



# MÉMOIRE DE STAGE MASTER 2

Stage du 1<sup>er</sup> février 2023 au 31 juillet 2023

Intitulé : Interruptions en espace utilisateur pour le réseau  
BXI

Charles GOEDEFROIT

Encadré par Alexandre DENIS (Inria), Grégoire PICHON (Atos) et  
Mathieu BARBE (Atos)

19 juillet 2023

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Presentation Inria . . . . .	3
1.2	Presentation Atos (Eviden) . . . . .	3
1.3	Environnement de travail . . . . .	3
1.4	Le cadre . . . . .	3
1.5	Presentation du plan . . . . .	4
<b>2</b>	<b>Context</b>	<b>4</b>
2.1	HPC . . . . .	4
2.2	OS bypass . . . . .	5
2.3	Polling . . . . .	6
2.3.1	Inconvénients du polling . . . . .	6
2.4	BXI . . . . .	6
2.5	MPI . . . . .	7
2.5.1	Les communications point à point . . . . .	7
2.5.2	Les communications collective . . . . .	7
2.5.3	La progression asynchrone . . . . .	8
2.6	NewMadeleine . . . . .	8
2.7	Travaux antérieur . . . . .	8
<b>3</b>	<b>Problématiques / Objectifs</b>	<b>9</b>
3.1	Sujet . . . . .	9
3.1.1	Nouveau mécanisme d'interruption en espace utilisateur	9
3.1.2	contentions... . . . .	9
3.2	Ne plus faire de Polling . . . . .	9
3.3	Objectifs . . . . .	9
3.3.1	Objectifs global . . . . .	9
3.3.2	Objectif du stage . . . . .	9
<b>4</b>	<b>Exploration de uintr</b>	<b>10</b>
4.1	Prérequis est accès . . . . .	10
4.2	Fonctionnement des uintr . . . . .	10
4.2.1	Détail . . . . .	10
4.2.2	Capacité présente et futur . . . . .	10
4.2.3	Exemple . . . . .	10
4.2.4	Partage du FD . . . . .	10
4.2.5	+ . . . . .	11
4.3	Tests du mécanisme . . . . .	11
4.4	Correction du patch pour l'appel système uintr_alt_stack . .	11
4.5	Mesure de la latence (Feedback Mathieu) . . . . .	11
4.6	Performances . . . . .	11

<b>5</b>	<b>Intégration dans NewMadeleine</b>	<b>11</b>
5.1	Présenté NewMadeleine details . . . . .	11
5.2	Modification . . . . .	12
5.3	Suite de tests . . . . .	12
5.4	Performances . . . . .	12
5.4.1	Résultats avec attente active . . . . .	12
5.4.2	Résultats recouvrement communication par du calcule	13
<b>6</b>	<b>Bilan</b>	<b>13</b>
<b>7</b>	<b>Remerciements</b>	<b>13</b>
<b>8</b>	<b>Annexes</b>	<b>14</b>
<b>9</b>	<b>Bibliographie</b>	<b>14</b>

# **1 Introduction**

J'ai effectué mon stage de 6 mois dans l'équipe-projet TADaaM du Centre Inria de l'université de Bordeaux. Le sujet du stage à été proposé par Alexandre Alexandre Denis (Inria) et Grégoire Pichon (Atos). Ils ont aussi encadré le stage aussi que Mathieu BARBE (Atos).

## **1.1 Présentation Inria**

Inria est l'institut national français de recherche en sciences et technologies du numérique. Compte plus de 3 900 chercheurs et ingénieurs au sein de 215 équipes-projets. La plupart des centres sont communes avec une grande université.

## **1.2 Présentation Atos (Eviden)**

Atos est un des leaders internationaux de la transformation numérique. Atos couvre un champ large d'activité : le cloud, la cybersécurité, les services, le conseil, les supercalculateurs... Atos compte 112 000 collaborateurs. Atos est en restructuration pour se séparer en 2 entités. L'entité qui nous intéresse pour ce stage est Eviden qui englobe les activités autour des supercalculateurs et du HPC. Cette restructuration est récente donc quand je parle d'Atos dans ce document je parle de la partie Eviden.

## **1.3 Environnement de travail**

Mon environnement de travail c'est les locaux du Centre Inria de l'université de Bordeaux. J'ai un bureau dans l'open space de l'équipe-projet TADaaM. Je peux participer et assister à des activités scientifiques intéressantes (séminaire, soutenance de thèse, soutenance HDR, activité diverse...). J'ai accès à la salle de visio conférence, notamment pour les réunions de suivi de stage hebdomadaire. Il y a aussi un Baby-foot, une cafétéria, une petite Médiathèque...

## **1.4 Le cadre**

Du côté de l'équipe TADaaM de Inria l'objectif de l'équipe est de faire de la recherche. Elle est composée de chercheurs, ingénieurs de recherche, post-doc, doctorant et stagiaire. La culture informatique de l'équipe est l'utilisation des environnements linux, des logiciels open source, des clusters de calcul HPC, le traitement des données et le système. L'équipe a mis à ma disposition un ordinateur portable avec une station de travail branchée à un écran, à internet, à un clavier et une souris. Inria donne aussi accès à un ensemble de services, boîte email, service de communication Mattermost, service de visio Webex, intranet avec le menu de la cafétéria... J'ai été libre

d'installer mon système d'exploitation, d'utilisé les outils informatique que je voulais...

Du coté d'Atos l'équipe BXI-LL (BXI Low Level) est une équipe don l'objectif est de fournir le support logiciels bas niveau pour les carte réseaux BXI. Cela consiste à développé les drivers de la carte, le support Lustre<sup>1</sup> ainsi que l'interface logiciels Portal4. Cette interface permet aux implémentations MPI d'utiliser le réseaux BXI. Le personnel est composée d'ingénieurs, de doctorant et de stagiaire. /\* TODO : vérifié \*/ La culture informatique de l'équipe est l'utilisation des cluster HPC, la programmation système et l'utilisation d'un système de sécurité PKI pour accédé aux ordinateurs et aux services interne. Atos nous a donné accès à une machine avec deux processeur Intel Saphir Rapid. Pour accédé à cette machine ils nous on fourni un accès ssh et un compte VPN. J'ai pu avoir tous les droits d'accès sur cette machine pour changé le noyaux du système.

## 1.5 Présentation du plan

Dans ce mémoire je vais commencé par vous présenté le context dans le quelle le stage ce place. Je continuerai par les objectifs à long term et les objectifs du stage. Je vous présenterai le nouveau mécanisme d'interruption et ce que j'ai fait avec. Je présenterai l'intégration de ce mécanisme dans la bibliothèque de communication NewMadeleine et les tests que l'on à effectué. Pour finir je ferai un bilan du stage et je parlerai de la suite.

## 2 Context

### 2.1 HPC

Le stage c'est déroulé dans le context du Calcule Haute Performance (HPC, High-Performance Computing en anglais). Le HPC utilise des supercalculateur pour la simulation numérique et le pré-apprentissage d'intelligences artificielles. Ces simulation simule la dynamique des fluides, la résistance structurelle, les interaction moléculaire, les flux d'aire...

Elles couvrent différant domaines :

- L'industrie : le médicales, l'automobiles, l'aviations, la constructions, l'aérospatiales...
- La défense : simulation atomique
- La recherche scientifiques : la création des galaxie, la fusion nucléaire, le climat...
- La météo

---

1. Lustre est un système de fichier parallèle et distribué. Il est utiliser pour l'I/O est ne fait pas partie de ce stage.

De nos jours les supercalculateurs sont composés de plusieurs ordinateurs que l'on appelle nœuds de calculs. Ceci est regroupé en grappe (cluster), il est tout vu et est utilisé comme une seule grande machine. Ce fonctionnement pose des questions sur la répartition du calcul, la distribution de la mémoire, la communication entre les différents nœuds... Dans le contexte du stage, nous nous sommes concentrés sur les communications entre nœuds. Les nœuds sont connectés entre eux par un réseau haute performance. Ce réseau est dédié aux communications et n'est pas forcément de type Ethernet. La gestion des nœuds est généralement effectuée par un second réseau plus traditionnel (Ethernet ; TCP/IP). Les réseaux haute performance ont une latence autour de quelques microsecondes. Chaque nœud possède une ou plusieurs cartes réseau et il faut les programmer.

## 2.2 OS bypass

Le stage se passe donc aussi dans un contexte système. Chaque nœud a son propre système d'exploitation qui permet la gestion des ressources, des processus, des fichiers, des périphériques. Pour cela le système a 2 espaces :

- un espace noyau où seul le code du système peut s'exécuter. Le code du système peut donc être modifié n'importe où dans la mémoire, exécuté n'importe où...
- un espace utilisateur où le code de l'utilisateur est exécuté. Cette espace est limitée par le noyau qui contrôle ce que fait l'utilisateur.

En temps normal les périphériques sont programmés directement depuis le noyau du système d'exploitation ceci pour des questions de sécurité, de standardisation des accès... Lors que l'on passe par le noyau (kernel en anglais) on a un surcoût qui n'est pas négligeable dans notre cas. Pour passer par le noyau on utilise généralement des appels système qui se présentent sous la forme d'une fonction. Un appel système va effectuer un changement de contexte (context switch) pour passer de l'espace utilisateur à l'espace noyau. Ce changement de contexte est coûteux car il sauvegarde les états du code de l'utilisateur avant d'exécuter celui du noyau. Une fois le context switch effectué le code noyau de l'appel système s'exécute avant de refaire un context switch pour cette fois passer de l'espace noyau à l'espace utilisateur, et donc restaurer l'état du code de l'utilisateur. Donc le fait de passer par un appel système coûte plusieurs microsecondes. On voit donc qu'on ne peut pas utiliser d'appel système car un appel système est déjà du même ordre de temps qu'une communication. En HPC on programme donc directement la carte réseau à partir de l'espace utilisateur (OS bypass en anglais). Pour cela on initialise toujours la carte à partir du noyau mais on fait une projection de la mémoire et des registres de la carte réseau dans l'espace d'adressage virtuelle du processus utilisateur.

Pour transmettre des événements à l'utilisateur les périphériques utilisent généralement les interruptions, qui passent par le système donc on ne les utilise

pas en HPC. En HPC on fait donc du polling.

## 2.3 Polling

Le polling consiste à scruter (poll) régulièrement si on a reçu un événement. Concrètement cela consiste à lire une zone mémoire modifiée par la carte réseau et voir si un bit est passé à 1. Pour cela il est possible de dédier un thread qui va faire de l'attente active, scruter sans cesse si un événement a été reçu. Mais on perd une unité de calcul lorsque ce que le thread est ordonnancer donc de la puissance de calcul, donc cette technique est peu utilisée. Il est aussi possible d'entrelacer le code de l'utilisateur avec des scrutations, c'est fréquemment utilisé mais ça oblige l'utilisateur de prendre en compte la progression. Une autre solution qui est utilisée par Pioman dans NewMadelaine (j'en parlerai plus tard) consiste à faire ces scrutations de manière opportuniste dans les threads qui ont fini leur calcul, mais pour cela il faut déjà utiliser plusieurs threads et avoir une application qui a des calculs d'une durée hétérogène.

Il faut donc, peut importe la technique, régulièrement scruter pour faire progresser les communications.

### 2.3.1 Inconvénients du polling

Dans le cas d'un thread dédié qui fait de l'attente active la réactivité est excellente hormis quand il y a plus de threads que d'unités de calcul est que le thread n'est pas ordonnancer.

Quand on entrelace le calcul et les scrutations la réactivité est moins bonne car il faut attendre que le calcul soit fini pour faire un poll. C'est ce qui est fait habituellement.

Quand on utilise les threads de façon opportuniste pour faire des scrutations la réactivité est moins bonne car il faut qu'il y ait un thread disponible.

Donc un choix doit être fait entre perdre de la capacité de calcul ou perdre en réactivité. En plus on perd un peu de temps de calcul à effectuer du polling.

## 2.4 BXI

BXI pour Bull eXascale Interconnect est un type de réseau d'interconnexion (réseau haute performance) développé par Atos. Historiquement développé par Bull qui a été racheté par Atos. Ce réseau est dédié aux communications entre nœuds. Il est composé de carte réseau BXI et de switch BXI. La carte BXI est capable de faire progresser la communication réseau sans aucune intervention du processeur (offload des communications réseau). Le processeur a juste à soumettre une commande dans une file sur la carte et elle s'occupe de tout. Le processeur peut ensuite récupérer une file d'événements pour savoir ce qu'il s'est passé, en somme faire un poll. La carte est

également capable de déclenché des interruptions. L'utilisation de la carte passe donc par une implémentation du protocole Portal4. L'utilisateur utilise donc le protocole pour envoyer et recevoir des paquets réseau.

## 2.5 MPI

MPI pour Message Passing Interface est une interface pour effectués des communications entre plusieurs processus, qui sont souvent sur des noeuds différant et qu'on appel *processus MPI*. Cette interface fourni une abstraction pour transmettre des données entre plusieurs processus. L'abstraction masque la complexité des communications. L'interface permet d'envoyer et de recevoir des messages. Pour cela il y a deux mode de communications :

### 2.5.1 Les communications point à point

C'est à dire entre deux processus MPI, aussi appelé One-to-One. Pour ce faire le processus MPI récepteur vas appelé la fonction `MPI_Recv` qui est bloquante et vas attendre la reception d'un message. L'émetteur vas lui faire un appel à la fonction `MPI_Send` qui est aussi bloquante et vas envoyer un message est attendre que la communication sois fini. L'utilisation des fonctions `MPI_Send` / `MPI_Recv` est donc totalement synchrone ce qui bloque le code. La norme propose aussi une version non bloquante de ces fonctions qui sont `MPI_Isend` et `MPI_Irecv`. Cette version ce contente de posté la communication et rend immédiatement la main. Pour la progression et verifier la terminaison il faut donc d'autre fonctions `MPI_Test` qui vérifie la progression, et la fait si nécessaire, et la fonction `MPI_Wait` qui attend activement la terminaison et s'occupe de la progression si nécessaire. Il est important de noté que la norme ne précise pas si la progression ce fait en tâche de fond ou non, c'est aux implémentations de la norme MPI de choisir. C'est pour cela que la progression peut ce faire au niveau de des fonctions `MPI_Wait` et `MPI_Test` ou être faire avant est donc l'appel aux fonctions s'occupe juste de la terminaison. L'envoi des messages peut donc être asynchrone.

### 2.5.2 Les communications collective

Les communications collective ce fond entre plusieurs processus. Il en existe de différent type :

- un processus vers plusieurs (One-to-All) par exemple un broadcast d'un message
- de plusieurs processus vers un seul (All-to-One) par exemple une reduction (e.g. un processus reçoit la somme des valeurs des autre processus)
- de plusieurs processus vers plusieurs (All-to-All) par exemple quand tous les processus on un message pour les autre



Pour les collective il existe également deux versions, bloquante et non bloquante, qui fonctionne de la même façon que les communications point à point.

### 2.5.3 La progression asynchrone

Pour faire progressé les communications de façon asynchrone il est possible de :

- faire des appels à `MPI_test` régulièrement et faire du calcul entre chaque appel. Cela nous permet de recouvrir la latence des communications par du calcul. La progression se faire également dans d'autre appel aux fonctions MPI. Quand il n'y a plus de calcul à faire on repasse à une progression synchrone par un appel à `MPI_Wait` sauf si la communication est déjà fini.
- utiliser un thread dédié aux progressions. Dans ce cas c'est la bibliothèque MPI qui s'occupe de la progression des communications en tâche de fond grâce à un thread dédié. Il faut donc faire attention au placement des threads et à prendre en compte qu'un thread est déjà utilisé par la bibliothèque de communication. Il faut aussi éviter d'appeler trop souvent `MPI_Test` car ça crée de la contention
- utilisation des threads de façon opportuniste c'est à dire quand un des thread à fini son calcul il va faire progressé les communications. C'est ce qui est fait par Pioman dans NewMadeleine.

## 2.6 NewMadeleine

- NewMadeleine est une bibliothèque de communication...
- elle est basée sur un système de progression asynchrone...
- supporte différents types de communication grâce à un système de driver (portal4 pour BXI, ibverb pour IB, shm...)
- ...

## 2.7 Travaux antérieurs

- Travaux de Mathieu Barbe durant un stage en 2019 sur l'utilisation d'interruption pour transmettre des événements réseau.
- Ces travaux visent à diminuer la latence due au fait de passer par un driver noyau.
- Il parle dans la perspective de traitement direct des interruptions depuis l'espace utilisateur ce qui mène à mon stage.

## 3 Problématiques / Objectifs

### 3.1 Sujet

#### 3.1.1 Nouveau mécanisme d'interruption en espace utilisateur

- très récent...
- sur les dernier CPU Inter Sapphire Rapide...
- 

#### 3.1.2 contentions...

### 3.2 Ne plus faire de Polling

- utiliser des interruption en HPC comme dans les autre domain
- quand un périphériques a besoin de remonté une information il n'y a plus besoin de polling

### 3.3 Objectifs

#### 3.3.1 Objectifs global

- L'objectifs à terme est de faire progressé les communications entre 2 noeuds sans utiliser du Polling
- Donc permettre au carte réseau BXI de d'utiliser les interruption en espace utilisateur.
- Réduire globalement le temps de calcul. (Feedback Mathieu)
- 

#### 3.3.2 Objectif du stage

- Comprendre le fonctionnement de uintr (à partir du manuel du CPU, des manuel du patch et des quelque document disponible)
- Connaître leurs propriétés.
- Mesurer leur performance.
- intégré le mécanisme dans le driver de mémoire partagé (shm) de NewMadeleine, pour des communication entre processus (ipc)
- faire progresser les communication à partir d'un handler d'interruption
- évité les problèmes du polling :
  - réactivité
  - mobilisation d'une ressource ou truffé le code de MPI\_Test (simplifie la vie à l'utilisateur)
- ...

## 4 Exploration de uintr

### 4.1 Prérequis est accès

- difficulté d'accès au processeur...
- J'ai eu accès au processeur 2 mois 1/2 après le début du stage
- J'ai donc eu accès à une machine avec 2 CPU SPR... VPN... détails Saphir Rapid et VPN
- il faut une version patché du noyaux linux pour utilisé c'est nouvelle interruption Ce patch n'est pas disponible en mainline linux. J'ai donc télécharger le patch fourni par intel, je l'ai compiler puis je l'ai installer sur la machine
- Pour utiliser les nouvelle instructions il faut une version récente de GCC... (version). Pas disponible avec LLVM-Clang

### 4.2 Fonctionnement des uintr

TODO : comparaison avec les interruption de base (Feedback Mathieu)

#### 4.2.1 Détail

- 5 instructions
- registre MSR pour l'init

#### 4.2.2 Capacité présente et futur

- process => process
- kernel => process
- device => process
- au niveaux des thread...
- masquage
- le thread dois être en ring-3
- peut fonctionné si le thread est endormi mais j'ai pas testé car ...

#### 4.2.3 Example

- exemple avec schéma

#### 4.2.4 Partage du FD

- Pour mes testes "jouer" j'ai utiliser `uintr_register_self()`
- pipe / socket / URL pour NewMadeleine (on en reparle en section XXX)
- `pidfd_getfd`

#### 4.2.5 +

- comparé au signaux ?
- TODO ...

#### 4.3 Tests du mécanisme

- entre process
- entre threads
- avec alt stack
- test d'écrasement des interruptions
- avec binding (plus pourquoi avec binding que comparaison avec et sans)
- avec turbo boost

#### 4.4 Correction du patch pour l'appel système uintr \_alt \_stack

TODO

#### 4.5 Mesure de la latence (Feedback Mathieu)

TODO : expliquer que le binding est important

#### 4.6 Performances

TODO

- Dans un premier temps j'ai pas bind les thread par pu donc j'avais de mauvais résultats.
- J'ai fait différents bind (on voit que c'est similaire sauf quand on passe entre 2 NUMA)
- si on monte la fréquence c'est mieux (turbo boost)
- 

### 5 Intégration dans NewMadeleine

#### 5.1 Présenté NewMadeleine details

- liste de progression recv
- liste de progression send
- p\_pw
- post
- poll
- driver
- nm\_schedule (appelé par nm\_wait, ...)
- pioman
- existe : progression du nm\_schedule ou de pioman vers le core\_task

## 5.2 Modification

TODO : on désactive le poll avec les driver qui on des handler. Mais on fait toujours un poll qui fonctionne du premier coup.

- ajout d'un driver sig\_shm
- ajout d'un driver uintr\_shm
- progression à partir du handler du driver vers les core\_task
- problématiques de gestion des interruptions
  - on ne peut pas faire d'attente dans les handler Les fonction doivent être async safe
  - on ne peut pas utiliser d'allocateur donc il faut utiliser des p\_pw déjà alloué
  - pour avoir un p\_pw disponible il faut faire progresser le communications
  - la progresser à une partie critique qui naissaisite un verrou
  - si on ne peut pas récupéré le core\_lock (try\_lock) il faut mettre à plus tard le trétement de l'interruption
  - on utilise donc une file lock-free
  - problématique des liste lock-free qui doivent être wait-free
    - le principe d'une lfqueue est d'attente si quelqu'un d'autre modifie la file
    - il faut donc une file wait-free
    - j'ai fait de la biblio
    - décrire les différante solutions
    - solution d'Alexandre
- gestion des multiple interruptions qui s'écrase, (prob\_any / pour les large si la progression à traiter le pipeline courant)
- quand est ce qu'on envois des interruption ?
  - au moment ou on poste le premier paquet d'envois, pour indiquer au recepneur que des données sont disponible (le recepneur à toujours un paquet de reception posté)
  - au moment ou une progression est fait du coté du recepneur, pour indiquer une reception a l'émetteur
  - l'émetteur reçois une iterruption est détermine si l'émission est fini, petit paquet, ou si il faut envoyer la suite, paquet large.

## 5.3 Suite de tests

Tous les tests ne passe pas ?

## 5.4 Performances

### 5.4.1 Résultats avec attente active

TODO : voire le sur coup des interruption

#### **5.4.2 Résultats recouvrement communication par du calcule**

TODO : courbe overlap reception

### **6 Bilan**

### **7 Remerciements**

8 Annexes

9 Bibliographie