

Mémoire de stage Master 2

Stage du 1er 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)

17 juillet 2023

Table des matières

1 Introduction

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

1.1 Presentation Inria

Inria est TODO : here

1.2 Presentation Atos (Eviden)

en restructuration je parle donc d'Atos dans ce document

1.3 Environnement de travail

- labo de recherche
- un bureau dans l'open space de l'équipe projet TADaaM.
- Salle de visio pour la réunion de suivie de stage toute le semaine
- Baby foot ?
- Cafeteria ?
- ... ?

1.4 Le cadre

Inria (TADaaM) :

- objectifs du service : faire de la recherche...
- type de personnel : chercheur, post-doc, ingénieur, doctorant et stagiaire
- Culture informatique : HPC, système... ?
- Matériel et logiciels mis à disposition :
  - un ordinateur portable
  - un doc avec écran, clavier et souris
  - service inria (email, mattermost, webex...)
- contraintes : je suis libre pour la démarche, les outils...

Atos (BXI-LL) :

- objectifs du service : développé le soft ba niveau pour la NIC BXI... (entre MPI et la carte) donc driver, lustre, portal...
- type de personnel : ingénieurs... ?
- Culture informatique : HPC, système... ?
- Matériel et logiciels mis à disposition :
  - une machine avec 2 CPU SPR accessible via VPN
- contraintes : je suis libre

1.5 Presentation du plan

Context dans le quelle le stage ce place puis les objectifs a long term et le objectifs du stage. Le déroulement du stage avec les résultats. Pour finir un bilan et la suite. TODO :

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

De nos jour les supercalculateurs sont composé de plusieurs ordinateur que l'on appelle noeud de calculs. Ceci sont regroupé en grappe (cluster), il sont tous vue est utiliser comme une seul grande machine. Ce fonctionnement pose des questions sur la répartition du calcule, la distribution de la mémoire, la communication entre les différant noeuds... Dans le contexte du stage, nous nous sommes concentrés sur les communications entre noeuds. Les noeuds sont connecté entre eux par un réseau haute performance. Ce réseau est dédiés au communications est n'est pas forcément de type Ethernet. La gestion des noeuds est généralement effectuée par un second réseau plus traditionnel (Ethernet ; TCP/IP). Ce genre de réseau ont une latence autour de quelque microseconde. Chaque noeuds possède une ou plusieurs carte réseau et il faut les programmé.

2.2 OS bypass

Le stage ce passe donc aussi dans un context système. Chaque noeuds à sont propre système d'exploitation qui permet la gestion des ressources, des processus, des fichiers, des périphériques. Pour cela le système à 2 espaces :
- un espace noyau où seul le code du système peut s'exécuter. Le code du système peut donc modifié n'importe quelle endrois de la mémoire, exécuté n'importe quelle instructions...

— un espace utilisateur où le code de l'utilisateur est exécuter. Cette espace est limité par le noyau qui controle ce que fait l'utilisateur.

En temps normale les périphériques sont programmé directement depuis le noyau du système d'exploitation ceci pour des questions de sécurité, de stendardisation des accés... Lors que l'on passe par le noyau (kernel en anglais) on à un surcoût qui n'est pas négligeable dans notre cas. Pour passer par le noyau on utilise généralement des appels système qui ce présente sous la forme d'une fonction. Un appel système vas effectué un changement de contexte (context switch) pour passé de l'espace utilisateur à l'espace noyau. Ce changement de context est coûteux car il sauvegarde les états du code de l'utilisateur avant d'exécuté 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 passé de l'espace noyau à l'espace utilisateur, et donc restoré l'états du code de l'utilisateur. Donc le fait de passé par un appel système coûte plusieurs microseconde. On vois 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 registre de la carte réseau dans l'espace d'adressage virtuelle du processus utilisateur.

Pour transmettre des événement à l'utilisateur les périphériques utilise généralement les interruptions, qui passe 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 scruté (poll) régulièrement si on à reçus un événement. Concrètement cela consiste à lire une zone mémoire modifier par la carte réseau et voire si un bit est passé à 1. Pour cela il es possible de dédié un thread qui vas faire de l'attente active, scruté sans cesse si un événement à été reçus. Mais on perd une unité de calcule lors ce que le thread est ordonnancer donc de la puissance de calcul, donc cette technique est peu utilisé. Il est aussi possible d'entrelacés le code de l'utilisateur avec des scrutation, c'est fréquemment utiliser mais ça oblige à l'utilisateur de prendre en compte la progression. Une autre solution qui est utiliser par Pioman dans NewMadeleine (j'en parlerai plus tard) consiste à faire ces scrutation de manière opportuniste dans les threads qui on fini leur calcul, mais pour cela il faut déjà utilisé plusieurs thread et avoir une application qui à des calculs d'une durée hétérogène.

Il faut donc, peut importe la technique, régulièrement scruté pour faire progressé les communications.

2.3.1 Inconvénients du polling

Dans le cas d'un thread dédier qui fait de l'attente active la réactivité est excellente hormis quand il y à plus de thread que d'unité de calcule est que le thread n'est pas ordonnancer.

Quand on entrelaces le calcule est les scrutations la réactivité est moins bonne car il faut attendre que le calcule sois fini pour faire un poll. C'est ce qui est fait habituellement.

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

Donc un chois dois être fait entre perdre de la capacité de calcule ou perdre en réactivité. En plus on perd un peut de temps de calcul à effectué du polling.

2.4 BXI

BXI est un type de réseau dédier pour le communications entre noeuds...

- les carte réseaux BXI (Bull eXascale Interconnect) sont dévelopé par Atos
- elle sont capable de faire des communications sans intervention du CPU
- le CPU à juste à soumettre une commande et la carte s'occupe du reste
- pour savoir si la communication est fini l'application peut poll ou recevoir une interruption en provenance de la carte
- portal4
-

2.5 MPI

- présenté MPI
- les communications asynchrone
- la progression ce fait :
  - au moment des appelle a la biblio (MPI_Isend, MPI_Irecv, MPI_test, MPI_wait).
  - grâce à un thread dédier
  - grâce à une politique d'utilisation des threads de façon opportunist (quand ils ne font pas de calcule).
  - ...
-

2.6 NewMadeleine

- NewMadeleine est une bibliothèque de communication...
- elle est basé sur un système de progression asynchrone...

— supporte différant type de communication grâce a un système de driver (portal4 pour BXI, ibverb pour IB, shm...)
— ...

2.7 Travaux antérieur

— Travaux de Mathieu Barbe durant un stage en 2019 sur l'utilisation d'interruption pour transmettre des événements réseau.
— Ces travaux vis à diminué la latence du au fait de passer par un driver noyaux.
— Il parle dans les perspective de directement traitement 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. (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...
— 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

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

— example 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
— avec turbo boost

4.4 Correction du patch pour l'appel système uintr_alt_stack

TODO

4.5 Mesure de la latence (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 vois que c'est similaire sauf quand on passe entre 2 NUMA)
— si on monte la frequence 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 recepteur que des données sont disponible (le recepteur à toujours un paquet de reception posté)

— au moment ou une progression est fait du coté du recepteur, pour indiquer une reception a l'éméteur
— l'éméteur reçois une iterruption est détermine si l'émision 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 Remerciements

7 Annexes