Parallel Scientific Computing

Course AMS301 — Fall 2023 — Lecture 4

Performance analysis of parallel programs
Dense linear algebra

Axel Modave

Performance analysis of parallel programs

Dense linear algebra

Performance analysis of parallel programs

For a given sequential program,
what speedup can be expected
when it is parallelized?

Qualities for an efficient parallel program

A good parallel program is a program that . . .

- minimizes the runtime,
- *(generally)* takes advantage of the compute power of the parallel machine,
- *(generally)* minimizes the communications and the waiting time.

Several tools to analyze the parallel performance

- Runtime, CPU time, computation time, communication time . . .
- Strong scaling and weak scaling,
- Speedup S and efficiency E.

Performance analysis — Time [1/2]

For a given parallel program, the **runtime** T is the time that elapses from the moment that a parallel computation starts to the moment that the last processor finishes execution.

The runtime depends on the slowest processor:

$$T \approx \max_p \left[T_{\text{comput}}|_p + T_{\text{comm}}|_p + T_{\text{wait}}|_p \right]$$

with, for each processor $p = 1 \ldots P$,

- $T_{\text{comput}}|_p$ = time dedicated to computations
- $T_{\text{comm}}|_p$ = time dedicated to communications
- $T_{\text{wait}}|_p$ = waiting time

Reminder:

```
MPI_Barrier(MPI_COMM_WORLD);
double time1 = MPI_Wtime();

// Lots of operations for all the processus

MPI_Barrier(MPI_COMM_WORLD);
double time2 = MPI_Wtime();

if(myRank == 0) cout << "Duration: " << time2-time1 << endl;
```

Performance analysis — Time [2/2]

For a given message, the **communication time** T_{comm} is given by

$$\boxed{T_{\text{comm}} \approx T_{\text{lat}} + L\, T_{\text{word}}}$$

with

- $T_{\text{lat}} =$ **latency time** *(independent of the size of the message)*
- $T_{\text{word}} =$ time to transfer a word
- $L =$ number of words in the message

Strategies

- One communication with a long message is generally . . .
  better than several sequential communications with short messages
  but worse than several parallel communications with short messages
- Communication times may be hidden behind computation times
  by using non-blocking communications *(when it is possible)*

Performance analysis — Scalability analysis [1/2]

The **scaling** or **scalability** of a parallel program is the ability to preserve the same efficiency when a larger number of processors P is used.

For a **strong scaling** analysis, P increases for a problem with a given size.
With $P\times$ more processors, can I solve a given problem $P\times$ more rapidly?

For a **weak scaling** analysis, P increases linearly with the size of the a problem.
With $P\times$ more processors, can I solve a $P\times$ larger problem with the same runtime?

In French:

- *Scaling/Scalability analysis = Analyse de scalabilité ou de passage à l'échelle*
- *Strong scaling = Scalabilité forte*
- *Weak scaling = Scalabilité faible*

Performance analysis — Speedup and efficiency

For a given parallel program, the **speedup** S is a number that measures the decrease of the runtime when P processors are used instead of 1 processor.

$$\boxed{S = \frac{T_{\text{sequential}}}{T_{\text{parallel}}}} \quad \in [0, P]$$

For a given parallel program, the **efficiency** E is the ratio between the actual speedup (S_{actual}) and the ideal speedup (S_{ideal}).

$$\boxed{E = \frac{S_{\text{actual}}}{S_{\text{ideal}}}} \quad \in [0, 1]$$

What speedup can be expected?

Illustration for a problem with different sizes:

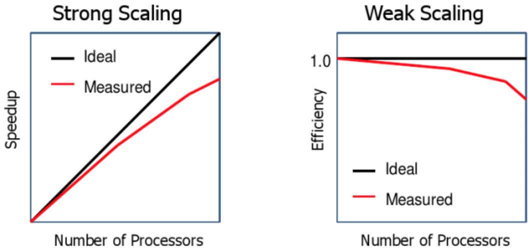
	P	1	2	4	8	16
Normal size	S	1.0	1.9	3.1	4.8	6.2
	E	1.0	0.95	0.78	0.60	0.32
Size $\times$ 2	S	1.0	1.9	3.6	6.5	10.8
	E	1.0	0.95	0.90	0.81	0.68
Size $\times$ 4	S	1.0	1.9	3.8	7.5	14.2
	E	1.0	0.95	0.95	0.94	0.89

The **scaling** or **scalability** of a parallel program is the ability to preserve the same efficiency when a larger number of processors P is used.

For a **strong scaling** analysis, P increases for a problem with a given size.
With $P\times$ more processors, can I solve a given problem $P\times$ more rapidly?

For a **weak scaling** analysis, P increases linearly with the size of the a problem.
With $P\times$ more processors, can I solve a $P\times$ larger problem with the same runtime?

Presentation of weak/strong scaling analyses



Warning: the size of the problem is constant in the first case,
and it increases linearly with the number of processors in the second case.

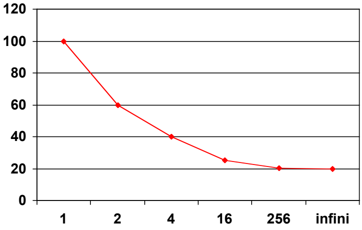
Performance analysis — Amdahl's law

Operations that must be performed sequentially prevent reaching the maximal speedup. A more reasonable goal is given by Amdahl's law.

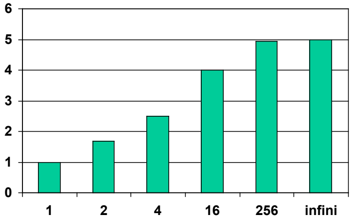
Amdahl's law

If β is the portion of the runtime of the sequential program corresponding to operations that cannot be parallelized,
then the maximum speedup that can be reached is $S_{\text{max}} = 1/\beta$.

Illustration for a problem with $\beta = 1/5$ and $T_{\textit{sequential}} = 100\,\text{sec}$:

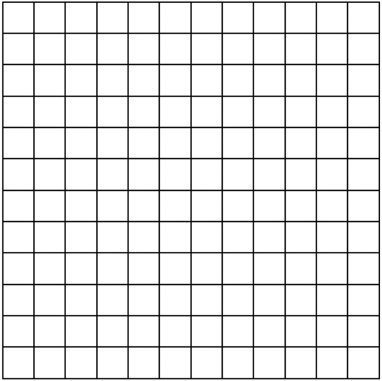


Parallel runtime T_{parallel} [sec]



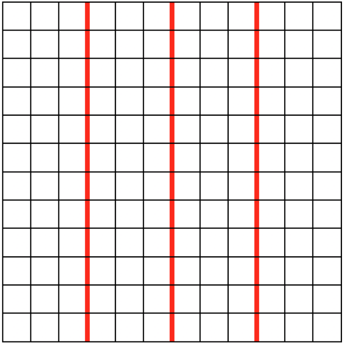
Speedup S

Here is a structured grid ...

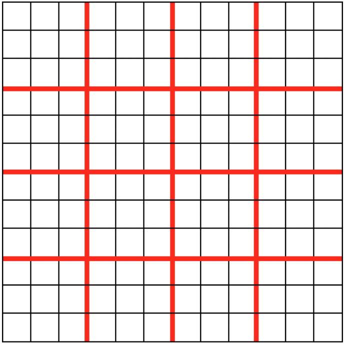


Partition for this grid?

Several partitions are possibles ...



1D partition



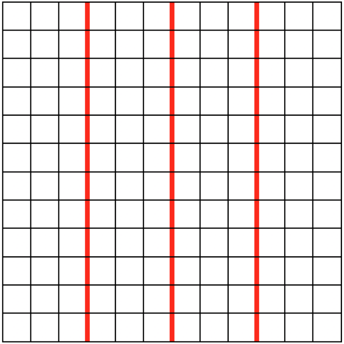
2D partition

The communication pattern and the size of the messages are different.

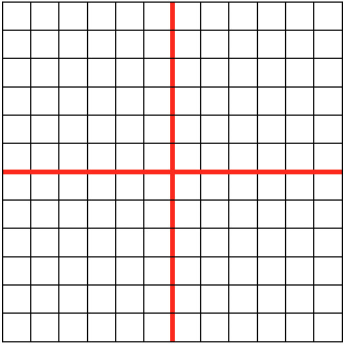
Intuitively, the grid has been partitioned by using the **method of coordinates**.

Performance analysis — Communications with a structured grid [3/4]

We consider P-partitions of a 2D grid of size $N \times N$.



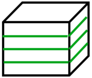
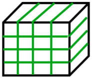
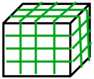
1D partition with $P = 4$



2D partition with $P = 4$

By subdomain ...	1D	2D	
Amount of transferred data	$\mathcal{O}(2N)$	$\mathcal{O}(4N/\sqrt{P})$	$\mathcal{O}(N)$
Number of operations	$\mathcal{O}(N^2/P)$	$\mathcal{O}(N^2/P)$	$\mathcal{O}(N^2)$
Ratio data / operations	$\mathcal{O}(2P/N)$	$\mathcal{O}(4\sqrt{P}/N)$	

We consider P-partitions of a 3D grid of size $N \times N \times N$.

By subdomain ...	 1D	 2D	 3D	
Amount of transferred data	$\mathcal{O}(2N^2)$	$\mathcal{O}(4N^2/P^{1/2})$	$\mathcal{O}(6N^2/P^{2/3})$	$\mathcal{O}(N^2)$
Number of operations	$\mathcal{O}(N^3/P)$	$\mathcal{O}(N^3/P)$	$\mathcal{O}(N^3/P)$	$\mathcal{O}(N^3)$
Ratio data / operations	$\mathcal{O}(2P/N)$	$\mathcal{O}(4P^{1/2}/N)$	$\mathcal{O}(6P^{1/3}/N)$	

Discussion

- The amount of transferred data is proportional to the **surface** of the interfaces. The number of operations is proportional to the **volume** of the subdomains.
- Increasing the number of subdomains decreases the number of operations by subdomain, but it increases the importance of the transferts. *(surface effect ↗)*
- For a large number of subdomains, it is interesting to use a partition with a high dimensionality. *(surface effect ↘)*

Performance analysis of parallel programs

Dense linear algebra

Linear algebra operations

Motivation

Linear algebra operations = Basic components for many algorithms
$\Longrightarrow$ *Efficient parallel algorithms are required for these operations.*

The algorithms depend on the structures of the matrices:

- Dense matrices without specific structures $\Longrightarrow$ BLAS
- Symmetric or hierarchical dense matrices *(e.g. FFT, boundary elements, ...)*
- Structured sparse matrix *(e.g. resulting from a finite difference discretization)*
- Unstructured sparse matrix *(e.g. resulting from a finite element discretization)*

BLAS $(Basic\ Linear\ Algebra\ Subroutines)$

The operations of dense linear algebra are categorized according to their complexity:

- $\mathcal{O}(N)$ operations: scalar product, addition of vectors — BLAS 1
- $\mathcal{O}(N^2)$ operations: matrix-vector product, addition of matrices — BLAS 2
- $\mathcal{O}(N^3)$ operations: matrix product — BLAS 3

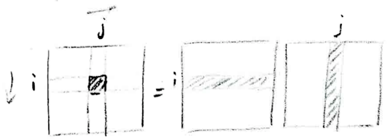
Optimized BLAS libraries are available for most environments (CPU/GPU).
They rely on partitions of the vectors/matrices into blocks.

Goal of this part: Parallel matrix product.

Parallel matrix product

We consider the matrix product: $\boxed{\mathbf{A} = \mathbf{B}\mathbf{C}}$ with $\mathbf{B}$ and $\mathbf{C} \in \mathbb{R}^{N \times N}$

Sequential algorithm
$\mathbf{A} \leftarrow \mathbf{0}$
for $i = 1, \ldots, N$ **do**
for $j = 1, \ldots, N$ **do**
for $k = 1, \ldots, N$ **do**
$A_{ij} \leftarrow A_{ij} + B_{ik}C_{kj}$
end
end
end



Analysis of the loops

- i-loop: iterations without dependence
- j-loop: iterations without dependence
- k-loop: iterations <u>with</u> dependence *($\longrightarrow$ accumulation of computed values)*

$\Longrightarrow$ *Parallelization of the i-loop and/or the j-loop. Several choices are possible!*

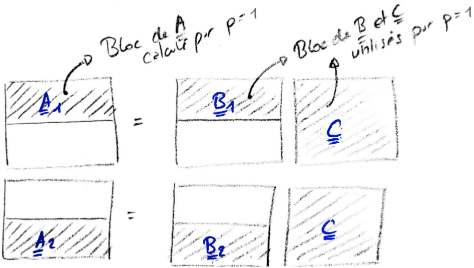
Parallel algorithm with 2 processes

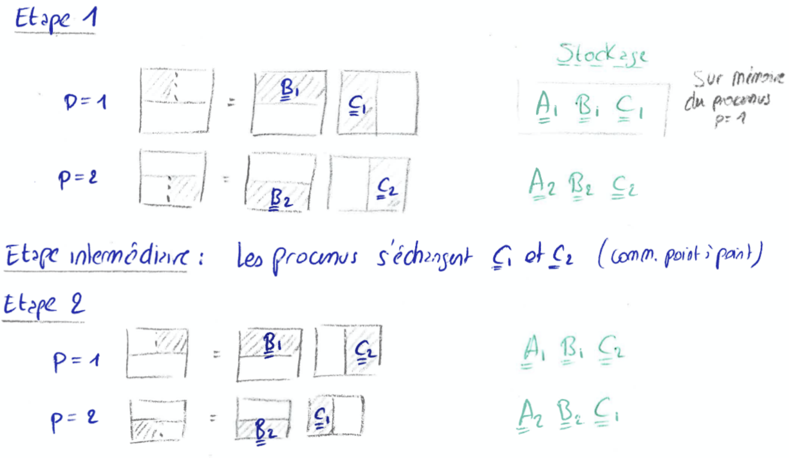
On each process $p = 1, 2$:

$\mathbf{A}_p \leftarrow \mathbf{0}$
$i_{\text{start},p} \leftarrow (p-1)N/2 + 1$
$i_{\text{end},p} \leftarrow (p-1)N/2$
for $i = i_{\text{start},p}, \ldots, i_{\text{end},p}$ **do**
  for $j = 1, \ldots, N$ **do**
    for $k = 1, \ldots, N$ **do**
      $A_{ij} \leftarrow A_{ij} + B_{ik}C_{kj}$
    end
  end
end

Discussion

- No communication between the processes
- Each process p compute $\mathbf{A}_p$, store $\mathbf{B}_p$ and $\mathbf{C}$ entirely





Discussion

- All the blocks of $\mathbf{C}$ are required by all the processes, but not at the same time. Initially, they are divided into the processes. Then, they are exchanged.
- At each step, each block is stored in only one in memory.

Perfect partition of data and computations!

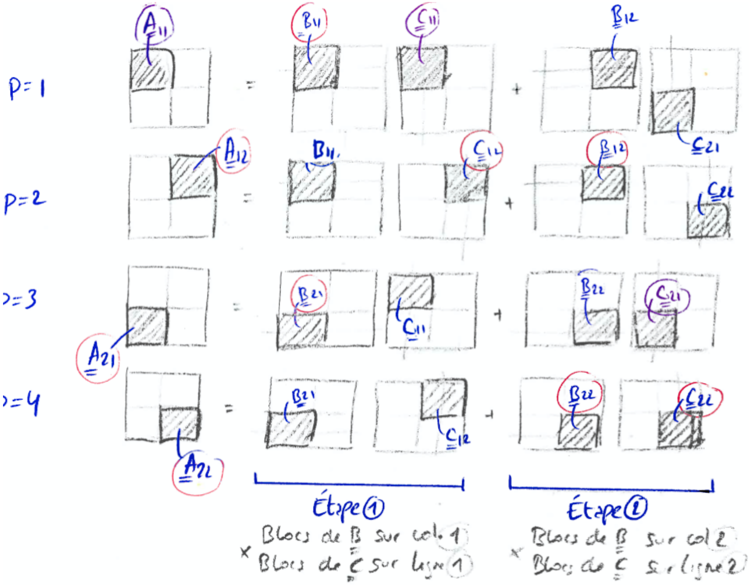
Partition in blocks *"lines"* *(straightforward extension of the case with 2 processes)*

Discussion

- Each process computes $N^2/4$ values of $\mathbf{A}$ ($N/4$ lines).
- At each step, the process p:
  - computes one block of $\mathbf{A}$ by using one block *"columns"* of $\mathbf{C}$.
  - sends this block *"columns"* of $\mathbf{C}$ to proc. $p-1$ *(or to the last proc.)*
  - receives a new block *"columns"* of $\mathbf{C}$ from proc. $p+1$ *(of to the first proc.)*
- We need 4 steps, with 4 communication phases at each step *(blocks of size $N^2/4$)*

Perfect division of computations and nearly-perfect division of data!

Partition in blocks *“squares”*



Partition in blocks *"squares"*

Init : Chaque processus stocke ses blocs :

$p=1$	B_{11} C_{11}
$p=2$	B_{12} C_{12}
$p=3$	B_{21} C_{21}
$p=4$	B_{22} C_{22}

Transferts :
- B_{11} envoyé de $p=1$ à $p=2$
- C_{11} envoyé de $p=1$ à $p=3$
- B_{21} envoyé de $p=3$ à $p=4$
- C_{12} envoyé de $p=2$ à $p=4$

} données nécessaires pour étape (1)

Etape (1)

Transferts :
- B_{12} envoyé de $p=2$ à $p=1$
- C_{21} envoyé de $p=3$ à $p=1$
- C_{22} envoyé de $p=4$ à $p=2$
- B_{22} envoyé de $p=4$ à $p=3$

Etape (2)

Discussion

- Each process computes $N^2/4$ valued of $\mathbf{A}$.
- Two steps are required, with 4 communications at each step *(blocks of size $N^2/4$)*
- *Perfect division of computations and nearly-perfect division of data!*
- *In total, $2\times$ less communication phases than the previous version!*

The matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are partitioned into $N_{\text{blk}} \times N_{\text{blk}}$ blocks:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} & \mathbf{B}_{13} \\ \mathbf{B}_{21} & \mathbf{B}_{22} & \mathbf{B}_{23} \\ \mathbf{B}_{31} & \mathbf{B}_{32} & \mathbf{B}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} & \mathbf{C}_{13} \\ \mathbf{C}_{21} & \mathbf{C}_{22} & \mathbf{C}_{23} \\ \mathbf{C}_{31} & \mathbf{C}_{32} & \mathbf{C}_{33} \end{bmatrix}$$

First parallel algorithm with $P = N_{\text{blk}}^2$ processes

Data: each process knows $\mathbf{B}$ and $\mathbf{C}$.

On each process (I, J) with $I, J = 1, \ldots, N_{\text{blk}}$:

$\mathbf{A}_{IJ} \leftarrow 0$

for $K = 1, \ldots, N_{\text{blk}}$ **do**

  $\mathbf{A}_{IJ} \leftarrow \mathbf{A}_{IJ} + \mathbf{B}_{IK}\mathbf{C}_{KJ}$

end

Result: each process (I, J) knows a block $\mathbf{A}_{IJ}$.

Discussion

- Matrices known by all the processes ...

Second parallel algorithm with $P = N_{\text{blk}}^2$ processes

Data: each process (I, J) knows $\mathbf{B}_{IJ}$ and $\mathbf{C}_{IJ}$.

On each process (I, J) with $I, J = 1, \dots, N_{\text{blk}}$:

$\mathbf{A}_{IJ} \leftarrow 0$

for $K = 1, \dots, N_{\text{blk}}$ **do**

- If $K \neq J$: Receive $\mathbf{B}_{IK}$ from process (I, K) and store in $\mathbf{B}_{\text{tmp}}$
- If $K \neq I$: Receive $\mathbf{C}_{KJ}$ from process (K, J) and store in $\mathbf{C}_{\text{tmp}}$
- If $K = J$: Send $\mathbf{B}_{IJ}$ to the other processes $(I, \bullet)$ and $\mathbf{B}_{\text{tmp}} = \mathbf{B}_{IJ}$
- If $K = I$: Send $\mathbf{C}_{IJ}$ to the other processes $(\bullet, J)$ and $\mathbf{C}_{\text{tmp}} = \mathbf{C}_{IJ}$
- $\mathbf{A}_{IJ} \leftarrow \mathbf{A}_{IJ} + \mathbf{B}_{\text{tmp}}\mathbf{C}_{\text{tmp}}$

end

Result: each process (I, J) knows a block $\mathbf{A}_{IJ}$.

Discussion

- Smaller memory requirement than for the previous approach.
- Need for N_{blk} steps in total, $2N_{\text{blk}}$ blocks are sent in total.
- Perfect division of computations and nearly-perfect division of data!

Summary

- **Performance analysis of parallel programs**
  - Runtime, communication time, latency
  - Strong/weak scaling
  - Speedup, efficiency, Amdahl's law
  - Surface/Volume effect
  - 1D/2D/3D partitions
- **Dense linear algebra**
  - BLAS 1, 2 and 3
  - Parallel algorithm for 2, 4 and N_{blk}^2 processes
  - Approach by block
  - *Easy and efficient parallelism!*