UNIVERSITE DU HAVRE
UFR des Sciences et Techniques

MÉMOIRE

Pour l'obtention du diplôme du Master en Mathématiques et informatique des systèmes complexes et distribués

Algorithmes Itératifs Asynchrones Distribués sur un Système Distribué Volatile

Présenté le 25 Juin 2013
Par
Mlle. Manel SLOUGUI

Encadrée par :

M. Moustafa Nakechbandi

M. Jean-Yves Colin

Année Universitaire 2012/2013

Remerciements

Je tiens à remercier chaleureusement Mr Jean-Yves Colin et Mr Moustafa Nakechbendi, pour m'avoir offert l'opportunité de réaliser ce sujet de stage, pour leurs nombreux conseils qu'ils m'ont prodigué, leurs disponibilité ainsi que leurs gentillesse.

Mots clés : algorithme parallèle itératif asynchrone, système linéaire, détection de la convergence, architecture distribuée volatile

Résumé :

J'ai réalisé mon stage de fin d'étude, dans le cadre du Master2 MATIS, au laboratoire **Litis** de l'université du havre au sein de l'équipe *Réseaux d'Interactions et Intelligence Collective*, le contexte de travail dans lequel notre étude évolue est : l'exécution de programmes sur un réseaux mobile ad hoc (MANet) qui par définition est instable et est constituée de différents équipement informatiques communiquant entre eux via différentes technologie comme le wifi, le bluetooth….etc. La volatilité des nœuds mais aussi celle des voies de communication entre ces derniers sont à prendre en considération lors du développement de n'importe quelle application destinée à être exécutée sur ce genre de réseau.

Le travaille effectué tout au long du stage porte précisément sur l'adaptation d'une méthode itérative dans le cadre du calcul itératif asynchrone dans un environnement distribuée volatile et cela afin de résoudre des systèmes linéaire.

L'objectif de ce projet est d'établir des statistiques par simulation pour différents types de système linéaire qui sont généralement issus des simulations de problèmes relatifs à la : Mécanique des Structures, dynamique des fluides, électromagnétique ou autre, de ce fait les matrices sont de différentes tailles, structures mais aussi de différentes catégorie c'est-à-dire : creuses, bandes, …etc.

La première partie de ce mémoire est dédiée a l'état de l'art et plus exactement a l'étude de l'existant concernant le calcul itératif parallèle asynchrone sur architecture distribuée volatile, au cours de ce chapitre les différentes approches du calcul itératif asynchrone qui ont été abordé lors d'études précédentes seront exposées, ainsi que les divers mécanismes mis en place pour la détection de la convergence et pour la tolérance au pannes seront eux aussi exposées.

La deuxième partie de ce mémoire est consacré en premier lieu à l'implémentation d'un algorithme itératif qui en deuxième lieu sera utilisé lors des divers tests faisant partie des approches que nous avons adopté et qui concerne le calcul itératif asynchrone sur architecture volatile. Dans le cadre de notre étude, l'exécution de la méthode itérative s'effectue sur une architecture séquentielle au lieu d'une architecture distribuée et la volatilité du réseau est simulée par l'ajout de lignes de retard. Nous terminerons notre mémoire avec une conclusion relative au travail effectué et aux résultats obtenus suite à celui-ci.

Keywords: parallel asynchronous iterative algorithm, linear system, detection of convergence, volatile distributed architecture

Abstract:

I realized my training study of the MASTER 2 option MATIS, in the laboratory **Litis** of the University of Le Havre with the team named: *Interaction Networks and Collective Intelligence*. The working environment in which our study evolves is: the implementation of programs on an ad hoc mobile networks (MANet) which is by definition unstable and is composed of various IT equipment that communicate together via different technologies such as wifi, bluetooth .... etc. The volatility of the nodes and the communication channels' volatility have to be considered during the development of any application that will be executed on such a network.

Our work concerns precisely the adaptation of an iterative method for the asynchronous iterative calculus in a volatile and distributed architecture, in order to solve linear systems.

The objective of this project is to establish statistical simulation for different types of linear systems that are generally derived from simulation of structural, fluid dynamics or electromagnetic problems, thus the matrices are of different sizes, structures and also different category: sparse, band, ...etc.

The first part of this thesis is dedicated to the study of the existing that concerns the parallel asynchronous iterative calculus on volatile distributed architecture. In this chapter the different approaches of asynchronous iterative calculus that have been treated in previous studies will be presented, and the various mechanisms that have been developed for the detection of convergence and the fault tolerance are also presented.

The second part of this thesis is devoted to the implementation of an iterative algorithm, that will be used for the different tests which are included in the approaches that we have adopted through our study of asynchronous iterative computation on a volatile architecture. In our work, the execution of the iterative method is performed on a sequential architecture instead of a distributed one and the volatility of the network is simulated by adding delay lines. We end this thesis with a brief conclusion concerning the work performed and the results obtained from it.

Présentation du LITIS

Le LITIS (Laboratoire d'**I**nformatique, de **T**raitement de l'**I**nformation et des **S**ystèmes) est née en **2006** du regroupement de **4** laboratoires de la haute Normandie, qui pour cette fusion ont communément partagé leurs moyens mais aussi leurs thèmes de recherche, **7** équipes de chercheurs sont affectées au laboratoire et que l'on retrouve sur les différents sites qui sont associé au laboratoire ; université du Havre, de Rouen et l'INSA de rouen.

Les trois principaux axes de recherche du laboratoire **LITIS** sont

1) Le Pôle combinatoire et algorithmes
Le Calcul Formel, le Codage et le Chiffrement entre autres font partie des domaines couverts par ce thème

2) Le Pôle traitement des masses de données
Parmi les domaines traitées dans le cadre de cette recherche il y'a le Traitement de l'information en biologie santé, Imagerie Médical, Systèmes de Transport Intelligent

3) Le Pôle interaction et systèmes et système complexe
Cette axe de recherche s'intéresse particulièrement a 2 domaines : Réseaux d'Interaction et Intelligence Collective mais aussi a un autre domaine qui est ; la Modélisation, Interactions et Usages.

Quelques chiffres concernant le **LITIS**

- date de sa création : **2006.**
- **75** enseignants chercheurs et autant de doctorants.
- **200** articles scientifiques publiés ces dernières années.
- **10** brevets déposés.

Table des matières

Table des Figures

Table des Tableaux

I. Introduction

Les architectures distribuées sont nécessaire à la résolution de gros problèmes qui de par leurs taille et leurs complexité ne peuvent pas se contenter d'une seule machine avec un espace mémoire et une puissance de calcul limitée.

Le problème est représenté par un ensemble de taches ; chacune d'entre elles est affectée à une machine faisant partie du réseau distribué, et une méthode de calcul parallèle doit être utilisée par l'algorithme pour que ces tâches parallèles puissent coopérer correctement et efficacement.

Les calculs des unités composant ces architectures sont combinés dans le but de résoudre le problème donné.

Certains aspects sont à prendre en considération afin d'exploiter au mieux les architectures distribuées pour résoudre les gros problèmes numériques :

- La communication entre les différentes machines doit être assurée sachant que l'échange de données ainsi que la détection de la convergence nécessite une synchronisation entre Les nœuds hétérogènes ce qui conduit à des périodes d'inactivité pour les unîtes les plus puissantes.
- La capacité de chaque nœud de l'architecture (hétérogénéité des nœuds composants l'architecture) doit être prise en compte lors de l'affectation des taches pour assurer des performances optimales.
- Le besoin d'assurer la stabilité de l'application nécessite de décentraliser certains mécanismes comme par exemple celui de la détection de la convergence.
- La conception d'un mécanisme de détection de panne mais aussi de restauration est nécessaire afin d'assurer un system tolérant aux pannes (volatilité des nœuds).

Les algorithmes itératifs asynchrones sont très bien adaptés aux architectures distribuées volatiles, du fait qu'ils assurent une tolérance aux pannes. Et ils éliminent aussi les problèmes occasionnés par la synchronisation entre les nœuds participants au calcul. Ils permettent donc de résoudre plusieurs problèmes qui sont la conséquence de la parallélisation et de l'exécution de programme dans un environnement distribué.

La résolution de gros systèmes linéaires par des algorithmes itératifs asynchrones et cela sur une architecture distribuée devient encore plus intéressante lorsqu'il y a une bonne affectation du calcul des différentes composantes du vecteur de solution sur les diverses machines constituant cette architecture, pour assurer ainsi une performance de calcul optimale.

II. ETAT DE L'ART

1. Architectures parallèles

De gros calculs sont nécessaires dans les applications industrielles et autres, afin d'assurer ces calculs on opte pour l'architecture parallèle. La performance de cette dernière est la combinaison de performances de ses composantes (latence, débit, puissance de calcul et de stockage, etc.), à travers ce type d'architecture il n'y a plus de limite quant à la mémoire ou bien processeur, le traitement se fait indépendamment et de manière accélérée.

Une architecture parallèle permet d'effectuer un calcul distribué, afin d'exploiter la puissance de calcul (traitement de données ainsi que leur stockage) d'un grand nombre d'ordinateurs. Il existe différentes architectures pour effectuer le calcul parallèle, tout en sachant que l'hétérogénéité et la distance entre les machines qui composent l'architecture distribuée influent sur la programmation mais aussi la performance des applications distribuées.

Il existe principalement deux types de machines parallèles [13] :

Type 1 : ***Machine parallèle a mémoire partagée*** **:** elles permettent une parallélisation de haut niveau et leurs utilisation est assez simple, le nombre de processeurs utilisés est limité, comme exemple de ce type de machine nous avons l'architecture SMP

Type 2 : ***Machine parallèle à mémoire distribuée*** **:** pour ces machines, chaque processeur possède sa propre mémoire, et le parallélisme se fait par échange de message, les communications entre les processeurs se font avec des appels de fonction des librairies standards comme PVM (Parallel Virtual Machine) ou bien MPI (Message Passing Interface), il y'a un partage transparent des ressources mais aussi une grande puissance de calcul.

Les super calculateurs sont habituellement les mieux adaptés pour assurer les gros calculs mais pour des raisons économiques les spécialistes de ce domaine ont opté pour le développement de nouvelles architectures parallèles, nous citons quelques exemples d'architectures parallèles : les clusters, les architectures AMP, architectures à mémoire partagée SMP, architectures à mémoire distribuée CRAY T3E (ancien modèle).

1.1. Définition d'une machine avec architecture Multi Processeur Symétrique (SMP)

L'architecture matérielle permet de combiner les puissances de traitement de plusieurs processeurs identiques simultanément, et de leur faire partager une mémoire commune [9].

1.2. Définition d'une machine avec architecture Multi Processeur Asymétrique (AMP)

C'est la même architecture que la SMP, la différence réside en l'hétérogénéité des processeurs, ce qui a pour effet positif d'augmenter les performances dans la mesure où on utiliserait des processeurs dédiés (spécialisés) à la réalisation de taches spécifiques.

1.3. Définition d'un Supercalculateur

Un ordinateur qui est composé de plusieurs milliers de processeurs et de disque dur et d'une très grande capacité mémoire afin d'atteindre les plus hautes performances possibles lors des calculs, ils sont généralement conçues pour les calculs numériques scientifiques comme le calcul matriciel ou vectoriel, mais pour des raisons économiques les spécialistes de ce domaine ont optée pour le développement d'autres architectures parallèles. Parmi les calculateurs les plus puissants au monde [3] il y a: Titan, Sequoia (IBM), K computer, Mira, etc.

1.4. Définition d'un Cluster

Un réseau sur lesquelles s'effectue des calculs parallèles est qui est composé de plusieurs PC multiprocesseurs homogènes (la même localisation) et qui sont regroupés logiquement ; une seule et même adresse réseau est affectée à l'ensemble des machines du cluster, de ce fait ce dernier est vu comme une seul entité par les autres machines et réseaux aussi, seules les machines appartenant à ce réseau sont autorisées à transmettre des messages au sein de ce dernier, une des principales caractéristiques des clusters est qu'il connait l'état de chacune des machines qui le composent mais aussi la mutualisation des ressources, la transparence, la performance (puissance de calcul et stockage), la Figure 1 illustre un schéma de cluster.

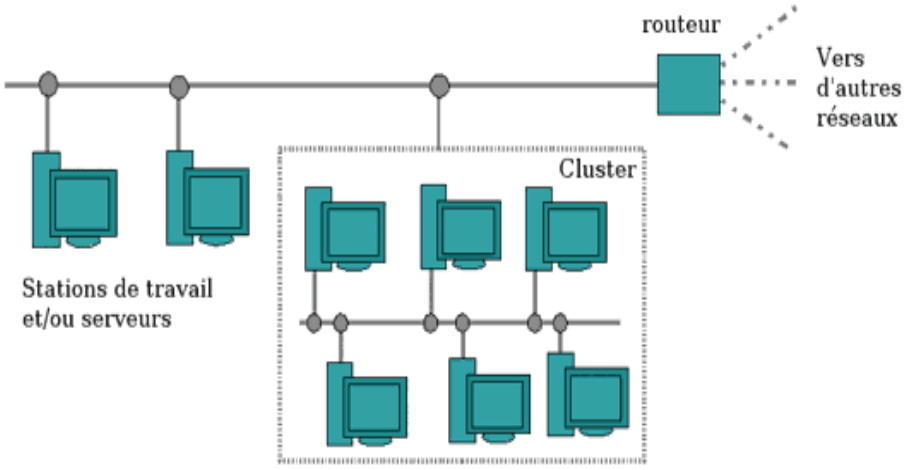


Figure 1 : schéma d'un cluster

Parmi les clusters les plus performants : le cluster chinois Tianhe-1A avec une performance de 2.57 petaflops, le cluster jaguar de 1.76 petaflops.

Différents types de clusters existent :

- ***Cluster local**:* il est composé d'ordinateurs homogènes interconnectés entre eux, et leurs situations géographique est proche les unes des autres.

- ***Cluster Distribué:*** il est composé de clusters locaux hétérogènes très espacés géographiquement mais reliés au sein d'un même réseau WAN.
- ***Architecture de calcul global:*** elle est composée de machines hétérogènes non utilisées des utilisateurs interconnectées via internet.

1.5 Définition d'une Grille de calcul [12]

Ensemble de systèmes hétérogènes (PC, Cluster,…) distribués à travers un réseau LAN, MAN ou WAN, sur lesquels des taches de calcul sont distribuées. Les grilles permettent de réquisitionner les ressources inutilisées de machines distribuées (disponibilité des ressources) sur un réseau afin d'augmenter la capacité de stockage mais aussi pour assurer la puissance de calcul, la Figure 2 illustre un schéma de grille de calcul.

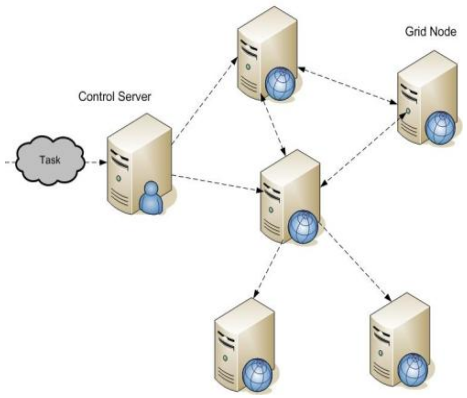


Figure 2 : schéma d'une grille

En France Le projet GRID 5000 vise à construire une plate-forme constituée d'une grille de calcul de grande taille qui permet l'interconnexion de 5000 PC repartis par grappes de 500 pc à travers différentes régions de France, la Figure 3 représente les sites de la plate forme GRID 500

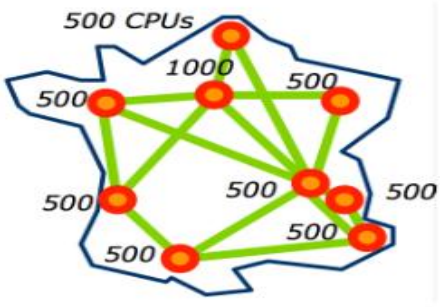


Figure 3 : Les sites de la plate forme GRID 5000

Il existe 3 grands types de topologie de grille de calcul [14]:

- **L'intragrille :**

Elle est constituée d'un ensemble de services et de ressources assez homogènes, qui appartiennent au même organisme et qui sont interconnectés via un réseau performant, le domaine de sécurité de ce type de grille est unique et est contrôlé par les administrateurs de l'organisme.

- **L'extragrille** :

L'extragrille est constituée de plusieurs intragrilles. Les ressources qui font partie de ce type de grille sont relativement dynamiques et sont interconnectées par un réseau hétérogène, et contrairement à l'intragrille, l'extragrille possède différents domaines de sécurité.

- **L'intergrille :**

Une intergrille consiste à agréger des gilles de différents organismes afin d'en obtenir une seule, cette dernière est alors composée de ressources très dynamiques qui sont interconnectées par un réseau très hétérogène. Ce type de grille possède aussi différents domaines de sécurité.

Les grilles sont caractérisées par leurs nature hétérogène, dynamique et volatile, et cela quelque soit le type auquel elles appartiennent :

- Hétérogénéité des ressources :
  Les ressources qui constituent la grille sont hétérogènes, et cela pour les logiciels, le matériel ainsi que les systèmes d'exploitation.

- Nature dynamique des ressources :
  La nature dynamique des ressources qui constituent la grille, nécessite la mise en place d'un mécanisme de tolérance aux pannes, et cela pour mieux s'adapter aux changements dynamiques de ces ressources en question.

- Passage à l'échelle :
  Les applications en charge de la gestion des ressources doivent être aptes à ordonnancer les taches sur les milliers d'ordinateurs qui constituent une grille.

2. Outils et environnements pour applications parallèles

La parallélisation permet de résoudre un problème en le décomposant et en l'exécutant sur différents processeurs communicants entre eux. Elle peut s'effectuer soit en décomposant les données qui constituent le problème ou bien en décomposant le problème lui-même en sous problèmes, ou bien les deux méthodes peuvent être fusionnées.

La communication et les échanges de données entre les différents processeurs constituant l'architecture parallèle différent selon l'architecture adoptée (mémoire partagée ou pas), une synchronisation entre les processeurs est nécessaire lorsque il y a un partage de mémoire même dans le cas ou le partage est virtuel et cela afin d'assurer la cohérence des données échangées qui sont lues et écrites.

Des environnements d'exécution pour algorithmes itératifs parallèles ont été développés afin de bénéficier des avantages des architectures distribuées tout en prenant en considération leurs hétérogénéités ainsi que leurs volatilités.
Pour les différentes architectures distribuées vues précédemment, nous citons quelques exemples d'environnements d'exécution parallèle qui y correspondent [4]:

- *Le cluster local* : PVM, OpenMP, MPI
- *Le cluster distribué :* Globus, Legion, Proactive, Padico
- *L'Architecture de calcul global:* P2P-MPI, JXTA, XtremWeb

2.1 Présentation des environnements

Différents outils sont utilisés pour développer des applications parallèles distribuées comme par exemple les bibliothèques de fonctions *MPI* (avec ses différentes versions améliorées) et *PVM* mais ces bibliothèques se sont avérées ne pas être très adaptées pour l'algorithme *IACA (*algorithme de calcul itératif asynchrone*)* que nous exposerons plus en détail par la suite dans ce rapport, d'où la nécessitée de mettre en place des modèles d'environnement beaucoup plus pertinents comme celui de JACE qui est une bibliothèque de communication. Mais aussi son autre version qui est JACE P2P.

2.1.1 Présentation de JACE *(Java Asychronous Computing Environement)* [6]

JACE est un environnement de programmation d'application distribuée et parallèle pour le calcul itératif asynchrone (utilisable pour les applications synchrones aussi), il a été développé par l'équipe « Algorithmique Numérique Distribuée » de l'université de Franche-Comté. Cet environnement permet d'établir une machine virtuelle à partir des différents sites hétérogènes distants, du fait d'utiliser le langage *java*, cet environnement possède plusieurs avantages parmi lesquelles nous citons :

- ✓ Peut créer une machine parallèle virtuelle en reliant des machines hétérogènes qui sont sur des sites distants les uns des autres.
- ✓ Utilise la notion de *thread* (ce qui permet de dissocier le calcul des communications et de ce fait assurer la parallélisation) et de *RMI*.
- ✓ Tolère les pertes de messages.
- ✓ Fournit une interface de programmation simple d'utilisation.

JACE se décompose architecturalement en quatre principaux éléments qui sont : les taches, le distributeur de taches, la console et le démon.

Le Démon : il est responsable de la synchronisation ainsi que des communications entre les différents nœuds, il doit être lancé sur chaque nœud de la machine virtuelle. Un démon exécute une ou plusieurs taches représentée(s) par des threads, ces derniers assurent une bonne asychronisation et cela en séparant entre le calcul et la communication (la communication se fait à base de thread : thread *Sender*, thread Receiver).

Les Taches : elles s'exécutent sur les différents nœuds et collaborent ensemble en s'échangeant différents messages et données dans le but de résoudre le même problème.

Le Distributeur de taches : se charge de distribuer les taches sur les différents sites composant la machine virtuelle.

2.1.2 Présentation de JACE P2P *(**J**ava **A**sychronou**s** **C**omputing **E**nvironement for **P**eer-to-**P**eer)*

JACE P2P est un environnement destiné aux plates formes pair- à-pair [4] et qui comme son prédécesseur JACE permet d'exécuter des algorithmes itératifs asynchrones parallèles (exécution de l'algorithme *AIAC*) dans une architecture distribuée et hétérogène, il n'est pas complètement tolérant au pannes (JACE n'est pas du tout tolérant au pannes). Beaucoup de ses mécanismes sont centralisés comme par exemple : la détection de convergence mais aussi celle des pannes. Contrairement à JACE, JACE P2P assure l'exécution d'application parallèle sur des architectures volatiles.

Pour pallier à certaines défaillances de JACE P2P, une nouvelle version complètement décentralisée a été mise au point (JACE P2P V2). Cette version qui est aussi complètement tolérante au pannes, utilise trois types de nœuds; les démons pour calculer (chargées de l'exécution des taches), les lanceurs pour lancer les calculs et les super nœuds qui représentent des points d'accès à cet environnement.

2.2 Détection de la convergence

Il existe différentes méthodes qui permettent de détecter la convergence des algorithmes itératifs [5] et cela afin de mettre un terme au calcul itératif. Certaines sont spécifiques à certains environnements, alors que d'autres suivent le schéma général de la détection de la convergence.

2.2.1 Description d'un schéma général pour la détection de convergence

La convergence globale n'est détectée que si tous les processus détectent une convergence locale. Cette dernière est établie selon différentes méthodes; la plus commune est celle de comparer la différence entre les valeurs d'une même variable issues d'itérations successives à un seuil **ε**.

L'arrêt total de l'algorithme itératif parallèle à la détection d'une convergence global pose problème dans le cas asynchrone. Et cela en considérant le cas ou les données fraiches n'ont pas été reçues et donc pas utilisées pour le calcul, de ce fait il peut y avoir une fausse détection de la

convergence locale. Pour y remédier une des approches adoptée est celle du *jeton*; il est défini comme un booléen qui vérifie la convergence local de toutes les unités de calcul mais aussi que les valeurs les plus récentes de toutes les données qui ont été envoyées, ont bien été reçues.

2.2.2 Exemples de mécanisme de détection de convergence dans des environnements asynchrones

- ***D**étection **C**entralisée de la **C**onvergence global dans l'environnement JACE* [4]:

Chaque *démon* qui détecte sa convergence local informe l'unité de calcul responsable de la détection de la convergence, cette dernière vérifie l'état des autres unités de calcul, pour détecter la convergence globale. Ce concept a été développé pour les applications itératives synchrones parallèles, et de ce fait il n'est pas très fiable pour le modèle *IACA*. Car dans ce model, les messages ne sont pas échangés régulièrement (il n'y a donc pas de différence dans les valeurs des variables entre les différentes itérations) ce qui augmente le risque de détection d'une fausse convergence locale qui impliquera celle d'une fausse convergence global. Et pour y remédier l'approche suivante est adoptée ; chaque fois qu'un nœud détecte une convergence local, il exécute un certain nombre d'itération (*pseudo période*) au bout desquelles il vérifie si la convergence est toujours atteinte en utilisant toutes les variables calculées par les autres processeurs et si c'est le cas, il en informe alors le nœud responsable.

- ***D**étection **D**écentralisée de la **C**onvergence global dans l'environnement JACP2P-V2*(*DCD*)

Un algorithme décentralisé de détection de convergence global a été implémentée dans JACEP2P. Il se déroule en deux phases : une détection de la convergence globale suivie d'une phase de vérification.

La détection de convergence local au niveau de chaque démon utilise le concept de pseudo période (un nombre minimum d'itérations est fixé pour lesquelles le démon qui a convergé localement reçoit des données fraiches des autres démons. Avec ces données avec il calculera une autre itération), pour éviter ainsi la détection de fausse convergence locale.

2.3 Les différents mécanismes de détection de panne et sa restauration

La tolérance aux pannes est une condition nécessaire pour assurer des applications fiables, et plus précisément dans le cas du calcul distribué. De ce fait un mécanisme de détection et restauration des pannes est nécessaire à mettre en place, sans que les performances du réseau ne soient affectées pour autant.

Différents mécanismes de tolérance aux pannes ont été développés, ces mécanismes incluent la détection de la panne et sa restauration. Parmi les modèles les plus intéressants [4] nous citons:

2.3.1 Détection de la panne

- **le modèle dual** (serveur centralisé) : parmi les modèles de mécanisme de détection de panne, nous trouvons ce modèle dont le principe est le suivant:
  Chaque nœud du réseau envoie un message *heartbeat* à chaque période *t* de temps au serveur central. Si après une période 2*t* le serveur ne reçoit rien d'un nœud, il lui envoie alors une *requête liveness* et s'il n'a toujours pas de réponse, le nœud est considéré en panne. Le mécanisme de restauration est déclenché alors. Afin d'envisager cette approche pour les applications de grandes échelles, une variante a été apportée. Elle consiste à dupliquer le serveur tel que chacune de ses copies, gère un sous groupe des machines composant le réseau, des messages *heartbeating* sont aussi échangés entre les serveurs.

- **Gossip protocol:** chaque nœud maintient une table avec l'ensemble des nœuds et leurs dernier *heartbeat*. Chaque nœud envoie sa table *heartbeat* à chaque *Tgossip*, le *heartbeat* d'un nœud qui n'a pas été modifié depuis un certain temps sera associé à un nœud défaillant et sa suppression de la table est effectuée, ce protocole est pratique pour les applications a grandes échelles et les architectures distribuées.

2.3.2 Mécanisme de restauration

Parmi les différents mécanismes existants de restauration en cas de détection de panne, [4,8] nous distinguons la stratégie de *Recouvrement arrière*. Elle consiste à restituer l'état ultérieur d'un système (les états du système sont sauvegardés périodiquement) afin d'avoir un état global cohérent. Cet état qui représente l'ensemble des processus constituant l'architecture distribuée avant la panne, et les dépendances causales doivent être respectée. Les *Points de control* sont utilisés afin de sauvegarder pour chaque nœud les données qui lui sont nécessaires afin de continuer son calcul après une panne. Cette technique se base sur deux approches principales:

- **Le Recouvrement arrière basé sur les points de reprises (coordonnés/non coordonnés)**

Les protocoles basés sur les points de reprises utilisent les points de reprises propres au processus afin de restituer un état global cohérent du système. En d'autres termes ce sont des points de contrôles qui sont utilisés pour sauvegarder pour chaque nœud les données qui lui sont nécessaires afin de continuer son calcul, nous notons deux types de points de reprises :

Points de reprise coordonnés : cette approche nécessite la synchronisation de tous les nœuds mais aussi la coordination des processus. Quand un nœud tombe en panne tous les nœuds participant au calcul s'arrêtent de calculer jusqu'à ce que le nœud défaillant soit remplacé. Un processus définis comme coordinateur établis son point de reprise et celui des autres processeurs de façon coordonnée, et la restauration des processus se fait selon leurs points de reprises respectifs. Le nœud remplaçant récupère les sauvegardes effectuées afin de relancer la tache et les autres nœuds aussi consultent le dernier point de contrôle et reprennent leurs taches

respectives. Malgré tout ses points positifs, ce mécanisme de restauration de panne engendre une latence qui est accentuée avec les architectures de grandes tailles.

Points de reprise non coordonnés : dans ce cas il n'y a pas de synchronisation entre les différents nœuds qui constituent l'architecture distribuée. Chaque processus établit un point de reprise et la ligne de recouvrement (reconstitution de l'état global cohérent le plus récent du système), se fait en fonction des dépendances causales entre les processus et leurs différents points de reprises locaux qui sont établis à différent temps. La reprise du processeur défaillant se fait par rapport à ces points de reprises.

L'autre approche de recouvrement arrière est :

- **Le Recouvrement arrière avec journalisation des messages**

Un état global est construit à partir des messages échangés entre le moment défaillant et le dernier point de reprise, les nœuds doivent être dotés de mécanisme de réémission de message. Il y a différents types de journalisation : pessimiste, optimiste, causal.

3. Les Méthodes Numériques

Beaucoup de problèmes physiques, mathématiques, naturelles peuvent être modélisés [7,10] par des équations mathématiques de la forme :

$$f(x) = 0 \ldots\ldots\ldots (3.1)$$

Cette équation peut être translatée sous la forme :

$$F(x) = x \ldots\ldots (3.2)$$

En posant : $F(x)=x-f(x)$, les solutions de l'équation (3.2) sont appelées points fixes de F.

Nous nous intéressons à la résolution de système linéaire de type $\boldsymbol{Ax}=\boldsymbol{b}$. Cette résolution se fait par le biais des méthodes numériques.

Il existe 2 classes principales pour les méthodes de résolution numériques

- Les méthodes directes
- Les méthodes itératives

Le premier type de méthode, calcule la solution exacte du problème en un nombre fini d'étapes et les particularités de l'application F ne sont pas prises en considération. Comme exemple de méthode directe nous avons *la méthode Cholesky, La méthode de Gauss.*

Un inconvénient de ces méthodes est que des erreurs d'arrondis peuvent venir fausser les résultats, en particulier sur les grandes matrices, ces méthodes sont souvent difficiles à paralléliser

Les méthodes itératives : leur principe consiste à itérer successivement sur le même bloc, jusqu'à arriver à la convergence. En d'autres termes obtenir une bonne approximation de la solution du problème tout en en utilisant les particularités de F et en tenant compte de conditions plus ou moins strictes. Cette classe regroupe un grand nombre d'algorithmes qui sont utilisés afin de résoudre le problème (3.1). Parmi ces méthodes nous trouvons la méthode de *Jacobi* et celle de *Gauss-Seidel* qui sont des méthodes de substitution successives.

Les algorithmes itératifs sont beaucoup plus faciles à paralléliser et ils souffrent beaucoup moins de problèmes d'arrondis.

Les méthodes itératives sont aussi nécessaires pour résoudre différentes classes de problèmes dont : les systèmes non linéaires avec des équations différentiels ordinaires ou bien des équations aux dérivées partielles.

Les méthodes des deux classes précédentes sont présentées mathématiquement comme des algorithmes séquentiels. Avec ces méthodes, les instructions sont exécutées séquentiellement sur une seule machine monoprocesseur. Résoudre de gros problème de cette manière n'est pas envisageable. Pour palier à ce problème, l'utilisation de plusieurs unités de calcul s'avère être nécessaire d'où la notion d'architecture parallèle; qui signifie que les différentes unités composants cette architecture travaillent parallèlement pour résoudre un problème donné. La parallélisation de ces algorithmes consiste à reformuler leurs écritures et cela en les décomposant en taches indépendantes les unes des autres, et en les affectant au différentes ressources de calcul qui composent l'architecture parallèle.

3.1 Méthodes itérative pour la résolution d'un système linaire $Ax=b$

3.1.1 Méthodes de point fixe

Afin de résoudre le système linéaire $\boldsymbol{Ax=b}$

Où A est une matrice symétrique définis positive dans $R^{n\times n}$, x et b sont des vecteurs de R^n

La matrice A est décomposée sous la forme $A = G-H$, tel que G est inversible.

Alors

$Ax=b \Leftrightarrow Gx = Hx + b \Leftrightarrow x = Mx+N$, tel que $M = G^{-1}H$ et $N= G^{-1} b$

Ce qui nous ramène à un problème de point fixe définis par l'équation (3.2) :

Nous définissons la suite récurrente :

$$\begin{cases} x \text{ vecteur de } R^n \\ x^{t+1} = Mx^t + N \ , k = 0, 1, 2, \ldots \end{cases} \qquad (3.3)$$

Si la suite définis ci-dessus est convergente, dans ce cas sa limite *x* est la solution du système.

Définition 3.1

Le rayon spectral $\rho(A)$ d'une matrice *A* est le plus grand des modules des valeurs propres de *A*

Théorème 3.1

La suite x^t définie par (3.3) converge vers la solution de (3.2) si et seulement si $\rho(A) < 1$.

Parmi les méthodes de point fixe, nous comptons la méthode de *Gauss-Seidel*

Méthode de *Gauss-Seidel*

Nous posons $A = D\text{-}E\text{-}F$, tel que *D* est la partie diagonale de *A*, *E* (resp. *F*) la partie triangulaire inferieur (resp. supérieur) changée de signe de *A*, nous supposons que tous les a_{ii} sont différents de 0, de ce fait *D* est inversible, en considérant : $G = D\text{-}E$ et $H = F$, nous obtenons :

$M = (D\text{-}E)^{-1} F$ et $N = (D\text{-}E)^{-1} b$

Le calcul consiste à résoudre directement le système : (D-E) $x^{t+1} = Fx^t + b$

$$\sum_{j=1}^{i} a_{ij} \ x_j^{t+1} \ = \ \sum_{j=i+1}^{n} a_{ij} \ x_j^t \ + b_i \qquad (3.4)$$

$$x_i^{t+1} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} \, x_j^{t+1} - \sum_{j=i+1}^{n} a_{ij} \, x_j^t}{a_{ii}} \qquad (3.5)$$

Définition 3.2

La matrice *A* est à diagonale strictement dominante si

$\forall i \ \ |a_{ii}| > \sum_{j=let j \neq i}^{n} |a_{ij}|$ ou

si $\forall i \ \ |a_{ii}| > \sum_{j=let j \neq i}^{n} |a_{ji}|$

Nous citons dans la suite quelques théorèmes qui concernent la convergence :

Théorème 3.2

Si *A* est une matrice à diagonale strictement dominante alors la méthode de *Gauss-Seidel* appliquée au système ***Ax=b*** est convergente pour tout x^0.

Théorème 3.3

Si *A* est une matrice définie positive alors la méthode de *Gauss-Seidel* appliquée au système ***Ax=b*** est convergente pour tout x^0.

De l'équation (3.5), nous déduisons l'algorithme suivant qui représente la résolution d'un système linéaire de type : ***Ax=b*** par la méthode *Gauss-Seidel*. Pour chaque itération nous voyons si il y a eu convergence pour s'arrêter, sinon le calcul de la composante x_i a l'itération $t+1$ se fait avec le calcul de x_j a l'itération t, quel que soit i,j appartiennent à n (le nombre de processeurs, auquel nous affectons le calcul de chaque tache ou bien le calcul de chaque composante x_i du vecteur x).

*Algorithme de résolution du system linéaire **Ax=b**, en utilisant la méthode Gauss-Seidel*

Début
Donnée : matrice ***A***, vecteur ***b***, vecteur initial $\boldsymbol{x^0}$, résidu ***eps***
Pour (t=1 étape 1 jusqu'à convergence)
{
Pour (i = 1 step 1 until i = n)
{

$$x_i^{t+1} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}\, x_j^{t+1} - \sum_{j=i+1}^{n} a_{ij}\, x_j^{t}}{a_{ii}}$$

}

Fin pour
Détecter convergence
}
Fin pour
Donner une approximation du vecteur *x* après convergence
Fin

Exemple d'application d'une méthode itérative

Afin de résoudre le système linéaire de type ***Ax=b*** en appliquant l'algorithme, décris précédemment et cela avec les données suivantes :

$A=\begin{pmatrix}2&1&0&0&0&0\\1&2&1&0&0&0\\0&1&2&1&0&0\\0&0&1&2&1&0\\0&0&0&1&2&1\\0&0&0&0&1&2\end{pmatrix}$, $b=\begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix}$, ***eps***=0.0001 et le vecteur initial $x^0=\begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix}$

Nous considérons que les étapes d'itérations de *Gauss-Seidel* pour ce système soient:

$$x_1=\frac{3{,}1-x_2}{2}$$

$x_i=\frac{5{,}1-x_{i-1}-x_{i+1}}{2}$ Pour i = 2,3,…,5

$$x_6=\frac{2{,}9-x_5}{2}$$

Dans la suite, on note l'indice comme le numéro de la composante du vecteur de solution *x* et l'exposant comme le numéro d'itération pour laquelle la composante est calculée.

Le système converge alors en 23 itérations avec les résultats intermédiaires suivants :

$x^0=\begin{pmatrix}0{,}4283\\0{,}1432\\0{,}2853\\0{,}2860\\0{,}1425\\0{,}4287\end{pmatrix}$, $x^1=\begin{pmatrix}0\\0\\0\\0\\0\\0{,}5\end{pmatrix}$, $x^2=\begin{pmatrix}0{,}5\\0{,}25\\0{,}375\\0{,}3125\\0{,}09375\\0{,}4531\end{pmatrix}$, ……, $x^{23}=\begin{pmatrix}0{,}4283\\0{,}1432\\0{,}2853\\0{,}2860\\0{,}1425\\0{,}4287\end{pmatrix}$

Et en modifiant le vecteur initial $x^0=\begin{pmatrix}10\\10\\10\\10\\10\\10\end{pmatrix}$

Le système converge vers la solution en 41 itérations comme suit :

$$x^0=\begin{pmatrix}10\\10\\10\\10\\10\\10\end{pmatrix}, x^1=\begin{pmatrix}-4.5\\-2.25\\-3.375\\-2.8125\\-3.09375\\2.046\end{pmatrix}, x^2=\begin{pmatrix}1.625\\1.375\\1.2187\\1.4375\\-1.2421\\1.1210\end{pmatrix}, \ldots\ldots\ldots\ldots, x^{41}=\begin{pmatrix}0{,}4283\\0{,}1431\\0{,}2853\\0{,}2860\\0{,}1426\\0{,}4286\end{pmatrix}$$

Les résultats sont vérifiés avec le logiciel SCILAB comme pour tous les calculs d'ailleurs qui seront effectués tout au long de notre travail.

Les algorithmes itératifs sont généralement présentés sous la forme : $x^{i+1}=F(x^i)$, et ils peuvent être parallélisés si x^i peut être divisé en m blocks x^i_k, tel que k appartient a {1,m} et delà on obtient : $x^{i+1}= F(x^i_1,x^i_2,\ldots\ldots\ldots,x^i_k)$

En reprenant le système linéaire illustré par l'exemple précédant, nous pouvons décomposer le vecteur de solution *x* en 2 blocs (m=2) par exemple et affecter le calcul des composantes constituant ces deux blocs à 2 processeurs distincts (n=2), la Figure 4 illustre l'idée de la parallélisation lors du calcul.

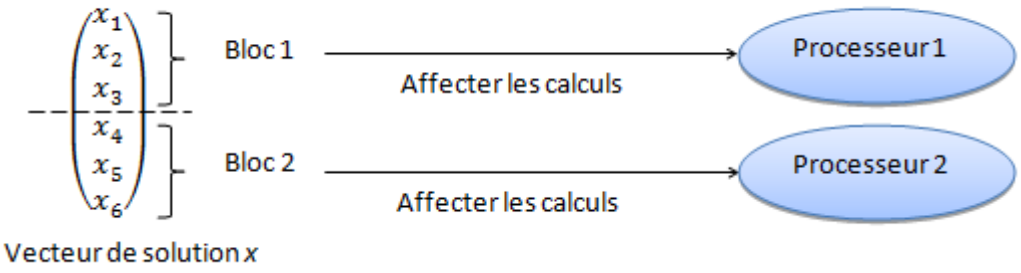


Figure 4 : Exemple de calcul parallèle

3.2 Les différents modes d'exécution parallèles [6]

Il y a le *mode synchrone* et *le mode asynchrone* dans l'exécution parallèle :

3.2. 1 Algorithme avec itération synchrone et communication synchrone(ISCS)

Les processeurs commencent une itération au même temps, le début du calcul local et la diffusion du résultat s'effectue simultanément pour tous les processeurs. Le temps d'inactivité entre les itérations est assez important, la Figure 5 illustre l'algorithme ISCS.

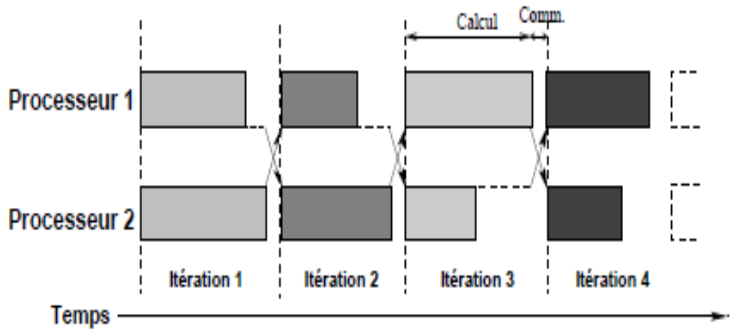


Figure 5 : Algorithme ISCS

3.2.2 Algorithme avec itération synchrone et communication asynchrone(ISCA)

Contrairement au modèle précédent, les résultats des calculs locaux sont transmis de manière asynchrone (aussitôt après la fin du calcul) ce qui réduit considérablement la période d'inactivité des processeurs. La synchronisation des itérations dans ce cas est relative au fait que les processeurs effectuent les mêmes itérations quelque soit le temps ***t***, même si les itérations ne commencent pas au même moment pour les différents processeurs, la Figure 6 illustre l'algorithme ISCA

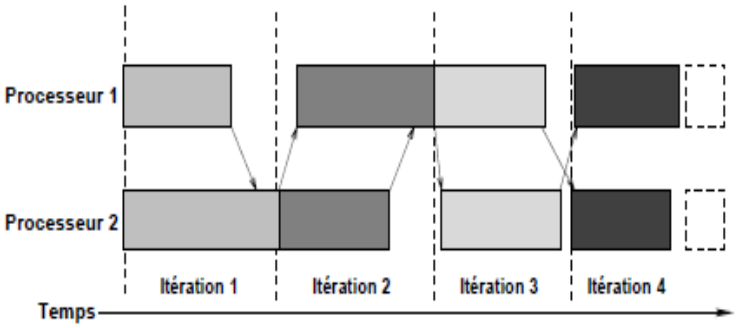


Figure 6 : Algorithme ISCA

3.2.3 Algorithme avec itération asynchrone et communication asynchrone(IACA)

Chaque processeur utilise les dernières valeurs calculées des données pour une nouvelle itération sans avoir à attendre les résultats calculés lors de l'itération précédente, les processeurs peuvent ne pas effectuer la même itération au même moment. Cette classe d'algorithme représente certains avantages qui sont; la suppression de la synchronisation entre les nœuds de calcul, la tolérance de perte de message mais aussi le recouvrement des communications par du calcul, de

ce fait elle est la mieux adaptée pour les méthodes itératives parallèles, qui sont généralement appliquées sur des grilles, la Figure 7 illustre l'algorithme IACA

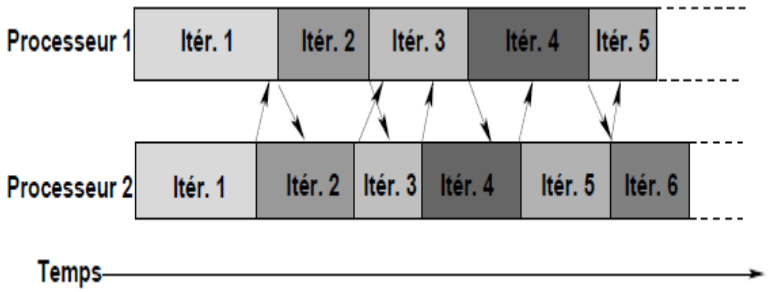


Figure 7 : Algorithme IACA

Etant donnée hétérogénéité des réseaux en terme de débit, puissance de calcul des processeurs, capacité de stockage de la mémoire,…etc. Et afin de ne pas perturber le temps de communication entre les différents nœuds du réseau, il est plus intéressant d'utiliser les méthodes asynchrones lors des calculs sur ce type d'architecture distribuée.

3.3 Modèle mathématique du calcul itératif asynchrone

Le modèle classique des algorithmes asynchrones est présenté [11,7] comme suit :

Soit B un espace de Banach, considéré comme un produit fini d'espace de Banach :

$$B = \prod_{i=1}^{m} B_i$$

Ou m est un entier naturel. La décomposition de tout vecteur $X \in B$ s'écrit de la façon :

$$X=(x_1, x_2,\ldots, x_m)$$

Ou $\forall\, i \in \{1,\ldots m\}$, $x_i \in B_i$

La norme $\|.\|_i$ est associée à chaque espace B_i

Soit F une application de $D(F) \subset B$ à valeurs dans $D(F)$, tel que $D(F) \neq \emptyset$

Le problème de point fixe est défini comme suit : $X=F(X)$, $X^* \in D(F)$

L'algorithme parallèle asynchrone classique (F, x^0, J, s) est défini par :

$$x_l^{t+1} = \begin{cases} \text{Donnée } x^0 = (x_{1,}^0, x_{2,}^0, x_{1,}^0, \dots, x_m^0) \\ \text{Pour } t = 1,2,\dots.. \\ \quad \text{Pour } l = 1, \dots, m \\ \quad \begin{cases} F_l\,(x_1^{s_1^l(t)}, \dots, x_m^{s_m^l(t)}) & \text{Si } l \in J(t) \\ x_l^t & \text{Si } l \notin J(t) \end{cases} \end{cases}$$

Tel que :

- $J(t)$ représente les composantes mises à jour à l'itération t
- $t\text{-}s^l_j(t)$ représente le retard éventuel du au l ème processeur lors du calcul du j ème bloc à l'itération t

- la notation suivante représente la modélisation de l'algorithme de *Gauss-Seidel* par blocs:

$$\begin{cases} s^l_i(t) = t & \text{Pour } t = 1,2,\dots \text{ et i=1,2,}\dots \\ J(t) = 1 + (t \bmod m) & \text{Pour } t=1,2,\dots \end{cases}$$

Hypothèses :

$\forall\, i, \{\, t \in \mathrm{N}, i \in J(t)\}$ est infinie
$\forall\, i, l \in \{1,\dots, \mathrm{m}\}, \forall\, t \in \mathrm{N}, s^l_i(t) \leq t$
$\forall\, i, l \in \{1,\dots.., \mathrm{m}\}, \lim\limits_{\mathrm{P}\to +\infty} s^l_i(t) = +\infty$

Des théorèmes qui concernent la convergence des problèmes numériques et non numériques de points fixe ont été mis au point :

- **Théorème 3.4** (***J. C. Miellou*** (1975, 1981))

Si

(a) $D(F) = \prod_{i=1}^{m} D_i(F)$

(b) $f(D(F)) \subset D(F)$

(c) $\exists\, x^* \in D(F)$, tel que $x^* = F(x^*)$

(d) $\| F(x)-x^* \|_\gamma \leq \beta \| x-x^* \|_\gamma, (0 < \beta < 1)$

$$\|.\|_\gamma = \max \frac{\|.\|_l}{\gamma l}, (\gamma \gg 0)$$

Alors n'importe quel algorithme asynchrone (F, x^0, J, s) converge vers le point fixe x^* de F

Le théorème de Miellou concerne les problèmes numériques en calcul de structure.

Théorème 3.5 (***D.P. Bertsekas*** (1989))

Soit B un espace de Banach, considéré comme un produit fini d'espace de Banach

$$B = \prod_{i=1}^{m} B_i$$

Si

Il existe une séquence d'ensemble non vides $\{B(k)\}$ avec :

$$\ldots \subset B(k+1) \subset B(k) \subset \ldots\ldots \subset B$$

(a) $F(x) \in B(k+1)$, $\forall\, k$ et $x \in B(k)$

(b) $\forall\, k$, il existe $B_i(k) \subset B_i$

Tel que :

$B(k) = B_1(k) \times B_2(k) \times \ldots\ldots\ldots B_m(k)$

Alors n'importe quel algorithme asynchrone (F, x^0, J, s), tel que $x^0 \in B(0)$, converge vers le point fixe x^* de F

Le théorème de Bertsekas trouve ses applications dans les problèmes non numériques et notamment pour le routage distribué dans les réseaux informatiques.

Après nous être assuré que si une méthode converge en mode synchrone elle convergera aussi en mode asynchrone, nous pouvons à présent étudier les méthodes asynchrones et leurs paramètres importants.

III. TESTS ET RESULTATS NUMERIQUES

4 Implémentation et tests

Tout au long de notre étude nous nous intéressons à essayer de comprendre mais aussi tenter de déterminer quel(s) est(sont) le(s) facteur(s) important(s) lors du calcul itératif asynchrone sur une architecture parallèle volatile dans le cadre de la résolution des systèmes linéaires de grandes dimension. En d'autres termes nous essayons de déterminer quels sont les facteurs ayant le plus d'effets sur le calcul itératif asynchrone; retard de calculs sur une ou plusieurs lignes, retard de calcul sur une ou plusieurs itérations, régularité des retards ou non,…etc. Et cela pour pouvoir faire une bonne affectation des taches sur les différents nœuds du réseau; affecter le calcul des composantes qui ont le plus d'impact aux machines les moins susceptible de tomber en panne (les moins exposées au pannes), l'optimisation du calcul est ainsi assurée .

4.1Technique de simulation

Pour notre étude, la résolution de système linéaire de la forme $\boldsymbol{Ax=b}$ se fera par la méthode de *Gauss-Seidel*. On effectuera une succession de test afin d'étudier la convergence du system précédent dans différentes conditions. Le système linéaire sera représenté par des matrices standards de type : creuse, bande,…etc. Et ces matrices seront de différentes tailles.

La programmation de l'algorithme itératif est faite sur une architecture séquentiel à défaut d'utiliser une architecture distribuée, et les pannes des différentes machines qui constituent cette architecture (volatilité du réseau) seront simulées par le retard de calcul d'une ou plusieurs composante(s) x_i du vecteur x (tel chaque composante $\boldsymbol{x_i}$ est affectée a un processeur i). Le retard de calcul sera programmée par le biais de lignes de retards que l'on ajoutera au code.

4.2 Algorithme développé

Comme nous l'avons mentionné précédemment, nous utiliserons l'algorithme de *Gauss-Seidel* dont nous avons détaillé son fonctionnement dans le chapitre précédant. Dans le cadre d'un calcul itératif asynchrone sur les systèmes linéaires de type: $\boldsymbol{Ax=b}$, l'algorithme inclura les retards de calcul des composantes x_i, des tests pour la détection de la convergence mais aussi le nombre d'itérations nécessaires à la convergence et cela pour les différentes conditions d'exécution du programme. Les itérations sont cycliques.

Hypothèse de calcul :

Nous considérons que pour les différents tests effectués, le vecteur x ainsi que le vecteur b sont initialisées à 1.

Le retard de calcul au quelle l'on soumet l'ensemble des composante x_i du vecteur x est le même pour les différentes série de tests et s'exprime par : *le nombre d'itérations sautées/le nombre total d'itérations nécessaire à la convergence* du système.

4.3 Implémentation de l'algorithme

Notre algorithme a été implémentée avec le langage JAVA, et l'IDE utilisée est ECLIPSE version 4.2.2, les déférentes classes utilisées sont les suivantes :

La classe TestGaussSeidelRetardee : C'est la classe principale dans laquelle on précise le nom du fichier de la matrice qui sera utilisée (matrice $\boldsymbol{A}$), ainsi que la marge d'erreur et le nombre d'itérations maximales permises. C'est dans cette classe que s'effectue le retard de calcul de certaines composantes avec l'instanciation de la classe *MarquerCyclique* ainsi que l'appel de la méthode *computeAvecRetard* définis dans la classe *GaussSeidelRetarde* et qui est chargée du calcul iteratif.

La classe GaussSeidelRetarde: Cette classe permet d'effectuer le calcul itératif asynchrone qui permet de résoudre le système linéaire $\boldsymbol{Ax=b}$ et cela en utilisant la méthode suivante :

- La méthode *compute Avec Retard* : Cette méthode prend en paramètre la matrice $\boldsymbol{A}$,les vecteurs b et x_1 (vecteur initial de la solution x), le nombre d'itération ainsi que la marge d'erreur permises et la (les) ligne(s) ou bien composante (s) x_i dont le calcul est retardé pour des itérations précises. Ces paramètres ainsi que la méthode de *Gauss-Seidel* servent à déterminer la solution mais également le nombre d'itérations nécessaires à la convergence du système dans le cas ou ce dernier converge.

La classe MarquerCyclique : Cette classe permet d'établir une représentation cyclique d'une suite de retard ou non à appliquer lors du calcul de la méthode itérative asynchrone. Parmi les méthodes utilisées dans cette classe nous retrouvons:

- La méthode *estRetarde* : Cette méthode teste si une ligne précise est à retarder pour une itération précise.
- La méthode *retarder*: Cette méthode permet de retarder des lignes précises pour des itérations précises en utilisant l'instanciation de la classe *CyclicArrayOfBoolean*.

La classe CyclicArrayOfBoolean : Cette classe permet de simuler un tableau cyclique dont la taille est le nombre total des itérations successives effectuées lors du calcul. Le cycle est à fixer dans la classe *TestGaussSeidel.*

4.4 Les données

Les systèmes linéaires qui sont résolus par des méthodes itératives sont représentés par des matrices standards.

4.4.1 Rappel sur les matrices

Définition d'une matrice creuse : Une matrice creuse est une matrice contenant beaucoup de zéro.

Définition d'une matrice bande : Une matrice bande est un cas particulier de matrice creuse, tel que les composants non nuls se regroupent autour de la diagonal

Définition d'une matrice Diagonale : Une matrice diagonale est une matrice carrée dont les coefficients en dehors de la diagonale principale sont nuls. La Figure 8 illustre la forme générale d'une matrice diagonale.

$$diag(a_1, a_2, ..., a_k) = \begin{pmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_k \end{pmatrix}$$

Figure 8 : Matrice diagonale

Définition d'une matrice triangulaire : Une matrice triangulaire est une matrice carrée dont une partie triangulaire des valeurs, délimitée par la diagonale principale est nulle. La Figure 9 illustre un exemple général d'une matrice triangulaire supérieure.

$$A = (a_{i,j}) = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & \cdots & a_{1,n} \\ 0 & a_{2,2} & & & a_{2,n} \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & a_{n,n} \end{bmatrix}$$

Figure 9 : Matrice triangulaire supérieure

Les matrices triangulaires supérieures ou inferieurs ainsi que les matrices diagonales sont des matrices Bandes

Nous avons entamé la phase de test avec une matrice bande assez simple que nous avons-nous même construit ; qui est la matrice présentée précédemment et de taille (**6*6**). Mais l'étude de la convergence n'a pas été très intéressante sur cette matrice du fait que le nombre d'itérations nécessaires à la convergence ne soit pas très grand, pour cette raison nous nous sommes tournées vers des bibliothèques beaucoup plus riches.

4.4.2 Bibliothèques de matrices

Dans le cadre de notre étude nous nous sommes intéressés à différentes bibliothèques de matrices, aussi diversifiées les unes que les autres et qui fournissent des matrices de différentes tailles et de différentes structures et cela pour l'étude de différents problèmes. Parmi les bibliothèques étudiées nous citons :

La Matrix Market Collection

Cette collection représente un référentiel de données de test à utiliser dans des études dans le cadre de l'algèbre linéaire numérique [1] et fournit 500 matrices creuses issues de différentes applications. Des outils et des services de génération de matrice aussi sont aussi mis au service de l'utilisateur.

Son interface est assez basique et la recherche de matrice se fait selon différents critères comme par exemple: la collection a laquelle appartient la matrice (la Harwell-Boeing collection, la SPARKSIT collection), son nom, ses propriétés (nombre de zéro, taille, forme …….), les domaines d'applications dans lesquelles elle est utilisée (la biochimie, l'économie, le control de trafic aérien, programmation linéaire, les circuits physique), les différentes sources (industriel, gouvernemental, académique),…ect.

De la documentation est aussi fournie afin de faciliter au mieux l'utilisation des matrices qui sont fournies. La documentation concerne aussi bien le format des fichiers de matrice fourni (fichier compressée), mais aussi des Tracés de structure qui permettent de visionner la structure et la densité d'une matrice donnée, ….etc.

Nous présentons ci-dessous un exemple de matrice répertoriée par cette bibliothèque et dont le tracé de sa structure est représenté par la Figure 10 comme suit :

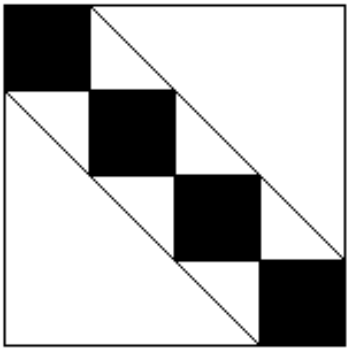


Figure 10 : Matrice qc324

La matrice *qc324* est une matrice bande par blocks de taille (324*324) qui appartient à la NEP collection (Non-Hermitian Eigenvalue Problem).

La University of Florida Sparse Matrix Collection

En accédant à cette bibliothèque [2], l'utilisateur notera aux premiers abords la différence du point de vue interface. En effet celle-ci est beaucoup plus agréable en comparaison avec la bibliothèque présentée précédemment, s'ajoute à cela les mises à jour qui sont beaucoup plus récente ici.

La collection de matrices creuses de l'université de Floride présente plus **2500** matrices relatives à différents problèmes. Les matrices sont disponibles sous différents formats : *MATLAB mat-file*, *Rutherford-Boeing*, et *Matrix Market* et c'est en ce dernier format que les matrices seront téléchargées dans le cadre de notre travail, ce qui donnera des fichier dont l'extension est : « *.mtx* » .

La manipulation de cette collection est d'autant plus facile grâce à l'interface *UFgui Java interface* dont l'aperçu est illustré par la Figure 11.Cette interface est mise à la disposition de l'utilisateur et lui permet de visualiser mais aussi de télécharger les matrices.

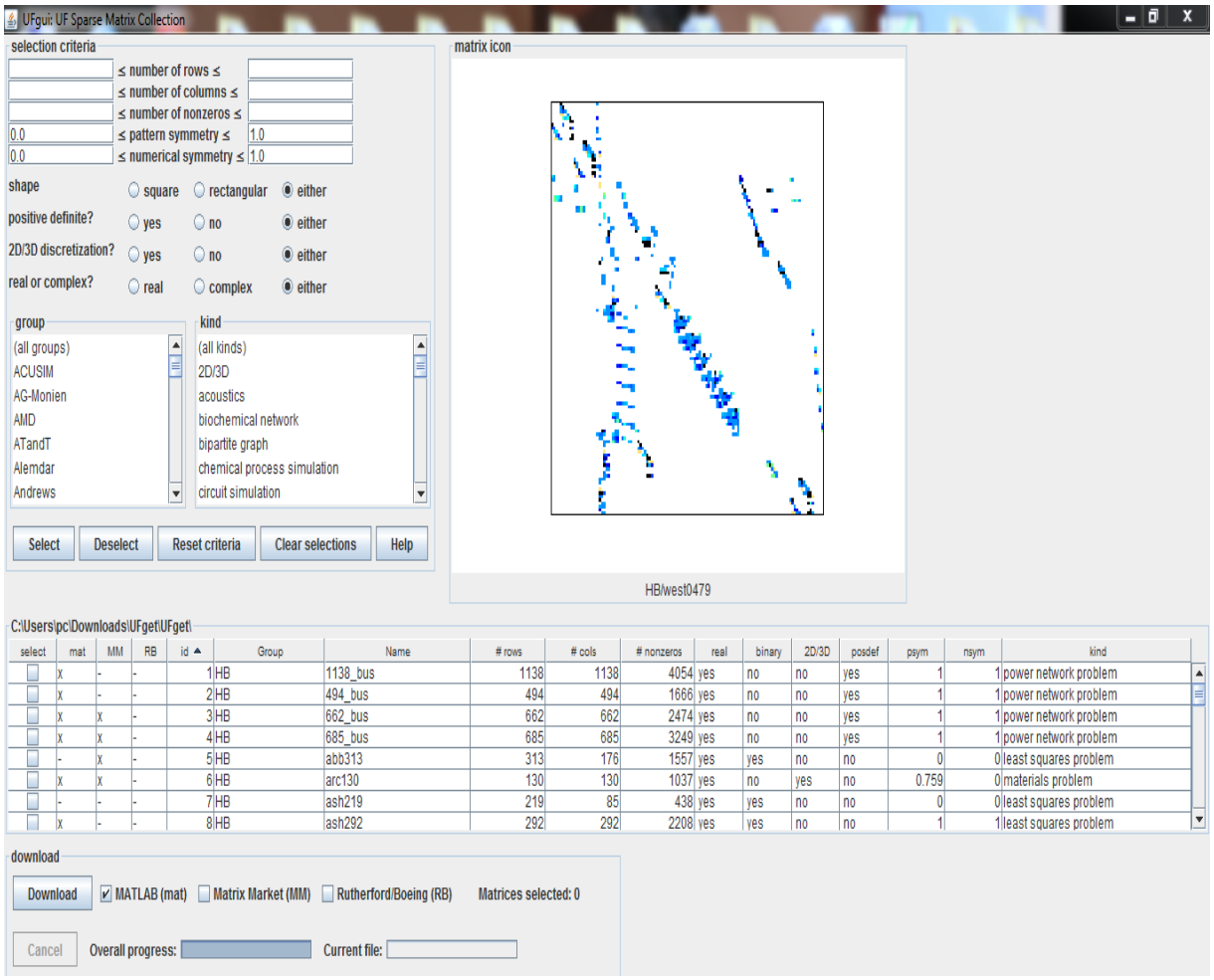


Figure 11 : Interface UFgui

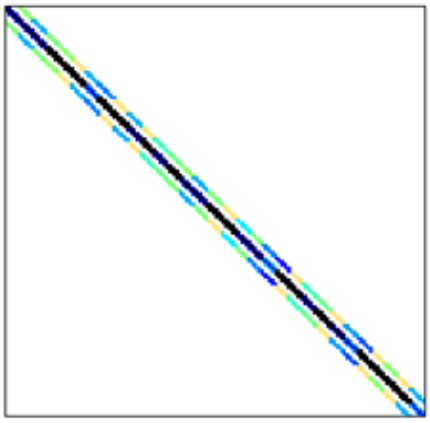
De cette interface on distingue clairement que les matrices sont répertoriées selon différents critères de classification qui sont beaucoup plus diversifiées que ceux de la bibliothèque citée précédemment :

Classification des matrices :

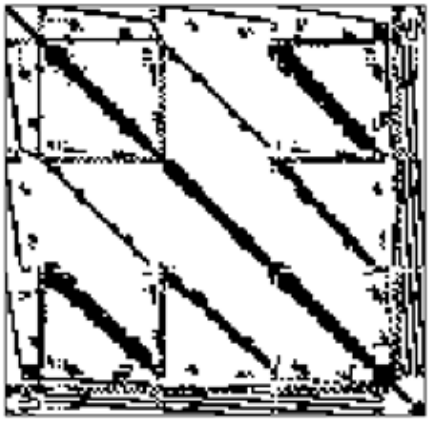
Des critères très variés sont utilisés pour répertorier les matrices de cette bibliothèque comme par exemple: l'identifiant ou bien le nom de la matrice, le groupe auquel appartient la matrice, le nombre de lignes et de colonnes qui constituent la taille de la matrice, le nombre d'éléments de la matrice différents de zéro, le degré de symétrie de la matrice,…etc.

Nous présentons ci-dessous quelques matrices répertoriées selon le type de problèmes pour lesquels elles sont utilisées :

a) Type problème Mécanique des structures :



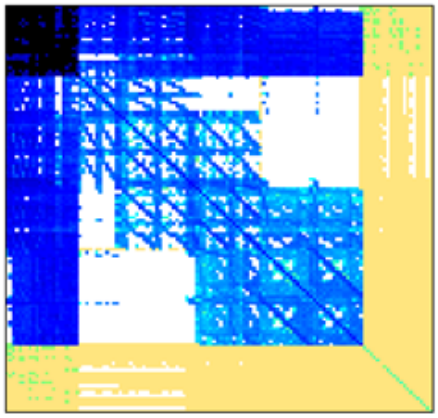
Matrice *Transport*



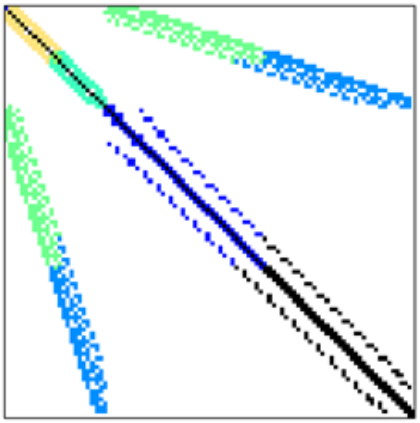
Matrice *fullb*

Figure 12 : Matrices pour problèmes de la mécanique des structures

b)Type problème électromagnétique :



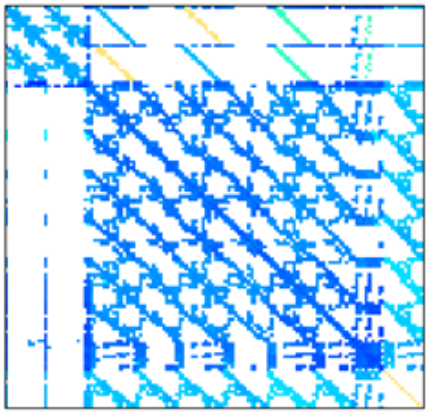
Matrice *gsm_106857*



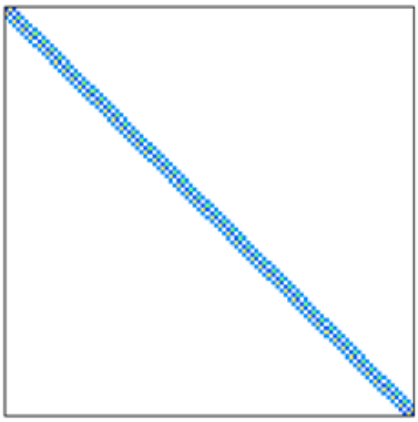
Matrice *bfwb398*

Figure 13 : Matrices pour problèmes électromagnétiques

c)Type problème dynamique des fluides :



Matrice *invext1_new*



Matrice *tub100*

Figure 14 : Matrice pour problèmes dynamiques des fluides

4.4.3 Choix de matrice

Après avoir passé en revue un certain nombre de ces matrices, nous choisissons pour nos tests quelques unes qui soient réalistes et calculables en un temps réalisable. Nous nous interessons aussi à celles qui sont assez creuses et avec une certaine structure comme par exemple les matrice bandes, et les matrices symétriques, et cela afin de faciliter les calculs effectués lors de l'application de la méthode itérative.

4.5 Tests et interprétations des résultats

Tout au long de notre étude nous émettons différentes hypothèses concernant le calcul itératif asynchrone et l'effet que peut avoir le retard de calcul d'une ou de plusieurs composantes sur la détection de la convergence lors de la résolution du système linéaire : $\boldsymbol{Ax=b}$. Ces hypothèses sont vérifiées au fur et à mesure tout au long de notre étude.

Le nombre d'itération pour lesquelles une ou plusieurs composantes ne sont pas calculées restent le même tout au long de l'ensemble des tests que nous effectuons.

4.5.1 Première approche basée sur des tests aléatoires

Cette première approche consiste à effectuer un certain nombre de tests sur différentes matrices pour essayer de déterminer quelles sont les composantes importantes lors du calcul.

Série de tests sur la première matrice

Pour commencer, nous effectuons une série de test sur la matrice bande de petite taille (6*6) que nous avons construit nous-mêmes lors d'une précédente utilisation et que nous rappelons à travers la Figure 15 :

$$A=\begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

Figure 15 : Matrice Bande de taille (6*6)

On a vue que l'utilisation d'un algorithme itératif de type *Gauss-Seidel* pour résoudre ce système linaire convergence en 23 itérations.

Retard de calcul des composantes x_i et l'effet sur la détection de la convergence

Retarder le calcul d'une seule composante à la fois

Nous commençons par retarder le calcul au hasard de la première composante x_1 (choisie au hasard). Les résultats sont résumés dans la Figure 16 comme suit :

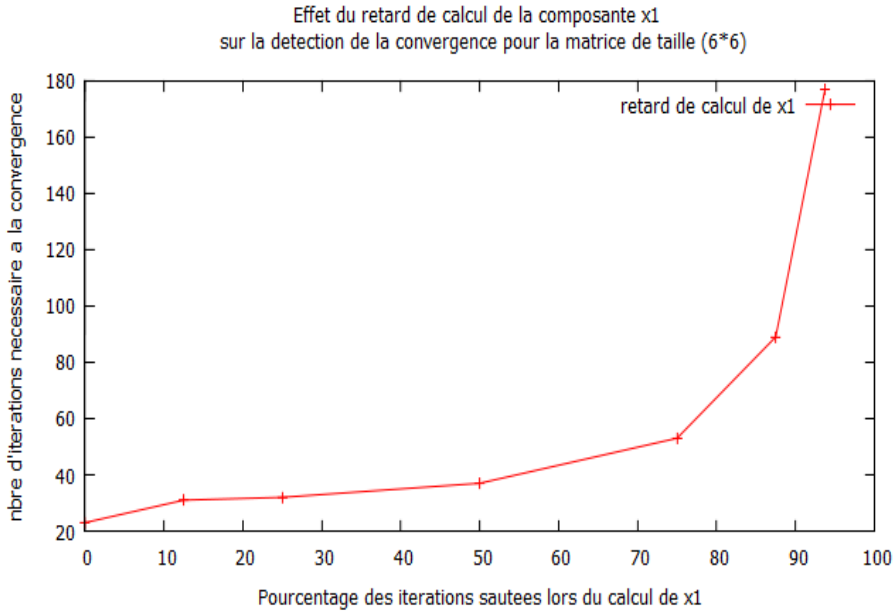


Figure 16 : Effet du retard de calcul de la composante xi dans la détection de convergence dans une matrice de taille (6*6)

On voit sur cette figure que plus le nombre d'itérations pour lesquelles la composante x_1 n'est pas calculée augmente, plus le nombre d'itérations nécessaire à la convergence augmente, ce qui est en somme logique.

Nous retardons pour le même nombre d'itérations le calcul de la composante de l'autre extrémité x_6, et constatons que l'effet est relativement similaire que lorsqu'on retarde la composante x_1.

A présent nous choisissons de retarder la composante du milieu x_3. Nous constatons une très grande différence par rapport aux résultats obtenus précédemment. La Figure 17 illustre les résultats obtenus :

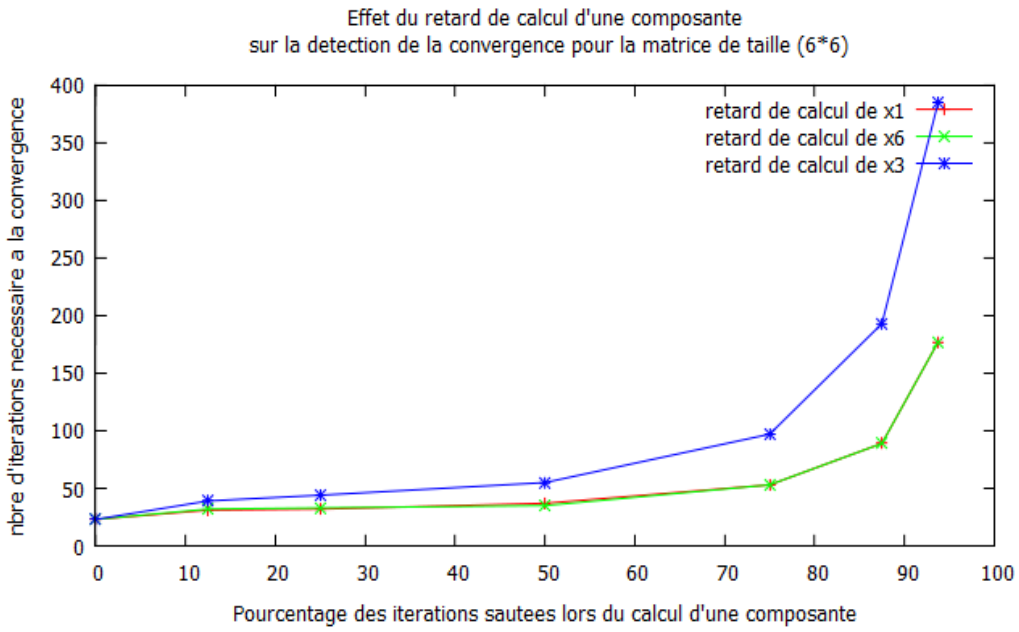


Figure 17 : Effet du retard de calcul de la composante x1, x3, x6 dans la détection de convergence dans la matrice de taille (6*6)

Hypothèse 1 :

De ces premiers tests, nous notons que pour une matrice creuse et de petite taille, le retard de calcul de la composante du milieu implique plus d'effet (augmentation du nombre d'itérations nécessaires à la convergence) que lorsqu'il s'agit des premières ou mêmes dernières composantes du vecteur *x*. Cela dit les premières composantes ont plus de 'poids' dans le calcul itératif asynchrone que les dernières composantes.

Série de test sur la deuxième matrice :

Nous tentons de vérifier l'hypothèse émise précédemment mais cette fois ci avec une matrice dont la taille est plus grande.

a) Présentation de la matrice :

La matrice *bfw62b* est une matrice de taille (62*62), elle est symétrique et assez creuse. En effet elle contient 342 éléments différents de 0. Elle fait partie des matrices utilisées pour résoudre des problèmes électromagnétiques, sans le moindre retard dans le calcul on note une convergence du système en 22 itérations. Son tracé de structure est représenté par la Figure 18 :

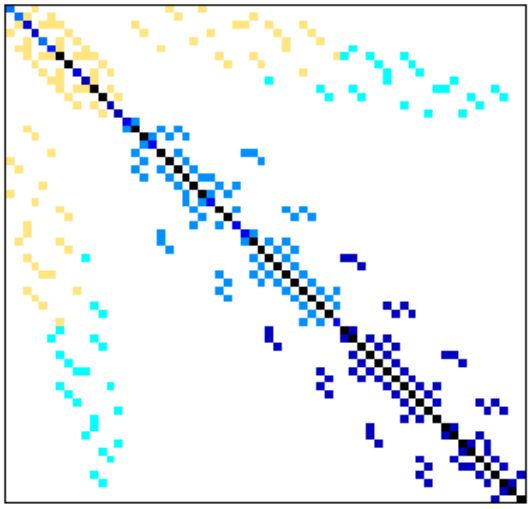


Figure 18 : Tracée de structure de la matrice bfwb62

b) Retard de calcul des composantes x_i et l'effet sur la détection de la convergence :

Retarder le calcul d'une seule composante à la fois :

Nous retardons pour cette matrice le calcul de x_1, x_{62} et x_{30} chacune à la fois, les résultats obtenus quant au nombre d'itérations nécessaires à la convergence sont représentés par la Figure 19 comme suit :

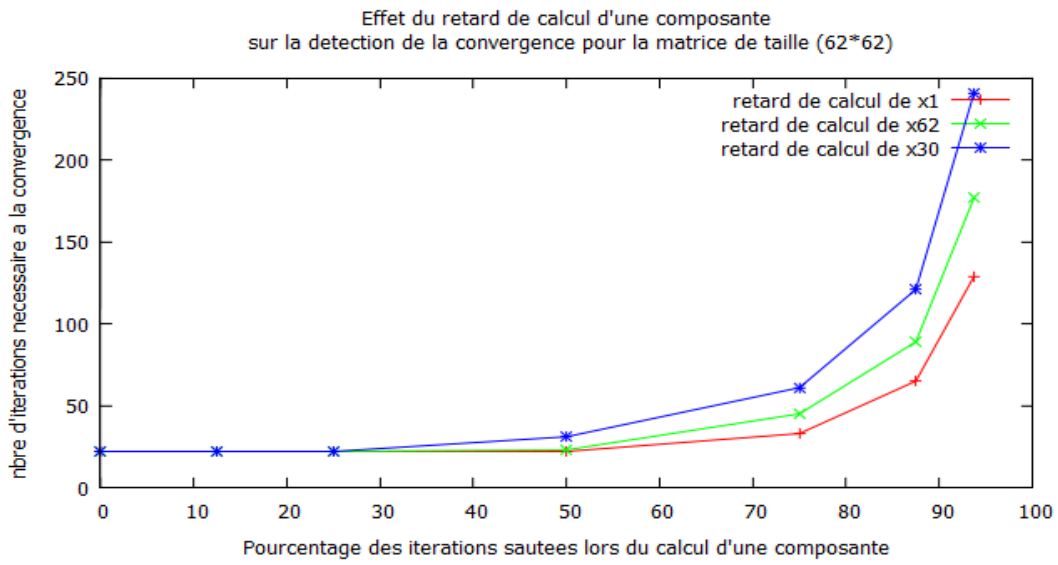


Figure 19 : Effet du retard de calcul d'une composante xi dans la détection de convergence dans une matrice de taille (62*62)

On voit sur cette figure que l'on obtient des résultats différents avec une matrice de taille plus grande, et cela dans la mesure où la dernière composante a plus d'impact que la première dans le calcul, mais la composante du milieu reste encore plus puissante que ces deux composantes de l'extrémité.

Retarder le calcul de plusieurs composantes à la fois :

Pour tester l'effet d'une composante du milieu par rapport à celles de l'extrémité, nous décidons de retarder le calcul de plusieurs composantes au même temps et notons l'effet que cela engendre sur la détection de la convergence. Les résultats obtenus sont représentés graphiquement à travers la Figure 20 comme suit :

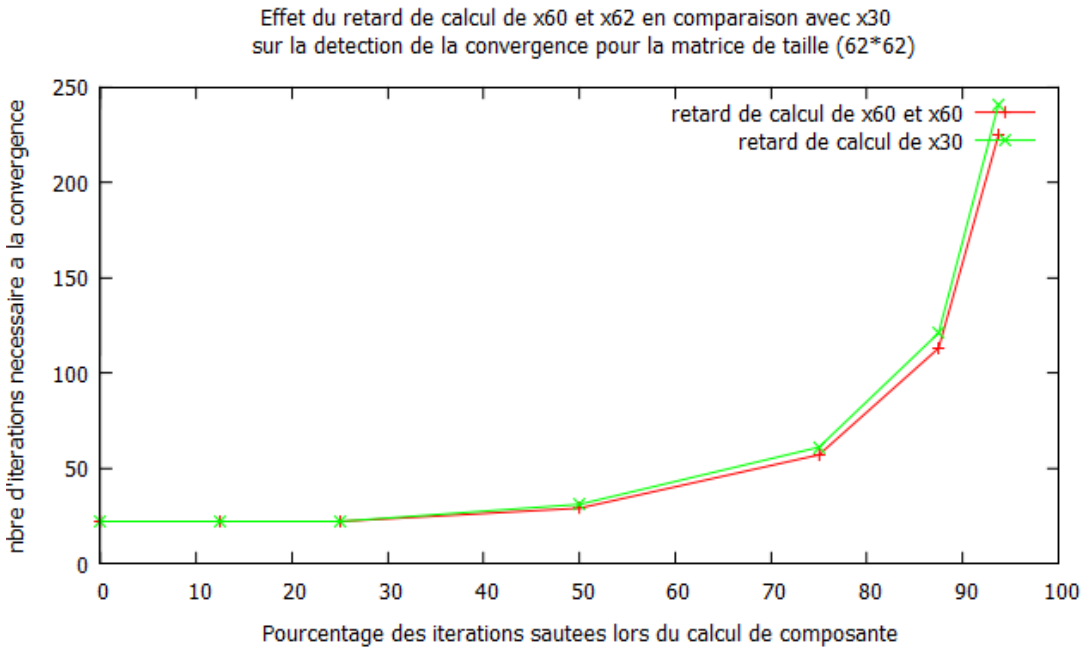


Figure 20 : Effet du retard de calcul de plus d'une composante xi dans la détection de la convergence

Nous constatons d'après cette figure qu'en retardant le calcul des deux dernières composantes au même temps, l'effet reste légèrement inferieur en comparaison avec celui résultant du retard de calcul appliqué seulement à la composante x_{30}, et cela malgré que les composantes dont le calcul est retardé aient beaucoup d'importance chacune dans la détection de la convergence du system.

Hypothèse 2 :

A partir des résultats de tests obtenus avec les deux matrices de différentes tailles utilisées, nous émettons l'hypothèse suivante :

Les composantes x_i du milieu du vecteur x sont celles dont le retard de calcul implique le plus d'itérations pour atteindre la convergence, s'ajoute à cela :

Si la matrice (matrice $\boldsymbol{A}$ du système linéaire $\boldsymbol{Ax=b}$) est de petite taille, alors les premières composantes auront plus de poids que les dernières dans la détection de la convergence et si la taille de la matrice grandit, dans ce cas ça sera les dernières composantes x_i qui auront le plus d'importance dans le calcul.

Nous entamons une troisième série de test avec une matrice dont la taille est beaucoup plus grande et cela afin de vérifier l'impact des dernières composantes ainsi que celle du milieu comme nous l'avons supposé précédemment.

Série de test sur la troisième matrice :

Nous enchainons nos tests avec la matrice suivante :

a) Présentation de la matrice :

La matrice sur laquelle portera notre troisième série de test est la matrice *Saylr1* qui est de taille (238*238). Elle fait partie des matrices utilisées pour résoudre les problèmes de la dynamique des fluides, avec cette matrice et sans le moindre retard de calcul, le system converge en 692 itérations. La Figure 21 représente le tracée de structure de cette matrice :

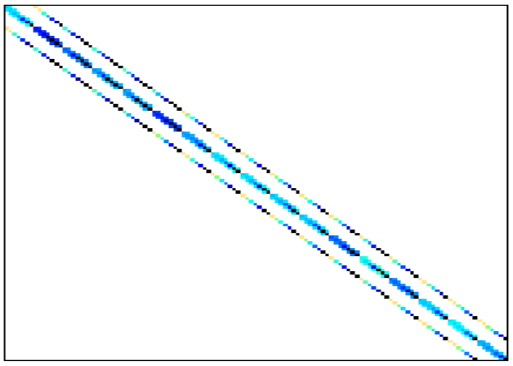


Figure 21 : Tracée de structure de la matrice Saylr1

b) Retard de calcul des composantes x_i et l'effet sur la détection de la convergence :

Retarder le calcul d'une seule composante à la fois :

Les résultats concernant le retard de calcul appliqué individuellement pour différentes composantes x_i et son effet sur la détection de la convergence sont illustrés dans la Figure 22 comme suit :

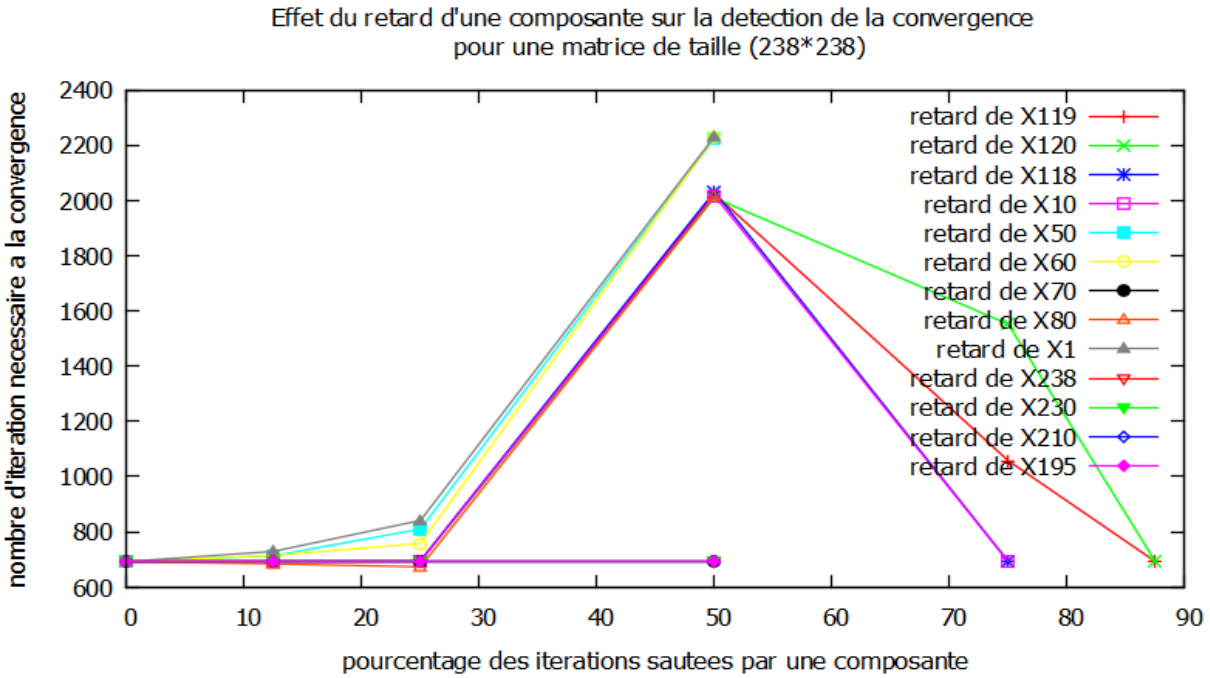


Figure 22 : Effet du retard de calcul de différentes composantes xi dans la détection de la convergence

On voit sur cette figure, que le retard de calcul des premières composantes a beaucoup plus d'effet sur la détection de la convergence en comparaison avec celui des composantes du milieu. Cet effet est d'autant plus grand si on le compare à celui du retard de calcul des dernières composantes du vecteur de *x*, de ce fait ces résultats contredisent l'hypothèse que nous avons émis précédemment (hypothése2) et qui concerne l'importance des composantes dans le calcul selon la taille de la matrice utilisée pour le système linéaire.

Nous détaillons dans la Figure 23 l'effet engendré par un retard de calcul individuel d'un plus grand nombre de composantes en nous concentrant sur les premières composantes ainsi que celle du milieu.

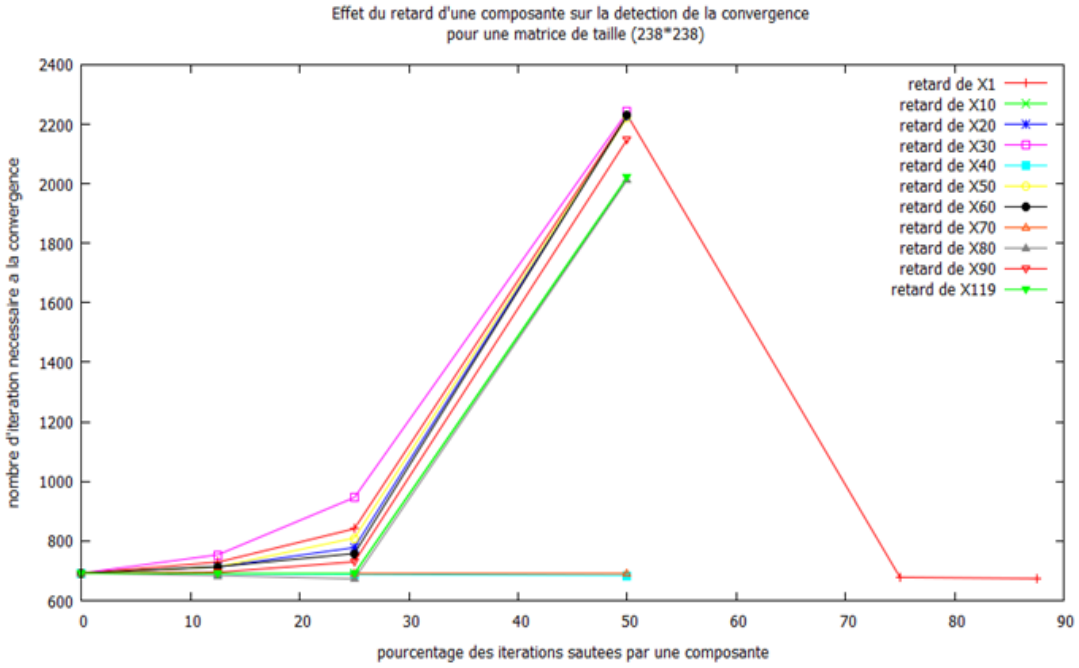


Figure 23 : Effet du retard de calcul de différentes premières composantes dans la détection de la convergence

De cette figure nous constatons qu'effectivement, le retard de calcul d'une composante parmi les premières a beaucoup plus d'impact que celui d'une composante du milieu et cela sur la détection de la convergence, dans le sens ou plus d'itérations sont nécessaires pour détecter la convergence.

Pour cette matrice de taille (238*238), et à ce niveau de tests les hypothèses émises précédemment ne sont pas vérifiées, nous continuons cela-dit avec d'autres tests sur cette même matrice.

Retarder le calcul de plusieurs composantes à la fois :

Nous allons à présent retarder le calcul de plusieurs composantes parmi les dernières et cela au même temps afin de comparer l'effet engendré avec celui résultant du retard de calcul d'une composante importante (x_{30}) dans la détection de la convergence. La Figure 24 représente les résultats obtenus :

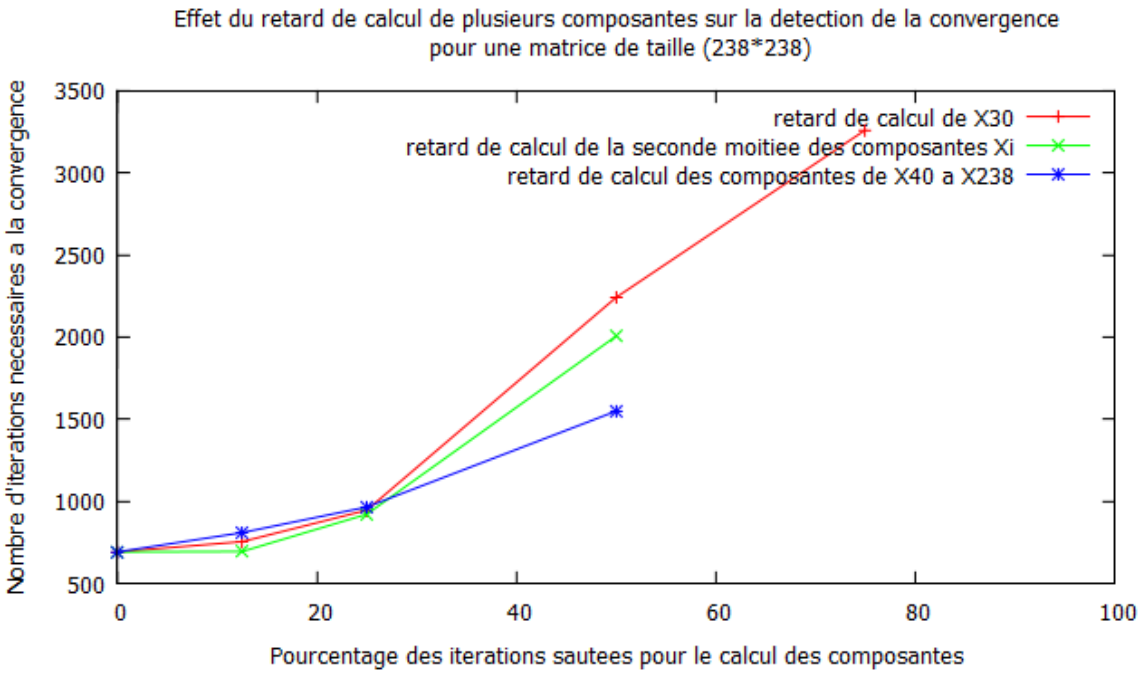


Figure 24 : Effet du retard de calcul de plusieurs composantes xi dans la détection de la convergence dans la matrice de taille (238* 238)

On voit de cette figure, que pour la matrice de taille (238*238), le retard de calcul de tout un bloc de dernières composantes n'a pas autant d'effet sur la détection de la convergence que le retard de calcul d'une seule composante très puissante dans le calcul.

En conclusion à cette première série de tests effectuée, nous n'avons pas obtenu de résultats concluants car nous n'avons pas pu définir quelles sont les composantes x_i importantes lors du calcul qui a pour but de résoudre le système linéaire $\boldsymbol{Ax=b}$ et cela pour des matrices de différentes tailles, mais aussi de différentes structures, ce qui nous pousse a réfléchir à une autre approche.

4.5.2 Approche de test basée sur la variation des valeurs des composantes pour des itérations successives :

Nous tentons une nouvelle fois de détecter la composante x_i du vecteur x, dont le retard de calcul a le plus d'impact sur la détection de la convergence et pour cela nous enchainons avec une autre série de test qui est basé sur la variation des valeurs des composantes. Et cela pour un certain nombre d'itération, tout en utilisant les matrices précédentes. Nous espérons à travers cette nouvelle approche, pouvoir définir l'importance d'une composante x_i dans le calcul et cela en fonction des fréquences de changement de sa valeur pour certaines itérations successives.

Nous effectuons une série de test pour chacune des matrices qui ont été testées dans l'approche précédente.

Série de test sur la première matrice

Nous commençons la première série de test avec la matrice présentée précédemment dans la Figure 15 qui est de taille (6*6). Les résultats obtenus sont regroupées dans la Table 1 comme suit :

Différence moyenne de valeurs pour les **5 premières itérations**	Différence moyenne de valeurs pour les **5 dernières itérations**	Différence moyenne de valeurs pour les **10 dernières itérations**	Différence moyenne de valeurs pour **les itérations de 6 à 15**	Différence moyenne de valeurs pour **toutes les itérations**
x_1 : 0.1145 x_3 : 0.7942 x_6 : 0.09391	x_1 : $6.9092*10^{-5}$ x_3 : $1.2605*10^{-4}$ x_6 : $4.1031*10^{-5}$	x_1 : $1.4353*10^{-4}$ x_3 : 2.6360 x_6 : $8.5982*10^{-5}$	x_1 : $6.9804*10^{-4}$ x_3 : $9.5494*10^{-4}$ x_6 : $4.4850*10^{-4}$	x_1 : 0.02932 x_3 : 0.02035 x_6 : 0.02380

Table 1 : Résultats des tests pour la matrice de taille (6*6)

D'après les résultats affichés dans le Table 1 et ceux des tests effectués lors de l'approche précédente sur cette même matrice, nous admettons que : les composantes dont les valeurs changent le plus durant les dernières itérations, sont celles dont le retard de calcul a le plus d'effet sur la détection de la convergence.

Série de test sur la deuxième matrice :

Nous effectuons la même série de test mais cette fois-ci c'est avec la matrice illustrée par la Figure 18 qui est de taille *(62*62)*.Les résultats sont illustrés dans la Table 2 comme suit :

Différence moyenne de valeurs pour les **5 premières itérations**	Différence moyenne de valeurs pour les **5 dernières itérations**	Différence moyenne de valeurs pour les **10 dernières itérations**	Différence moyenne de valeurs pour **les itérations de 6 à 15**	Différence moyenne de valeurs pour **toutes les itérations**
x_1 : 17567,4249 x_{30} : 3370, 4147 x_{62} : 2969,2086	x_1 : 17095,008 x_{30} : 1994,6443 x_{62} : 2965,9887	x_1 : 9324,5512 x_{30} : 1088,0096 x_{62} : 1617,8126	x_1 : 3,4720 x_{30} : 24,9910 x_{62} : 0,8189	x_1 : 8667,1996 x_{30} : 1352,7203 x_{62} : 1484,1747

Table 2 : Résultats des tests pour la matrice de taille (62*62)

D'après les résultats mentionnés dans le tableau mais aussi d'après ceux obtenus avec les tests effectués lors de l'approche précédente sur cette même matrice, nous admettons que lorsque la taille de la matrice augmente, les dernières itérations du calcul sont toujours celles qui permettent de définir quelles sont les composantes qui ont le plus d'impact sur le calcul, mais contrairement à la matrice précédente qui est de taille plus petite; les composantes dont les valeurs changent le

plus lors des dernières itérations, sont celles dont le retard de calcul a le moins d'effet sur la détection de la convergence

Série de test sur la troisième matrice

Nous refaisons la même série de test cette fois ci avec une matrice de taille encore plus grande, il s'agit de la matrice présentée précédemment par la Figure 21 qui est de taille *(238*238),* et les résultats obtenus sont résumées dans le Table 3 comme suit :

Différence moyenne de valeurs pour les **5 premières itérations**	Différence moyenne de valeurs pour les **5 dernières itérations**	Différence moyenne de valeurs pour les **10 dernières itérations**	Différence moyenne de valeurs pour **les itérations de 6 à 15**	Différence moyenne de valeurs pour **toutes les itérations**
x_1 : 0.01956 x_{30} : 0.01488 x_{62} : 0.0099	x_1 : 0.001887 x_{30} : 0.003069 x_{62} : 0.009731	x_1 : $7.1663*10^{-4}$ x_{30} : 0.001102 x_{62} : 0.003273	x_1 : $1.7137*10^{-4}$ x_{30} : $1.8335*10^{-4}$ x_{62} : $1.6388*10^{-4}$	x_1 : 0.0032 x_{30} : 0.002777 x_{62} : 0.002885

Table 3 : Résultats des tests pour une matrice de taille (238*238)

Les résultats des tests quant à cette matrice sont semblables à ceux de la matrice qui est de taille (62*62) (les composantes dont les valeurs changent le plus durant les dernières itérations sont celles dont le retard de calcul a le moins d'effet sur la détection de la convergence).De ce point en commun entre ces deux matrices nous pourrions arriver à un résultat intéressant. Seulement lorsqu'on teste une autre matrice (la matrice *tub100* de taille (100*100), dont le tracé de structure est représenté par la Figure 15 et qui converge en 3023 itérations**)**, nous notons que les composantes dont les valeurs varient le plus pour les 523 dernières itérations, sont celles qui ont le plus d'impact lors du calcul itératif.

Nous avons aussi étudié les variations de valeurs pour les différentes composantes des matrices précédentes et cela lors de la première moitié des itérations effectuées ainsi que pour la seconde moitié, mais les résultats obtenus ne sont pas très significatifs.

En conclusion à cette approche, qui consiste à essayer de déterminer l'importance des composantes x_i lors de la résolution de système linéaire et cela en fonction des variations des valeurs de composantes pour certaines itérations successives; nous ne somme pas arrivés à une conclusion intéressante concernant les matrices de façon générale et encore moins concernant les matrices que nous avons utilisé pour notre étude.

4.5.3 Approche adoptée

Nous avons effectué différents tests lors des approches précédentes afin de prévoir les composantes importantes et celles qui le sont moins lors du calcul pour la résolution du système linéaire de la forme: ***Ax=b*** avec la méthode *Gauss-Seidel*. Et tout cela afin de faire une bonne affectation des taches pour le calcul sur les différentes machines de l'architecture parallèle, mais malheureusement ces tests n'ont pas révélé un grand résultat. Nous tentons alors l'approche suivante, qui consiste à considérer le retard cumulé lors du calcul d'une composante x_i, et le repartitionner pour le calcul des différentes composantes qui sont affectées au machines constituants l'architecture parallèle, en d'autres termes au lieu de retarder plusieurs fois le calcul d'une seule et même composante x_i (retard qui en réalité est causé par la défaillance de la machine à la quelle est affecté le calcul de la composante x_i), différentes composantes seront retardées pour un même nombre de retard de calcul. Ceci revient à simuler le changement plus ou moins régulier des localisations des calculs des composantes dans la grille de calcul, de façon à ce que les retards que pourraient causer certains processeurs soient répartis dans tous les calculs et non pas concentrés sur une seule composante.

Nous considérons que le retard de calcul des composantes dans le cadre de la nouvelle approche adoptée est assez grand, il équivaut à **9/10** itérations ratées, par nécessitée et dans le cadre d'un scenario adopté pour cette approche et que nous verrons plus en détails par la suite, nous sommes contraints à déterminer pour chaque matrice étudiée, la composante x_i qui a le plus d'effet sur le calcul (en simulant des retards de calcul de **3/4** itérations ratées).

Série de test sur la première matrice :

Nous commençons notre première série de test pour cette nouvelle approche avec la matrice *bfwb62* qui est de taille (62*62) et qui a été présentée précédemment.

Retarder le calcul d'une seule composante :

Nous retardons individuellement le retard de calcul des composantes x_1 et x_{62} pour 9/10 itérations chacune, et comparons les résultats obtenus avec le même retard de calcul mais appliqué à la composante qui a le plus d'effet sur le calcul pour cette matrice (x_{42}), les résultats obtenus sont représentées dans la Table 4 comme suit :

Nbre d'itérations nécessaire a la convergence pour le retard de calcul de x_{42} pour **9/10**	Nbre d'itérations nécessaire a la convergence pour le retard de calcul de x_{62} pour **9/10**	Nbre d'itérations nécessaire a la convergence pour le retard de calcul de x_1 pour **9/10**
161 itérations	111 itérations	81 itérations

Table 4 : Résultats du retard de calcul d'une composante pour matrice de taille (62*62)

On voit d'après ce tableau qu'appliquer un grand retard de calcul sur une composante x_i influe considérablement sur le nombre d'itérations nécessaires à la convergence, et cet effet est d'autant plus grand lorsque s'agit de retarder la composante qui a le plus d'effet dans le calcul, ce qui est en somme logique.

Combiner le retard de calcul de plusieurs composantes à la fois :

Nous voulons comparer l'application d'un retard total sur une seule composante, au même retard total divisé sur plusieurs composantes. Dans les exemples qui suivent, on comparera plus précisément un retard de 9/10 itérations pour la composante la plus importante, avec un retard réparti de 8/10 itérations pour cette composante et du reste du retard (1/10) pour l'une des autres plus importantes composantes. Ceci revient en fait à simuler par exemple que certains calculs de composantes changent de processeurs en cours de route, ce qui fait que une composante souffrira moins de retard en permanence. Nous appellerons pire scénario possible dans ce cas la situation dans laquelle la composante la plus importante est tout de même à peine moins retardée (8/10 au lieu de 9/10 itérations ratées) et le retard (1/10) est reporté sur une autre composante aussi importante ou presque.

Nous appliquons à présent le retard de calcul pour la composante x_{42} et avec d'autres composantes au même temps. Les résultats obtenus de ces tests numériques sont résumés dans la Table 5 comme suit :

Nbre d'itérations nécessaire à la convergence pour le retard de calcul de x_{42} pour 9/10 itérations	Nbre d'itérations nécessaire a la convergence en retardant le calcul de x_{42} pour 1/10 itérations et 8 composantes voisines pour 8/10 itérations.	Nbre d'itérations nécessaire a la convergence en retardant le calcul de x_{42} pour 1/10 itérations et x_{40} pour 8/10 itérations
161 itérations	24 itérations	72 itérations

Table 5 : Résultats du retard de calcul de plusieurs composantes pour matrice de taille (62*62)

A partir de ce tableau nous constatons qu'en retardant la composante la plus puissante dans le calcul (x_{42}) au même temps qu'en retardant ses différentes composantes voisines les plus proches (qui elles aussi ont un certain poids dans le calcul), cela a beaucoup moins d'effet que de retarder son calcul à elle seule pour plusieurs itérations successives pendant le calcul.

Nous poussons nos tests un peu plus loin pour cette matrice en envisageant le pire scenario qui puisse arriver si on opte pour cette approche qui pour rappel consiste à répartir l'ensemble des retards de calcul qui peuvent survenir lors du calcul parallèle, et cela sur les différents processeurs auxquels on affecte le calcul des différentes composantes x_i du vecteur de solution x.

Le pire scenario en question consiste à ce que le calcul de la composante x_{42} soit retardé pour 8/10 itérations et que l'une de ses composantes voisines pour laquelle l'effet du retard de calcul est assez similaire au sien, (x_{46} par exemple) soit retardé pour 1/10 itérations. Le résultat obtenu est représenté à travers la Figure 25 :

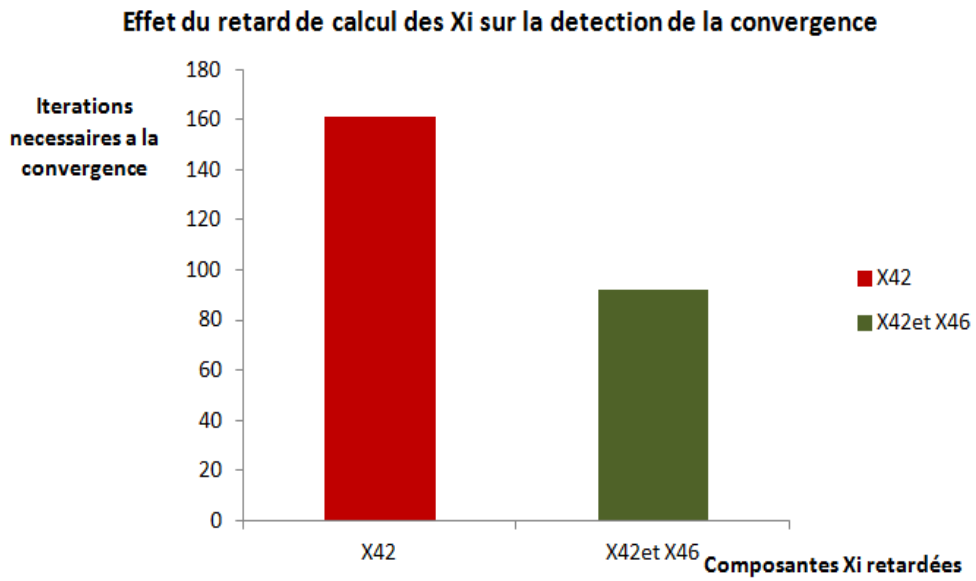


Figure 25 : Effet du retard de calcul d'une seule composante comparée a celui d'une combinaison de composantes xi dans la détection de la convergence dans la matrice de taille (62*62)

De la figure nous voyons que même en envisageant ce scenario, l'effet du retard de calcul de la composante x_{42} à elle seule est inégalé.

En conclusion pour cette nouvelle approche et en considérant la matrice *bfwb62* de taille (62*62), dispatcher les différents retards de calcul qui peuvent avoir lieu lors du calcul parallèle et cela pour les différentes composantes, vaut bien mieux que de retarder uniquement le calcul d'une seule composante et en l'occurrence celle dont le retard de calcul a le plus d'impact sur la détection de la convergence.

Série de test sur la deuxième matrice :

Nous enchainons notre deuxième série de test dans le cadre de cette même approche avec le même retard de calcul qui est de 9/10 itérations ratées par les différentes composantes, et les tests sont les mêmes que ceux effectués précédemment mais cette fois-ci nous travaillerons avec une matrice de taille un peu plus grande, il s'agit en effet d'une matrice de taille (100*100) et c'est est la matrice *tub100* .

Retarder le calcul d'une seule composante :

Nous testons pour cette matrice l'effet d'un grand retard appliqué individuellement au calcul de certaines composantes, parmi lesquelles figure celle qui a le plus d'effet dans le calcul, les résultats numériques sont présentés dans la table 6:

Nbre d'itérations nécessaire a la convergence pour le retard de calcul de x_{39} pour 9/10 itérations	Nbre d'itérations nécessaire a la convergence pour le retard de calcul de x_{86} pour 9/10 itérations	Nbre d'itérations nécessaire a la convergence pour le retard de calcul de x_2 pour 9/10 itérations
6761 itérations	33811 itérations	3093 itérations

Table 6 : Résultats du retard de calcul d'une composante pour matrice (100*100)

Nous constatons de ce tableau que pour cette matrice de taille (100*100), et en appliquant un grand retard pour le calcul d'une composante dont l'effet est parmi les plus grands sur le calcul, cela entraine un très grand nombre d'itérations nécessaires à la convergence.

Combiner le retard de calcul de plusieurs composantes:

A présent nous combinons le retard de calcul de plusieurs composantes au même temps et illustrons les résultats obtenus dans la Table 7 :

Nbre d'itérations nécessaire a la convergence pour le retard de calcul de x_{39} pour 9/10 itérations	Nbre d'itérations nécessaire a la convergence pour le retard de calcul de x_{39} pour 1/10 et 8 composantes voisines pour 8/10 itérations	Nbre d'itérations nécessaire a la convergence pour le retard de calcul de x_{39} pour 1/10 itérations et x_{41} pour 8/10 itérations
6761 itérations	3918 itérations	5395 itérations

Table 7 : Résultats de combinaison du retard de différentes composantes pour la matrice (100*100)

De ce tableau, on voit que même lorsqu'on combine le retard de calcul des différentes composantes qui sont très puissantes dans le calcul, l'effet n'est pas aussi grand que lorsque l'on retarde le calcul d'une seule composante et cela pour plusieurs itérations successives, surtout si cette composante figure parmi les plus puissantes dans le calcul et pour confirmer ce constat, nous appliquons le pire scenario qui a été envisagé précédemment et qui concerne la répartition des retards de calcul pour les différentes composantes. De ce fait on applique un retard de calcul de 8/10 itérations à la composante la plus puissante (dans ce cas c'est la composante x_{39}), et un retard de calcul équivalant à 1/10 itérations ratées lors du calcul d'une composante tout aussi puissante dans le calcul (x_{37}) . Le résultat obtenu est représenté par la Figure 26 comme suit :

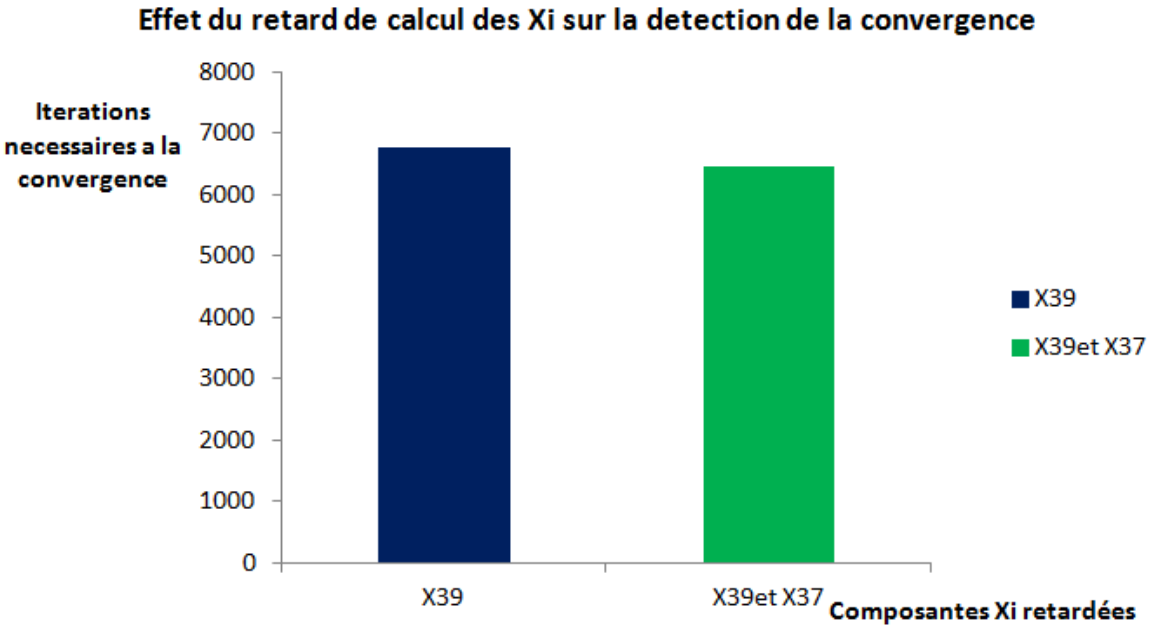


Figure 26 : Effet du retard de calcul d'une seule composante comparée a celui d'une combinaison de composantes xi dans la détection de la convergence dans la matrice de taille (100*100)

De cette figure nous voyons que retarder pour plusieurs itérations le calcul de la composante la plus puissante (x_{42}) a beaucoup plus d'effet que de retarder son calcul ainsi que celui d'une composante tout autant puissante et cela pour le même retard de calcul.

En conclusion à cette nouvelle approche et en considérant la matrice *tub100* de taille (100*100), combiner le retard de calcul de plusieurs composantes est préférable au cumul de ce même retard de calcul pour une seule et même composante.

Série de test sur la troisième matrice :

Pour cette troisième série de test, nous choisissons la matrice *saylr1*, dont la taille est encore plus grande (238*238) que les matrices précédentes.

Après avoir déterminé que la composante x_{38} est celle dont le retard de calcul implique le plus d'effets sur la détection de la convergence (un plus grand nombre d'itérations est nécessaire pour la détection de la convergence), nous exécutons le pire scenario qui concerne le repartitionnement des retard de calcul des composantes x_i. Pour cette matrice ce scenario consiste à retarder la composante x_{38} pour 8/10 itérations et la composante tout autant puissante x_{37} pour 1/10 itération. Les résultats obtenus sont représentés à travers la Figure 27 :

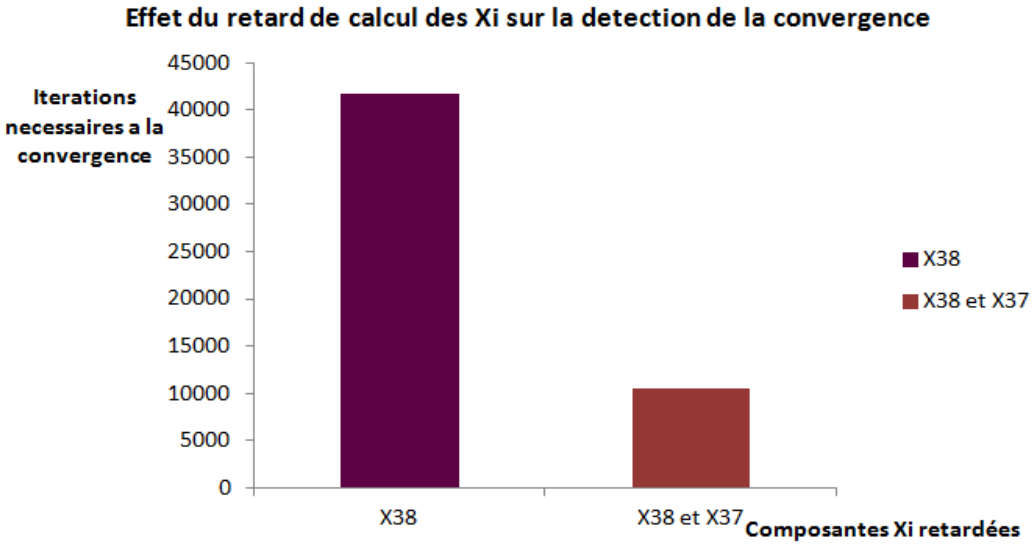


Figure 27 : Effet du retard de calcul d'une seule composante comparée a celui d'une combinaison de composantes xi dans la détection de la convergence dans la matrice de taille (238*238)

Nous constatons de cette figure que retarder le calcul de la composante x_{38} à elle seule pour un grand nombre d'itérations a beaucoup plus d'effet que de retarder son calcul au même temps

qu’avec celui d’une autre composante tout autant puissante dans le calcul, et cela pour le même retard.

En conclusion pour cette matrice ainsi que pour toutes les matrices testées jusqu'à présent lors de cette nouvelle approche adoptée, il s’est avéré qu’il est plus intéressant de repartir le retard de calcul qui est lié à une composante et cela pour les différentes composantes. En d’autres termes pour la résolution de system linéaire de type $\boldsymbol{Ax=b}$, l’affectation des taches de calcul aux machines de l’architecture distribuée ne doit pas être fixe tout au long du processus de calcul. De cette façon la volatilité d’une ressource ne pénalisera pas trop le calcul d’une composante. Car si c’est une composante importante dans le calcul qui est longtemps retardée, cela se répercutera aussitôt sur le calcul global et en l’occurrence sur la détection de la convergence du système linéaire.

IV. Conclusion et perspectives

Dans ce travail de recherche, nous nous sommes intéressés à l'exécution de programmes dans un réseau hétérogène et volatile (MANet) et plus précisément à l'implémentation d'une méthode itérative pour le calcul asynchrone dans le cadre de la résolution de système linéaire dans un environnement distribué et volatile. Nous avons cherché à déterminer les facteurs importants pour la détection de la convergence d'un système linéaire afin d'assurer une résolution de ce dernier de manière optimale. Nous avons proposé une approche basée sur des tests effectués, et qui contribue à résoudre partiellement des problèmes résultants de la volatilité de l'architecture distribuée pour le calcul parallèle.

Notre travail a commencé par une étude bibliographique qui concerne le calcul itératif asynchrone sur des architectures distribuées et volatiles. Nous avons effectué suite à cela des séries de tests concernant la détection de la convergence lors de la résolution de système linéaire de type $\boldsymbol{Ax=b}$ et cela pour différentes conditions (systèmes linéaire de différente taille, retards de calcul des composantes,…etc). A l'issue de ces nombreux tests nous avons constaté que l'importance des composantes x_i dans le calcul asynchrone qui détermine la convergence varie selon le système, et qu'il est assez délicat de la déterminer au préalable et de manière générale pour tous les systèmes linéaires. Suite à ce constat, nous avons soumis une approche qui vise à réduire l'effet de la volatilité des architectures hétérogènes et distribuées sur le calcul parallèle. Notre approche consiste à considérer le retard de calcul auquel peut être soumise une composante importante et de le repartir pour les différentes composantes du vecteur x et cela tout au long du processus de résolution du système linéaire, le retard en question est engendré par la défaillance de la machine chargée du calcul. Ceci revenait à simuler le changement plus ou moins régulier des localisations des calculs des composantes dans la grille de calcul, de façon à ce que les retards que pourraient causer certains processeurs soient répartis dans tous les calculs et non pas concentrés sur une composante.

Notre approche donne lieu à une perspective de recherche, En effet sa mise en pratique nécessite la conception d'un système chargé de l'affectation des taches de calcul aux différentes ressources d'une architecture distribuée, la mise au point de ce système nécessitera qu'il effectue des changements d'affectation de manière périodique, afin de minimiser ainsi l'effet de panne des machines défectueuses sur le calcul. Parmi les idées envisageables, on peut citer :

- L'idée de provoquer régulièrement un mélange aléatoire ('*shuffle'*) entre les processeurs de façon à ce qu'aucune composante ne reste en permanence sur le même processeur. On évite ainsi les cas catastrophiques dans lesquels une composante importante se retrouve en permanence retardée beaucoup.

- l'idée améliorant la précédente de construire un anneau virtuel entre les processeurs et à régulièrement faire en sorte en cours de calcul que chaque processeur donne à son voisin de droite l'état du calcul qu'il était en train de faire. De cette façon, aucune composante critique ne resterait longtemps au même endroit, et ne serait pas retardée longtemps si par malheur elle se trouve à un endroit fortement ralenti.

- l'idée de gérer les processeurs comme un 'pool' de ressources, et de donner à chaque composante une priorité dynamique croissante d'exécution au fur et à mesure que la date du dernier calcul pour cette ligne est loin dans le passé. De cette façon, aucune ligne ne prendra beaucoup de retard par rapport aux autres, et la tolérance aux pannes est gérée automatiquement. Une redondance n'est pas exclue pour ajouter un mécanisme de tolérance aux pannes, qui sont toujours possibles dans des systèmes volatiles. La priorité dynamique pourra être augmentée en fonction du temps, ou en fonction du retard entre composantes, par exemple.

Bibliographie

[1] Matrix Market http://math.nist.gov/MatrixMarket/.

[2] The University of Florida Sparse Matrix Collection
http://www.cise.ufl.edu/research/sparse/matrices/

[3] le Nouvel Oservateur –Science et avenir
http://sciencesetavenir.nouvelobs.com/high-tech/20121204.OBS1284/classement-quels-sont-les-plus-puissants-supercalculateurs-au-monde.html

[4] CHARR J. C. *Calcul itératif asynchrone a grande échelle sur des architectures hétérogènes et volatiles.* Thèse de doctorat, Université de Franche-Comté, France, 2009.

[5] CHAU M. *Algorithmes Parallèles Asynchrones pour la Simulation Numerique.* Thèse de doctorat, Institut national polytechnique de toulouse, France, 2005.

[6] MAZOUZI K. *JACE : un environnement d'exécution distribué pour le calcul itératif asynchrone.* Thèse de doctorat, Université de Franche-Comté, France, 2005

[7] BENAHMED A. *Un résultat de converge des algorithmes parallèles asynchrones, Application aux operateurs maximaux fortement monotone*. Thèse de doctorat, Université Mohamed Premier, Maroc, 2005

[8] FERTRE M. *Intégration d'un mécanisme de reprise d'applications parallèles dans un système d'exploitation pour grappe.*Rapport de stage de master 2, Université de Rennes 1, France, 2005.

[9] BOIS G. *System embarquee*. Polycopié cours, Polytechnique Montreal, Canada, 2013, https://moodle.polymtl.ca/pluginfile.php/81015/course/section/16828/DSP_ASIP.pdf.

[10] PASTRE D. *Methodes iteratives*. Polycopié cours, Universitee Paris 5, France, 2002, http://www.math-info.univ-paris5.fr/~pastre/meth-num/MN/A-Jacobi-GaussSeidel-gradients/cours-jacobi-gaussseidel-gradients.pdf.

[11] NAKECHBANDI M. *Complexité spatiale et dynamique*, Polycopiés cours M2-S1-CPX, Université du Havre, 2013, http://nakechb.free.fr/M2-CPX

[12] CAPPELLO F, et al. *Grid'5000: a large scale and highly reconfigurable Grid experimental testbed*, http://www.labri.fr/perso/ejeannot/publications/grid05.pdf

[13] MARCHAND A. *Les architectures parallèles-MPI*, Observatoire Paris, France, http://dio.obspm.fr/PDF/archi_mpi.pdf

[14] CANTONE L, UNAL Résoul. *Le Grid Computing et son utilisation dans les entreprises et les industries*, polycopié cours Master 2 SIR 2004-2005, Université Lyon 1, 2004, http://web.univ-pau.fr/~cpham/M2SIR/BIBLIO/DOC04-05/Grid.doc

[15] Techno-science.net : Matrices http://www.techno-science.net/?onglet=glossaire&definition=5201