An introduction to parallel algorithms

Knut Mørken

Department of Informatics
Centre of Mathematics for Applications
University of Oslo

Winter School on Parallel Computing
Geilo
January 20–25, 2008

Adding n numbers

Suppose we have n real numbers $(a_i)_{i=1}^n$ and want to compute their sum s.

In mathematics

$$s = \sum_{i=1}^{n} a_i.$$

Adding n numbers

Suppose we have n real numbers $(a_i)_{i=1}^n$ and want to compute their sum s.

In mathematics

$$s = \sum_{i=1}^{n} a_i.$$

On a computer

$s = 0;$
for $i = 1, 2, \ldots, n$
    $s = s + a_i;$

Algorithm

An algorithm is a precise prescription of how to accomplish a task.

Algorithm

An algorithm is a precise prescription of how to accomplish a task.

Two important issues determine the character of an algorithm:

- Which operations are available to us?

Algorithm

An algorithm is a precise prescription of how to accomplish a task.

Two important issues determine the character of an algorithm:

- Which operations are available to us?
- In which order can the operations be performed?

Algorithm

An algorithm is a precise prescription of how to accomplish a task.

Two important issues determine the character of an algorithm:

- Which operations are available to us?
- In which order can the operations be performed?
    - One at a time (sequentially).

Algorithm

An algorithm is a precise prescription of how to accomplish a task.

Two important issues determine the character of an algorithm:

- Which operations are available to us?
- In which order can the operations be performed?
  - One at a time (sequentially).
  - Several at once (in parallel).

Primitive operations

We will assume that the basic operations available to us are

- The four arithmetic operations (with two arguments)
- Comparison of numbers (if-tests)
- Elementary mathematical functions (trigonometric, exponential, logarithms, roots, . . .)

Primitive operations

We will assume that the basic operations available to us are

- The four arithmetic operations (with two arguments)
- Comparison of numbers (if-tests)
- Elementary mathematical functions (trigonometric, exponential, logarithms, roots, . . .)

We will measure computing time (complexity) by counting the number of time steps necessary to complete all the arithmetic operations of an algorithm.

Order of operations

In traditional (sequential) programming it is assumed that a computer can only perform one operation at a time.

Adding n numbers

$s = 0$;
for $i = 1, 2, \ldots, n$
    $s = s + a_i$;

Order of operations

In traditional (sequential) programming it is assumed that a computer can only perform one operation at a time.

Adding n numbers

$s = 0;$
for $i = 1, 2, \ldots, n$
$\quad s = s + a_i;$

Mathematically, however, many problems exhibit considerable freedom in the order in which operations are performed.

Order of operations

We can add the n numbers in any order

$$\sum_{i=1}^{n} a_i = a_{\pi_1} + a_{\pi_2} + \cdots + a_{\pi_n}$$

where $(\pi_1, \ldots, \pi_n)$ is any permutation of the integers 1, ..., n.

This can be exploited to speed up the addition if we have the possibility of performing several operations simultaneously.

Parallel addition

Suppose that $n = 10$. Then we have

$$s = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 + a_9 + a_{10},$$

Parallel addition

Suppose that $n = 10$. Then we have

$$s = \underbrace{a_1 + a_2}_{a_1^1} + \underbrace{a_3 + a_4}_{a_2^1} + \underbrace{a_5 + a_6}_{a_3^1} + \underbrace{a_7 + a_8}_{a_4^1} + \underbrace{a_9 + a_{10}}_{a_5^1},$$

Parallel addition

Suppose that $n = 10$. Then we have

$$
\begin{aligned}
s &= \underbrace{a_1 + a_2}_{a_1^1} + \underbrace{a_3 + a_4}_{a_2^1} + \underbrace{a_5 + a_6}_{a_3^1} + \underbrace{a_7 + a_8}_{a_4^1} + \underbrace{a_9 + a_{10}}_{a_5^1}, \\
&= a_1^1 + a_2^1 + a_3^1 + a_4^1 + a_5^1,
\end{aligned}
$$

Parallel addition

Suppose that $n = 10$. Then we have

$$s = \underbrace{a_1 + a_2}_{a_1^1} + \underbrace{a_3 + a_4}_{a_2^1} + \underbrace{a_5 + a_6}_{a_3^1} + \underbrace{a_7 + a_8}_{a_4^1} + \underbrace{a_9 + a_{10}}_{a_5^1},$$

$$= \underbrace{a_1^1 + a_2^1}_{a_1^2} + \underbrace{a_3^1 + a_4^1}_{a_2^2} + \underbrace{a_5^1}_{a_3^2},$$

Parallel addition

Suppose that $n = 10$. Then we have

$$\begin{aligned} s &= \underbrace{a_1 + a_2}_{a_1^1} + \underbrace{a_3 + a_4}_{a_2^1} + \underbrace{a_5 + a_6}_{a_3^1} + \underbrace{a_7 + a_8}_{a_4^1} + \underbrace{a_9 + a_{10}}_{a_5^1}, \\ &= \underbrace{a_1^1 + a_2^1}_{a_1^2} + \underbrace{a_3^1 + a_4^1}_{a_2^2} + \underbrace{a_5^1}_{a_3^2}, \\ &= a_1^2 + a_2^2 + a_3^2, \end{aligned}$$

Parallel addition

Suppose that $n = 10$. Then we have

$$\begin{aligned} s &= \underbrace{a_1 + a_2}_{a_1^1} + \underbrace{a_3 + a_4}_{a_2^1} + \underbrace{a_5 + a_6}_{a_3^1} + \underbrace{a_7 + a_8}_{a_4^1} + \underbrace{a_9 + a_{10}}_{a_5^1}, \\ &= \underbrace{a_1^1 + a_2^1}_{a_1^2} + \underbrace{a_3^1 + a_4^1}_{a_2^2} + \underbrace{a_5^1}_{a_3^2}, \\ &= \underbrace{a_1^2 + a_2^2}_{a_1^3} + \underbrace{a_3^2}_{a_2^3}, \end{aligned}$$

Parallel addition

Suppose that $n = 10$. Then we have

$$\begin{aligned} s &= \underbrace{a_1 + a_2}_{a_1^1} + \underbrace{a_3 + a_4}_{a_2^1} + \underbrace{a_5 + a_6}_{a_3^1} + \underbrace{a_7 + a_8}_{a_4^1} + \underbrace{a_9 + a_{10}}_{a_5^1}, \\ &= \underbrace{a_1^1 + a_2^1}_{a_1^2} + \underbrace{a_3^1 + a_4^1}_{a_2^2} + \underbrace{a_5^1}_{a_3^2}, \\ &= \underbrace{a_1^2 + a_2^2}_{a_1^3} + \underbrace{a_3^2}_{a_2^3}, \\ &= a_1^3 + a_2^3. \end{aligned}$$

Parallel addition

This means that if we have 5 computing units at our disposal, we can add the 10 numbers in 4 time steps.

In general, this technique allows us to add n numbers in $\lceil \log_2 n \rceil$ time steps if we have $n/2$ computing units.

This means that if we have 5 computing units at our disposal, we can add the 10 numbers in 4 time steps.

In general, this technique allows us to add n numbers in $\lceil \log_2 n \rceil$ time steps if we have $n/2$ computing units.

Adding n numbers

$\boldsymbol{a}^0 = \boldsymbol{a}$;
for $j = 1, 2, \ldots, \lceil \log_2 n \rceil$
    parallel: $a^j_{i/2} = a^{j-1}_{i-1} + a^{j-1}_i, \quad i = 2, 4, 6, \ldots, |\boldsymbol{a}^{j-1}|$;

This means that if we have 5 computing units at our disposal, we can add the 10 numbers in 4 time steps.

In general, this technique allows us to add n numbers in $\lceil \log_2 n \rceil$ time steps if we have $n/2$ computing units.

Adding n numbers

$\boldsymbol{a}^0 = \boldsymbol{a}$;
for $j = 1, 2, \ldots, \lceil \log_2 n \rceil$
    parallel: $a^j_{i/2} = a^{j-1}_{i-1} + a^{j-1}_i, \quad i = 2, 4, 6, \ldots, |\boldsymbol{a}^{j-1}|$;
    if $|\boldsymbol{a}^{j-1}| \,\%\, 2 > 0$ then $a^j_{-1} = a^{j-1}_{-1}$;

Parallel addition

There is a recursive version of the previous algorithm where each function call is given to a new processor.

Adding n numbers (recursive version)

sum(a, n)
{
    if $n > 1$ then
        return(sum($\boldsymbol{a}, 1, n//2$) $+$ sum($\boldsymbol{a}, n//2 + 1, n$))
    else
        return(a_1);
}

Problem

Add 1000 integers, each with 5 digits.

Parallel computing in practice

Problem

Add 1000 integers, each with 5 digits.

Resources

This group of people.

Parallel computing in practice

Problem

Add 1000 integers, each with 5 digits.

Resources

This group of people.

Method

1. While more than one number:
    1. I share out the numbers evenly, at least two numbers each
    2. You add your numbers
    3. You pass your results back to me

Dot product

If $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$ then

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i.$$

Dot product

If $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$ then

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i.$$

Provided we have n processors this can be computed in $\lceil \log_2 n \rceil + 1$ time steps:

Inner product

1. Compute the n products in parallel.
2. Compute the sum.

Matrix multiplication

Let $\boldsymbol{A} \in \mathbb{R}^{n,n}$ and $\boldsymbol{B} \in \mathbb{R}^{n,n}$. Then the product $\boldsymbol{AB}$ is a matrix in $\mathbb{R}^{n,n}$ which is given by

$$\boldsymbol{AB} = \begin{pmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \vdots \\ \boldsymbol{a}_n \end{pmatrix} \begin{pmatrix} \boldsymbol{b}_1 & \boldsymbol{b}_2 & \cdots & \boldsymbol{b}_n \end{pmatrix}$$

$$= \begin{pmatrix} \boldsymbol{a}_1 \cdot \boldsymbol{b}_1 & \boldsymbol{a}_1 \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_1 \cdot \boldsymbol{b}_n \\ \boldsymbol{a}_2 \cdot \boldsymbol{b}_1 & \boldsymbol{a}_2 \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_2 \cdot \boldsymbol{b}_n \\ \vdots & \vdots & \ddots & \dots \\ \boldsymbol{a}_n \cdot \boldsymbol{b}_1 & \boldsymbol{a}_n \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_n \cdot \boldsymbol{b}_n \end{pmatrix}.$$

Parallel matrix multiplication

Recall that one dot product of length n can be computed in $\lceil \log_2 n \rceil + 1$ time steps provided we have n processors at our disposal.

Parallel matrix multiplication

Recall that one dot product of length n can be computed in $\lceil \log_2 n \rceil + 1$ time steps provided we have n processors at our disposal.

Since all the n^2 dot products in the matrix product are independent, we can also compute $\boldsymbol{AB}$ in

$$\lceil \log_2 n \rceil + 1$$

time steps, provided we may use n^3 processors.

Parallel matrix multiplication

Alternative algorithm

Give each of the n^2 dot products in

$$\begin{pmatrix} \boldsymbol{a}_1 \cdot \boldsymbol{b}_1 & \boldsymbol{a}_1 \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_1 \cdot \boldsymbol{b}_n \\ \boldsymbol{a}_2 \cdot \boldsymbol{b}_1 & \boldsymbol{a}_2 \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_2 \cdot \boldsymbol{b}_n \\ \vdots & \vdots & \ddots & \dots \\ \boldsymbol{a}_n \cdot \boldsymbol{b}_1 & \boldsymbol{a}_n \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_n \cdot \boldsymbol{b}_n \end{pmatrix}.$$

to different processors.

Parallel matrix multiplication

Alternative algorithm

Give each of the n^2 dot products in

$$\begin{pmatrix} \boldsymbol{a}_1 \cdot \boldsymbol{b}_1 & \boldsymbol{a}_1 \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_1 \cdot \boldsymbol{b}_n \\ \boldsymbol{a}_2 \cdot \boldsymbol{b}_1 & \boldsymbol{a}_2 \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_2 \cdot \boldsymbol{b}_n \\ \vdots & \vdots & \ddots & \ldots \\ \boldsymbol{a}_n \cdot \boldsymbol{b}_1 & \boldsymbol{a}_n \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_n \cdot \boldsymbol{b}_n \end{pmatrix}.$$

to different processors.

Accumulation of the dot product on one processor requires n time steps, provided we have n^2 processors.

Strassen's algorithm (sequential)

The product of the two block matrices

$$\begin{pmatrix} \boldsymbol{C}_{1,1} & \boldsymbol{C}_{1,2} \\ \boldsymbol{C}_{2,1} & \boldsymbol{C}_{2,2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} \end{pmatrix} \begin{pmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{B}_{1,2} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{pmatrix}$$

Strassen's algorithm (sequential)

The product of the two block matrices

$$\begin{pmatrix} \boldsymbol{C}_{1,1} & \boldsymbol{C}_{1,2} \\ \boldsymbol{C}_{2,1} & \boldsymbol{C}_{2,2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} \end{pmatrix} \begin{pmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{B}_{1,2} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{pmatrix}$$

can be computed by first calculating

$$\begin{aligned} \boldsymbol{H}_1 &= \boldsymbol{A}_{1,1}(\boldsymbol{B}_{1,2} - \boldsymbol{B}_{2,2}), \\ \boldsymbol{H}_2 &= \boldsymbol{A}_{2,2}(\boldsymbol{B}_{2,1} - \boldsymbol{B}_{1,1}), \\ \boldsymbol{H}_3 &= (\boldsymbol{A}_{2,1} + \boldsymbol{A}_{2,2})\boldsymbol{B}_{1,1}, \\ \boldsymbol{H}_4 &= \boldsymbol{a}_{2,2}(\boldsymbol{B}_{2,1} - \boldsymbol{B}_{1,1}), \end{aligned}$$

Strassen's algorithm (sequential)

The product of the two block matrices

$$\begin{pmatrix} \boldsymbol{C}_{1,1} & \boldsymbol{C}_{1,2} \\ \boldsymbol{C}_{2,1} & \boldsymbol{C}_{2,2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} \end{pmatrix} \begin{pmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{B}_{1,2} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{pmatrix}$$

can be computed by first calculating

$$\begin{aligned} \boldsymbol{H}_1 &= \boldsymbol{A}_{1,1}(\boldsymbol{B}_{1,2} - \boldsymbol{B}_{2,2}), & \boldsymbol{H}_5 &= (\boldsymbol{A}_{1,1} + \boldsymbol{A}_{2,2})(\boldsymbol{B}_{1,1} + \boldsymbol{B}_{2,2}), \\ \boldsymbol{H}_2 &= \boldsymbol{A}_{2,2}(\boldsymbol{B}_{2,1} - \boldsymbol{B}_{1,1}), & \boldsymbol{H}_6 &= (\boldsymbol{A}_{1,2} - \boldsymbol{A}_{2,2})(\boldsymbol{B}_{2,1} + \boldsymbol{B}_{2,2}), \\ \boldsymbol{H}_3 &= (\boldsymbol{A}_{2,1} + \boldsymbol{A}_{2,2})\boldsymbol{B}_{1,1}, & \boldsymbol{H}_7 &= (\boldsymbol{A}_{1,1} - \boldsymbol{A}_{2,1})(\boldsymbol{B}_{1,1} + \boldsymbol{B}_{1,2}). \\ \boldsymbol{H}_4 &= \boldsymbol{a}_{2,2}(\boldsymbol{B}_{2,1} - \boldsymbol{B}_{1,1}), \end{aligned}$$

Strassen's algorithm (sequential)

The product of the two block matrices

$$\begin{pmatrix} \boldsymbol{C}_{1,1} & \boldsymbol{C}_{1,2} \\ \boldsymbol{C}_{2,1} & \boldsymbol{C}_{2,2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} \end{pmatrix} \begin{pmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{B}_{1,2} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{pmatrix}$$

can be computed by first calculating

$$\begin{aligned} \boldsymbol{H}_1 &= \boldsymbol{A}_{1,1}(\boldsymbol{B}_{1,2} - \boldsymbol{B}_{2,2}), & \boldsymbol{H}_5 &= (\boldsymbol{A}_{1,1} + \boldsymbol{A}_{2,2})(\boldsymbol{B}_{1,1} + \boldsymbol{B}_{2,2}), \\ \boldsymbol{H}_2 &= \boldsymbol{A}_{2,2}(\boldsymbol{B}_{2,1} - \boldsymbol{B}_{1,1}), & \boldsymbol{H}_6 &= (\boldsymbol{A}_{1,2} - \boldsymbol{A}_{2,2})(\boldsymbol{B}_{2,1} + \boldsymbol{B}_{2,2}), \\ \boldsymbol{H}_3 &= (\boldsymbol{A}_{2,1} + \boldsymbol{A}_{2,2})\boldsymbol{B}_{1,1}, & \boldsymbol{H}_7 &= (\boldsymbol{A}_{1,1} - \boldsymbol{A}_{2,1})(\boldsymbol{B}_{1,1} + \boldsymbol{B}_{1,2}). \\ \boldsymbol{H}_4 &= \boldsymbol{a}_{2,2}(\boldsymbol{B}_{2,1} - \boldsymbol{B}_{1,1}), \end{aligned}$$

Then

$$\begin{aligned} \boldsymbol{C}_{1,1} &= \boldsymbol{H}_4 + \boldsymbol{H}_5 + \boldsymbol{H}_6 - \boldsymbol{H}_2, & \boldsymbol{C}_{2,1} &= \boldsymbol{H}_{2,1} + \boldsymbol{H}_{2,2}, \\ \boldsymbol{C}_{1,2} &= \boldsymbol{H}_{1,1} + \boldsymbol{H}_{1,2}, & \boldsymbol{C}_{2,2} &= \boldsymbol{H}_1 + \boldsymbol{H}_1 - \boldsymbol{H}_3 - \boldsymbol{H}_7. \end{aligned}$$

Strassen's algorithm

Strassen's algorithm may be applied recursively to multiply together any two matrices.

Theorem (Complexity of Strassen's algorithm)

If $\boldsymbol{A}$ and $\boldsymbol{B}$ are $n \times n$ matrices, then the (sequential) complexity of Strassen's algorithm is $O(n^{2.71})$.

Strassen's algorithm

Strassen's algorithm may be applied recursively to multiply together any two matrices.

Theorem (Complexity of Strassen's algorithm)

If $\boldsymbol{A}$ and $\boldsymbol{B}$ are $n \times n$ matrices, then the (sequential) complexity of Strassen's algorithm is $O(n^{2.71})$.

More elaborate algorithms exist that have complexity $O(n^{2.376})$ (Coppersmith-Winograd).

Complexity of matrix algorithms

Theorem (Equivalence of matrix operations)

Matrix multiplication, computation of determinants, matrix inversion and solution of a linear system of equations all have the same computational complexity (sequential and parallel).

Solving linear systems of equations

The standard Gaussian elimination algorithm is a bit cumbersome to parallelise, we therefore consider a classical iterative algorithm instead.

Solving linear systems of equations

Suppose we have the linear system of equations

$$
\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1, \\
a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2, \\
&\vdots \\
a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n.
\end{aligned}
$$

Solving linear systems of equations

Suppose we have the linear system of equations

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1, \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2, \\ &\vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n. \end{aligned}$$

If $a_{i,i} \neq 0$ for $i = 1, \ldots, n$, then we can solve equation i for x_i,

$$x_1 = (b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \cdots - a_{1,n}x_n)/a_{1,1},$$

Solving linear systems of equations

Suppose we have the linear system of equations

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1, \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2, \\ &\vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n. \end{aligned}$$

If $a_{i,i} \neq 0$ for $i = 1, \ldots, n$, then we can solve equation i for x_i,

$$\begin{aligned} x_1 &= (b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \cdots - a_{1,n}x_n)/a_{1,1}, \\ x_2 &= (b_2 - a_{2,1}x_1 - a_{2,3}x_3 - \cdots - a_{2,n}x_n)/a_{2,2}, \\ &\vdots \end{aligned}$$

Solving linear systems of equations

Suppose we have the linear system of equations

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1, \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2, \\ &\vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n. \end{aligned}$$

If $a_{i,i} \neq 0$ for $i = 1, \ldots, n$, then we can solve equation i for x_i,

$$\begin{aligned} x_1 &= (b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \cdots - a_{1,n}x_n)/a_{1,1}, \\ x_2 &= (b_2 - a_{2,1}x_1 - a_{2,3}x_3 - \cdots - a_{2,n}x_n)/a_{2,2}, \\ &\vdots \\ x_n &= (b_n - a_{n,1}x_1 - a_{n,2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{n,n}. \end{aligned}$$

Solving linear systems of equations

$$
\begin{aligned}
x_1 &= (b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \cdots - a_{1,n}x_n)/a_{1,1} \\
x_2 &= (b_2 - a_{2,1}x_1 - a_{2,3}x_3 - \cdots - a_{2,n}x_n)/a_{2,2} \\
&\vdots \\
x_n &= (b_n - a_{n,1}x_1 - a_{n,2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{n,n}
\end{aligned}
$$

If we choose an initial estimate $\boldsymbol{x}^0$ for the solution, we can use these equations to compute a new estimate $\boldsymbol{x}^1$, then $\boldsymbol{x}^2$ etc. This is called Jacobi's method.

Solving linear systems of equations

$$\begin{aligned} x_1 &= (b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \cdots - a_{1,n}x_n)/a_{1,1} \\ x_2 &= (b_2 - a_{2,1}x_1 - a_{2,3}x_3 - \cdots - a_{2,n}x_n)/a_{2,2} \\ &\vdots \\ x_n &= (b_n - a_{n,1}x_1 - a_{n,2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{n,n} \end{aligned}$$

If we choose an initial estimate $\boldsymbol{x}^0$ for the solution, we can use these equations to compute a new estimate $\boldsymbol{x}^1$, then $\boldsymbol{x}^2$ etc. This is called Jacobi's method.

Under suitable conditions on the coefficient matrix these iterations will converge to the true solution.

Solving linear systems of equations

$$
\begin{aligned}
x_1 &= (b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \cdots - a_{1,n}x_n)/a_{1,1} \\
x_2 &= (b_2 - a_{2,1}x_1 - a_{2,3}x_3 - \cdots - a_{2,n}x_n)/a_{2,2} \\
&\vdots \\
x_n &= (b_n - a_{n,1}x_1 - a_{n,2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{n,n}
\end{aligned}
$$

If we choose an initial estimate $\boldsymbol{x}^0$ for the solution, we can use these equations to compute a new estimate $\boldsymbol{x}^1$, then $\boldsymbol{x}^2$ etc. This is called Jacobi's method.

Under suitable conditions on the coefficient matrix these iterations will converge to the true solution.

Note that if the calculations are performed sequentially, we may make use of the new values of x_1, x_2, ..., x_{i-1} when we compute x_i; this is called Gaus-Seidel iteration and converges faster than Jacobi iteration.

Jacobi's method in parallel

Jacobi's method is perfect for parallel implementation:

$$
\begin{aligned}
x_1 &= (b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \cdots - a_{1,n}x_n)/a_{1,1} \\
x_2 &= (b_2 - a_{2,1}x_1 - a_{2,3}x_3 - \cdots - a_{2,n}x_n)/a_{2,2} \\
&\vdots \\
x_n &= (b_n - a_{n,1}x_1 - a_{n,2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{n,n}
\end{aligned}
$$

Jacobi's method in parallel

Jacobi's method is perfect for parallel implementation:

$$
\begin{aligned}
x_1 &= (b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \cdots - a_{1,n}x_n)/a_{1,1} \\
x_2 &= (b_2 - a_{2,1}x_1 - a_{2,3}x_3 - \cdots - a_{2,n}x_n)/a_{2,2} \\
&\vdots \\
x_n &= (b_n - a_{n,1}x_1 - a_{n,2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{n,n}
\end{aligned}
$$

Each of n processors is given the task of computing one x_i.
Requires n time steps per iteration.

Hybrid Jacobi/Gaus-Seidel

Suppose the system has dimension 100 000 and we only have 1000 processors. Assume that an initial solution $\boldsymbol{x}^0$ is given.

Hybrid Jacobi/Gaus-Seidel

Suppose the system has dimension 100 000 and we only have 1000 processors. Assume that an initial solution $\boldsymbol{x}^0$ is given.

First iteration:

1. For $i = 1, 2, \ldots, 100$:

Hybrid Jacobi/Gaus-Seidel

Suppose the system has dimension 100 000 and we only have 1000 processors. Assume that an initial solution $\boldsymbol{x}^0$ is given.

First iteration:

1. For $i = 1, 2, \ldots, 100$:
   1. Set $j_1 = 1000i + 1$ and $j_2 = 1000(i + 1)$

Hybrid Jacobi/Gaus-Seidel

Suppose the system has dimension 100 000 and we only have 1000 processors. Assume that an initial solution $\boldsymbol{x}^0$ is given.

First iteration:

1. For $i = 1, 2, \ldots, 100$:
   1. Set $j_1 = 1000i + 1$ and $j_2 = 1000(i + 1)$
   2. Give equations $j_1, \ldots, j_2$ to the processors

Hybrid Jacobi/Gaus-Seidel

Suppose the system has dimension 100 000 and we only have 1000 processors. Assume that an initial solution $\boldsymbol{x}^0$ is given.

First iteration:

1. For $i = 1, 2, \ldots, 100$:
   1. Set $j_1 = 1000i + 1$ and $j_2 = 1000(i + 1)$
   2. Give equations $j_1, \ldots, j_2$ to the processors
   3. Compute $x_{j_1}^1, \ldots, x_{j_2}^1$ from $x_1^1, \ldots, x_{j_1-1}^1, x_{j_1}^0, \ldots, x_n^0$

Hybrid Jacobi/Gaus-Seidel

Suppose the system has dimension 100 000 and we only have 1000 processors. Assume that an initial solution $\boldsymbol{x}^0$ is given.

First iteration:

1. For $i = 1, 2, \ldots, 100$:
   1. Set $j_1 = 1000i + 1$ and $j_2 = 1000(i + 1)$
   2. Give equations $j_1, \ldots, j_2$ to the processors
   3. Compute $x_{j_1}^1, \ldots, x_{j_2}^1$ from $x_1^1, \ldots, x_{j_1-1}^1, x_{j_1}^0, \ldots, x_n^0$

Hybrid Jacobi/Gaus-Seidel

Suppose the system has dimension 100 000 and we only have 1000 processors. Assume that an initial solution $\boldsymbol{x}^0$ is given.

First iteration:

1. For $i = 1, 2, \ldots, 100$:
   1. Set $j_1 = 1000i + 1$ and $j_2 = 1000(i + 1)$
   2. Give equations $j_1, \ldots, j_2$ to the processors
   3. Compute $x_{j_1}^1, \ldots, x_{j_2}^1$ from $x_1^1, \ldots, x_{j_1-1}^1, x_{j_1}^0, \ldots, x_n^0$

Repeat until convergence.

Conclusion

The number of time steps in many (mathematical) operations can be reduced considerably if multiple operations can be performed simultaneously.

Conclusion

The number of time steps in many (mathematical) operations can be reduced considerably if multiple operations can be performed simultaneously.

Often other algorithms than the traditional sequential ones are the most efficient.

Conclusion

The number of time steps in many (mathematical) operations can be reduced considerably if multiple operations can be performed simultaneously.

Often other algorithms than the traditional sequential ones are the most efficient.

The actual choice of algorithm is usually highly dependent on the type of processor and how memory is organised.

Problem

Add 1000 integers, each with 5 digits.

Parallel computing in practice

Problem

Add 1000 integers, each with 5 digits.

Resources

This group of people.

Parallel computing in practice

Problem

Add 1000 integers, each with 5 digits.

Resources

This group of people.

Method

1. While more than one number:
    1. I share out the numbers evenly, at least two numbers each
    2. You add your numbers
    3. You pass your results back to me

Parallel computing in practice

Method (intelligent)

Assumption: You are arranged in a strict hierarchy

1. I share out the numbers evenly

Parallel computing in practice

Method (intelligent)

Assumption: You are arranged in a strict hierarchy

1. I share out the numbers evenly
2. While more than one active computer:

Method (intelligent)

Assumption: You are arranged in a strict hierarchy

1. I share out the numbers evenly
2. While more than one active computer:
    1. You add your numbers

Parallel computing in practice

Method (intelligent)

Assumption: You are arranged in a strict hierarchy

1. I share out the numbers evenly
2. While more than one active computer:
    1. You add your numbers
    2. You pass your result to someone more important than you

Parallel computing in practice

Method (intelligent)

Assumption: You are arranged in a strict hierarchy

1. I share out the numbers evenly
2. While more than one active computer:
    1. You add your numbers
    2. You pass your result to someone more important than you
3. I receive the result from my top assistant

Parallel computing in practice

Method (intelligent)

Assumption: You are arranged in a strict hierarchy

1. I share out the numbers evenly
2. While more than one active computer:
   1. You add your numbers
   2. You pass your result to someone more important than you
3. I receive the result from my top assistant

Parallel computing in practice

Method (intelligent)

Assumption: You are arranged in a strict hierarchy

1. I share out the numbers evenly
2. While more than one active computer:
    1. You add your numbers
    2. You pass your result to someone more important than you
3. I receive the result from my top assistant

The intelligence and communication skills of our processors are important, not just their computational skills.