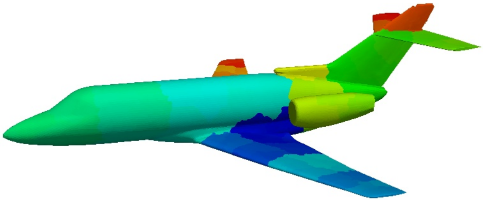


Asynchronous domain decomposition methods

Space domain decomposition - Schwarz methods

Frédéric Magoulès
Univ. Paris Saclay, CentraleSupélec (France)

Outline

01 History of Schwarz domain decomposition methods

Definition and motivation

Definition (Domain decomposition)

Domain decomposition (DD) is a "divide and conquer" technique for arriving at the solution of problem defined over a domain from the solution of related subproblems posed on subdomains.

- **Motivating assumption #1 :** the solution of the subproblems is qualitatively or quantitatively easier than the original
- **Motivating assumption #2 :** the original problem does not fit into the available memory space
- **Motivating assumption #3 (parallel context) :** the subproblems can be solved with some concurrency

Remarks on definition

- "Divide and conquer' is not a fully satisfactory description
  - "divide, conquer, and *combine*" is better
  - combination is often through iterative means
- True "divide-and-conquer" (only) algorithms are rare in computing (unfortunately)
- It might be preferable to focus on "subdomain composition" rather than "domain decomposition"

We often think we know all about "two" because two is "one and one". We forget that we have to make a study of "and."

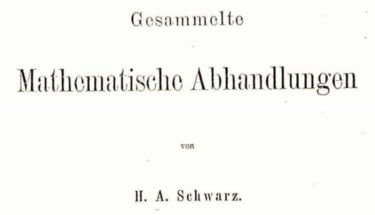
A.S. Eddington (1882-1944)

Remarks on definition

- Domain decomposition has generic and specific senses within the universe of parallel algorithms
  - generic sense : any data decomposition (considered in contrast to task decomposition)
  - specific sense : the domain is the domain of definition of an operator equation (differential, integral, algebraic)
- In a generic sense the process of constructing a parallel program consists of
  - Decomposition into tasks
  - Assignment of tasks to processes
  - Orchestration of processes
    - **Communication and synchronization**
  - Mapping of processes to processors

On the early history of domain decomposition

H.A. Schwarz (1870). Über einen Grenzübergang durch alternierendes Verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich, 15 :272-286, 1870.*

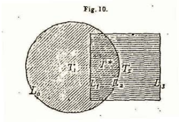

Gesammelte

Mathematische Abhandlungen

von

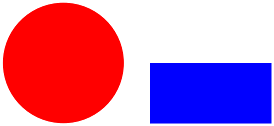
H. A. Schwarz.

“Die unter dem Namen Dirichletsches Princip bekannte Schlussweise, welche in gewissem Sinne als das Fundament des von Riemann entwickelten Zweiges der Theorie der analytischen Functionen angesehen werden muss, unterliegt, wie jetzt wohl allgemein zugestanden wird, hinsichtlich der Strenge sehr begründeten Einwendungen, deren vollst ?ndige Entfernung meines Wissens den Anstrengungen der Mathematiker bisher nicht gelungen ist.”

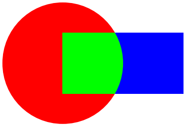


Motivation and explanation

- Convenient analytic means (separation of variables) are available for the regular problems in the subdomains,



but not for the irregular "keyhole" problem defined by their union



Motivation and explanation

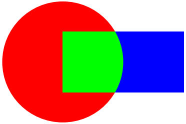
- Convenient analytic means (separation of variables) are available for the regular problems in the subdomains, but not for the irregular "keyhole" problem defined by their union
- Schwarz iteration defines a functional map from the values defined along (either) artificial interior boundary segment completing a subdomain (arc or segments) to an updated set of values

Motivation and explanation

- Convenient analytic means (separation of variables) are available for the regular problems in the subdomains, but not for the irregular "keyhole" problem defined by their union
- Schwarz iteration defines a functional map from the values defined along (either) artificial interior boundary segment completing a subdomain (arc or segments) to an updated set of values
- A contraction map is derived for the error
- Rate of convergence is not necessarily rapid - this was not a concern of Schwarz
- Subproblems are not solved concurrently - neither was this Schwarz' concern

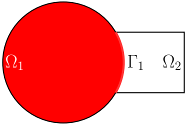
Classical alternating Schwarz method

Schwarz invents a method to proof that the infimum is attained : for a general domain $\Omega := \Omega_1 \cup \Omega_2$



Classical alternating Schwarz method

Schwarz invents a method to proof that the infimum is attained : for a general domain $\Omega := \Omega_1 \cup \Omega_2$

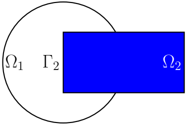


$$
\begin{aligned}
\Delta u_1^1 &= 0, && \text{in } \Omega_1 \\
u_1^1 &= g, && \text{on } \partial\Omega \cap \overline{\Omega}_1 \\
u_1^1 &= u_2^0, && \text{on } \Gamma_1
\end{aligned}
$$

solve on the disk
With arbitrary $u_2^0 = 0$

Classical alternating Schwarz method

Schwarz invents a method to proof that the infimum is attained : for a general domain $\Omega := \Omega_1 \cup \Omega_2$

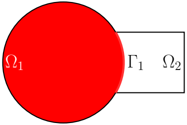


$$\begin{aligned} \Delta u_2^1 &= 0, && \text{in } \Omega_2 \\ u_2^1 &= g, && \text{on } \partial\Omega \cap \overline{\Omega}_2 \\ u_2^1 &= u_1^1, && \text{on } \Gamma_2 \end{aligned}$$

solve on the rectangle

Classical alternating Schwarz method

Schwarz invents a method to proof that the infimum is attained : for a general domain $\Omega := \Omega_1 \cup \Omega_2$

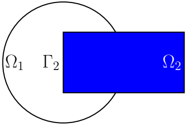


$$\begin{aligned} \Delta u_1^2 &= 0, & \text{in } \Omega_1 \\ u_1^2 &= g, & \text{on } \partial\Omega \cap \overline{\Omega}_1 \\ u_1^2 &= {\color{blue} u_2^1}, & \text{on } \Gamma_1 \end{aligned}$$

solve on the disk

Classical alternating Schwarz method

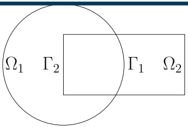
Schwarz invents a method to proof that the infimum is attained : for a general domain $\Omega := \Omega_1 \cup \Omega_2$



$$
\begin{aligned}
\Delta u_2^2 &= 0, && \text{in } \Omega_2 \\
u_2^2 &= g, && \text{on } \partial\Omega \cap \overline{\Omega}_2 \\
u_2^2 &= u_1^2, && \text{on } \Gamma_2
\end{aligned}
$$

solve on the rectangle

Classical alternating Schwarz method



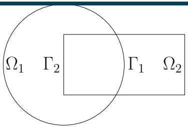
$$\begin{aligned} \Delta u_1^n &= 0, && \text{in } \Omega_1 \\ u_1^n &= g, && \text{on } \partial\Omega \cap \overline{\Omega}_1 \\ u_1^n &= u_2^{n-1}, && \text{on } \Gamma_1 \end{aligned}$$

solve on the disk

$$\begin{aligned} \Delta u_2^n &= 0, && \text{in } \Omega_2 \\ u_2^n &= g, && \text{on } \partial\Omega \cap \overline{\Omega}_2 \\ u_2^n &= u_1^n, && \text{on } \Gamma_2 \end{aligned}$$

solve on the rectangle

Classical alternating Schwarz method



$$\begin{aligned} \Delta u_1^n &= 0, && \text{in } \Omega_1 \\ u_1^n &= g, && \text{on } \partial\Omega \cap \overline{\Omega}_1 \\ u_1^n &= {\color{blue}u_2^{n-1}}, && \text{on } \Gamma_1 \end{aligned}$$

solve on the disk

$$\begin{aligned} \Delta u_2^n &= 0, && \text{in } \Omega_2 \\ u_2^n &= g, && \text{on } \partial\Omega \cap \overline{\Omega}_2 \\ u_2^n &= {\color{red}u_1^n}, && \text{on } \Gamma_2 \end{aligned}$$

solve on the rectangle

Theorem (H.A. Schwarz, 1869)

The iterative algorithm converges and the convergence rate is linked with the size of the overlap.

On the early history of parallel Schwarz

P.-L. Lions (1988) On the Schwarz alternating method I. *in First International Symposium on Domain Decomposition Methods for Partial Differential Equations (Paris, 1987), SIAM, Philadelphia, PA, pp.1-42, 1988.*

On the Schwarz Alternating Method. I

P. L. LIONS*

Introduction.

In [1], H.A. Schwarz proposed an iterative method for the solution of classical boundary value problems for harmonic functions : it consists in solving successively a similar problem in subdomains, going alternatively from one to the other as we recall more precisely below. The convergence of this process was proven by the use of the maximum principle. Since then, this method was studied by various authors including S.L. Sobolev [2], S.G. Michlin [3], M. Prager [4], D. Morgenstern [5], I. Babuska [6], R. Courant and D. Hilbert [7], F.E. Browder [8]... In some of these references the variational interpretation of the method as convenient successive projections was emphasized.

More recently, the interest in such iterative methods was renewed because of the applications to the numerical analysis of boundary value problems. This method was then considered as a method to decompose the original

*Ceremade, University Paris-Dauphine, Place de Lattre de Tassigny, 75775 Paris Cedex 16, France.

1

"The final extension we wish to consider concerns "parallel" versions of the Schwarz alternating method . . . / . . . u_j^{n+1} *is solution of* $-\Delta u_j^{n+1} = f$ *in* Ω_i *and* $u_j^{n+1} = u_j^n$ *on* $\partial\Omega_i \cap \Omega_j$*."*

Alternating and parallel Schwarz method

For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= u_2^{n-1}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= u_1^n, && \text{on } x = 0 \end{aligned}$$

Alternating and parallel Schwarz method

For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= u_2^{n-1}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= u_1^n, && \text{on } x = 0 \end{aligned}$$

- **Parallel Schwarz method (P-L. Lions 1988) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= u_2^{n-1}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= u_1^{n-1}, && \text{on } x = 0 \end{aligned}$$

Alternating and parallel Schwarz method

For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= u_2^{n-1}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= u_1^n, && \text{on } x = 0 \end{aligned}$$

- **Parallel Schwarz method (P-L. Lions 1988) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= u_2^{n-1}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= u_1^{n-1}, && \text{on } x = 0 \end{aligned}$$

Remark

Can be solved with two processors in parallel, one processor computes for Ω_1 and one processor computes for Ω_2 !

Illustration on an academic model

For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

$\mathcal{L}u = \partial_{xx} u$
$f = 0$
$\Omega = (0, 1)$, $\Omega_1 = (0, \frac{1}{2} + \frac{L}{2})$, $\Omega_2 = (\frac{1}{2} - \frac{L}{2}, 1)$.

Alternating Schwarz method

For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= u_2^{n-1}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= u_1^n, && \text{on } x = 0 \end{aligned}$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :

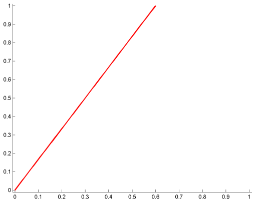
Alternating Schwarz method

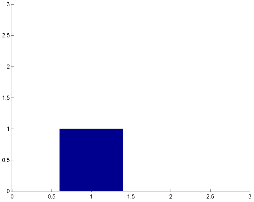
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$
\begin{aligned}
\mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \qquad & \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\
u_1^n &= {\color{blue}u_2^{n-1}}, && \text{on } x = L \qquad & u_2^n &= {\color{red}u_1^n}, && \text{on } x = 0
\end{aligned}
$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





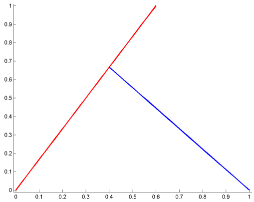
Alternating Schwarz method

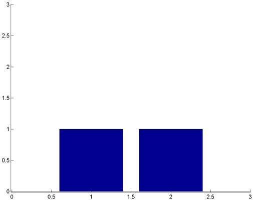
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$
\begin{aligned}
\mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \qquad & \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\
u_1^n &= {\color{blue}u_2^{n-1}}, && \text{on } x = L \qquad & u_2^n &= {\color{red}u_1^n}, && \text{on } x = 0
\end{aligned}
$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





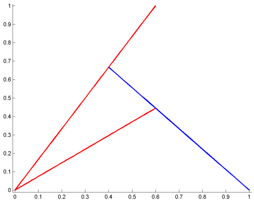
Alternating Schwarz method

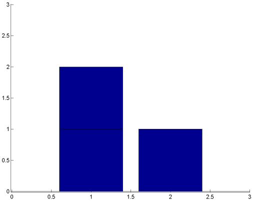
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= u_2^{n-1}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= u_1^n, && \text{on } x = 0 \end{aligned}$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





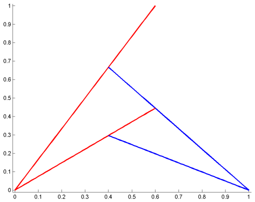
Alternating Schwarz method

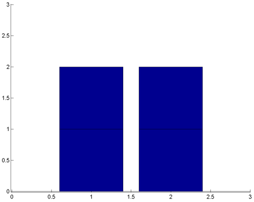
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= {\color{blue}u_2^{n-1}}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= {\color{red}u_1^n}, && \text{on } x = 0 \end{aligned}$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





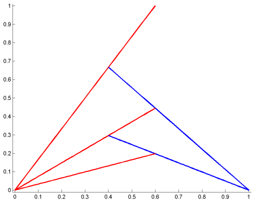
Alternating Schwarz method

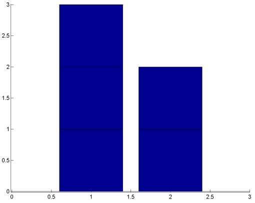
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= u_2^{n-1}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= u_1^n, && \text{on } x = 0 \end{aligned}$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





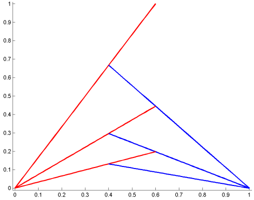
Alternating Schwarz method

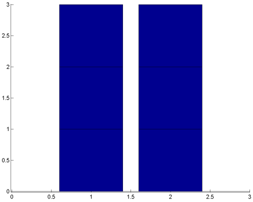
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Alternating Schwarz method (H.A. Schwarz 1869) :**

$$
\begin{aligned}
\mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \qquad & \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\
u_1^n &= {\color{blue}u_2^{n-1}}, && \text{on } x = L \qquad & u_2^n &= {\color{red}u_1^n}, && \text{on } x = 0
\end{aligned}
$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





Parallel Schwarz method

For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Parallel Schwarz method (P-L. Lions 1988) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, \quad \text{in } \Omega_1 \\ u_1^n &= u_2^{n-1}, \quad \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, \quad \text{in } \Omega_2 \\ u_2^n &= u_1^{n-1}, \quad \text{on } x = 0 \end{aligned}$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :

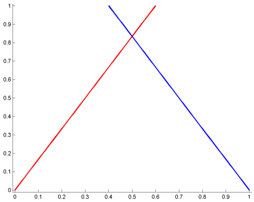
Parallel Schwarz method

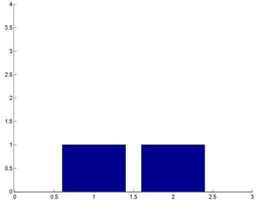
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Parallel Schwarz method (P-L. Lions 1988) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= {\color{blue}u_2^{n-1}}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= {\color{red}u_1^{n-1}}, && \text{on } x = 0 \end{aligned}$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





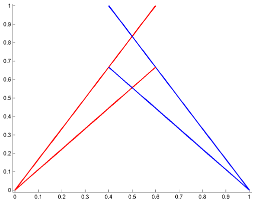
Parallel Schwarz method

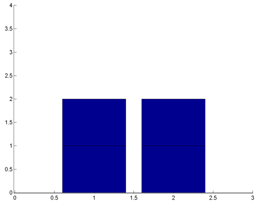
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Parallel Schwarz method (P-L. Lions 1988) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \qquad & \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_1^n &= {\color{blue}u_2^{n-1}}, && \text{on } x = L \qquad & u_2^n &= {\color{red}u_1^{n-1}}, && \text{on } x = 0 \end{aligned}$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





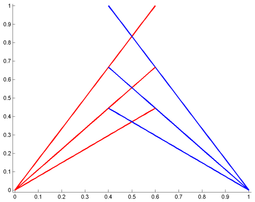
Parallel Schwarz method

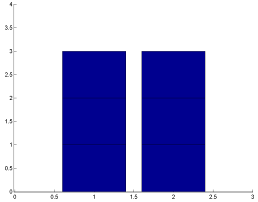
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Parallel Schwarz method (P-L. Lions 1988) :**

$$\begin{aligned} \mathcal{L}u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= {\color{blue}u_2^{n-1}}, && \text{on } x = L \end{aligned} \qquad \begin{aligned} \mathcal{L}u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= {\color{red}u_1^{n-1}}, && \text{on } x = 0 \end{aligned}$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





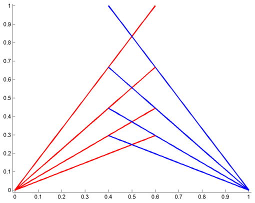
Parallel Schwarz method

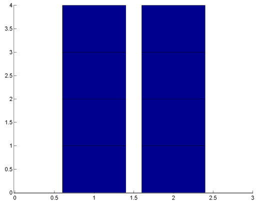
For $\mathcal{L}u = f$ in $\Omega = \mathbb{R}^2$, $\Omega_1 = (-\infty, L) \times \mathbb{R}$, $\Omega_2 = (0, \infty) \times \mathbb{R}$.

- **Parallel Schwarz method (P-L. Lions 1988) :**

$$\begin{array}{rclll} \mathcal{L}u_1^n & = & f, & \text{in } \Omega_1 \\ u_1^n & = & u_2^{n-1}, & \text{on } x = L \end{array} \qquad \begin{array}{rclll} \mathcal{L}u_2^n & = & f, & \text{in } \Omega_2 \\ u_2^n & = & u_1^{n-1}, & \text{on } x = 0 \end{array}$$

Screenshots of Schwarz solution (left) versus number of iterations (right) :





One possible improvement : other interface conditions

P.-L. Lions (1990) On the Schwarz alternating method III. *A variant for nonoverlapping subdomains, Partial Differential Equations (Houston, TX, 1989) SIAM, Philadelphia, PA, pp.202-223, 1990*

$$\begin{aligned}
-\Delta u_1^n &= f, && \text{in } \Omega_1 \\
u_1^n &= 0, && \text{on } \partial\Omega_1 \cap \partial\Omega \\
\left(\tfrac{\partial}{\partial n_1} + \alpha\right) u_1^n &= \left(-\tfrac{\partial}{\partial n_2} + \alpha\right) u_2^{n-1}, && \text{on } \partial\Omega_1 \cap \overline{\Omega}
\end{aligned}$$

with n_1 and n_2 the outward normal on the boundary of the subdomains

$$\begin{aligned}
-\Delta u_2^n &= f, && \text{in } \Omega_2 \\
u_2^n &= 0, && \text{on } \partial\Omega_2 \cap \partial\Omega \\
\left(\tfrac{\partial}{\partial n_2} + \alpha\right) u_2^n &= \left(-\tfrac{\partial}{\partial n_1} + \alpha\right) u_1^{n-1}, && \text{on } \partial\Omega_2 \cap \overline{\Omega}
\end{aligned}$$

with $\alpha \in \mathbb{R}$ and $\alpha > 0$.

One possible improvement : other interface conditions

P.-L. Lions (1990) On the Schwarz alternating method III. *A variant for nonoverlapping subdomains, Partial Differential Equations (Houston, TX, 1989) SIAM, Philadelphia, PA, pp.202-223, 1990*

$$\begin{aligned} -\Delta u_1^n &= f, && \text{in } \Omega_1 \\ u_1^n &= 0, && \text{on } \partial\Omega_1 \cap \partial\Omega \\ (\tfrac{\partial}{\partial n_1} + \alpha) u_1^n &= (-\tfrac{\partial}{\partial n_2} + \alpha) u_2^{n-1}, && \text{on } \partial\Omega_1 \cap \overline{\Omega} \end{aligned}$$

with n_1 and n_2 the outward normal on the boundary of the subdomains

$$\begin{aligned} -\Delta u_2^n &= f, && \text{in } \Omega_2 \\ u_2^n &= 0, && \text{on } \partial\Omega_2 \cap \partial\Omega \\ (\tfrac{\partial}{\partial n_2} + \alpha) u_2^n &= (-\tfrac{\partial}{\partial n_1} + \alpha) u_1^{n-1}, && \text{on } \partial\Omega_2 \cap \overline{\Omega} \end{aligned}$$

with $\alpha \in \mathbb{R}$ and $\alpha > 0$.

Theorem (P.L. Lions, 1990)

The iterative algorithm converges with **and without** *overlap.*

02 Why asynchronous Schwarz domain decomposition methods ?

Towards extreme-scale simulations
How does synchronous parallel Schwarz method work ?
How does asynchronous parallel Schwarz method work ?

Towards extreme-scale simulations

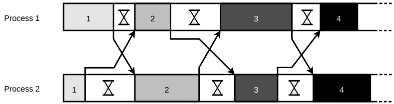
Domain decomposition are extremely efficient for solving PDEs in parallel, but data exchange synchronization between the processors become a problem when dealing with more than 10.000 processors.

- How to perform extremely large scale simulation ?
- How to use large number of processors/core (> 10.000) ?
- How to manage fault tolerance ?

Solution might be new **chaotic or asynchronous** parallel iterative domain decomposition methods

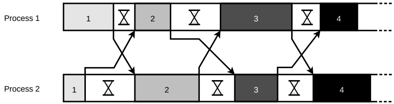
Iterative algorithms classification

- Synchronous Iteration and Synchronous Communication

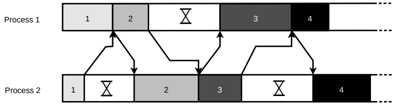


Iterative algorithms classification

- Synchronous Iteration and Synchronous Communication

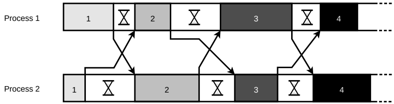


- Synchronous Iteration and Asynchronous Communication

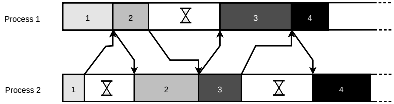


Iterative algorithms classification

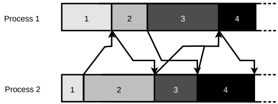
- Synchronous Iteration and Synchronous Communication



- Synchronous Iteration and Asynchronous Communication



- Asynchronous Iteration and Asynchronous Communication



Introduction to asynchronous iterative algorithms

- Principles
  - When a process has finished one iteration, it start a new one immediately
  - It uses the latest available data
  - It sends its data asynchronously
- Remark
  - When new data arrives, the previous one is discarded (even if it has never been read)
  - The sending of data may be skipped if the previous send is not finished

Introduction to asynchronous iterative algorithms

- Advantages
  - No time lost for synchronization
  - Work with unreliable communication, i.e., fault tolerance
  - Not limited by the slowest node, i.e., heterogeneous cluster/grid
  - Take advantage of fast connection when available without been limited by the slowest connection
  - Also interesting for very large super computer ...
- Disadvantages
  - Much more complex mathematical convergence conditions
  - More complicated to program, i.e., need of a new communication library

Short bibliography - Async. iterations theory

- Rosenfeld (1969) : A case study in programming for parallel-processors
- Chazan, Miranker (1969) : Chaotic relaxation
- Miellou (1975) : Algorithmes de relaxation chaotique à retards
- Baudet (1978) : Async. iterative methods for multiprocessors
- El Tarazi (1982) : Some convergence results for async. algorithms
- Bertsekas, Tsitsiklis (1989) : Parallel and distributed computation : Numerical methods *(book)*
- Üresin, Dubois (1990) : Parallel async. algorithms for discrete data
- Frommer, Szyld (1994) : Async. two-stage iterative methods
- El Baz, Spiteri, Miellou, Gazen (1996) : Async. iterative algorithms with flexible communication for nonlinear problems
- Frommer, Szyld (1998) : Async. iterations with flexible communication for linear systems
- Strikwerda (2002) : A probabilistic analysis of async. iteration

Short bibliography - Async. domain decomposition

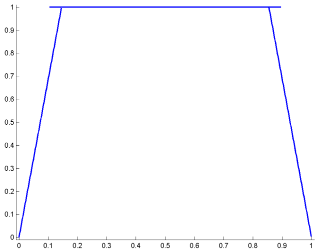
- Miellou (1982) : Variantes synchrones et asynchrones de la méthode alternée de Schwarz (Report, Univ. de Besancon)
- Hart, McCormick (1989) : Async. multilevel adaptive methods for solving partial differential equations on multiprocessors : Basic ideas
- Evans, Deren (1991) : An async. parallel algorithm for solving a class of nonlinear simultaneous equations $\rightarrow$ Async. Schwarz alternating method
- Bru, Migallón, Penadés, Szyld (1995) : Parallel, synchronous and async. two-stage multisplitting methods
- Spitéri, Miellou, El Baz (1995) : Async. Schwarz alternating method for the solution of nonlinear partial differential equations
- Bahi, Miellou, Rhofir (1997) : Async. multisplitting methods for nonlinear fixed point problems
- Frommer, Schwandt, Szyld (1997) : Async. weighted additive Schwarz methods
- Magoulès, Szyld, Venet (2017) : Async. optimized Schwarz methods with and without overlap
- Magoulès, Venet (2018) : Async. iterative sub-structuring methods
- Wolfson-Pou, Chow (2019) : Async. multigrid methods
- Glusa, Boman, Chow, Rajamanickam, Szyld (2020) : Scalable async.

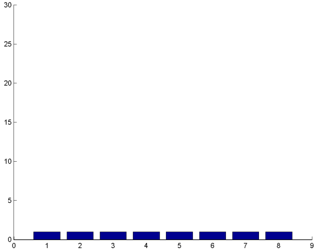
"Possibly the kind of methods which will allow the next generation of parallel machines to attain the expected potential."

Frommer and Szyld, 2000

Synchronous parallel Schwarz method

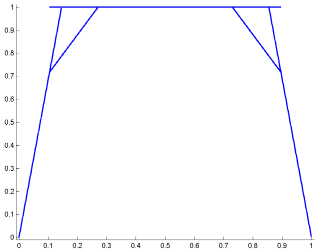
Screenshots of Schwarz solution (left) versus number of iterations (right) :

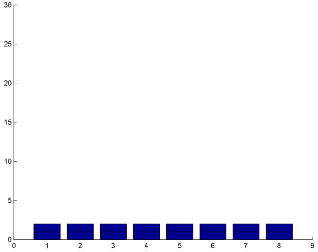




Synchronous parallel Schwarz method

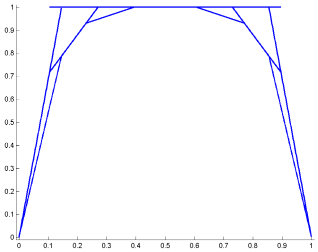
Screenshots of Schwarz solution (left) versus number of iterations (right) :

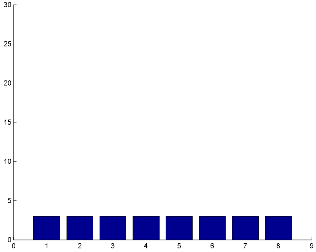




Synchronous parallel Schwarz method

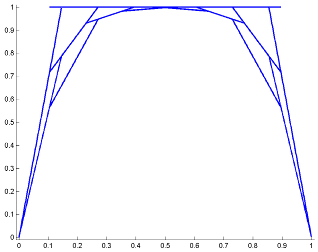
Screenshots of Schwarz solution (left) versus number of iterations (right) :

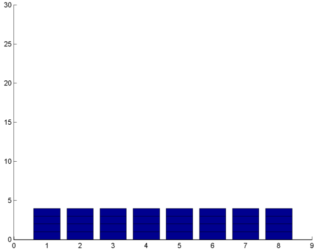




Synchronous parallel Schwarz method

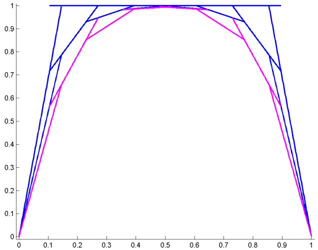
Screenshots of Schwarz solution (left) versus number of iterations (right) :

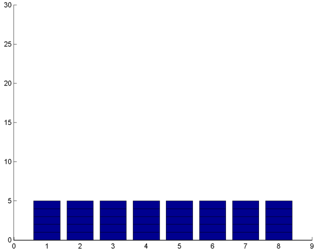




Synchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

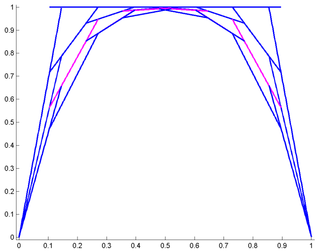


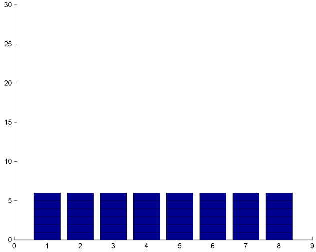


When processor 3 meets unexpected delay during iteration number 5, all other processors are waiting for it, and . . .

Synchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

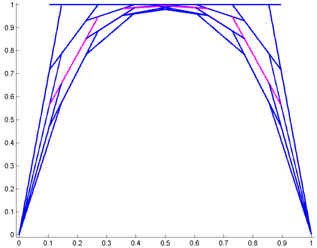


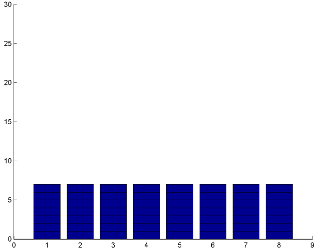


...when processor 3 has finished its iteration, all other processors start the next iteration.

Synchronous parallel Schwarz method

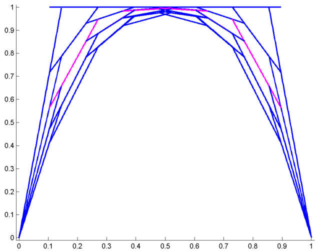
Screenshots of Schwarz solution (left) versus number of iterations (right) :

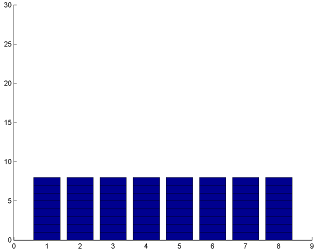




Synchronous parallel Schwarz method

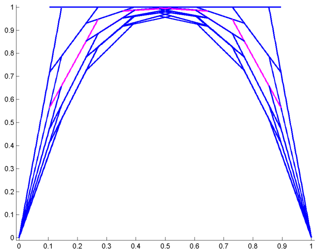
Screenshots of Schwarz solution (left) versus number of iterations (right) :

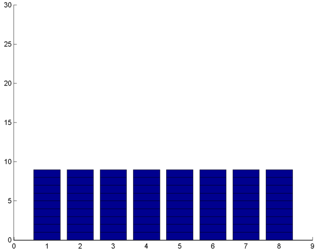




Synchronous parallel Schwarz method

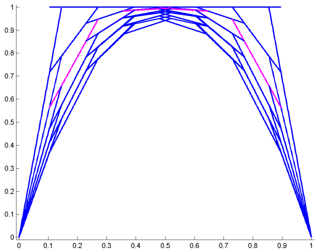
Screenshots of Schwarz solution (left) versus number of iterations (right) :

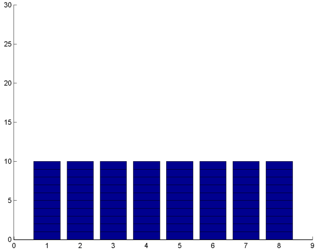




Synchronous parallel Schwarz method

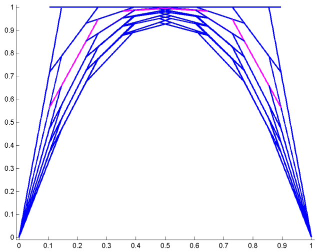
Screenshots of Schwarz solution (left) versus number of iterations (right) :

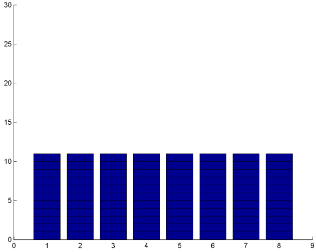




Synchronous parallel Schwarz method

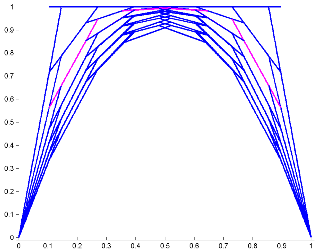
Screenshots of Schwarz solution (left) versus number of iterations (right) :

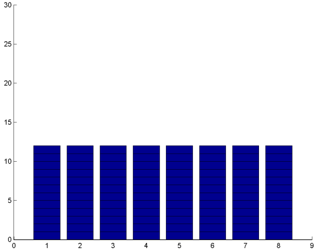




Synchronous parallel Schwarz method

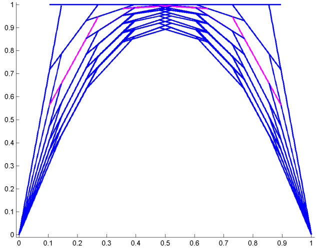
Screenshots of Schwarz solution (left) versus number of iterations (right) :

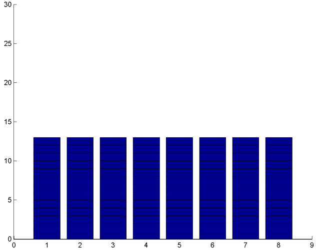




Synchronous parallel Schwarz method

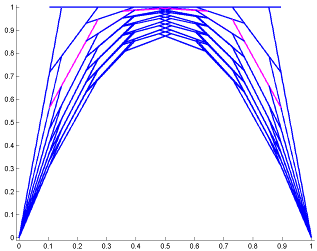
Screenshots of Schwarz solution (left) versus number of iterations (right) :

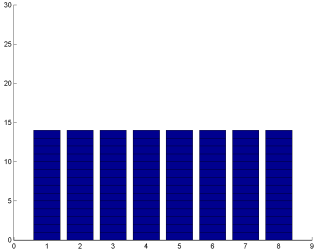




Synchronous parallel Schwarz method

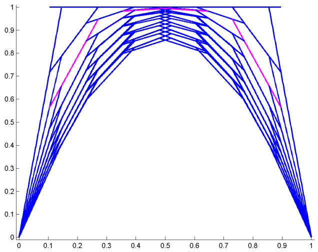
Screenshots of Schwarz solution (left) versus number of iterations (right) :

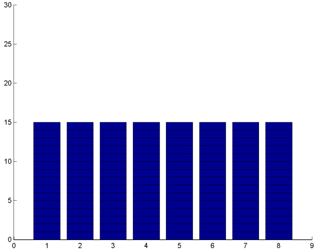




Synchronous parallel Schwarz method

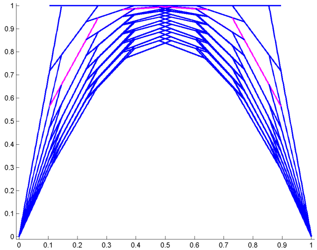
Screenshots of Schwarz solution (left) versus number of iterations (right) :

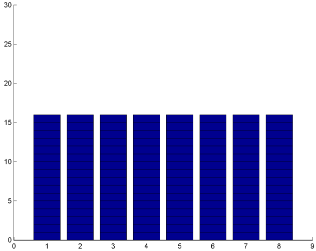




Synchronous parallel Schwarz method

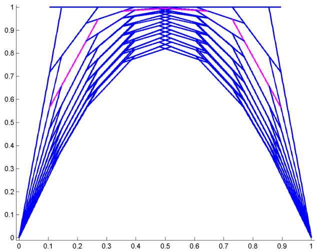
Screenshots of Schwarz solution (left) versus number of iterations (right) :

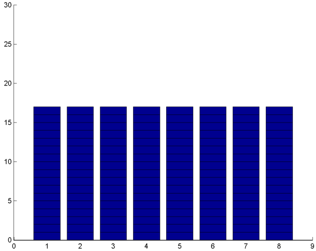




Synchronous parallel Schwarz method

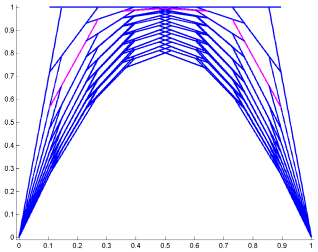
Screenshots of Schwarz solution (left) versus number of iterations (right) :

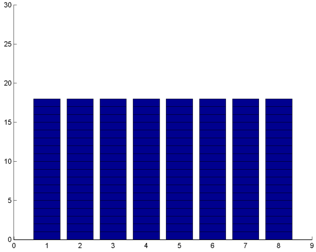




Synchronous parallel Schwarz method

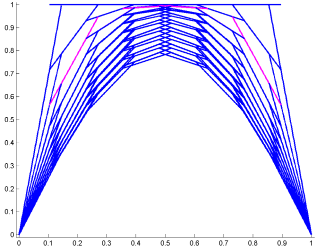
Screenshots of Schwarz solution (left) versus number of iterations (right) :

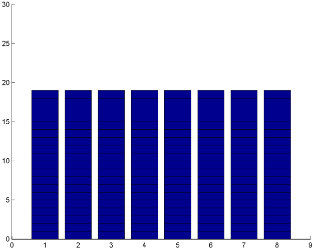




Synchronous parallel Schwarz method

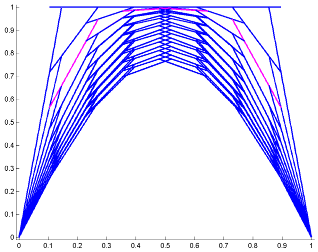
Screenshots of Schwarz solution (left) versus number of iterations (right) :

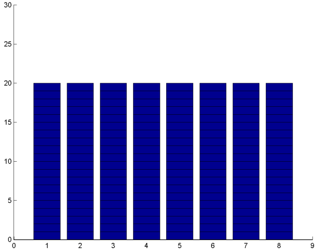




Synchronous parallel Schwarz method

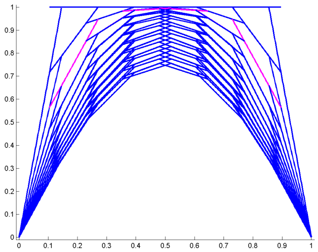
Screenshots of Schwarz solution (left) versus number of iterations (right) :

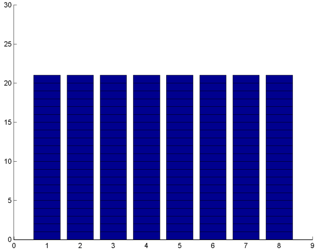




Synchronous parallel Schwarz method

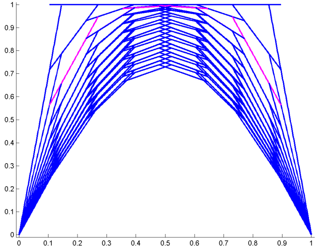
Screenshots of Schwarz solution (left) versus number of iterations (right) :

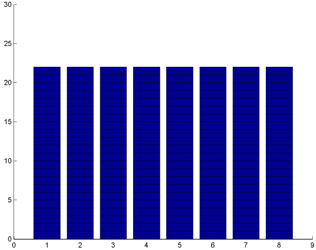




Synchronous parallel Schwarz method

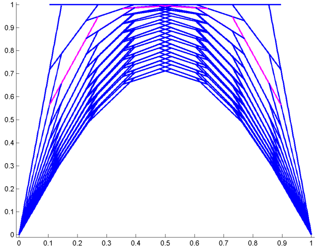
Screenshots of Schwarz solution (left) versus number of iterations (right) :

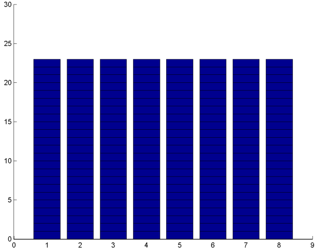




Synchronous parallel Schwarz method

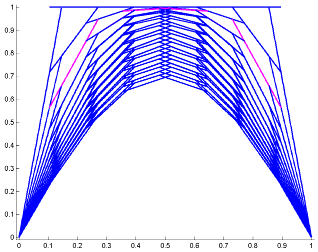
Screenshots of Schwarz solution (left) versus number of iterations (right) :

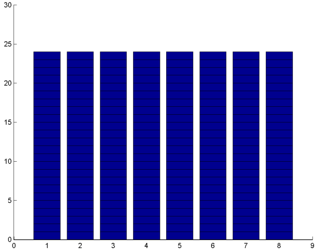




Synchronous parallel Schwarz method

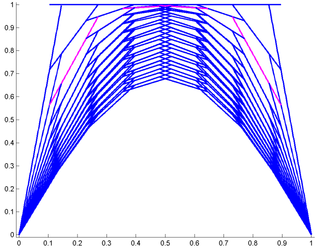
Screenshots of Schwarz solution (left) versus number of iterations (right) :

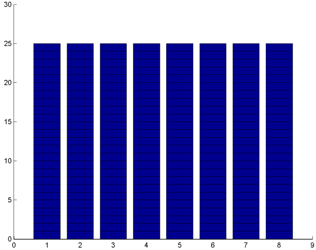




Synchronous parallel Schwarz method

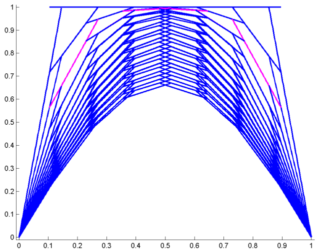
Screenshots of Schwarz solution (left) versus number of iterations (right) :

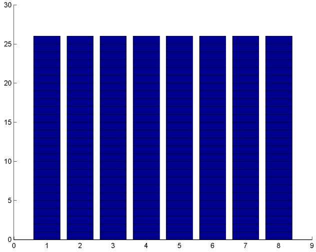




Synchronous parallel Schwarz method

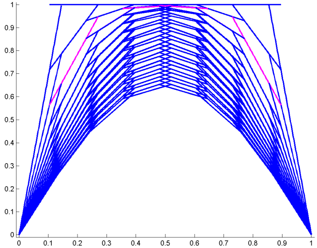
Screenshots of Schwarz solution (left) versus number of iterations (right) :

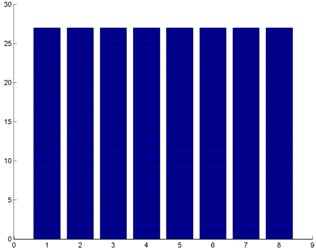




Synchronous parallel Schwarz method

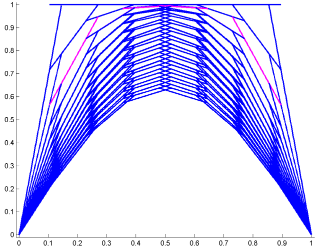
Screenshots of Schwarz solution (left) versus number of iterations (right) :

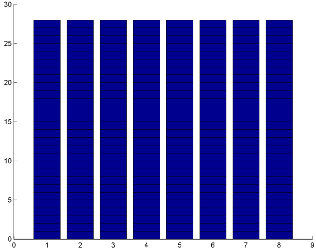




Synchronous parallel Schwarz method

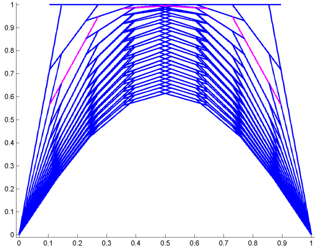
Screenshots of Schwarz solution (left) versus number of iterations (right) :

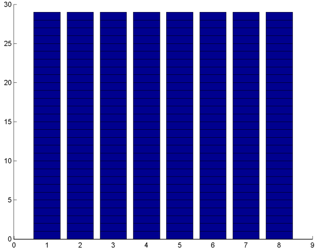




Synchronous parallel Schwarz method

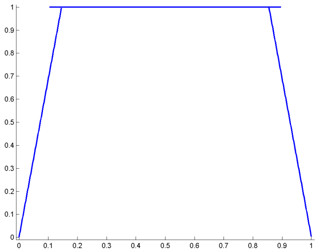
Screenshots of Schwarz solution (left) versus number of iterations (right) :

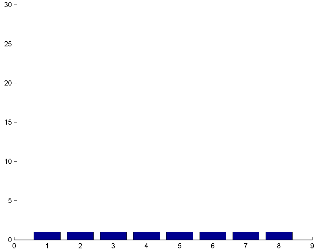




Asynchronous parallel Schwarz method

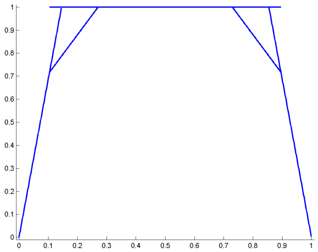
Screenshots of Schwarz solution (left) versus number of iterations (right) :

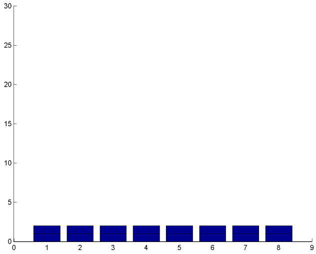




Asynchronous parallel Schwarz method

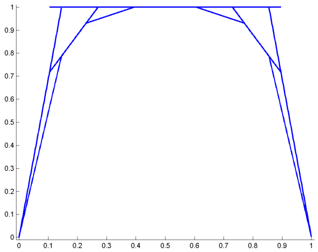
Screenshots of Schwarz solution (left) versus number of iterations (right) :

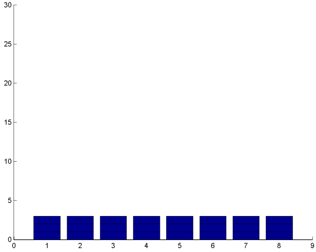




Asynchronous parallel Schwarz method

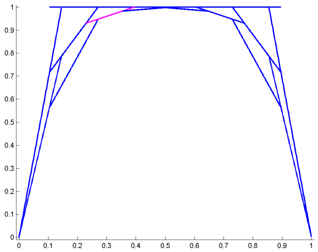
Screenshots of Schwarz solution (left) versus number of iterations (right) :

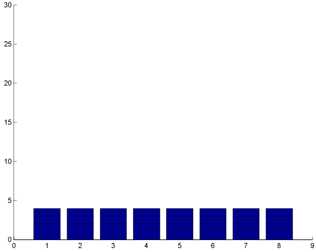




Asynchronous parallel Schwarz method

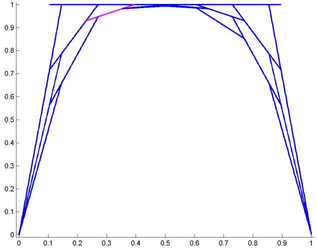
Screenshots of Schwarz solution (left) versus number of iterations (right) :

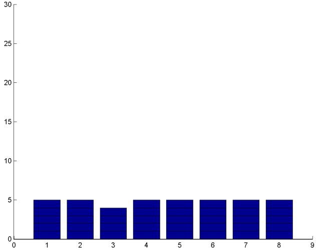




Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

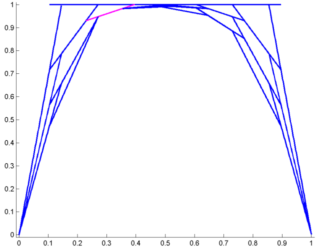


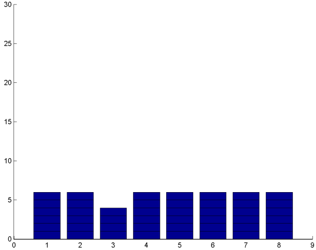


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and . . .

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

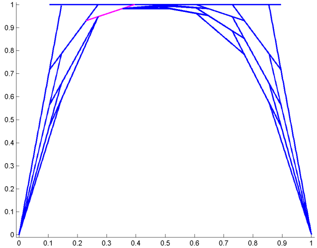


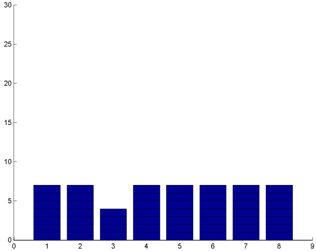


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and . . .

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

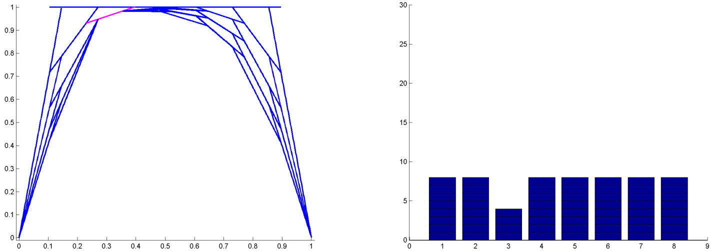




When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and ...

Asynchronous parallel Schwarz method

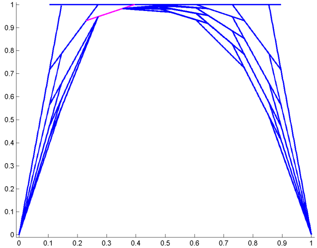
Screenshots of Schwarz solution (left) versus number of iterations (right) :

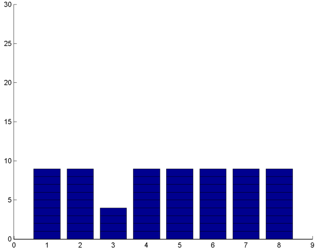


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and . . .

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

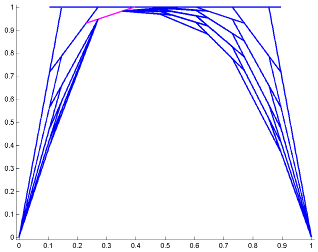


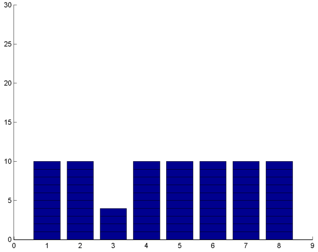


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and . . .

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

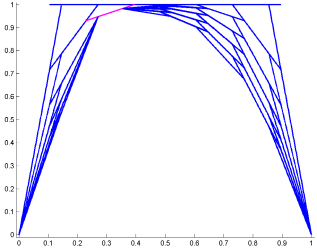


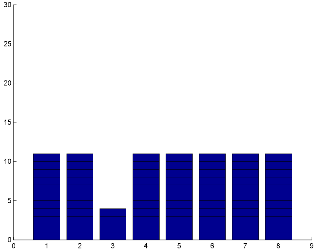


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and . . .

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

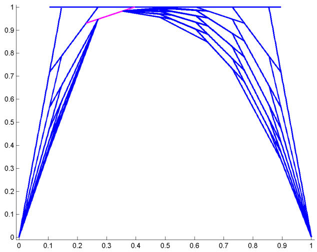


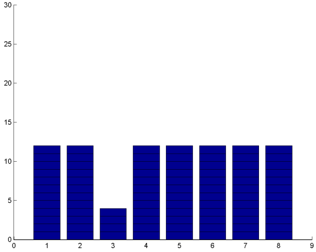


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and . . .

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

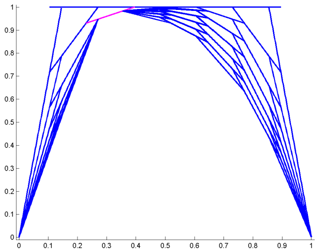


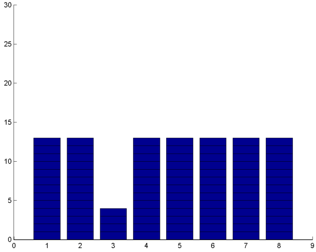


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and ...

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

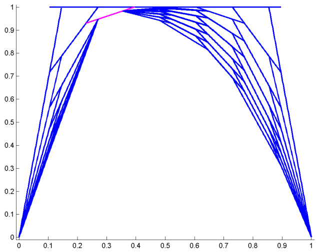


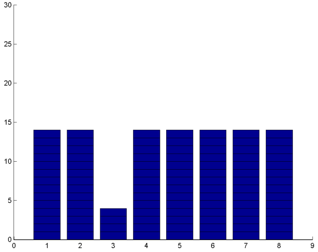


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and . . .

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

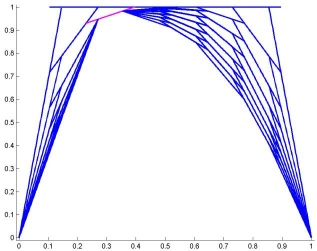


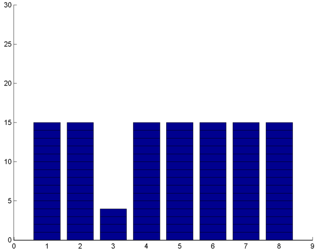


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and ...

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

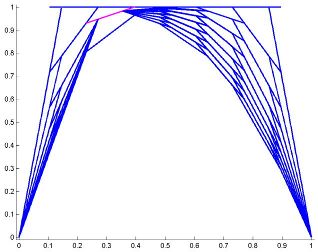


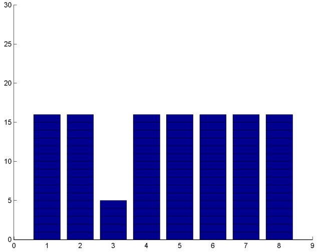


When processor 3 meets unexpected delay during iteration number 5, no processors wait for it, and . . .

Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :

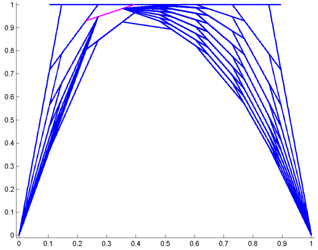


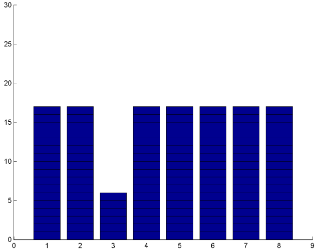


. . . when processor 3 has finished iteration number 5, it joins other processors work **and benefits from their last results**.

Asynchronous parallel Schwarz method

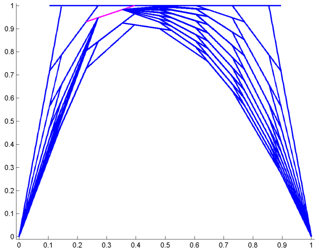
Screenshots of Schwarz solution (left) versus number of iterations (right) :

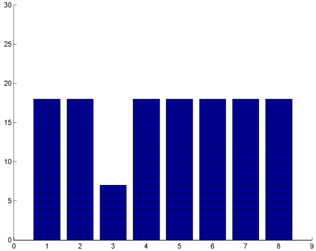




Asynchronous parallel Schwarz method

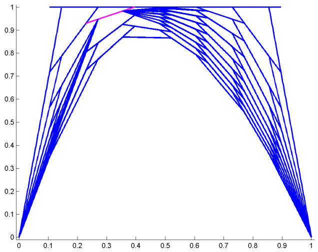
Screenshots of Schwarz solution (left) versus number of iterations (right) :

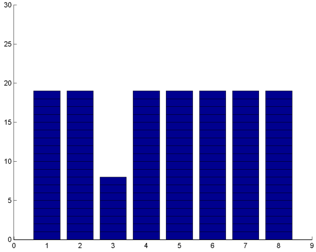




Asynchronous parallel Schwarz method

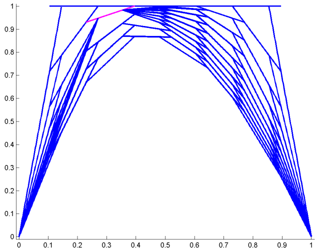
Screenshots of Schwarz solution (left) versus number of iterations (right) :

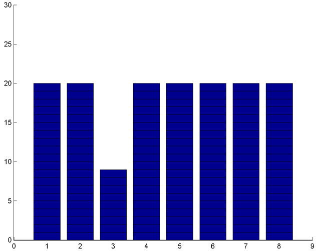




Asynchronous parallel Schwarz method

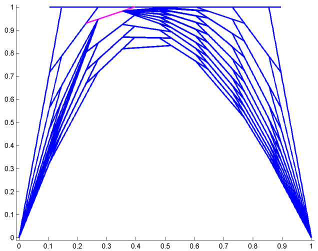
Screenshots of Schwarz solution (left) versus number of iterations (right) :

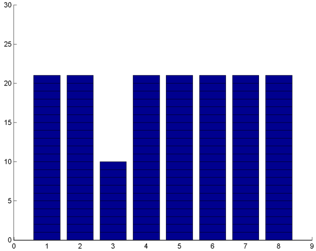




Asynchronous parallel Schwarz method

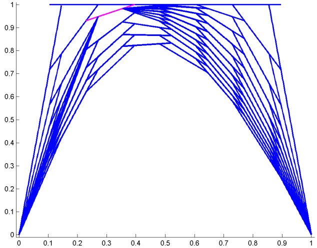
Screenshots of Schwarz solution (left) versus number of iterations (right) :

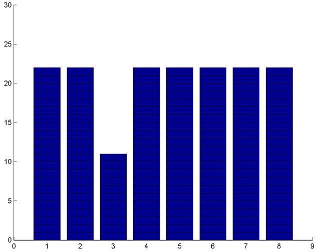




Asynchronous parallel Schwarz method

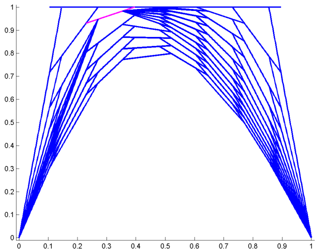
Screenshots of Schwarz solution (left) versus number of iterations (right) :

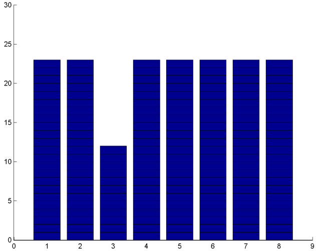




Asynchronous parallel Schwarz method

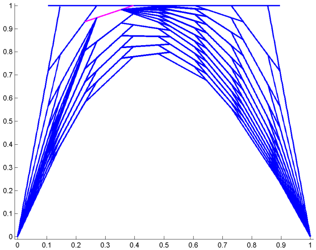
Screenshots of Schwarz solution (left) versus number of iterations (right) :

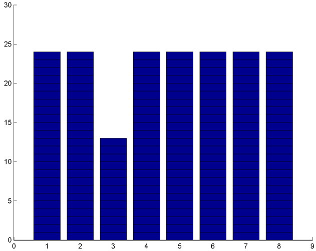




Asynchronous parallel Schwarz method

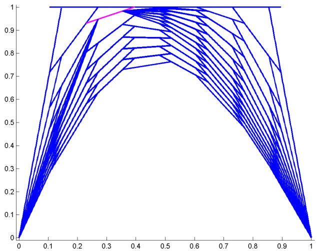
Screenshots of Schwarz solution (left) versus number of iterations (right) :

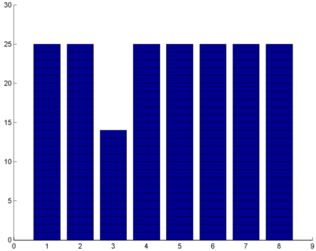




Asynchronous parallel Schwarz method

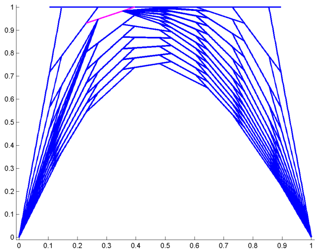
Screenshots of Schwarz solution (left) versus number of iterations (right) :

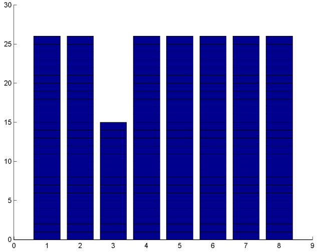




Asynchronous parallel Schwarz method

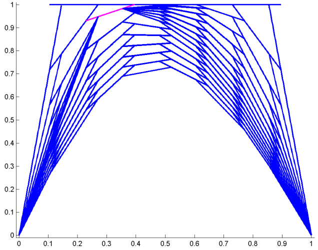
Screenshots of Schwarz solution (left) versus number of iterations (right) :

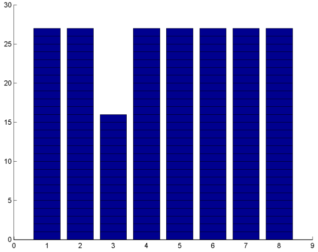




Asynchronous parallel Schwarz method

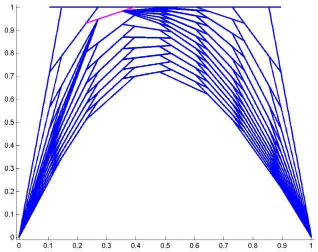
Screenshots of Schwarz solution (left) versus number of iterations (right) :

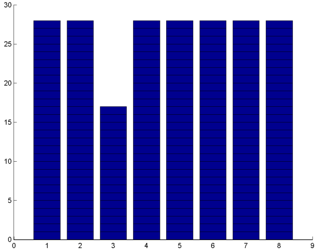




Asynchronous parallel Schwarz method

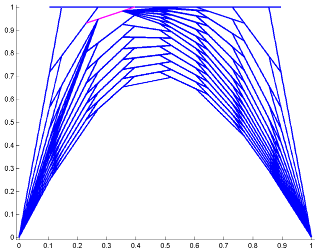
Screenshots of Schwarz solution (left) versus number of iterations (right) :

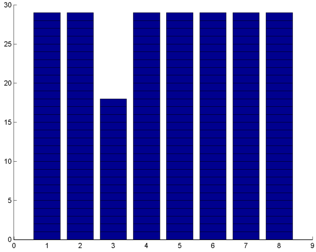




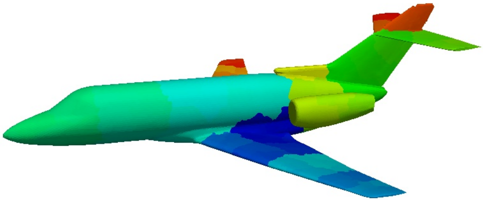
Asynchronous parallel Schwarz method

Screenshots of Schwarz solution (left) versus number of iterations (right) :





The End



Asynchronous domain decomposition methods

Serial parallel iterations vs parallel serial iterations

Guillaume Gbikpi-Benissan, Frédéric Magoulès
Univ. Paris Saclay, CentraleSupélec (France)

Outline

Outline

01 Parallel computing

Scalability' limits
Serial parallel iterations
Parallel serial iterations

Introduction

Parallel processing

- Speedup limit of parallel processing
  [Amdahl, 1967] . . .

Introduction

Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*

$t(p)$: processing time using p processors
α : serial proportion of the processing (data management)

Introduction

Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*

$t(p)$: processing time using p processors
α : serial proportion of the processing (data management)
$\Rightarrow$ Theoretical speedup :

$$s(p,\alpha) = \frac{t(1)}{t(p)} \leq \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

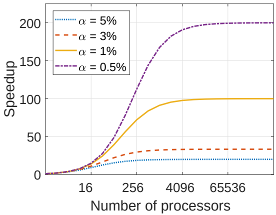
Introduction

Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*

$t(p)$: processing time using p processors
α : serial proportion of the processing (data management)
$\Rightarrow$ Theoretical speedup :

$$s(p,\alpha) = \frac{t(1)}{t(p)} \leq \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

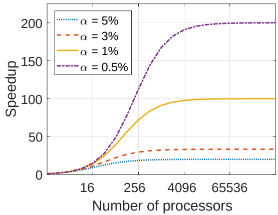


Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*

$t(p)$: processing time using p processors
α : serial proportion of the processing (data management)
$\Rightarrow$ Theoretical speedup :

$$s(p, \alpha) = \frac{t(1)}{t(p)} \leq \frac{1}{\alpha + \frac{1-\alpha}{p}}$$



Minimize α : input/output, pre/post-processing, ..., and *inter-process communication*

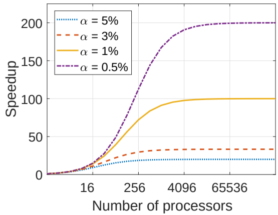
Introduction

Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*

$t(p)$: processing time using p processors
α : serial proportion of the processing (data management)
$\Rightarrow$ Theoretical speedup :

$$s(p,\alpha) = \frac{t(1)}{t(p)} \leq \frac{1}{\alpha + \frac{1-\alpha}{p}}$$



Minimize α : input/output, pre/post-processing, ..., and *inter-process communication*

- Speedup limit of load balancing and *fault-tolerance*

Introduction

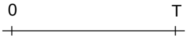
Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation . . .

Introduction

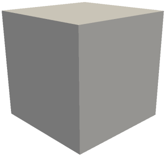
Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation



Partial differential equations

$\delta(u(s,t),s,t) = 0, \quad t \in \mathbb{R}^+, \; s \in \Omega \subset \mathbb{R}^3$



Introduction

Parallel processing

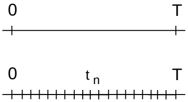
- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

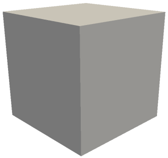
Partial differential equations

$$\delta(u(s,t),s,t)=0, \quad t\in\mathbb{R}^+,\ \ s\in\Omega\subset\mathbb{R}^3$$

Time discretization

$$\alpha(u(s,t_{n+1}))=\beta(u(s,t_n))$$

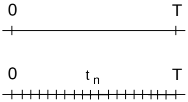




Introduction

Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation



Partial differential equations

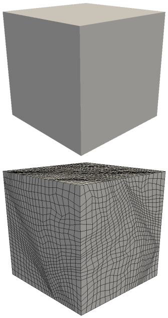
$\delta(u(s,t),s,t) = 0, \quad t \in \mathbb{R}^+, \;\; s \in \Omega \subset \mathbb{R}^3$

Time discretization

$$\alpha(u(s,t_{n+1})) = \beta(u(s,t_n))$$

Space discretization

$$AU_{n+1} = B_n, \quad U_{n+1} \in \mathbb{C}^m$$



Introduction

Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

Partial differential equations

$$\delta(u(s,t),s,t) = 0, \quad t \in \mathbb{R}^+, \ s \in \Omega \subset \mathbb{R}^3$$

Time discretization

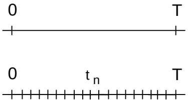
$$\alpha(u(s,t_{n+1})) = \beta(u(s,t_n))$$

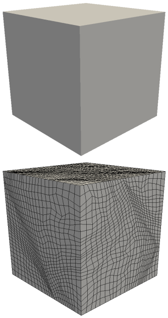
Space discretization

$$AU_{n+1} = B_n, \quad U_{n+1} \in \mathbb{C}^m$$

Sparse system of equations

$$Ax = b, \quad x \in \mathbb{C}^m$$





Introduction

Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

Partial differential equations

$$\delta(u(s,t),s,t) = 0, \quad t \in \mathbb{R}^+, \ \ s \in \Omega \subset \mathbb{R}^3$$

Time discretization

$$\alpha(u(s,t_{n+1})) = \beta(u(s,t_n))$$

Space discretization

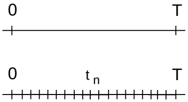
$$AU_{n+1} = B_n, \quad U_{n+1} \in \mathbb{C}^m$$

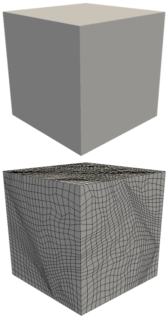
Sparse system of equations

$$Ax = b, \quad x \in \mathbb{C}^m$$

Iterative methods

$$x^{k+1} = f(x^k)$$

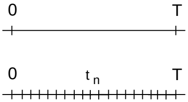




Introduction

Parallel processing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation



Partial differential equations

$$\delta(u(s,t),s,t)=0, \quad t\in\mathbb{R}^+,\ s\in\Omega\subset\mathbb{R}^3$$

Time discretization

$$\alpha(u(s,t_{n+1}))=\beta(u(s,t_n))$$

Space discretization

$$AU_{n+1}=B_n, \quad U_{n+1}\in\mathbb{C}^m$$

Sparse system of equations

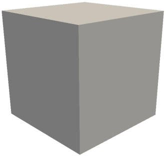
$$Ax=b, \quad x\in\mathbb{C}^m$$

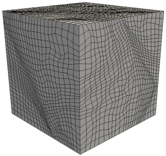
Iterative methods

$$x^{k+1}=f(x^k)$$

$$x_i^{k+1}=f_i(x^k),\ \forall i\in\{1,\ldots,p\}$$

$$x:=\begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}, \quad p\leq m$$





Parallel computing

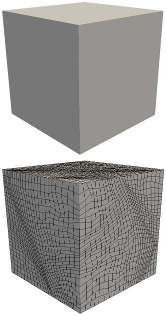
Introduction

Parallel computing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

$\delta(u(s,t),s,t) = 0, \quad t \in [0,T], \quad s \in \Omega$

- Domain decomposition methods (in space)



Introduction

Parallel computing

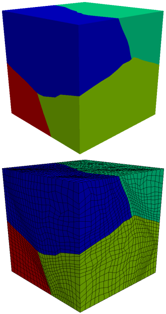
- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

$\delta(u(s,t),s,t) = 0, \quad t \in [0,T], \quad s \in \Omega$

- Domain decomposition methods (in space)

Different subdomains

$$\Omega^{(1)}, \ldots, \Omega^{(p)}$$



Introduction

Parallel computing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

$\delta(u(s,t),s,t) = 0, \quad t \in [0,T], \quad s \in \Omega$

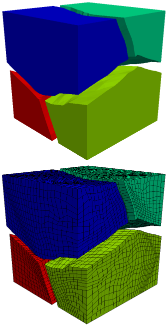
- Domain decomposition methods (in space)

Different subdomains

$$\Omega^{(1)}, \dots, \Omega^{(p)}$$

Parallel solutions

$$u^{(1)}(s, t_{n+1}), \dots, u^{(p)}(s, t_{n+1})$$



Introduction

Parallel computing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

$\delta(u(s,t), s, t) = 0, \quad t \in [0, T], \quad s \in \Omega$

- Domain decomposition methods (in space)

Different subdomains

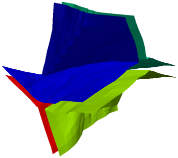
$$\Omega^{(1)}, \dots, \Omega^{(p)}$$

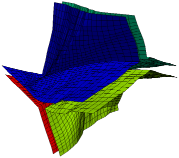
Parallel solutions

$$u^{(1)}(s, t_{n+1}), \dots, u^{(p)}(s, t_{n+1})$$

Consistency across interfaces Γ_j^i, $i, j \in \{1, \dots, p\}$

$$u^{(i)}(s_{\Gamma_j^i}, t_{n+1}) = u^{(j)}(s_{\Gamma_j^i}, t_{n+1})$$





Introduction

Parallel computing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

$\delta(u(s,t),s,t)=0, \quad t\in [0,T], \quad s\in\Omega$

- Domain decomposition methods (in space)

Different subdomains

$$\Omega^{(1)},\ldots,\Omega^{(p)}$$

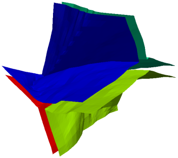
Parallel solutions

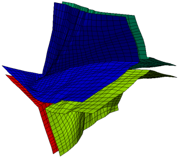
$$u^{(1)}(s,t_{n+1}),\ldots,u^{(p)}(s,t_{n+1})$$

Consistency across interfaces Γ_j^i, $i,j\in\{1,\ldots,p\}$

$$u^{(i)}(s_{\Gamma_j^i},t_{n+1})=u^{(j)}(s_{\Gamma_j^i},t_{n+1})$$

+ parallel input/output
+ parallel pre/post-processing





Introduction

Parallel computing

- Speedup limit of parallel processing *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

$\delta(u(s,t),s,t)=0, \quad t \in [0,T], \quad s \in \Omega$

- Domain decomposition methods (in space)

Different subdomains

$$\Omega^{(1)},\dots,\Omega^{(p)}$$

Parallel solutions

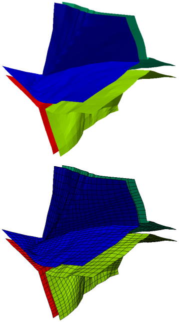
$$u^{(1)}(s,t_{n+1}),\dots,u^{(p)}(s,t_{n+1})$$

Consistency across interfaces Γ_j^i, $i,j \in \{1,\dots,p\}$

$$u^{(i)}(s_{\Gamma_j^i},t_{n+1}) = u^{(j)}(s_{\Gamma_j^i},t_{n+1})$$

$+$ parallel input/output
$+$ parallel pre/post-processing

$+$ static load balancing



Introduction

Parallel computing

- Speedup limit *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

$\delta(u(s,t),s,t)=0, \quad t\in [0,T], \quad s\in\Omega$

- Domain decomposition methods

Different subdomains

$$\Omega^{(1)},\dots,\Omega^{(p)}$$

Parallel solutions

$$u^{(1)}(s,t_{n+1}),\dots,u^{(p)}(s,t_{n+1})$$

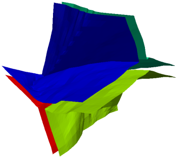
Consistency across interfaces Γ_j^i, $i,j\in\{1,\dots,p\}$

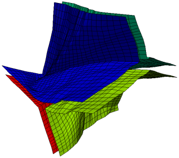
$$u^{(i)}(s_{\Gamma_j^i},t_{n+1})=u^{(j)}(s_{\Gamma_j^i},t_{n+1})$$

$+$ parallel input/output
$+$ parallel pre/post-processing

$+$ static load balancing

$-$ serial parallel iterations
$\Rightarrow$ blocking inter-process synchronization





Introduction

Parallel computing

- Speedup limit *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation

$\delta(u(s,t), s, t) = 0, \quad t \in [0, T], \quad s \in \Omega$

- Domain decomposition methods

Different subdomains

$$\Omega^{(1)}, \dots, \Omega^{(p)}$$

Parallel solutions

$$u^{(1)}(s, t_{n+1}), \dots, u^{(p)}(s, t_{n+1})$$

Consistency across interfaces Γ_j^i, $i, j \in \{1, \dots, p\}$

$$u^{(i)}(s_{\Gamma_j^i}, t_{n+1}) = u^{(j)}(s_{\Gamma_j^i}, t_{n+1})$$

\+ parallel input/output

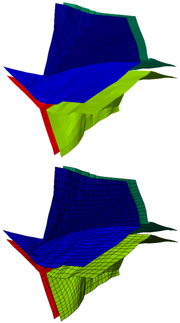
\+ parallel pre/post-processing

\+ static load balancing

− serial parallel iterations

⇒ blocking inter-process synchronization

⇒ − non fault-tolerant



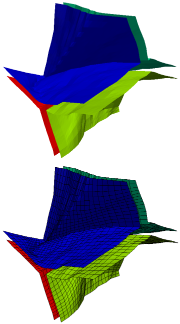
Introduction

Parallel computing

- Speedup limit [*Amdahl, 1967*]
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation
- Domain decomposition methods

+ static load balancing
– serial parallel iterations
– non fault-tolerant

- Asynchronous iterations [*Chazan and Miranker, 1969*]



Introduction

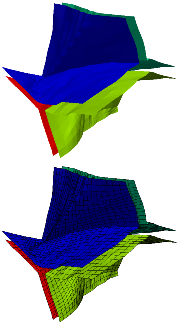
Parallel computing

- Speedup limit *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation
- Domain decomposition methods

+ static load balancing
– serial parallel iterations
– non fault-tolerant

- Asynchronous iterations *[Chazan and Miranker, 1969]*

+ parallel serial iterations
$\Rightarrow$ no speedup limit



Introduction

Parallel computing

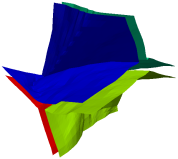
- Speedup limit *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation
- Domain decomposition methods

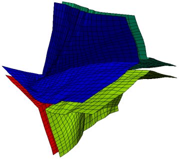
+ static load balancing
– serial parallel iterations
– non fault-tolerant

- Asynchronous iterations *[Chazan and Miranker, 1969]*

+ parallel serial iterations
⇒ no speedup limit

+ non-deterministic inter-process communication ⇒ fault-tolerance





Introduction

Parallel computing

- Speedup limit *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation
- Domain decomposition methods

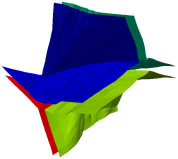
+ static load balancing
– serial parallel iterations
– non fault-tolerant

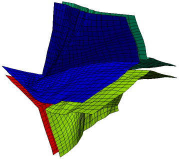
- Asynchronous iterations *[Chazan and Miranker, 1969]*

+ parallel serial iterations
⇒ no speedup limit

+ non-deterministic inter-process communication ⇒ fault-tolerance

+ dynamic adaptation to unbalanced load





Introduction

Parallel computing

- Speedup limit *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation
- Domain decomposition methods

+ static load balancing
– serial parallel iterations
– non fault-tolerant

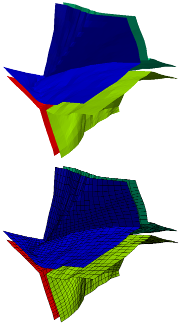
- Asynchronous iterations *[Chazan and Miranker, 1969]*

+ parallel serial iterations
⇒ no speedup limit

+ non-deterministic inter-process communication ⇒ fault-tolerance

+ dynamic adaptation to unbalanced load

– low convergence rate



Introduction

Parallel computing

- Speedup limit *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation
- Domain decomposition methods

+ static load balancing
– serial parallel iterations
– non fault-tolerant

- Asynchronous iterations *[Chazan and Miranker, 1969]*

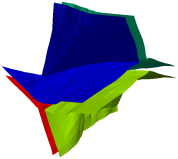
+ parallel serial iterations
$\Rightarrow$ no speedup limit

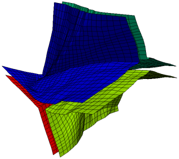
+ non-deterministic inter-process communication $\Rightarrow$ fault-tolerance

+ dynamic adaptation to unbalanced load

– low convergence rate

Eventual consistency across interfaces ?
(convergence conditions)





Introduction

Parallel computing

- Speedup limit *[Amdahl, 1967]*
- Speedup limit of load balancing and fault-tolerance
- Numerical simulation
- Domain decomposition methods

+ static load balancing
– serial parallel iterations
– non fault-tolerant

- Asynchronous iterations *[Chazan and Miranker, 1969]*

+ parallel serial iterations
⇒ no speedup limit

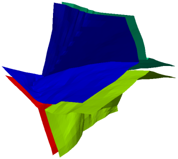
+ non-deterministic inter-process communication ⇒ fault-tolerance

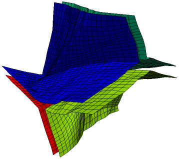
+ dynamic adaptation to unbalanced load

– low convergence rate

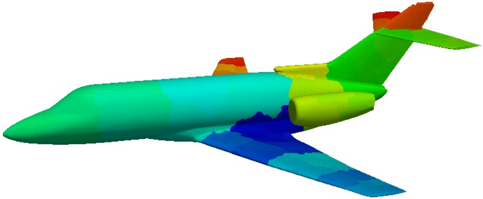
Eventual consistency across interfaces ?
(convergence conditions)

Consistency reached ?
(convergence detection)





The End



Asynchronous domain decomposition methods

Asynchronous iterative methods

Guillaume Gbikpi-Benissan, Frédéric Magoulès
Univ. Paris Saclay, CentraleSupélec (France)

Outline

Outline

01 Synchronous and asynchronous iterative methods

How synchronous iterations work ?
How asynchronous iterations work ?

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

$$\rho(M^{-1}N) < 1$$

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods ⇒ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

$$\rho(M^{-1}N) < 1$$

Parallel computing with p processors, $p < n$

$$f(x) = \begin{bmatrix} f_1(x) & \cdots & f_p(x) \end{bmatrix}^{\mathsf{T}}$$

$$x = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^{\mathsf{T}}$$

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

$$\rho(M^{-1}N) < 1$$

Parallel computing with p processors, $p \le n$

$$f(x) = \begin{bmatrix} f_1(x) & \cdots & f_p(x) \end{bmatrix}^\mathsf{T}$$

$$x = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^\mathsf{T}$$

$$x_i^{k+1} = f_i(x_1^k, \dots, x_p^k), \quad \forall i \in \{1, \dots, p\}$$

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

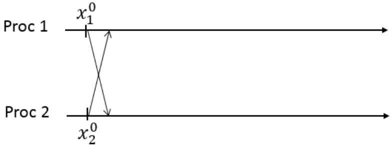
$$\rho(M^{-1}N) < 1$$

Parallel computing with p processors, $p \leq n$

$$f(x) = \begin{bmatrix} f_1(x) & \cdots & f_p(x) \end{bmatrix}^\mathsf{T}$$

$$x = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^\mathsf{T}$$

$$x_i^{k+1} = f_i(x_1^k, \dots, x_p^k), \quad \forall i \in \{1, \dots, p\}$$



Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k \in \mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k \to \infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

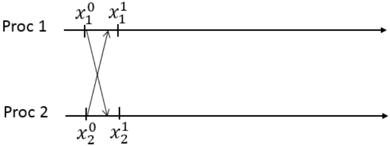
$$\rho(M^{-1}N) < 1$$

Parallel computing with p processors, $p < n$

$$f(x) = \begin{bmatrix} f_1(x) & \cdots & f_p(x) \end{bmatrix}^\mathsf{T}$$

$$x = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^\mathsf{T}$$

$$x_i^{k+1} = f_i(x_1^k, \dots, x_p^k), \quad \forall i \in \{1, \dots, p\}$$



$$x_1^1 := f_1(x_1^0, x_2^0) \quad x_2^1 := f_2(x_1^0, x_2^0)$$

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

$$\rho(M^{-1}N) < 1$$

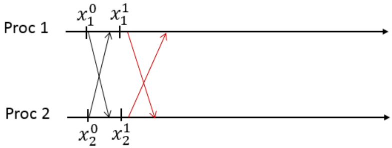
Parallel computing with p processors, $p < n$

$$f(x) = \begin{bmatrix} f_1(x) & \cdots & f_p(x) \end{bmatrix}^\mathsf{T}$$

$$x = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^\mathsf{T}$$

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

speedup limit



$$x_1^1 := f_1(x_1^0, x_2^0) \qquad x_2^1 := f_2(x_1^0, x_2^0)$$

wait wait

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

$$\rho(M^{-1}N) < 1$$

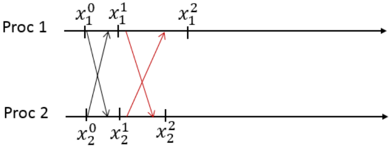
Parallel computing with p processors, $p \leq n$

$$f(x) = \begin{bmatrix} f_1(x) & \cdots & f_p(x) \end{bmatrix}^\mathsf{T}$$

$$x = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^\mathsf{T}$$

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

speedup limit



$$x_1^1 := f_1(x_1^0, x_2^0) \qquad x_2^1 := f_2(x_1^0, x_2^0)$$

wait　　wait

$$x_1^2 := f_1(x_1^1, x_2^1) \qquad x_2^2 := f_2(x_1^1, x_2^1)$$

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

$$\rho(M^{-1}N) < 1$$

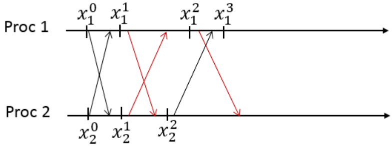
Parallel computing with p processors, $p \leq n$

$$f(x) = \begin{bmatrix} f_1(x) & \cdots & f_p(x) \end{bmatrix}^{\mathsf{T}}$$

$$x = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^{\mathsf{T}}$$

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

speedup limit



$$x_1^1 := f_1(x_1^0, x_2^0) \qquad x_2^1 := f_2(x_1^0, x_2^0)$$

wait wait

$$x_1^2 := f_1(x_1^1, x_2^1) \qquad x_2^2 := f_2(x_1^1, x_2^1)$$

$$x_1^3 := f_1(x_1^2, x_2^2)$$

wait

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

$$\rho(M^{-1}N) < 1$$

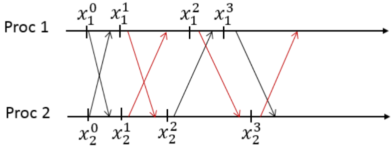
Parallel computing with p processors, $p \leq n$

$$f(x) = \begin{bmatrix} f_1(x) & \cdots & f_p(x) \end{bmatrix}^\mathsf{T}$$

$$x = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^\mathsf{T}$$

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

speedup limit



$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
wait	wait
$x_1^2 := f_1(x_1^1, x_2^1)$	$x_2^2 := f_2(x_1^1, x_2^1)$
$x_1^3 := f_1(x_1^2, x_2^2)$	wait
wait	$x_2^3 := f_2(x_1^2, x_2^2)$

Asynchronous iterative methods

Problem

$$Ax = b, \quad x \in \mathbb{C}^n$$

Splitting

$$A = M - N$$

Mapping

$$f(x) := M^{-1}Nx + M^{-1}b$$

Fixed-point problem

$$Ax = b \iff x = f(x)$$

Iterative methods $\Rightarrow$ sequence $\{x^k\}_{k\in\mathbb{N}}$:

$$x^{k+1} = f(x^k)$$

Convergence from any initial vector x^0

$$\lim_{k\to\infty} x^k = x^*, \quad f(x^*) = x^*$$

Convergence condition (sufficient and necessary)

$$\rho(M^{-1}N) < 1$$

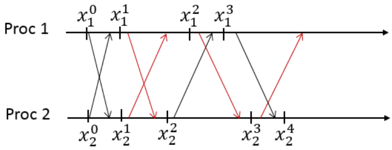
Parallel computing with p processors, $p < n$

$$f(x) = \begin{bmatrix} f_1(x) & \cdots & f_p(x) \end{bmatrix}^{\mathsf{T}}$$

$$x = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^{\mathsf{T}}$$

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

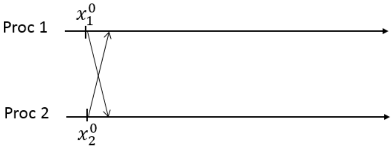
speedup limit



$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
wait	wait
$x_1^2 := f_1(x_1^1, x_2^1)$	$x_2^2 := f_2(x_1^1, x_2^1)$
$x_1^3 := f_1(x_1^2, x_2^2)$	wait
wait	$x_2^3 := f_2(x_1^2, x_2^2)$
wait	$x_2^4 := f_2(x_1^3, x_2^3)$

Asynchronous iterative methods

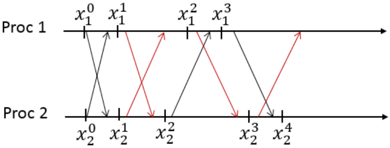
Asynchronous iterations



Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \dots, x_p^k), \quad \forall i \in \{1, \dots, p\}$$

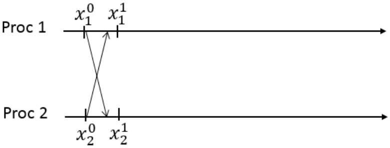
delay $\Rightarrow$ speedup limit



$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
wait	wait
$x_1^2 := f_1(x_1^1, x_2^1)$	$x_2^2 := f_2(x_1^1, x_2^1)$
$x_1^3 := f_1(x_1^2, x_2^2)$	wait
wait	$x_2^3 := f_2(x_1^2, x_2^2)$
wait	$x_2^4 := f_2(x_1^3, x_2^3)$

Asynchronous iterative methods

Asynchronous iterations

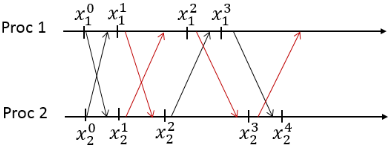


$x_1^1 := f_1(x_1^0, x_2^0) \quad x_2^1 := f_2(x_1^0, x_2^0)$

Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

delay $\Rightarrow$ speedup limit



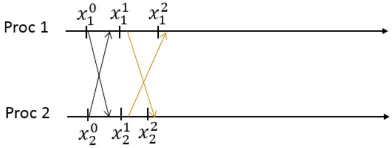
$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
wait	wait
$x_1^2 := f_1(x_1^1, x_2^1)$	$x_2^2 := f_2(x_1^1, x_2^1)$
$x_1^3 := f_1(x_1^2, x_2^2)$	wait
wait	$x_2^3 := f_2(x_1^2, x_2^2)$
wait	$x_2^4 := f_2(x_1^3, x_2^3)$

Asynchronous iterative methods

Asynchronous iterations	Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

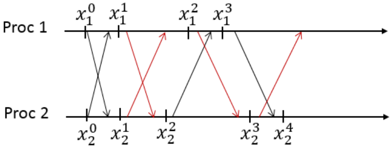
delay $\Rightarrow$ low convergence rate



$x_1^1 := f_1(x_1^0, x_2^0) \quad x_2^1 := f_2(x_1^0, x_2^0)$

$x_1^2 := f_1(x_1^1, x_2^0) \quad x_2^2 := f_2(x_1^0, x_2^1)$

delay $\Rightarrow$ speedup limit

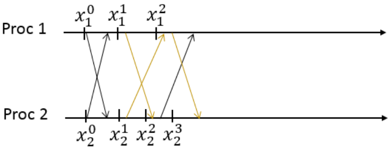


$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
wait	wait
$x_1^2 := f_1(x_1^1, x_2^1)$	$x_2^2 := f_2(x_1^1, x_2^1)$
$x_1^3 := f_1(x_1^2, x_2^2)$	wait
wait	$x_2^3 := f_2(x_1^2, x_2^2)$
wait	$x_2^4 := f_2(x_1^3, x_2^3)$

Asynchronous iterative methods

Asynchronous iterations

delay $\Rightarrow$ low convergence rate



$x_1^1 := f_1(x_1^0, x_2^0)$ $\quad$ $x_2^1 := f_2(x_1^0, x_2^0)$

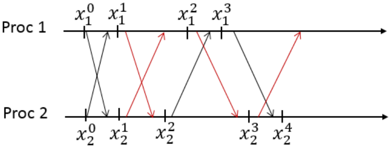
$x_1^2 := f_1(x_1^1, x_2^0)$ $\quad$ $x_2^2 := f_2(x_1^0, x_2^1)$

$x_2^3 := f_2(x_1^1, x_2^2)$

Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \dots, x_p^k), \quad \forall i \in \{1, \dots, p\}$$

delay $\Rightarrow$ speedup limit



$x_1^1 := f_1(x_1^0, x_2^0)$ $\quad$ $x_2^1 := f_2(x_1^0, x_2^0)$

wait $\quad$ wait

$x_1^2 := f_1(x_1^1, x_2^1)$ $\quad$ $x_2^2 := f_2(x_1^1, x_2^1)$

$x_1^3 := f_1(x_1^2, x_2^2)$ $\quad$ wait

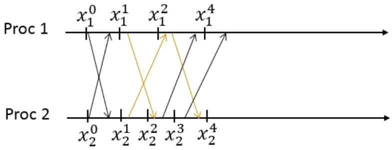
wait $\quad$ $x_2^3 := f_2(x_1^2, x_2^2)$

wait $\quad$ $x_2^4 := f_2(x_1^3, x_2^3)$

Asynchronous iterative methods

Asynchronous iterations

delay $\Rightarrow$ low convergence rate

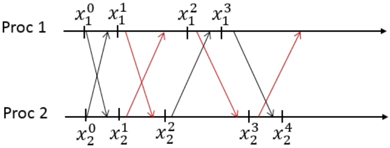


$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
$x_1^2 := f_1(x_1^1, x_2^0)$	$x_2^2 := f_2(x_1^0, x_2^1)$
$x_1^3 := x_1^2$	$x_2^3 := f_2(x_1^1, x_2^2)$
$x_1^4 := f_1(x_1^3, x_2^2)$	$x_2^4 := f_2(x_1^2, x_2^3)$

Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

delay $\Rightarrow$ speedup limit

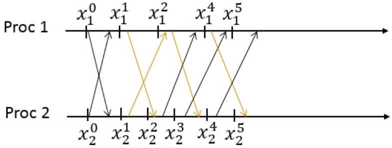


$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
wait	wait
$x_1^2 := f_1(x_1^1, x_2^1)$	$x_2^2 := f_2(x_1^1, x_2^1)$
$x_1^3 := f_1(x_1^2, x_2^2)$	wait
wait	$x_2^3 := f_2(x_1^2, x_2^2)$
wait	$x_2^4 := f_2(x_1^3, x_2^3)$

Asynchronous iterative methods

Asynchronous iterations

delay $\Rightarrow$ low convergence rate

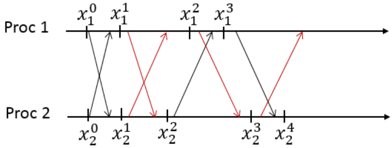


$$
\begin{array}{ll}
x_1^1 := f_1(x_1^0, x_2^0) & x_2^1 := f_2(x_1^0, x_2^0) \\
x_1^2 := f_1(x_1^1, x_2^0) & x_2^2 := f_2(x_1^0, x_2^1) \\
x_1^3 := x_1^2 & x_2^3 := f_2(x_1^1, x_2^2) \\
x_1^4 := f_1(x_1^3, x_2^2) & x_2^4 := f_2(x_1^2, x_2^3) \\
x_1^5 := f_1(x_1^4, x_2^3) & x_2^5 := f_2(x_1^2, x_2^4)
\end{array}
$$

Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \dots, x_p^k), \quad \forall i \in \{1, \dots, p\}$$

delay $\Rightarrow$ speedup limit



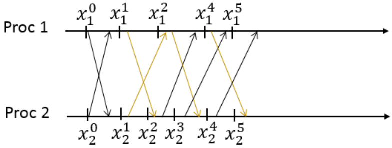
$$
\begin{array}{ll}
x_1^1 := f_1(x_1^0, x_2^0) & x_2^1 := f_2(x_1^0, x_2^0) \\
\text{wait} & \text{wait} \\
x_1^2 := f_1(x_1^1, x_2^1) & x_2^2 := f_2(x_1^1, x_2^1) \\
x_1^3 := f_1(x_1^2, x_2^2) & \text{wait} \\
\text{wait} & x_2^3 := f_2(x_1^2, x_2^2) \\
\text{wait} & x_2^4 := f_2(x_1^3, x_2^3)
\end{array}
$$

Asynchronous iterative methods

Asynchronous iterations

$$x_i^{k+1} = f_i(x_1^{\tau_1^i(k)}, \dots, x_p^{\tau_p^i(k)}), \quad \forall i \in P^k$$
$$x_i^{k+1} = x_i^k, \quad \forall i \notin P^k$$

delay $\Rightarrow$ low convergence rate



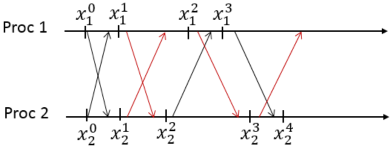
$$\begin{array}{ll}
x_1^1 := f_1(x_1^0, x_2^0) & x_2^1 := f_2(x_1^0, x_2^0) \\
x_1^2 := f_1(x_1^1, x_2^0) & x_2^2 := f_2(x_1^0, x_2^1) \\
x_1^3 := x_1^2 & x_2^3 := f_2(x_1^1, x_2^2) \\
x_1^4 := f_1(x_1^3, x_2^2) & x_2^4 := f_2(x_1^2, x_2^3) \\
x_1^5 := f_1(x_1^4, x_2^3) & x_2^5 := f_2(x_1^2, x_2^4)
\end{array}$$

$P^k \subset \{1, \dots, p\}, \qquad \tau_j^i(k) \leq k$

Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \dots, x_p^k), \quad \forall i \in \{1, \dots, p\}$$

delay $\Rightarrow$ speedup limit



$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
wait	wait
$x_1^2 := f_1(x_1^1, x_2^1)$	$x_2^2 := f_2(x_1^1, x_2^1)$
$x_1^3 := f_1(x_1^2, x_2^2)$	wait
wait	$x_2^3 := f_2(x_1^2, x_2^2)$
wait	$x_2^4 := f_2(x_1^3, x_2^3)$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

Asynchronous iterations

$$\begin{aligned} x_i^{k+1} &= f_i(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}), \quad && \forall i \in P^k \\ x_i^{k+1} &= x_i^k, && \forall i \notin P^k \end{aligned}$$

Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

Convergence condition (necessary and sufficient)

$$\rho(M^{-1}N) < 1$$

02 Mathematical convergence of asynchronous iterative methods

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

Asynchronous iterations

$$\begin{aligned} x_i^{k+1} &= f_i(x_1^{\tau_1^i(k)}, \dots, x_p^{\tau_p^i(k)}), \quad && \forall i \in P^k \\ x_i^{k+1} &= x_i^k, && \forall i \notin P^k \end{aligned}$$

Convergence condition (necessary and sufficient)
[Chazan and Miranker, 1969]

$$\rho(|M^{-1}N|) < 1$$

Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \dots, x_p^k), \quad \forall i \in \{1, \dots, p\}$$

Convergence condition (necessary and sufficient)

$$\rho(M^{-1}N) < 1$$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

Asynchronous iterations

$$\begin{aligned} x_i^{k+1} &= f_i(x_1^{\tau_1^i(k)}, \dots, x_p^{\tau_p^i(k)}), & \forall i \in P^k \\ x_i^{k+1} &= x_i^k, & \forall i \notin P^k \end{aligned}$$

Convergence condition (necessary and sufficient)
[Chazan and Miranker, 1969]

$$\rho(M^{-1}N) \leq \quad \rho(|M^{-1}N|) < 1$$

Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \dots, x_p^k), \quad \forall i \in \{1, \dots, p\}$$

Convergence condition (necessary and sufficient)

$$\rho(M^{-1}N) < 1$$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

Asynchronous iterations

$$\begin{aligned} x_i^{k+1} &= f_i(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}), & \forall i \in P^k \\ x_i^{k+1} &= x_i^k, & \forall i \notin P^k \end{aligned}$$

Convergence condition (necessary and sufficient)
[Chazan and Miranker, 1969]

$$\rho(M^{-1}N) \leq \rho(|M^{-1}N|) < 1$$

Synchronous iterations

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \quad \forall i \in \{1, \ldots, p\}$$

Convergence condition (necessary and sufficient)

$$\rho(M^{-1}N) < 1$$

General fixed-point problems

$$f^{(k)}(x, x, \ldots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : E^m \mapsto E, \quad m \in \mathbb{N}^*$$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

[Chazan and Miranker, 1969] (necessary and sufficient) : $\rho(|M^{-1}N|) < 1$

- General fixed-point problems

$$f^{(k)}(x, x, \ldots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : \; E^m \mapsto E, \quad m \in \mathbb{N}^*$$

$$m = 1, \quad f^{(k)} \equiv f, \quad \forall k$$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

[Chazan and Miranker, 1969] (necessary and sufficient) : $\rho(|M^{-1}N|) < 1$

- General fixed-point problems

$$f^{(k)}(x, x, \dots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : E^m \mapsto E, \quad m \in \mathbb{N}^*$$

$$m = 1, \quad f^{(k)} \equiv f, \quad \forall k$$

[Miellou, 1975] (sufficient)

$$|f(x) - f(y)| \leq T|x - y|$$

$$T \geq O, \quad \rho(T) < 1, \quad |x| = (|x_1|, \dots, |x_p|)$$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

[Chazan and Miranker, 1969] (necessary and sufficient) : $\rho(|M^{-1}N|) < 1$

- General fixed-point problems

$$f^{(k)}(x, x, \dots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : E^m \mapsto E, \quad m \in \mathbb{N}^*$$

$$m = 1, \quad f^{(k)} \equiv f, \quad \forall k$$

[Miellou, 1975] (sufficient)

$$|f(x) - f(y)| \leq T|x - y|$$

$T \geq O, \;\; \rho(T) < 1, \;\; |x| = (|x_1|, \dots, |x_p|)$

[El Tarazi, 1982] (sufficient)

$$\|f(x) - f(y)\|_\infty^w \leq \alpha \|x - y\|_\infty^w$$

$$w > 0, \;\; \alpha < 1, \;\; \|x\|_\infty^w = \max_i |x_i|/w_i$$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

[Chazan and Miranker, 1969] (necessary and sufficient) : $\rho(|M^{-1}N|) < 1$

- General fixed-point problems

$$f^{(k)}(x, x, \ldots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : E^m \mapsto E, \quad m \in \mathbb{N}^*$$

$$m = 1, \quad f^{(k)} \equiv f, \quad \forall k$$

[Miellou, 1975] (sufficient)

$$|f(x) - f(y)| \leq T|x - y|$$

$T \geq O, \ \ \rho(T) < 1, \ \ |x| = (|x_1|, \ldots, |x_p|)$

[El Tarazi, 1982] (sufficient)

$$\|f(x) - f(y)\|_\infty^w \leq \alpha\|x - y\|_\infty^w$$

$w > 0, \ \ \alpha < 1, \ \ \|x\|_\infty^w = \max_i |x_i|/w_i$

[Bertsekas, 1983] (sufficient)

$$f(S^{(t)}) \subset S^{(t+1)} \subset S^{(t)}$$

$S^{(t)} = S_1^{(t)} \times \cdots \times S_p^{(t)}, \quad \lim_{t\to\infty} S^{(t)} = \{x^*\}$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

[Chazan and Miranker, 1969] (necessary and sufficient) : $\rho(|M^{-1}N|) < 1$

- General fixed-point problems

$$f^{(k)}(x, x, \dots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : E^m \mapsto E, \quad m \in \mathbb{N}^*$$

$$m = 1, \quad f^{(k)} \equiv f, \quad \forall k$$

$$m = 1$$

[Miellou, 1975] (sufficient)

$$|f(x) - f(y)| \leq T|x - y|$$

$$T \geq O, \quad \rho(T) < 1, \quad |x| = (|x_1|, \dots, |x_p|)$$

[El Tarazi, 1982] (sufficient)

$$\|f(x) - f(y)\|_\infty^w \leq \alpha \|x - y\|_\infty^w$$

$$w > 0, \quad \alpha < 1, \quad \|x\|_\infty^w = \max_i |x_i|/w_i$$

[Bertsekas, 1983] (sufficient)

$$f(S^{(t)}) \subset S^{(t+1)} \subset S^{(t)}$$

$$S^{(t)} = S_1^{(t)} \times \cdots \times S_p^{(t)}, \quad \lim_{t \to \infty} S^{(t)} = \{x^*\}$$

[Frommer and Szyld, 1994] (sufficient)

$$\|f^{(k)}(x) - f^{(k)}(y)\|_\infty^w \leq \alpha \|x - y\|_\infty^w, \quad \forall k$$

$$w > 0, \quad \alpha < 1, \quad \|x\|_\infty^w = \max_i |x_i|/w_i$$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

[Chazan and Miranker, 1969] (necessary and sufficient) : $\rho(|M^{-1}N|) < 1$

- General fixed-point problems

$$f^{(k)}(x, x, \dots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : E^m \mapsto E, \quad m \in \mathbb{N}^*$$

$$m = 1, \quad f^{(k)} \equiv f, \quad \forall k$$

[Miellou, 1975] (sufficient)

$$|f(x) - f(y)| \leq T|x - y|$$

$$T \geq O, \quad \rho(T) < 1, \quad |x| = (|x_1|, \dots, |x_p|)$$

[El Tarazi, 1982] (sufficient)

$$\|f(x) - f(y)\|_\infty^w \leq \alpha \|x - y\|_\infty^w$$

$$w > 0, \quad \alpha < 1, \quad \|x\|_\infty^w = \max_i |x_i|/w_i$$

[Bertsekas, 1983] (sufficient)

$$f(S^{(t)}) \subset S^{(t+1)} \subset S^{(t)}$$

$$S^{(t)} = S_1^{(t)} \times \dots \times S_p^{(t)}, \quad \lim_{t \to \infty} S^{(t)} = \{x^*\}$$

$$m = 1$$

[Frommer and Szyld, 1994] (sufficient)

$$\|f^{(k)}(x) - f^{(k)}(y)\|_\infty^w \leq \alpha \|x - y\|_\infty^w, \quad \forall k$$

$$w > 0, \quad \alpha < 1, \quad \|x\|_\infty^w = \max_i |x_i|/w_i$$

[Frommer and Szyld, 2000] (sufficient)

$$f^{(k)}(S^{(t)}) \subset S^{(t+1)} \subset S^{(t)}, \quad \forall k$$

$$S^{(t)} = S_1^{(t)} \times \dots \times S_p^{(t)}, \quad \lim_{t \to \infty} S^{(t)} = \{x^*\}$$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

[Chazan and Miranker, 1969] (necessary and sufficient) : $\rho(|M^{-1}N|) < 1$

- General fixed-point problems

$$f^{(k)}(x, x, \dots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : E^m \mapsto E, \quad m \in \mathbb{N}^*$$

$m = 1, \quad f^{(k)} \equiv f, \quad \forall k$

$m \geq 1, \quad f^{(k)} \equiv f, \quad \forall k$

[Miellou, 1975] (sufficient) :
$|.|$-contraction

[El Tarazi, 1982] (sufficient) :
$\|.\|_\infty^w$-contraction

[Bertsekas, 1983] (sufficient) :
$\{S^{(t)}\}$-contraction

$$m = 1$$

[Frommer & Szyld, 1994] (sufficient) :
$\|.\|_\infty^w$-contraction, $\forall k$

[Frommer & Szyld, 2000] (sufficient) :

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

[Chazan and Miranker, 1969] (necessary and sufficient) : $\rho(|M^{-1}N|) < 1$

- General fixed-point problems

$$f^{(k)}(x, x, \ldots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : E^m \mapsto E, \quad m \in \mathbb{N}^*$$

$m = 1, \quad f^{(k)} \equiv f, \quad \forall k$

[Miellou, 1975] (sufficient) :
$|.|$-contraction

[El Tarazi, 1982] (sufficient) :
$\|.\|_\infty^w$-contraction

[Bertsekas, 1983] (sufficient) :
$\{S^{(t)}\}$-contraction

$m = 1$

[Frommer & Szyld, 1994] (sufficient) :
$\|.\|_\infty^w$-contraction, $\forall k$

[Frommer & Szyld, 2000] (sufficient) :

$m \geq 1, \quad f^{(k)} \equiv f, \quad \forall k$

$X := (x^{(1)}, \ldots, x^{(m)}), \; Y := (y^{(1)}, \ldots, y^{(m)})$

[Baudet, 1978] (sufficient)

$|f(X) - f(Y)| \leq T \max\{|x^{(1)} - y^{(1)}|, \ldots, |x^{(m)} - y$

$T \geq O, \; \rho(T) < 1, \; (\max\{|x|, |y|\})_i = \max\{|x_i|$

Asynchronous iterative methods

- Linear problems
  $Ax = b \iff M^{-1}Nx + M^{-1}b = x$

[Chazan and Miranker, 1969] (necessary and sufficient) : $\rho(|M^{-1}N|) < 1$

- General fixed-point problems

$$f^{(k)}(x, x, \ldots, x) = x, \quad \forall k \in \mathbb{N}, \quad f^{(k)} : E^m \mapsto E, \quad m \in \mathbb{N}^*$$

$$m = 1, \quad f^{(k)} \equiv f, \quad \forall k$$

[Miellou, 1975] (sufficient) :
$|.|$-contraction

[El Tarazi, 1982] (sufficient) :
$\|.\|_\infty^w$-contraction

[Bertsekas, 1983] (sufficient) :
$\{S^{(t)}\}$-contraction

$$m = 1$$

[Frommer & Szyld, 1994] (sufficient) :
$\|.\|_\infty^w$-contraction, $\forall k$

[Frommer & Szyld, 2000] (sufficient) :

$$m \geq 1, \quad f^{(k)} \equiv f, \quad \forall k$$

$$X := (x^{(1)}, \ldots, x^{(m)}), \;\; Y := (y^{(1)}, \ldots, y^{(m)})$$

[Baudet, 1978] (sufficient)

$$|f(X) - f(Y)| \leq T \max\{|x^{(1)} - y^{(1)}|, \ldots, |x^{(m)} - y$$

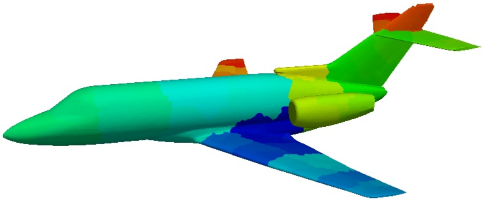
$$T \geq O, \;\; \rho(T) < 1, \;\; (\max\{|x|, |y|\})_i = \max\{|x_i|$$

[El Tarazi, 1982] (sufficient)

$$\|f(X) - x^*\|_\infty^w \leq \alpha \max\{\|x^{(l)} - x^*\|_\infty^w\}_{1 \leq l \leq m}$$

$$w > 0, \quad \alpha < 1, \quad \|x\|_\infty^w = \max_i |x_i| / w_i$$

The End



Asynchronous domain decomposition methods

Space domain decomposition - optimized Schwarz methods

Frédéric Magoulès
Univ. Paris Saclay, CentraleSupélec (France)

Outline

01 Synchronous optimized Schwarz domain decom-position

Extension to Helmholtz equation
Optimized Schwarz for Helmholtz equation
From a model problem to an industrial one
Engineering applications

Extension to the Helmholtz equation

- Schwarz algorithm
  - ▷ with overlap $\Rightarrow$ convergence for the high frequencies only
  - ▷ without overlap $\Rightarrow$ no convergence

- Introduction of new interface conditions

$$
\begin{array}{rcll}
(-\Delta - \omega^2)u_1^{n+1} & = & 0, & \text{in } \Omega_1 \\
(\partial_x + \mathcal{A}_1)u_1^{n+1}(L,y) & = & (\partial_x + \mathcal{A}_1)u_2^{n}(L,y) & \\
(-\Delta - \omega^2)u_2^{n} & = & 0, & \text{in } \Omega_2 \\
(\partial_x - \mathcal{A}_2)u_2^{n}(0,y) & = & (\partial_x - \mathcal{A}_2)u_1^{n-1}(0,y) &
\end{array}
$$

- How to define the "best" operators $\mathcal{A}_1$ and $\mathcal{A}_2$?
- How to define "easy to use" operators ?

Short bibliography

- Després (1991) : Helmholtz, interface conditions
- Charton, Nataf, Rogier (1991) : Convection diffusion, interface conditions
- Nataf, Rogier, de Sturler (1994) : Optimal interface conditions, one way splitting
- Benamou (1995) : Helmholtz, interface conditions
- Collino, Ghanemi, Joly (1998) : Maxwell, optimal operator
- Chevalier, Nataf (1998) : Helmholtz, optimized second order interface conditions
- Cai, Cassarin, Eliott, Widlund (1998) : Helmholtz, interface conditions
- Gander, Halpern, Nataf (1998) : Parabolic, optimized interface conditions

Short bibliography (cont.)

- Toselli (1999) : Helmholtz, Schwarz with overlap and PML
- Dolean, Lanteri (2001) : Euler Equation, optimized interface conditions
- Gander, Magoulès, Nataf (2001) : Helmholtz, optimized zeroth and second order interface conditions, asymptotic analysis
- Garbey, Tromeur-Dervout (2002) : Aitken-Schwarz method
- **Magoulès, Ivanyi, Topping (2004) : Helmholtz, engineering science**
- Maday, Magoulès (2005) : Optimized interface conditions for highly heterogeneous media
- . . .

Robin type interface conditions

With an operator of the form

$$\mathcal{A}_1\ u = (p + iq)\ u, \text{ and } \mathcal{A}_2\ u = (p + iq)\ u$$

Theorem (Gander, Magoulès, Nataf)

The optimal choice is

$$p^* = q^* = \sqrt{\frac{\sqrt{\omega^2 - \omega_-^2}\ \sqrt{k_{\max}^2 - \omega^2}}{2}},$$

and the asymptotic convergence rate upon h for $k_{max} = \pi/h$ *is*

$$\kappa(p, q, k) = 1 - 2\frac{\sqrt{2}(\omega^2 - \omega_-^2)^{1/4}}{\sqrt{\pi}}\sqrt{h} + O(h).$$

Unequal Robin type interface conditions

With an operator of the form

$$\mathcal{A}_1\ u = (p_1 + iq_1)\ u, \text{ and } \mathcal{A}_2\ u = (p_2 + iq_2)\ u$$

Theorem (Gander, Halpern, Magoulès)

The optimal choice is

$$p_1^* = q_1^* = \frac{1}{\sqrt{2}}\left((\omega^2 - \omega_-^2)(k_{\max}^2 - \omega^2)\right)^{\frac{3}{8}} \times \left(\sqrt{\omega^2 - \omega_-^2} + \sqrt{k_{\max}^2 - \omega^2} + \sqrt{k_{\max}^2 - \omega_-^2 + \sqrt{\omega^2 - \omega_-^2}\sqrt{k_{\max}^2 - \omega^2}}\right)^{-\frac{1}{2}},$$

$$p_2^* = q_2^* = \frac{1}{\sqrt{2}}\left((\omega^2 - \omega_-^2)(k_{\max}^2 - \omega^2)\right)^{\frac{1}{8}} \times \left(\sqrt{\omega^2 - \omega_-^2} + \sqrt{k_{\max}^2 - \omega^2} + \sqrt{k_{\max}^2 - \omega_-^2 + \sqrt{\omega^2 - \omega_-^2}\sqrt{k_{\max}^2 - \omega^2}}\right)^{\frac{1}{2}}$$

and the asymptotic convergence rate upon h for $k_{max} = C/h$ is

$$\kappa(p_1^*, q_1^*, p_2^*, q_2^*, k) = 1 - \frac{4\pi^{\frac{3}{4}}}{C}(\omega^2 - \omega_-^2)^{\frac{1}{4}} h^{\frac{1}{4}} + O(\sqrt{h}).$$

Interface conditions with tangential derivatives

With an operator of the form

$$\mathcal{A}_1 \; u = \alpha_1 \; u + \beta_1 \frac{\partial^2 u}{\partial \tau^2}, \text{ and } , \mathcal{A}_2 \; u = \alpha_2 \; u + \beta_2 \frac{\partial^2 u}{\partial \tau^2}$$

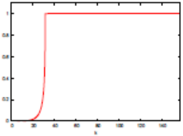
Theorem (Gander, Halpern, Magoulès)

The iterative algorithm with optimized second order interface conditions converges two times faster than with optimized zeroth order interface conditions. The optimal choice is

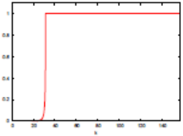
$$\alpha_1^* = \alpha_2^* = \frac{\alpha^* \beta^* - \omega^2}{\alpha^* + \beta^*}, \quad \beta_1^* = \beta_2^* = \frac{1}{\alpha^* + \beta^*}$$

where $\alpha^* = p_1^* + iq_1^*$ *and* $\beta^* = p_2^* + iq_2^*$ *are the optimized coefficients issued from the unequal Robin type interface conditions.*

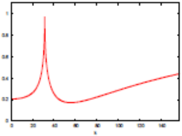
Comparison of some convergence rates



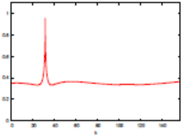
Taylor order 0



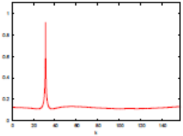
Taylor order 2



Optimized order 0



Optimized order 0 unequal



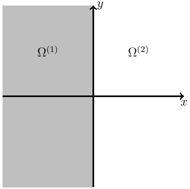
Optimized order 2

Remark

CPU time for one iteration is the same for all methods !

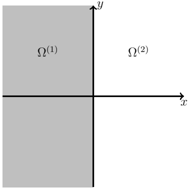
From a model problem to an industrial one

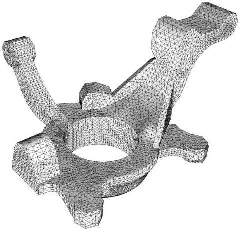
- Optimized interface conditions developed for
  - a two sub-domains splitting with a straight line interface
  - regular meshes



From a model problem to an industrial one

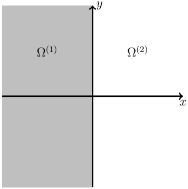
- Optimized interface conditions developed for
  - a two sub-domains splitting with a straight line interface
  - regular meshes
- Optimized interface conditions appear to be
  - extensible to arbitrary mesh partitioning
  - robust with regular and non-regular meshes
  - weakly dependent upon the shape of interfaces



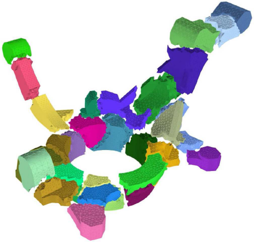


From a model problem to an industrial one

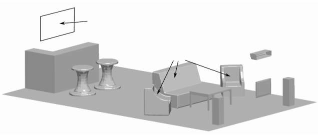
- Optimized interface conditions developed for
  - a two sub-domains splitting with a straight line interface
  - regular meshes

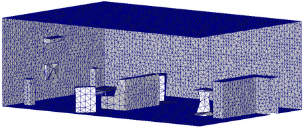


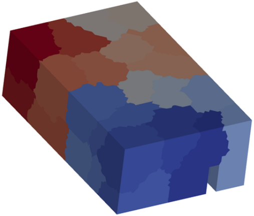
- Optimized interface conditions appear to be
  - extensible to arbitrary mesh partitioning
  - robust with regular and non-regular meshes
  - weakly dependent upon the shape of interfaces

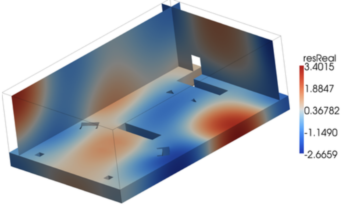


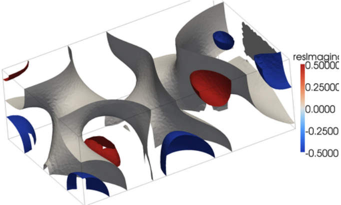
Architectural engineering - Appartment soundproofing





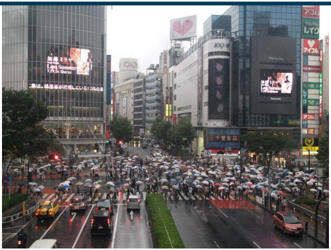




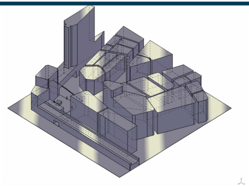


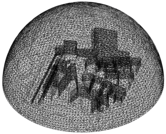
Optimized 0th (1022 iter.), Optimized 2nd (524 iter., 451 iter., 340 iter.)

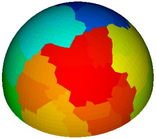
Environmental engineering - Noise pollution

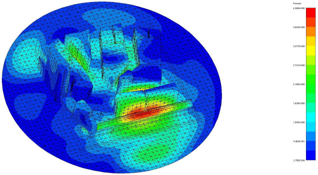


Environmental engineering - Noise pollution



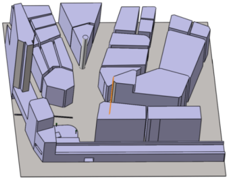


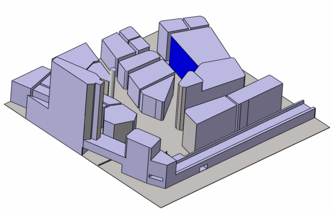


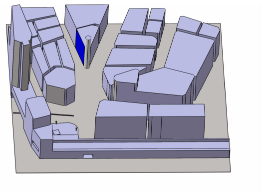


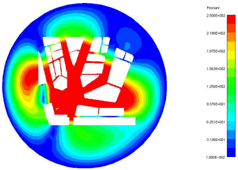
Taylor (3254 iter.), Optimized 0th (1656 iter.), Optimized 2nd (947 iter.)

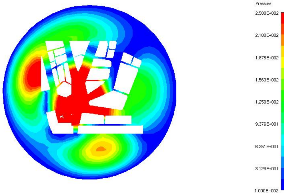
Environmental engineering - Noise pollution

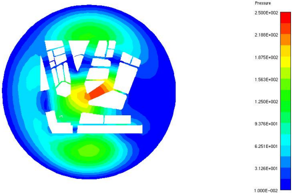




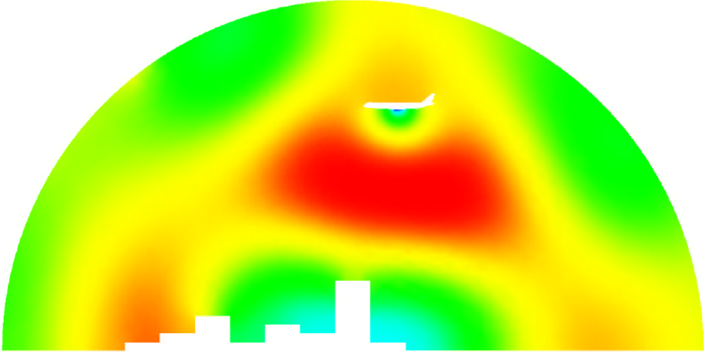






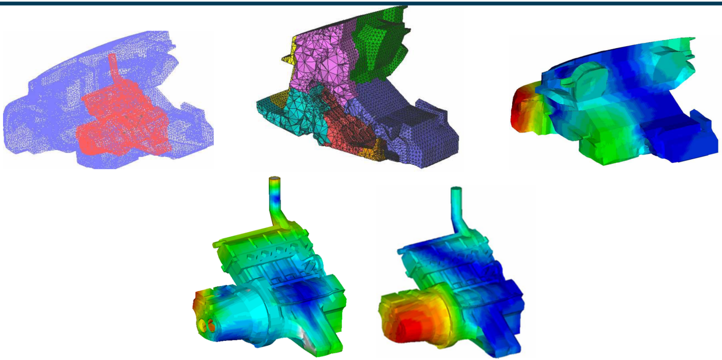


Aerospace engineering - Sound radiation from airplane



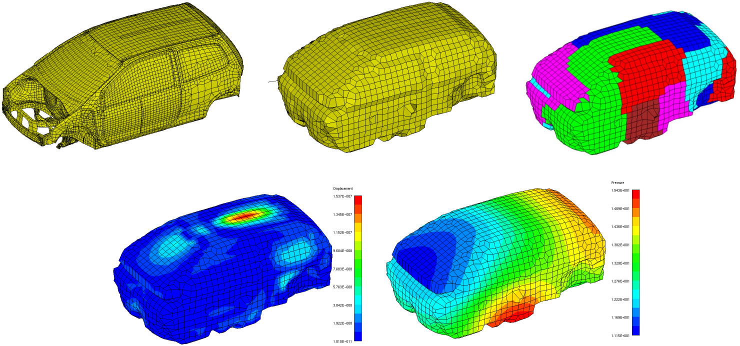
Optimized interface conditions reduces significantly the CPU time.
Taylor 0th (194 iter.), Optimized 0th (142 iter.), Optimized 2nd (72 iter.)

Automotive engineering - Engine compartment



Optimized interface conditions reduces significantly the CPU time. Taylor 0th (1069 iter.), Optimized 0th (531 iter.), Taylor 2nd (1105 iter.), Optimized 2nd (354 iter.)

Automotive engineering - Car compartment



Optimized interface conditions reduces significantly the CPU time.
Taylor (702 iter.), Optimized 0th (390 iter.), Optimized 2nd (162 iter.)

02 Asynchronous optimized Schwarz domain decomposition methods

Asynchronous optimized Schwarz method

Theorem (Magoulès, Szyld, Venet)

In the case of a one way splitting, the asynchronous iterative parallel Schwarz algorithm with optimal interface conditions converges with and without overlap.

Asynchronous optimized Schwarz method

Theorem (Magoulès, Szyld, Venet)

In the case of a one way splitting, the asynchronous iterative parallel Schwarz algorithm with optimal interface conditions converges with and without overlap.

# process	optimized Schwarz	# iterations	total time
4096	asynch	3465–3886	2345 sec.
4096	synch	3948	3198 sec.

The efficiency of the synchronous algorithm is rapidly decreasing with the number of process. Opposite the asynchronous version scales much more.

F. Magoulès, D.B. Szyld, and C. Venet. *Asynchronous optimized Schwarz methods with and without overlap.* Numerische Mathematik, 137(1) :199-227, 2017.

Asynchronous optimized Schwarz method

Theorem (El Haddad, Garay, Magoulès, Szyld)

For any positive value of the relative overlap, there exist a computable range of value for which the asynchronous optimized Schwarz method converges.

Asynchronous optimized Schwarz method

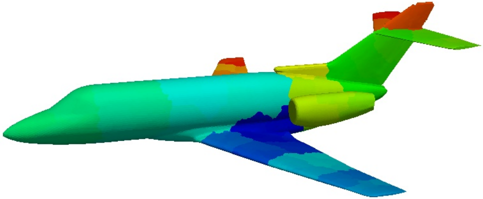
Theorem (El Haddad, Garay, Magoulès, Szyld)

For any positive value of the relative overlap, there exist a computable range of value for which the asynchronous optimized Schwarz method converges.

# process	optimized Schwarz	# iterations	total time
16	asynch	151–224	1.73 sec.
16	synch	109	2.79 sec.
25	asynch	261–497	1.10 sec.
25	synch	187	2.42 sec.

M El Haddad, F Garay, JC, Magoules, DB Szyld. *Synchronous and asynchronous optimized Schwarz methods for one-way subdivision of bounded domains. Numerical Linear Algebra with Applications 27(2), 2020.*

The End



Asynchronous domain decomposition methods

Space domain decomposition - Przemieniecki

Frédéric Magoulès
Univ. Paris Saclay, CentraleSupélec (France)

Outline

Substructuring domain decomposition method

J.S. Przemieniecki (1963)

Asynchronous substructuring domain decomposition method

Asynchronous substructuring method

01 Substructuring domain decomposition method

J.S. Przemieniecki (1963)

On the early history of substructuring

J.S. Przemieniecki (1963) Matrix structural analysis of substructures.
Am. Inst. Aero. Astro. J., 1 :138-147, 1963.

138 AIAA JOURNAL VOL. 1, NO. 1

Matrix Structural Analysis of Substructures

J. S. PRZEMIENIECKI*
Air Force Institute of Technology

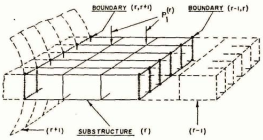


Fig. 1. Typical substructure with fixed boundaries.

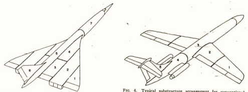


Fig. 3. Typical substructure arrangement for delta aircraft.

Fig. 4. Typical substructure arrangement for conventional aircraft.

"In the present method each substructure is first analyzed separately, assuming that all common boundaries with adjacent substructures are completely fixed : these boundaries are then relaxed simultaneously and the actual boundary displacements are determined from the equations of equilibrium of forces at the boundary joints. The substructures are then analyzed separately again under the action of specified external loading and the previously determined boundary displacements."

Motivation and explanation

- For Kron : direct Gaussian elimination has superlinear complexity
  - union of subproblems and the connecting problem (each also superlinear) could be solved in fewer overall operations than one large problem
- For Przemieniecki : full airplane structural analysis would not fit in memory of available computers
  - individual subproblems fit in memory

Motivation and explanation

- Let problem size be N, number of subdomains be P, and memory capacity be M
- Let problem solution complexity be $N^a, (a > 1)$
- Then subproblem solution complexity is $(N/P)^a$
- Let the cost of connecting the subproblems be $c(N, P)$

- Kron wins

$$\text{if} \quad P * (N/P)^a + c(N, P) < N^a$$

$$\text{or} \quad c(N, P) < N^a(1 - P^{1-a})$$

But Kron does not win directly if $a = 1$!

- Przemieniecki wins if

$$(N/P) < M$$

Przemieniecki's prediction

"From past experiences with the analysis of aircraft structures, it is evident that some form of structural partitioning is usually necessary, either because different methods of analysis are used on different structural components or because of the limitations imposed by digital computers. Even when the next generation of faster and larger digital computers becomes a well-established tool for the analysis of aircraft structures, it seems rather doubtful, because of the large number of unknowns, that the substructure displacement method of analysis would be wholly superseded by an overall analysis carried out on the complete structure."

J.S. Przemieniecki (1963)

More in the Domain Decomposition Methods Lectures of O. Widlund, D. Keyes, . . .

02 Asynchronous substructuring domain decomposition method

Asynchronous substructuring method

Asynchronous substructuring method

Theorem (Magoulès, Venet)

The asynchronous substructuring method converges.

Asynchronous substructuring method

Theorem (Magoulès, Venet)

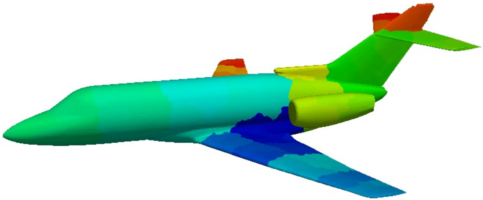
The asynchronous substructuring method converges.

# process	sub-structuring	# iterations	total time
1024	asynch..	122945-147245	656 sec.
1024	synch.	128024	834 sec.

The efficiency of the synchronous algorithm is rapidly decreasing with the number of process. Opposite the asynchronous version scales much more.

F. Magoulès and C. Venet. *Asynchronous iterative sub-structuring methods.* Mathematics and Computers in Simulation, 145 :34-49, 2018.

The End



Asynchronous domain decomposition methods

Time domain decomposition

Guillaume Gbikpi-Benissan, Frédéric Magoulès, Qinmeng Zou
Univ. Paris Saclay, CentraleSupélec (France)

Outline

Short bibliography

Asynchronous time-parallel methods

- Mitra (1987) : Asynchronous relaxations for the numerical solution of differential equations by parallel processors
- Bahi, Griepentrog, Miellou (1996) : Parallel treatment of a class of differential-algebraic systems $\rightarrow$ Asynchronous waveform relaxation method
- Bahi (1996) : Asynchronous Runge-Kutta methods for differential-algebraic systems
- S. Martin (1999) : Parallele asynchrone Waveform-Relaxation für Anfangswertaufgaben (Master's thesis, University of Wuppertal)
- Magoulès, Gbikpi-Benissan (2018) : Asynchronous parareal time discretization for partial differential equations
- Magoulès, Zou (2020) : Asynchronous time-parallel method based on Laplace transform

Outline

01 Waveform relaxation $\varnothing$

02 Parareal algorithm

Short bibliography

Pareal algorithm
[Lions et al., 2001], [Bal and Maday, 2002]

$$\begin{aligned} \frac{\partial u}{\partial t} + Au &= f, \quad t \in [0, T] \\ u(t=0) &= u_0 \end{aligned}$$

- Consists of a two-level time discretization
  [Lions et al, 2001]
- Based on multiple shooting and multi-grid approaches in parallel-in-time discretization
  [Chartier and Philippe, 1993] ; [Horton and Vandewalle, 1995]
- Convergence in a finite number of iterations (at iteration k, the k-th time sub-domain solution is 'exact')
  [Farhat, 2003]
- Generalized to a larger class of coarse solvers (not necessarily based on coarse time grid)

Synchronous Parareal time discretization

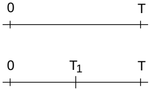
Partial differential equation

$$\delta(u(s,t), s, t) = 0, \quad t \in [0, T], \quad s \in \Omega$$
$$u(\Omega, 0) \text{ given}$$

Synchronous Parareal time discretization

Partial differential equation

$$\delta(u(s,t),s,t) = 0, \quad t \in [0,T], \quad s \in \Omega$$
$$u(\Omega,0) \text{ given}$$



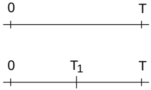
Two time intervals

$$\delta(u_0(s,t),s,t) = 0, \quad t \in [0, T_1] \qquad\qquad \delta(u_1(s,t),s,t) = 0, \quad t \in [T_1, T]$$
$$u_0(\Omega,0) = u(\Omega,0) \qquad\qquad u_1(\Omega, T_1) = u_0(\Omega, T_1)$$

Synchronous Parareal time discretization

Partial differential equation

$$\delta(u(s,t),s,t) = 0, \quad t \in [0,T], \quad s \in \Omega$$
$$u(\Omega,0) \text{ given}$$



Two time intervals

$$\delta(u_0(s,t),s,t) = 0, \quad t \in [0,T_1] \qquad \delta(u_1(s,t),s,t) = 0, \quad t \in [T_1,T]$$
$$u_0(\Omega,0) = u(\Omega,0) \qquad u_1(\Omega,T_1) = u_0(\Omega,T_1)$$

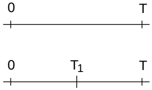
Parallel solutions

$$\widetilde{u}_0 \equiv u_0 \qquad \delta(\widetilde{u}_1(s,t),s,t) = 0, \quad t \in [T_1,T]$$
$$\lambda_0(\Omega) = u_0(\Omega,0) \qquad \widetilde{u}_1(\Omega,T_1) = \lambda_1(\Omega)$$

Synchronous Parareal time discretization

Partial differential equation

$$\delta(u(s,t),s,t) = 0, \quad t \in [0, T], \quad s \in \Omega$$
$$u(\Omega, 0) \text{ given}$$



Two time intervals

$$\delta(u_0(s,t),s,t) = 0, \quad t \in [0, T_1] \qquad \delta(u_1(s,t),s,t) = 0, \quad t \in [T_1, T]$$
$$u_0(\Omega, 0) = u(\Omega, 0) \qquad u_1(\Omega, T_1) = u_0(\Omega, T_1)$$

Parallel solutions

$$\widetilde{u}_0 \equiv u_0 \qquad \delta(\widetilde{u}_1(s,t),s,t) = 0, \quad t \in [T_1, T]$$
$$\lambda_0(\Omega) = u_0(\Omega, 0) \qquad \widetilde{u}_1(\Omega, T_1) = \lambda_1(\Omega)$$

Consistency across interface T_1

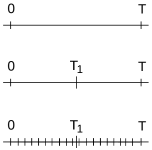
$$\lambda_1(\Omega) = \widetilde{u}_0(\Omega, T_1)$$

Synchronous Parareal time discretization

Two time intervals

$$\delta(u_0(s,t),s,t)=0, \quad t\in[0,T_1] \qquad \delta(u_1(s,t),s,t)=0, \quad t\in[T_1,T]$$
$$u_0(\Omega,0)=u(\Omega,0) \qquad u_1(\Omega,T_1)=u_0(\Omega,T_1)$$

Parallel solutions



$$\widetilde{u}_0\equiv u_0 \qquad \delta(\widetilde{u}_1(s,t),s,t)=0, \quad t\in[T_1,T]$$
$$\lambda_0(\Omega)=u_0(\Omega,0) \qquad \widetilde{u}_1(\Omega,T_1)=\lambda_1(\Omega)$$

Consistency across interface T_1

$$\lambda_1(\Omega)=\widetilde{u}_0(\Omega,T_1)$$

Fine time discretization

$$\alpha(\widetilde{u}_0(s,t_{n+1}))=\beta(\widetilde{u}_0(s,t_n)) \qquad \alpha(\widetilde{u}_1(s,t_{n+1}))=\beta(\widetilde{u}_1(s,t_n))$$

$\Rightarrow$ parallel fine propagator

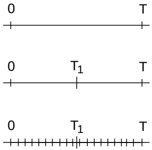
$$F(\lambda_0):=\widetilde{u}_0(\Omega,T_1) \qquad F(\lambda_1):=\widetilde{u}_1(\Omega,T)$$

Synchronous Parareal time discretization

Two time intervals

$$\delta(u_0(s,t),s,t) = 0, \quad t \in [0, T_1] \qquad \delta(u_1(s,t),s,t) = 0, \quad t \in [T_1, T]$$
$$u_0(\Omega,0) = u(\Omega,0) \qquad u_1(\Omega,T_1) = u_0(\Omega,T_1)$$

Parallel solutions



$$\widetilde{u}_0 \equiv u_0 \qquad \delta(\widetilde{u}_1(s,t),s,t) = 0, \quad t \in [T_1, T]$$
$$\lambda_0(\Omega) = u_0(\Omega,0) \qquad \widetilde{u}_1(\Omega,T_1) = \lambda_1(\Omega)$$

Consistency across interface T_1

$$\lambda_1(\Omega) = \widetilde{u}_0(\Omega, T_1)$$

Fine time discretization

$$\alpha(\widetilde{u}_0(s,t_{n+1})) = \beta(\widetilde{u}_0(s,t_n)) \qquad \alpha(\widetilde{u}_1(s,t_{n+1})) = \beta(\widetilde{u}_1(s,t_n))$$

$\Rightarrow$ parallel fine propagator

$$F(\lambda_0) := \widetilde{u}_0(\Omega, T_1) \qquad F(\lambda_1) := \widetilde{u}_1(\Omega, T)$$

Coarse time discretization

$$\alpha(u(s,T_1)) = \beta(u(s,0)), \quad \alpha(u(s,T)) = \beta(u(s,T_1))$$

$\Rightarrow$ serial coarse propagator

$$G(u(\Omega,0)) := \widetilde{u}(\Omega,T_1), \quad G(\widetilde{u}(\Omega,T_1)) = \widetilde{u}(\Omega,T)$$

Synchronous Parareal time discretization

Parallel solutions

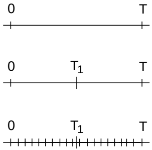
$$\delta(\widetilde{u}_0(s,t),s,t)=0,\quad t\in[0,T_1] \qquad\qquad \delta(\widetilde{u}_1(s,t),s,t)=0,\quad t\in[T_1,T]$$

$$\lambda_0(\Omega)\text{ given} \qquad\qquad \widetilde{u}_1(\Omega,T_1)=\lambda_1(\Omega)$$

Consistency across interface T_1

$$\lambda_1(\Omega)=\widetilde{u}_0(\Omega,T_1)$$

Parareal iterations



$$F(\lambda_0):=\widetilde{u}_0(\Omega,T_1)$$

$$F(\lambda_1)=\widetilde{u}_1(\Omega,T)$$

$$G(u(\Omega,0)):=\widetilde{u}(\Omega,T_1)$$

$$G(\widetilde{u}(\Omega,T_1))=\widetilde{u}(\Omega,T)$$

Synchronous Parareal time discretization

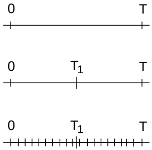
Parallel solutions

$$\delta(\widetilde{u}_0(s,t), s, t) = 0, \quad t \in [0, T_1] \qquad \delta(\widetilde{u}_1(s,t), s, t) = 0, \quad t \in [T_1, T]$$

$$\lambda_0(\Omega) \text{ given} \qquad \widetilde{u}_1(\Omega, T_1) = \lambda_1(\Omega)$$

Consistency across interface T_1

$$\lambda_1(\Omega) = \widetilde{u}_0(\Omega, T_1)$$



Parareal iterations

$$\lambda_0^0 := u_0(\Omega, 0)$$

$$\lambda_1^0 := G(\lambda_0^0) \quad \text{wait} \quad [0, T_1]$$

$$F(\lambda_0) := \widetilde{u}_0(\Omega, T_1)$$

$$F(\lambda_1) = \widetilde{u}_1(\Omega, T)$$

$$G(u(\Omega, 0)) := \widetilde{u}(\Omega, T_1)$$

$$G(\widetilde{u}(\Omega, T_1)) = \widetilde{u}(\Omega, T)$$

Prediction

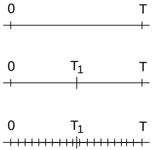
Synchronous Parareal time discretization

Parallel solutions

$$\delta(\widetilde{u}_0(s,t),s,t)=0,\quad t\in[0,T_1] \qquad \delta(\widetilde{u}_1(s,t),s,t)=0,\quad t\in[T_1,T]$$

$$\lambda_0(\Omega)\ \text{given} \qquad \widetilde{u}_1(\Omega,T_1)=\lambda_1(\Omega)$$

Consistency across interface T_1



$$\lambda_1(\Omega)=\widetilde{u}_0(\Omega,T_1)$$

Parareal iterations

$$\lambda_0^0 := u_0(\Omega,0)$$

$$\lambda_1^0 := G(\lambda_0^0) \qquad \text{wait} \qquad [0,T_1]$$

$$\lambda_2^0 := G(\lambda_1^0) \qquad [T_1,T]$$

$$F(\lambda_0) := \widetilde{u}_0(\Omega,T_1)$$

$$F(\lambda_1) = \widetilde{u}_1(\Omega,T)$$

$$G(u(\Omega,0)) := \widetilde{u}(\Omega,T_1)$$

$$G(\widetilde{u}(\Omega,T_1)) = \widetilde{u}(\Omega,T)$$

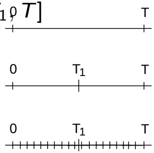
Prediction

Synchronous Parareal time discretization

Parallel solutions

$$\delta(\widetilde{u}_0(s,t),s,t) = 0, \quad t \in [0, T_1] \qquad \delta(\widetilde{u}_1(s,t),s,t) = 0, \quad t \in [T_1, T]$$

$$\lambda_0(\Omega) \text{ given} \qquad \widetilde{u}_1(\Omega, T_1) = \lambda_1(\Omega)$$



Consistency across interface T_1

$$\lambda_1(\Omega) = \widetilde{u}_0(\Omega, T_1)$$

Parareal iterations

$$\lambda_0^0 := u_0(\Omega, 0)$$

$$\lambda_1^0 := G(\lambda_0^0) \qquad \text{wait} \qquad [0, T_1]$$

$$\lambda_2^0 := G(\lambda_1^0) \qquad [T_1, T]$$

$$F(\lambda_0^0) \qquad F(\lambda_1^0) \qquad [0, T_1]\ [T_1, T]$$

$$F(\lambda_0) := \widetilde{u}_0(\Omega, T_1)$$

$$F(\lambda_1) = \widetilde{u}_1(\Omega, T)$$

$$G(u(\Omega, 0)) := \widetilde{u}(\Omega, T_1)$$

$$G(\widetilde{u}(\Omega, T_1)) = \widetilde{u}(\Omega, T)$$

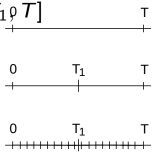
Prediction

Synchronous Parareal time discretization

Parallel solutions

$$\delta(\widetilde{u}_0(s,t),s,t) = 0, \quad t \in [0, T_1] \qquad \delta(\widetilde{u}_1(s,t),s,t) = 0, \quad t \in [T_1, T]$$

$$\lambda_0(\Omega) \text{ given} \qquad \widetilde{u}_1(\Omega, T_1) = \lambda_1(\Omega)$$



Consistency across interface T_1

$$\lambda_1(\Omega) = \widetilde{u}_0(\Omega, T_1)$$

Parareal iterations

$$\lambda_0^0 := u_0(\Omega, 0)$$

$$\lambda_1^0 := G(\lambda_0^0) \qquad \text{wait} \qquad [0, T_1]$$

$$\lambda_2^0 := G(\lambda_1^0) \qquad [T_1, T]$$

$$F(\lambda_0^0) \qquad F(\lambda_1^0) \qquad [0, T_1]\ [T_1, T]$$

$$\lambda_0^1 := \lambda_0^0$$

$$\lambda_1^1 := G(\lambda_0^1) + F(\lambda_0^0) - G(\lambda_0^0) \qquad \text{wait} \qquad [0, T_1]$$

$$F(\lambda_0) := \widetilde{u}_0(\Omega, T_1)$$

$$F(\lambda_1) = \widetilde{u}_1(\Omega, T)$$

$$G(u(\Omega, 0)) := \widetilde{u}(\Omega, T_1)$$

$$G(\widetilde{u}(\Omega, T_1)) = \widetilde{u}(\Omega, T)$$

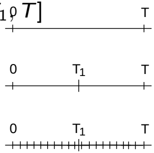
Prediction
Gap

Synchronous Parareal time discretization

Parallel solutions

$$\delta(\widetilde{u}_0(s,t), s, t) = 0, \quad t \in [0, T_1] \qquad \delta(\widetilde{u}_1(s,t), s, t) = 0, \quad t \in [T_1, T]$$

$$\lambda_0(\Omega) \text{ given} \qquad \widetilde{u}_1(\Omega, T_1) = \lambda_1(\Omega)$$



Consistency across interface T_1

$$\lambda_1(\Omega) = \widetilde{u}_0(\Omega, T_1)$$

Parareal iterations

$$\lambda_0^0 := u_0(\Omega, 0) \qquad F(\lambda_0) := \widetilde{u}_0(\Omega, T_1)$$

$$\lambda_1^0 := G(\lambda_0^0) \qquad \text{wait} \qquad [0, T_1] \qquad F(\lambda_1) = \widetilde{u}_1(\Omega, T)$$

$$\lambda_2^0 := G(\lambda_1^0) \qquad [T_1, T]$$

$$F(\lambda_0^0) \qquad F(\lambda_1^0) \qquad [0, T_1]\ [T_1, T] \qquad G(u(\Omega, 0)) := \widetilde{u}(\Omega, T_1)$$

$$\lambda_0^1 := \lambda_0^0 \qquad G(\widetilde{u}(\Omega, T_1)) = \widetilde{u}(\Omega, T)$$

$$\lambda_1^1 := G(\lambda_0^1) + F(\lambda_0^0) - G(\lambda_0^0) \qquad \text{wait} \qquad [0, T_1]$$

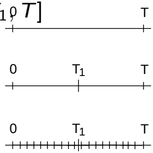
Prediction
Gap
Corrected prediction

Synchronous Parareal time discretization

Parallel solutions

$$\delta(\widetilde{u}_0(s,t),s,t)=0, \quad t\in[0,T_1] \qquad \delta(\widetilde{u}_1(s,t),s,t)=0, \quad t\in[T_1,T]$$

$$\lambda_0(\Omega)\ \text{given} \qquad \widetilde{u}_1(\Omega,T_1)=\lambda_1(\Omega)$$



Consistency across interface T_1

$$\lambda_1(\Omega)=\widetilde{u}_0(\Omega,T_1)$$

Parareal iterations

$$\lambda_0^0 := u_0(\Omega,0)$$

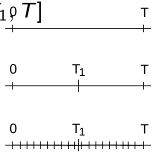
$\lambda_1^0 := G(\lambda_0^0)$	wait	$[0, T_1]$
	$\lambda_2^0 := G(\lambda_1^0)$	$[T_1, T]$
$F(\lambda_0^0)$	$F(\lambda_1^0)$	$[0, T_1]\ [T_1, T]$
$\lambda_0^1 := \lambda_0^0$		
$\lambda_1^1 := G(\lambda_0^1)+F(\lambda_0^0)-G(\lambda_0^0)$	wait	$[0, T_1]$
	$\lambda_2^1 := G(\lambda_1^1)+F(\lambda_1^0)-G(\lambda_1^0)$	$[T_1, T]$

Synchronous Parareal time discretization

Parallel solutions

$$\delta(\widetilde{u}_0(s,t),s,t)=0,\quad t\in[0,T_1] \qquad \delta(\widetilde{u}_1(s,t),s,t)=0,\quad t\in[T_1,T]$$

$$\lambda_0(\Omega)\text{ given} \qquad \widetilde{u}_1(\Omega,T_1)=\lambda_1(\Omega)$$



Consistency across interface T_1

$$\lambda_1(\Omega)=\widetilde{u}_0(\Omega,T_1)$$

Parareal iterations

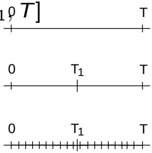
$\lambda_0^0 := u_0(\Omega,0)$		
$\color{red}{\lambda_1^0} := G(\lambda_0^0)$	$\color{red}{\text{wait}}$	$[0, \color{red}{T_1}]$
	$\lambda_2^0 := G(\color{red}{\lambda_1^0})$	$[\color{red}{T_1}, T]$
$F(\lambda_0^0)$	$F(\lambda_1^0)$	$[0, T_1]$ $[T_1, T]$
$\lambda_0^1 := \lambda_0^0$		
$\color{red}{\lambda_1^1} := G(\lambda_0^1)+F(\lambda_0^0)-G(\lambda_0^0)$	$\color{red}{\text{wait}}$	$[0, \color{red}{T_1}]$
	$\lambda_2^1 := G(\color{red}{\lambda_1^1})+F(\lambda_1^0)-G(\lambda_1^0)$	$[\color{red}{T_1}, T]$
$\lambda_0^2 := \lambda_0^1$	$F(\lambda_1^1)$	$[T_1, T]$

Synchronous Parareal time discretization

Parallel solutions

$$\delta(\widetilde{u}_0(s,t), s, t) = 0, \quad t \in [0, T_1] \qquad \delta(\widetilde{u}_1(s,t), s, t) = 0, \quad t \in [T_1, T]$$

$$\lambda_0(\Omega) \text{ given} \qquad \widetilde{u}_1(\Omega, T_1) = \lambda_1(\Omega)$$



Consistency across interface T_1

$$\lambda_1(\Omega) = \widetilde{u}_0(\Omega, T_1)$$

Parareal iterations

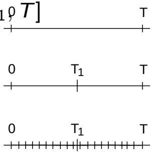
$\lambda_0^0 := u_0(\Omega, 0)$		
$\lambda_1^0 := G(\lambda_0^0)$	wait	$[0, T_1]$
	$\lambda_2^0 := G(\lambda_1^0)$	$[T_1, T]$
$F(\lambda_0^0)$	$F(\lambda_1^0)$	$[0, T_1]$ $[T_1, T]$
$\lambda_0^1 := \lambda_0^0$		
$\lambda_1^1 := G(\lambda_0^1) + F(\lambda_0^0) - G(\lambda_0^0)$	wait	$[0, T_1]$
	$\lambda_2^1 := G(\lambda_1^1) + F(\lambda_1^0) - G(\lambda_1^0)$	$[T_1, T]$
$\lambda_0^2 := \lambda_0^1$	$F(\lambda_1^1)$	$[T_1, T]$
$\lambda_1^2 := \lambda_1^1$	$\lambda_2^2 := G(\lambda_1^2) + F(\lambda_1^1) - G(\lambda_1^1)$	$[T_1, T]$

Synchronous Parareal time discretization

Parallel solutions

$$\delta(\widetilde{u}_0(s,t), s, t) = 0, \quad t \in [0, T_1] \qquad \delta(\widetilde{u}_1(s,t), s, t) = 0, \quad t \in [T_1, T]$$

$$\lambda_0(\Omega) \text{ given} \qquad \widetilde{u}_1(\Omega, T_1) = \lambda_1(\Omega)$$



Consistency across interface T_1

$$\lambda_1(\Omega) = \widetilde{u}_0(\Omega, T_1)$$

Parareal iterations

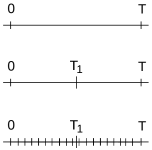
$$\begin{array}{ccc}
\lambda_0^0 := u_0(\Omega, 0) & & \\
\lambda_1^0 := G(\lambda_0^0) & \text{wait} & [0, T_1] \\
 & \lambda_2^0 := G(\lambda_1^0) & [T_1, T] \\
F(\lambda_0^0) & F(\lambda_1^0) & [0, T_1]\ [T_1, T] \\
\lambda_0^1 := \lambda_0^0 & & \\
\lambda_1^1 := G(\lambda_0^1) + F(\lambda_0^0) - G(\lambda_0^0) & \text{wait} & [0, T_1] \\
 & \lambda_2^1 := G(\lambda_1^1) + F(\lambda_1^0) - G(\lambda_1^0) & [T_1, T] \\
\lambda_0^2 := \lambda_0^1 & F(\lambda_1^1) & [T_1, T] \\
\lambda_1^2 := \lambda_1^1 & \lambda_2^2 := G(\lambda_1^2) + F(\lambda_1^1) - G(\lambda_1^1) & [T_1, T]
\end{array}$$

$$\lambda_i^{k+1} = G(\lambda_{i-1}^{k+1}) + F(\lambda_{i-1}^k) - G(\lambda_{i-1}^k), \qquad [T_{i-1}, T_i]$$

Synchronous Parareal time discretization

Parareal iterations

$$\lambda_i^{k+1} = G(\lambda_{i-1}^{k+1}) + F(\lambda_{i-1}^{k}) - G(\lambda_{i-1}^{k}), \qquad [T_{i-1}, T_i]$$



$$\delta(\widetilde{u}_i(s,t), s, t) = 0$$

$$\widetilde{u}_i(\Omega, T_i) = \lambda_i(\Omega)$$

$$F(\lambda_i) = \widetilde{u}_i(\Omega, T_{i+1})$$

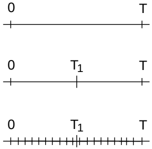
$$G(\widetilde{u}(\Omega, T_i)) = \widetilde{u}(\Omega, T_{i+1})$$

Prediction G
Gap $F - G$
Corrected prediction
$G+$

Synchronous Parareal time discretization

Parareal iterations

$$\lambda_i^{k+1} = G(\lambda_{i-1}^{k+1}) + F(\lambda_{i-1}^{k}) - G(\lambda_{i-1}^{k}), \qquad [T_{i-1}, T_i]$$



Theorem

PDE extension of ODE result

[Gander & Vandewalle, 2007]

$$\delta(\widetilde{u}_i(s,t), s, t) = 0$$

$$\widetilde{u}_i(\Omega, T_i) = \lambda_i(\Omega)$$

Iterative error

$$\|\lambda^k - \lambda^*\|_\infty \le \alpha^k \|\lambda^0 - \lambda^*\|_\infty$$

$$\alpha = \frac{1-\theta^N}{1-\theta}\|F - G\|, \ \ \theta \neq 1, \ \theta \ge \|G\|$$

$$F(\lambda_i) = \widetilde{u}_i(\Omega, T_{i+1})$$

$$G(\widetilde{u}(\Omega, T_i)) = \widetilde{u}(\Omega, T_{i+1})$$

Prediction G

Gap $F - G$

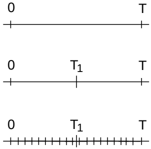
Corrected prediction

$G+$

Synchronous Parareal time discretization

Parareal iterations

$$\lambda_i^{k+1} = G(\lambda_{i-1}^{k+1}) + F(\lambda_{i-1}^{k}) - G(\lambda_{i-1}^{k}), \qquad [T_{i-1}, T_i]$$



Theorem

PDE extension of ODE result [Gander & Vandewalle, 2007]

$$\delta(\widetilde{u}_i(s,t), s, t) = 0$$

$$\widetilde{u}_i(\Omega, T_i) = \lambda_i(\Omega)$$

Iterative error

$$\|\lambda^k - \lambda^*\|_\infty \leq \alpha^k \|\lambda^0 - \lambda^*\|_\infty$$

$$\alpha = \frac{1-\theta^N}{1-\theta}\|F-G\|, \;\; \theta \neq 1, \; \theta \geq \|G\|$$

$$F(\lambda_i) = \widetilde{u}_i(\Omega, T_{i+1})$$

$$G(\widetilde{u}(\Omega, T_i)) = \widetilde{u}(\Omega, T_{i+1})$$

Sufficient convergence condition with $\|G\| < 1$ (i.e., stability region)

$$\|G\| + \|F-G\| < 1 + \|G\|^N \|F-G\|$$

Prediction G

Gap $F - G$

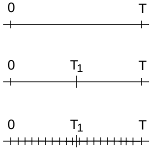
Corrected prediction $G+$

Asynchronous Parareal time discretization

Parareal iterations

$$\lambda_i^{k+1} = G(\lambda_{i-1}^{k+1}) + F(\lambda_{i-1}^{k}) - G(\lambda_{i-1}^{k}), \qquad [T_{i-1}, T_i]$$

$$\lambda_i^{k+1} = G\left(\lambda_{i-1}^{\tau_{i-1}^{i,1}(k)}\right) + F\left(\lambda_{i-1}^{\tau_{i-1}^{i,2}(k)}\right) - G\left(\lambda_{i-1}^{\tau_{i-1}^{i,2}(k)}\right)$$



Theorem
PDE extension of ODE result
[Gander & Vandewalle, 2007]

Iterative error

$$\|\lambda^k - \lambda^*\|_\infty \leq \alpha^k \|\lambda^0 - \lambda^*\|_\infty$$

$$\alpha = \frac{1-\theta^N}{1-\theta}\|F-G\|, \;\; \theta \neq 1, \; \theta \geq \|G\|$$

Sufficient convergence condition with $\|G\| < 1$ (i.e., stability region)

$$\|G\| + \|F-G\| < 1 + \|G\|^N \|F-G\|$$

Theorem (Benissan)

$$\delta(\widetilde{u}_i(s,t), s, t) = 0$$

$$\widetilde{u}_i(\Omega, T_i) = \lambda_i(\Omega)$$

$$F(\lambda_i) = \widetilde{u}_i(\Omega, T_{i+1})$$

$$G(\widetilde{u}(\Omega, T_i)) = \widetilde{u}(\Omega, T_{i+1})$$

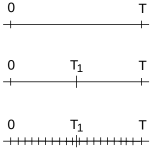
Prediction G
Gap $F - G$
Corrected prediction
$G+$

Asynchronous Parareal time discretization

Parareal iterations

$$\lambda_i^{k+1} = G(\lambda_{i-1}^{k+1}) + F(\lambda_{i-1}^{k}) - G(\lambda_{i-1}^{k}), \qquad [T_{i-1}, T_i]$$

$$\lambda_i^{k+1} = G\left(\lambda_{i-1}^{\tau_{i-1}^{i,1}(k)}\right) + F\left(\lambda_{i-1}^{\tau_{i-1}^{i,2}(k)}\right) - G\left(\lambda_{i-1}^{\tau_{i-1}^{i,2}(k)}\right)$$



Theorem
PDE extension of ODE result
[Gander & Vandewalle, 2007]

Iterative error

$$\|\lambda^k - \lambda^*\|_\infty \leq \alpha^k \|\lambda^0 - \lambda^*\|_\infty$$

$$\alpha = \frac{1-\theta^N}{1-\theta}\|F-G\|, \ \ \theta \neq 1, \ \theta \geq \|G\|$$

Sufficient convergence condition with $\|G\| < 1$ (i.e., stability region)

$$\|G\| + \|F-G\| < 1 + \|G\|^N \|F-G\|$$

Theorem
Asynchronous Iterative error

$$\|\lambda^k - \lambda^*\|_\infty \leq \widetilde{\alpha}^{\tau(k)} \|\lambda^0 - \lambda^*\|_\infty$$

$$\widetilde{\alpha} = \|G\| + \|F-G\|, \quad \lim_{k\to\infty} \tau(k) = \infty$$

$$\delta(\widetilde{u}_i(s,t), s, t) = 0$$

$$\widetilde{u}_i(\Omega, T_i) = \lambda_i(\Omega)$$

$$F(\lambda_i) = \widetilde{u}_i(\Omega, T_{i+1})$$

$$G(\widetilde{u}(\Omega, T_i)) = \widetilde{u}(\Omega, T_{i+1})$$

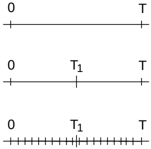
Prediction G
Gap $F-G$
Corrected prediction
$G+$

Asynchronous Parareal time discretization

Parareal iterations

$$\lambda_i^{k+1} = G(\lambda_{i-1}^{k+1}) + F(\lambda_{i-1}^{k}) - G(\lambda_{i-1}^{k}), \qquad [T_{i-1}, T_i]$$

$$\lambda_i^{k+1} = G\left(\lambda_{i-1}^{\tau_{i-1}^{i,1}(k)}\right) + F\left(\lambda_{i-1}^{\tau_{i-1}^{i,2}(k)}\right) - G\left(\lambda_{i-1}^{\tau_{i-1}^{i,2}(k)}\right)$$



Theorem
PDE extension of ODE result
[Gander & Vandewalle, 2007]

Iterative error

$$\|\lambda^k - \lambda^*\|_\infty \leq \alpha^k \|\lambda^0 - \lambda^*\|_\infty$$

$$\alpha = \frac{1-\theta^N}{1-\theta}\|F-G\|, \;\; \theta \neq 1, \; \theta \geq \|G\|$$

Sufficient convergence condition with $\|G\| < 1$ (i.e., stability region)

$$\|G\| + \|F-G\| < 1 + \|G\|^N \|F-G\|$$

Theorem
Asynchronous Iterative error

$$\|\lambda^k - \lambda^*\|_\infty \leq \widetilde{\alpha}^{\tau(k)} \|\lambda^0 - \lambda^*\|_\infty$$

$$\widetilde{\alpha} = \|G\| + \|F-G\|, \quad \lim_{k\to\infty} \tau(k) = \infty$$

Asynchronous convergence (sufficient)

$$\|G\| + \|F-G\| < 1$$

$$\delta(\widetilde{u}_i(s,t), s, t) = 0$$

$$\widetilde{u}_i(\Omega, T_i) = \lambda_i(\Omega)$$

$$F(\lambda_i) = \widetilde{u}_i(\Omega, T_{i+1})$$

$$G(\widetilde{u}(\Omega, T_i)) = \widetilde{u}(\Omega, T_{i+1})$$

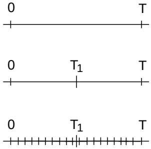
Prediction G
Gap $F - G$
Corrected prediction
$G+$

Asynchronous Parareal time discretization

Parareal iterations

$$\lambda_i^{k+1} = G(\lambda_{i-1}^{k+1}) + F(\lambda_{i-1}^{k}) - G(\lambda_{i-1}^{k}), \qquad [T_{i-1}, T_i]$$

$$\lambda_i^{k+1} = G\left(\lambda_{i-1}^{\tau_{i-1}^{i,1}(k)}\right) + F\left(\lambda_{i-1}^{\tau_{i-1}^{i,2}(k)}\right) - G\left(\lambda_{i-1}^{\tau_{i-1}^{i,2}(k)}\right)$$



Theorem
PDE extension of ODE result
[Gander & Vandewalle, 2007]

Iterative error

$$\|\lambda^k - \lambda^*\|_\infty \leq \alpha^k \|\lambda^0 - \lambda^*\|_\infty$$

$$\alpha = \frac{1-\theta^N}{1-\theta}\|F-G\|, \;\; \theta \neq 1, \; \theta \geq \|G\|$$

Sufficient convergence condition with $\|G\| < 1$ (i.e., stability region)

$$\|G\| + \|F-G\| < 1 + \|G\|^N \|F-G\|$$

Theorem
Asynchronous Iterative error

$$\|\lambda^k - \lambda^*\|_\infty \leq \widetilde{\alpha}^{\tau(k)} \|\lambda^0 - \lambda^*\|_\infty$$

$$\widetilde{\alpha} = \|G\| + \|F-G\|, \quad \lim_{k\to\infty} \tau(k) = \infty$$

Asynchronous convergence (sufficient)

$$\|G\| + \|F - G\| < 1$$

Asymptotic sync. convergence (left) implies async. convergence (top)

$$\|G\| + \|F - G\| < 1$$

$$\delta(\widetilde{u}_i(s,t), s, t) = 0$$

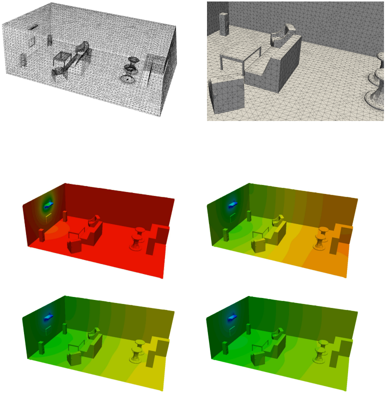
$$\widetilde{u}_i(\Omega, T_i) = \lambda_i(\Omega)$$

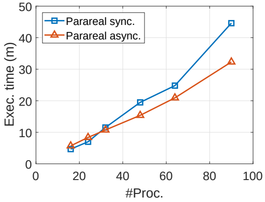
$$F(\lambda_i) = \widetilde{u}_i(\Omega, T_{i+1})$$

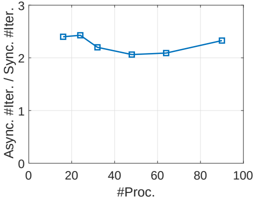
$$G(\widetilde{u}(\Omega, T_i)) = \widetilde{u}(\Omega, T_{i+1})$$

Prediction G
Gap $F - G$
Corrected prediction
$G+$

Time domain decomposition







$$t_{n+1} - t_n = 0.002, \qquad T_{i+1} - T_i = 0.2$$

$$N = \#\text{Proc}, \qquad T = N \times (T_{i+1} - T_i)$$

Want to know more ?

- Asynchronous Parareal time discretization for partial differential equations
  F. Magoulès, G. Gbikpi-Benissan
  SIAM Journal on Scientific Computing 40 (6), C704-C725, 2018
  + proof of the asynch. convergence of Parareal
  + application to the heat equation
- F. Magoulès, G. Gbikpi-Benissan, Q. Zou
  Asynchronous iterations of Parareal algorithm for option pricing models
  Mathematics 6 (4), 45, 2018
  + application to option pricing

03 Laplace transform

Laplace transform

Time-dependent problem

$$\frac{\partial u}{\partial t} + g(x,t) = K(u)\frac{\partial^2 u}{\partial x^2}.$$

Laplace transform :

$$\lambda_i U - u(x, T_j) + G = K(\bar{u})\frac{d^2 U}{dx^2}.$$

Inverse Laplace transform :

$$u(x, T_{j+1}) \approx \frac{\ln 2}{\Delta T}\sum_{i=1}^{p} w_i U(\lambda_i; x), \quad \lambda_i = \frac{i \ln 2}{\Delta T},$$

where

$$w_i = (-1)^{\frac{p}{2}+i}\sum_{k=\frac{1+i}{2}}^{\min(i,\frac{p}{2})}\frac{k^{\frac{p}{2}}(2k)!}{(\frac{p}{2}-k)!k!(k-1)!(i-k)!(2k-i)!}.$$

Short bibliography

- Number of processors p should be even
  Original Laplace transform $\rightarrow$ numerical method
  [Stehfest, 1970]

- Original method : $u(t) = \frac{1}{2\pi i} \int_\Gamma e^{zt} w(z) dz$, $w(z) = \int_0^\infty e^{-zt} u(t) dt$
  Deformation of integral contour, new quadrature schemes
  [Sheen, Sloan and Thomée, 1999]

- Direct variant : Stehfest's algorithm, convection-diffusion equation
  [Crann, Davies, Lai and Leong, 1998]

- Iterative method : nonlinear equation
  [Lai, Parrott, Rout and Honnor, 2005]

Asynchronous Laplace algorithm

Laplace operator :

$$\mathcal{L}_i\left(u^k\right) = U(z_i), \quad i \in \{1, \ldots, p\}, \quad k \in \mathbb{N}.$$

Gaver-Stehfest operator :

$$u_i^{k+1} = \mathcal{G}_i(U) = \frac{\ln 2}{t}\omega_i U(z_i), \quad i \in \{1, \ldots, p\}, \quad k \in \mathbb{N}.$$

Summation operator :

$$u^k = \mathcal{S}\left(u_1^k, \ldots, u_p^k\right) = \sum_{i=1}^{p} u_i^k, \quad k \in \mathbb{N}.$$

Final form :

$$u_i^{k+1} = \left(\mathcal{L}_i \circ \mathcal{G}_i \circ \mathcal{S}\right)\left(u_1^k, \ldots, u_p^k\right), \quad i \in \{1, \ldots, p\}, \quad k \in \mathbb{N}.$$

Asynchronous Laplace algorithm

Piecewise operator :

$$f_i = \mathcal{L}_i \circ \mathcal{G}_i \circ \mathcal{S}, \quad i \in \{1, \dots, p\}.$$

Piecewise iterations :

$$u_i^{k+1} = f_i\left(u_1^k, \dots, u_p^k\right), \quad i \in \{1, \dots, p\}, \quad k \in \mathbb{N}.$$

Asynchronous form :

$$u_i^{k+1} = \begin{cases} f_i\left(u_1^{\tau_{i,1}(k)}, \dots, u_p^{\tau_{i,p}(k)}\right), & i \in P^{(k)}, \\ u_i^k, & i \notin P^{(k)}. \end{cases}$$

Option pricing problem

Black-Scholes PDE :

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} = rV,$$

- The most influential model
  "Nobel Memorial Prize in Economic Sciences" in 1997
  [Black and Scholes, 1973]
- Initial and boundary conditions for European call option
  - $V(S,T) = \max(S - E, 0)$
  - $V(0,t) = 0$
  - $V(S,t) \sim S - Ee^{-r(T-t)}$ *as* $S \to +\infty$
- V : predictive option price ; S : stoke price ; E : strike price
  T : time to maturity ; r : risk-free interest rate ; σ : volatility

Experimental results

Table – Convergence zone of synchronous Laplace transform method

T	convergence interval of p	remarks
0.01	6, 8, 10, 12	$p < 6$: inaccurate ; $p > 12$: divergent
0.1	4, 6, 8, 10, 12	$p < 4$: inaccurate ; $p > 12$: divergent
1	4, 6, 8, 10, 12	$p < 4$: inaccurate ; $p > 12$: divergent

Table – Convergence zone of asynchronous Laplace transform method

T	convergence interval of p	remarks
0.01	6	$p < 6$: inaccurate ; $p > 6$: divergent
0.1	6	$p < 6$: inaccurate ; $p > 6$: divergent
1	4, 6	$p < 4$: inaccurate ; $p > 6$: divergent

Experimental results

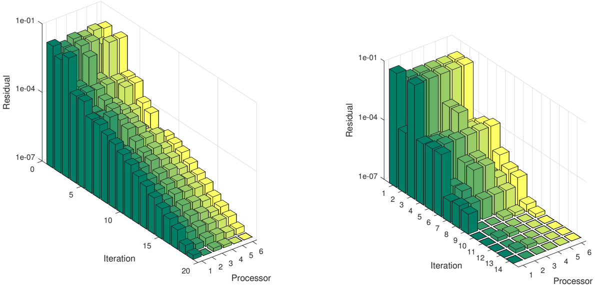
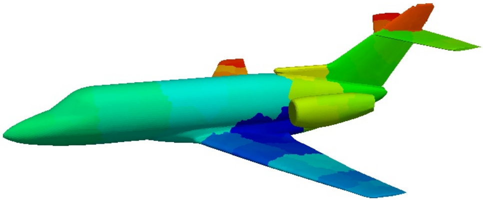


Figure – An example of successive asynchronous Laplace iteration

Want to know more ?

- F. Magoulès, Q. Zou
  Asynchronous time-parallel method based on Laplace transform
  International Journal of Computer Mathematics, 1-16, 2020
  \- limitation similar to the synchronous case
  \+ application to option pricing

The End



Asynchronous domain decomposition methods

30 years of asynchronous convergence detection.

Guillaume Gbikpi-Benissan, Frédéric Magoulès

Univ. Paris Saclay, CentraleSupélec (France)

Outline

01 Asynchronous convergence detection

Setting of the problem

Set of admissible solutions $S^{(*)}$
Residual error evaluation consists of

$$r(x) < \varepsilon \implies x \in S^{(*)}$$

Distributed evaluation consists of

$$r(x) = \sigma(r_1(x), \ldots, r_p(x))$$

with σ a reduction operator

Example with Euclidean norm

$$r(x) = \|x - f(x)\|$$

$$r_i(x) = \|x_i - f_i(x)\|^2$$

$$\sigma(\alpha_1, \ldots, \alpha_p) = \left(\sum_{i=1}^{p} \alpha_i \right)^{1/2}$$

$$r(x) = \sigma(r_1(x), \ldots, r_p(x)) = \left(\sum_{i=1}^{p} \|x_i - f_i(x)\|^2 \right)^{1/2}$$

Setting of the problem

Asynchronous iterative model

$$x_i^{k+1} = f_i(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}), \quad \forall i \in P^{(k)}$$

Implicit sequence $\{x^k\}_{k\in\mathbb{N}}$
Explicit sequences $\{x_1^{k^{(1)}}\}_{k^{(1)}\in\mathbb{N}}$ to $\{x_p^{k^{(p)}}\}_{k^{(p)}\in\mathbb{N}}$
Convergence detection

$$r(\bar{x}) < \varepsilon, \quad \bar{x} = \begin{bmatrix} x_1^{k_1} & \ldots & x_p^{k_p} \end{bmatrix}^\mathsf{T}$$

In asynchronous iterative model, how to define $\bar{x}$?

Synchronous iterative model

$$x_i^{k^{(i)}} = x_i^k \iff k^{(i)} = k$$

$$\bar{x} = \begin{bmatrix} x_1^k & \ldots & x_p^k \end{bmatrix}^\mathsf{T} = x^k$$

$$r(\bar{x}) = \sigma(r_1(x^k), \ldots, r_p(x^k))$$

In synchronous iterative model, $\bar{x}$ is defined as x^k !

Distributed approaches

- Modification of the iterative procedure to ensure *finite-time termination* [Bertsekas and Tsitsiklis, 1989], [El Baz, 1996], [Savari and Bertsekas, 1996] → exact, mathematical assumptions, intrusive
- Predictive approximation of the number of iterations required to reach convergence [Evans and Chikohora, 1998] → protocol-free, heuristic
- Monitoring of consistency and persistence of local convergence [Bahi *et al.*, 2005, 2008] → heuristic, intrusive
- Evaluation of diameter of solutions nested sets $S^{(*)} \subset \cdots \subset S^{(k+1)} \subset S^{(k)} \subset \cdots S^{(0)}$ by means of "macro-iterations" [Miellou *et al.*, 2008] → exact, intrusive
- Explicit evaluation of $r(\bar{x})$ from global state snapshot [Savari and Bertsekas, 1996], [Magoulès and Benissan, IEEE, 2018] → exact, control message size in $\mathcal{O}(n)$
- Explicit evaluation of an upper bound of $r(\bar{x})$ from global state snapshot [Magoulès and Benissan, IEEE, 2018] → exact, control message size in $\mathcal{O}(1)$, assumption on known comm. delays
- Explicit evaluation of an upper bound of $r(\bar{x})$ without snapshot [Benissan and Magoulès, AES, 2020] exact, → protocol-free, assumption on unknown comm. delays

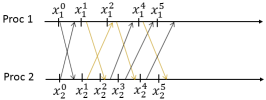
Distributed approaches

Problem

$$f(x) = x, \quad x \in E$$

Asynchronous iterations

$$x_i^{k+1} = f_i(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}), \quad i \in P^k$$



Convergence detection

$$\bar{x} := \left(x_1^{k_1}, \ldots, x_p^{k_p}\right) \simeq x^*$$

Approach

- Finite time termination *[Bertsekas and Tsitsiklis, 1989], [El Baz, 1996], [Savari and Bertsekas, 1996], ...*

Can be formulated as :
(i) processor i is disabled when $\|x_i^{k_i} - x_i^{k_i-1}\| < \epsilon \Rightarrow$ no-'send', but still 'recv'
(ii) when all processors are disabled and no message in transit $\Rightarrow$ termination

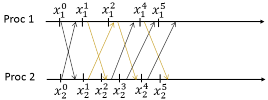
- non-centralized algorithm
- difficult implementation, because activated/disabled status change during (ii)
- no guarantee for residual to be equal to a threshold

Distributed approaches

Problem

$$f(x) = x, \quad x \in E$$

Asynchronous iterations



$$x_i^{k+1} = f_i(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}), \quad i \in P^k$$

Convergence detection

$$\bar{x} := \left(x_1^{k_1}, \ldots, x_p^{k_p}\right) \simeq x^*$$

Approach

- Predicted termination (finite number of iterations) *[Evans and Chikohora, 1998]*

Can be formulated as :
(i) predict the expected number of iterations of iterative scheme

- not often used in the literature

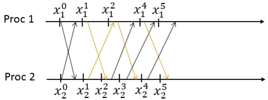
Distributed approaches

Problem

$$f(x) = x, \quad x \in E$$

Asynchronous iterations

$$x_i^{k+1} = f_i(x_1^{\tau_1^i(k)}, \dots, x_p^{\tau_p^i(k)}), \quad i \in P^k$$



Convergence detection

$$\bar{x} := \left(x_1^{k_1}, \dots, x_p^{k_p}\right) \simeq x^*$$

Approach

- Local convergence monitoring *[Bahi et al, 2005, 2008]*

Can be formulated with leader-election *[Bahi et al, 2005]*
(i) during the time when P_i sends a message of local convergence to the 'leader', it can diverge $\Rightarrow$ a cancellation-message must be send
(ii) a maximum transmission time on the network is supposed
(iii) once $P_j, \forall j$ has send a convergence message, if during this time no cancellation message has been send $\Rightarrow$ global convergence

- no guarantee for residual to be equal to a threshold
- no saving of local converged x_i $\Rightarrow$ the last computed one is taken

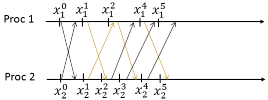
Distributed approaches

Problem

$$f(x) = x, \quad x \in E$$

Asynchronous iterations

$$x_i^{k+1} = f_i(x_1^{\tau_1^i(k)}, \dots, x_p^{\tau_p^i(k)}), \quad i \in P^k$$



Convergence detection

$$\bar{x} := \left(x_1^{k_1}, \dots, x_p^{k_p}\right) \simeq x^*$$

Approach

- Nested-sets-based supervised termination *[Miellou et al, 2008]*

Can be formulated as
(i) definition of macro-iteration, i.e., when $P_j, \forall j$ has done one iteration with at least one update from its neighbors
(ii) compute $\|x_j^k - x_j^{k-1}\|$, with k = macro-iteration $\Leftrightarrow$ embeded sets V_{k-1} and V_k
(iii) if $\|x_j^k - x_j^{k-1}\| < \epsilon \Rightarrow$ (if synchronous convergence $\Rightarrow$ asynchronous convergence)

- only for L_∞-norm

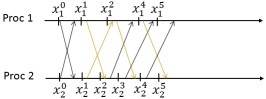
+ guarantee for residual to be lower than a given threshold

Distributed approaches

Problem

$$f(x) = x, \quad x \in E$$

Asynchronous iterations



$$x_i^{k+1} = f_i(x_1^{\tau_1^i(k)}, \dots, x_p^{\tau_p^i(k)}), \quad i \in P^k$$

Convergence detection

$$\bar{x} := \left(x_1^{k_1}, \dots, x_p^{k_p} \right) \simeq x^*$$

Approach

- Snapshot-based supervised termination *[Savari and Bertsekas, 1996]*

Can be re-formulated [Magoulès and Benissan, IEEE, 2018] with 'snapshot' as :
(i) when processor i satisfies $\|x_i^{k_i} - x_i^{k_i - 1}\| < \epsilon \Rightarrow$ processor i saves $\bar{x}_i$, then sends it to all its neighbors
(ii) when $P_j, j \neq i$ receives the 'snapshot' message, it does the same
(iii) at the end, all $P_j, \forall j$ has $\bar{x}_i$ and $\bar{x}_j, \forall j \neq i$
(iv) $P_j, \forall j$ computes residual $r(\bar{x}) = \|A\bar{x} - b\|$

$+$ guarantee for residual to be equal to a threshold

Distributed approaches

Asynchronous convergence detection approaches

- Finite time termination *[Bertsekas and Tsitsiklis, 1989], [El Baz, 1996], [Savari and Bertsekas, 1996], . . .*
- Predicted termination (finite number of iterations) *[Evans and Chikohora, 1998]*
- Local convergence monitoring *[Bahi et al, 2005, 2008]*
- Nested-sets-based supervised termination *[Miellou et al, 2008]*
- Snapshot-based supervised termination *[Savari and Bertsekas, 1996]*

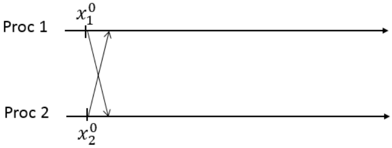
	Intrusiveness	Centralization	Effectiveness	Messages size
Bertsekas & Tsitsiklis, 1989	altered iterations	–	reliable	–
El Baz, 1996	altered iterations	–	reliable	–
Savari & Bertsekas, 1996	altered iterations	–	reliable	–
Evans & Chikohora, 1998	non-intrusive	no reduction	heuristic	0
Bahi *et al*, 2005	non-intrusive	one reduction	heuristic	$\mathcal{O}(1)$
Bahi *et al*, 2008	piggybacking	two reductions	heuristic	$\mathcal{O}(1)$
Miellou *et al*, 2008	–	–	reliable	–
Savari & Bertsekas, 1996	non-intrusive	two reductions	exact	$\mathcal{O}(n)$

Distributed snapshot

Problem

$$f(x) = x, \quad x \in E$$

Distributed snapshot *[Chandy and Lamport, 1985]*



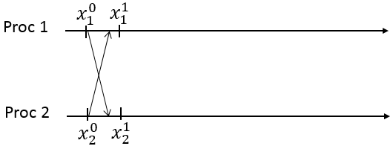
- Initiator / First *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
  3. Start recording neighbors' messages
- On *marker* reception
  1. Stop recording corresponding neighbor's messages
- On computation message reception
  1. Record message

Distributed snapshot

Problem

$$f(x) = x, \quad x \in E$$

Distributed snapshot *[Chandy and Lamport, 1985]*



$x_1^1 := f_1(x_1^0, x_2^0)$ $\quad$ $x_2^1 := f_2(x_1^0, x_2^0)$

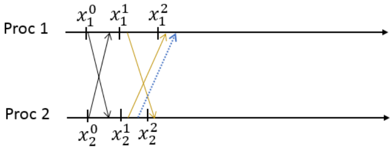
- Initiator / First *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
  3. Start recording neighbors' messages
- On *marker* reception
  1. Stop recording corresponding neighbor's messages
- On computation message reception
  1. Record message

Distributed snapshot

Problem

$$f(x) = x, \quad x \in E$$

Distributed snapshot *[Chandy and Lamport, 1985]*



$$x_1^1 := f_1(x_1^0, x_2^0) \qquad x_2^1 \qquad x_2^1 := f_2(x_1^0, x_2^0)$$

$$x_1^2 := f_1(x_1^1, x_2^0) \qquad x_2^2 := f_2(x_1^0, x_2^1)$$

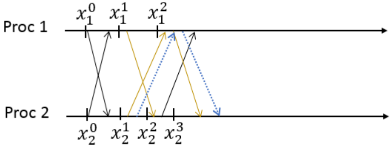
- Initiator / First *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
  3. Start recording neighbors' messages
- On *marker* reception
  1. Stop recording corresponding neighbor's messages
- On computation message reception
  1. Record message

Distributed snapshot

Problem

$$f(x) = x, \quad x \in E$$

Distributed snapshot *[Chandy and Lamport, 1985]*



- Initiator / First *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
  3. Start recording neighbors' messages
- On *marker* reception
  1. Stop recording corresponding neighbor's messages
- On computation message reception
  1. Record message

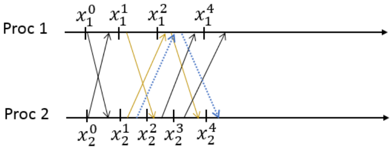
$$\begin{array}{lll} & x_1^1 := f_1(x_1^0, x_2^0) \\ & x_1^2 := f_1(x_1^1, x_2^0) \\ x_1^2 & x_1^3 := x_1^2 \end{array} \qquad \begin{array}{lll} x_2^1 & x_2^1 := f_2(x_1^0, x_2^0) & \\ & x_2^2 := f_2(x_1^0, x_2^1) & \\ & x_2^3 := f_2(x_1^1, x_2^2) & x_1^1 \end{array}$$

Distributed snapshot

Problem

$$f(x) = x, \quad x \in E$$

Distributed snapshot *[Chandy and Lamport, 1985]*



$$\begin{array}{llcl}
 & x_1^1 := f_1(x_1^0, x_2^0) & x_2^1 & x_2^1 := f_2(x_1^0, x_2^0) & \\
 & x_1^2 := f_1(x_1^1, {\color{orange}x_2^0}) & & x_2^2 := f_2({\color{orange}x_1^0}, x_2^1) & \\
{\color{blue}x_1^2} & x_1^3 := x_1^2 & & x_2^3 := f_2({\color{orange}x_1^1}, x_2^2) & {\color{blue}x_1^1} \\
 & x_1^4 := f_1(x_1^3, x_2^2) & & x_2^4 := f_2({\color{orange}x_1^2}, x_2^3) & {\color{blue}x_1^2}
\end{array}$$

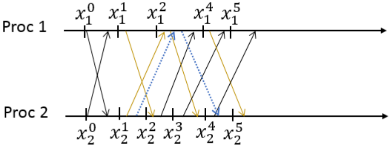
- Initiator / First *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
  3. Start recording neighbors' messages
- On *marker* reception
  1. Stop recording corresponding neighbor's messages
- On computation message reception
  1. Record message

Distributed snapshot

Problem

$$f(x) = x, \quad x \in E$$

Distributed snapshot *[Chandy and Lamport, 1985]*



	$x_1^1 := f_1(x_1^0, x_2^0)$	x_2^1	$x_2^1 := f_2(x_1^0, x_2^0)$	
	$x_1^2 := f_1(x_1^1, x_2^0)$		$x_2^2 := f_2(x_1^0, x_2^1)$	
x_1^2	$x_1^3 := x_1^2$		$x_2^3 := f_2(x_1^1, x_2^2)$	x_1^1
	$x_1^4 := f_1(x_1^3, x_2^2)$		$x_2^4 := f_2(x_1^2, x_2^3)$	x_1^2
	$x_1^5 := f_1(x_1^4, x_2^3)$		$x_2^5 := f_2(x_1^2, x_2^4)$	

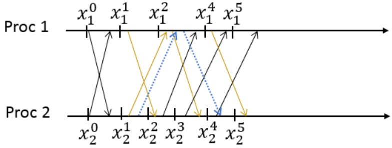
- Initiator / First *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
  3. Start recording neighbors' messages
- On *marker* reception
  1. Stop recording corresponding neighbor's messages
- On computation message reception
  1. Record message

Distributed snapshot

Problem

$$f(x) = x, \quad x \in E$$

Distributed snapshot *[Chandy and Lamport, 1985]*



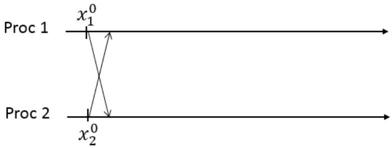
- Initiator / First *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
  3. Start recording neighbors' messages
- On *marker* reception
  1. Stop recording corresponding neighbor's messages
- On computation message reception
  1. Record message

$$\begin{array}{ll}
 & x_1^1 := f_1(x_1^0, x_2^0) \\
 & x_1^2 := f_1(x_1^1, x_2^0) \\
x_1^2 & x_1^3 := x_1^2 \\
 & x_1^4 := f_1(x_1^3, x_2^2) \\
 & x_1^5 := f_1(x_1^4, x_2^3) \\
 & (x_1^2, ?)
\end{array}$$

$$\begin{array}{lll}
x_2^1 & x_2^1 := f_2(x_1^0, x_2^0) & \\
 & x_2^2 := f_2(x_1^0, x_2^1) & \\
 & x_2^3 := f_2(x_1^1, x_2^2) & x_1^1 \\
 & x_2^4 := f_2(x_1^2, x_2^3) & x_1^2 \\
 & x_2^5 := f_2(x_1^2, x_2^4) & \\
 & (x_1^2, x_2^1) &
\end{array}$$

Snapshot based asynch. convergence detection

Asynchronous iterations snapshot (AIS) *[Magoulès and Benissan, IEEE, 2018, Proposition 1]*, based on general distributed snapshot from *[Chandy and Lamport, 1985]*

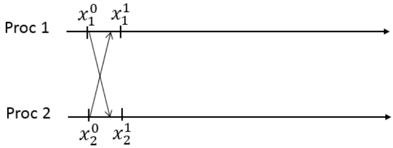


- On local convergence or first *marker* reception
    1. Record local state
    2. Send *marker* to neighbors
- On *marker* reception
    1. Record last corresponding neighbor's message

$\bar{x}^{(1)} := (?, ?)$ $\bar{x}^{(2)} := (?, ?)$

Snapshot based asynch. convergence detection

Asynchronous iterations snapshot (AIS) *[Magoulès and Benissan, IEEE, 2018, Proposition 1]*, based on general distributed snapshot from *[Chandy and Lamport, 1985]*



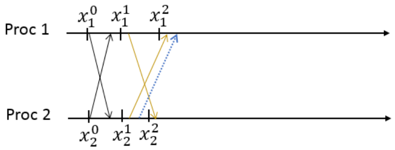
$x_1^1 := f_1(x_1^0, x_2^0)$ $\quad$ $x_2^1 := f_2(x_1^0, x_2^0)$

$\bar{x}^{(1)} := (?, ?)$ $\quad$ $\bar{x}^{(2)} := (?, ?)$

- On local convergence or first *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
- On *marker* reception
  1. Record last corresponding neighbor's message

Snapshot based asynch. convergence detection

Asynchronous iterations snapshot (AIS) *[Magoulès and Benissan, IEEE, 2018, Proposition 1]*, based on general distributed snapshot from *[Chandy and Lamport, 1985]*



$x_1^1 := f_1(x_1^0, x_2^0)$ $\quad$ x_2^1 $\quad$ $x_2^1 := f_2(x_1^0, x_2^0)$

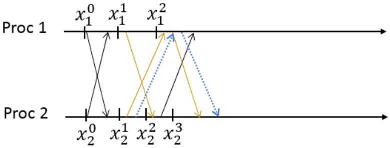
$x_1^2 := f_1(x_1^1, x_2^0)$ $\quad$ $x_2^2 := f_2(x_1^0, x_2^1)$

$\bar{x}^{(1)} := (?, ?)$ $\quad$ $\bar{x}^{(2)} := (?, x_2^1)$

- On local convergence or first *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
- On *marker* reception
  1. Record last corresponding neighbor's message

Snapshot based asynch. convergence detection

Asynchronous iterations snapshot (AIS) *[Magoulès and Benissan, IEEE, 2018, Proposition 1]*, based on general distributed snapshot from *[Chandy and Lamport, 1985]*



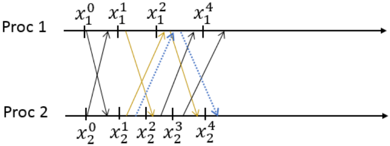
$$
\begin{array}{llll}
 & x_1^1 := f_1(x_1^0, x_2^0) & x_2^1 & x_2^1 := f_2(x_1^0, x_2^0) \\
 & x_1^2 := f_1(x_1^1, x_2^0) & & x_2^2 := f_2(x_1^0, x_2^1) \\
x_1^2 & x_1^3 := x_1^2 & x_2^1 & x_2^3 := f_2(x_1^1, x_2^2)
\end{array}
$$

$$\bar{x}^{(1)} := (x_1^2, x_2^1) \qquad \bar{x}^{(2)} := (?, x_2^1)$$

- On local convergence or first *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
- On *marker* reception
  1. Record last corresponding neighbor's message

Snapshot based asynch. convergence detection

Asynchronous iterations snapshot (AIS) *[Magoulès and Benissan, IEEE, 2018, Proposition 1]*, based on general distributed snapshot from *[Chandy and Lamport, 1985]*



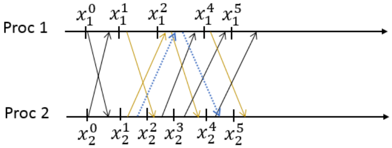
$$
\begin{array}{lll|lll}
 & x_1^1 := f_1(x_1^0, x_2^0) & & x_2^1 & x_2^1 := f_2(x_1^0, x_2^0) & \\
 & x_1^2 := f_1(x_1^1, x_2^0) & & & x_2^2 := f_2(x_1^0, x_2^1) & \\
x_1^2 & x_1^3 := x_1^2 & x_2^1 & & x_2^3 := f_2(x_1^1, x_2^2) & \\
 & x_1^4 := f_1(x_1^3, x_2^2) & & & x_2^4 := f_2(x_1^2, x_2^3) & x_1^2
\end{array}
$$

$$\bar{x}^{(1)} := (x_1^2, x_2^1) \qquad \bar{x}^{(2)} := (x_1^2, x_2^1)$$

- On local convergence or first *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
- On *marker* reception
  1. Record last corresponding neighbor's message

Snapshot based asynch. convergence detection

Asynchronous iterations snapshot (AIS) *[Magoulès and Benissan, IEEE, 2018, Proposition 1]*, based on general distributed snapshot from *[Chandy and Lamport, 1985]*



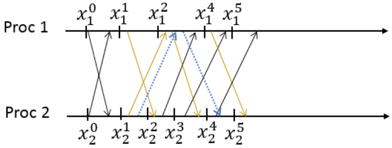
$$
\begin{array}{lll@{\qquad}lll}
 & x_1^1 := f_1(x_1^0, x_2^0) & x_2^1 & & x_2^1 := f_2(x_1^0, x_2^0) & \\
 & x_1^2 := f_1(x_1^1, x_2^0) & & & x_2^2 := f_2(x_1^0, x_2^1) & \\
x_1^2 & x_1^3 := x_1^2 & x_2^1 & & x_2^3 := f_2(x_1^1, x_2^2) & \\
 & x_1^4 := f_1(x_1^3, x_2^2) & & & x_2^4 := f_2(x_1^2, x_2^3) & x_1^2 \\
 & x_1^5 := f_1(x_1^4, x_2^3) & & & x_2^5 := f_2(x_1^2, x_2^4) &
\end{array}
$$

$$\bar{x}^{(1)} := (x_1^2, x_2^1) \qquad \bar{x}^{(2)} := (x_1^2, x_2^1)$$

- On local convergence or first *marker* reception
  1. Record local state
  2. Send *marker* to neighbors
- On *marker* reception
  1. Record last corresponding neighbor's message

Snapshot (non-FIFO) based asynch. convergence detection

Non first-in-first-out Asynchronous iterations snapshot (NFAIS) *[Magoulès and Benissan, IEEE, 2018, Proposition 2]*
⇒ marker received before computation message
⇒ inconsistent message recording



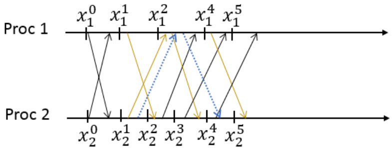
$$\begin{array}{lll|lll}
 & x_1^1 := f_1(x_1^0, x_2^0) & & x_2^1 & x_2^1 := f_2(x_1^0, x_2^0) & \\
 & x_1^2 := f_1(x_1^1, x_2^0) & & & x_2^2 := f_2(x_1^0, x_2^1) & \\
x_1^2 & x_1^3 := x_1^2 & x_2^? & & x_2^3 := f_2(x_1^1, x_2^2) & \\
 & x_1^4 := f_1(x_1^3, x_2^2) & & & x_2^4 := f_2(x_1^2, x_2^3) & x_1^? \\
 & x_1^5 := f_1(x_1^4, x_2^3) & & & x_2^5 := f_2(x_1^2, x_2^4) &
\end{array}$$

$$\bar{x}^{(1)} := (x_1^2, x_2^?) \qquad \bar{x}^{(2)} := (x_1^?, x_2^1)$$

- Embed computation message into marker (based on *[Savari and Bertsekas, 1996]*)
  ⇒ marker size : $\mathcal{O}(n)$
- Send marker after m successive local convergences
  ⇒ assumption for non FIFO characterization : marker crosses at most m computation messages
  ⇒ $\bar{x}^{(1)} \neq \bar{x}^{(2)}$
  ⇒ $\bar{x} = ?, \quad r(\bar{x})?$
- ~~Send flag-marker after m subsequent iterations ⇒ flag armed if continuous local~~ under the assumption that a marker can not be delivered slower than a computation message

Snapshot (non-FIFO) based asynch. convergence detection

Non first-in-first-out Asynchronous iterations snapshot (NFAIS) *[Magoulès and Benissan, IEEE, 2018, Proposition 2]*
$\Rightarrow$ marker received before computation message
$\Rightarrow$ inconsistent message recording



$$
\begin{array}{llll}
 & x_1^1 := f_1(x_1^0, x_2^0) & x_2^1 & x_2^1 := f_2(x_1^0, x_2^0) \\
 & x_1^2 := f_1(x_1^1, x_2^0) & & x_2^2 := f_2(x_1^0, x_2^1) \\
x_1^2 & x_1^3 := x_1^2 \qquad x_2^? & & x_2^3 := f_2(x_1^1, x_2^2) \\
 & x_1^4 := f_1(x_1^3, x_2^2) & & x_2^4 := f_2(x_1^2, x_2^3) \quad x_1^? \\
 & x_1^5 := f_1(x_1^4, x_2^3) & & x_2^5 := f_2(x_1^2, x_2^4)
\end{array}
$$

$$\bar{x}^{(1)} := (x_1^2, x_2^?) \qquad \bar{x}^{(2)} := (x_1^?, x_2^1)$$

- Embed computation message into marker (based on [Savari and Bertsekas, 1996])
  $\Rightarrow$ marker size : $\mathcal{O}(n)$
- Send marker after m successive local convergences
  $\Rightarrow$ assumption : marker crosses at most m computation messages
  $\Rightarrow \bar{x}^{(1)} \neq \bar{x}^{(2)}$
  $\Rightarrow \bar{x} = \begin{bmatrix} \bar{x}_1^{(1)} & \cdots & \bar{x}_p^{(p)} \end{bmatrix}^{\mathsf{T}}$
  $\Rightarrow \widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)})$

Snapshot (non-FIFO) based asynch. convergence detection

$$\bar{x} := \begin{bmatrix} \bar{x}_1^{(1)} & \cdots & \bar{x}_p^{(p)} \end{bmatrix}^\mathsf{T}$$

Exact residual $r(x) = \sigma(r_1(x), \ldots, r_p(x))$
Approximate residual $\widetilde{r}(x^{(1)}, \ldots, x^{(p)}) := \sigma(r_1(x^{(1)}), \ldots, r_p(x^{(p)}))$

$$\rightarrow \widetilde{r}(x, \ldots, x) = r(x)$$

Theorem : *[Magoulès and Benissan, IEEE, 2018]*
Global residual bounded by approximated residual as

$$r(\bar{x}) < \widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)}) + g(p, m, f)\varepsilon$$

$$\downarrow$$

$$\widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)}) < \varepsilon \;\Rightarrow\; r(\bar{x}) < (1 + g(p, m, f))\varepsilon$$

with ε used for local convergence

$$\downarrow$$

Corollary : *[Magoulès and Benissan, IEEE, 2018]*
$\|\|\|_\infty^w$–based residual
Relation between the exact and approximate residual is

$$\varepsilon = \frac{\widetilde{\varepsilon}}{1 + g(p, m, f)} \;\Rightarrow\; r(\bar{x}) < \widetilde{\varepsilon}$$

Generalization of non-FIFO AIS

$$r(\bar{x}) < \widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)}) + g(p, m, f)\varepsilon$$

$$\uparrow$$

Contraction property of fixed-point mappings
$\|f(x) - f(y)\| \leq \alpha\|x - y\|, \quad \alpha < 1$

$$\downarrow$$

$$|r(\bar{x}) - \widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)})| \leq h(\|\bar{x} - \bar{x}^{(1)}\|, \ldots, \|\bar{x} - \bar{x}^{(p)}\|)$$

General assumption on asynchronous iterations
$\lim_{k\to+\infty} \tau_j^i(k) = +\infty$

$$\downarrow$$

$$\|\bar{x} - \bar{x}^{(i)}\| < \delta^{(i)}(p, f)$$

$$r(\bar{x}) < \widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)}) + \delta(p, f)$$

- Inconsistent snapshot
  $\Rightarrow$ assumption : marker crosses at most m computation messages
  $\Rightarrow \bar{x}^{(1)} \neq \bar{x}^{(2)}$
  $\Rightarrow \bar{x} = \begin{bmatrix} \bar{x}_1^{(1)} & \ldots & \bar{x}_p^{(p)} \end{bmatrix}^{\mathsf{T}}$
  $\Rightarrow \widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)})$

  $$\varepsilon = \frac{\widetilde{\varepsilon}}{1 + g(p, m, f)}$$

  $\Rightarrow r(\bar{x}) < \widetilde{\varepsilon}$
  $\Rightarrow \|\bar{x} - \bar{x}^{(i)}\| < g^{(i)}(p, m, f)\varepsilon$

- No snapshot
  $\Rightarrow$ arbitrary $\bar{x}^{(i)}$

  $$\varepsilon = \widetilde{\varepsilon} - \delta(p, f)$$

Snapshot based asynch. convergence detection

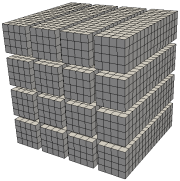
- Snapshot-based supervised termination *[Savari & Bertsekas, 1996]*
- Snapshot-based supervised termination NFAIS *[Magoulès & Benissan, IEEE, 2018]*
- Protocol-free termination *[Benissan & Magoulès, AES, 2020]*

	Intrusiveness	Centralization	Effectiveness	Messages size
Savari & Bertsekas, 1996	non-intrusive	two reductions	exact	$\mathcal{O}(n)$
Magoulès & Benissan, 2018	non-intrusive	one reduction	reliable	$\mathcal{O}(1)$
Benissan & Magoulès, 2020	non-intrusive	one reduction	reliable	0

Asynchronous convergence detection

Convection-diffusion :

$$\frac{\partial u}{\partial t} - \nu \Delta u + a.\nabla u = s$$



Problem size :

$$n = 185^3 = 6,331,625$$

Solver :

Block-Jacobi splitting
+
Gauss-Seidel on blocks

Residual threshold :

$$\varepsilon = 10^{-6}$$

Asynchronous convergence detection

NFAIS : $m = 1$
NBS : Non-blocking synchronization (snapshot without local convergence condition)
p = number of processors ; r = global residual after termination

	Synchronous			NBS		
p	$r \times 10^7$	wt	k	$r \times 10^7$	wt	k
168	8.33	701	281916	5.03	**536**	346226
240	8.31	516	284118	6.18	**378**	366231
360	8.33	382	287557	5.72	**250**	355394
480	8.32	302	289933	5.40	**202**	406611
600	8.32	278	292163	5.23	**168**	432390

	Savari & Bertsekas, 1996			NFAIS		
p	$r \times 10^7$	wt	k	$r \times 10^7$	wt	k
168	6.55	641	319703	6.54	640	319349
240	6.52	462	342476	6.42	463	343295
360	6.71	310	335008	5.19	314	339204
480	6.43	249	380524	6.63	250	383745
600	6.55	207	404544	6.06	209	410621

Experimentation : stability of computation platform

SGI ICE X supercomputer
Network : FDR Infiniband network (56 Gb/s)
Node : 2×12-cores Intel Haswell Xeon (2.30 GHz), 48 GB RAM
MPI : SGI-MPT

Residual threshold $\varepsilon := 10^{-6}$
Problem size (small) $n = 150^3 = 3,375,000$

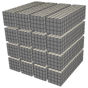
	[Benissan & Magoulès, 2020] No snapshot		[Magoulès & Benissan, 2018] Exact snapshot	
p	min r^*	max r^*	min r^*	max r^*
48	1.28e-06	1.47e-06	5.29e-07	6.81e-07
96	8.52e-07	1.11e-06	5.13e-07	6.33e-07
144	9.55e-07	2.39e-06	5.91e-07	6.05e-07
192	1.03e-06	1.29e-06	5.08e-07	5.83e-07
240	9.28e-07	1.47e-06	4.79e-07	5.55e-07
480	9.69e-07	2.83e-06	4.50e-07	5.76e-07
600	9.39e-07	1.32e-06	3.74e-07	5.28e-07

$$\varepsilon - 0.2 \times 10^{-6} < r^* < \varepsilon + 1.9 \times 10^{-6}$$

Convection-diffusion
$\frac{\partial u}{\partial t} - \nu \Delta u + \vec{a}.\nabla u = s$

Backward Euler in time
Centered FD in space

Domain partitioning



Global Jacobi
Local Gauss-Seidel

Final residual error
$r^* = \|A\bar{x}^* - b\|_\infty$
Expected precision
$r^* < \widetilde{\varepsilon} := 10^{-6}$
Number of iterations
$k_{\max} = \max_i k^{(i)}$

Experimentation : stability of computation platform

SGI ICE X supercomputer
Network : FDR Infiniband network (56 Gb/s)
Node : 2×12-cores Intel Haswell Xeon (2.30 GHz), 48 GB RAM
MPI : SGI-MPT

Residual threshold $\varepsilon := 10^{-6}$
Problem size (small) $n = 150^3 = 3,375,000$

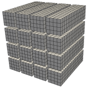
	[Benissan & Magoulès, 2020] No snapshot		[Magoulès & Benissan, 2018] Exact snapshot	
p	wtime (s)	$k_{\max}$	wtime (s)	$k_{\max}$
48	46	15894	61	16219
96	24	17353	31	17445
144	16	17698	21	17928
192	12	18122	16	18256
240	10	17596	13	18013
480	5	20356	7	20504
600	4	19627	6	20303

$$\varepsilon - 0.2 \times 10^{-6} < r^* < \varepsilon + 1.9 \times 10^{-6}$$

Convection-diffusion
$\frac{\partial u}{\partial t} - \nu \Delta u + \vec{a}.\nabla u = s$

Backward Euler in time
Centered FD in space

Domain partitioning



Global Jacobi
Local Gauss-Seidel

Final residual error
$r^* = \|A\bar{x}^* - b\|_\infty$
Expected precision
$r^* < \widetilde{\varepsilon} := 10^{-6}$
Number of iterations
$k_{\max} = \max_i k^{(i)}$

Experimentation : minimum residual threshold

Residual threshold $\varepsilon := 10^{-6}$

$$\varepsilon - 0.2 \times 10^{-6} < r^* < \varepsilon + 1.9 \times 10^{-6}$$

Residual threshold $\varepsilon := 4 \times 10^{-7}$
Problem size (small) $n = 150^3 = 3,375,000$

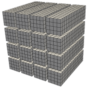
	[Benissan & Magoulès, 2020] No snapshot			
p	min r^*	max r^*	wtime (s)	$k_{\max}$
48	3.88e-07	6.28e-07	51	17386
96	4.06e-07	4.64e-07	26	18973
144	3.83e-07	7.90e-07	17	19298
192	4.24e-07	8.16e-07	13	19696
240	4.05e-07	5.83e-07	10	19060
480	4.29e-07	9.46e-07	6	22259
600	4.16e-07	8.54e-07	5	21221

$$\varepsilon - 0.2 \times 10^{-7} < r^* < \varepsilon + 5.5 \times 10^{-7}$$

Convection-diffusion
$\frac{\partial u}{\partial t} - \nu \Delta u + \vec{a}.\nabla u = s$

Backward Euler in time
Centered FD in space

Domain partitioning



Global Jacobi
Local Gauss-Seidel

Final residual error
$r^* = \|A\bar{x}^* - b\|_\infty$
Expected precision
$r^* < \widetilde{\varepsilon} := 10^{-6}$
Number of iterations
$k_{\max} = \max_i k^{(i)}$

Experimentation : full-size problem

Problem size $n = 150^3 = 3,375,000$
Residual threshold $\varepsilon := 10^{-6}$
$$\varepsilon - 0.2 \times 10^{-6} < r^* < \varepsilon + 1.9 \times 10^{-6}$$
Residual threshold $\varepsilon := 4 \times 10^{-7}$
$$\varepsilon - 0.2 \times 10^{-7} < r^* < \varepsilon + 5.5 \times 10^{-7}$$

Problem size $n = 185^3 = 6,331,625$
Residual threshold $\varepsilon := 10^{-7}$

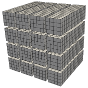
	[Benissan & Magoulès, 2020] No snapshot		[Magoulès & Benissan, 2018] Exact snapshot	
p	min r^*	max r^*	min r^*	max r^*
144	1.14e-07	1.81e-07	5.28e-07	6.20e-07
192	1.10e-07	2.12e-07	6.03e-07	6.10e-07
240	1.01e-07	2.15e-07	5.22e-07	5.62e-07
360	1.12e-07	1.91e-07	5.12e-07	5.49e-07
480	1.38e-07	3.11e-07	3.81e-07	5.49e-07
600	1.51e-07	2.01e-07	4.05e-07	4.98e-07

$$\varepsilon < r^* < \varepsilon + 2.2 \times 10^{-7}$$

Convection-diffusion
$$\frac{\partial u}{\partial t} - \nu \Delta u + \vec{a}.\nabla u = s$$

Backward Euler in time
Centered FD in space

Domain partitioning



Global Jacobi
Local Gauss-Seidel

Final residual error
$r^* = \|A\bar{x}^* - b\|_\infty$
Expected precision
$r^* < \widetilde{\varepsilon} := 10^{-6}$
Number of iterations
$k_{\max} = \max_i k^{(i)}$

Experimentation : full-size problem

Problem size $n = 150^3 = 3,375,000$
Residual threshold $\varepsilon := 10^{-6}$

$$\varepsilon - 0.2 \times 10^{-6} < r^* < \varepsilon + 1.9 \times 10^{-6}$$

Residual threshold $\varepsilon := 4 \times 10^{-7}$

$$\varepsilon - 0.2 \times 10^{-7} < r^* < \varepsilon + 5.5 \times 10^{-7}$$

Problem size $n = 185^3 = 6,331,625$
Residual threshold $\varepsilon := 10^{-7}$

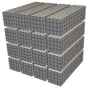
	[Benissan & Magoulès, 2020] No snapshot		[Magoulès & Benissan, 2018] Exact snapshot	
p	wtime (s)	$k_{\max}$	wtime (s)	$k_{\max}$
144	411	229848	437	186790
192	315	243487	337	199160
240	253	246580	275	203892
360	168	240124	184	198632
480	135	272549	147	225131
600	114	293166	123	240475

$$\varepsilon < r^* < \varepsilon + 2.2 \times 10^{-7}$$

Convection-diffusion
$\frac{\partial u}{\partial t} - \nu \Delta u + \vec{a}.\nabla u = s$

Backward Euler in time
Centered FD in space

Domain partitioning



Global Jacobi
Local Gauss-Seidel

Final residual error
$r^* = \|A\bar{x}^* - b\|_\infty$
Expected precision
$r^* < \widetilde{\varepsilon} := 10^{-6}$
Number of iterations
$k_{\max} = \max_i k^{(i)}$

Want to know more ?

- F. Magoulès, G. Gbikpi-Benissan
  JACK : an asynchronous communication kernel library for iterative algorithms
  The Journal of Supercomputing 73 (8), 3468-3487, 2017
  + parallel threads, continuous MPI_Recv, continuous residual
  - residual local convergence monitoring [Bahi, 2005]
- F. Magoulès, G. Gbikpi-Benissan
  Distributed convergence detection based on global residual error under asynchronous iterations
  IEEE Transactions on Parallel and Distributed Systems 29 (4), 819-829, 2018
  + all type of residuals with snapshot (NFAIS)
- F. Magoulès, G. Gbikpi-Benissan
  JACK2 : An MPI-based communication library with non-blocking synchronization for asynchronous iterations
  Advances in Engineering Software 119, 116-133, 2018
  + multiple channels, MPI_Irecv, all type of residuals (NFAIS [Magoulès & Gbikpi-Benissan, 2018] [Savari-Bertsekas, 1996] [Bahi, 2005, 2008])
- G. Gbikpi-Benissan, F. Magoulès
  Protocol-free asynchronous iterations termination
  Advances in Engineering Software 146, 102827, 2020
  + residual (approximate NFAIS based on snapshot)
  ++ residual (approximated global residual with MPIallreduce)

02 Asynchronous library

JACE library (Bahi et al., 2004)

JACE means a Java Asynchronous Computing Environment.
At a glance :

- Message passing environment based on Java Remote Method Invocation (RMI)
- Communication routines which naturally fit asynchronous iterations semantics
- Built for grid environments
- JACEP2P (Bahi et al., 2006) : node failure/disconnection support in peer-to-peer volatile environments
- JACEP2P-V2 (Charr et al., 2009) : fault-tolerant implementation of asynchronous convergence detection from Bahi et al., 2008

CRAC library (Couturier and Domas, 2007)

CRAC means a Grid Environment to Solve Scientific Applications with Asynchronous Iterative Algorithms.
At a glance :

- C++ counterpart of JACE
- Builtin low-level communication middleware for multi-site clusters (grid environments)
- Not based on MPI
- Last release in 2009

JACK library (Benissan, Magoulès, Zou, 2017)

JACK means Just an Asynchronous Computations Kernel.
At a glance :

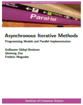
- C++ library on top of MPI (with C API available)
- ease the implementation of (any) asynchronous iterative algorithms
- allow multiple residual detection procedures
- running on multiple plateforms (SuperComputers, Multi-GPUs, Clouds)


F. Magoulès and G. Gbikpi-Benissan. *JACK : An asynchronous communication kernel library for iterative algorithms.* The Journal of Supercomputing, 73(8) :3468-3487, 2017.


F. Magoulès and G. Gbikpi-Benissan. *JACK2 : An MPI-based communication library with non-blocking synchronization for asynchronous iterations.* Advances in Engineering Software, 119 :116-133, 2018.

Want to know more ?

Culminated in one book :



G. Gbikpi-Benissan, Q. Zou, and F. Magoulès. *Asynchronous Iterative Methods : Programming Models and Parallel Implementation*. Institute of Computer Science, Antony, France, 2018. Hardcover 200 pages. ISBN : 978-2-490255-01-6

+ Provides simple asynchronous programming models
+ Allows to quickly upgrade your synchronous code to an asynchronous one
+ Features detailed concrete examples using a message-passing approach (MPI)
+ Provides good understanding of asynchronous convergence detection
+ Describes C++ and C language bindings of asynchronous iterations

Want to know more ?

With simple programming approach :

```
while (res_norm >= res_thresh) {
  // -- previous solution
  for (int i = 0; i < n; i++) {
    vec_U_prev[i] = vec_U[i];
  }
  // -- Jacobi algorithm
  Compute(vec_U, mat_A_loc, vec_F_loc);
  // -- communication
  jack_comm.Send();
  jack_comm.Recv();
  for (int j = 0; j < size - 1; j++) {
    for (int i = 0; i < rbufs_sizes[j]; i++) {
      vec_U[jbegin[j] + i] = recv_bufs[j][i];
    }
  }
  // -- residual
  for (int i = ibegin; i < iend; i++) {
    res_vec_buf_loc[i-ibegin] = std::abs(vec_U[i] - vec_U_prev[i]);
  }
  // -- maximum norm
  res_norm_loc = res_vec_buf_loc[0];
  for (int i = 1; i < lbuf_size; i++) {
    if (res_norm_loc < res_vec_buf_loc[i]) {
      res_norm_loc = res_vec_buf_loc[i];
    }
  }
  // -- convergence
  lconv_flag = (res_norm_loc < res_thresh);
  jack_conv.UpdateResidual();
  numb_iters++;
}
```

"The learning curve is steep, but the productivity gains are well worth the effort."

Ryan Paul, Vim's 20th anniversary, 2011

The End