Make distribué

Evaluation de performance

Cyril Gaunet
Nicolas Follet
Thomas Trompette
Samy Beyou

24/11/2017

I - Méthodes de test

Pour évaluer les performances de notre Make Distribué, nous avons d'abord utilisé le makefile d'exemple qui **liste les nombres premiers**. Ce makefile, hautement parallélisable, distribue la compilation de la cible finale sur 20 noeuds directement. Il s'agit de la configuration la plus simple pour tester les performances car peu de sérialisation, c'est donc sur ce même makefile de test que nous avons effectué la mesure d'un **intervalle de confiance**.
Les mesure se sont faites en plusieurs parties:

- Tout d'abord l'exécution successive pour un nombre de coeurs de plus en plus grand pour aboutir sur une courbe de performance en fonction du nombre de coeurs. Nous avons par ailleurs effectué plusieurs fois ce test pour avoir une idée plus précise de la distribution des temps d'exécution.
- Ensuite, nous avons comparé les valeurs obtenus avec celles du "make -j" parallèle de GNU. Ceci sur la même machine.
- Puis, pour aboutir à une intervalle de confiance, nous avons fixé le nombre de coeurs à 15 et répété l'exécution sur les mêmes processeurs une trentaine de fois.

Pour s'assurer du **bon fonctionnement de notre programme**, nous avons comparé la liste de nombre premiers obtenue par le make de GNU avec le nôtre en distribué, en utilisant la commande **diff,** il s'avère que les deux résultats sont **exactement similaires**, notre programme est donc cohérent.

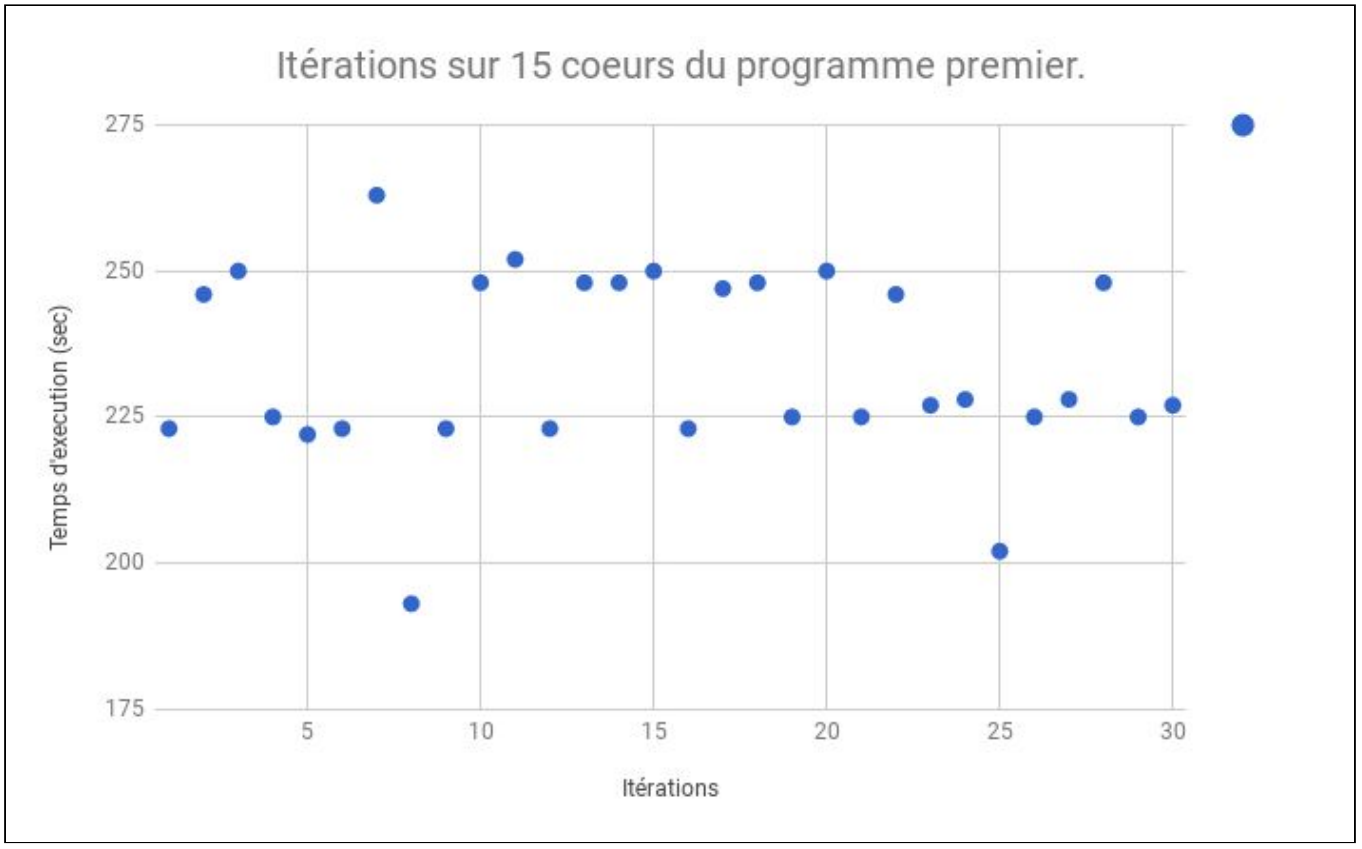
II - Scripts

En dehors des scripts de déploiement permettant de mettre en place l'environnement sur la plateforme grid'5000, nous avons conçu des scripts chargés d'automatiser les tests.
Le script **plot.sh** permet ainsi de générer les courbes du temps d'exécution en fonction du nombre de coeurs à partir du fichier timetest.txt regroupant les valeurs mesurées.
Le script **temps.sh** fait varier l'éxecution sur le nombre de coeurs.. Il place ensuite le temps d'exécution dans un fichier "time.txt". Le but final étant de générer des courbes à l'aide de gnuplot. Cependant, nous n'avons pas réussi à créer de script fonctionnel. Lorsque l'on exécute les lignes du script une par une le fichier se remplit bien mais le script lui, rajoute des pageFault dans le fichier de sortie. Nous pensons que l'erreur provient de la façon dont la commande time sort ses mesures.
Remarque : toutes les informations pour le déploiement et l'exécution sont disponibles dans le README

III - Résultats

1) Intervalle de confiance

L'intervalle de confiance a été calculé sur les machines graphene du site Nancy (processeur : Intel Xeon X3440 2.53 GHz). 30 itérations ont été faites.



On utilise ensuite la formule suivante:

$$\left[\overline{x} - 2\frac{\sigma(X)}{\sqrt{n}}; \overline{x} + 2\frac{\sigma(X)}{\sqrt{n}}\right]$$

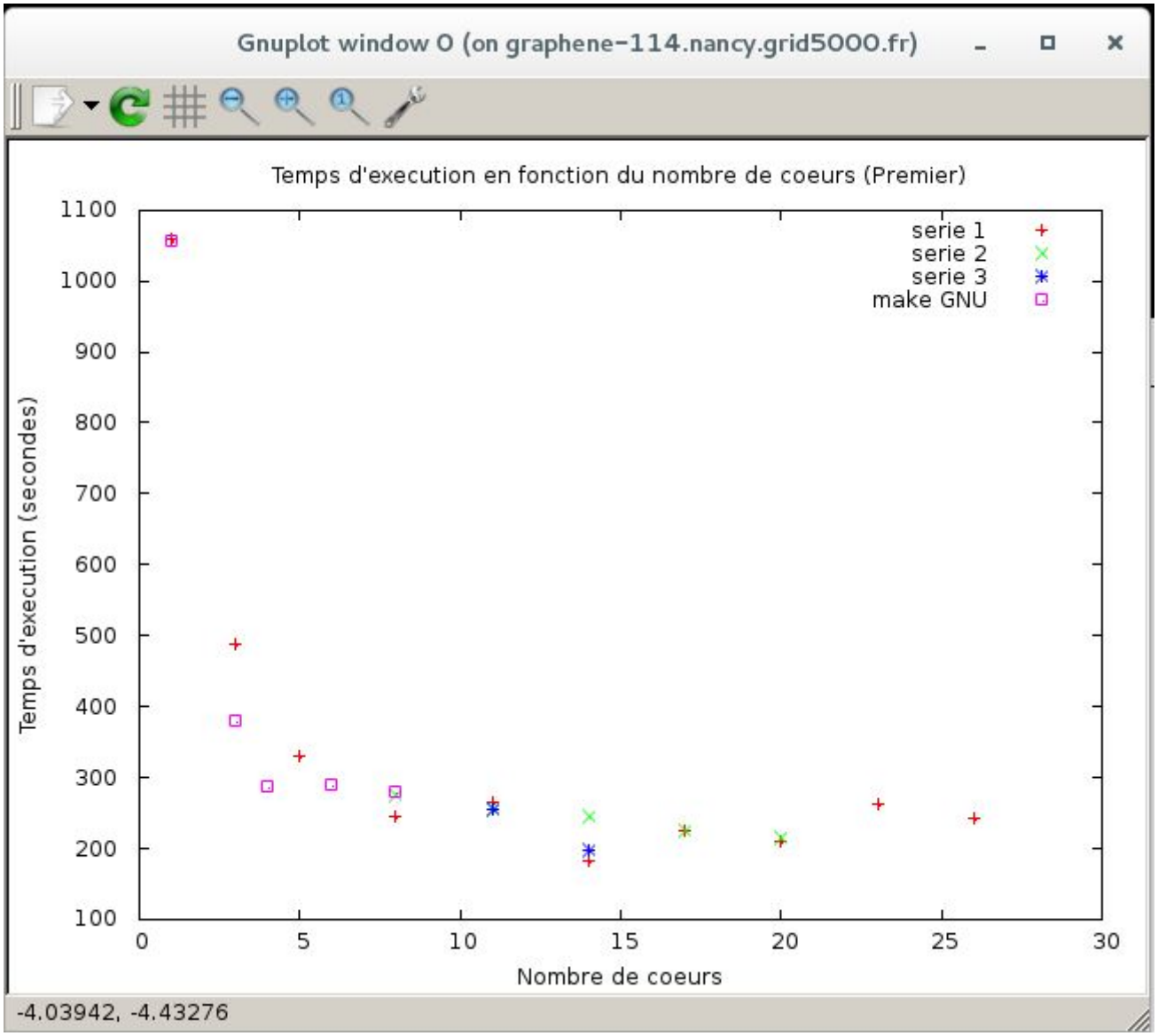
Intervalle de confiance à 95%
(Distribution normale)

On obtient finalement comme intervalle de confiance pour le temps d'exécution, **95% de chances que le temps d'exécution Te soit** :

227 s < Te < 241 s

De plus, **l'écart type** est de 17 secondes et la **moyenne** est de 234 secondes.

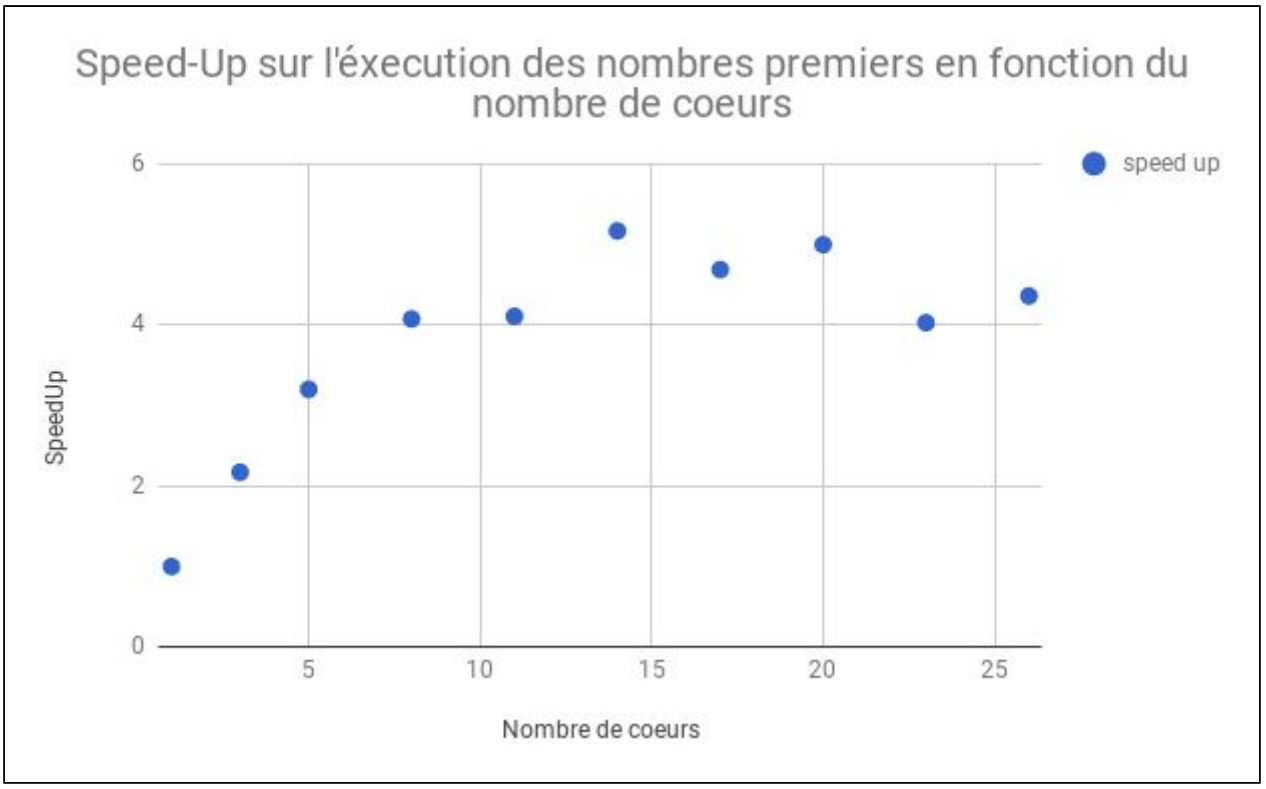
2) Courbe de Performance

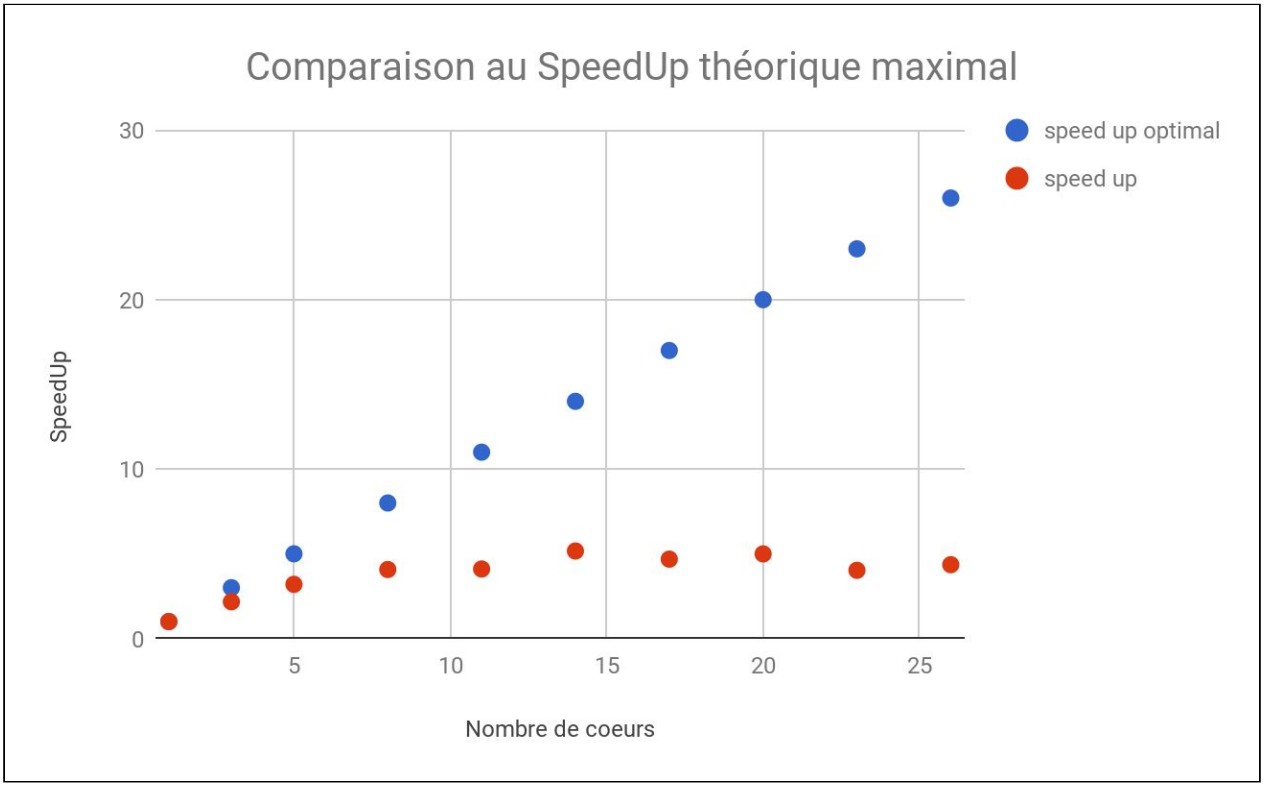


La courbe ci-dessus a été obtenue à l'aide de plusieurs séries de test en fonction du nombre de coeurs, sur premier. Nous avons utilisé GNUplot pour générer le nuage de points. Ils ont été effectués sur le site Nancy et notamment le node graphene.
On remarque notamment que, pour ce programme : hautement parallélisable, peu sérielle, et simple à exécuter de manière distribuée, nos **valeurs sont proches du GNU make** -j, parallèle que l'on a testé jusqu'à 8 processeurs. De plus, à partir d'un **certain nombre de coeurs le speedup n'augmente plus**, c'est à dire que le gain de temps est minime voire négatif si les processeurs sont trop nombreux.

Le **speed-up** maximal est atteint pour 14 processeurs, avec une durée d'exécution de 3 minutes 28 secondes, il est de :

Speed_up = T(seriel)/T(parallel) = 1059/208 = 5.09



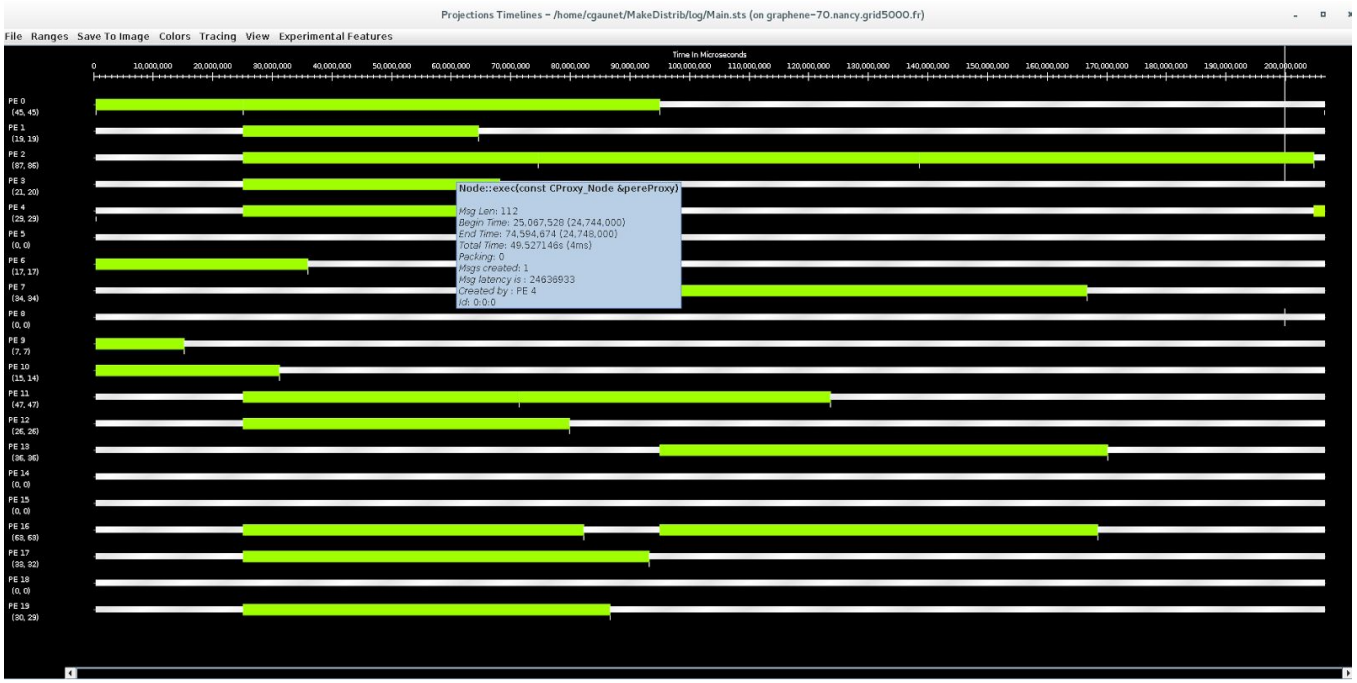


Nous avons également calculé **l'efficacité** pour ce nombre de processeurs, elle est de :

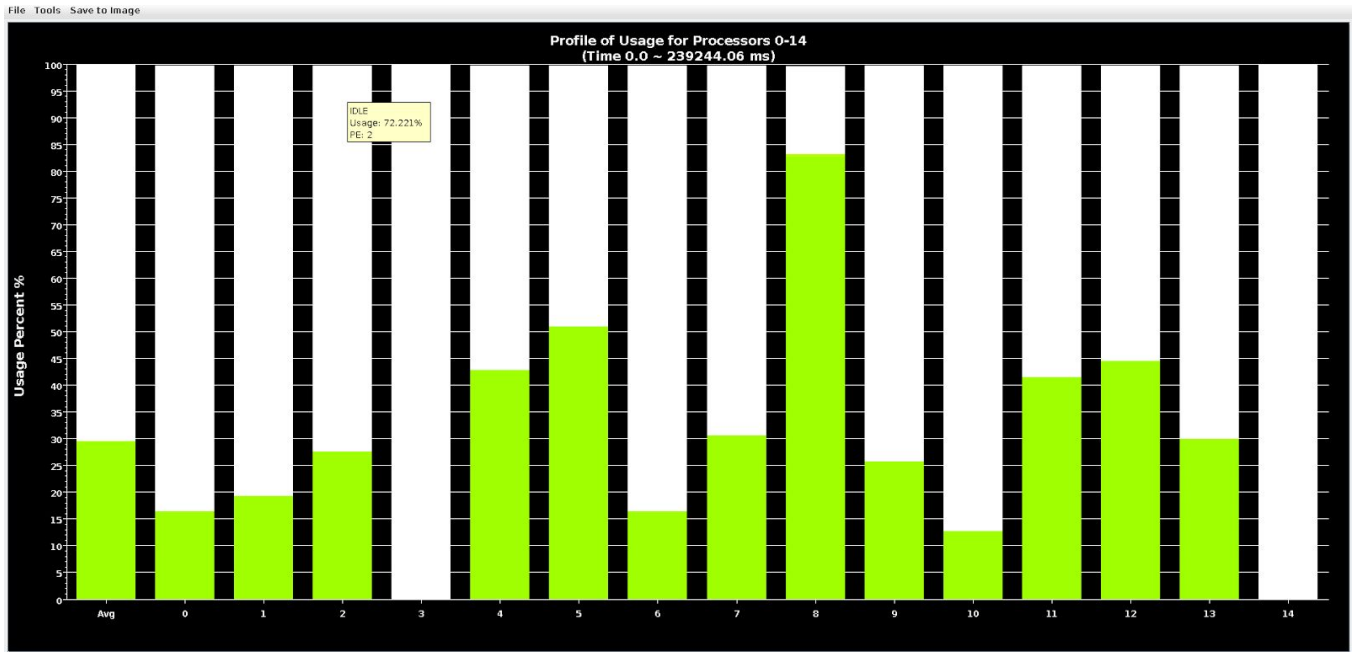
Efficacité = T(seriel)/(nombre_coeurs*T(parallel)) = 0.36

3) Load Balancing

L'outil Projections associé au langage Charm++ nous a permis de visualiser plus précisément la distribution des tâches sur les processeurs. Les captures d'écran ci-dessous relatent du calcul des nombres premiers, distribué optimalement sur 20 coeurs (car 20 dépendances).



Travail de chaque coeur au cours de l'éxecution



Charge de travail au cours de l'exécution

4) Tests Premier100

Nous avons finalement effectué quelques mesures sur le makefile premier modifié pour chercher davantage de nombre premiers (les 100 000 000 premiers). Voici un ordre de grandeur:

- avec 20 coeurs 2,5 GHz à Nancy: 20m 31s
- avec 100 coeurs de 2,4 à 2,5 GHz: 7m 18s

IV - Conclusion

L'utilisation de Charm++ pour la parallélisation est relativement simple à mettre en place, malgré sa documentation très faible en ligne.
Les résultats que l'on obtient, en les comparant au GNU make -j nous montrent que le programme **fonctionne relativement bien pour paralléliser** les tâches.
De plus, le fait qu'il n'y ait pas de différence de fichiers dans la liste des nombres premiers indique que **toutes les tâches** - ou Chares en Charm++ - **sont terminées**.

On peut en revanche émettre des doutes concernant **l'efficacité du load balancing**! En effet, en utilisant l'outil Projections, on remarque que les tâches ne sont pas distribuées de manière optimale, car certains processus travaillent plus que d'autres, et d'autres ne travaillent même pas.
De plus, notre code aurait pu être amélioré en supprimant **dynamiquement les dépendances** de l'arbre que l'on construit à partir du Makefile parsé.