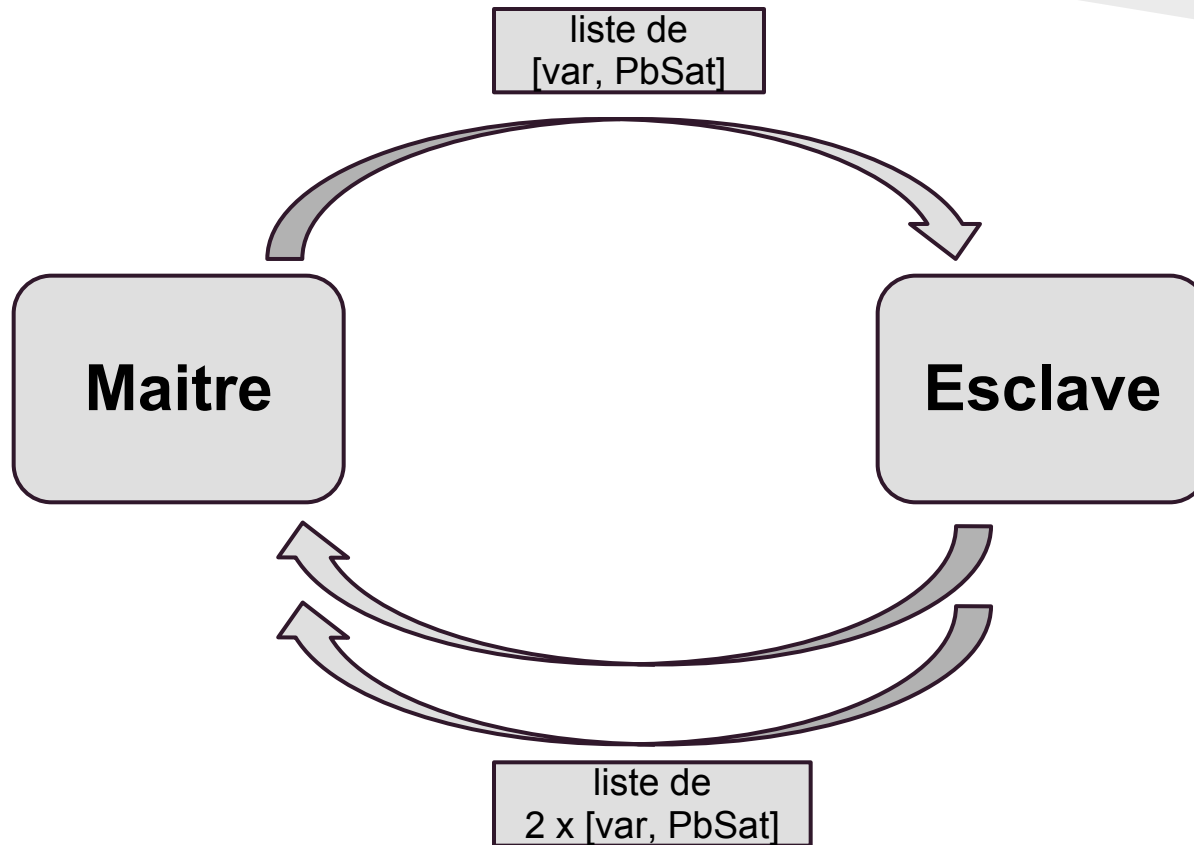


# Résolution de SAT

*Architecture Parallèle et Distribuée*

Adlen Afane  
Edouard de Lansalut  
Nicolas Schmidt

# Schéma d'ensemble



# Schéma d'ensemble

## Maitre:

- Dépile les travaux en attente et envoie les différents batchs de problèmes aux esclaves en communication bloquante
- Récupère les réponses en communication bloquante et traite les résultats
  - Ajout à la file
  - Annonce de réussite
- Différents algorithmes de gestion de la file:
  - FIFO
  - Aléatoire
  - Priorité aux problèmes avec le moins de variables restantes

# Schéma d'ensemble

## Esclave:

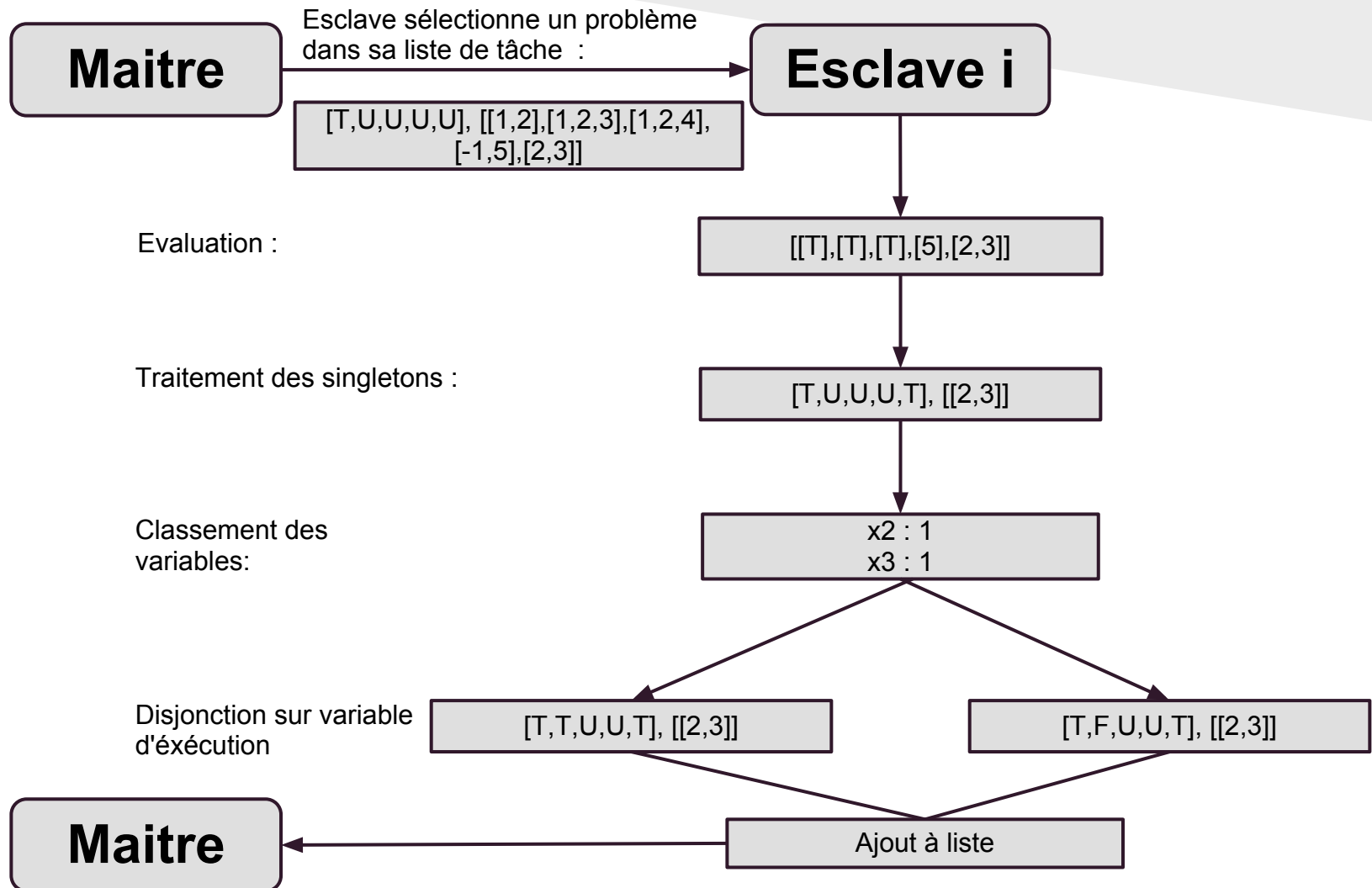
- Reçoit une liste de problèmes SAT et d'affectation des variables
- TANT QUE liste non vide :
  - Évalue les clauses du problème pour cet ensemble de variables
  - Gère les clauses ayant un seul littéral
  - SI on peut déterminer problème SAT vrai ou faux :
    - Envoie un message au maître
  - SINON :
    - Calcule la meilleure variable non encore affectée sur laquelle effectuer une disjonction
    - Génère les 2 (problème SAT, affectation des variables) correspondants
    - Stocke dans la liste de résultats
- Renvoie les résultats au maître en communication bloquante

## Comment effectuer la disjonction ?

Classement basé sur la fréquence d'apparition des variables et leur répartition entre l'expression “xi” et sa négation “xi barré”

# Schéma d'ensemble - Esclave


## Exemple



# Choix et problèmes techniques

- Installation de mpi4py
- Communication asynchrone
- Choix de l'architecture Maître / Esclave
- Gestion de la file d'attente
- Alimentations des statistiques avec un script
- Représentativité des tests

# Tests de l'algorithme

- 
- Petits problèmes à la main
  - Utilisation de "boolsat.com"
  - Utilisation de problèmes fournis par UBC
    - Gros problèmes à satisfiabilité connue
    - Création d'un script de test (environ 1800 tests)
    - Enregistrement de stats à chaque lancement.

	Variables	Clauses
uf20-91	20	91
uf50-218	50	218
uuf50-218	50	218
uf75-325	75	325
uuf75-325	75	325
uf100-430	100	430
uuf100-430	100	430

# Influence des parametres

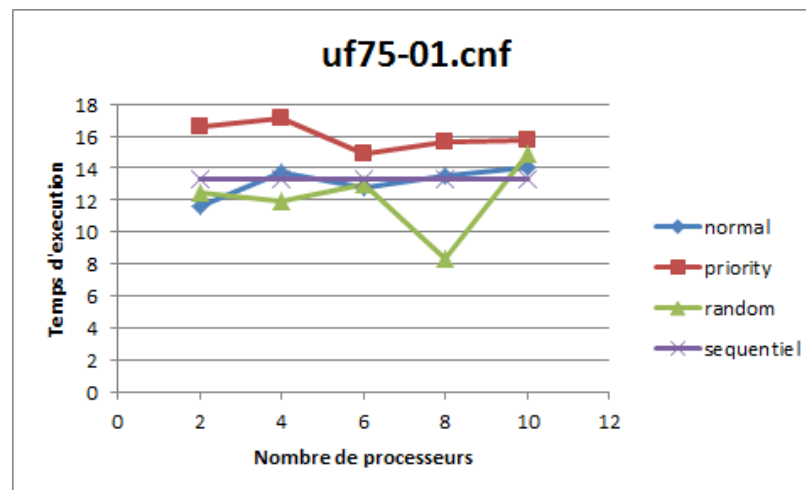
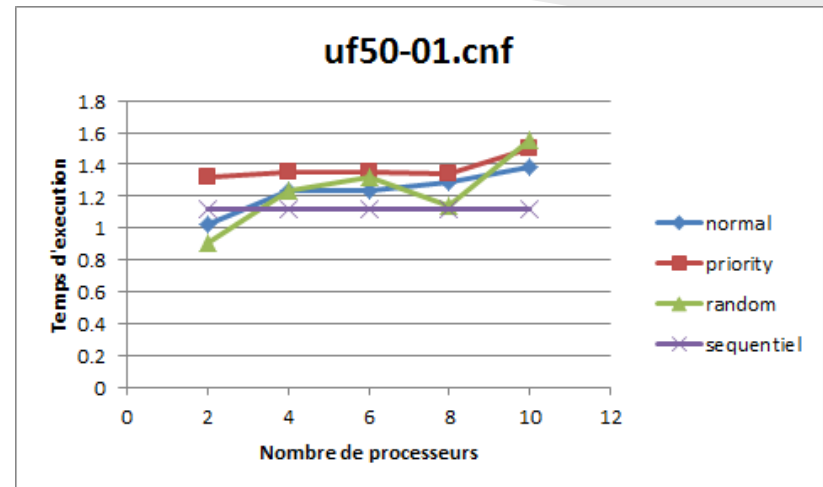
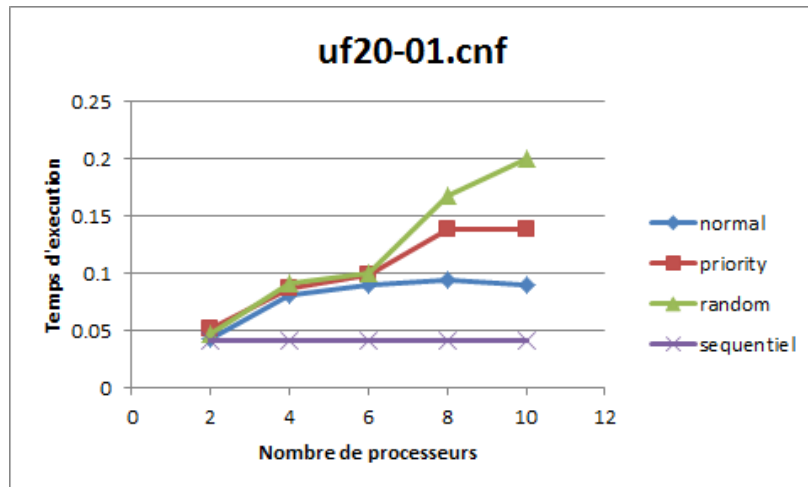
- Nombre de processus Processus
- Taille du probleme (# clauses et variables)
- Nombre de travaux par message entre le maître et un esclave
- Mode de gestion de la pile du maître

**1787 tests**



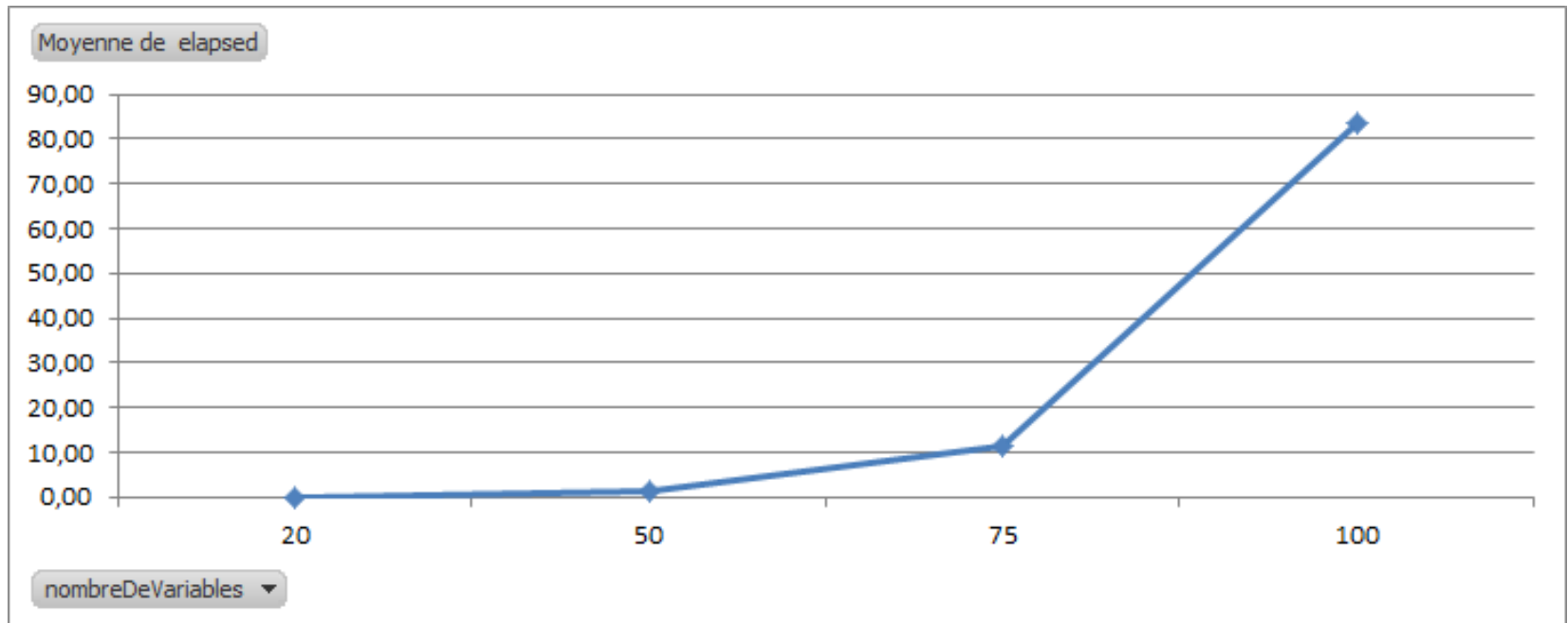
# Performances

## En fonction du nombre de processus



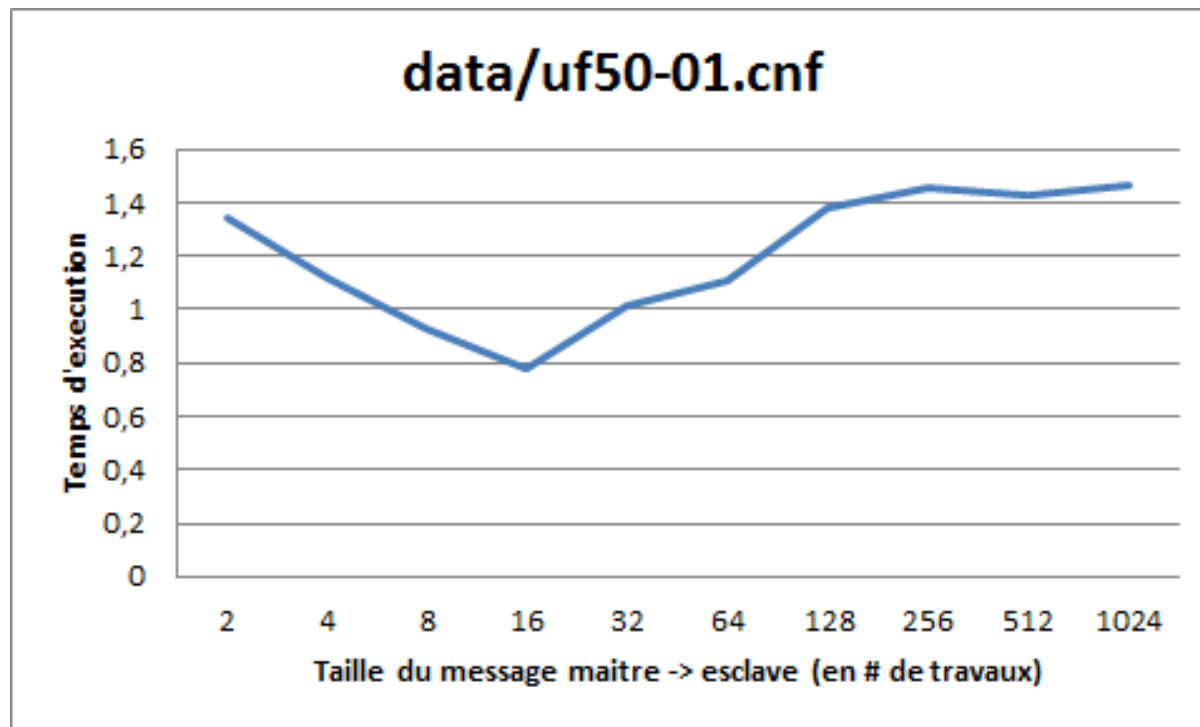
# Performances

En fonction de la taille du problème



# Performances

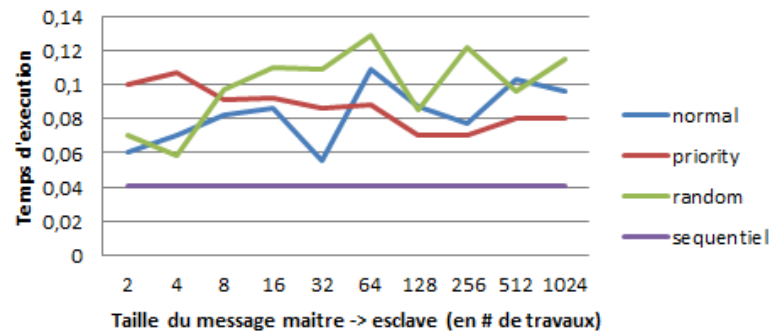
En fonction de la taille de batch



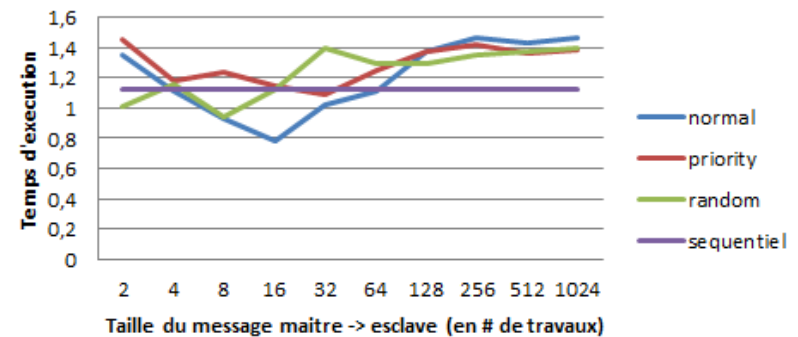
# Performances

En fonction de la gestion de la pile du maître

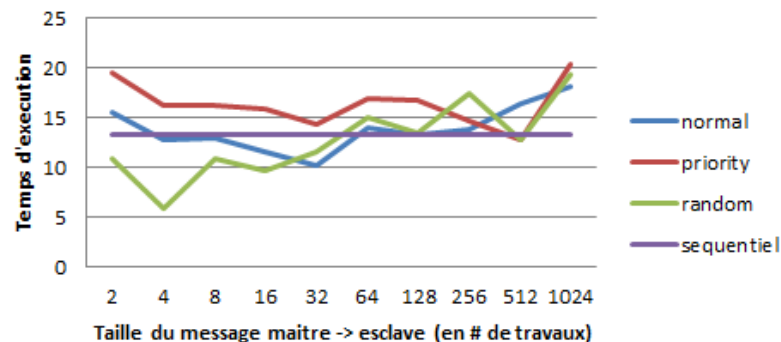
Comparaison Gestion de file  
data/uf20-01.cnf



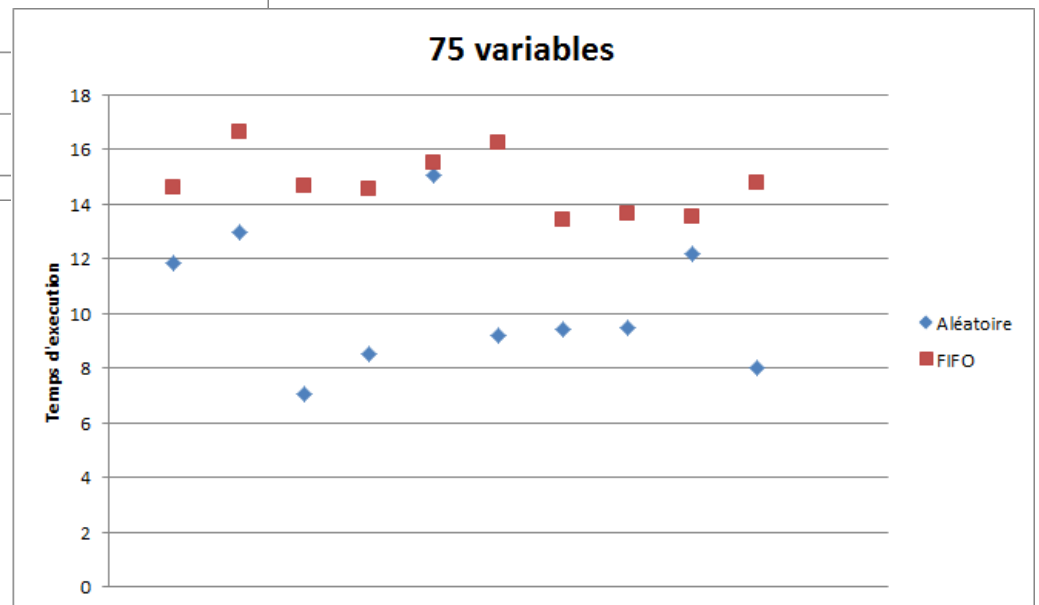
Comparaison Gestion de file  
data/uf50-01.cnf



Comparaison Gestion de file  
data/uf75-01.cnf



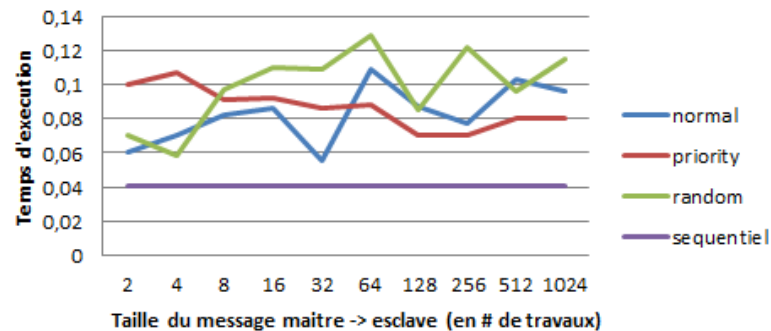
# En fonction de la gestion de la pile du maitre



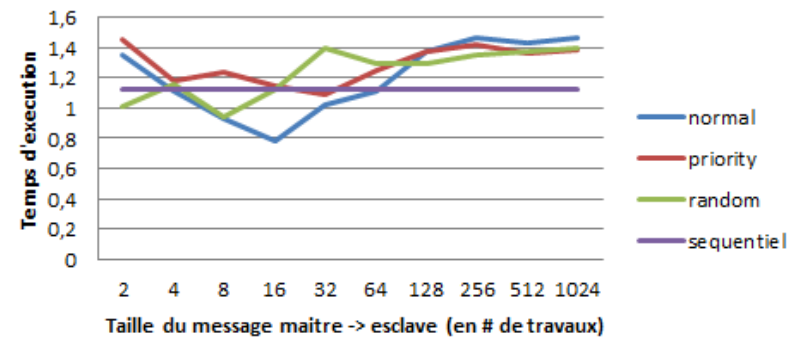
# Performances

## Parallèle VS Distribué

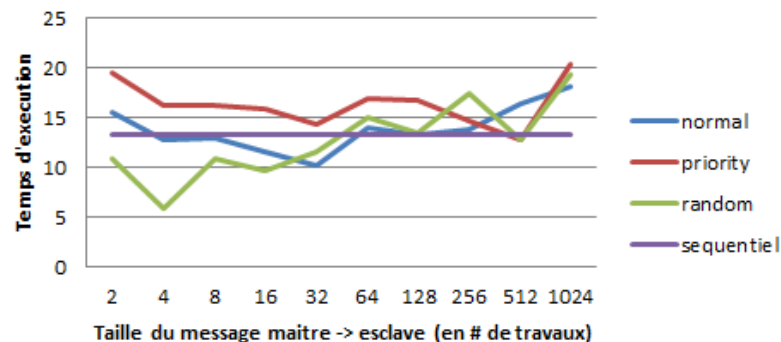
Comparaison Gestion de file  
data/uf20-01.cnf



Comparaison Gestion de file  
data/uf50-01.cnf



Comparaison Gestion de file  
data/uf75-01.cnf



# Conclusion sur les performances

Des 1787 tests, on déduit:

- influence du nombre de processus difficile à observer ici
- le temps de résolution explose comme prévu en fonction de la taille du problème
- Il semble exister une taille optimale de batch (taille des message maitre -> esclave)
- la gestion de pile aléatoire par le maitre est plus efficace
- La performance de notre distribution est plus évidente sur les gros problèmes

Merci pour votre attention