Performances de la programmation multi-thread

Eric Beuque Cyril Moutenet

URF Sciences et Techniques, Université de Franche-Comté, Besançon

06 février 2009

Tuteur : Laurent Philippe
Jury : Olga Kouchnarenko

Plan

Introduction

- Choix des langages à étudier
  - C (Posix Thread, OpenMP, fork)
  - C++ (TBB)
  - Java (Thread, Rmi)
  - C# (Thread) (Mono sous linux)
- Plates-formes
  - OS : Windows, Linux
  - Type de processeurs : Core2Duo, P4 Hyperthreading, Bi-Proc QuadCore
- Choix des algorithmes de test
  - Fibonacci
  - Multiplication de matrices
- Etude des bibliothèques de synchronisation

Plan

Définition

Processus léger

Un processus léger (thread) est un processus qui possède une mémoire partagée avec le programme qui le lance.

Multi-threading

Le Multi-threading décrit des tâches qui peuvent être exécutées de façon simultanée.

Hyperthreading

L'Hyperthreading est une caractéristique d'un processeur à deux processeurs logiques sur un processeur physique.

Outils de programmation (1)

POSIX

pThread sous UNIX

OpenMP

API C/C++ et Fortran. Pour le développement a granularité faible proche du séquentiel.

MPI

Exploitation des ordinateurs distants par passage de messages et offre la possibilité de réaliser des calculs parallèles à mémoire distribuée.

Outils de programmation (2)

TBB

Calcul parallèle pour C++, avec abstraction des threads.

JSR166y

Bibliothèque pour Java équivalente au package `java.util.concurrent`.

Plan

Etude des performances

Algorithmes :

- Fibonacci
- Multiplication de matrices

Tests effectués :

- Comparaison des différents langages et bibliothèques
- Comparaison entre le non threadé et le multi-threadé
- Tests des différents processeurs mono et multi-cœurs

Plan

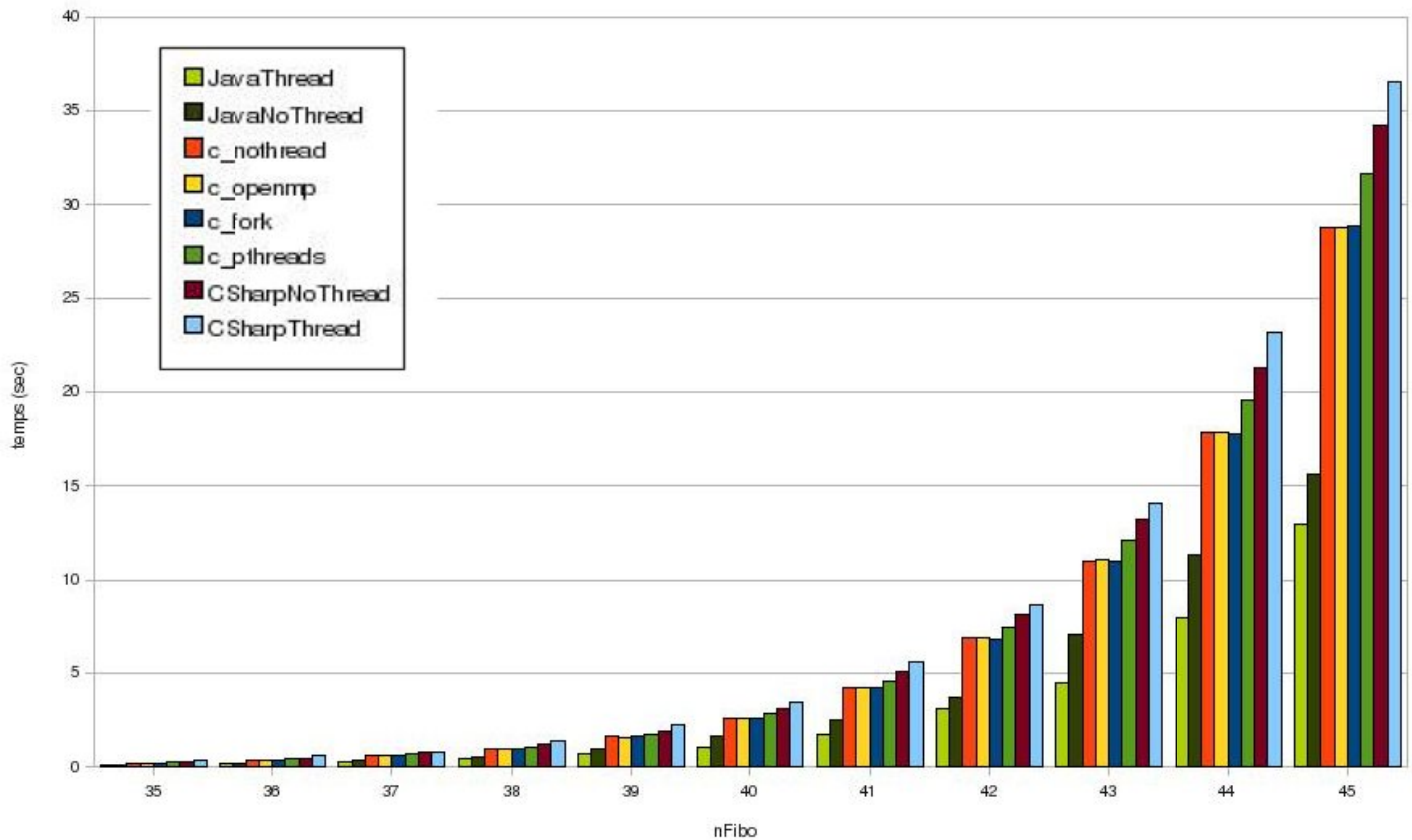
Test sur Fibonacci - Algorithme

Algorithm 1 Calcule n fois le nombre de Fibonacci d'un entier

```
1: nFibo ← lire()                          ▷ Lecture du nombre à calculer
2: nbFois ← lire()                         ▷ Lecture du nombre de fois à calculer
3: dateDebut ← now()
4: for i = 1 to nbFois do                  ▷ Début de la région parallèle
5:     fibo(nFibo)
6: end for                                 ▷ Fin de la région parallèle
7: dateFin ← now()
```

Comparaison des différentes bibliothèques

Tests sans threads : $1 \times fibo(n)$



Java > *C* > *CSharp*

Java 2 X plus rapide

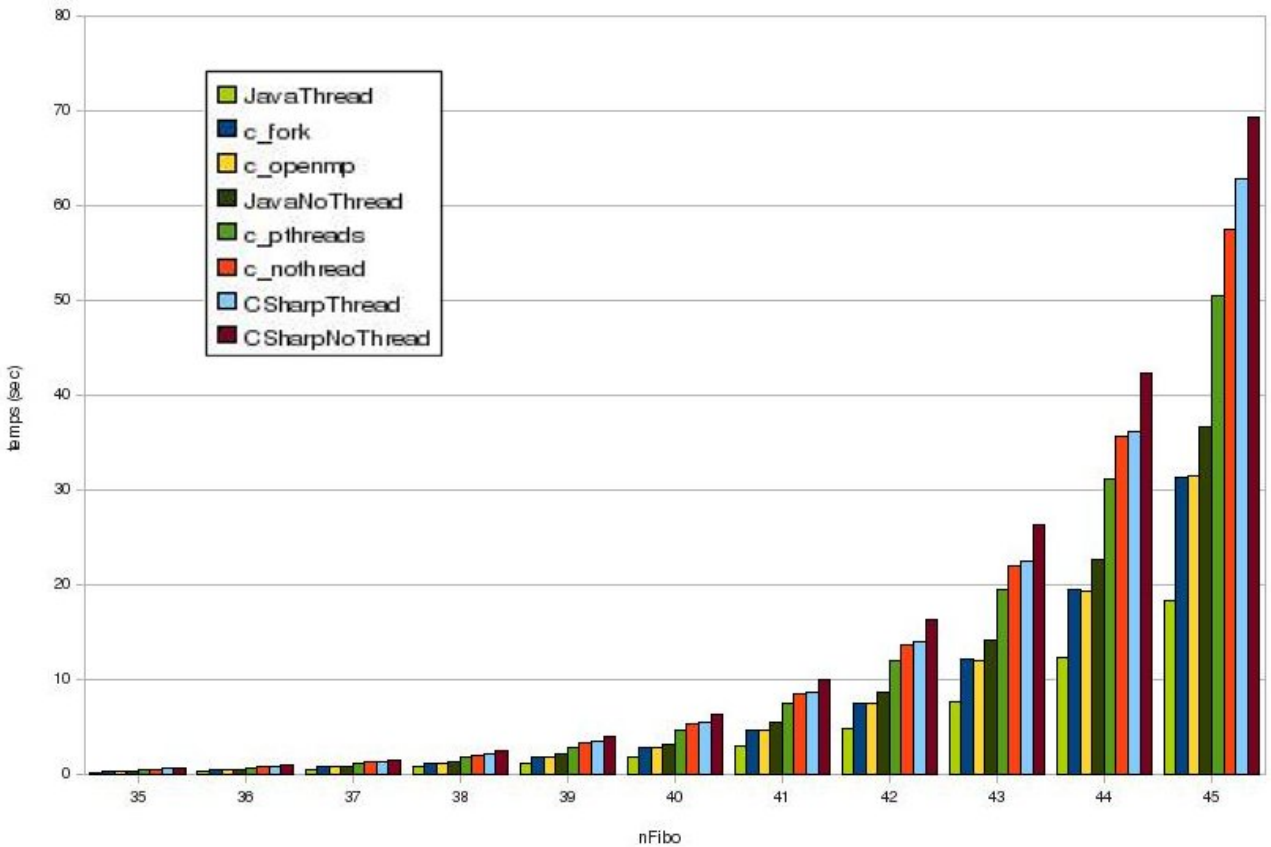
Thread ≈ *NonThread*

Meilleur : *JavaThread*

Linux Core2Duo T9300 2,5GHz

Comparaison des différentes bibliothèques

Tests avec 2 threads : $2 \times fibo(n)$



Linux Core2Duo T9300 2,5GHz

Java > *C* > *CSharp*

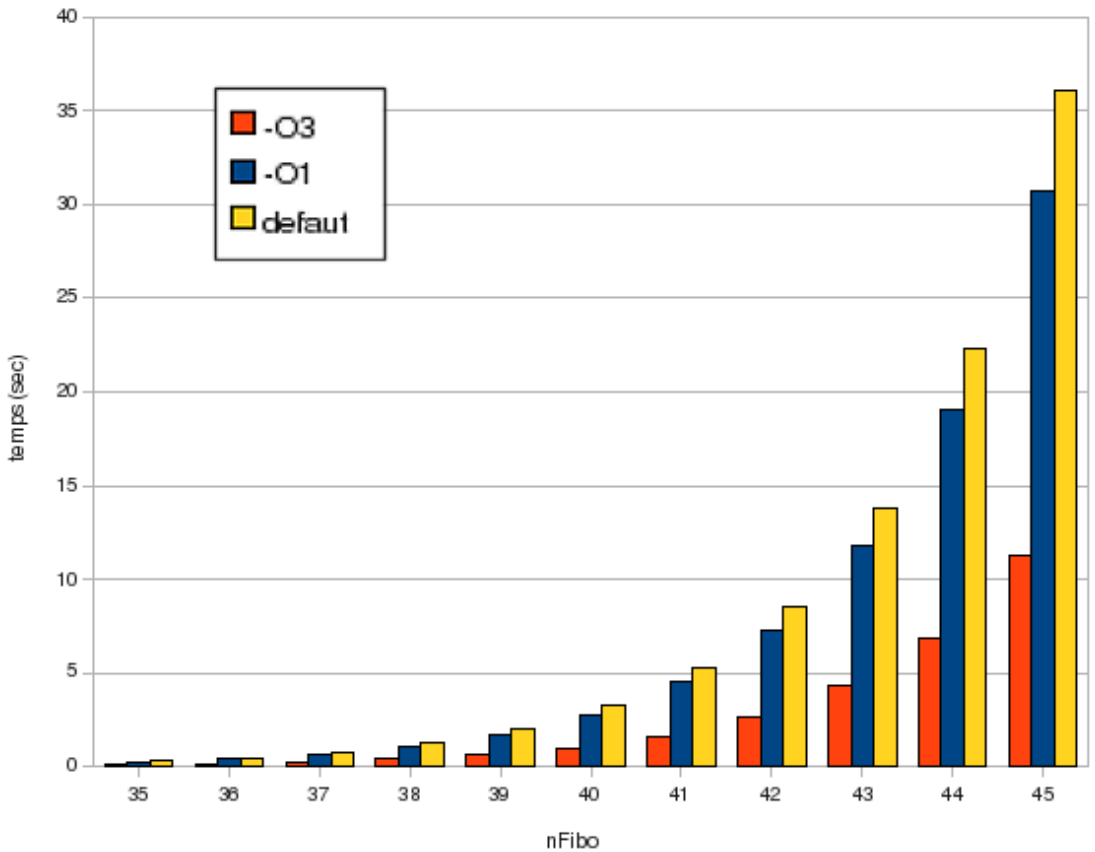
Thread > *NonThread*

Meilleur : *JavaThread*

Faible : *CSharpMono*

Comparaison du niveau d'optimisation du compilateur en C

Tests du niveau de compilation de GCC



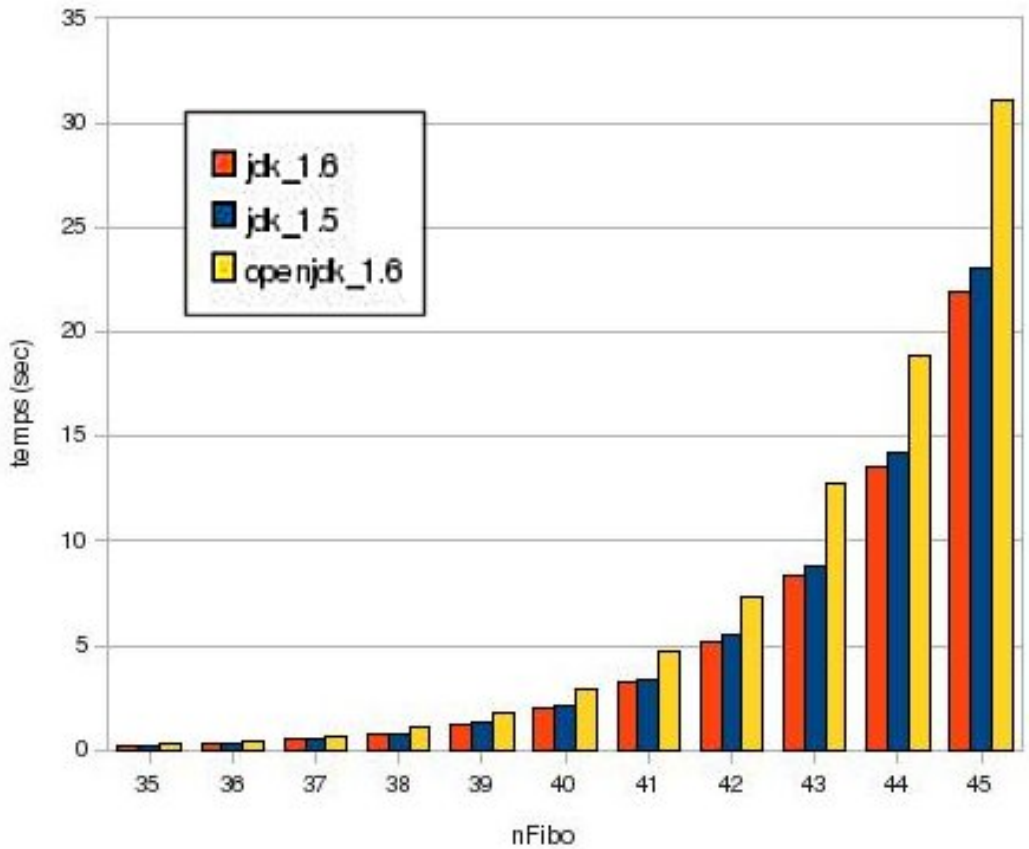
Linux Core2Duo T9300 2,5GHz

Meilleur : *O*3

3 X plus rapide

Comparaison des machines virtuelle Java

Tests des JVM Sun 1.5, 1.6 et OpenJDK 1.6

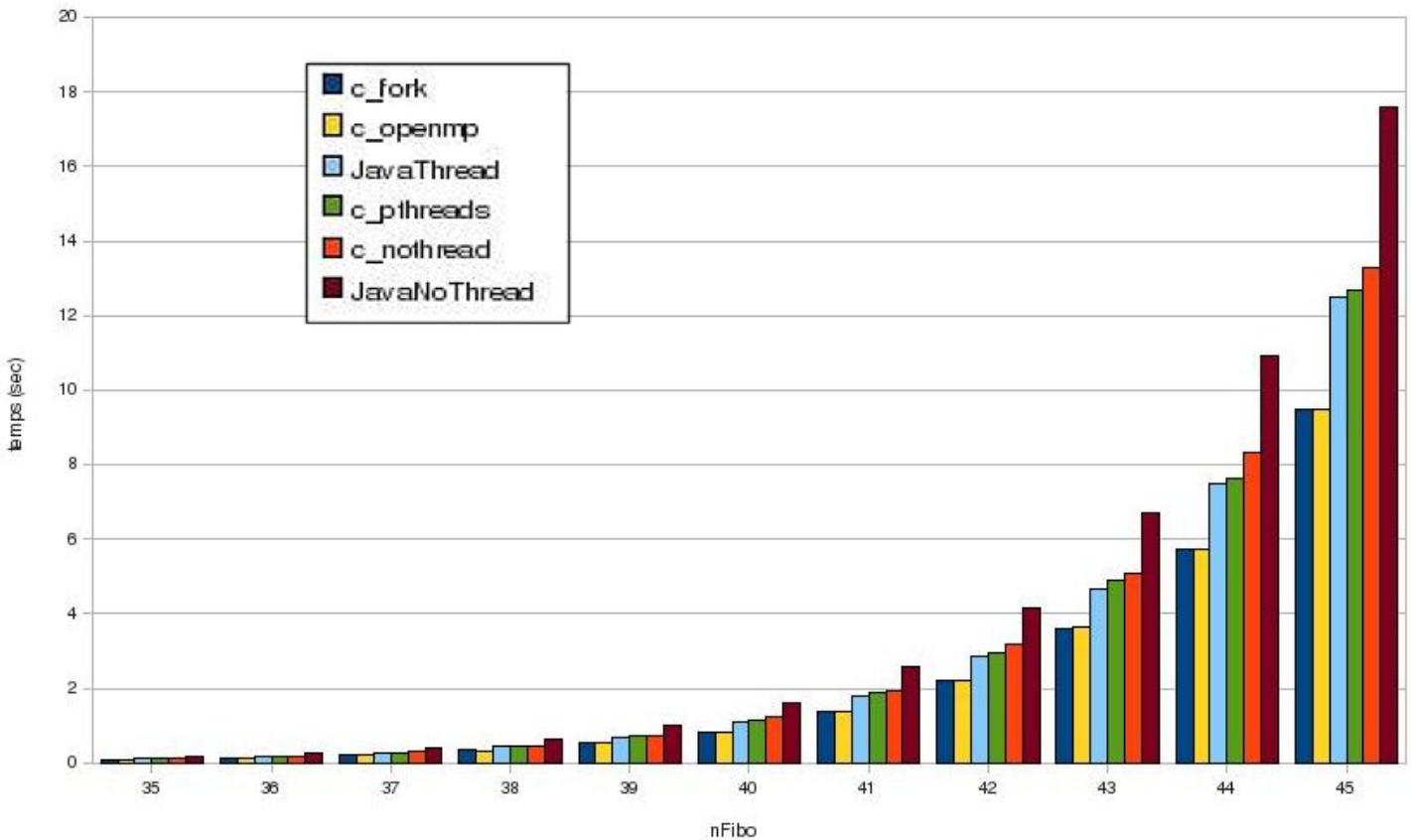


Linux Core2Duo T9300 2,5GHz

$SunJDK1.5 \approx SunJDK1.6 > OpenJDK1.6$

Comparaison finale entre Java et C

Tests entre SunJDK1.6 et C en -O3



$C > Java$

Meilleur :
Fork, *OpenMP*

Linux Core2Duo T9300 2,5GHz

Plan

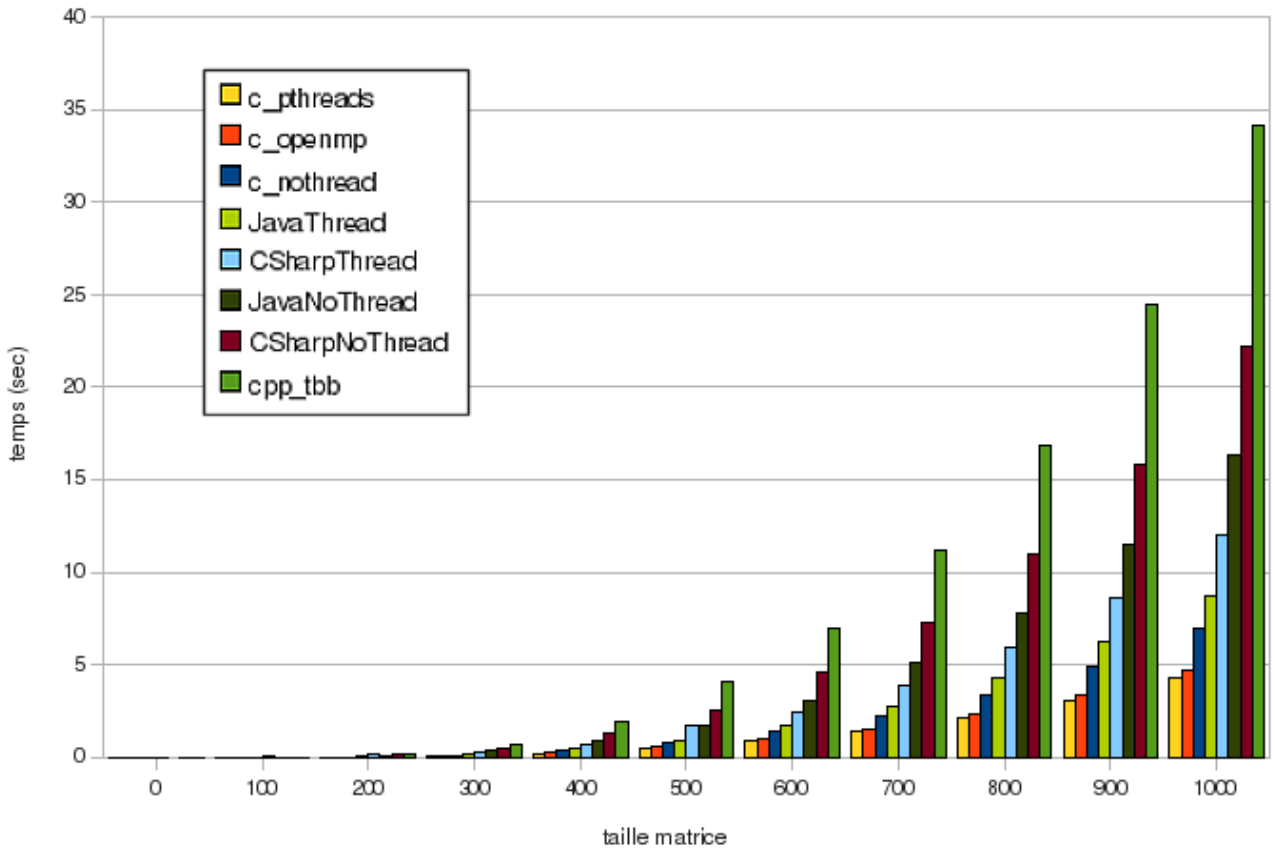
Test sur la multiplication de matrices - Algorithme

Algorithm 2 Calcule la multiplication d'une matrice A par B, et met le résultat dans C

```
1:  dateDebut ← now()
2:  A, B ← initialiser(mSize)                    ▷ Initialise les matrices
3:  for i = 0 to mSize do                        ▷ Début de la région parallèle
4:      for j = 0 to mSize do
5:          for k = 0 to mSize do
6:              C[i][j] ← C[i][j] + A[i][k] * B[k][j]
7:          end for
8:      end for
9:  end for                                      ▷ Fin de la région parallèle
10: dateFin ← now()
```

Comparaison des technologies de processeurs

Tests Core 2 Duo - 2 threads



Linux Core2Duo T9300 2,5GHz

Gain $\approx x2$ avec threads

TBB : *faible* $<$ *CSharp*

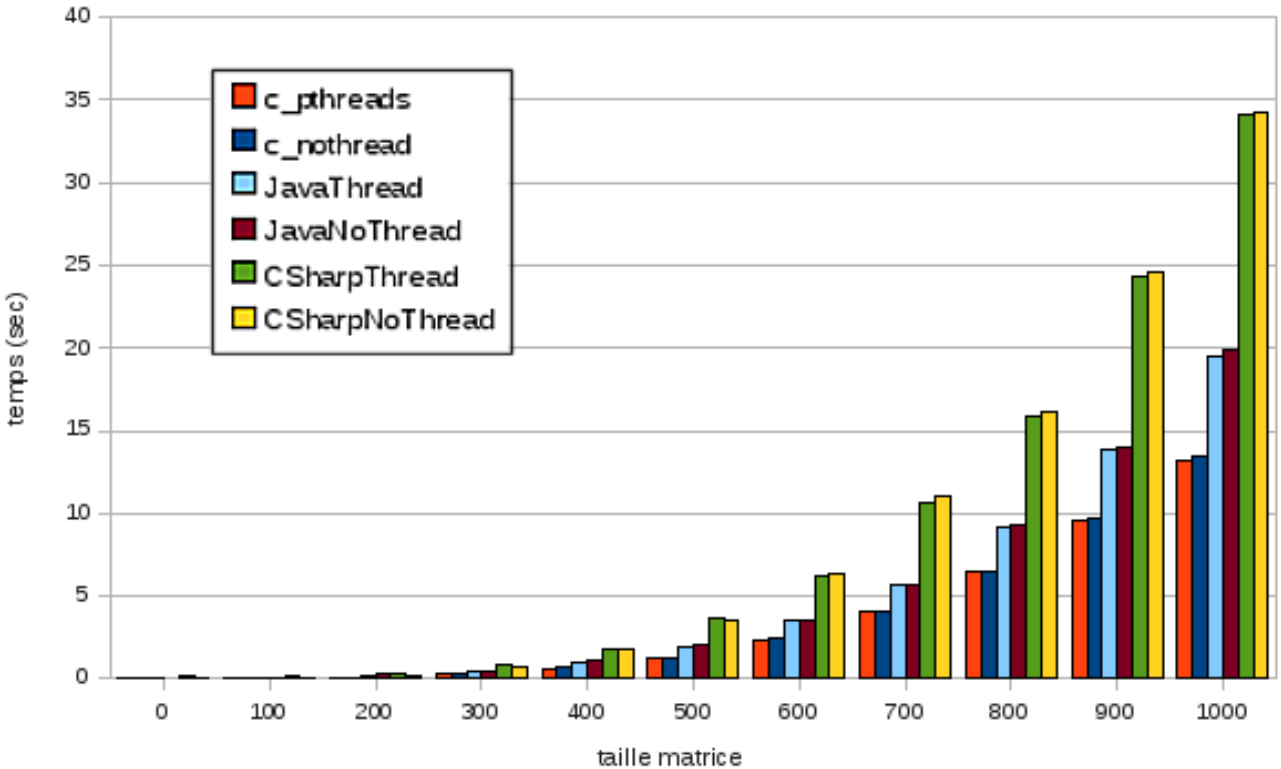
CNoThread $>$ *Java*

PThread $\approx$ *OpenMP*

Comparaison des technologies de processeurs

Tests P4 Hyperthreading - 2 threads

Pas de gestion des threads

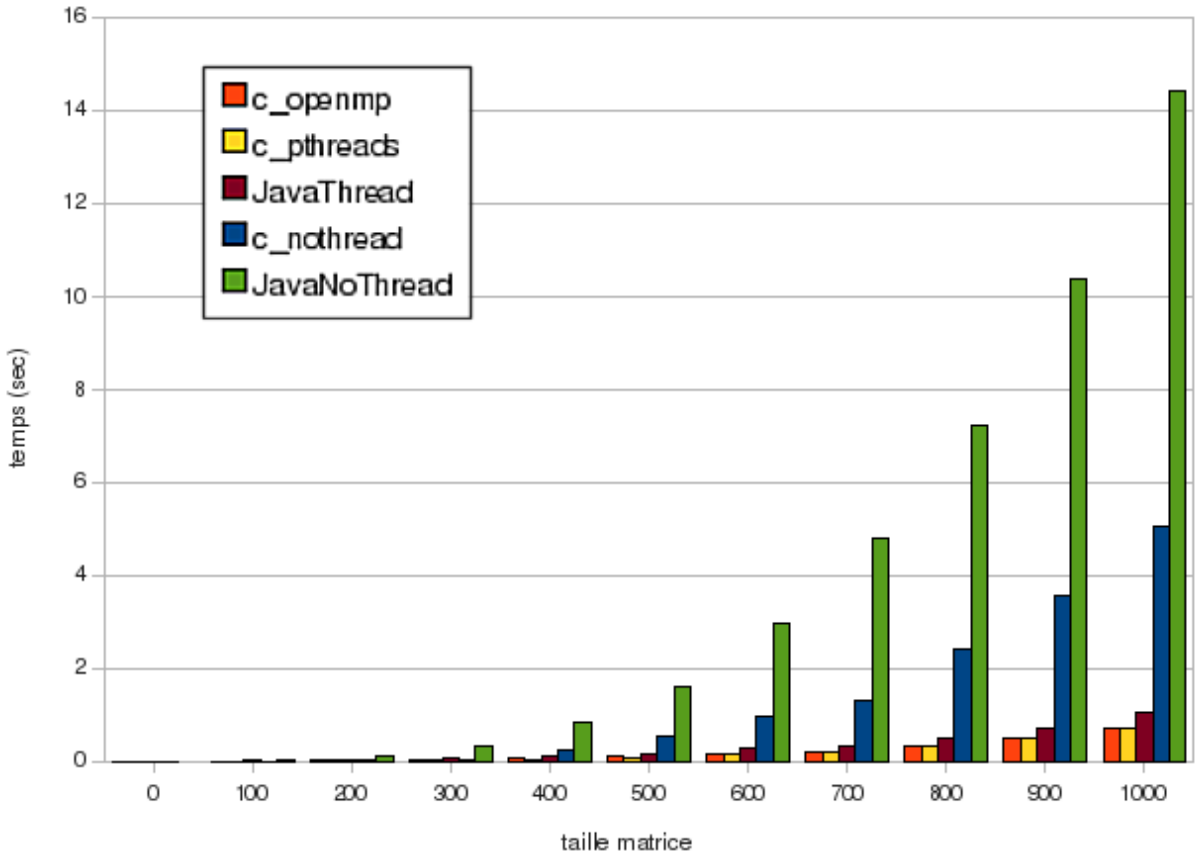


Linux P4 HyperThreading

Avec threads ⇔ Sans thread

Comparaison des technologies de processeurs

Tests Xeon 2 x Quad Core - 8 threads



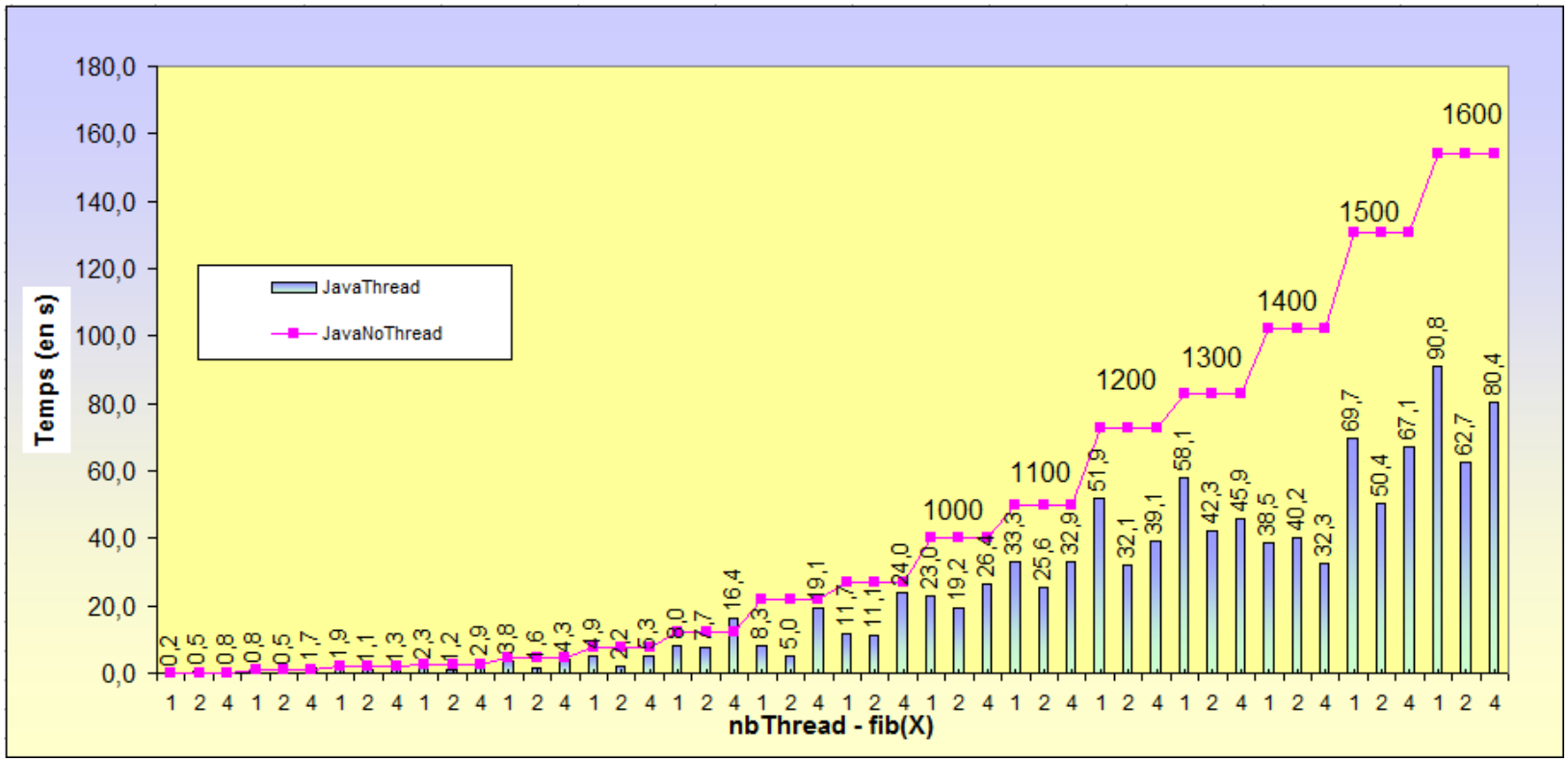
Linux Pentium Xeon X5460 3.16GHz (2 processeurs à 4 cœurs)

Gain :
Java : 14x
C : 7x

Performance des threads

Tests Core 2 Duo T5500

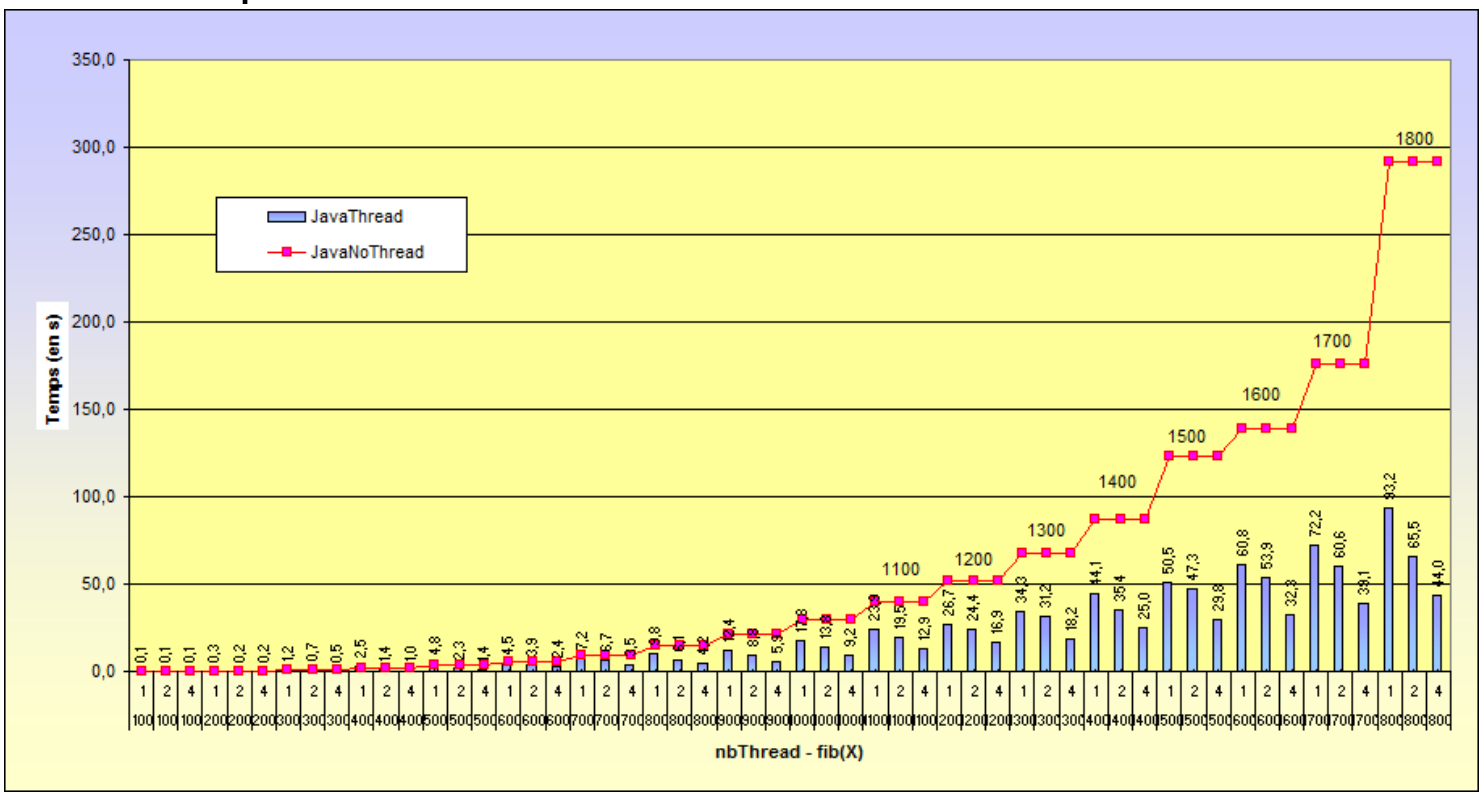
Mauvaise exploitation des threads



Windows T5500 1.66Ghz

Performance des threads
Pentium 4D (Dual Core)

Bonne exploitation des threads

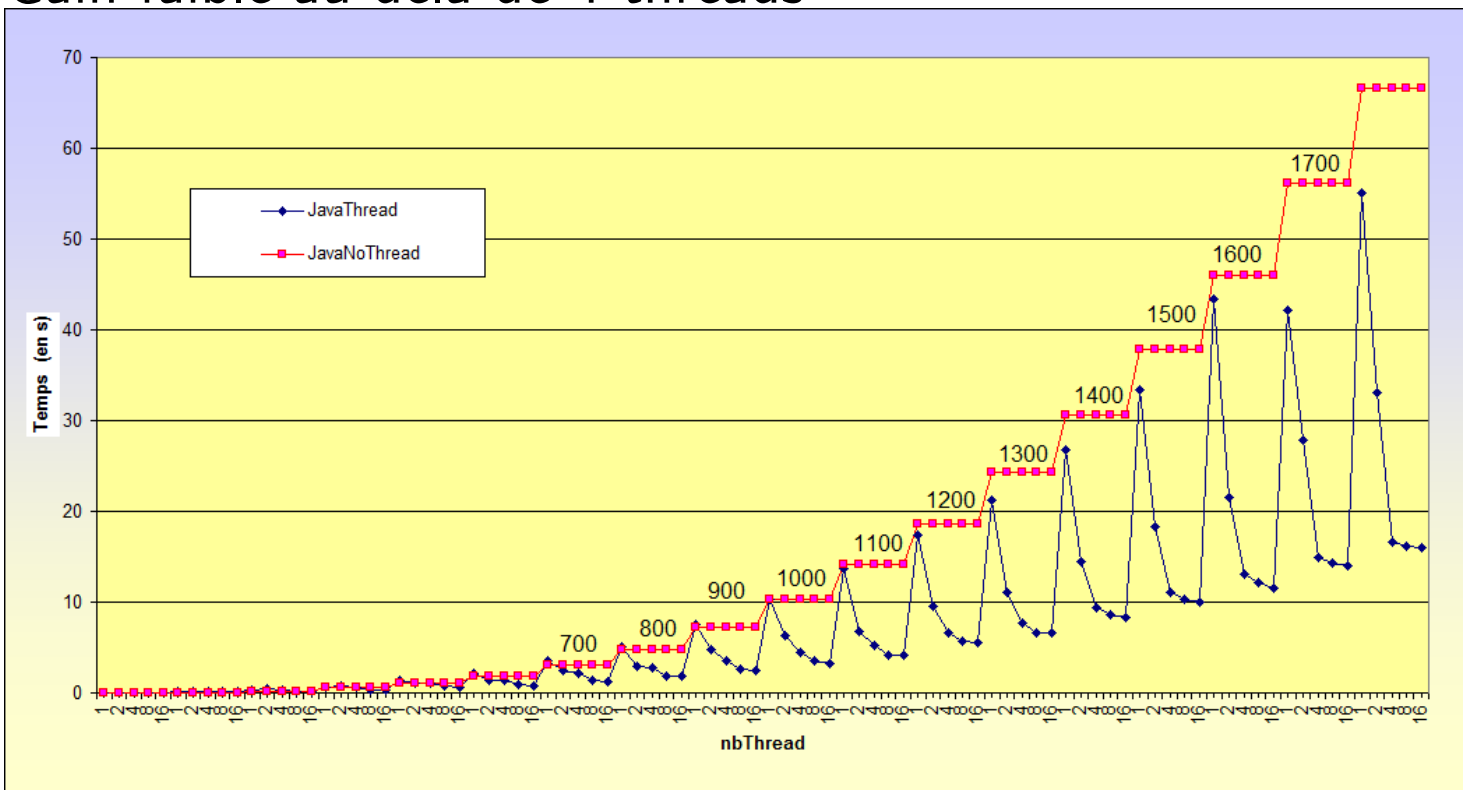


Linux P4 D 2.8GHz

Performance des threads

Pentium Xeon X5460 (2 x Quad Core)

Gain faible au delà de 4 threads

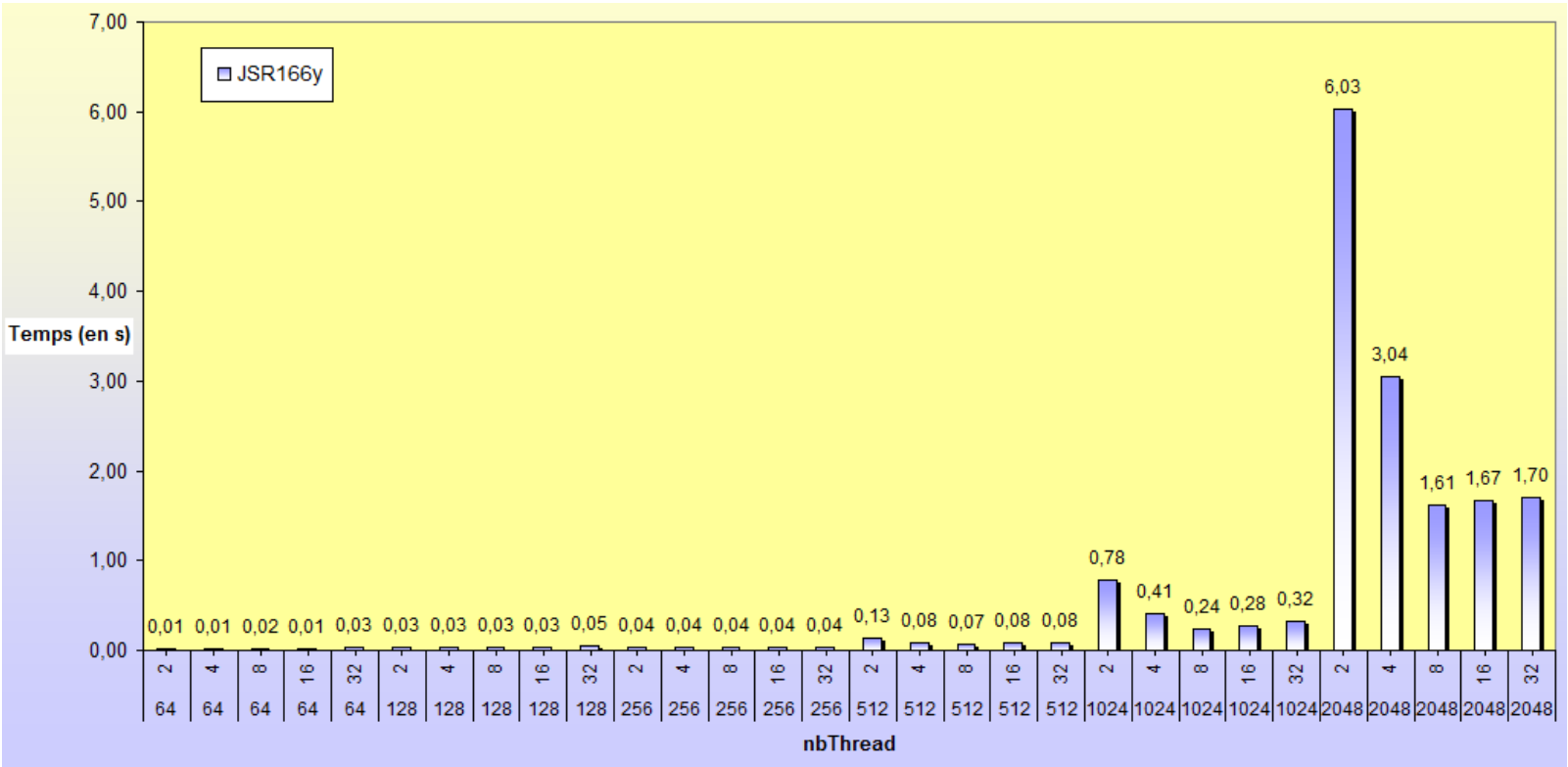


Linux Pentium Xeon X5460 3.16GHz (2 processeurs à 4 cœurs)

Bibliothèque JSR166y

Autre algorithme

Algorithme 10x plus performant.



Linux Pentium Xeon X5460 3.16GHz (2 processeurs à 4 cœurs)

Plan

Création de thread

- Temps de création d'un thread

c_openmp	c_pthreads	CSharpThread	JavaThread
21.4 µs	14.6 µs	8905.1464 µs	161.4 µs

FIG.: Temps de création d'un thread sur un Linux Core2Duo T9300

- Utilisation mémoire
  Difficile à mesurer

Plan

Synchronisation

- Mutex : ReentrantLock, pthread_mutex_t, omp_lock_t, Mutex...
- Sémaphore : Monitor, Semaphore...
- Section critique : synchronized{}, lock{}, #pragma omp critical{}...
- Modèle producteur-consommateur
- Autres bibliothèques de synchronisation :
  - Java : concurrent (LinkedBlockingQueue, ExecutorService, ...)
  - CSharp : Monitor, ReaderWriterLock, ...

Conclusion

- Bilan :
  - Technique (connaissances, tests limités...)
  - Humain (expérience...)
- Conclusion