RESUME DES OUTILS QUI EXISTENT POUR TESTER LES PERFORMANCES DE CODES PARALLELES

Laure Coquart, en collaboration avec l'équipe assistance scientifique et l'équipe système

Décembre 2005

1. Introduction

Il y a peu d'outils gratuits sur IBM qui fassent et profiler, c'est-à-dire qui permettent de récupérer les données hardware, comme le nombre de fois où une routine a été appelée ou le nombre d'opérations flottantes réalisées, et qui permettent également de récupérer les informations concernant les traces MPI du programme, pour une étude plus fine des performances et des communications dans le programme. Il y en a également peu parmi ceux qui existent qui soient simples et faciles à mettre en place, sans implémenter le programme. Les données obtenues sur les différents logiciels présentées ci-dessous proviennent d'internet ainsi que des documents référencés en [1] et [2].

2. Outils de profiling et de traces MPI

2.1 PE Benchmarker (PCT, PVT)

PE Benchmarker est un produit IBM AIX, non payant. Il est construit sur la bibliothèque **DPCL** (Dynamic Probe Class Library). Il utilise les outils **PCT** (Performance Collection Tool), **UTE** (Uniform Trace Environment) et **PVT** (Profile Visualization Tool). Il n'y a pas besoin d'instrumenter les codes pour utiliser ces différents outils.

PCT permet de collecter les données hardware ainsi que les traces MPI selon que l'on sélectionne le mode *select profile* ou *select trace* dans le script de soumission.

Select profile donne accès à un grand nombre de données différentes, aussi bien le nombre de fois où une fonction est appelée que le nombre d'opération flottantes réalisées, et le nombre de données transférées de la mémoire cache vers le CPU. Les fichiers de sortie au format *file.cdf.** sont ensuite visualisables par **PVT**. Il y a un fichier de sortie par processus MPI.

Select trace donne accès aux traces MPI des différentes fonctions ou routines du programme parallèle. La récupération des évènements MPI est très simple avec ce logiciel car il n'y a pas besoin d'implémenter le code. Le code est « vampirisé » de l'extérieur en implémentant des sondes sur les fichiers les plus pertinents. Les résultats sont ensuite visualisables par **JUMPSHOT** mais

il faut d'abord les transformer au format file.slog2. Une description de jumpshot est donnée dans la partie 4 du document.
L'environnement **UTE**, avec *uteconvert* permet de transformer les fichiers de sortie en fichiers au format ute, puis *traceTOslog2* permet de les transformer en slog2 lisibles par jumpshot. Il y a en général un seul fichier de sortie.
Les scripts peuvent être soumis soit en interactif en utilisant la commande :
pct –c –s script_pct.cmd
soit à la fin d'un fichier de soumission de job en batch, après les commandes habituelles en tapant également :
pct –c –s script_pct.cmd

Un script type *pcpourtpvt_script.cmd* pour PCT utilisé avec *select profile* ainsi qu'un script type *pctpourtrace_script.cmd* pour PCT utilisé avec *select trace* se trouvent dans le répertoire /home/coquart/exemples/PCT-x.

Nous donnons ici une rapide description du logiciel graphique PVT, que l'on lance simplement par la commande *pvt &*. La figure (1) montre la fenêtre du logiciel, avec le nombre d'appels aux différentes routines du code qui ont été implémentées.
On peut observer que les routines où on passe le plus de temps sont cpmd.f, fftutil.f, fftmain.f. Il sera ensuite intéressant d'implémenter le traçage des évènements MPI au niveau de ces fichiers.

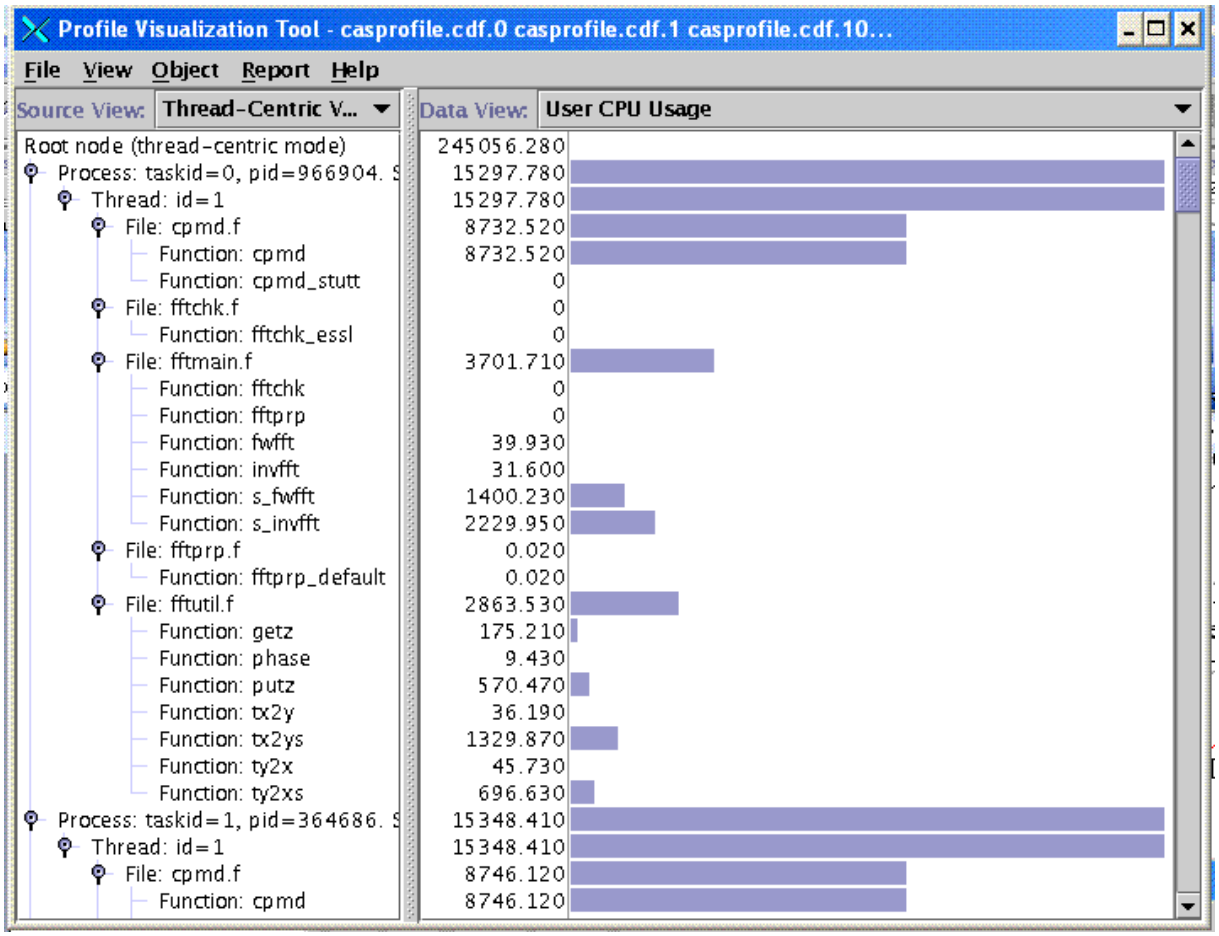


Figure 1 : Visualisation PVT

2.2 IBM's Hardware Performance Monitor (HPM, HPCT)

HPM est un produit IBM qui appartient au logiciel **HPCT** (High performance Computing Toolkit) qui est payant. Cet outil est intéressant dans la mesure où il fournit les performances globales du programme avec *hpmcount* sans avoir besoin d'implémenter le programme. Cela permet une première analyse du code. Par contre il faut instrumenter le programme pour avoir des traces MPI (avec libhpm et hpmviz), ce qui demande une bonne connaissance de celui-ci.

Le package **HPCT** contient aussi Xprofiler. Xprofiler donne à peu près les mêmes résultats que Prof et Gprof (voir ci-dessous) mais avec une partie graphique. **HPCT** contient également un logiciel MPI Tracer/Profiler pour mesurer les performances MPI sans avoir besoin d'implémenter le code. Ces résultats sont ensuite visualisables par PeekPerf GUI.

2.3 Tuning and Analysis Utilities (TAU)

C'est un logiciel dit très puissant et il est gratuit. Il est fourni par " The University of Oregon Performance Research Laboratory, the LANL Advanced Computing Laboratory and the Research Center Julich in Germany" et on peut le télécharger à l'adresse : http://www.cs.uoregon.edu/research/tau/home.php.
Il est basé sur l'interface PAPI (Performance Application Programming Interface) et il permet de faire du profiling, des traces MPI et de la visualisation.

Malheureusement, il faut instrumenter les programmes sources pour avoir des informations sur les traces MPI, ce qui complique beaucoup la tâche et nécessite de bien connaître les codes.

Ses fichiers de sortie sont visualisables par Vampir (payant), Paraver (payant) et Jumpshot (gratuit).

2.4 Paradyn

Ce logiciel utilise directement les fichiers de sortie binaires de la librairie Dyninst API (Dynamic Instrumentation Librairy). Il est gratuit et téléchargeable à l'adresse : http://www.paradyn.org/.
J'ai trouvé de la doc sur internet sur comment le faire marcher en interactif. Il n'y a pas besoin d'instrumenter le code. Cette documentation se trouve dans le répertoire /home/coquart/exemples/doc.

Pour l'instant problèmes d'implémentation car il manque une librairie. Voici le type de sortie qu'il génère (pas très lisible) :

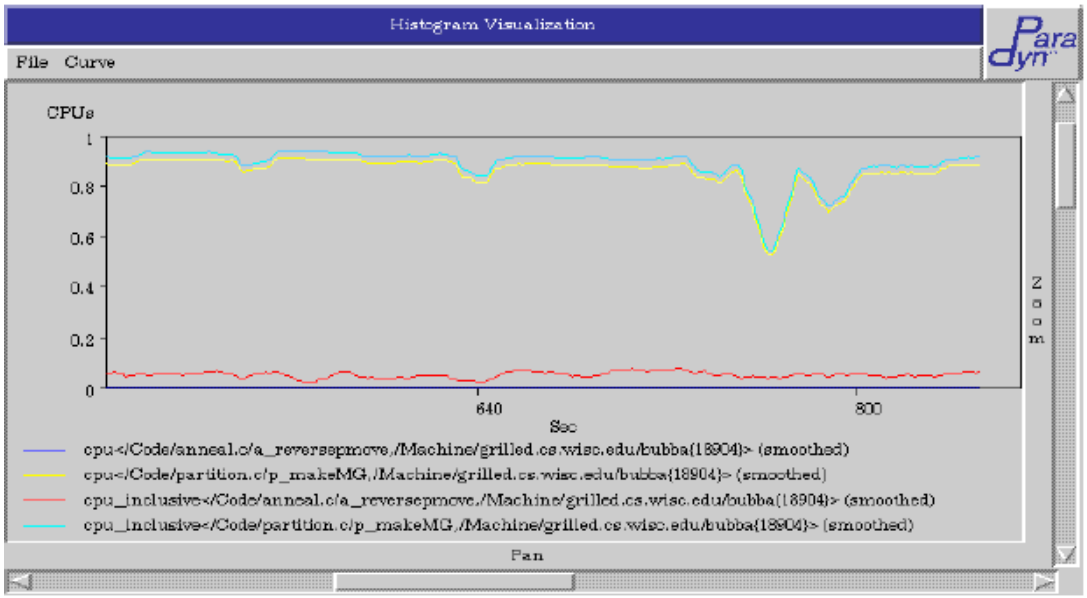


Figure 2 : Exemple de visualisation du temps CPU consommé avec Paradyn

2.5 SvPablo

Ce logiciel est gratuit mais il faut instrumenter le code pour l'utiliser. On peut le télécharger à l'adresse :
http://pablo.renci.org/Project/SVPablo/SvPabloOverview.htm.
Une documentation sur SvPablo se trouve dans le répertoire /home/coquart/exemples/doc.

2.6 SCALEA

Il faut implémenter le code pour l'utiliser. On le trouve à l'adresse : http://dps.uibk.ac.at/projects/scalea/.

2.7 DEvelopment Environment for Parallel programs (DEEP/MPI)

C'est un logiciel commercial payant (il est fourni par Crescent Bay Software http://www.crescentbaysoftware.com/end_user.html).

Il est construit sur l'interface PAPI (Performance Application Programming Interface). Il suffit d'utiliser la bibliothèque mpiprof à l'édition de liens donc il n'est pas nécessaire d'implémenter le code.

3. Outils de profiling

3.1 ToolGear

Ce logiciel est gratuit et a été développé au LLNL (Lawrence Livermore National Laboratory). On peut le télécharger à l'adresse suivante : http://www.llnl.gov/CASC/download/download_home.html. Il est construit sur des outils standards tels que DPCL (Dynamic Probe Class Library) ou PAPI (Performance Application Programming Interface).
Problèmes de librairies à l'installation.

3.2 Dynaprof

Ce logiciel est basé sur l'interface PAPI (Performance Application Programming Interface). Il est gratuit et téléchargeable à l'adresse : http://www.cs.utk.edu/~mucci/dynaprof. Il ne nécessite pas d'implémenter les codes. Il donne des résultats sous forme de statistiques.

Malheureusement je n'ai pas réussi à le faire marcher pour l'instant.

3.3 Prof, Gprof et Tprof

Ces trois commandes sont deux outils de base sur la majorité des systèmes UNIX tels qu'IBM AIX et sont gratuits contrairement à Xprofiler, comme on l'a déjà vu (qui fait partie du package HPCT qui est payant). On obtient des informations sur les codes en ajoutant simplement une option à la compilation (dans FFLAGS):

-p pour prof ; *-pg* pour gprof

A l'exécution, les fichiers de sortie obtenus s'appellent respectivement mon.out.nuproc pour prof et gmont.out.nuproc pour gprof par défaut. Les commandes prof/gprof sont ensuite utilisées pour les lire. Par exemple :

gprof nom_exe gmon.out.0 gmon.out.1 gmon.out.2 ….

Ci-dessous un exemple d'une toute petite partie du fichier de sortie de gprof, sur un programme très simple qui fait une somme sur deux processeurs :

```
-----------------------------------------------

                0.00    0.00       2/852         .realloc_y_heap [332]
                0.00    0.00     323/852         .free_y_heap [25]
                0.00    0.00     527/852         .malloc_y_heap [17]
[12]     0.0    0.00    0.00     852           .__heap_unlock [12]
                0.00    0.00       1/1           ._global_unlock_common [430]

-----------------------------------------------
```

La routine .__heap_unlock a été appelée par .realloc_y_heap (2 fois), par .free_y_heap (323 fois), par .malloc_y_heap (527 fois) tandis qu'elle-même a appelé ._global_unlock_common.

Ces fichiers sont difficilement lisibles mais ils sont très utiles lorsque les logiciels graphiques deviennent payants.

Pour créer le tmon.out.prof, il ne suffit pas d'exécuter le programme. Il faut utiliser la syntaxe suivante :

Poe tprof –v –m –s –p nom_exe *–x* nom_exe

Et on visualise le fichier de sortie après. De même que pour prof et gprof les données recueillies sont difficilement lisibles.

3.4 Xprofiler

Il fait partie du logiciel **HPCT** (High performance Computing Toolkit) donc il est payant.
Xprofiler est un logiciel graphique basé sur gprof donc il s'utilise de la même façon et il donne le même type de résultats que ceux décrits en (2.3) qui me semblent difficilement lisibles. Sinon, il donne en plus le type de sortie ci-dessous, figure (3), où la taille des boîtes est proportionnelle au temps CPU passé dans les fonctions correspondantes :

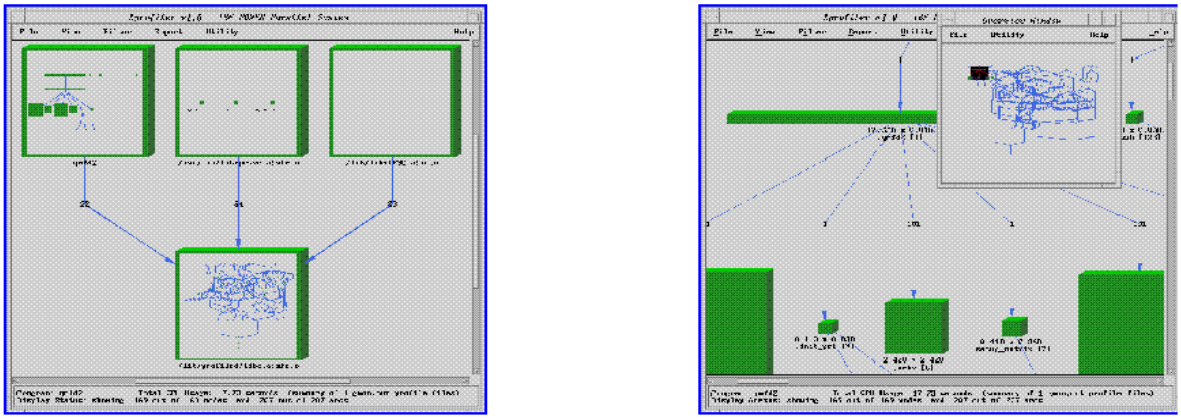


Figure 3 : Visualisation avec Xprofiler

4. Outils de trace MPI

4.1 MPItrace

Mpitrace est également une bibliothèque qui permet de collectionner les évènements MPI et il est payant. On peut le télécharger sur le site du « European Center for Parallelism of Barcelona » (CEPBA) à l'adresse http://www.cepba.upc.edu/paraver/permanent.htm.
On le trouve aussi avec HPM. On charge la librairie au moment de la compilation. Il n'y a donc pas besoin d'implémenter le code.
Il y a également une documentation dans le répertoire /home/coquart/exemples/doc.

4.2 Integrated Performance Monitoring (IPM)

C'est un produit du **NERSC** (National Energy Research Scientific Computing Center).
Il permet d'avoir un résumé des communications MPI. Il est en Open Source et on peut le télécharger à l'adresse http://www.nersc.gov/projects/ipm/.
Il est intéressant dans les résultats qu'il donne sur le pourcentage passé dans les différentes communications comme le montre la figure (4) ci-dessous, mais il faut implémenter le code de calcul. De plus, les résultats obtenus sur les évènements MPI et présentés de façon statistique ne sont pas facilement lisibles, figure (5) :

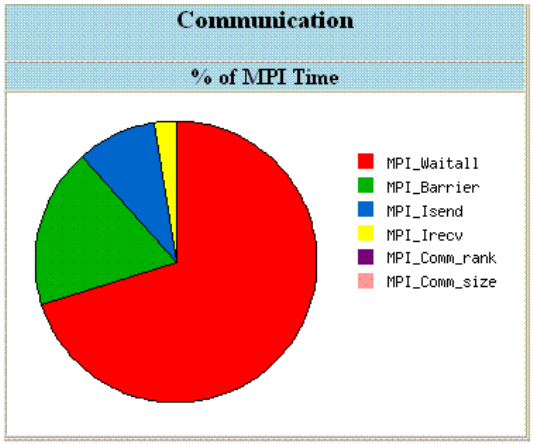
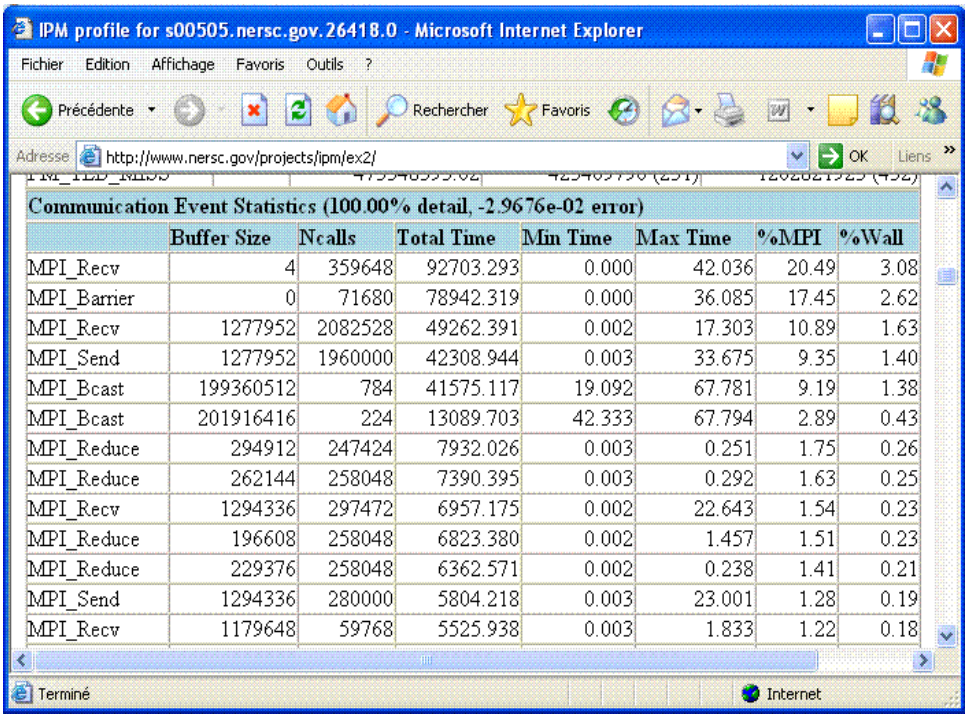


Figure 4 : Répartition des appels aux routines MPI



Communication Event Statistics (100.00% detail, -2.9676e-02 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Recv	4	359648	92703.293	0.000	42.036	20.49	3.08
MPI_Barrier	0	71680	78942.319	0.000	36.085	17.45	2.62
MPI_Recv	1277952	2082528	49262.391	0.002	17.303	10.89	1.63
MPI_Send	1277952	1960000	42308.944	0.003	33.675	9.35	1.40
MPI_Bcast	199360512	784	41575.117	19.092	67.781	9.19	1.38
MPI_Bcast	201916416	224	13089.703	42.333	67.794	2.89	0.43
MPI_Reduce	294912	247424	7932.026	0.003	0.251	1.75	0.26
MPI_Reduce	262144	258048	7390.395	0.003	0.292	1.63	0.25
MPI_Recv	1294336	297472	6957.175	0.002	22.643	1.54	0.23
MPI_Reduce	196608	258048	6823.380	0.002	1.457	1.51	0.23
MPI_Reduce	229376	258048	6362.571	0.002	0.238	1.41	0.21
MPI_Send	1294336	280000	5804.218	0.003	23.001	1.28	0.19
MPI_Recv	1179648	59768	5525.938	0.003	1.833	1.22	0.18

Figure 5 : Visualisation IPM

4.3 Perfometer

Ce logiciel est construit sur PAPI (Performance Application Programming Interface) et il faut implémenter le code pour l'utiliser.

5. Outils de visualisation des traces MPI

5.1 Jumpshot

Jumpshot est basé sur la bibliothèque MPICH (qui correspond à la bibliothèque MPI standard). C'est un produit fournit gratuitement par le Laboratoire National d'Argone. Le chargement du logiciel Jumpshot se fait à l'adresse suivante : http://www-unix.mcs.anl.gov/perfvis/.

PCT utilisé avec *Select trace* permet d'obtenir des évènements MPI, en général dans un seul fichier (trace.0), et non par processus comme pour PVT. Ce fichier doit être ensuite transformé en fichier à la norme UTE puis au format SLOG2 pour être lisible par Jumpshot.

uteconvert –n1 –o ute_trace trace.0

uteTOslog2 -n1 –o ute_trace.slog2 ute_trace
ou
ute2slog2 ute_trace

puis
jumpshot ute_trace.slog2 &

Au lancement, le logiciel présente trois fenêtres : une avec le fichier qui a été téléchargé, une avec la légende des couleurs caractérisant le type d'échanges effectués, et la dernière avec les évènements MPI collectés. Une documentation se trouve dans le répertoire /home/coquart/exemples/doc. Les exemples détaillés ci-dessous se trouvent également dans /home/coquart/exemples.

Au premier abord, lorsqu'on ne connait pas un programme, cet outil permet de visualiser quelles sont les communications MPI au sein de celui-ci.

5.1.1 Exemple de communication point à point

Voici ce que l'on visualise sous jumpshot pour un programme réalisé sur deux processeurs qui font des mpi_send() et mpi_recv() d'un tableau y(i,j,k) en parallélisant sur k. Le processeur 0 fait un mpi_recv() du tableau de l'autre processeur pour ensuite afficher le résultat y(i,j,k), tandis que le porcesseur 1 fait un mpi_send(). La figure (6) correspond à la légende des figures (7,8,9,10), avec les différents évènements MPI du programme.

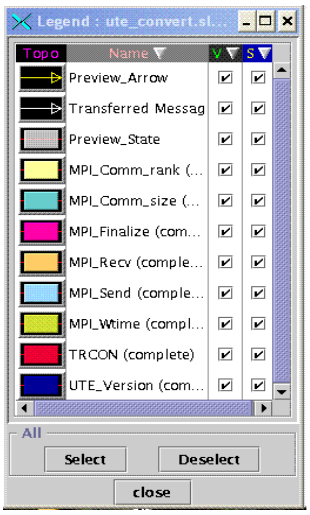


Figure 6 : Légende des Figures 7, 8, 9, 10

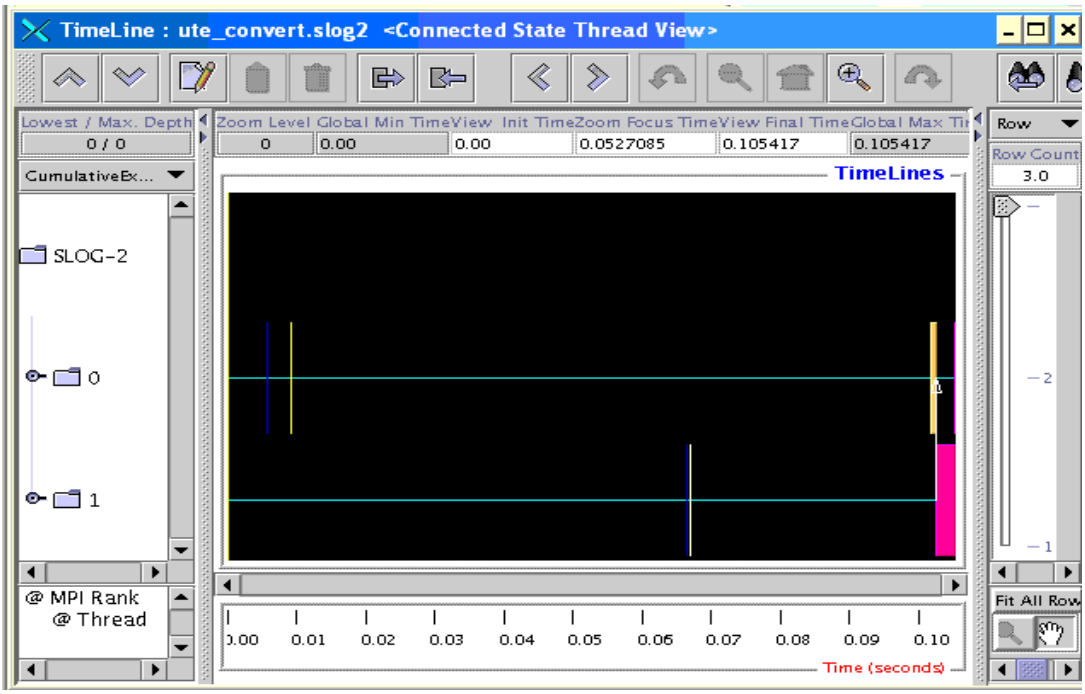


Figure 7 : Visualisation sous Jumpshot avec deux processeurs

Concernant la légende, figure (6), on peut changer la couleur de la représentation d'un évènement MPI en cliquant sur le bouton gauche de la souris. Les nouvelles couleurs choisies ne seront prises en compte que si l'on effectue une opération dans la fenêtre qui présente les résultats. Avec le bouton droit de la souris on peut sélectionner l'ordre alphabétique (par défaut) ou non. V et S qui apparaissent correspondent respectivement à « Visibility » et à « Searchability ». On peut ainsi faire disparaître certains évènements MPI, alléger le graphique et ne conserver que quelques évènements dans la fenêtre de résultats, en jouant sur la Visibilité. Par contre pour S je ne vois pas à quoi ça sert car les évènement ne disparaissent que partiellement.

Sur la figure (7), on retrouve les différents évènements MPI du code. Le processus 0 commence par écrire dans le fichier *trace*, puis il appelle la routine mpi_wtime(). Il appelle ensuite la routine mpi_recv() et se met en attente. On peut observer la flèche qui correspond aux échanges des données du processeur 1 vers le processeur 0, puis l'appel à la routine mpi_send() du processeur 1.

En zoomant, figure (8), on observe qu'il y a bien un nouvel appel de mpi_wtime par le processeur 0 puis un mpi_finalize() (en rose) par les deux processeurs successivement :

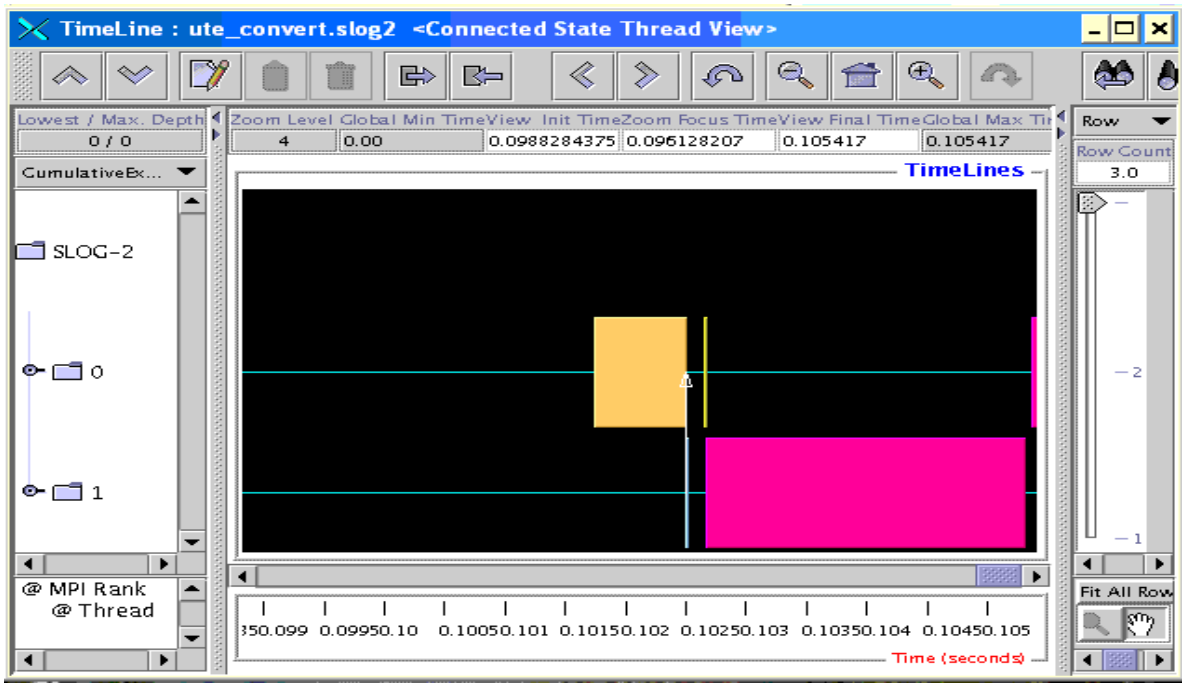


Figure 8 : Visualisation sous Jumpshot en zoomant

S'il y a quatre processeurs, la structure du programme reste la même et on obtient, en zoomant sur la fin du diagramme :

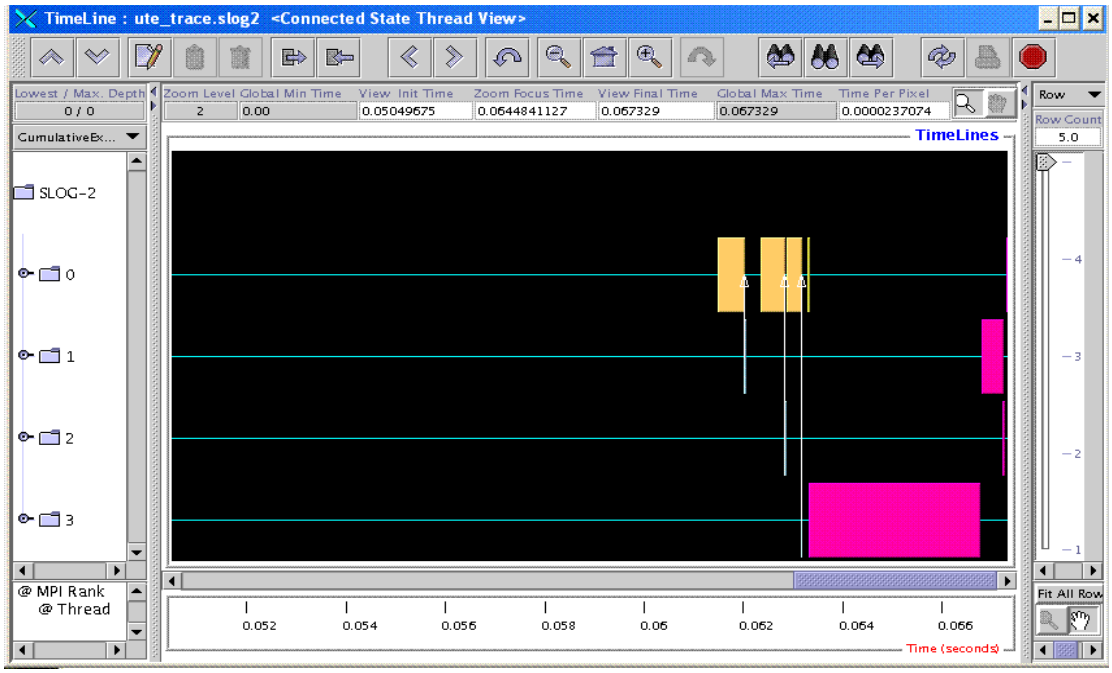


Figure 9 : Communications point à point entre quatre processeurs

En cliquant sur le bouton gauche de la souris en mode loupe, on peut zoomer sur les zones qui nous intéressent. On obtient des informations supplémentaires sur les évènements en utilisant la souris. En cliquant sur le bouton de droite de la souris, une fenêtre « Drawable Info » si on est sur une routine MPI, ou « Time Info Box » si on n'est pas sur une routine bien définie.

On peut également obtenir des histogrammes en cliquant sur STATISTICS dans une fenêtre qui s'appelle « Duration Info Box ». Cette fenêtre est obtenue en cliquant sur le bouton de droite (une ligne verticale apparaît) de la souris puis en faisant glisser la nouvelle ligne jusqu'au point où l'on souhaite s'arrêter, comme le montre la figure (10).

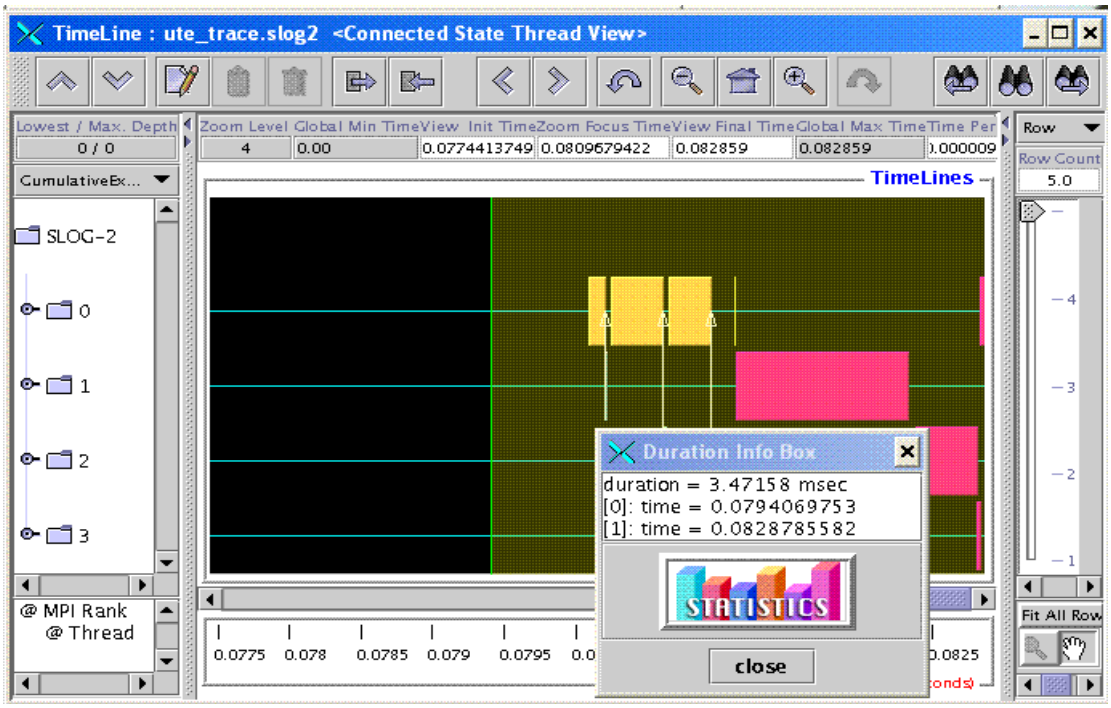


Figure 10 : Visualisation de la fenêtre Duration Info Box

Une fois que l'on a l'histogramme, on peut visualiser des « Summary Info Box » (si « states only » ou « all » est sélectionner en haut à gauche) en cliquant sur le bouton droit de la souris.

De même, en haut à droite de la fenêtre, il y a deux choix possibles pour la souris : en mode loupe ou en mode main. En mode main, en cliquant sur le bouton de gauche de la souris, on peut simplement déplacer en bloc les résultats d'un côté ou d'un autre.

5.1.2 Exemple de communication globale

Il s'agit ici de réaliser une somme avec quatre processeurs et de récupérer la somme par le processeur 0 avec un mpi_allreduce().

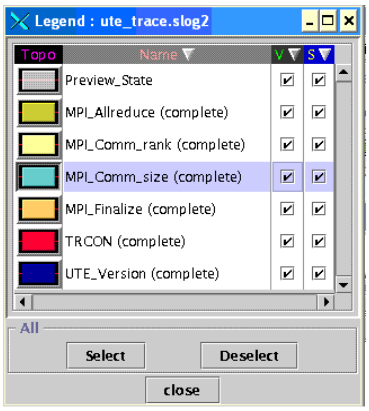


Figure 11 : Légende de la Figure 12

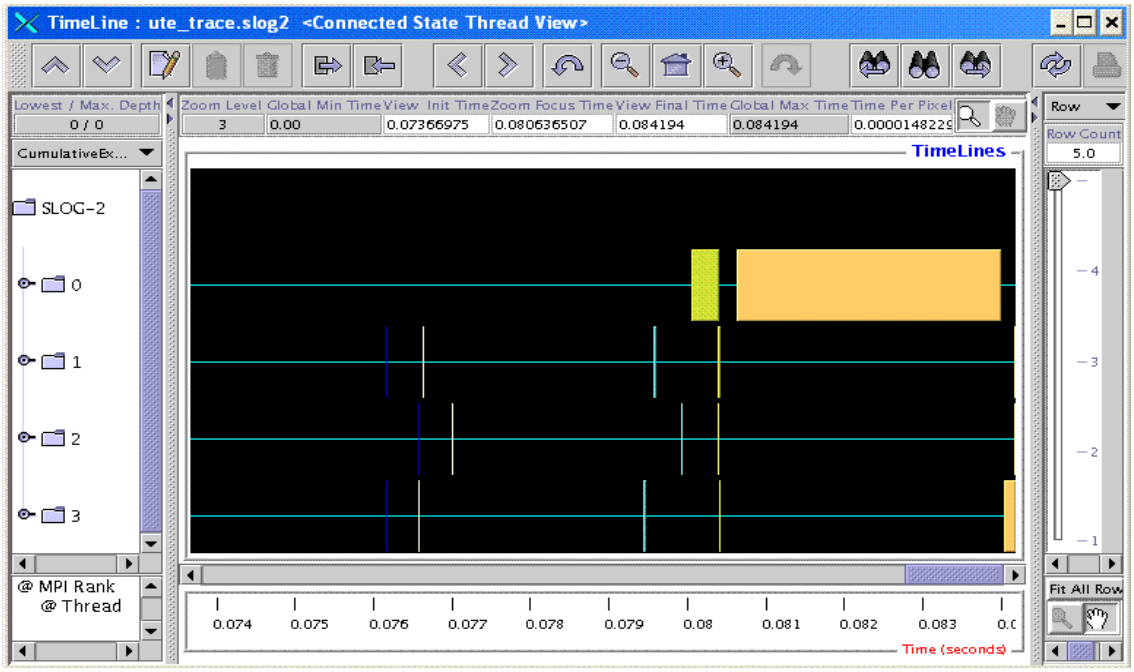


Figure 12 : Communications globales entre quatre processeurs

On observe le mpi_allreduce() puis le mpi_finalize() sur la figure (12). Par contre, contrairement aux communications point à point, il n'a pas de flèches pour symboliser les échanges de messages.

5.1.3 Programme complexe

Les programmes ci-dessus sont très simples et les « variables » observées correspondent aux vraies opérations réalisées directement dans le programme. Lorsque le programme est plus compliqué, le logiciel ne travaille plus directement sur des « variables réelles » mais sur des états représentatifs appelés : « preview event », « preview state » et « preview arrow ». Un « event » est un point sur dans la fenêtre Jumpshot. Un « state » est un évènement dont le temps au départ et le temps au final est le même. Un « arrow » est une flèche dont le temps au départ peut être différent du temps à l'arrivée. On a vu dans les exemples précédents que les échanges de messages étaient quasi instantanés.

Grâce aux états représentatifs (« preview event », « preview state » et « preview arrow »), le logiciel donne une vision du code dans son ensemble. Par exemple, un « preview arrow » représente une collection de vrais échanges de messages dont la quantité est proportionnelle à l'épaisseur du trait du « preview arrow ». D'autre part, ces échanges de messages ont pu se produire entre le début et la fin de la flèche « preview arrow » (voir figures 8 et 9). Ensuite, le temps passé dans les différentes routines d'échanges de messages est moyenné et représentés par les « preview state ». La superficie est alors proportionnelle au temps passé dans ces routines, puis ces états fictifs sont par exemple imbriqués les uns dans les autres.
Plus on zoom sur une région et plus on se rapproche des états réels du code.

La figure (14) ci-dessous présente les résultats obtenus sur 16 processeurs sur le code complexe dont on a déjà présenté les résultats sous PVT figure (1) en 1.1. Seules les routines cpmd.f, fftutil.f et fftmain.f ont été implémentées, cpmd.f étant le programme principal.

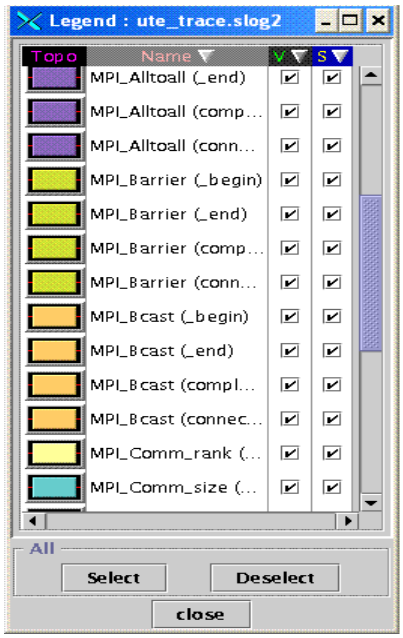


Figure 13 : Légende de la figure 14 et 15

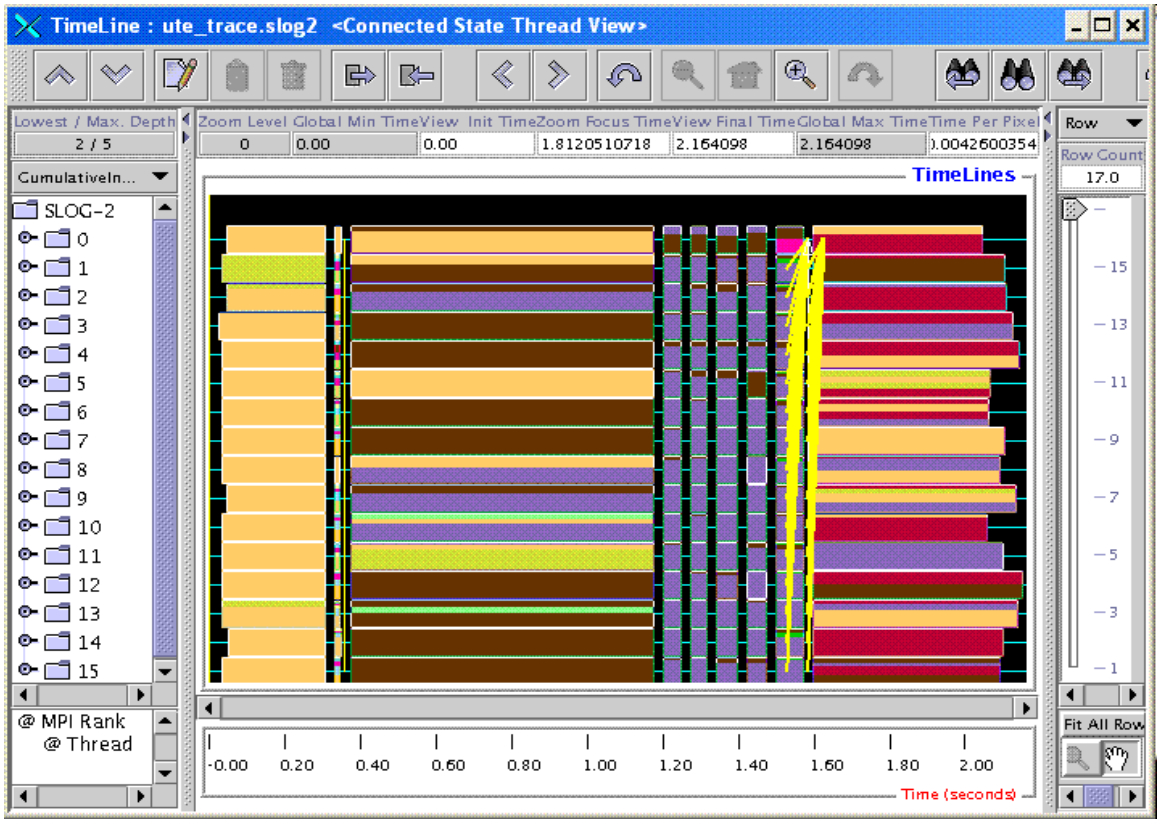


Figure 14 : Visualisation sous Jumpshot

La figure (14) donne une image globale de ce qui se passe au niveau de ces routines. Les « prevent arrow » en jaune symbolisent une grande quantité de messages échangés et qui vont des processeurs (15,…,1) au processeur 0, comme l'indique la flèche blanche. La flèche blanche correspond à un vrai échange de messages contrairement aux traits jaunes qui n'en sont qu'une représentation. Le choix de représentation des « preview state » correspond en haut à gauche à « Cumulative Inclusion Ratio ». Dans ce mode de représentation où les communications s'emboîtent, le temps passé dans un type de communication est proportionnel à la superficie du « preview state » représenté. On a toujours en haut les temps min et max d'exécution du code, ainsi que ceux entre lesquels on se situe en zoomant. En cliquant sur le bouton droit de la souris, on retrouve la « Duration Info Box » avec les informations temporelles sur les différents types d'évènements MPI pour le processeur choisi, ou la « Time Info Box ».

Si on zoom assez fort, on obtient la figure (15) :

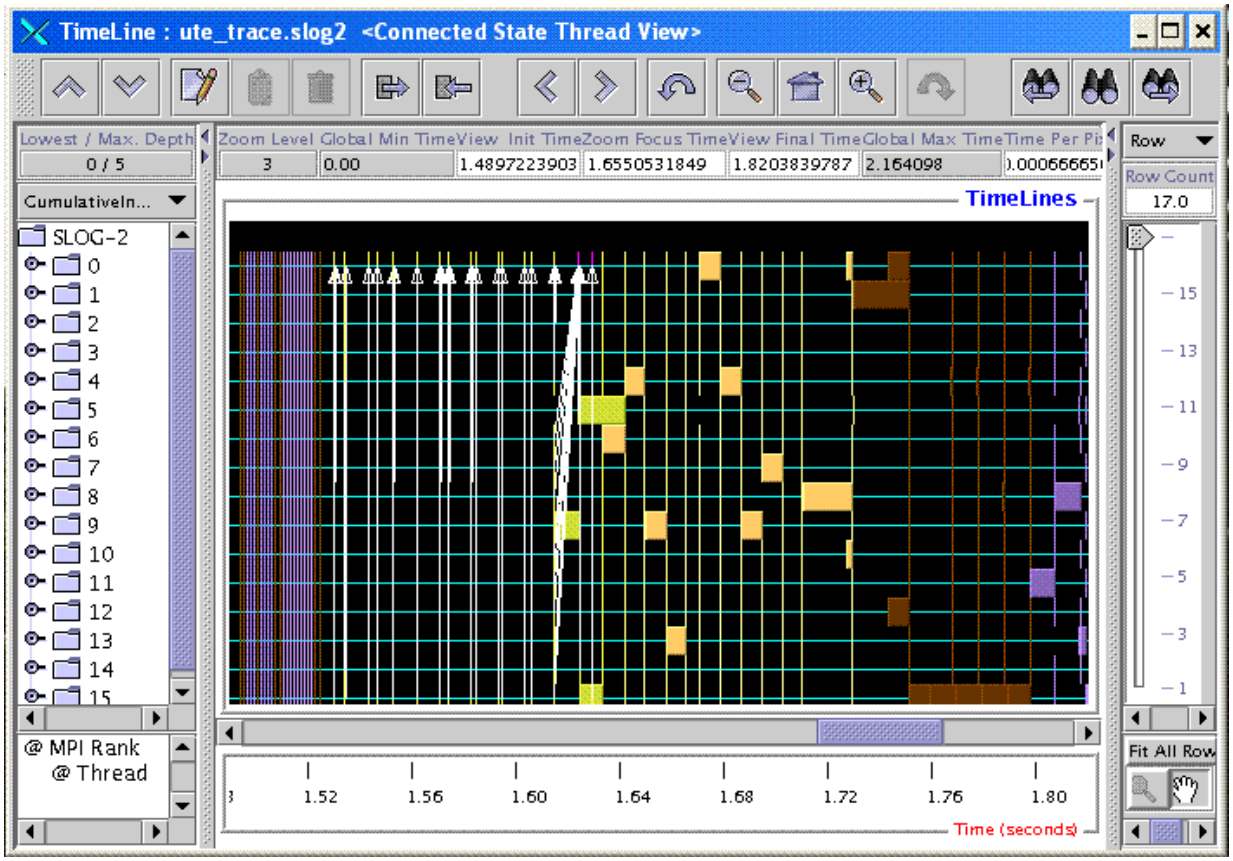


Figure 15 : Visualisation sous Jumpshot

On observe alors les vrais évènements MPI qui apparaissent. Les évènements MPI qui apparaissent dans la légende correspondent à ceux des routines qui ont été implémentées dans le script *pctpourtrace_script.cmd*. C'est ainsi que dans l'exemple présenté ici le mpi_init() n'apparaîtra pas car il n'est pas directement appelé par le programme principal qui a été implémenté. Au contraire, le mpi_finalize() sera sur le graphique car la routine où il se trouve est directement appelée par le programme principal.

Pour une analyse encore plus fine, il faut implémenter chacune des trois routines mais cela coûte en temps de calcul. C'est ce qui a été fait pour le programme principal car le code est connu et on a pu vérifier à priori que les deux routines fftmain.f et fftutil.f ne faisaient que du calcul. On s'aperçoit qu'on obtient les mêmes résultats que ceux obtenus sur la figure (14). Le code considéré est donc très bien écrit puisque le temps de communication est concentré dans la routine où on passe le plus de temps et non disséminé dans le programme, et les transformées de Fourier qui sont au cœur de ce programme ne font que calculer.

5.2 Vampirtrace/Vampir

Vampirtrace est une librairie qui permet de collecter les traces MPI. Ces traces sont ensuite visualisables avec le logiciel Vampir qui lui est payant et distribué par Intel à l'adresse http://www.hlrs.de/organization/par/services/tools/performance/vampir.html.
Il y a une documentation sur Vampirtrace dans le répertoire /home/coquart/exemples/doc.

Vampir donne le même type de sortie que Jumpshot.

5.3 Paraver

Paraver a été mis au point au « European Center for Parallelism of Barcelona » (CEPBA) (http://www.cepba.upc.edu/paraver/) et il est payant.

Ce logiciel donne le même type de sortie que Jumpshot.

5.4 Paragraph

Il faut implémenter le code pour l'utiliser.

6. Conclusion

Il ressort de cette étude que l'outil **gratuit** le plus pertinent et le **plus simple** à utiliser pour étudier les performances d'un code parallèle correspond au PE Benchmarker (avec PCT, PVT) couplé au logiciel de visualisation Jumpshot.

[1] High Performance Tools and technologies, Michael Collette, Bob Corey and John Johnson December 2004, Lawrence Livermore National Laboratory, U. S. Department of Energy

[2] Jumpshot-4 Users Guide, Anthony Chan, David Ashton, Rusty Lusk, William Gropp, May 2005, Mathematics and Computer Science Division, Argone National Laboratory