PERFORMANCE

Speedup

- Number of cores = p
- Serial run-time = T_{serial}
- Parallel run-time = $T_{parallel}$

linear speedup

$$T_{parallel} = T_{serial} / p$$

Speedup of a parallel program

$$S = \frac{T_{serial}}{T_{parallel}}$$

Efficiency of a parallel program

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

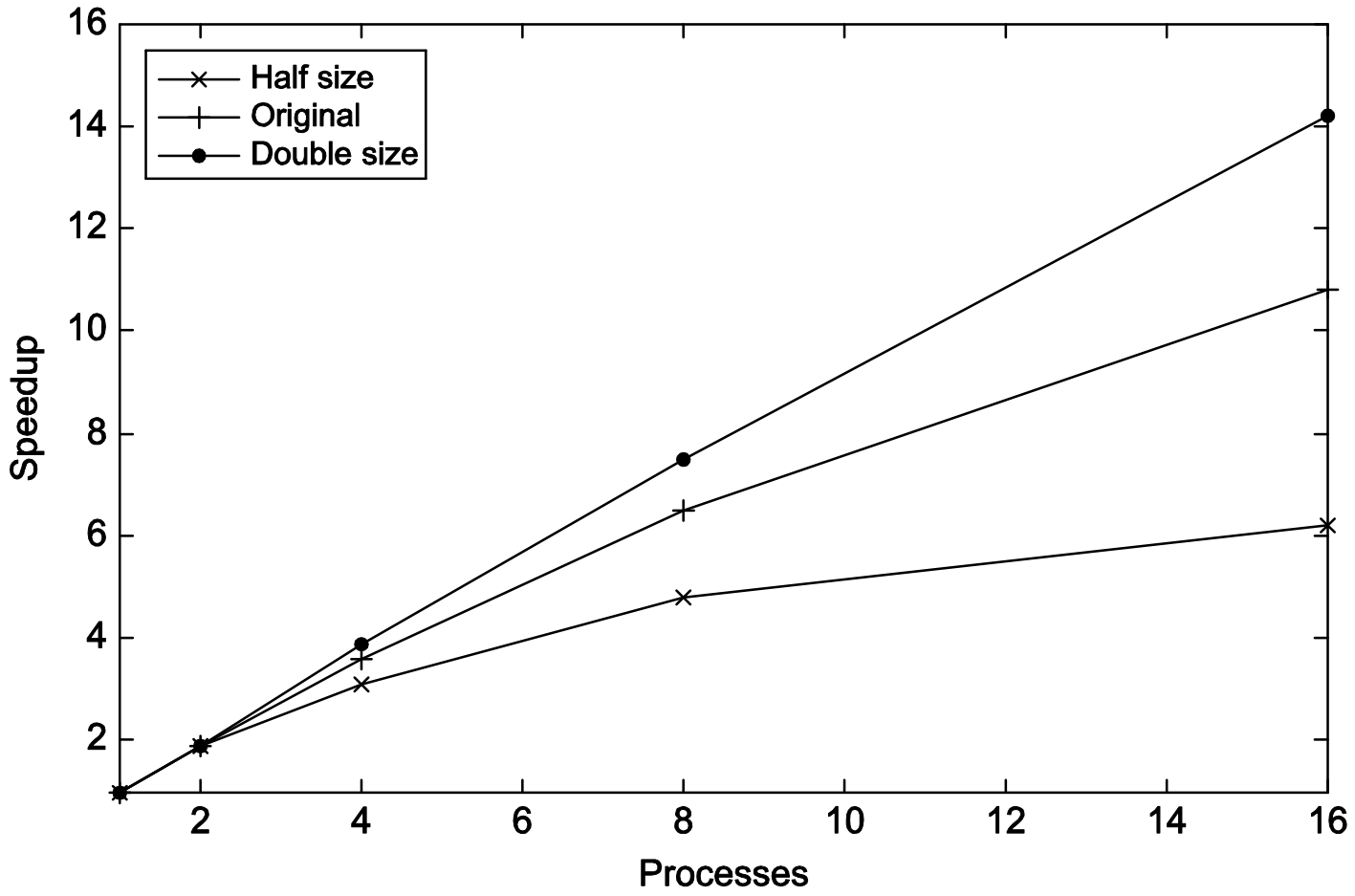
Speedups and efficiencies of a parallel program

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

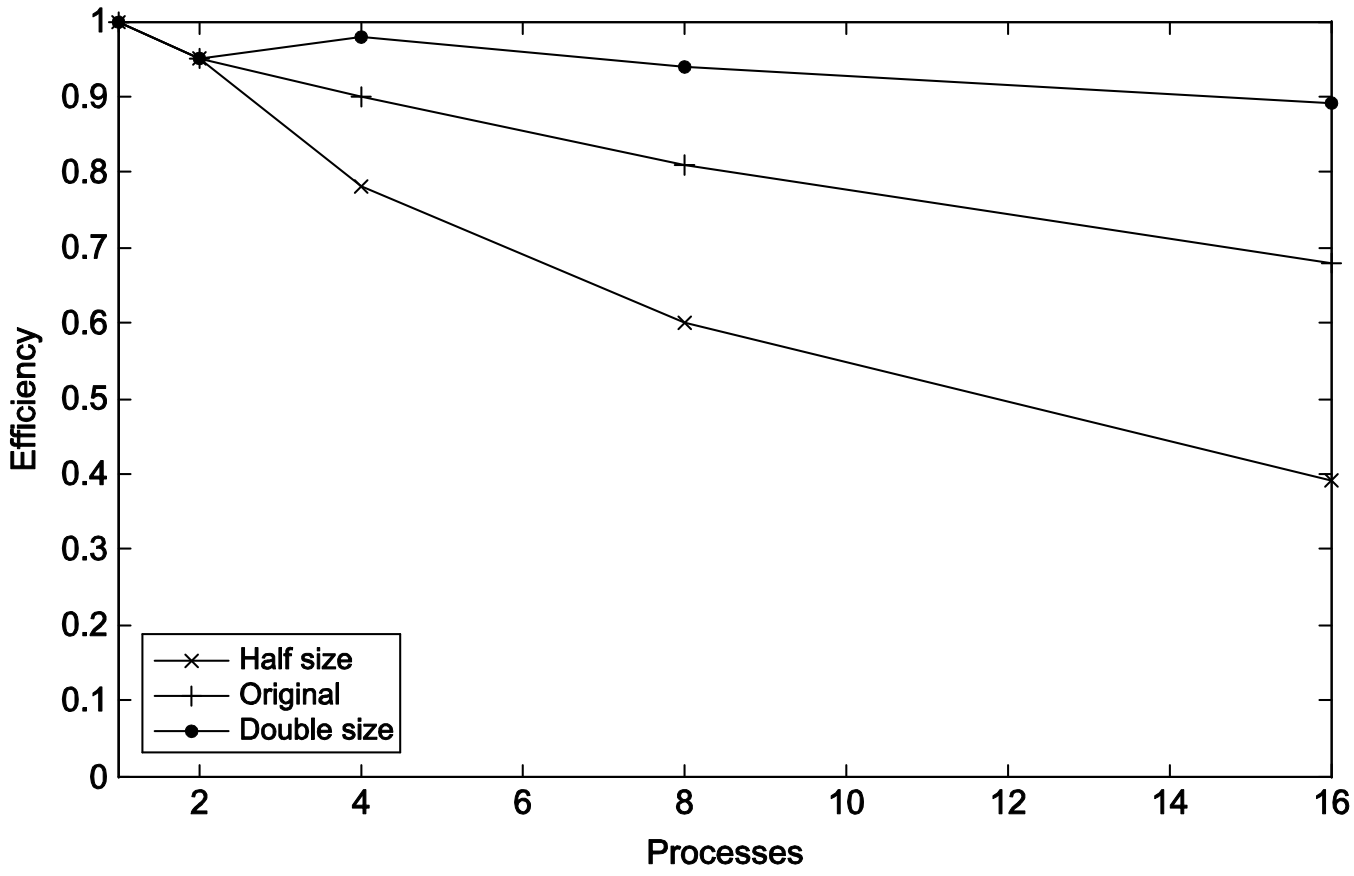
Speedups and efficiencies of parallel program on different problem sizes

	p	1	2	4	8	16
Half	S	1.0	1.9	3.1	4.8	6.2
	E	1.0	0.95	0.78	0.60	0.39
Original	S	1.0	1.9	3.6	6.5	10.8
	E	1.0	0.95	0.90	0.81	0.68
Double	S	1.0	1.9	3.9	7.5	14.2
	E	1.0	0.95	0.98	0.94	0.89

Speedup


Half size
Original
Double size
Speedup
Processes
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16

Efficiency


Efficiency
Processes
Half size
Original
Double size
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8
10
12
14
16

Effect of overhead

$$T_{parallel} = T_{serial} / p + T_{overhead}$$

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.

Example

- We can parallelize 90% of a serial program.
- Parallelization is “perfect” regardless of the number of cores p we use.
- T_{serial} = 20 seconds
- Runtime of parallelizable part is

$$0.9 \times T_{serial} / p = 18 / p$$

Example (cont.)

- Runtime of "unparallelizable" part is

$$0.1 \times T_{serial} = 2$$

- Overall parallel run-time is

$$T_{parallel} = 0.9 \times T_{serial} / p + 0.1 \times T_{serial} = 18 / p + 2$$

Example (cont.)

- Speed up

$$S = \frac{T_{serial}}{0.9 \times T_{serial} / p + 0.1 \times T_{serial}} = \frac{20}{18 / p + 2}$$

Scalability

- In general, a problem is scalable if it can handle ever increasing problem sizes.
- If we increase the number of processes/threads and keep the efficiency fixed without increasing problem size, the problem is strongly scalable.
- If we keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads, the problem is weakly scalable.

Taking Timings

- What is time?
- Start to finish?
- A program segment of interest?
- CPU time?
- Wall clock time?

Taking Timings

```
double start, finish;
. . .
start = Get_current_time();
/* Code that we want to time */
. . .
finish = Get_current_time();
printf("The elapsed time = %e seconds\n", finish-start);
```

theoretical function

MPI_Wtime

omp_get_wtime

Taking Timings

```
private double start, finish;
. . .
start = Get_current_time();
/* Code that we want to time */
. . .
finish = Get_current_time();
printf("The elapsed time = %e seconds\n", finish-start);
```

```
private double start, finish;
. . .
start = Get_current_time();
/* Code that we want to time */
. . .
finish = Get_current_time();
printf("The elapsed time = %e seconds\n", finish-start);
```

Taking Timings

```
shared double global_elapsed;
private double my_start, my_finish, my_elapsed;
. . .
/* Synchronize all processes/threads */
Barrier();
my_start = Get_current_time();

/* Code that we want to time */
. . .

my_finish = Get_current_time();
my_elapsed = my_finish - my_start;

/* Find the max across all processes/threads */
global_elapsed = Global_max(my_elapsed);
if (my_rank == 0)
   printf("The elapsed time = %e seconds\n", global_elapsed);
```

Concluding Remarks (1)

- Serial systems
  - The standard model of computer hardware has been the von Neumann architecture.
- Parallel hardware
  - Flynn's taxonomy.
- Parallel software
  - We focus on software for homogeneous MIMD systems, consisting of a single program that obtains parallelism by branching.
  - SPMD programs.

Concluding Remarks (2)

- Input and Output
  - We'll write programs in which one process or thread can access stdin, and all processes can access stdout and stderr.
  - However, because of nondeterminism, except for debug output we'll usually have a single process or thread accessing stdout.

Concluding Remarks (3)

- Performance
  - Speedup
  - Efficiency
  - Amdahl's law
  - Scalability
- Parallel Program Design
  - Foster's methodology