer_overhead = start_time - tmp;
            total_time += end_time - start_time - timer_overhead;
        }
    local_time = total_time / Iterations;
    communication_time[i] = maximum of all n local_time values;
  }
```

Ping-Pong Method for Collective Communication

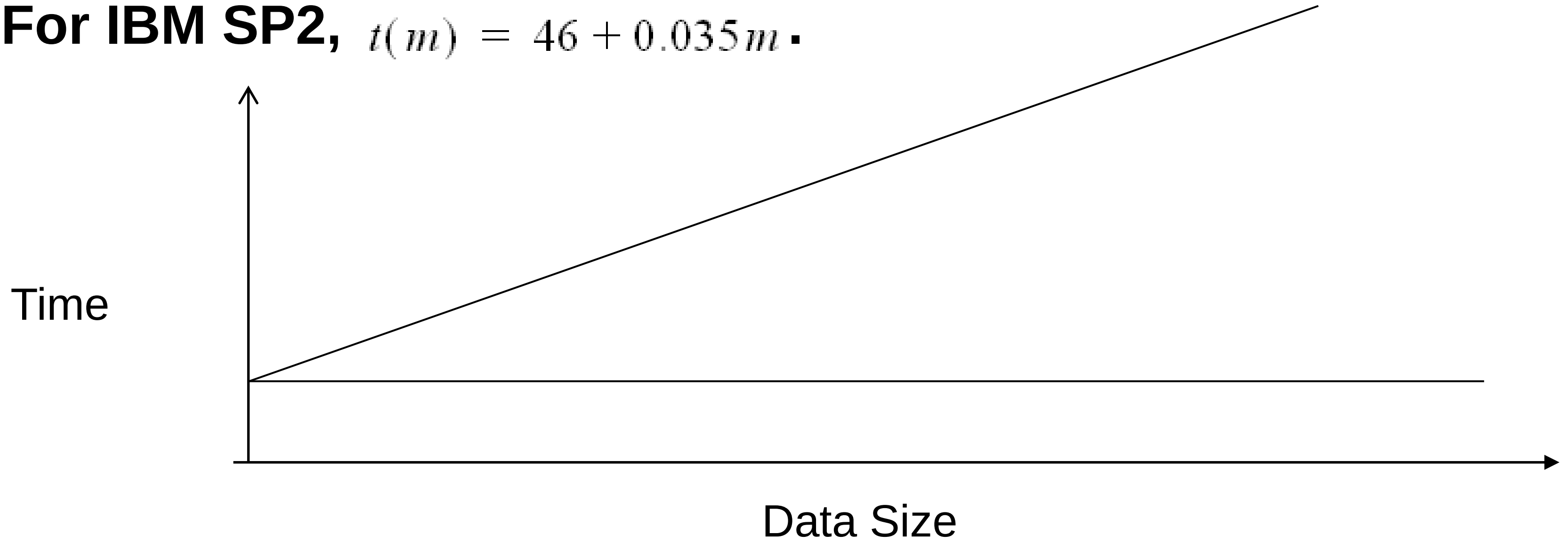
```
for ( i=0 ; i < Runs ; i++) {
 if (my_node_id ==0) {
     tmp = Second();
     start_time = Second();
   }
   node 0 broadcasts an empty message to all n nodes ;
   for ( j=0 ; j< Iterations ; j++)
     the_collective_routine_being_measured ;
   all nodes perform an empty reduction to node 0 ;
   if (my_node_id ==0) {
     end_time = Second();
     timer_overhead = start_time - tmp ;
     communication_time[i] =end_time - start_time - timer_overhead;
   }
 }
```

Point-to-Point Communication Model

- The *communication overhead, t*(*m*), is a linear function of the message length *m* (in bytes):

$$t(m) = t_0 + m / r_\infty$$

- t_0 is the *start-up time* in μs.
- r_∞ is the *asymptotic bandwidth* in MB/s.
- The *half-peak length*, denoted by $m_{1/2}$ bytes, is the message length required to achieve half of the asymptotic bandwidth.
- For IBM SP2, $t(m) = 46 + 0.035m$.

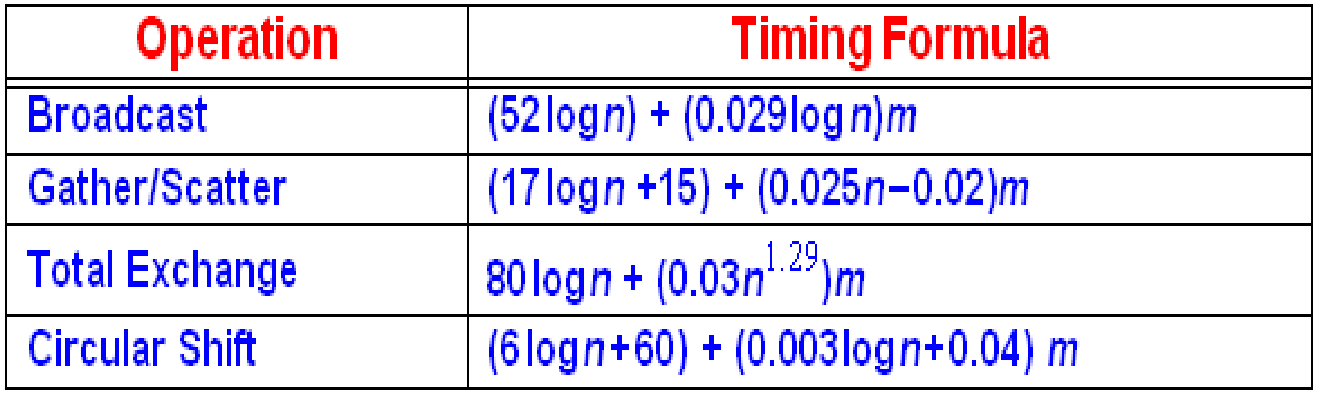


Collective Communication

- **In a *broadcast* operation, processor 0 sends an *m*-byte message to all *n* processors.**
- **In a *gather* operation, processor 0 receives an *m*-byte message from each of the *n* processors, so in the end, *mn* bytes are received by processor 0.**
- **In a *scatter* operation, processor 0 sends a distinct *m*-byte message to each of the *n* processors, so in the end, *mn* bytes are sent by processor 0.**
- **In a *total* exchange operation, every processor exchanges a distinct *m*-byte message to each of the *n* processors, so in the end, mn^2 bytes are communicated .**
- **In a *circular-shift* operation, processor *i* sends an *m*-byte message to processor *i* + 1, and processor *n* – 1 sends *m* bytes back to processor 0.**

Collective Communication Model

- The *collective communication overhead T*(*m, n*) is now a function of both *m* and *n*. The *start-up latency* depends only on *n.*
- $T(m, n) = t_0(n) + m/r_\infty(n)$
- *On IBM SP2, the derived formulas for the five collective operations are:*



Operation	Timing Formula
Broadcast	$(52 \log n) + (0.029 \log n)m$
Gather/Scatter	$(17 \log n + 15) + (0.025n - 0.02)m$
Total Exchange	$80 \log n + (0.03n^{1.29})m$
Circular Shift	$(6 \log n + 60) + (0.003 \log n + 0.04)\ m$

Amdahl's Law: Fixed Problem Size

Consider a problem with a fixed workload *W.*

Assume the workload *W* can be divided into two parts:

$W = \alpha W + (1-\alpha)W$ **, where α percent of *W* must be executed sequentially, and the remaining 1–α percent can be executed by *n* nodes simultaneously.**

Assuming all overheads are ignored, a *fixed-load speedup* is defined by:

$$S_n = \frac{W}{\alpha W + (1-\alpha)(W/n)} = \frac{n}{1+(n-1)\alpha} \rightarrow \frac{1}{\alpha} \qquad \text{as } n \rightarrow \infty$$

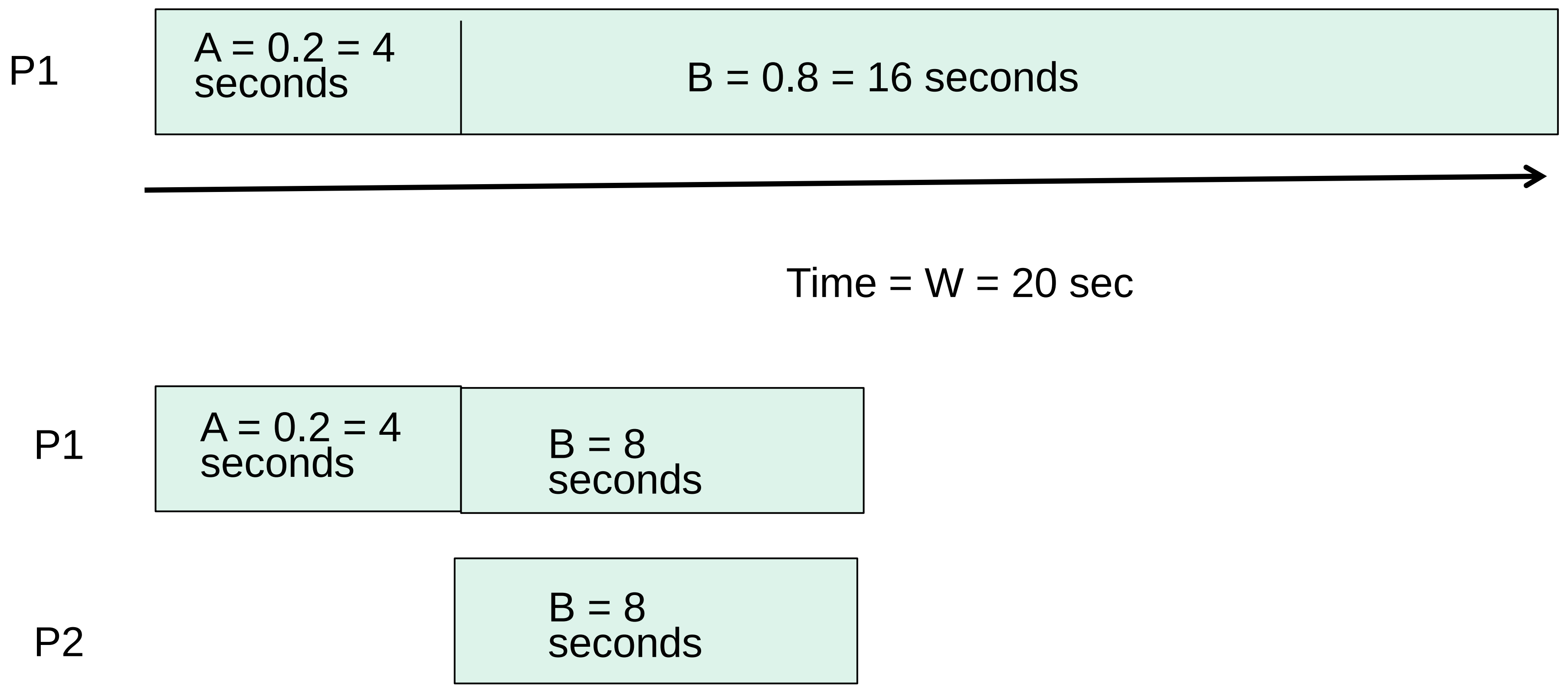
Amdahl's law has several implications:

- For a given workload, the maximal speedup has an upper-bound of 1/α

- To achieve good speedup, it is important to make the sequential bottleneck α as small as possible

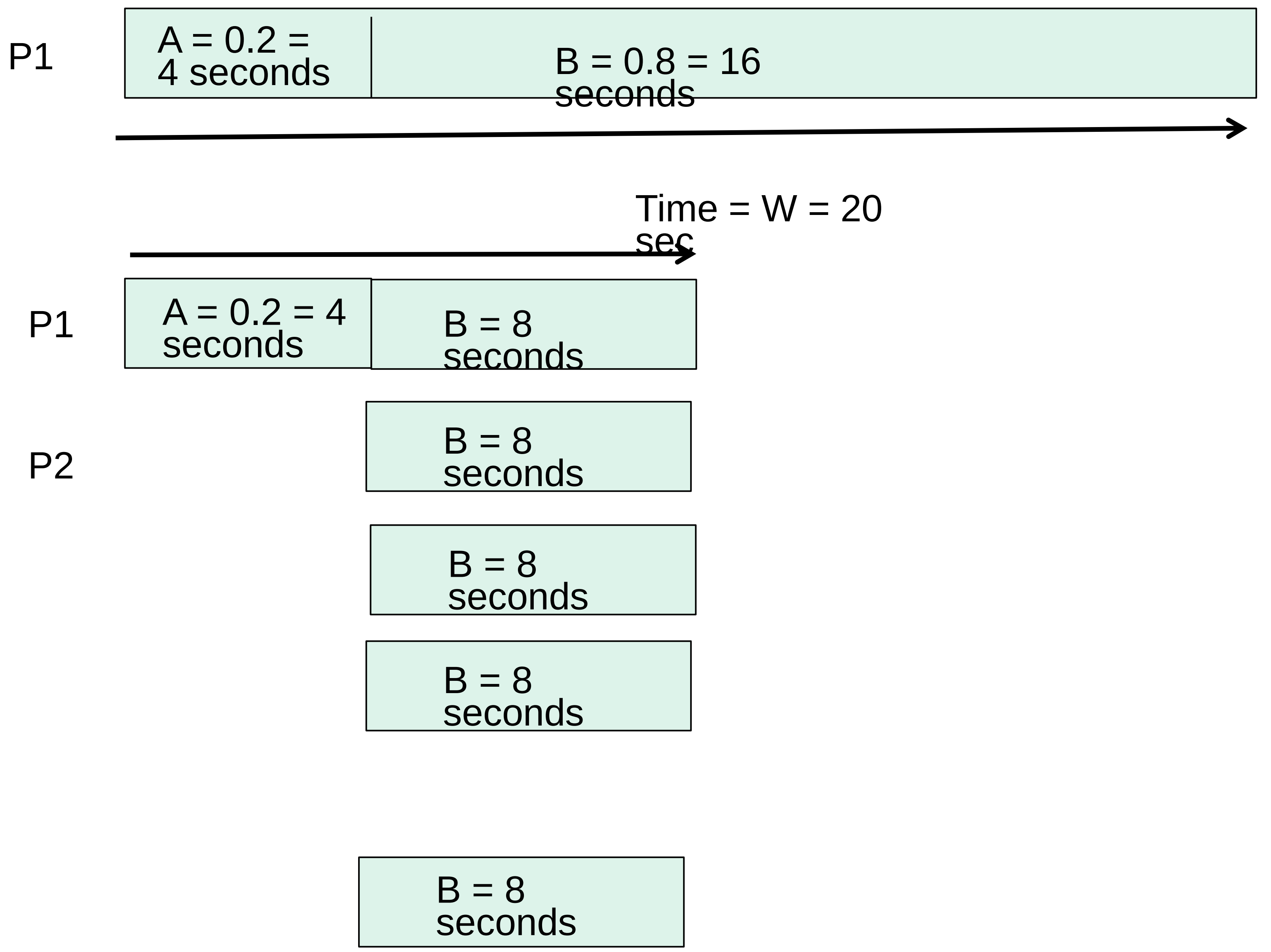
- When a problem consists of the above two portions, we should make the larger portion executed faster

Amdahl’s Law


P1
A = 0.2 = 4 seconds
B = 0.8 = 16 seconds
Time = W = 20 sec
P1
A = 0.2 = 4 seconds
B = 8 seconds
P2
B = 8 seconds

If the code has 1 percent code which cannot be parallelized, what is the maximum speedup using 10 processors?

Non-Amdahl's Law (Barsis-Gustafson)



Including the overheads, S_n becomes

$$S_n = \frac{W}{\alpha W + (1-\alpha)(W/n) + T_o}$$

$$= \frac{n}{1 + (n-1)\alpha + \dfrac{nT_o}{W}} \rightarrow \frac{1}{\alpha + \dfrac{T_o}{W}} \quad \text{as} \quad n \rightarrow \infty$$

- **The performance of a parallel program is limited not only by the sequential bottleneck, but also by the average overhead.**

A Critical Look at Amdhal's Law

- **The fixed-workload speedup drops as the sequential bottleneck α and the average overhead increase.**
- **The problem of a sequential bottleneck cannot be solved just by increasing the number of processors in a system.**
- **This property has imposed a very pessimistic view on parallel processing over the two decades.**
- **A major assumption in Amdahl's law is that the problem size (workload) is fixed and cannot scale to match the available computing power as the machine size increases.**
- **This often leads to a diminishing return when a larger system is employed to solve a small problem.**

Gustafson's Law: Fixed Time

- **As the machine size increases, the problem size may increase with execution time unchanged.**
- **Examples are computational fluid dynamics problems in weather forecasting.**
- **Let the original problem have a workload of *W*, of which** α **percent is sequential and** $1-\alpha$ **percent can be executed in parallel.**
- **On a single node machine, the execution time is *W*.**
- **On an *n*-node machine, we scale the workload to *W'* =** $\alpha W+(1-\alpha)nW$**. The parallel execution time on *n* nodes is still *W.***
- ***However, the sequential time for executing the scaled-up workload is W' =*** $\alpha W+(1-\alpha)nW$***.***

Scaled Speedup without Overhead

- **The *fixed-time speedup* with scaled workload is defined as:**

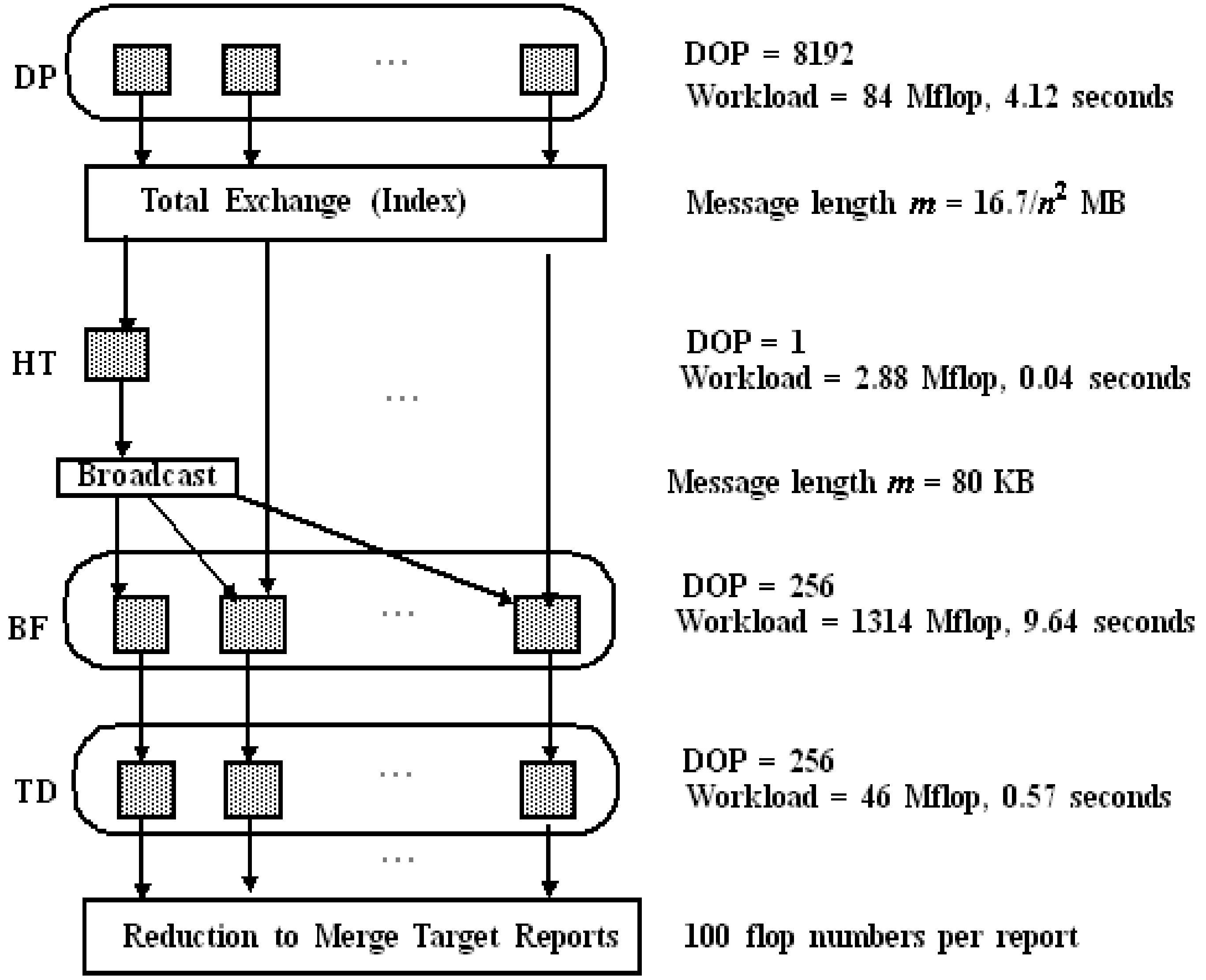
$$S_n' = \frac{\text{Sequential time for scaled-up workload}}{\text{Parallel time for scaled-up workload}}$$

$$? = \frac{\alpha W + (1-\alpha) n W}{W} = \alpha + (1-\alpha) n$$

- **Fixed-time speedup is a linear function of *n*, if the workload is scaled up to maintain a fixed execution time.**

Upper bound on the speedup of APT Benchmark

N is a problem parameter.


DP
Total Exchange (Index)
HT
Broadcast
BF
TD
Reduction to Merge Target Reports
DOP = 8192
Workload = 84 Mflop, 4.12 seconds
Message length $m = 16.7/n^2$ MB
DOP = 1
Workload = 2.88 Mflop, 0.04 seconds
Message length m = 80 KB
DOP = 256
Workload = 1314 Mflop, 9.64 seconds
DOP = 256
Workload = 46 Mflop, 0.57 seconds
100 flop numbers per report

Zero-overhead prediction

- **The maximal parallelism is max(8192, 1, 256, 256) = 8192.**
- **The total workload is W = 1447 Mflop.**
- **The sequential execution time is T_1 = 14.37 seconds.**
- **The critical path is:**

$$T_\infty = \frac{4.12}{8192} + \frac{0.04}{1} + \frac{9.64}{256} + \frac{0.57}{256} = 0.08 \text{ s}$$

- **Thus the maximum performance is:**

$$P_\infty = W / T_\infty = 1447 / 0.08 = 18087 \ \textbf{Mflop/s}$$

- **The average parallelism is:**

$$T_1 / T_\infty = 14.37 / 0.08 = 180$$

The interaction overhead in the APT benchmark

- **The interaction overhead is the sum of the three communications:**

$$T_{interact} = T_{exchange} + T_{bcast} + T_{reduce}$$

- **The overhead of total exchange of 16.7 MB is:**

$$T_{exchange} = 80\log n + 0.03n^{1.29} m \ \mu s = 0.00008\log n + 0.5n^{-0.71} \ \text{second}$$

- **The broadcast overhead is expressed by:**

$$T_{bcast} = 52\log n + (0.029\log n)m \ \mu s = 0.00237\log n \ \ \text{seconds}$$

- **The time to reduce *n* flop numbers, one from each of *n* nodes, is 20log*n*+23 μs. We need to combine *n* target reports, each having 100 flop numbers. The estimated reduction overhead is:**

$$T_{reduce} = 100(20\log n + 23) \ \mu s = 0.002\log n + 0.0023 \ \ \text{seconds}$$

- **The total interaction overhead is then:**

$$T_o = T_{interact} = 0.5n^{-0.71} + 0.00445\log n + 0.0023$$

Expected execution time of the APT benchmark

- Assume $n \le 256$. The total execution time using *n* nodes is:

$$T_n = T_{comp} + T_{par} + T_{interact}$$

$$? = \frac{14.33}{n} + 0.5n^{-0.71} + 0.00445 \log n + 0.0423$$

- The total workload is *W* = 1447 Mflop or 14.37 seconds on an IBM SP2 node.

Estimating the speedup using Amdahl ‘s law

- **Assume that $T_0(\infty) \approx T_0(256)$.**
- **When all overhead is ignored, the workload equals *W* = 14.37 s.**
- **The sequential component is the HT step, which accounts for α = 0.04/14.37 = 0.278%.**
- **By Amdahl’s law, the speedup has an upper-bound of 1/α = 359.**
- **T_o(256)=0.0479 s.**
- **The average overhead is $T_o(\infty)/W$ = 0.0479/14.37 = 0.00333.**
- **The speedup has an upper-bound of a tighter value 1/ (0.00278 + 0.00333) = 163.**

Scaled Speedup with Overhead

- The scaled speedup becomes:

$$S_n' = \frac{\alpha W + (1-\alpha) nW}{W + T_o} = \frac{\alpha + (1-\alpha)n}{1 + T_o / W}$$

Parallel program with the scaled workload *W'* = $\alpha W + (1-\alpha)$*nW* has the same fixed *computation time* as the sequential time for the original workload *W.*

- ***This overhead T_o*** **is a function of *n.***

- **T_o could have components that are increasing, or constant with respect to *n.***

- ***The generalized Gustafson's law can achieve a linear speedup by controlling the overhead as a decreasing function with respect to n*. But this is often difficult to achieve.**

Scaled speedup without no overhead in APT

- The workload can be approximated as $W = 0.011N + 2.8N + 2.88$ Mflop or $W = 0.00016N + 0.015N + 0.04$ s, for arbitrary *N*.
- For *N* = 256, *W* = 1447 Mflop or 14.37 seconds.
- The sequential execution time is *W* = 14.37 seconds.
- The essentially sequential HT step accounts for α = 0.04/ 14.37 = 0.278% of the total workload.
- For 128 nodes, the scaled speedup is:

$$S_n' = \alpha + (1-\alpha)n = 0.00278 + 0.99722n = 127.65$$

- The sequential time for the scaled workload is : $\alpha W+(1–\alpha)nW$=127.6514.37 = 1834 seconds. We can substitute this value in the time workload expression to solve for *N*: which yields an *N*=3339.

$$0.00016N^2 + 0.015N + 0.04 = 1834$$

- **Thus *N* is scaled up 3339/256=13 times.**
- **Substitute this value into the flop workload expression:**

$$W = 0.011N^2 + 2.8N + 2.88 = 131996$$

- **Therefore, 131,996 Mflops need to be performed in the scaled workload.**
- **The flop workload has been scaled up 131996/1447=91 times.**

APT Scaled Speedup with Oberheads

- The scaled speedup on 128 nodes when considering all overheads.
- Assume that the message length in the total exchange step is *m* = 256*N* ^{2}/*n* bytes. The communication overheads for the broadcast and the reduction steps do not change with *N*.
- The scaled speedup is obtained as follows:

$$S_n' = \frac{\alpha + (1-\alpha)n}{1 + T_o / W} = \frac{0.00278 + 0.99722n}{1 + T_o / W} = \frac{127.65}{1 + T_o / 14.37}$$

- The overhead for total exchange is:

$$T_{index} = 80\log n + 0.03n^{1.29} m \ \mu s = 0.00008\log n + 85.62n^{-0.71} \text{ seconds}$$

- The total overhead is calculated by:

$$T_o = T_{index} + T_{bcast} + T_{reduce} = 85.62n^{-0.71} + 0.00445\log n + 0.0023$$

- For n = 128, T_o is evaluated as 2.75 seconds.
- The scaled average overhead is T_o/W = 2.75/14.37 = 0.1914.
- The scaled speedup is 127.65/1.1914 = 107, which is less than the 127.65 speedup obtained when overheads are ignored.

Isoefficiency

- *Isoefficiency* metric is used to characterize system scalability.
- The efficiency of any system can be expressed as a function of the workload *W* and the machine size $E = f(W, n)$.
- If we fix the efficiency to some constant and solve the efficiency equation for workload *W*, the resulting function is called the *isoefficiency function* of the system.
- The smaller the isoefficiency, the less the workload needs to increase relative to the increase of machine size while keeping the same efficiency, thus the better the scalability of the system.

Two matrix multiplication schemes

- **Given two matrix multiplication schemes *A* and *B*.**
- ***The sequential time to multiply two*** $N \times N$ ***matrices is about*** $T_1 = cN^3$ ***and the efficiency is defined as*** $E = T_1 / (nT_n)$.

Schemes	Parallel Execution Time T_n	Efficiency *E*
A	$cN^3/n + bN^2/\sqrt{n}$	$E(A) = \dfrac{1}{1 + (b\sqrt{n})/(cN)}$
B	$2cN^3/n + bN^2/(2\sqrt{n})$	$E(B) = \dfrac{1}{2 + (b\sqrt{n})/(2cN)}$

Example

- **System *B* has only half the overhead of system *A*, but at the expense of doubling the computation time.**
- **Which system is more scalable, assuming we want to maintain (1) *E* = 1/3 and (2) *E* = 1/4?**
- **The workload of the $N \times N$ matrix multiplication problem is approximately** $W = N^3$ **.**

E = 1/3

- **For system *A*,** $(b\sqrt{n})/(cN) = 2$ **, thus the isoefficiency function is:**

$$W(A) = (b/(2c))^3 n^{1.5}$$

- **For system *B*, the isoefficiency function is:**

$$W(B) = (b/(2c))^3 n^{1.5}$$

- **Thus by isoefficiency, both systems are equally scalable.**

E = 1/4

- **For system *A*, $(b\sqrt{n})/(cN) = 3$, the isoefficiency function is:**

$$W(A) = (b/(3c))^3 n^{1.5}$$

- **Similarly, for system B, the isoefficiency function is:**

$$W(B) = (b/(4c))^3 n^{1.5}$$

- **Thus system *B* has a smaller isoefficiency and is more scalable than system *A.***

- ***To maintain a constant efficiency, doubling the machine size requires increasing the workload by a factor of 2.82.***

- ***A more scalable system by isoefficiency does not necessarily run faster.***