

# Scalability of operations

Victor Eijkhout

Fall 2019

# Justification

Parallel operations are supposed to be faster than their sequential counterparts. In this section we will explore how to quantify this, and we will see examples where the same result can be computed with different efficiencies.

# Simple model of parallel computation

- $\alpha$ : message latency
- $\beta$ : time per word (inverse of bandwidth)
- $\gamma$ : time per floating point operation

Send  $n$  items and do  $m$  operations:

$$cost = \alpha + \beta \cdot n + \gamma \cdot m$$

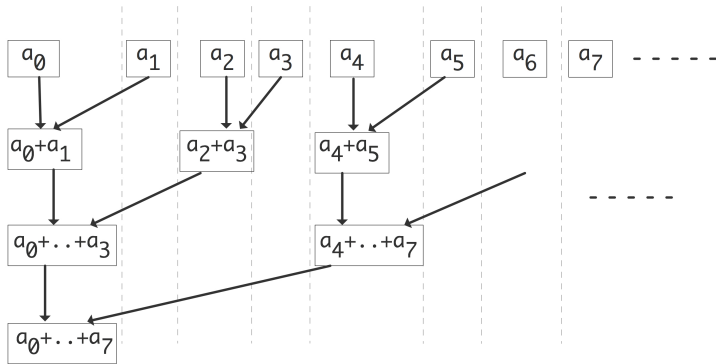
Pure sends: no  $\gamma$  term,

pure computation: no  $\alpha, \beta$  terms,

sometimes mixed: reduction

# Model for collectives

- One simultaneous send and receive:
- doubling of active processors
- collectives have a  $\alpha \log_2 p$  cost component



# Broadcast

	$t = 0$	$t = 1$	$t = 2$
$p_0$	$x_0 \downarrow, x_1 \downarrow, x_2 \downarrow, x_3 \downarrow$	$x_0 \downarrow, x_1 \downarrow, x_2 \downarrow, x_3 \downarrow$	$x_0, x_1, x_2, x_3$
$p_1$		$x_0 \downarrow, x_1 \downarrow, x_2 \downarrow, x_3 \downarrow$	$x_0, x_1, x_2, x_3$
$p_2$			$x_0, x_1, x_2, x_3$
$p_3$			$x_0, x_1, x_2, x_3$

On  $t = 0$ ,  $p_0$  sends to  $p_1$ ; on  $t = 1$   $p_0, p_1$  send to  $p_2, p_3$ .

Optimal complexity:

$$\lceil \log_2 p \rceil \alpha + n\beta.$$

Actual complexity:

$$\lceil \log_2 p \rceil (\alpha + n\beta).$$

Good enough for short vectors.

# Reduce

Optimal complexity:

$$\lceil \log_2 p \rceil \alpha + n\beta + \frac{p-1}{p} \gamma n.$$

Spanning tree algorithm:

	$t = 1$	$t = 2$	$t = 3$
$p_0$	$x_0^{(0)}, x_1^{(0)}, x_2^{(0)}, x_3^{(0)}$	$x_0^{(0:1)}, x_1^{(0:1)}, x_2^{(0:1)}, x_3^{(0:1)}$	$x_0^{(0:3)}, x_1^{(0:3)}, x_2^{(0:3)}, x_3^{(0:3)}$
$p_1$	$x_0^{(1)} \uparrow, x_1^{(1)} \uparrow, x_2^{(1)} \uparrow, x_3^{(1)} \uparrow$		
$p_2$	$x_0^{(2)}, x_1^{(2)}, x_2^{(2)}, x_3^{(2)}$	$x_0^{(2:3)} \uparrow, x_1^{(2:3)} \uparrow, x_2^{(2:3)} \uparrow, x_3^{(2:3)} \uparrow$	
$p_3$	$x_0^{(3)} \uparrow, x_1^{(3)} \uparrow, x_2^{(3)} \uparrow, x_3^{(3)} \uparrow$		

Running time

$$\lceil \log_2 p \rceil (\alpha + n\beta + \frac{p-1}{p} \gamma n).$$

Good enough for short vectors.

## Long vector broadcast

	$t = 0$	$t = 1$	$t = 2$	$t = 3$
$p_0$	$x_0 \downarrow, x_1, x_2, x_3$	$x_0, x_1 \downarrow, x_2, x_3$	$x_0, x_1, x_2 \downarrow, x_3$	$x_0, x_1, x_2, x_3 \downarrow$
$p_1$		$x_1$		
$p_2$			$x_2$	
$p_3$				$x_3$

takes  $p - 1$  messages of size  $N/p$ , for a total time of

$$T_{\text{scatter}}(N, P) = (p - 1)\alpha + (p - 1) \cdot \frac{N}{p} \cdot \beta.$$

## Long vector broadcast

After the scatter do a bucket-allgather:

	$t = 0$	$t = 1$	<i>etcetera</i>
$p_0$	$x_0 \downarrow$	$x_0$	$x_3 \downarrow$ $x_0, x_2, x_3$
$p_1$	$x_1 \downarrow$	$x_0 \downarrow, x_1$	$x_0, x_1, x_3$
$p_2$	$x_2 \downarrow$	$x_1 \downarrow, x_2$	$x_0, x_1, x_2$
$p_3$	$x_3 \downarrow$	$x_2 \downarrow, x_3$	$x_1, x_2, x_3$

Each partial message gets sent  $p - 1$  times, so this stage also has a complexity of

$$T_{\text{bucket}}(N, P) = (p - 1)\alpha + (p - 1) \cdot \frac{N}{p} \cdot \beta.$$

Better if  $N$  large.



# Allgather

Gather  $n$  elements: each processor owns  $n/p$ ;  
optimal running time

$$\lceil \log_2 p \rceil \alpha + \frac{p-1}{p} n \beta.$$

	$t = 1$	$t = 2$	$t = 3$
$p_0$	$x_0 \downarrow$	$x_0 x_1 \downarrow$	$x_0 x_1 x_2 x_3$
$p_1$	$x_1 \uparrow$	$x_0 x_1 \downarrow$	$x_0 x_1 x_2 x_3$
$p_2$	$x_2 \downarrow$	$x_2 x_3 \uparrow$	$x_0 x_1 x_2 x_3$
$p_3$	$x_3 \uparrow$	$x_2 x_3 \uparrow$	$x_0 x_1 x_2 x_3$

Same time as gather, half of gather-and-broadcast.

# Reduce-scatter

	$t = 1$	$t = 2$	$t = 3$
$p_0$	$x_0^{(0)}, x_1^{(0)}, x_2^{(0)} \downarrow, x_3^{(0)} \downarrow$	$x_0^{(0:2:2)}, x_1^{(0:2:2)} \downarrow$	$x_0^{(0:3)}$
$p_1$	$x_0^{(1)}, x_1^{(1)}, x_2^{(1)} \downarrow, x_3^{(1)} \downarrow$	$x_0^{(1:3:2)} \uparrow, x_1^{(1:3:2)}$	$x_1^{(0:3)}$
$p_2$	$x_0^{(2)} \uparrow, x_1^{(2)} \uparrow, x_2^{(2)}, x_3^{(2)}$	$x_2^{(0:2:2)}, x_3^{(0:2:2)} \downarrow$	$x_2^{(0:3)}$
$p_3$	$x_0^{(3)} \uparrow, x_1^{(3)} \uparrow, x_2^{(3)}, x_3^{(3)}$	$x_0^{(1:3:2)} \uparrow, x_1^{(1:3:2)}$	$x_3^{(0:3)}$

$$\lceil \log_2 p \rceil \alpha + \frac{p-1}{p} n(\beta + \gamma).$$

## Efficiency and scaling

# Speedup

- Single processor time  $T_1$ , on  $p$  processors  $T_p$
- speedup is  $S_p = T_1/T_p$ ,  $S_p \leq p$
- efficiency is  $E_p = S_p/p$ ,  $0 < E_p \leq 1$

Many caveats

- Is  $T_1$  based on the same algorithm? The parallel code?
- Sometimes superlinear speedup.
- Can the problem be run on a single processor?
- Can the problem be evenly divided?

# Limits on speedup/efficiency

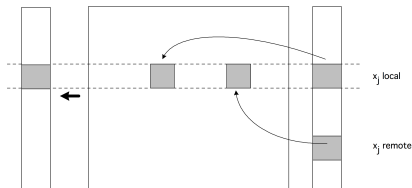
- $F_s$  sequential fraction,  $F_p$  parallelizable fraction
- $F_s + F_p = 1$
- $T_1 = (F_s + F_p)T_1 = F_s T_1 + F_p T_1$
- Amdahl's law:  $T_p = F_s T_1 + F_p T_1 / p$
- $P \rightarrow \infty$ :  $T_P \downarrow T_1 F_s$
- Speedup is limited by  $S_P \leq 1/F_s$ , efficiency is a decreasing function  $E \sim 1/P$ .
- loglog plot: straight line with slope  $-1$

# Scaling

- Amdahl's law: strong scaling  
same problem over increasing processors
- Often more realistic: weak scaling  
increase problem size with number of processors,  
for instance keeping memory constant
- Weak scaling:  $E_p > c$
- example (below): dense linear algebra

# Parallel matrix-vector product; general

- Assume a division by block rows
- Every processor  $p$  has a set of row indices  $I_p$

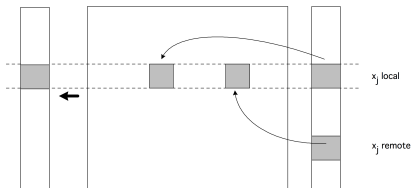


Mvp on processor  $p$ :

$$\forall i \in I_p: y_i = \sum_j a_{ij} x_j = \sum_q \sum_{j \in I_q} a_{ij} x_j$$

# Local and remote operations

Local and remote parts:



$$\forall i \in I_p : y_i = \sum_{j \in I_p} a_{ij} x_j + \sum_{q \neq p} \sum_{j \in I_q} a_{ij} x_j$$

Local part  $I_p$  can be executed right away,  $I_q$  requires communication.

Note possible overlap communication and computation;  
only used in the sparse case



# Dense MVP

- Separate communication and computation:
- first allgather
- then matrix-vector product

# Cost computation 1.

Algorithm:

Step	Cost (lower bound)
Allgather $x_i$ so that $x$ is available on all nodes	
Locally compute $y_i = A_i x$	$\approx 2 \frac{n^2}{p} \gamma$

# Allgather

Assume that data arrives over a binary tree:

- latency  $\alpha \log_2 P$
- transmission time, receiving  $n/P$  elements from  $P - 1$  processors

Algorithm with cost:

Step	Cost (lower bound)
Allgather $x_i$ so that $x$ is available on all nodes	$\lceil \log_2(P) \rceil \alpha + \frac{P-1}{P} n \beta \approx \log_2(P) \alpha + n \beta$
Locally compute $y_i = A_i x$	$\approx 2 \frac{n^2}{P} \gamma$

# Parallel efficiency

$$E_p^{1\text{D-row}}(n) = \frac{S_p^{1\text{D-row}}(n)}{p} = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{p}{2n} \frac{\beta}{\gamma}}.$$

Strong scaling, weak scaling?

# Optimistic scaling

Processors fixed, problem grows:

$$E_p^{1\text{D-row}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{p}{2n} \frac{\beta}{\gamma}}.$$

Roughly  $E_p \sim 1 - n^{-1}$

# Strong scaling

Problem fixed,  $p \rightarrow \infty$

$$E_p^{\text{1D-row}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{p}{2n} \frac{\beta}{\gamma}}.$$

# Strong scaling

Problem fixed,  $p \rightarrow \infty$

$$E_p^{\text{1D-row}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{p}{2n} \frac{\beta}{\gamma}}.$$

Roughly  $E_p \sim p^{-1}$



# Weak scaling

Memory fixed:

$$M = n^2/p$$

$$E_p^{1\text{D-row}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{p}{2n} \frac{\beta}{\gamma}} = \frac{1}{1 + \frac{\log_2(p)}{2M} \frac{\alpha}{\gamma} + \frac{\sqrt{p}}{2\sqrt{M}} \frac{\beta}{\gamma}}$$

# Weak scaling

Memory fixed:

$$M = n^2/p$$

$$E_p^{1\text{D-row}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{p}{2n} \frac{\beta}{\gamma}} = \frac{1}{1 + \frac{\log_2(p)}{2M} \frac{\alpha}{\gamma} + \frac{\sqrt{p}}{2\sqrt{M}} \frac{\beta}{\gamma}}$$

Does not scale:  $E_p \sim 1/\sqrt{p}$

problem in  $\beta$  term: too much communication

# Two-dimensional partitioning

$x_0$ $a_{00}$ $a_{01}$ $a_{02}$ $y_0$ $a_{10}$ $a_{11}$ $a_{12}$ $a_{20}$ $a_{21}$ $a_{22}$ $a_{30}$ $a_{31}$ $a_{32}$	$x_3$ $a_{03}$ $a_{04}$ $a_{05}$ $a_{13}$ $a_{14}$ $a_{15}$ $y_1$ $a_{23}$ $a_{24}$ $a_{25}$ $a_{33}$ $a_{34}$ $a_{35}$	$x_6$ $a_{06}$ $a_{07}$ $a_{08}$ $a_{16}$ $a_{17}$ $a_{18}$ $a_{26}$ $a_{27}$ $a_{28}$ $y_2$ $a_{37}$ $a_{37}$ $a_{38}$	$x_9$ $a_{09}$ $a_{0,10}$ $a_{0,11}$ $a_{19}$ $a_{1,10}$ $a_{1,11}$ $a_{29}$ $a_{2,10}$ $a_{2,11}$ $a_{39}$ $a_{3,10}$ $a_{3,11}$
$x_1$ $a_{40}$ $a_{41}$ $a_{42}$ $y_4$ $a_{50}$ $a_{51}$ $a_{52}$ $a_{60}$ $a_{61}$ $a_{62}$ $a_{70}$ $a_{71}$ $a_{72}$	$x_4$ $a_{43}$ $a_{44}$ $a_{45}$ $a_{53}$ $a_{54}$ $a_{55}$ $y_5$ $a_{63}$ $a_{64}$ $a_{65}$ $a_{73}$ $a_{74}$ $a_{75}$	$x_7$ $a_{46}$ $a_{47}$ $a_{48}$ $a_{56}$ $a_{57}$ $a_{58}$ $a_{66}$ $a_{67}$ $a_{68}$ $y_6$ $a_{77}$ $a_{77}$ $a_{78}$	$x_{10}$ $a_{49}$ $a_{4,10}$ $a_{4,11}$ $a_{59}$ $a_{5,10}$ $a_{5,11}$ $a_{69}$ $a_{6,10}$ $a_{6,11}$ $a_{79}$ $a_{7,10}$ $a_{7,11}$
$x_2$ $a_{80}$ $a_{81}$ $a_{82}$ $y_8$ $a_{90}$ $a_{91}$ $a_{92}$ $a_{10,0}$ $a_{10,1}$ $a_{10,2}$ $a_{11,0}$ $a_{11,1}$ $a_{11,2}$	$x_5$ $a_{83}$ $a_{84}$ $a_{85}$ $a_{93}$ $a_{94}$ $a_{95}$ $y_9$ $a_{10,3}$ $a_{10,4}$ $a_{10,5}$ $a_{11,3}$ $a_{11,4}$ $a_{11,5}$	$x_8$ $a_{86}$ $a_{87}$ $a_{88}$ $a_{96}$ $a_{97}$ $a_{98}$ $a_{10,6}$ $a_{10,7}$ $a_{10,8}$ $y_{10}$ $a_{11,7}$ $a_{11,7}$ $a_{11,8}$	$x_{11}$ $a_{89}$ $a_{8,10}$ $a_{8,11}$ $a_{99}$ $a_{9,10}$ $a_{9,11}$ $a_{10,9}$ $a_{10,10}$ $a_{10,11}$ $a_{11,9}$ $a_{11,10}$ $a_{11,11}$

# Two-dimensional partitioning

$x_0$ $a_{00}$ $a_{01}$ $a_{02}$ $y_0$ $a_{10}$ $a_{11}$ $a_{12}$ $a_{20}$ $a_{21}$ $a_{22}$ $a_{30}$ $a_{31}$ $a_{32}$	$x_3$   $y_1$	$x_6$   $y_2$	$x_9$   $y_3$
$x_1 \uparrow$   $y_4$	$x_4$   $y_5$	$x_7$   $y_6$	$x_{10}$   $y_7$
$x_2 \uparrow$   $y_8$	$x_5$   $y_9$	$x_8$   $y_{10}$	$x_{11}$   $y_{11}$

# Key to the algorithm

- Consider block  $(i, j)$
- it needs to multiply by the  $x$ s in column  $j$
- it produces part of the result of row  $i$

# Algorithm

- Collecting  $x_j$  on each processor  $p_{ij}$  by an *allgather* inside the processor columns.
- Each processor  $p_{ij}$  then computes  $y_{ij} = A_{ij}x_j$ .
- Gathering together the pieces  $y_{ij}$  in each processor row to form  $y_i$ , distribute this over the processor row: combine to form a *reduce-scatter*.
- Setup for the next  $A$  or  $A^t$  product

# Analysis 1.

Step	Cost (lower bound)
Allgather $x_i$ 's within columns	$\lceil \log_2(r) \rceil \alpha + \frac{r-1}{p} n \beta$ $\approx \log_2(r) \alpha + \frac{n}{c} \beta$
Perform local matrix-vector multiply	$\approx 2 \frac{n^2}{p} \gamma$
Reduce-scatter $y_i$ 's within rows	

# Reduce-scatter

Time:

$$\lceil \log_2 p \rceil \alpha + \frac{p-1}{p} n(\beta + \gamma).$$



Step	Cost (lower bound)
Allgather $x_i$ 's within columns	$\lceil \log_2(r) \rceil \alpha + \frac{r-1}{p} n \beta$ $\approx \log_2(r) \alpha + \frac{n}{c} \beta$
Perform local matrix-vector multiply	$\approx 2 \frac{n^2}{p} \gamma$
Reduce-scatter $y_i$ 's within rows	$\lceil \log_2(c) \rceil \alpha + \frac{c-1}{p} n \beta + \frac{c-1}{p} m \gamma$ $\approx \log_2(r) \alpha + \frac{n}{c} \beta + \frac{n}{c} \gamma$

# Efficiency

Let  $r = c = \sqrt{p}$ , then

$$E_p^{\sqrt{p} \times \sqrt{p}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{\sqrt{p}}{2n} \frac{(2\beta + \gamma)}{\gamma}}$$

# Strong scaling

Same story as before for  $p \rightarrow \infty$ :

$$E_p^{\sqrt{p} \times \sqrt{p}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{\sqrt{p}}{2n} \frac{(2\beta + \gamma)}{\gamma}} \sim p^{-1}$$

No strong scaling

# Weak scaling

Constant memory  $M = n^2/p$ :

$$E_p^{\sqrt{p} \times \sqrt{p}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{\sqrt{p}}{2n} \frac{(2\beta + \gamma)}{\gamma}}$$

# Weak scaling

Constant memory  $M = n^2/p$ :

$$E_p^{\sqrt{p} \times \sqrt{p}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{\sqrt{p}}{2n} \frac{(2\beta + \gamma)}{\gamma}} = \frac{1}{1 + \frac{\log_2(p)}{2M} \frac{\alpha}{\gamma} + \frac{1}{2\sqrt{M}} \frac{(2\beta + \gamma)}{\gamma}}$$

# Weak scaling

Constant memory  $M = n^2/p$ :

$$E_p^{\sqrt{p} \times \sqrt{p}}(n) = \frac{1}{1 + \frac{p \log_2(p)}{2n^2} \frac{\alpha}{\gamma} + \frac{\sqrt{p}}{2n} \frac{(2\beta + \gamma)}{\gamma}} = \frac{1}{1 + \frac{\log_2(p)}{2M} \frac{\alpha}{\gamma} + \frac{1}{2\sqrt{M}} \frac{(2\beta + \gamma)}{\gamma}}$$

Weak scaling:

for  $p \rightarrow \infty$  this is  $\approx 1/\log_2 P$ :

only slowly decreasing.

# LU factorizations

- Needs a cyclic distribution
- This is very hard to program, so:
- Scalapack, 1990s product, not extendible, impossible interface
- Elemental: 2010s product, extendible, nice user interface (and it is way faster)