Test de Primalité

Jean-Didier Pailleux - Robin Feron - Romain Robert - Damien Thenot - Maxence Joulin

UVSQ

11/01/2018

Introduction

- **Nombre Premier :** Entier divisible par 1 et lui-même.
  pause
- **Test Probabiliste :** Test avec marge d'erreur très faible mais rapide.
- **Test Deterministe :** Test fiable mais plus lent.
- 2^{64} **:** Taille maximale des nombre a tester $=>$ Unsigned long long int
- **Nombre Hautement Composé :** Entier qui possède strictement plus de diviseur que les nombres qui le précède

Introduction

Plan

Les méthodes naïves

- Le Crible d'Eratosthène : efficace mais coûteux en terme de mémoire (Memory Bound)
- Les divisions euclidienne et PGCD : simples mais couteûses en calculs (Computation Bound)

Les méthodes naïves

- Le Crible d'Eratosthène : efficace mais coûteux en terme de mémoire (Memory Bound)
- Les divisions euclidienne et PGCD : simples mais couteûses en calculs (Computation Bound)

Les méthodes modernes

- Pocklington : Factorisation partielle en nombres premiers
- AKS : Résolution d'inconnues selon le petit théorème de Fermat : complexité polynomiale très interressante pour les grands nombres

Miller Rabin

Utilisation du petit théorème de Fermat :
$a^p - 1 \equiv 1 \pmod{p}$.
Ne permet que d'affirmer qu'un nombre est probablement premier. Un nombre d'iteration suffisant peut permettre d'atteindre des taux d'erreurs infimes pour un temps de calcul très faible.

Miller Rabin

Utilisation du petit théorème de Fermat :
$a^p - 1 \equiv 1 \pmod{p}$.
Ne permet que d'affirmer qu'un nombre est probablement premier. Un nombre d'iteration suffisant peut permettre d'atteindre des taux d'erreurs infimes pour un temps de calcul très faible.

Les nombres hautement composés

- Méthode naive : Chercher le nombre de diviseurs de tout les nombres inferieurs à N
- Méthode utilisant une propriété de leur forme : une décomposition en nombres premiers à facteurs décroissants.

Plan

Langages de programmation

- Le langage C++ : langage adapté pour la programmation procédurale et orientée objet + bibliothèques pour les grands entiers supérieur , chronométrer et autre.
- Le langage Shell : enchaînement de commandes pour les tests.

Implémentation

Langages de programmation

- Le langage C++ : langage adapté pour la programmation procédurale et orientée objet + bibliothèques pour les grands entiers supérieur , chronométrer et autre.
- Le langage Shell : enchaînement de commandes pour les tests.

Outils

- La bibliothèque GMP : manipulation de nombres supérieur à 2^{64}.
- La bibliothèque NTL : fonctions pour l'arithmétique modulaire disponible.
- CMkake : pour la compilation du projet.
- Github : dépôt du projet + travail collaboratif.

Plan

Lancement d'une phase de test :

- Appel d'un script bash ou lancement en ligne de commande.
- Script bash => test sur un fichier ou plage de valeurs ou test normal.

Options

a : Tous les algorithmes
e : Euclide (computation bound)
m : Crible d'eratosthene
i : Miller-Rabin
H : Nombre hautement composé def
k : AKS
o : Modulo (computation bound)
p : Pocklington
h : Nombre hautement composé naive

- Lequel utiliser ?
- Itération ?
- Combien de nombre ?
- Donner les nombres

Analyse des Résultats

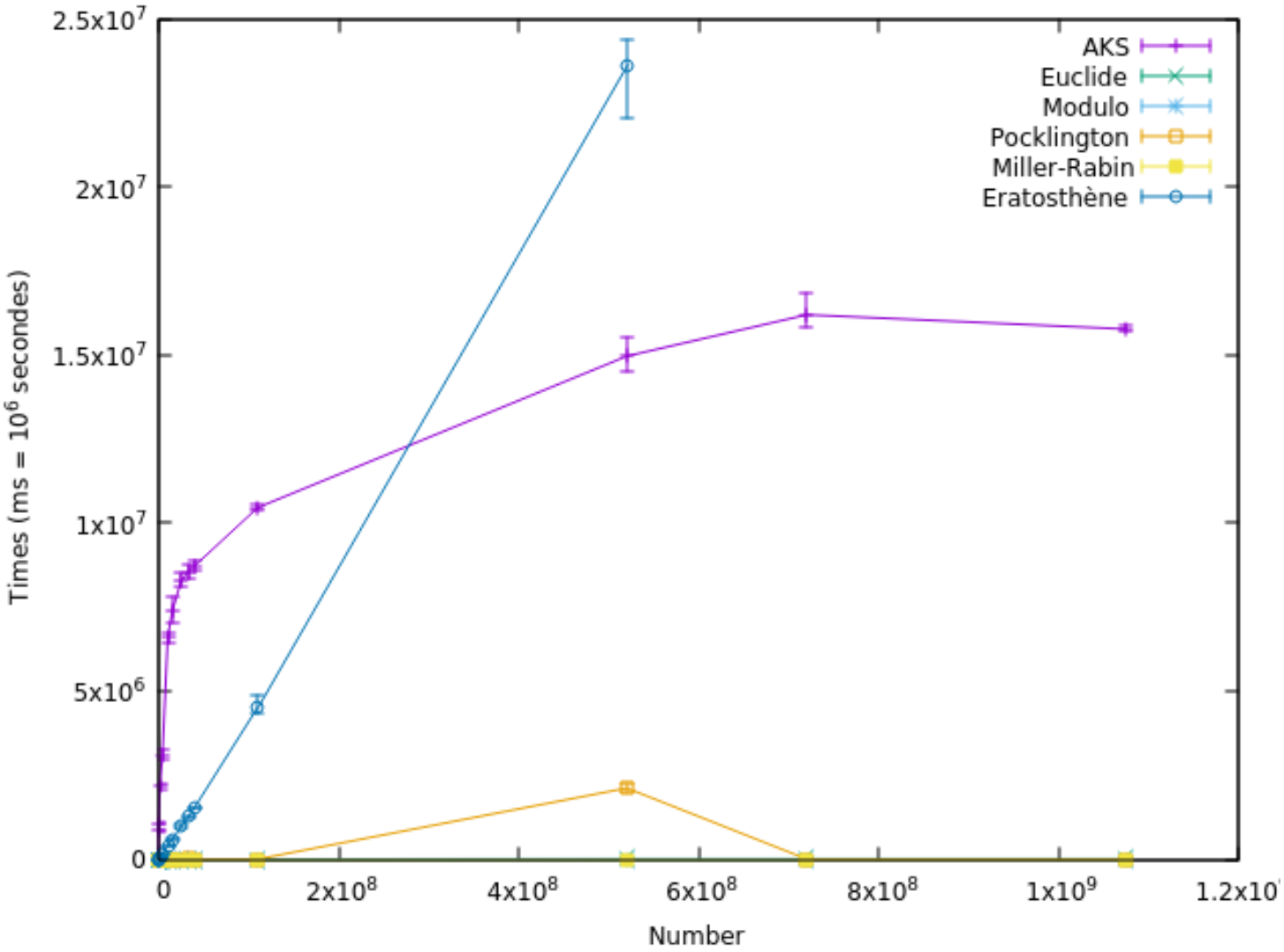


FIGURE – Évolution du temps d'exécution pour les 6 tests de primalité (AKS, Pocklington, Miller-Rabin, Euclide, Ératosthène et Modulo).

Analyse des Résultats

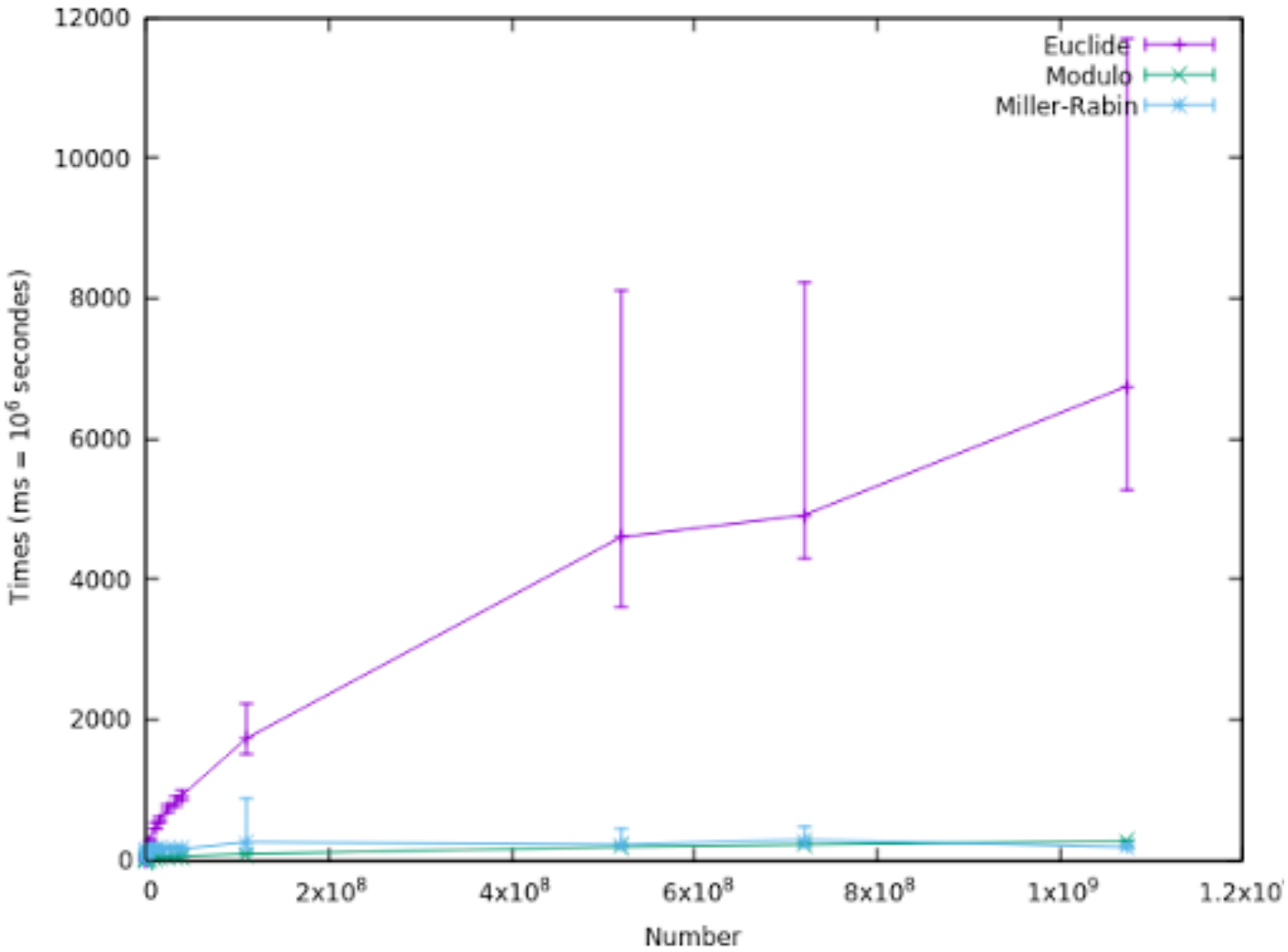


FIGURE – Zoom de la figure 1 sur (Miller-Rabin, Euclide et Modulo).

Analyse des Résultats

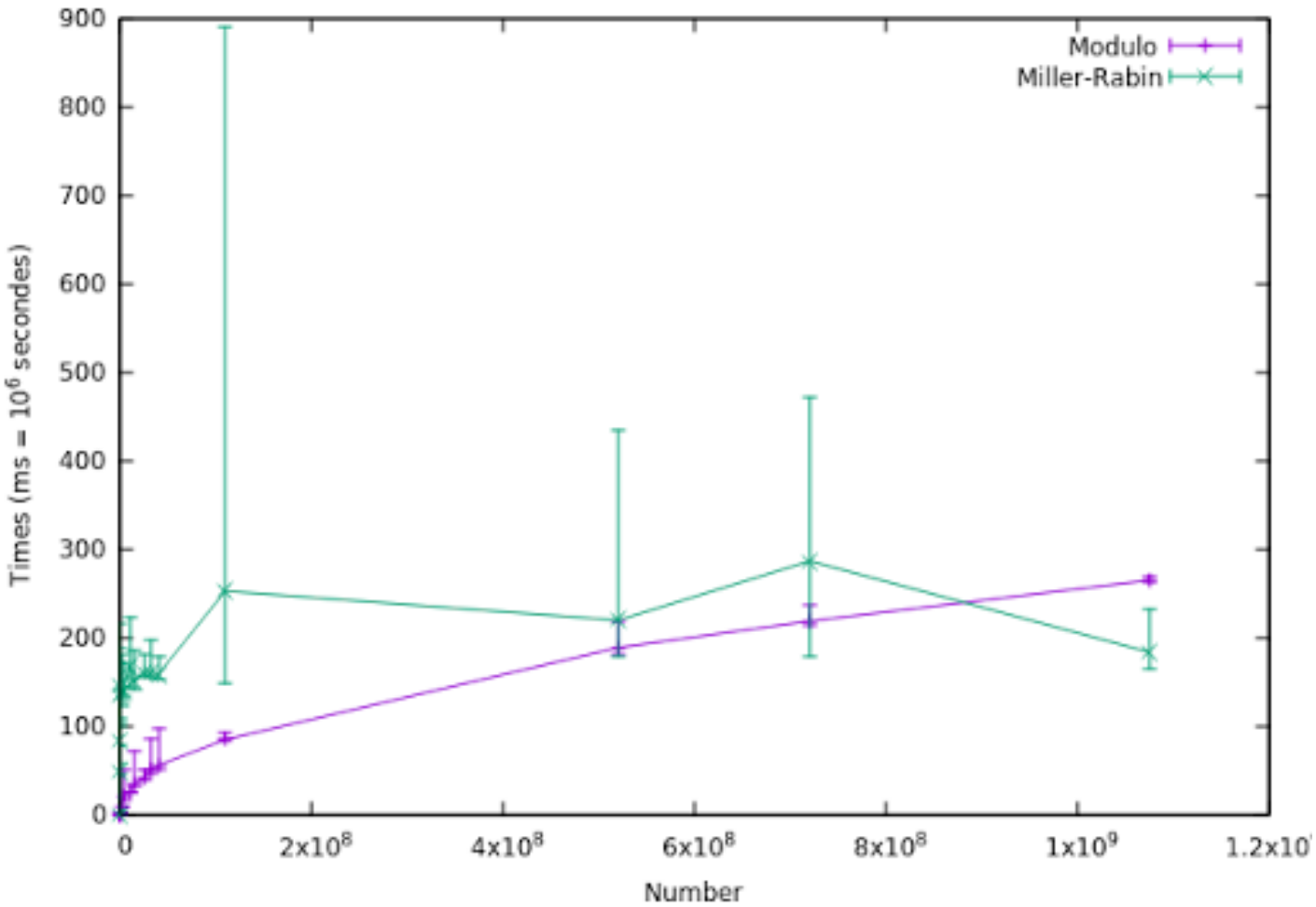
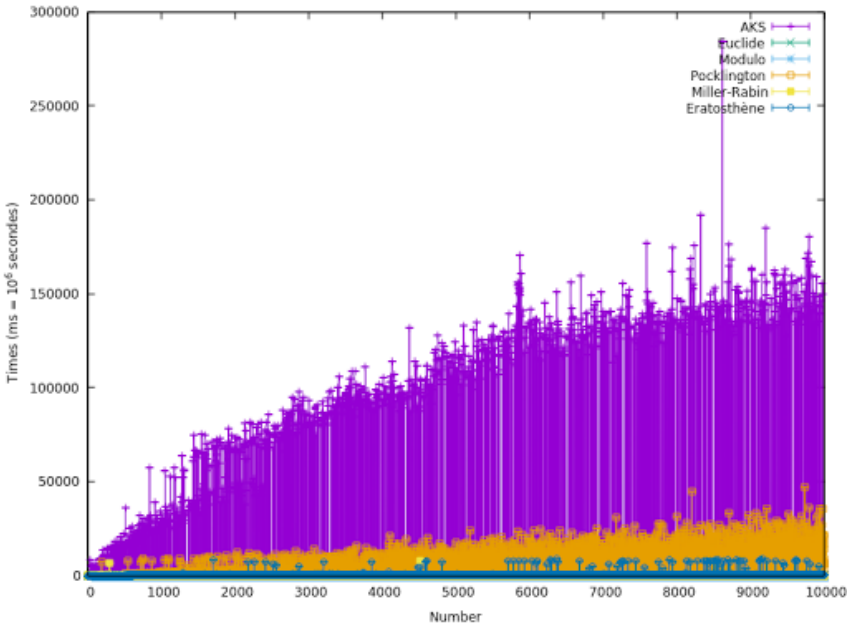
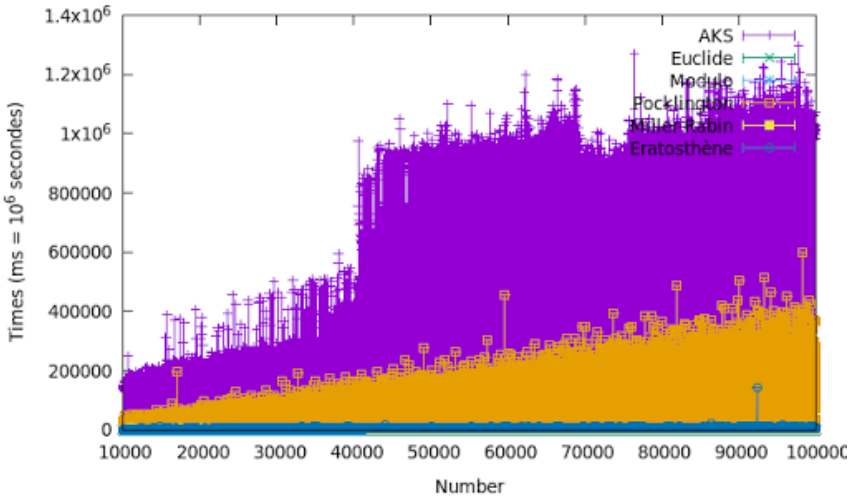


FIGURE – Zoom de la figure 1 sur (Miller-Rabin et Modulo).

Figures - *Évolution du temps d'exécution pour les 6 tests de primalité sur une plage de données*



Plage [1 : 10000]
100% de réussite pour Miller-Rabin



Plage [10000 : 100000]
100% de réussite pour Miller-Rabin

Analyse des Résultats

- L'algorithme AKS met en moyenne 118064433 μs soit environ 118,06s
- L'algorithme d'Euclide met en moyenne 11937 μs soit environ 0,012s
- L'algorithme appliquant le modulo met en moyenne 389 μs soit environ 0,0004s
- L'algorithme de Pocklington met en moyenne 41178556 μs soit environ 41,17s
- L'algorithme d'Eratosthène met en moyenne 2674942 μs soit environ 2,67s
- L'algorithme de Miller-Rabin met en moyenne 163611 μs soit environ 0,163s

Plan

Bilan Technique

- **Eratosthène :** Création d'un tableau de taille N+1 dans le crible => limité au niveau de la RAM pour N grand sur nos machines. Complexité de N pour le remplissage de la liste memory_bound.
- **Euclide :** Effectue $\sqrt{2^{log_2(n)}}$ divisions euclidiennes => Exécution en temps exponentiel
- **Pocklington :** Limite causé par la factorisation du nombre N-1 => factorisation très longues pour N très grand.
- **Miller-Rabin :** Résultats faux dans certains cas + Nombre d'itérations demandé élevé pour un meilleur résultat => augmentation du temps d'exécution.
- **AKS :** Avantage : Sa complexité en $log(n)^{12}$.
  Inconvénient : Utilisation de NTL qui effectue des vérifications superflue + Implémentation compliquée.

Plan

Conclusion

- De grands temps de calculs pour les tests déterministes.
- Méthodes naïves efficaces pour les petits nombres.
- Tests probabiliste une bonne idée ?
- **Possibilités de parallélisation :** Une ouverture sur des techniques de calcul en parallèle pourrait être appliqués.

Organisation interne du groupe

Tableau de répartition du travail :

Tâches	Jean-Didier	Maxence	Romain	Robin	Damien
Eratosthène/Memory Bound	x				
Euclide/Computation Bound		x			
AKS			x		
Pocklington					x
Miller-Rabin				x	
Highly Composite	x				
Cmake	x	x			
Script/main	x	x			