Complément master PSMSC : Calcul Parallèle

Reda EL JAI

1 TP1 et 2 : MPI

Dans ce qui suit nous allons réaliser une analyse de performance des algorithmes P2P, bcast et P2P-i-la vus en tp avec différentes valeurs de lookahead. On se base sur la figure 1 avec ses subfigures de a, b et c. Elles représentent la vitesse d'éxécution des instructions en fonction de la taille de la matrice. On a modifié le script bench.sh en ajoutant une quatrième taille de matrice valant 4096 en plus de 1024, 2048 et 3072. Pour chaque algorithme on réalise 5 itérations. Et pour le P2P non bloquant, on teste avec 16 valeurs d'overhead différentes. On compare les performances sur un même grid. Puis on effectue une étude de scalabilité en passant à 3x3 puis 4x4. Cette analyse et complétée par les traces d'éxécution générées par MPE des différents algorithmes pour les exécutions sur les différents grid. Ces traces permettent de s'assurer essentiellement du bon fonctionnement des algorithmes.

1.1 Analyse de performance

Puisque l'augmentation de nombre de noeuds augmente la disparité entre les performances, pour évaluer un algorithme sur différentes tailles on choisit le grid 4x4. Puis on fixe la matrice de taille 4096 pour comparer les vitesses d'éxecution intergrid.

Une remarque générale : on remarque que l'augmentation de la vitesse moyenne en fonction de la matrice se fait linéairement.

1.1.1 P2P

La vitesse d'éxecution passe d'une vitesse moyenne de 20Gflops/s pour une valeur de m = 1024 à 140Gflops/s pour m = 4096. En passant d'un grid de 2x2 à 3x3 puis 4x4, la vitesse reste constante. les performances de P2P ne dépendent pas du nombre de noeuds.

1.1.2 P2P

La vitesse d'éxecution passe d'une vitesse moyenne de 20Gflops/s pour une valeur de m = 1024 à 140Gflops/s pour m = 4096. En passant d'un grid de 2x2 à 3x3 puis 4x4, la vitesse reste quasiment constante. Les performances de P2P ne dépendent presque pas du nombre de noeuds.

1.1.3 bcast

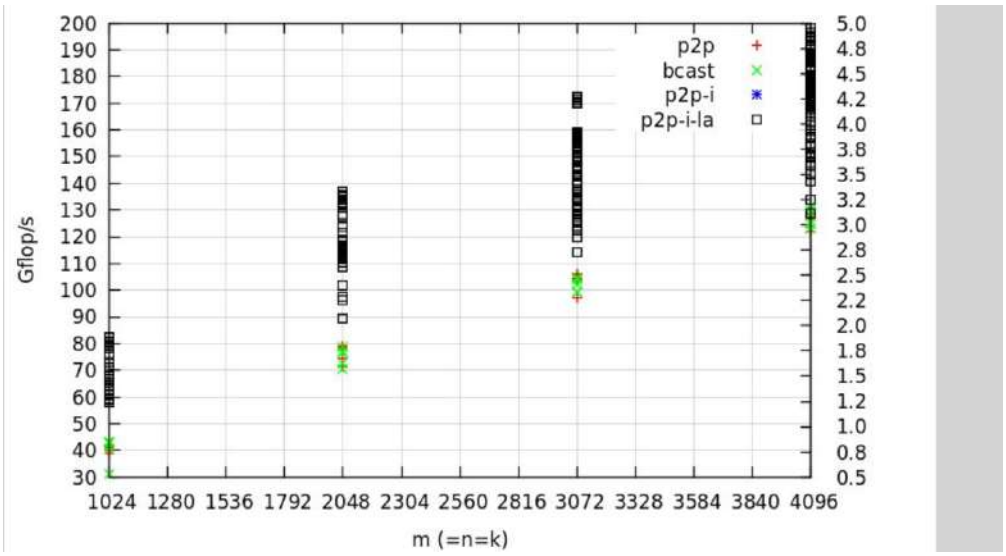
La vitesse d'éxecution passe d'une vitesse moyenne de 40Gflops/s pour une valeur de m = 1024 à 190Gflops/s pour m = 4096. En passant d'un grid de 2x2 à 3x3 puis 4x4, la vitesse passe de 120, 130,140 approximativement. Les performances de bcast s'ameliorent sensiblement en augmentant le nombre de noeuds.

1.1.4 P2P-i-la

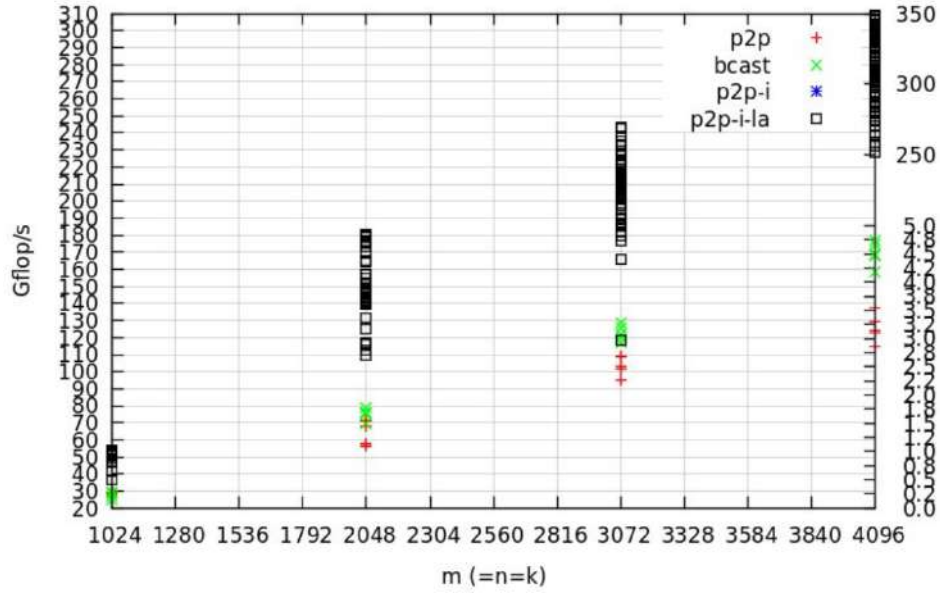
La vitesse d'éxecution passe d'une vitesse maximale de 60Gflops/s pour une valeur de m = 1024 à 350 pour m = 4096 en passant par 185 et 245 pour m valant 2048 et 3072 respectivement . En passant d'un grid de 2x2 à 3x3 puis 4x4, la vitesse passe de 250, 310, 395 approximativement. Les performances de P2P-i-la s'ameliore en augmentant le nombre de noeuds. L'augmentation du lookahead permet également d'augmenter la vitesse : en effet pour une taille fixée de 4096 et un grid 4x4, passer d'un lookahead de 1 à 16 permet de gagner jusqu'à 120 Gflops/s en allant de 280 Gflops/s à 400 Gflops/s. Ceci s'explique par une plus grande rapidité d'accès à la mémoire.

1.1.5 Récapitulatif

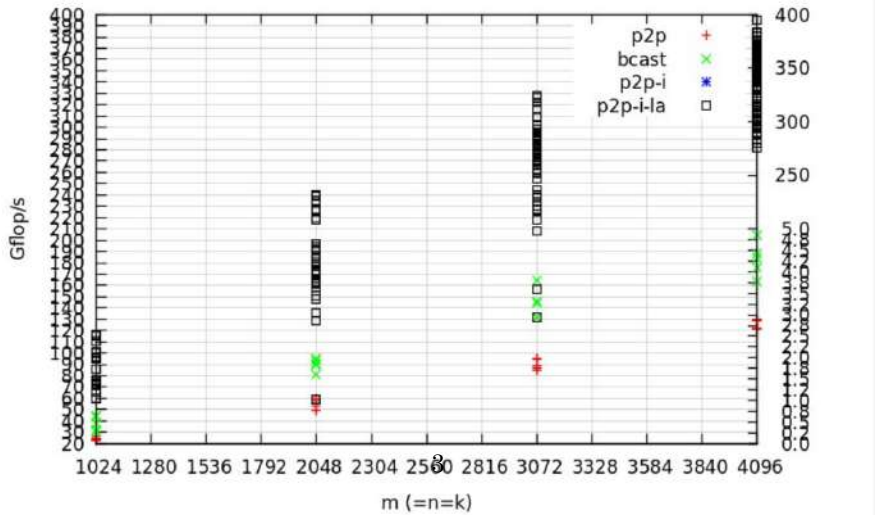
P2P-i-la reste le plus performant de algorithmes surtout en augmentant le lookahead. Vient ensuite le bcast et enfin le P2P. L'analyse des traces suivantes permet de confirmer ce classement.



(a) 2x2



(b) 3x3



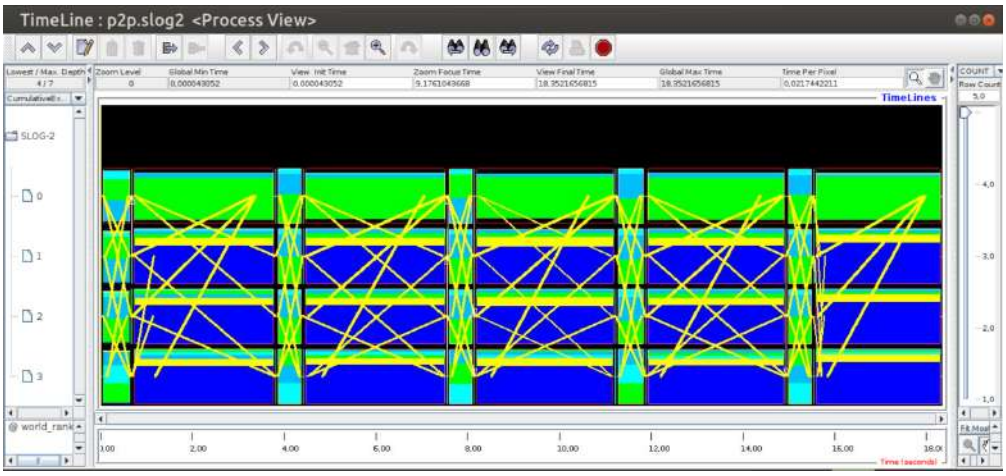
(c) 4x4

Figure 1: nombre d'opérations par seconde en fonction de la taille de matrice pour différentes tailles de matrice

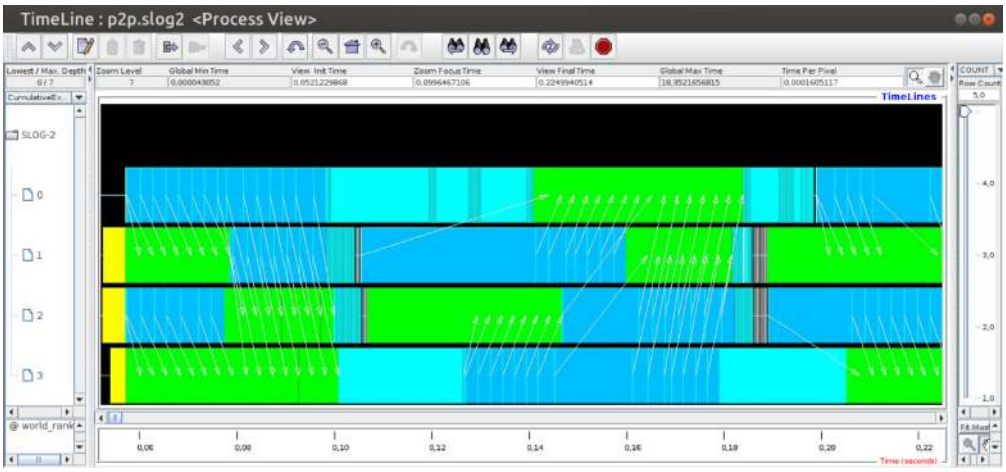
1.2 Traces des algorithmes

Les traces suivantes représentent les requêtes MPI qui s'éxécutent sur chaque noeud en fonction du temps. Elle permettent de valider le fonctionnement des algorithmes. L'algorithme P2P utilise un SSend, synchrone et bloquant(figure 2). Le temps perdu en attendant la réception de chaque message envoyé avant d'envoyer le second ralentit l'algorithme par rapport à l'algorithme non bloquant qui utilise un Isend et où il n'y a pas ce temps de latence (en gris figure 8). Pour le bcast, tous les noeuds sont en position bcast que ce soit l'émetteur ou le récepteur. Dans ce qui suit 2 traces, global des 5 itérations puis un zoom sur la partie intéressante, de chaque algorithme et pour chaque grid.

1.3 P2P

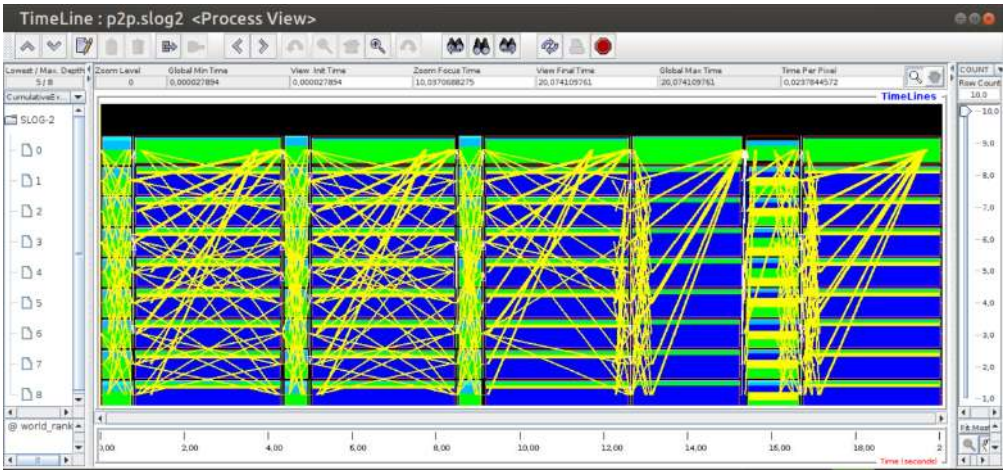


(a) trace des 5 itérations

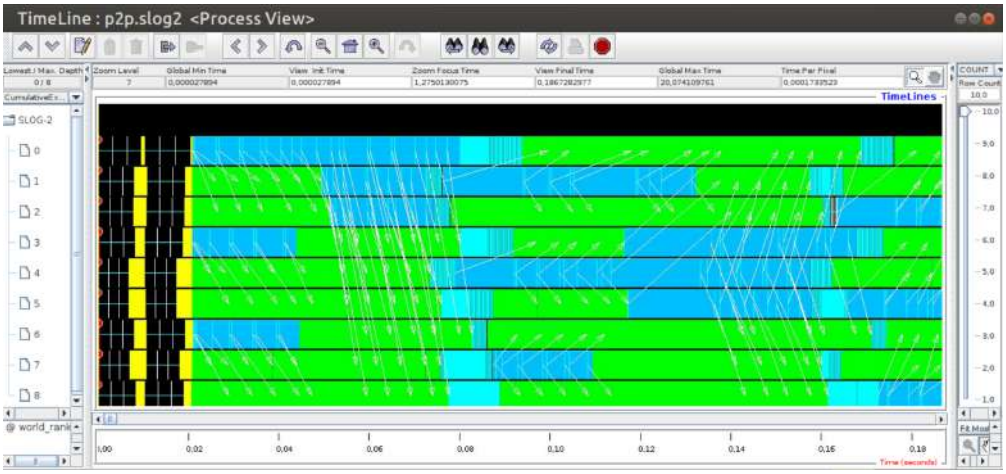


(b) zoom sur la 1ère itération

Figure 2: Trace d'éxécution de l'algorithme P2P pour un grid 2 x 2

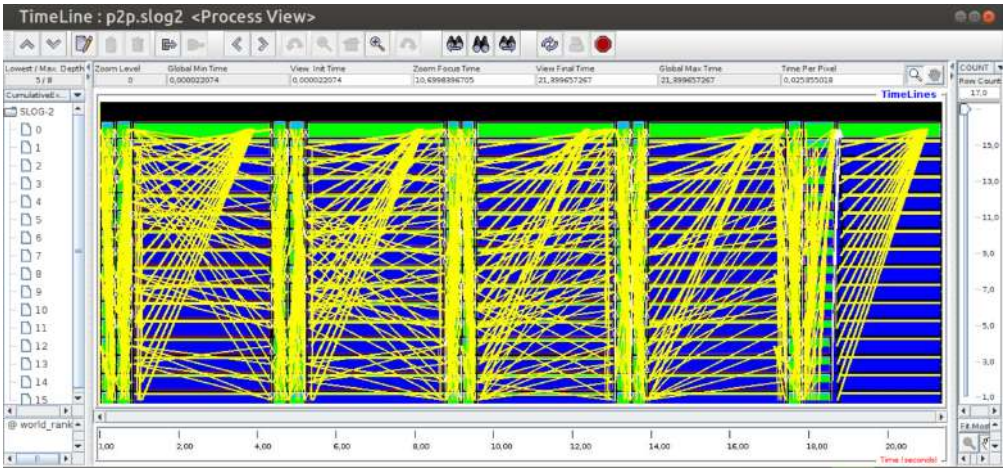


(a) trace des 5 itérations

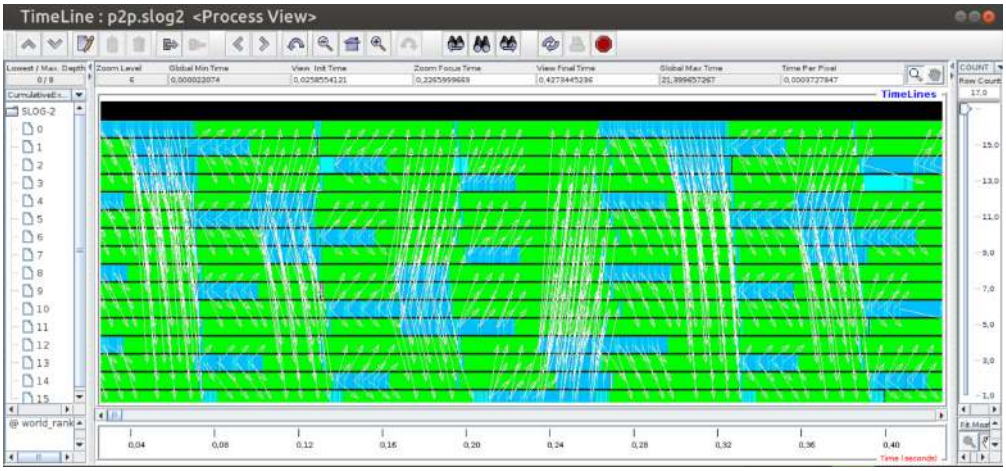


(b) zoom sur la 1ère itération

Figure 3: Trace d'éxécution de l'algorithme P2P pour un grid 3x3



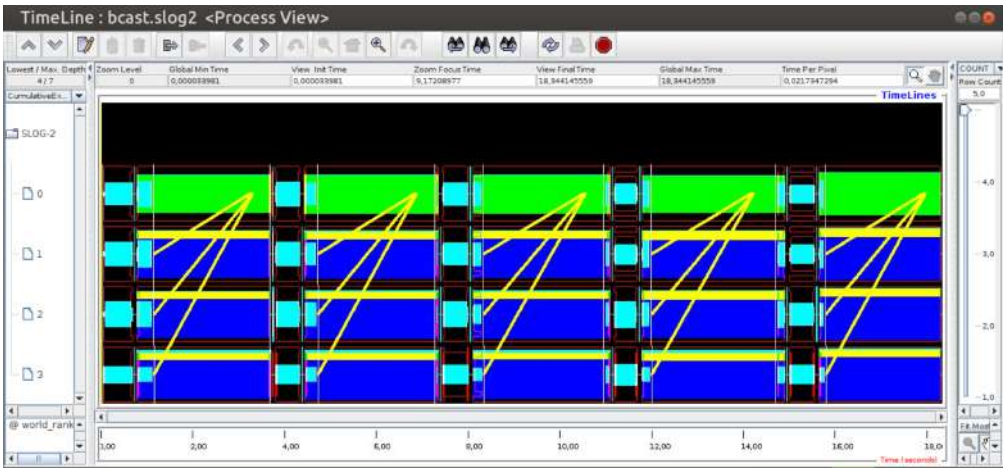
(a) trace des 5 itérations



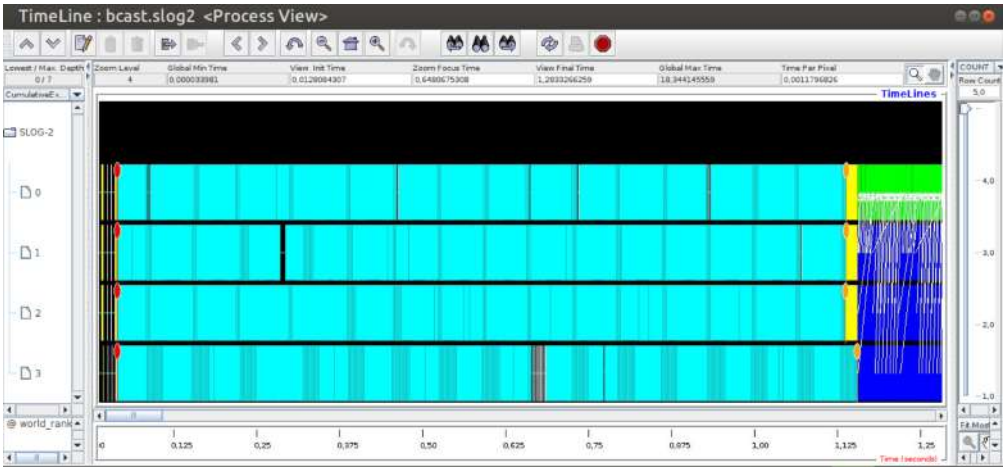
(b) zoom sur la 1ère itération

Figure 4: Trace d'éxécution de l'algorithme P2P pour un grid 4x4

1.4 Bcast

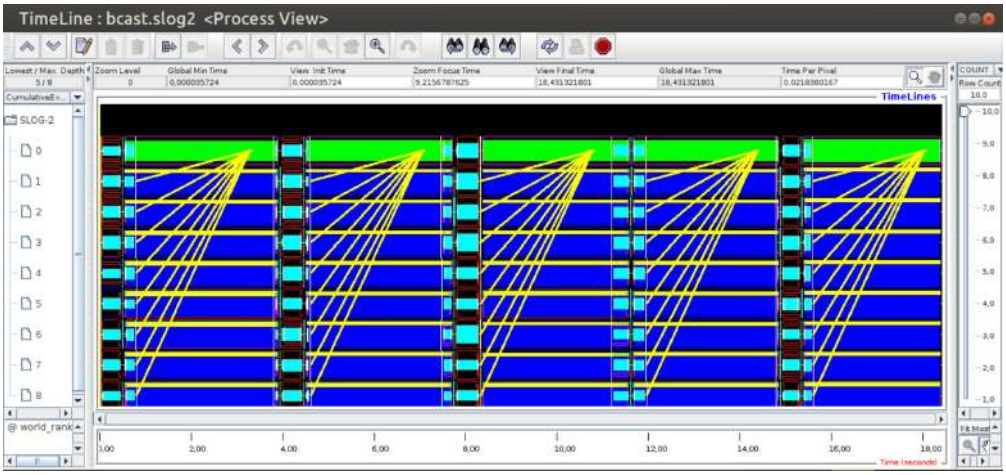


(a) trace des 5 itérations

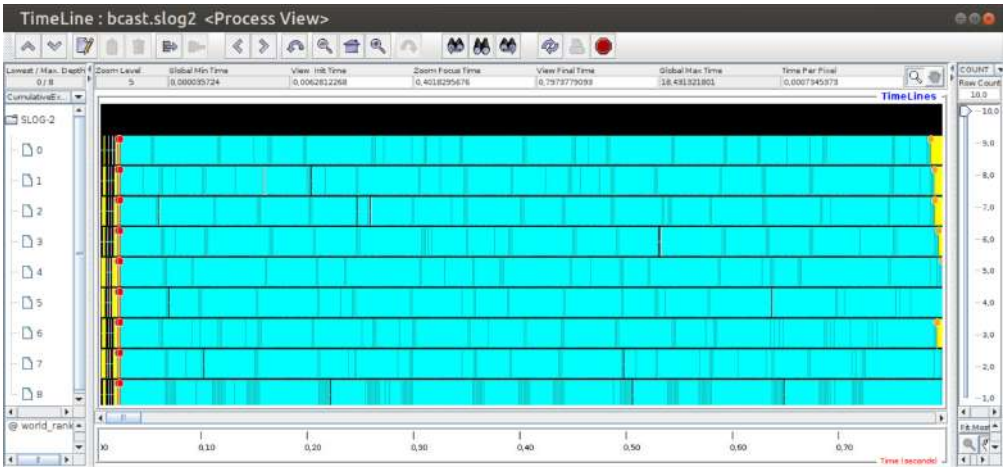


(b) zoom sur la 1ère itération

Figure 5: Trace d'éxécution de l'algorithme bcast pour un grid 2x2

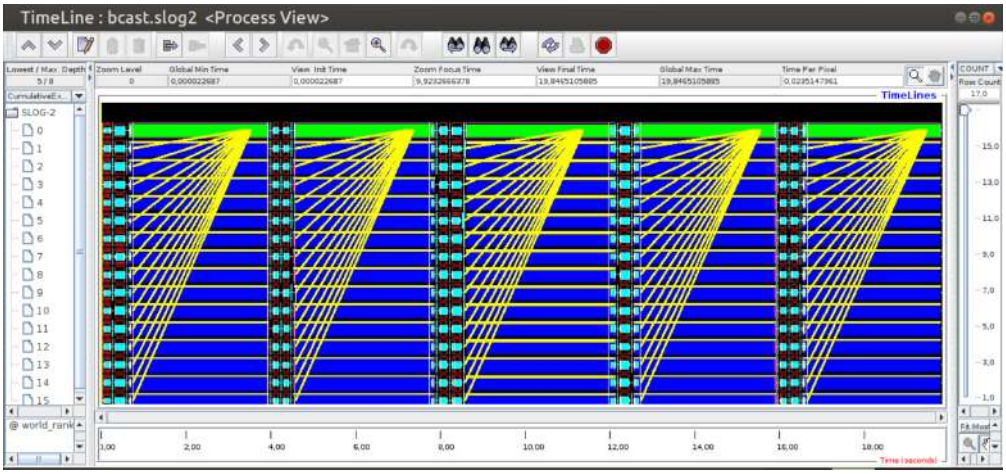


(a) trace des 5 itérations

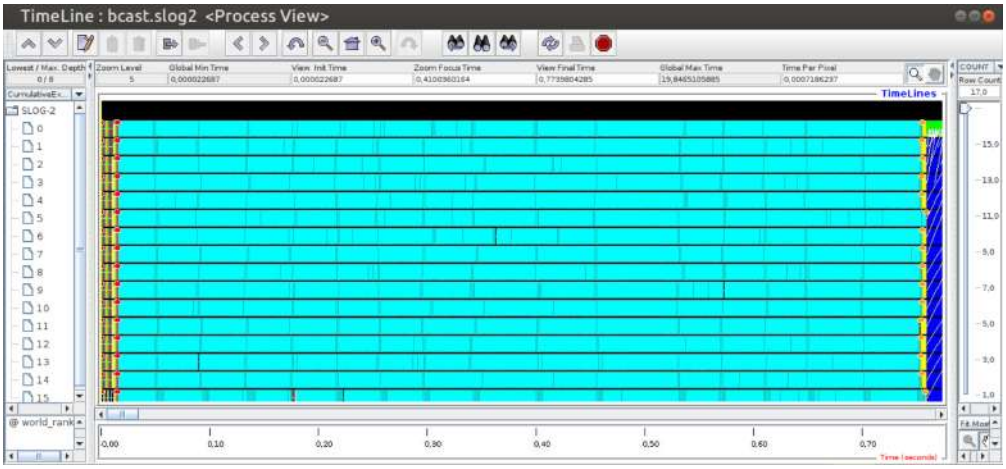


(b) zoom sur la 1ère itération

Figure 6: Trace d'éxécution de l'algorithme bcast pour un grid 3x3



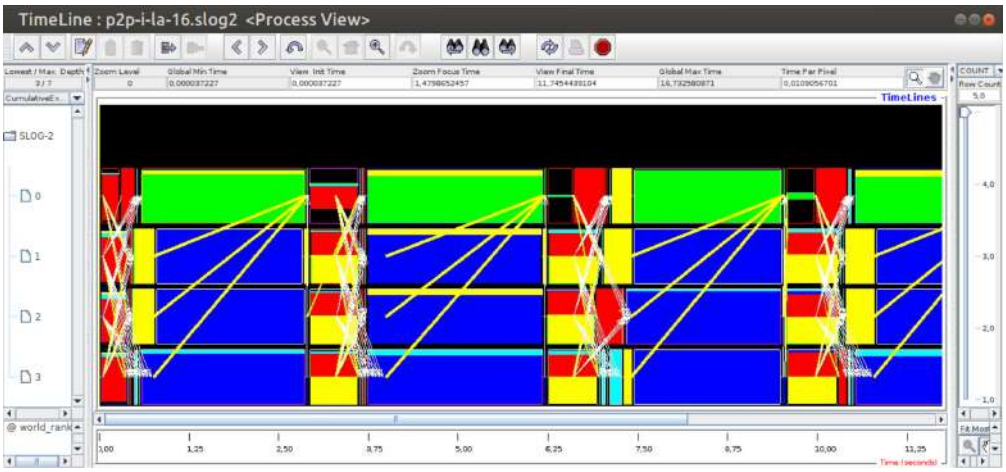
(a) trace des 5 itérations



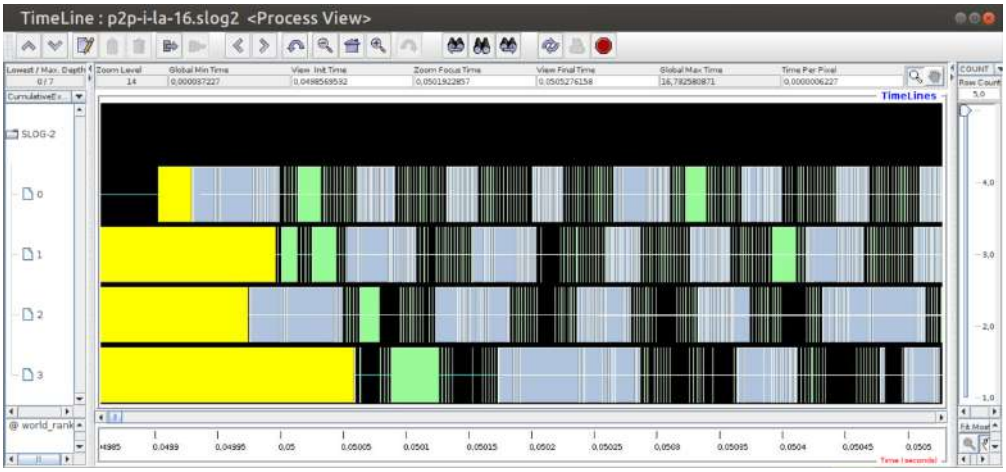
(b) zoom sur la 1ère itération

Figure 7: Trace d'éxécution de l'algorithme bcast pour un grid 4x4

1.5 P2P-i-la pour un lookahead de 16



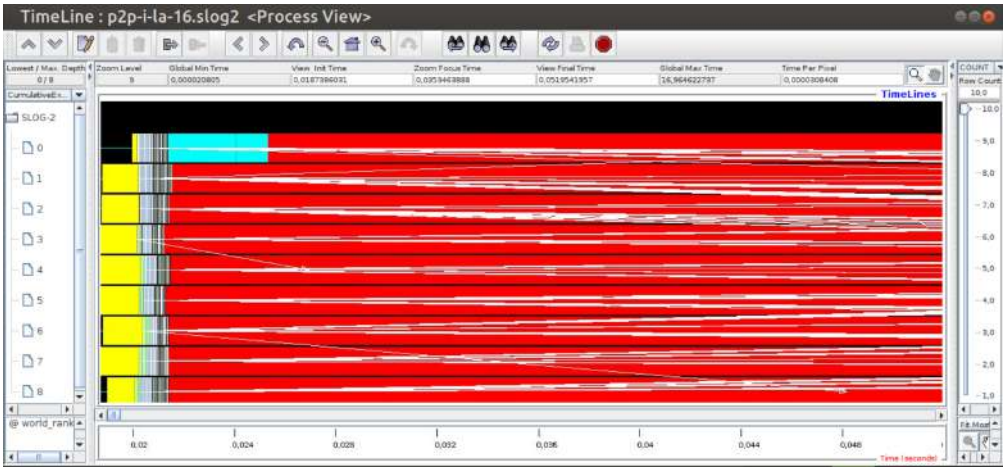
(a) trace des 5 itérations



(b) zoom sur la 1ère itération

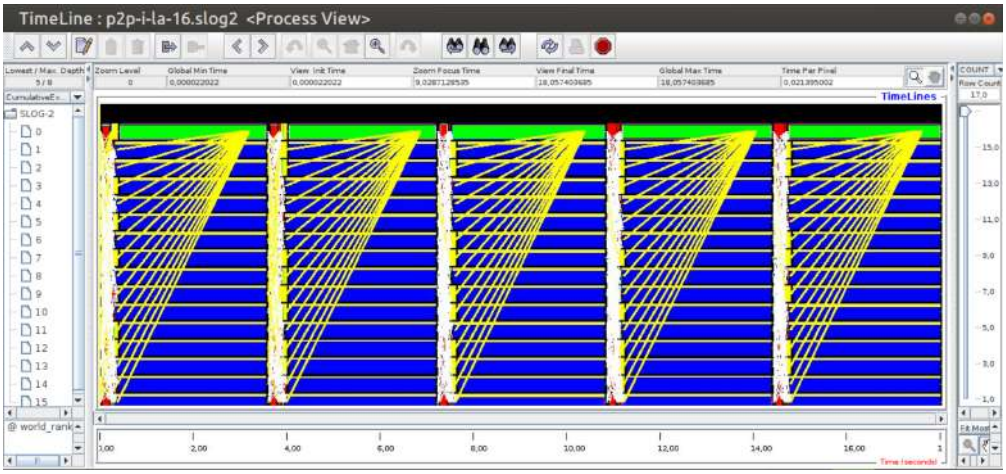
Figure 8: Trace d'éxécution de l'algorithme p2p non bloquant avec lookahead 16 et grid 2x2

(a) trace des 5 itérations



(b) zoom sur la 1ère itération

Figure 9: Trace d'éxécution de l'algorithme p2p non bloquant avec lookahead 16 et grid 3x3



(a) trace des 5 itérations

(b) zoom sur la 1ère itération

Figure 10: Trace d'éxécution de l'algorithme p2p non bloquant avec lookahead 16 et grid 4x4

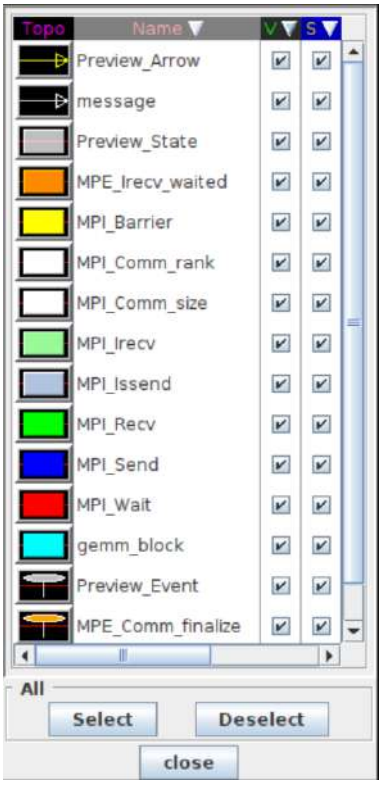


Figure 11: Légende

2 TP3 : GPU

2.1 GPU vs CPU

Le fichier gpu_vs_cpu permet de comaprer les tracés du temps d'éxécution de Gemm en fonction de la taille de la matrice. Le script C a été changé pour prendre en paramètre la taille maximale de l'intervalle des tailles sur l'axe des abscisses. L'éxécution suivante produit la fiure 12 : ./gpu_vs_cpu .

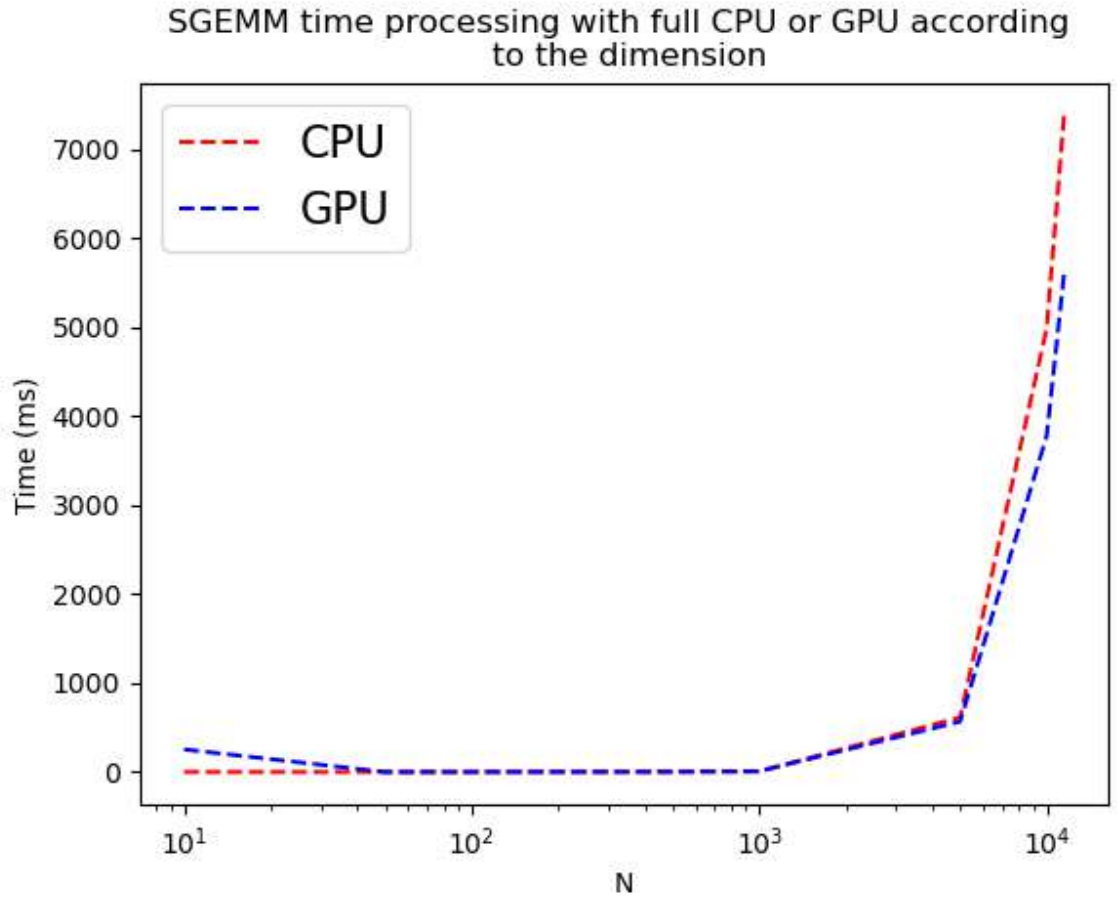


Figure 12: Cpu vs Cpu

2.2 GPU with CPU

Le script gpu_with_cpu.C a été modifié et prend en argument la taille de la matrice et produit en sortie un fichier csv indéxée par la taille de matrice. Le fichier gpu_with_cpu.py prend en argument un tableau de toutes les matrices dont on veut comparer les tracés et qui ont été généré précédemment par l'éxécutable du fichier C. L'enchaînement d'instructions qui suit produit la figure 13. On y remarque notamment un résultat qui semble contre-intuitif, puisque une utilisation totale de gpu n'est pas l'optimum du temps d'éxécution. Une co-utilisation de gpu et cpu à 60% de gpu paraît être plus efficace.

./gpu_with_cpu 1000
./gpu_with_cpu 5000
./gpu_with_cpu 11500
python3 gpu_with_cpu.py 1000 5000 11500

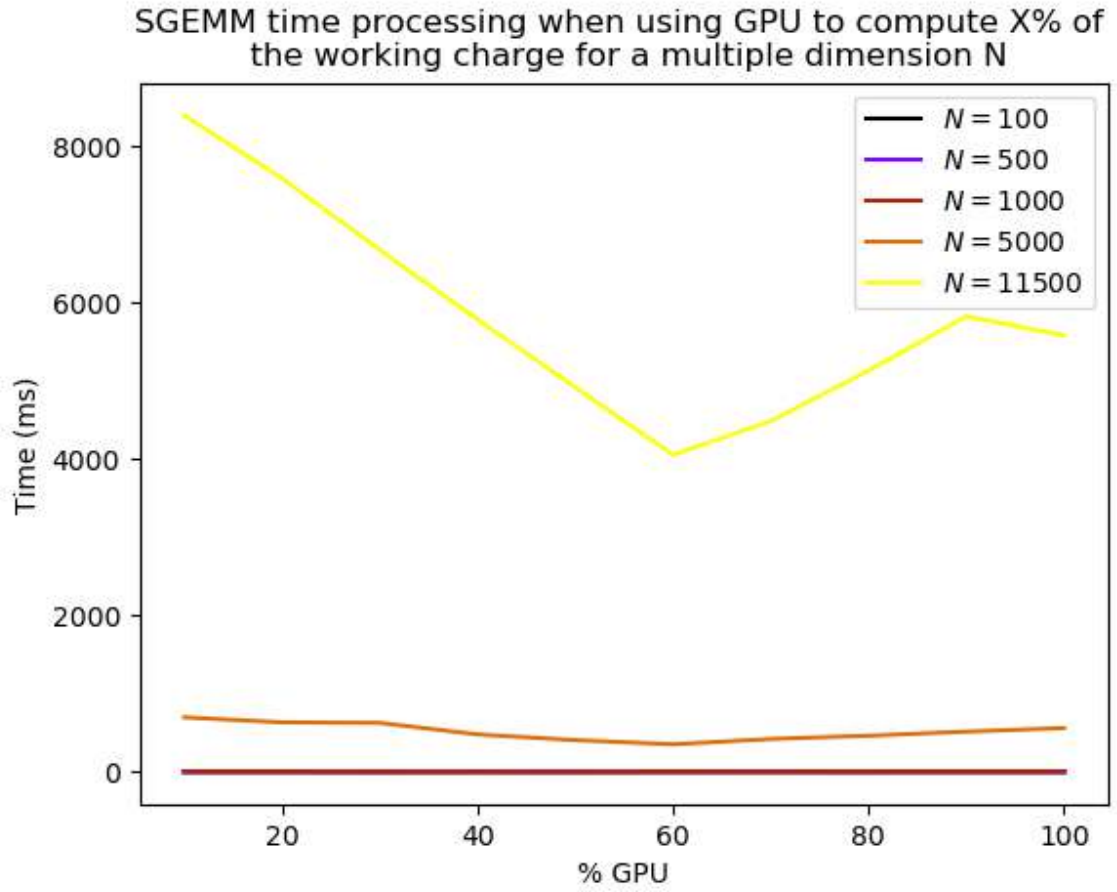


Figure 13: Gpu with Cpu