Parallel Computing for Science & Engineering

Introduction to Parallel Computing

TACC
THE UNIVERSITY OF TEXAS AT AUSTIN
Texas Advanced Computing Center

Outline

- Overview
- Theoretical background
- Parallel computing systems
- Parallel programming models
- MPI/OpenMP examples

OVERVIEW

What is Parallel Computing?

- Parallel computing: use of multiple processors or computers working together on a common task.
  - Each processor works on part of the problem
  - Processors can exchange information

The Basic Idea

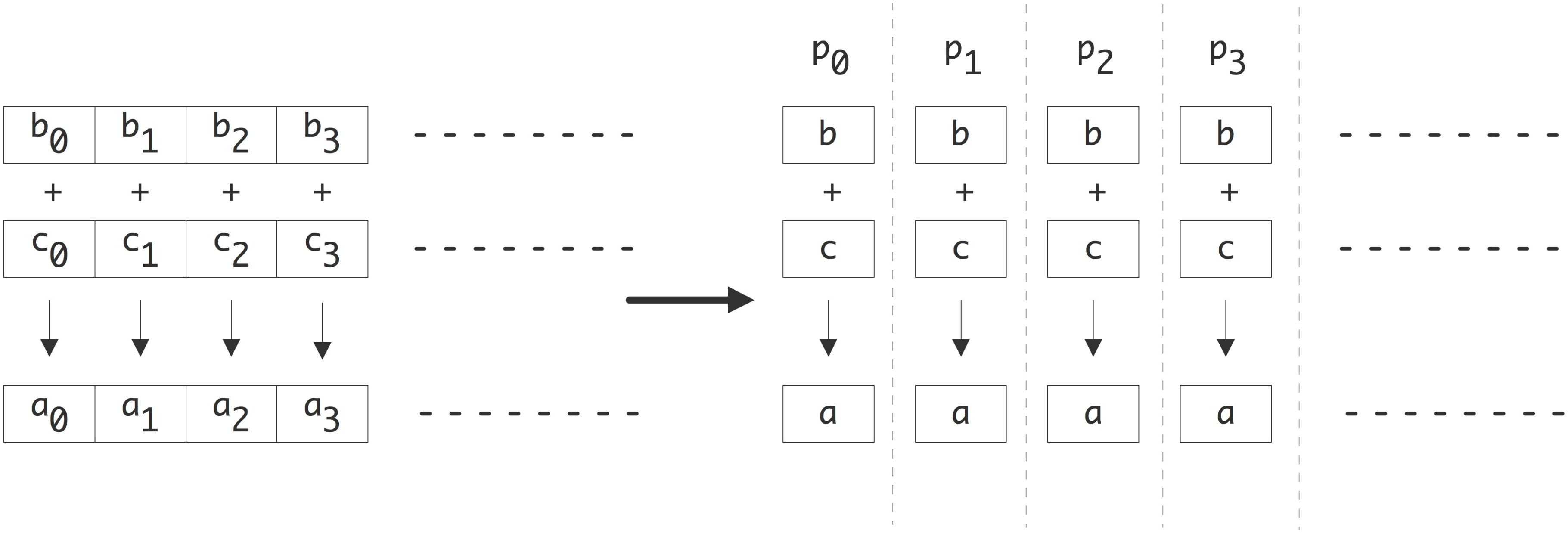
- Spread operations over many processors
- If *n* operations take time *t* on 1 processor,
- Does this become *t/p* on *p* processors (*p*<=*n*)?

```
for (i=0; i<n; i++)
  a[i] = b[i]+c[i];
```

```
a = b+c   !Fortran arrays
```

Idealized version: every process has one array element

The Basic Idea (Idealized Version)



The Basic Idea

- Spread operations over many processors
- If *n* operations take time *t* on 1 processor…
- …does this become *t/p* on *p* processors (*p*<=*n*)?

```
for (i=0; i<n; i++)
   a[i] = b[i]+c[i];
```

```
a = b+c
```

Idealized version:
every process has one
array element

```
for (i=my_low; i<my_high; i++)
   a[i] = b[i]+c[i];
```

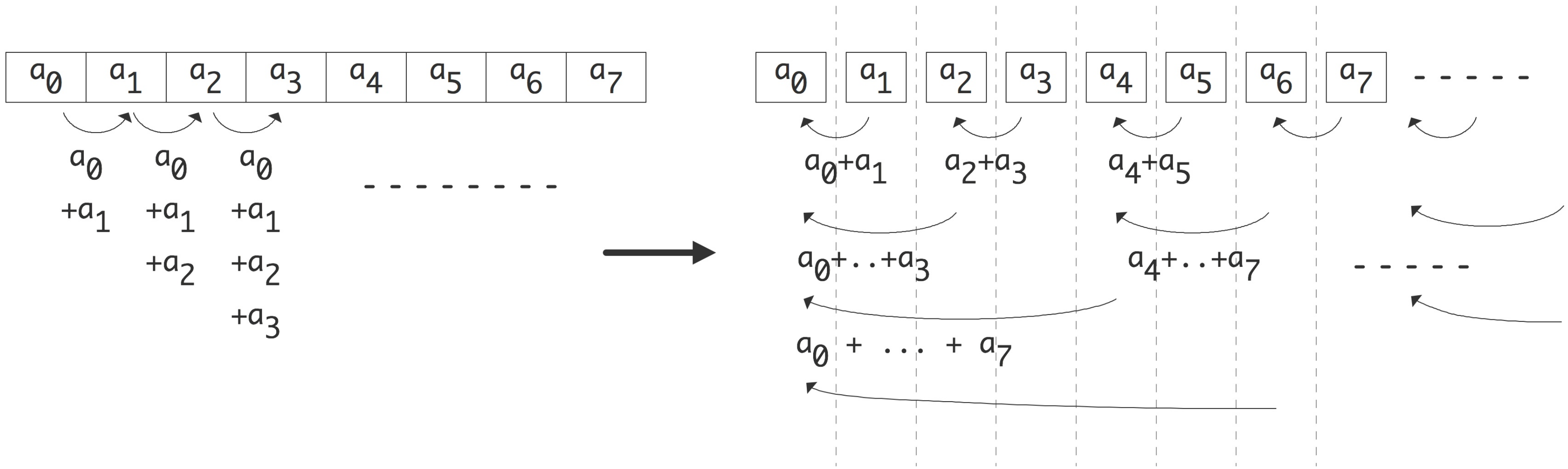
Slightly less ideal:
each processor has
part of the array

The Basic Idea (Continued)

- Spread operations over many processors
- If *n* operations take time *t* on 1 processor,
- Does it always become *t/p* on *p* processors (*p*<=*n*)?

```
s = sum( x[i], from i=0 to i=n-1 )
```

The Basic Idea (Continued)



The Basic Idea (Continued)

- Spread operations over many processors
- If *n* operations take time *t* on 1 processor,
- Does it always become *t/p* on *p* processors (*p<=n*)?

```
s = sum( x[i], from i=0 to i=n-1 )
```

```
for (p=0; p<n/2; p++)
  x[2p,0] = x[2p]+x[2p+1]
for (p=0; p<n/4; p++)
  x[4p,1] = x[4p]+x[4p+2]
for ( .. p<n/8 .. )

Et cetera
```

Conclusion: n operations can be done with n/2 processors, in total time log2n.

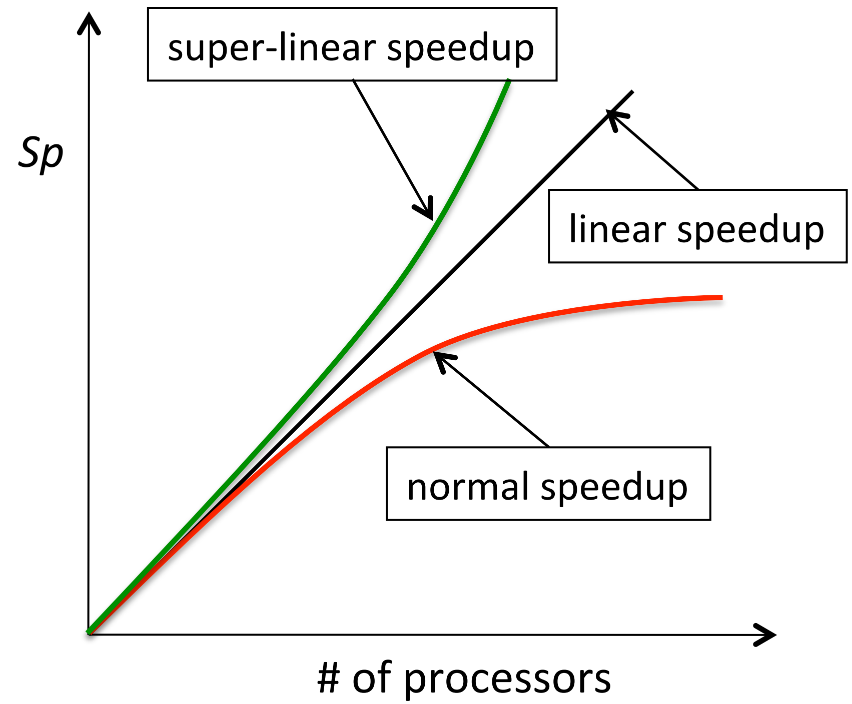
Theoretical question: can addition be done faster?

Practical question: can we even do this?

THEORETICAL BACKGROUND

Speedup & Parallel Efficiency

- Speedup: $$S_p = \frac{T_s}{T_p}$$
  - *p* = # of processors
  - *Ts* = execution time of the sequential algorithm
  - *Tp* = execution time of the parallel algorithm with *p* processors
  - *Sp*= *P* (linear speedup: ideal)



- Parallel efficiency: $$E_p = \frac{S_p}{p} = \frac{T_s}{pT_p}$$

Limits of Parallel Computing

- Theoretical Upper Limits
  - Amdahl's Law
- Practical Limits
  - Load balancing
  - Non-computational sections
- Other Considerations
  - Time to re-write code

Amdahl's Law

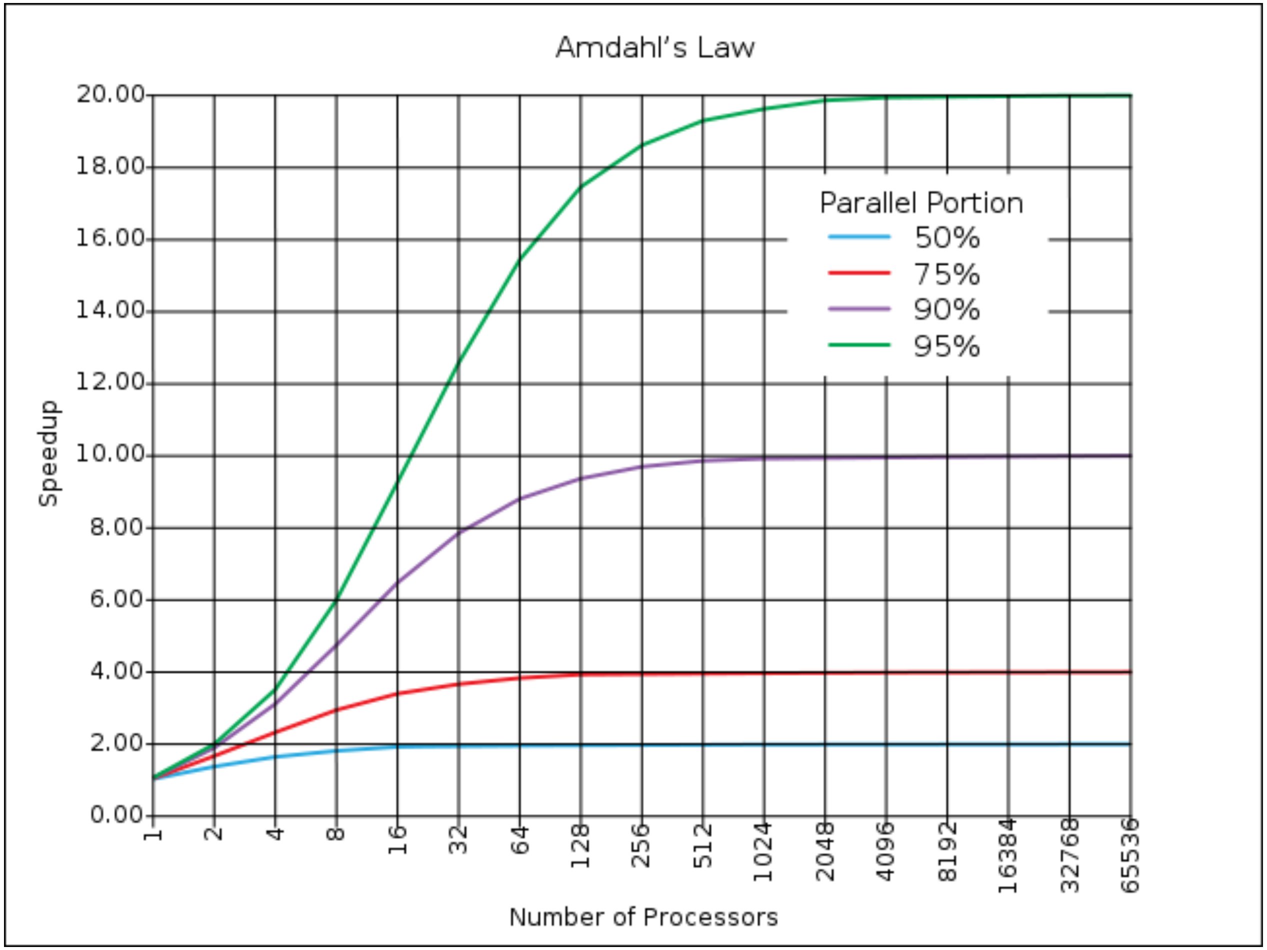
- All parallel programs contain parallel sections and serial sections
- Serial sections limit the parallel effectiveness
- Amdahl's Law states this formally
  - Effect of multiple processors on speed up

$$S_P \le \frac{T_S}{T_P} = \frac{1}{f_s + \frac{f_p}{P}} \rightarrow \frac{1}{f_s}, p \rightarrow \infty$$

where

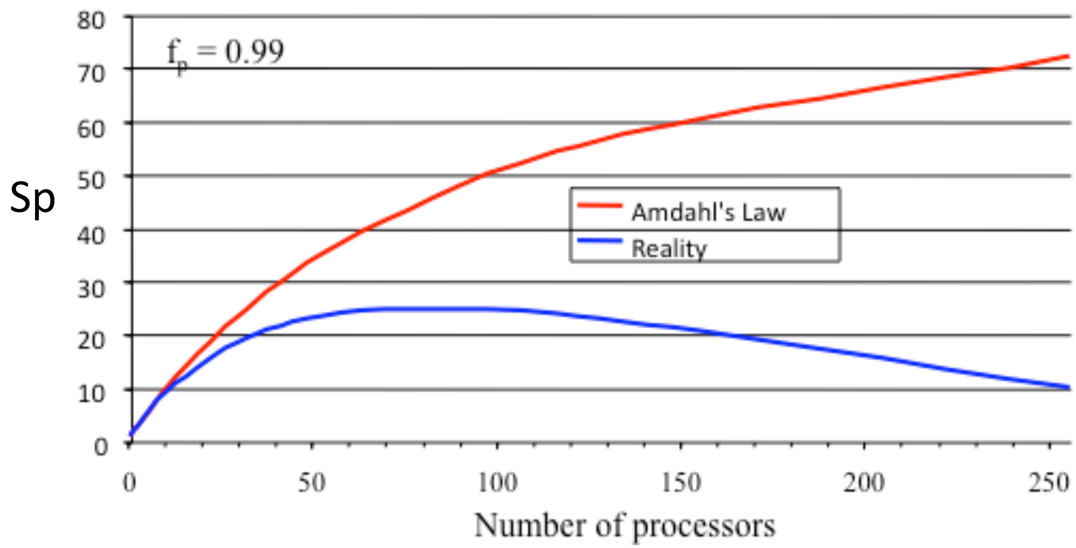
- f_s = serial fraction of code
- f_p = parallel fraction of code
- *P* = number of processors

Amdahl's Law


Amdahl's Law
Parallel Portion
50%
75%
90%
95%
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

Practical Limits: Amdahl's Law vs. Reality

- In reality, the situation is even worse than predicted by Amdahl's Law due to:
  - Load balancing (waiting)
  - Scheduling (shared processors or memory)
  - Cost of Communications
  - I/O



Gustafson's Law

- Effect of multiple processors on run time of a problem with a *fixed amount of parallel work per processor.*

$$S_P \leq P - \alpha \cdot (P-1)$$

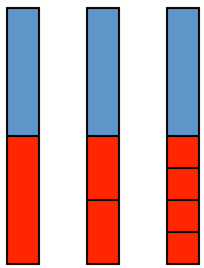
  - α is the fraction of non-parallelized code where the parallel work per processor is fixed (not the same as f_p from Amdahl's)
  - P is the number of processors

TACC

Comparison of Amdahl and Gustafson

Amdahl : fixed work

$$f_p = 0.5$$



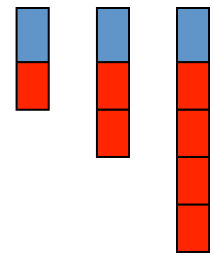
$$S \le \frac{1}{f_s + f_p / N}$$

$$S_2 \le \frac{1}{0.5 + 0.5/2} = 1.3$$

$$S_4 \le \frac{1}{0.5 + 0.5/4} = 1.6$$

Gustafson : fixed work per processor

$$\alpha = 0.5$$



$$S_p \le P - \alpha \cdot (P - 1)$$

$$S_2 \le 2 - 0.5(2-1) = 1.5$$

$$S_4 \le 4 + 0.5(4-1) = 2.5$$

Scaling: Strong vs. Weak

- We want to know how quickly we can complete analysis on a particular data set by increasing the PE count
  - Amdahl's Law
  - Known as "strong scaling"

- We want to know if we can analyze more data in approximately the same amount of time by increasing the PE count
  - Gustafson's Law
  - Known as "weak scaling"

We gratefully acknowledge the sponsorship of Chevron Corporation, whose generous support of TACC has made possible this Scientific Computing Curriculum and other student-focused initiatives.

License