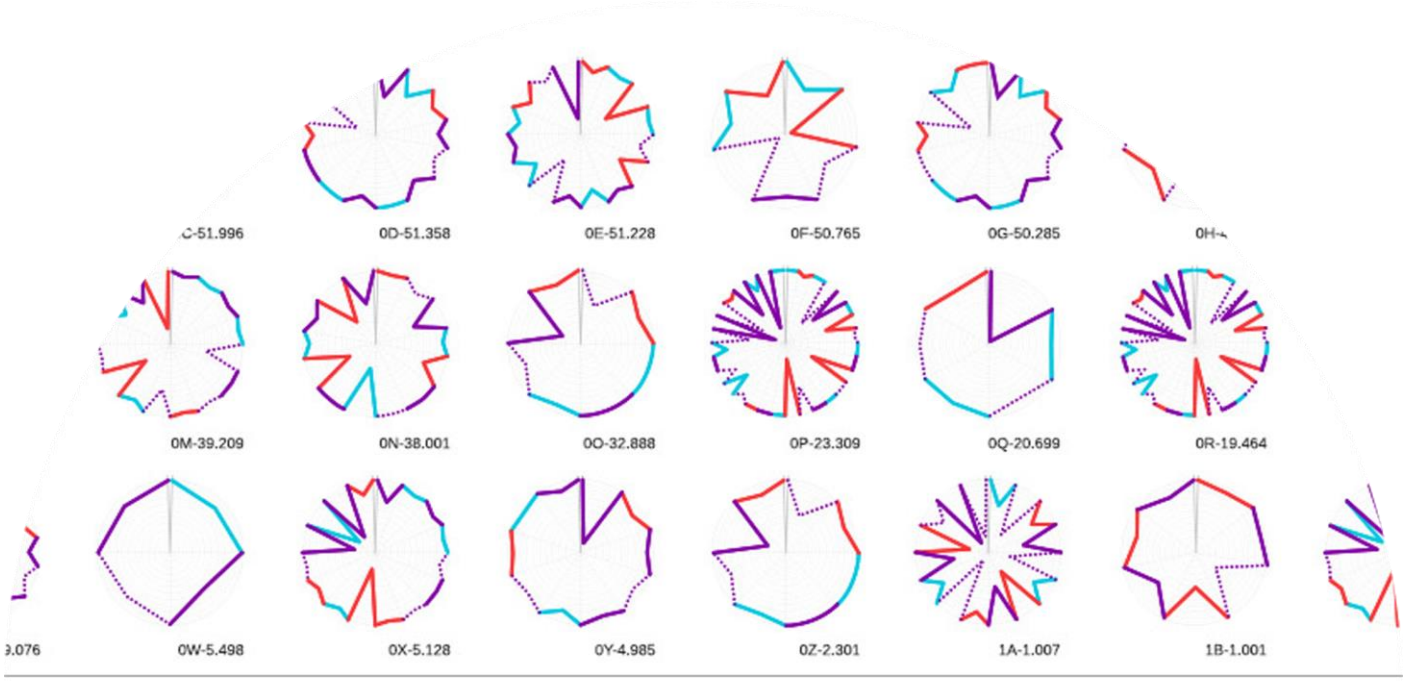
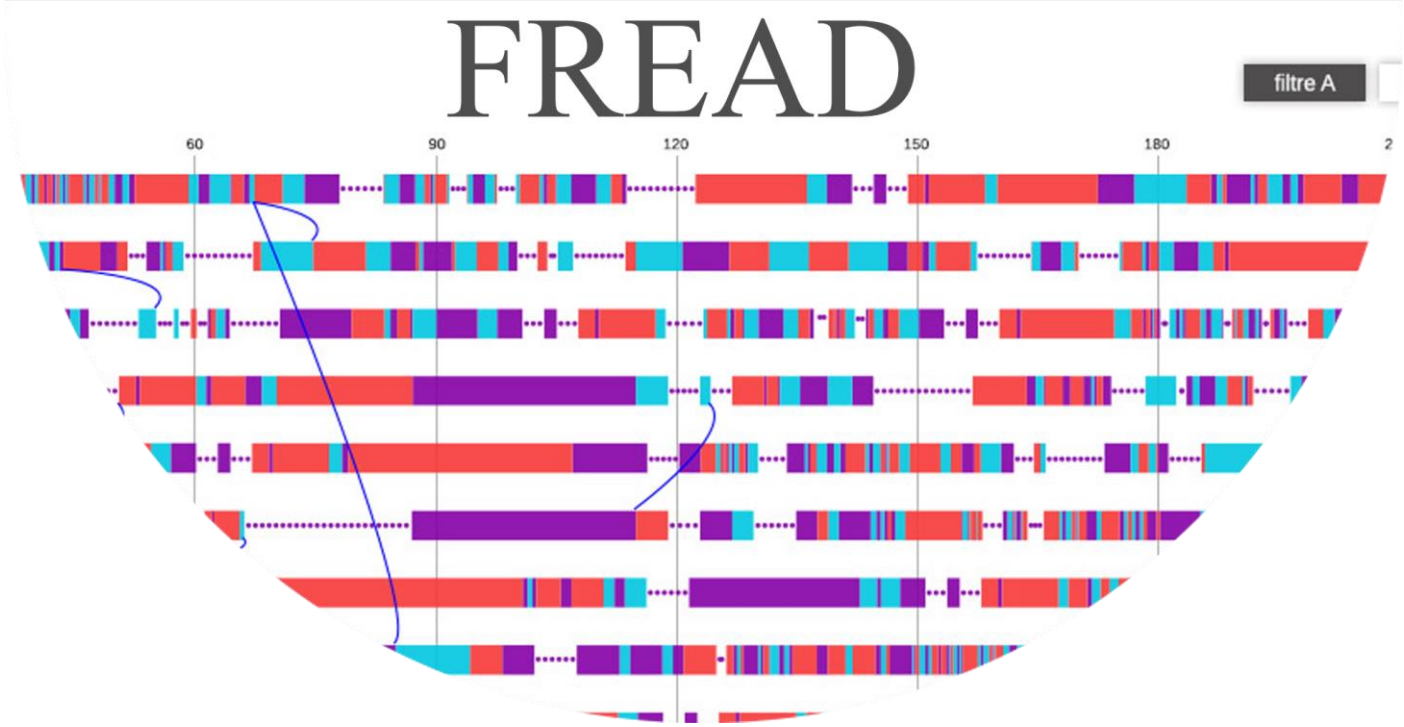

0D-51.358
0E-51.228
0F-50.765
0G-50.285
0M-39.209
0N-38.001
0O-32.888
0P-23.309
0Q-20.699
0R-19.464
0W-5.498
0X-5.128
0Y-4.985
0Z-2.301
1A-1.007
1B-1.001

FREAD


filtre A
60
90
120
150
180

Fast Reading Datas

FREAD est un **logiciel de visualisation de traces d'exécution** de programmes multiprocessus (ou en anglais, multithreaded). Comme d'autres logiciels de visualisation de traces d'exécution, il permet, à partir de données brutes relatant l'exécution d'un programme, de rendre des diagrammes **lisibles par le développeur,** dans le but d'analyser les performances du programme en vue de l'améliorer.

Actuellement, le logiciel VITE existe sur le marché et permet de rendre des traces d'exécution de manière similaire. Mais ce logiciel est peu ergonomique et ne permet pas la lecture de traces dépassants plusieurs Giga Octets sans risquer de voir l'ordinateur ralentir. FREAD veut palier ces deux manques. Il se veut **plus ergonomique et puissant** dans son utilisation.

De plus, FREAD ne se contentera pas seulement d'afficher les traces d'exécutions, difficilement lisibles (elles peuvent s'étaler sur plusieurs centaines de milliers de lignes). Il permettra d'afficher des **traces préalablement traitées** par un algorithme développé par **François Trahay** qui permet de détecter des motifs reconnaissables dans une trace. Nous nous basons sur cette **reconnaissance de motifs** pour optimiser les performances de rendu et l'ergonomie.

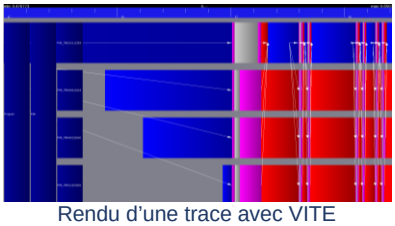
Le logiciel est distribué de manière libre, sous la **licence BSD**. Il est fait de manière à pouvoir être utilisé et modifié facilement par le plus grand nombre de personnes, chercheurs, entreprises ou particuliers. Dans cette optique de rendre FREAD souple et adaptable à différents formats de traces, le parseur, prenant en charge le format PAJE dans un premier temps, a été pensé pour être **facilement remplacé** par un parseur lisant un autre format si besoin en est. Pour cela, un parseur adapté doit être codé et ajouté à FREAD, l'interfaçage se faisant via des fonctions prédéfinis qui seront appelées par FREAD lors de la lecture.

Les motifs sont l'idée phare du projet. En effet, leur détection permet une factorisation code qui réduit grandement la taille des fichiers lus. Cette factorisation nous permet d'afficher des données sur ces motifs, et d'utiliser le système de culling. **Le culling** est bien connu des professionnels du jeu vidéo. Il permet d'optimiser la vitesse de rendu en passant par une étape de rendu plus grossier qui donne une idée générale du rendu final.

L'idée étant pour le développeur, travaillant sur des logiciels multiprocessus de pouvoir, à travers la trace**, repérer l'exécution de certaines des fonctions clés du programme**, afin de voir si celles-ci travaillent de **manière optimale** dans la plupart des cas. Ainsi, avec FREAD, le développeur aura dans sa boite à outils un outil ergonomique, pratique, facile et rapide d'utilisation.

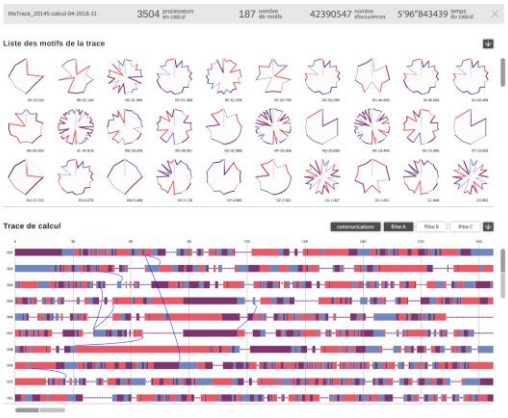
Premières Recherche

Le projet a été initialement proposé par François Trahay pour répondre à une problématique. Comment rendre de manière lisible et rapide des traces d'exécutions pouvant faire jusqu'à plusieurs giga octets ? Il se base sur des projets similaires ayant déjà été réalisé, VITE et EZTrace.



Rendu d'une trace avec VITE

Mais l'interface des logiciels existant n'était pas très ergonomique le problème d'un rendu plus clair, lisible et épuré se posait. Céline Mast une étudiante en design a proposé une idée d'interface de rendu pour ce nouveau logiciel. Elle se veut beaucoup plus clair et agréable tout en incluant un nouveau système de visualisation que sont les motifs d'exécution.



On peut reconnaitre sur la partie basse de similitudes avec VITE. Dans un premier temps notre recherche a été sur la compréhension de l'interface qui nous a été proposée afin d'adapter au mieux notre code. Pour cela nous nous sommes tourné, avec les conseils de François Trahay, vers la librairie SFML qui permet de faire des rendus rapides de motifs à l'écran.

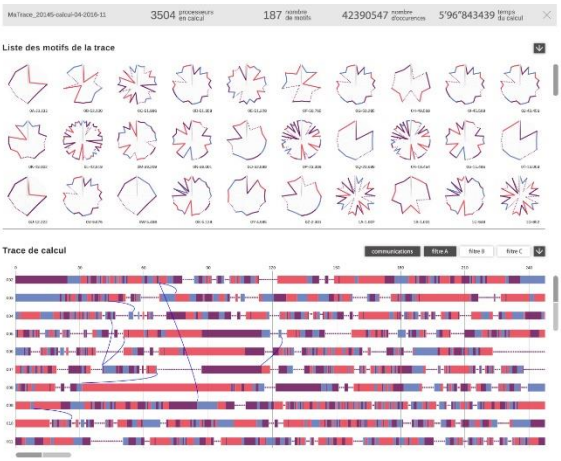
Pour optimiser le rendu il nous a fallu réfléchir à une architecture qui nous permettrai d'avoir à la fois un affichage rapide, même si toutes les informations à lire n'ont pas encore été traité, et un code modulable, pour permettre à des futurs utilisateurs de modifier le parseur. Nous nous sommes donc penché sur une architecture en trois temps, un processus de rendu qui qui gère l'affichage et les interactions avec l'utilisateur, un processus code qui gère la mémoire et les données nécessaires au rendu et un processus parseur qui s'occupera de lire la trace.

L'autre problématique majeur était d'avoir un rendu rapide et clair des données. Pour cela nous sommes partis sur le système de culling. Le principe est de n'afficher que les informations pertinentes. C'est-à-dire qu'elles doivent avoir une taille minimale à l'écran pour être affichées, mais aussi que le motif affiché doit être assez important à l'écran pour que l'on affiche le motif précis. Sinon on affiche un motif moyen représentant les durées moyennes qui ont été préalablement calculé.

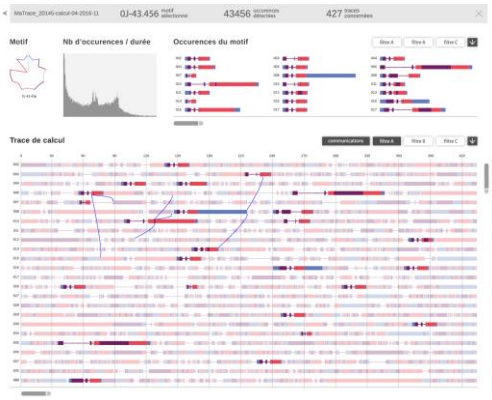
Les problématiques, objectifs et design du projet étant déjà bien définis à ses débuts, nous sommes restés sur cette lancé en créant une architecture capable de répondre au mieux à toutes ces exigences.

Scénario d'utilisation

L'utilisateur arrive sur notre outil et se voit proposer l'importation d'une trace, une fois celle-ci trouvée et chargée l'interface principale se dévoile.



Elle est décomposée en 2 grandes parties celle du dessus comportant tous les motifs trouvés dans la trace, puis celle du dessous comportant la trace totale sous la forme de longue ligne colorée représentant chaque thread et les différents événements les comportant.

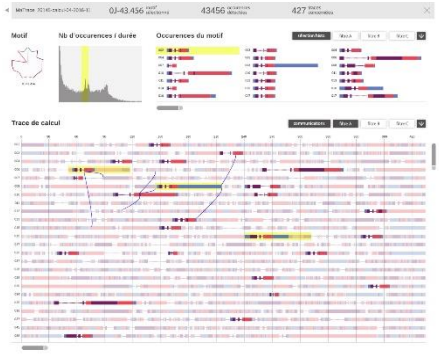


S'il clique sur un motif, il voit alors toutes les occurrences de ce motif à sa droite ainsi qu'un histogramme de répartition, de plus ils sont repérables en sur opacité sur la zone du bas qui se sera agrandi, pour détailler au mieux le profil.

Il pourra choisir plusieurs modes de tri pour les occurrences du pattern, par taille, processus, date, etc …

En cliquant sur une occurrence, celle-ci et son occurrence dans la trace apparaissent en sur brillance. Les liens affichés à l'écran ne concernent que l'occurrence et le reste de la trace n'ayant aucun lien avec ce pattern sont toujours opacifiés.

L'utilisateur aussi accès à un diagramme de répartition du nombre de motifs par intervalle de temps d'exécutions. Ce diagramme se base sur une échelle logarithmique afin de s'adapter le plus possible à l'affichage recherché.



Il peut sélectionner une zone sur ce diagramme afin de mettre en surbrillance tous les motifs concernés. Ainsi grâce à ces schémas faciles et rapides à lire et à la connaissance que le développeur a de son code il peut lier les motifs avec certains morceaux de son code afin d'en étudier les performances.

A quoi s'attendre ?

FREAD est un outil qui permettra de visualiser des traces. Sans surprise il ressemblera au design prévu à l'origine. On peut voir ici la reconnaissance des éléments de la trace par le parseur.

```
5 "ST_Thread" "MPI_Barrier" "1.0 0.0 1.0" "STV_MPI_Barrier"
5 "ST_Thread" "MPI_BCast" "1.0 0.0 1.0" "STV_MPI_BCast"
5 "ST_Thread" "MPI_Gather" "1.0 0.0 1.0" "STV_MPI_Gather"
5 "ST_Thread" "MPI_Scatter" "1.0 0.0 1.0" "STV_MPI_Scatter"
5 "ST_Thread" "MPI_Allgather" "1.0 0.0 1.0" "STV_MPI_Allgather"
5 "ST_Thread" "MPI_Alltoall" "1.0 0.0 1.0" "STV_MPI_Alltoall"
5 "ST_Thread" "MPI_Reduce" "1.0 0.0 1.0" "STV_MPI_Reduce"
5 "ST_Thread" "MPI_Allreduce" "1.0 0.0 1.0" "STV_MPI_Allreduce"
5 "ST_Thread" "MPI_Reduce_scatter" "1.0 0.0 1.0" "STV_MPI_reduce_scatter"
5 "ST_Thread" "MPI_Scan" "1.0 0.0 1.0" "STV_MPI_Scan"
5 "ST_Thread" "MPI_Ibarrier" "1.0 0.0 1.0" "STV_MPI_Ibarrier"
5 "ST_Thread" "MPI_Ibcast" "1.0 0.0 1.0" "STV_MPI_Ibcast"
5 "ST_Thread" "MPI_Igather" "1.0 0.0 1.0" "STV_MPI_Igather"
5 "ST_Thread" "MPI_Iscatter" "1.0 0.0 1.0" "STV_MPI_Iscatter"
5 "ST_Thread" "MPI_Iallgather" "1.0 0.0 1.0" "STV_MPI_Iallgather"
5 "ST_Thread" "MPI_Ialltoall" "1.0 0.0 1.0" "STV_MPI_Ialltoall"
5 "ST_Thread" "MPI_Ireduce" "1.0 0.0 1.0" "STV_MPI_Ireduce"
5 "ST_Thread" "MPI_Iallreduce" "1.0 0.0 1.0" "STV_MPI_Iallreduce"
5 "ST_Thread" "MPI_Ireduce_scatter" "1.0 0.0 1.0" "STV_MPI_Ireduce_scatter"
5 "ST_Thread" "MPI_Iscan" "1.0 0.0 1.0" "STV_MPI_Iscan"
5 "ST_Thread" "MPI_Overlap" "1.0 1.0 0.0" "STV_MPI_Overlap"
5 "ST_Thread" "MPI_Probe" "0.6 0.1 0.9" "STV_MPI_Probe"
4 "MPI collective communication" "CT_Program" "CT_Thread" "CT_Thread" "L_MPI_C
4 "MPI point to point communication" "CT_Program" "CT_Thread" "CT_Thread" "L_M
4 "MPI SPAWN" "CT_Program" "CT_Thread" "CT_Thread" "L_MPI_SPAWN"
2 "MPI Send" "CT_Thread" "E_MPI_CommSend"
2 "MPI_Recv" "CT_Thread" "E_MPI_CommRecv"
```

Trace Brut

```
1 name : Process - type : CT_Program - alias : CT_Process
1 name : Thread - type : CT_Process - alias : CT_Thread
2 name : Program state - type : CT_Program - alias : ST_Program
2 name : Process state - type : CT_Process - alias : ST_Process
2 name : Thread state - type : CT_Thread - alias : ST_Thread
2 name : User state - type : CT_Thread - alias : ST_User
32 name : EZTrace Flush - type : ST_Thread - color : 255 255 255 - alias : STV_FLUSH
32 name : Blocked - type : ST_Thread - color : 255 0 0 - alias : STV_Blocked
32 name : Working - type : ST_Thread - color : 0 0 255 - alias : STV_Working
32 name : Critical Section - type : ST_Thread - color : 0 255 0 - alias : STV_Critica
32 name : User Event - type : ST_User - color : 0 255 0 - alias : STV_User_Event_Gree
32 name : User Event - type : ST_User - color : 255 0 0 - alias : STV_User_Event_Red
32 name : User Event - type : ST_User - color : 25 127 204 - alias : STV_User_Event_B
32 name : User Event - type : ST_User - color : 255 255 0 - alias : STV_User_Event_Ye
32 name : User Event - type : ST_User - color : 255 0 255 - alias : STV_User_Event_Pi
32 name : EZTrace synchronization - type : ST_Thread - color : 255 255 255 - alias :
4 name : User Event - type : CT_Thread - alias : E_UserEvent
4 name : SIGNAL Received - type : CT_Thread - alias : E_SigSegv
8 name : Current CPU - type : CT_Thread - color : 0 0 0 - alias : V_CURCPU
32 name : stdio_read - type : ST_Thread - color : 0 0 0 - alias : stdio_read
32 name : stdio_pread - type : ST_Thread - color : 0 0 0 - alias : stdio_pread
32 name : stdio_readv - type : ST_Thread - color : 0 0 0 - alias : stdio_readv
32 name : stdio_fread - type : ST_Thread - color : 0 0 0 - alias : stdio_fread
32 name : stdio_write - type : ST_Thread - color : 0 0 0 - alias : stdio_write
32 name : stdio_pwrite - type : ST_Thread - color : 0 0 0 - alias : stdio_pwrite
32 name : stdio_writev - type : ST_Thread - color : 0 0 0 - alias : stdio_writev
32 name : stdio_fwrite - type : ST_Thread - color : 0 0 0 - alias : stdio_fwrite
32 name : stdio_select - type : ST_Thread - color : 0 0 0 - alias : stdio_select
```

Donnés du parseur

A ce stade on ne voit que des donnés bruts analysés par le parseur. Maintenant il faut s'imaginer que cette analyse faite par l'ordinateur va bientôt donner ceci :

Prévu pour être utilisé en parallèle avec son Interface de développement préférée. Ergonomique et lisible FREAD égaillera les écrans mornes des développeurs !