Université Pierre et Marie Curie

Mastère de sciences et technologies

MENTION INFORMATIQUE
2014 – 2015

Spécialité : SESI

SYSTÈMES ÉLECTRONIQUES ET SYSTÈMES INFORMATIQUES

Exécution de plusieurs systèmes d'exploitation sur une puce manycore CC-Numa sécurisée

RAPPORT DE PRÉSOUTENANCE

11 mai 2015

PRÉSENTÉ PAR

JEAN-BAPTISTE BRÉJON

ENCADRANT

QUENTIN MEUNIER

Laboratoire d'accueil : LIP6

SOC - ALSOC

Table des matières

Table des figures

Chapitre 1

Contexte du stage

Ce stage s'inscrit dans le cadre du projet ANR TSUNAMY [1]. Ce projet adresse la problématique de la manipulation sécurisée des données personnelles et privées sur l'architecture CC-NUMA [2] TSAR [1] [3]. Ce stage s'intitule *Exécution de plusieurs systèmes d'exploitation sur une puce manycore CC-Numa sécurisée*. Il vise à proposer une solution de confiance par construction permettant d'exécuter plusieurs systèmes d'exploitation (SE) de manière indépendante sur la même architecture, en garantissant le cloisonnement des systèmes d'exploitation entre eux.

1.1 L'architecture TSAR

TSAR est une architecture à mémoire distribuée organisée en clusters. Ceux-ci sont composés de :
- Quatre processeurs ayant chacun un cache de niveau 1 et partageant un cache de niveau 2.
- Un interconnect local
- Un concentrateur d'interruption (XICU)
- Un Direct Memory Access (DMA) [4]

Les clusters sont connectés entre eux par un réseau sur puce (NoC [5]) de topologie Mesh-2D.

1.2 La plateforme

La plateforme (figure 1.1) utilisée durant ce stage comporte quatre clusters. Le cluster 0 contient, en plus des composants déjà énoncés, des périphériques non répliqués :
- un contrôleur de disque (IOC)
- un contrôleur de terminaux (TTY)
- une Read Only Memory (ROM)

Chaque cluster contient un banc mémoire de capacité 62Mo (0x03e000000). La répartition des adresses par cluster est faite de la manière suivante :

Coordonnées (x, y) du cluster	Numérotation TSAR	Plage d'adresses
(0, 0)	0	0x00000000 – 0x03e00000
(0, 1)	1	0x40000000 – 0x43e00000
(1, 0)	2	0x80000000 – 0x83e00000
(1, 1)	3	0xc0000000 – 0xc3e00000

La plateforme TSUNAMY contient, devant chaque initiateur, une HAT [6] (voir section 1.6).

1. https ://www.tsunamy.fr/trac/tsunamy
2. Cache-Coherent Non Uniform Memory Access
3. Tera Scale Architecture : https ://www-soc.lip6.fr/trac/tsar
4. réalise des transfert de données de taille arbitraire d'un endroit de la mémoire à un autre
5. Network On Chip : permet le routage des requêtes en dehors des clusters
6. Hardware Address Translator

1.3 Système d'exploitation ALMOS

ALMOS [2] est un système d'exploitation développé au département SoC[7] du laboratoire LIP6[8]. Il a été développé par Ghassan ALMALESS dans le cadre de sa thèse. ALMOS cherche à répondre aux problématiques liées aux architectures CC-NUMA manycore, notamment le placement optimisé des données.

Nous considérons plusieurs instances de système d'exploitation dans la plateforme qui seront toutes des instances d'ALMOS mais qui pourraient être un autre système d'exploitation. Les instances des SE seront présentes sur les disques de la plateforme au démarrage de la simulation.

1.4 Virtualisation

Le principe de la virtualisation [3] est de permettre à plusieurs SE de s'exécuter en même temps sur la même machine. Les motivations pour l'utilisation de cette approche sont nombreuses (utilisation optimale des ressources matérielles, faciliter la migration des machines virtuelles d'une machine physique à une autre...). Voici la définition de quelques termes qui seront utilisés tout au long de ce rapport :

- Une VM (Virtual Machine) : est une machine émulée par un hyperviseur, il s'agit d'un sous-ensemble de la plateforme TSAR complète.
- Un hyperviseur, aussi appelé moniteur de machines virtuelles, est un environnement d'exécution de VMs.
- Un système invité est un SE qui s'exécute dans une VM.
- Un système hôte est le SE dans lequel l'hyperviseur s'exécute.

Par l'intermédiaire de la virtualisation, chaque VM doit avoir l'impression de s'exécuter en mode privilégié, d'avoir un espace d'adressage continu et de manipuler de la mémoire physique. Elle doit aussi avoir accès à des périphériques, bien que ceux-ci ne sont pas forcement les mêmes que ceux de la machine hôte.

Il existe trois approches pour l'accès au matériel : la full-virtualisation, la para-virtualisation et l'utilisation directe du matériel.

1.4.1 La full-virtualisation

Dans cette approche, le matériel est complètement émulé par l'hyperviseur. L'avantage est qu'il n'y a pas besoin de modifier le code du système invité, il est donc possible de virtualiser n'importe quel SE. Mais il y a aussi un inconvénient de taille lié aux performances. En effet, dès que le système invité veut accéder au matériel, il doit passer par l'hyperviseur. De plus ces requêtes ne sont pas forcement optimisées car le système invité n'a aucune connaissance du matériel réellement présent sur la machine hôte.

1.4.2 La para-virtualisation

Le principe de cette approche consiste à modifier le système invité de manière à ce qu'il fonctionne en collaboration avec l'hyperviseur. Le système invité associe une page vers une adresse physique sans passer par l'hyperviseur. Ainsi les accès aux périphériques s'effectueront plus rapidement.

1.4.3 Solution matérielle

Cette solution est apparue en 2003 avec Intel-Vt, et a été rajoutée dans AMD-V depuis 2006. Le principe est d'ajouter un niveau de privilège pour les VMs. Ainsi lors d'un appel système ou d'une instruction privilégiée exécutée par une VM, celle-ci peut directement exécuter son instruction sur le processeur qui appellera une fonction de l'hyperviseur. Avec cette solution on peut utiliser les instructions spéciales fournies par les processeurs sans avoir à modifier le système invité.

Ces trois solutions ne conviennent pas au projet ANR TSUNAMY, car elles donnent trop de pouvoir à l'hyperviseur. On souhaiterait en effet lui interdire l'accès à la mémoire des VMs une fois celles-ci lancées. Cela n'est possible que dans une architecture ou les ressources sont physiquement distinctes.

7. http ://www-soc.lip6.fr/
8. http ://www.lip6.fr/

1.5 L'hyperviseur

Un hyperviseur [4], [5] permet la virtualisation d'un ou plusieurs systèmes d'exploitation sur la même plateforme.

On peut distinguer deux types d'hyperviseur. Pour le premier, qui est dit natif, le code s'exécute directement sur la plateforme. Il n'y a donc aucun intermédiaire entre le matériel et l'hyperviseur. Le second type d'hyperviseur est un logiciel qui s'exécute à l'intérieur d'un système d'exploitation. L'hyperviseur qui sera développé durant ce stage est du premier type.

le rôle de l'hyperviseur est de distribuer les ressources de la plateforme (clusters et canaux des périphériques) entre les différentes VMs virtualisées. Les clusters alloués pour les VMs devront respecter une contrainte topologique, à savoir former une structure rectangulaire. Cette contrainte est justifiée par le fait qu'ALMOS doit avoir une connaissance de la topologie réelle pour être efficace.

L'hyperviseur doit se charger de démarrer et d'isoler les VMs.

Chaque VM sera exécutée non modifiée. Elle produira donc des adresses entre 0x0000 0000 et 0xFFFF FFFF. Or, on souhaite contraindre la VM aux clusters qui lui ont été alloués. La plateforme modifiée pour TSUNAMY fournit un composant, appelé HAT (Hardware Translation Table), réalisant la traduction des adresse emises par la VM (0x0000 0000 - 0xFFFF FFFF) vers l'espace mémoire des clusters qui ont été alloués à la VM (voir section 1.6). Les adresses contenues dans l'espace mémoire des clusters sont appelées adresses machine.

La Hat permet aussi de répondre à d'autres problématiques posées par le projet ANR TSUNAMY, notamment la manipulation privée des données et le fait que cette manipulation doit ce faire de manière sécurisée. En effet l'hyperviseur peut être le sujet d'attaques ou de bugs potentiels, il ne doit donc pas pouvoir accéder à l'intérieur des machine virtuelles une fois celle ci lancées (on le dit aveugle). Il en est de même pour les machines virtuelles, une fois lancées toutes les requêtes envoyées depuis l'intérieur de la machine virtuelle ne doivent pas pouvoir en sortir. La HAT doit néanmoins faire une exception lorsque la requête est à destination d'un périphérique non répliqué. Les HATs permettent aussi L'isolation des machines virtuelles vis-à-vis des autres machines virtuelles et de l'hyperviseur.

1.6 La HAT

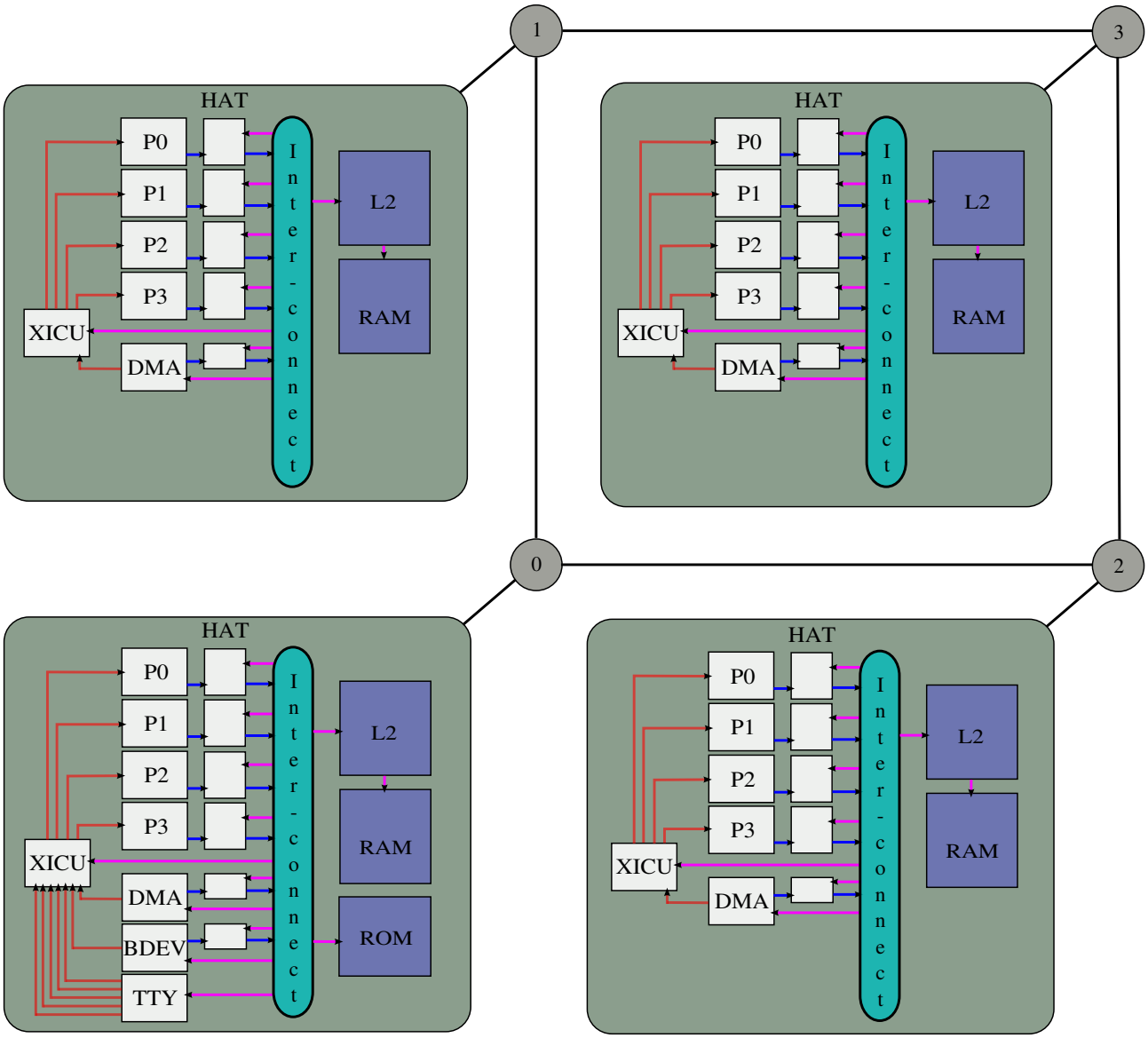


FIGURE 1.1 – Représentation de la plateforme

La figure 1.1 est une représentation de la plateforme telle quelle m'a été fournie. C'est une plateforme à 4 clusters basée sur l'architecture TSAR avec un nouveau composant devant chaque initiateur : le composant HAT.

La HAT un composant matériel configurable placé devant tous les initiateurs [9] d'un cluster. De la même façon qu'une MMU [10], la HAT traduit à la volée des adresses physiques en adresses machine, mais à granularité cluster. La configuration de la HAT est confiée à l'hyperviseur, elle sera effectuée sur tous les clusters de la VM lors de son lancement. La HAT permet l'isolation des machines virtuelles entre elles et l'hyperviseur. Elle permet de restreindre la portée des requêtes émises par les machines virtuelles au(x) cluster(s) qui leurs ont été alloués.

Les registres à configurer dans les HATs sont les suivants :

HAT_MODE - R/W Ce registre permet de choisir le mode de la HAT parmi les 4 suivants :

HAT_MODE_BLOCKED Ne laisse passer aucune requête

HAT_MODE_FAILURE Erreur de traduction

HAT_MODE_IDENTITY Laisse passer les requêtes sans transformation

HAT_MODE_ACTIVATE Laisse passer les requêtes et effectue une transformation

HAT_MX0 - W : Contient la valeur en X de la position du cluster 0 de la machine virtuelle dans la plateforme

HAT_MY0 - W : Contient la valeur en Y de la position du cluster 0 de la machine virtuelle dans la plateforme

HAT_PXL - W : Contient le log2 de la largeur en X de la machine virtuelle

HAT_PYL - W : Contient le log2 de la largeur en Y de la machine virtuelle

HAT_TTY_PBA - W : Contient l'adresse physique de base du premier canal des TTYs

HAT_TTY_MASK - W : Permet de déterminer la plage d'adresse accessible

HAT_FB_PBA - W : Contient l'adresse physique de base du segment mémoire du Frame Buffer

HAT_FB_MASK - W : Permet de déterminer la plage d'adresse accessible

HAT_IOC_PBA - W : Contient l'adresse physique de base du canal du contrôleur de disque

HAT_IOC_MASK - W : Permet de déterminer la plage d'adresse accessible

La figure 1.2 montre un exemple de traduction d'adresse par la MMU et par la HAT dans une plateforme de 4 clusters. L'adresse est émise par un processeur se trouvant dans une VM lancée sur les clusters 1 et 3. La largeur en X de la VM est de 2 et de 1 en Y. Pxl représente le nombre de bits nécessaires pour coder la largeur en X de la plateforme, soit 1 bit dans l'exemple. Pyl est l'homologue de pxl mais pour la largeur en Y, soit 0 bit dans l'exemple.

La MMU traduit l'adresse 0x83681424 en 0xB1487424. Dans une VM de deux clusters, l'adresse commençant par 0xB est présente dans le banc mémoire du 2ème cluster (plage d'adresse physique : 0x8000 0000 - 0xFFFF FFFFF). Ici, le 2ème cluster de la VM est le cluster 3, il correspond au 4ème cluster dans la plateforme (plage d'adresse machine : 0xC000 0000 - 0xFFFF FFFF).

La HAT va donc fournir la traduction d'une adresse appartenant au 2ème cluster de la VM vers le 4ème cluster de la plateforme.

9. composant ayant la capacité d'initier un transfert (couple requêtes réponse)
10. Memory Management Unit : fournit en s'indexant sur une table des traduction d'adresse à granularité page (une traduction physique pour chaque page virtuelle)

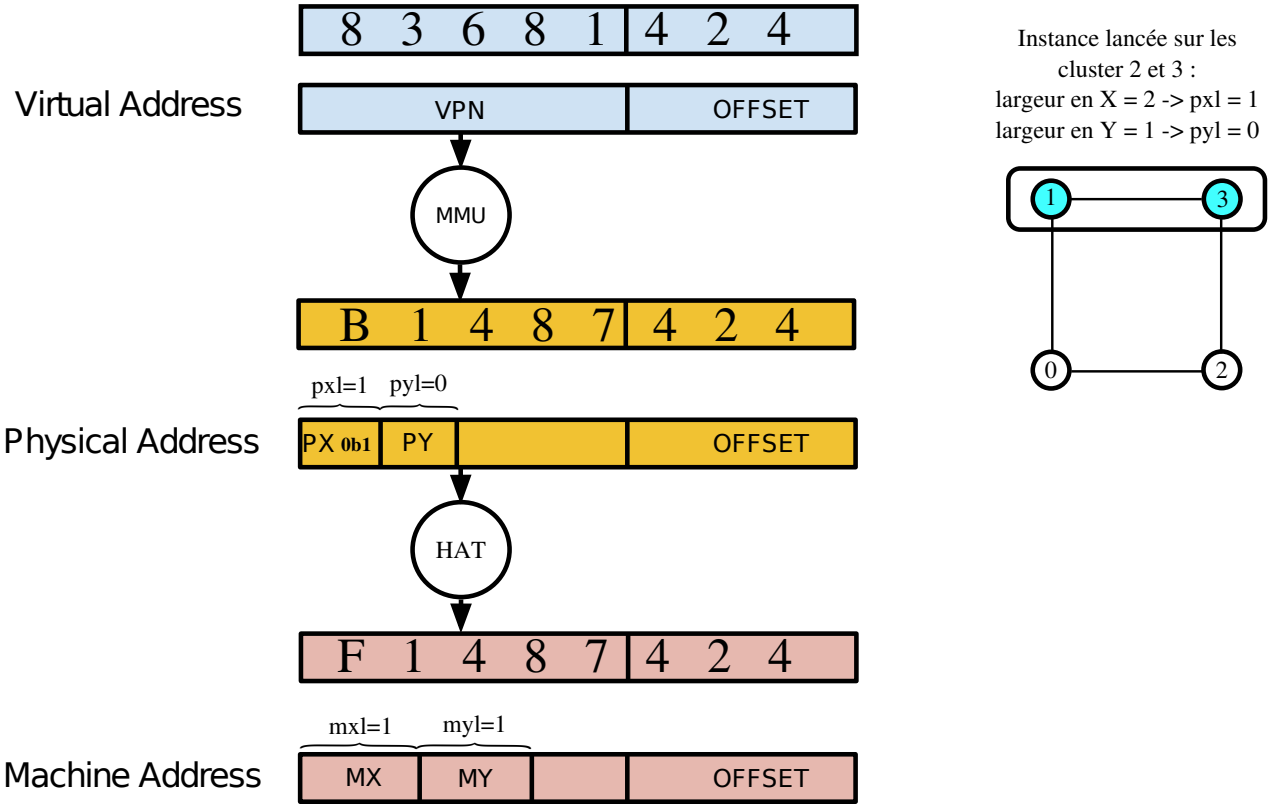


FIGURE 1.2 – Exemple de traduction

Tableau représentant les traductions générées par la HAT dans le cas de la VM présentée dans la figure 1.2 :

adresse émises	traduction		adresse émise	traduction
0x00000000	0x40000000		0x80000000	0xc0000000
0x10000000	0x50000000		0x90000000	0xd0000000
0x20000000	0x60000000		0xa0000000	0xe0000000
0x30000000	0x70000000		0xb0000000	0xf0000000
0x40000000	0x40000000		0xc0000000	0xc0000000
0x50000000	0x50000000		0xd0000000	0xd0000000
0x60000000	0x60000000		0xe0000000	0xe0000000
0x70000000	0x70000000		0xf0000000	0xf0000000

Chapitre 2

Définition et analyse du problème

Les objectifs de ce stage sont les suivants :

1. Adaptation d'une plateforme TSAR à 4 clusters pour la rendre compatible avec les besoins du projet (intégration des HATs, modification du routage des interruptions, adaptation du nombre de canaux des périphériques...)
2. Définition d'un mécanisme de boot de l'hyperviseur et des processeurs non actifs
3. Ecriture de la partie "VMLoader" (Virtual Machine Loader) de l'hyperviseur, en charge d'initialiser la mémoire du kernel, de configurer les HAT et de réveiller les processeurs
4. Réalisation du monitoring des différents systèmes lancés via un multiplexage de leur terminaux

2.1 Canaux des périphériques

Les canaux des périphérique ne posent de problème que dans le cas des périphériques non répliqués : le composant IOC et le composant TTY.

Composant MULTI_TTY :
Le composant TTY est déjà multi-canaux. Lors d'une utilisation classique, dans laquelle une instance d'ALMOS est lancée sur toute la plateforme, ce composant est initialisé avec 4 canaux car ALMOS a besoin de 4 terminaux. Comme présenté dans les objectifs, les TTYs devront être multiplexés entre les différentes VMs lancées. Le nombre de fenêtres "terminal" (processus Unix gérant la fenêtre) ne doit pas changer en fonction du nombre de VMs possibles sur la plateforme : une fenêtre pour le terminal hyperviseur, et 4 fenêtres pour toutes les instances d'ALMOS. L'affichage de ces 4 fenêtres devra donc pouvoir être changé pour afficher les informations de l'une ou l'autre des VMs lancées sur la plateforme.

Composant IOC :
Chaque VM a son propre disque, et la gestion de ces disques doit se faire de façon transparente. Or le composant actuel ne gère pas plusieurs disques.

Le composant IOC fait l'objet d'un autre stage se déroulant parallèlement à celui-là, ce composant devra à terme permettre de crypter et decrypter le code et les données d'une VM. Néanmoins, pour pouvoir tester l'hyperviseur et lancer plusieurs VM, il sera nécessaire d'adapter le composant actuel de façon à le rendre multicanaux.

2.2 Routage des interruptions

Comme vu dans la figure 1.1, les interruptions des périphériques non répliqués (IOC, TTY) sont dirigées vers l'XICU du cluster 0 (cluster hyperviseur). Cela signifie qu'une VM lancée sur la plateforme n'a pas de moyen direct de se faire signifier la fin d'un transfert de l'IOC ou la frappe d'un caractère au clavier sur un des terminal qui lui a été associé. Une des solution serait de passer par l'hyperviseur : l'hyperviseur signalerait à un

des processeurs de l'instance la levée de l'interruption qui lui aura été signalée (via une IPI). Cela implique que l'XICU d'un cluster appartenant à l'instance puisse recevoir une requête de l'hyperviseur. Or, comme expliqué précédemment, on se l'interdit. Il faut donc développer un mécanisme pour transmettre l'interruption sans l'intervention directe de l'hyperviseur.

2.3 Démarrage d'une VM

Le démarrage d'une VM s'effectuera grâce à un shell lancé sur le terminal hyperviseur. Le but est de démarrer la VM dans un environnement sécurisé.

La séquence de démarrage d'une VM est la suivante :

1. Allocation des clusters
2. Allocation des canaux des périphériques non répliqués
3. Construction de la description de l'architecture
4. Chargement depuis le disque du bootloader et du kernel du SE
5. Isolation des clusters alloués à la VM
6. Réveil des processeurs alloués à la VM par l'hyperviseur (envoi d'une IPI en spécifiant l'adresse du bootloader)

Le problème dans cette séquence de démarrage se situe dans les étapes 5 et 6. En effet, l'hyperviseur s'exécute sur le cluster 0 de la plateforme et ne peut pas envoyer d'IPI à une VM isolée. L'étape d'isolation des clusters consiste en la configuration des HATs de telle sorte qu'aucune écriture venant de l'extérieur de la VM, i.e. des clusters qui ne lui ont pas été alloués, ne soit pas acceptée par le cluster visé.

Inverser les étapes 5 et 6 n'est pas non plus satisfaisant, car cela signifierait que la VM démarre dans un environnement non isolé, et peut donc accéder à toute la plateforme. ALMOS a été compilé pour s'exécuter dans le cluster 0 de la VM à une certaine adresse : là ou l'hyperviseur l'a chargé en mémoire (étape 4). À ce moment-là, l'isolation n'est pas encore réalisée, et les HATs ne fournissent dont pas de traduction. Les requêtes des processeurs ne sont donc pas dirigées vers la RAM du cluster où se trouve le code d'ALMOS. Il est donc nécessaire de trouver une autre solution qui conserve l'ordre des étapes 5 et 6.

Chapitre 3

Principe de la solution envisagée

3.1 Solution au problème des canaux des périphériques

En ce qui concerne l'IOC, la solution est simple : il faut plusieurs canaux. Le nombre de canaux optimal est déterminé par le minimum entre le nombre de disques et le nombre de clusters - 1 ; cette valeur peut être vue comme le nombre maximum de VM qui peuvent être lancées sur la machine.

Pour les TTYs, il est nécessaire de créer un nouveau composant à partir de celui existant. Ce nouveau composant permettra le support par une fenêtre de N "fichiers" (un fichier étant un terminal virtuel utilisé par une VM ALMOS). Le multiplexage de ces fichiers sur la fenêtre sera effectué par le nouveau composant, et le choix de la VM à afficher par une commande entrée par l'utilisateur sur le terminal hyperviseur. L'utilisateur pourra spécifier dans la commande le numéro du terminal virtuel que l'on souhaite afficher sur les 4 fenêtres des VMs ALMOS.

3.2 Solution au problème du routage des interruptions

La solution envisagée pour répondre à ce problème est d'utiliser un périphérique permettant d'émettre une requête sur le réseau lorsqu'il reçoit une interruption matérielle. Ce composant est appelé IOPIC pour "In/Out Programmable Interrupt Controller" et il existe déjà dans la bibliothèque Soclib.

Ce nouveau périphérique sera placé dans le cluster où se trouve les périphériques non-répliqués, c'est à dire le cluster 0. Il permet de traduire un certain nombre d'interruptions matérielles (HWI) en interruptions logicielles (WTI). L'adresse à laquelle chaque WTI sera envoyée peut être configurée, l'hyperviseur y mettra l'adresse de l'XICU correspondant au cluster 0 d'une VM.

Ce sera donc l'IOPIC qui signifiera, via le réseau sur puce, à l'XICU du cluster 0 d'une VM qu'un transfert est terminé sur un des périphériques alloués à cette VM.

Les fenêtres peuvent afficher alternativement les informations des VMs. Il faut donc que les interruptions générées lorsqu'une touche est frappée soient redirigées vers la bonne VM. L'IOPIC devra donc être reconfiguré lorsque l'utilisateur fera la demande d'afficher un autre terminal virtuel.

3.3 Démarrage d'une VM

L'idée pour le démarrage d'une VM est que chaque processeur contenu dans les clusters appartenant à la future VM exécute du code hyperviseur permettant l'isolation de la VM : ce code doit effectuer l'activation de la HAT du processeur qui l'exécute. Autrement dit, la future VM est isolée de l'intérieur.

L'hyperviseur devra donc réveiller un processeur appartenant à la VM en plaçant l'adresse du point d'entrée d'ALMOS dans une zone mémoire réservée aux arguments. Cet envoi d'IPI est possible car la VM n'est pas encore isolée. Lorsque le processeur se réveillera, il sautera à l'adresse de ce bout code que l'on appellera RHA (Remote HAT Activation). Une fois dans la fonction RHA, le processeur se chargera de récupérer l'adresse du point d'entrée d'ALMOS préalablement stockée, d'activer sa HAT, puis de sauter au point d'entrée d'ALMOS.

Plus tard, ALMOS réveillera les autres processeurs, qui exécuteront la fonction RHA.

Chapitre 4

Identification des tâches à accomplir

4.1 Tâches

Hyperviseur

L'hyperviseur sera basé sur un code déjà existant. Les fonctionnalités déjà présentes sur celui sont les suivantes :

- Fonction permettant la configuration du composant MULTI_TTY
- Fonction permettant la configuration du composant IOC
- API pour lire depuis le disque un fichier ELF (fichier exécutable d'unix)

Les fonctions à rajouter sont les suivantes :

- Démarrage des processeurs inactifs et de l'hyperviseur
- Construction de la structure représentant l'architecture
- Configuration des HATs
- Configuration de l'IOPIC
- Développement du mécanisme de boot d'une VM
- Développement des fonctions permettant la configuration du composant XICU
- Développement du Shell hyperviseur

Matériel

En ce qui concerne le matériel, il faut créer deux autres composants :

MULTI_IOC : Ce composant est sensiblement le même que l'IOC mais supporte plusieurs disques et est multi-canaux.

MULTI_TTY_VT : comme pour l'IOC, le composant MULTI_TTY_VT sera fortement inspiré du composant MULTI_TTY mais supportera plusieurs fichiers (terminaux virtuels) par fenêtre (canal) et ajustera le nombre de canaux. Auparavant, le nombre de canaux était le même que le nombre de fenêtres, il faudra dans notre cas multiplier celui-ci par le nombre de terminaux virtuels.

4.1.1 Tâche 1 : Démarrage d'une VM sur le cluster 1

Cette tâche nécessitera l'implémentation des fonctions suivantes :

- Démarrage des processeurs inactifs et de l'hyperviseur
- Construction de la structure représentant l'architecture
- Configuration des HATs
- Développement du mécanisme de démarrage d'une VM
- Développement des fonctions permettant la configuration du composant XICU

On redirigera le fil d'interruption de l'IOC et les quatre fils des TTYs vers le cluster 1 pour que ALMOS soit en mesure de lire ses applications depuis le disque.

4.1.2 Tâche 2 : Introduction IOPIC

Cette tâche consistera en l'introduction de l'IOPIC dans la plateforme ainsi que le développement des pilotes de configuration de celui-ci. Les lignes d'interruption de l'IOC et les 4 autres lignes d'interruption des TTYs n'ont plus besoin d'être redirigées dans la plateforme. Elles seront routées de façon logicielle. Cette partie nécessitera des modifications dans ALMOS pour que celui-ci supporte les WTI envoyées par l'IOPIC.

4.1.3 Tâche 3 : Composant MULTI_IOC - Démarrage de deux VM

La première VM sera placée sur les clusters 1 et 3 et la seconde sur le cluster 2. Cette tâche nécessitera l'implémentation du composant MULTI_IOC. La plateforme sera créée avec 9 terminaux physiques (fenêtres) qui seront distribués comme suit :

- les 4 premiers seront réservés à la première instance
- les 4 suivants seront réservés à la seconde instance
- le dernier terminal sera réservé à l'hyperviseur

4.1.4 Tâche 4 : Shell hyperviseur

Le développement du shell hyperviseur est nécessaire pour réaliser le multiplexage des terminaux, il doit dont être fait avant. La seule commande qui sera disponible sur le shell sera la suivante :

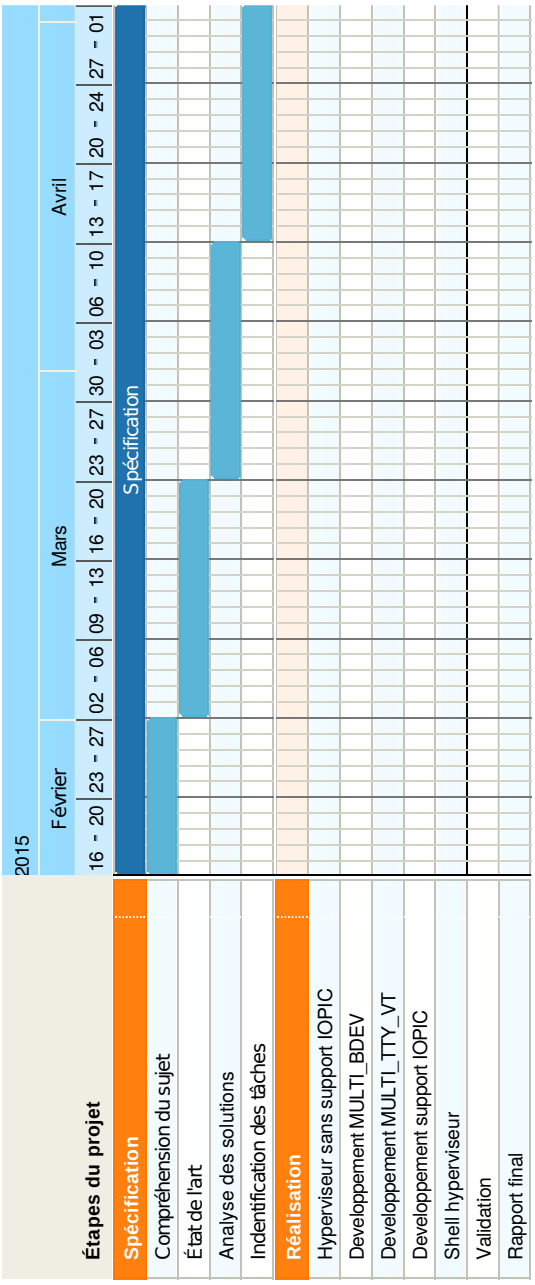
run : Permet le lancement d'une VM, l'utilisateur devra spécifier en argument de cette commande le numéro de l'instance du SE ainsi que le nombre de clusters sur lesquels devra être lancée la VM.

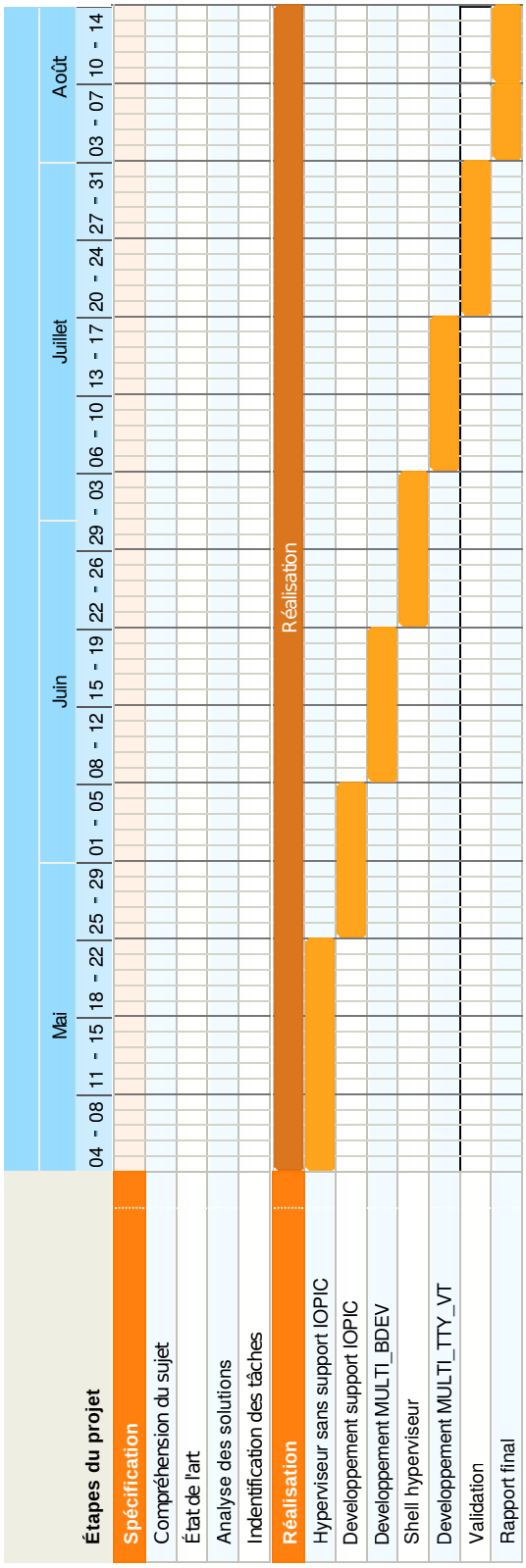
4.1.5 Tâche 5 : MULTI_TTY_VT

Cette tâche consistera en la conception du composant MULTI_TTY_VT, le développement des pilotes dans l'hyperviseur ainsi que l'introduction d'une nouvelle commande dans le shell hyperviseur :

switch : Permet de basculer l'affichage des quatre terminaux réservés aux VMs. L'utilisateur devra spécifier en argument de cette commande le numéro de la VM qu'il souhaite afficher sur les 4 terminaux.

4.2 Planning


2015
Février
Mars
Avril
16 - 20
23 - 27
02 - 06
09 - 13
16 - 20
23 - 27
30 - 03
06 - 10
13 - 17
20 - 24
27 - 01
Étapes du projet
Spécification
Compréhension du sujet
État de l'art
Analyse des solutions
Indentification des tâches
Réalisation
Hyperviseur sans support IOPIC
Developpement MULTI_BDEV
Developpement MULTI_TTY_VT
Developpement support IOPIC
Shell hyperviseur
Validation
Rapport final
Spécification


Mai
Juin
Juillet
Août
04 - 08
11 - 15
18 - 22
25 - 29
01 - 05
08 - 12
15 - 19
22 - 26
29 - 03
06 - 10
13 - 17
20 - 24
27 - 31
03 - 07
10 - 14
Étapes du projet
Spécification
Compréhension du sujet
État de l'art
Analyse des solutions
Indentification des tâches
Réalisation
Hyperviseur sans support IOPIC
Developpement support IOPIC
Developpement MULTI_BDEV
Shell hyperviseur
Developpement MULTI_TTY_VT
Validation
Rapport final
Réalisation

Chapitre 5

Procédure de recette

La procédure de recette de ce stage sera principalement fonctionnelle. Elle consistera en la création et l'utilisation de deux instances ALMOS sur une plateforme décrite ci-dessous :
- 4 clusters
- Un composant MULTI_TTY_VT dans le cluster 0
- Un composant MULTI_IOC dans le cluster 0

Un cluster contient : 4 processeurs, une XICU, un DMA, une RAM, une HAT devant chaque initiateur.

La procédure de recette se déroulera comme suit :

1. Lancer une VM ALMOS sur le cluster 1 sans IOPIC
2. Lancer une VM ALMOS sur le cluster 1 avec IOPIC
3. Démarrage de deux VM ALMOS
4. Démarrage des VMs en utilisant le shell hyperviseur
5. Lancer l'application "hello" sur les deux VMs

Étape 1 : Validation de la construction de la structure décrivant l'architecture, de la configuration des HATs et du mécanisme de démarrage
Dans cette étape on s'attend à voir ALMOS démarrer sur les 4 terminaux qui lui ont été réservés. Le code du shell d'ALMOS se situe sur le disque. L'affichage du shell validera donc le bon fonctionnement du disque et que le routage de l'interruption est correct. Une fois le shell affiché, la saisie d'une touche au clavier ainsi que son affichage sur le shell d'ALMOS validera les TTYs ainsi que le routage des interruptions correspondant au 4 canaux associés à ALMOS.

Étape 2 : Validation de l'introduction de l'IOPIC ainsi que sa configuration
Le processus de validation de cette étape est le même que celui présenté à l'étape 1.

Étape 3 : Validation du fonctionnement du composant MULTI_IOC et de la bonne configuration de l'IOPIC pour les 2 VMs
On considèrera que le composant MULTI_IOC est validé si on observe que les ALMOS affichent leurs shells. Cela signifiera qu'ils ont réussi à lire le code du shell depuis leurs disques respectifs.

Étape 4 : Validation du bon fonctionnement du shell hyperviseur
On vérifiera que la commande "run" permet de lancer effectivement une VM sur le nombre de clusters souhaité et avec le disque souhaité, et que le placement correspond à celui voulu par l'hyperviseur.

Étape 5 : Réalisation du composant MULTI_TTY_VT
Validation du composant MULTI_TTY_VT, de la commande "switch" du shell hyperviseur ainsi que du reroutage des interruption lorsque celle ci est appelée. Cette étape sera validée si la commande "switch" permet

effectivement de basculer l'affichage des 4 terminaux d'une VM à l'autre et que les interruptions sont bien redirigées vers la VM affichée sur les 4 terminaux. La redirection des interruptions nécessitera une reconfiguration de l'IOPIC. Cette étape sera considérée validée lorsque, après avoir basculé l'affichage des terminaux, l'appui d'une touche au clavier sera pris en compte par l'instance effectivement affichée à ce moment là.

Description de l'application "hello" : L'application "hello" d'almos crée un thread sur chacun des processeurs de la VM. Dans lequel chacun des processeurs affiche un message contenant son numéro unique.

Bibliographie

[1] Alain Greiner. Tsar : a scalable, shared memory, many-cores architecture with global cache coherence. In *9th International Forum on Embedded MPSoC and Multicore (MPSoC'09)*, volume 15, 2009.

[2] Ghassan Almaless. *Conception et réalisation d'un système d'exploitation pour des processeurs many-cores*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2014.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5) :164–177, 2003.

[4] Christoffer Dall and Jason Nieh. Kvm/arm : The design and implementation of the linux arm hypervisor. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, pages 333–348. ACM, 2014.

[5] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm : the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.