ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Υποστήριξη ετερογενών αρχιτεκτονικών σε εικονικά περιβάλλοντα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αθανάσιος Κάτσιος

Επιβλέπων: Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Υποστήριξη ετερογενών αρχιτεκτονικών σε εικονικά περιβάλλοντα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αθανάσιος Κάτσιος

Επιβλέπων: Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή επιτροπή την 30η Οκτωβρίου 2017.

.....	.....	.....
Γ. Γκούμας	Ν. Κοζύρης	Δ. Σούντρης
Επίκουρος Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Μάρτιος 2017

.....
Αθανάσιος Κάτσιος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα τελευταία χρόνια, καθώς η εκθετική αύξηση της υπολογιστικής ισχύος συνεχίζεται, ένα εμπόδιο που προκύπτει τόσο για τις φορητές υπολογιστικές συσκευές (smartphones, tablets), όσο και για τους υπερυπολογιστές, είναι αυτό της υπερβολικής κατανάλωσης ενέργειας. Μία από τις μεθόδους που εφαρμόζεται για την μείωση της κατανάλωσης είναι η χρήση ετερογενών υπολογιστικών συστημάτων, δηλαδή υπολογιστών με επεξεργαστές διαφορετικών χαρακτηριστικών, όπως συχνότητα, τάση λειτουργίας. Ως αποτέλεσμα, μπορούν να χρησιμοποιηθούν οι επεξεργαστές υψηλής συχνότητας για τις απαιτητικές εργασίες, και οι επεξεργαστές χαμηλότερης συχνότητας για τις υπόλοιπες, επιτυγχάνοντας έτσι σημαντική εξοικονόμηση ενέργειας, χωρίς να υπάρχει αντίστοιχη μείωση της επίδοσης.

Επομένως, για την αξιοποίηση των ετερογενών αρχιτεκτονικών, θα πρέπει να εισάγουμε την κατάλληλη υποστήριξη στο λειτουργικό σύστημα.

Στο πλαίσιο αυτής της εργασίας μελετάμε τις ετερογενείς αρχιτεκτονικές, και σχεδιάζουμε μία ολοκληρωμένη λύση για την επίτευξη της εικονικοποίησης σε αυτές. Στην συνέχεια, υλοποιούμε αυτό το μοντέλο στην πλατφόρμα εικονικοποιήσης Xen. Έπειτα, δημιουργούμε και χρησιμοποιούμε με επιτυχία ετερογενείς εικονικές μηχανές σε σύστημα ετερογενούς αρχιτεκτονικής ARM big.LITTLE. Για την αξιολόγηση του μοντέλου, εκτελούμε μετροπρογράμματα σε αυτές τις εικονικές μηχανές και παρουσιάζουμε τα αποτελέσματα. Δείχνουμε ότι μπορούμε να προσφέρουμε την δυνατότητα εκτέλεσης ετερογενών εικονικών μηχανών με μηδαμινή επιβάρυνση.

Λέξεις κλειδιά: Εικονικοποίηση, Εικονικές μηχανές, Ετερογενή Υπολογιστικά Συστήματα

Abstract

Nowadays, as computing power is increasing exponentially, power con-sumption is a major issue both for mobile computing devices as well as supercomputers. One of the solutions proposed is the use of heterogeneous architectures, namely computing devices with diverse characteristics such as frequency of operation, voltage etc. As a result, the system can make use of the high-end computing resources for compute-intensive operations, and the low-end resources for less critical operations. This has a direct effect in minizing power consumption without sacrificing performance.

In order to exploit heterogeneity, we need to introduce support in the operating system layer.

In this work, we study heterogeneous architectures and design a full-stack solution for virtualization. We implement this mechanism on the Xen hypervisor and, as a result, we are able to boot heteregenous virtual machines on an ARM big.LITTLE development board. To evaluate our approach, we run micro-benchmarks in these virtual machines and present our results. We demonstrate that we are able to expose heterogeneity to virtual machines with minimal to no overhead.

Keywords: virtualization, virtual Machines, ARM, big.LITTLE, heterogeneous computing

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους καθηγητές κ. Γεώργιο Γκούμα και κ. Νεκτάριο Κοζύρη για τις πολύτιμες γνώσεις και τα εναύσματα αναζήτησης που μου πρόσφεραν στις διαλέξεις τους.

Ευχαριστώ ιδιαίτερα τον διδάκτωρα Αναστάσιο Νάνο για την αμέριστη βοήθεια του στην συγγραφή της διπλωματικής εργασίας.

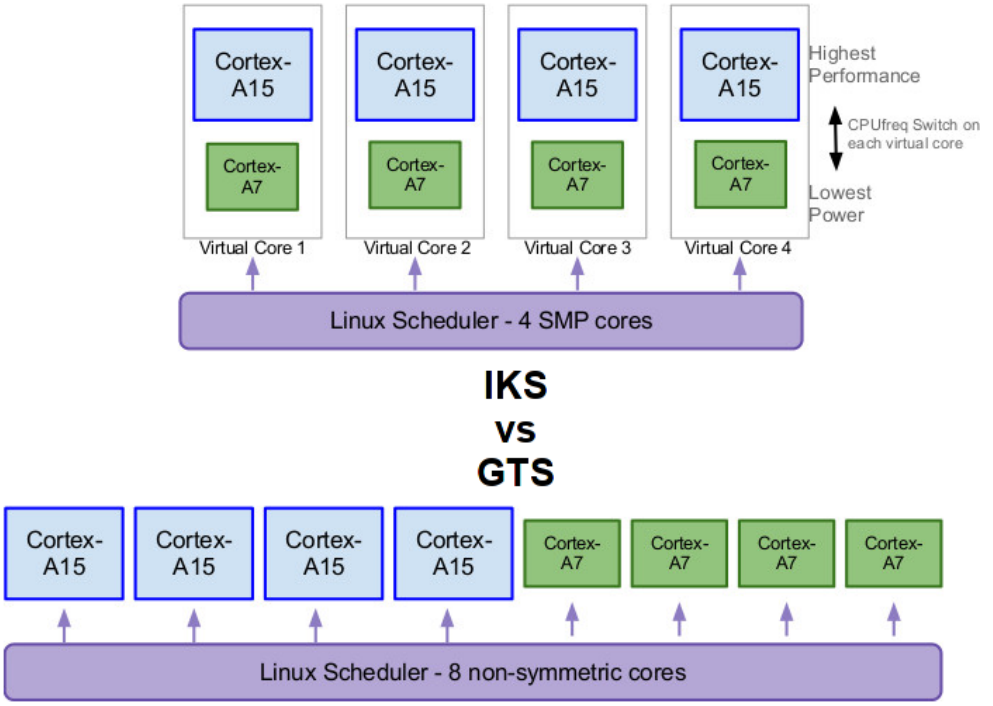
Τέλος, ευχαριστώ την οικογένεια μου για την στήριξη και συμπαράσταση σε όλη την διάρκεια των σπουδών μου.

Περιεχόμενα

1 Εισαγωγή

Το 2011, η ARM, η κύρια κατασκευαστική εταιρεία των επεξεργαστών αρχιτεκτονικής ARM για smartphones και άλλες φορητές συσκευές, ανακοίνωσε την τεχνολογία big.LITTLE, η οποία προσφέρει την δυνατότητα συνδυασμού επεξεργαστικών μονάδων διαφορετικού είδους και χαρακτηριστικών (συχνότητα, τάση λειτουργίας) σε ένα socket [19]. Ως αποτέλεσμα, μία υπολογιστική εργασία μπορεί να τρέξει είτε στους επεξεργαστές τύπου big που διαθέτουν συνήθως μεγαλύτερη συχνότητα λειτουργίας και καλύτερες επιδόσεις, είτε στους επεξεργαστές τύπου LITTLE που διαθέτουν συνήθως μικρότερη συχνότητα λειτουργίας, αλλά πολύ χαμηλότερη ενεργειακή κατανάλωση.

Βέβαια, για την αξιοποίηση της συγκεκριμένης τεχνολογίας, θα πρέπει να υπάρχει υποστήριξη και από τον πυρήνα του λειτουργικού συστήματος, ο οποίος θα πρέπει να είναι σε θέση να αναγνωρίζει την ύπαρξη των διαφορετικών τύπων επεξεργαστών, και να χρησιμοποιεί τον καταλληλότερο για κάθε εργασία: οι εργασίες που απαιτούν μεγάλη υπολογιστική ισχύ θα πρέπει να χρονοδρομολογούνται στους πυρήνες τύπου big που διαθέτουν υψηλή απόδοση, ενώ οι υπόλοιπες θα πρέπει να χρονοδρομολογούνται στους LITTLE, για λόγους εξοικονόμησης ενέργειάς. Στο linux, η υποστήριξη αυτή υλοποιήθηκε από τον μη κερδοσκοπικό οργανισμό Linaro και την ARM, με την υλοποίηση δύο νέων μοντέλων, του In-Kernel Switcher (IKS) και του Global Task Scheduling (GTS). Το πρώτο χωρίζει τις επεξεργαστικές μονάδες σε ζεύγη, όπου κάθε ζεύγος διαθέτει έναν big και έναν LITTLE πυρήνα. Από κάθε ζεύγος το πολύ ένας πυρήνας μπορεί να είναι ενεργός μία χρονική στιγμή, επομένως συνολικά το πολύ οι μισοί πυρήνες μπορούν να είναι ενεργοί. Το δεύτερο μοντέλο, που υλοποιήθηκε αργότερα, υποστηρίζει πλήρως τις ετερογενείς αρχιτεκτονικές. Όλες οι επεξεργαστικές μονάδες μπορούν να λειτουργούν ταυτόχρονα, και το λειτουργικό σύστημα αποφασίζει δυναμικά σε ποιά από αυτές θα τρέξει ένα νήμα, ανάλογα με τον φόρτο εργασίας και το ιστορικό εκτέλεσης του [13].



Σχήμα 1.1: IKS vs GTS

Παρ' όλ' αυτά, σήμερα, 6 χρόνια μετά την ανακοίνωση της big.LITTLE αρχιτεκτονικής, δεν υπάρχει ακόμα η δυνατότητα αξιοποίησής αυτής, καθώς και άλλων ετερογενών αρχιτεκτονικών, στα πλαίσια της εικονικοποίησης. Για παράδειγμα, ο Xen Hypervisor, ο μοναδικός Bare-Metal Hypervisor ανοικτού κώδικα δεν εκτελεί καμία ενέργεια για την διάκριση των ΚΜΕ. Ως αποτέλεσμα, σε big.LITTLE περιβάλλον, όλες οι ΚΜΕ αναγνωρίζονται ως LITTLE, και οι εικονικές μηχανές που εκτελούνται παρουσιάζουν σφάλμα (kernel crash) αν χρονοδρομολογηθούν σε ΚΜΕ που είναι στην πραγματικότητα τύπου big.

Στα πλαίσια της συγκεκριμένης εργασίας σχεδιάζουμε, υλοποιούμε, και παρουσιάζουμε μία ολοκληρωμένη λύση για την επίλυση του παραπάνω προβλήματος στον Xen Hypervisor. Βασικό συστατικό αυτού του μηχανισμού είναι η κατηγοριοποίηση των επεξεργαστικών μονάδων (επιταχυντών ή και συμβατικών επεξεργαστών) σε **κλάσεις επίδοσης** (cpuclass). Τα κριτήρια που χρησιμοποιούμε είναι ο τύπος της επεξεργαστικής μονάδας (in-order, out-of-order, vectorized, κλπ.), ο χρονισμός του ρολογιού (clock frequency) και ενδείξεις ενεργειακής κατανάλωσης (power consumption metrics). Εισάγουμε αυτή τη γνώση στο επίπεδο του λειτουργικού συστή-

ματος (στον hypervisor), και με αυτό τον τρόπο ο έλεγχος διαθεσιμότητας και ο διαμοιρασμός των επεξεργαστικών πόρων γίνεται βέλτιστα από το επίπεδο λογισμικού που χρονοδρομολογεί την πρόσβαση των εφαρμογών στο υλικό. Επιπλέον, επεκτείνουμε τα εργαλεία του Xen Project ώστε να είναι δυνάτος ο έλεγχος από τον χρήστη της νέας λειτουργίας. Τέλος, εκτελούμε και παρουσίαζουμε κάποιες μετρικές, έτσι ώστε να επιβεβαιώσουμε την ορθή λειτουργία του μηχανισμού.

2 Θεωρητικό Υπόβαθρο

2.1 Ετερογενή Συστήματα

2.1.1 Νόμος του Moore

Το 1965, ένας από τους ιδρυτές της εταιρείας κατασκευής μικροεπεξεργαστών Intel, ο Gordon Moore έκανε την πρόβλεψη ότι για τα επόμενα δέκα χρόνια, η πυκνότητα των τρανζίστορ σε ένα ολοκληρωμένο κύκλωμα θα διπλασιάζεται κάθε χρόνο [17]. Το 1975, κοιτάζοντας ξανά τα δεδομένα για την επόμενη δεκαετία, αναθεώρησε την πρόβλεψή του θέτοντας το διάστημα που απαιτείται για τον διπλασιασμό των τρανζίστορ ενός πυκνού ολοκληρωμένου κυκλώματος στα δύο έτη [18]. Η συγκεκριμένη πρόβλεψη επαληθεύθηκε τις επόμενες δεκαετίες, και ονομάστηκε ”Νόμος του Moore”.

Όμως, η συνεχή αύξηση της πυκνότητας των τρανζίστορ, δεν συνεπάγεται και ανάλογη αύξηση της απόδοσης των επεξεργαστών. Το γεγονός αυτό οφείλεται στις παρακάτω αιτίες:

- Ο μεγαλύτερος αριθμός τρανζίστορ συνεπάγεται και αύξηση της παραγόμενης θερμότητας. Έτσι, η χρήση μονοπύρηνων συστημάτων σε υψηλές συχνότητες λειτουργίας καθίσταται αδύνατη [21].

- Ο ρυθμός αύξησης της ταχύτητας των μνημών καθώς και του διαύλου επικοινωνίας με το πέρασμα των χρόνων, είναι πολύ μικρότερος από αυτόν των επεξεργαστών [21]. Επομένως, οι εντολές πρόσβασης στην μνήμη προκαλούν μεγάλη καθυστέρηση στα προγράμματά μας.

Τελικά, οι παραπάνω περιορισμοί οδηγούν στην εξάλειψη των μονοπύρηνων επεξεργαστών, και στην αναζήτηση εναλλακτικών τρόπων επιτάχυνσης των υπολογιστικών συστημάτων.

2.1.2 Πολυπύρηνα Συστήματα

Ένας από αυτούς τους τρόπους είναι η χρήση πολυπύρηνων επεξεργαστών, δηλαδή επεξεργαστών με περισσότερες από μία υπολογιστικές μονάδες ανά τσιπ. Οι επεξεργαστές αυτοί πρωτοχρησιμοποιήθηκαν σε εξυπηρετητές και κέντρα δεδομένων, αλλά πλέον χρησιμοποιούνται ακόμα και σε smartphones.

Τα συστήματα αυτά ταξινομήθηκαν από τον Michael Flynn στις παρακάτω κατηγορίες [12]:

- **Single Instruction Single Data (SISD)**
  Ένα σύστημα που δεν παρουσιάζει κανένα παραλληλισμό.
- **Single Instruction Multiple Data (SIMD)**
  Ένα σύστημα που έχει την δυνατότητα να επεξεργαστεί με τον ίδιο τρόπο ένα μεγάλο όγκο δεδομένων. Χρησιμοποιείται για λειτουργίες όπως την πρόσθεση διανυσμάτων.
- **Multiple Instruction Single Data (MISD)**
  Ένα σύστημα που έχει την δυνατότητα να εκτελέσει διαφορετικούς υπολογισμούς πάνω στα ίδια δεδομένα. Έχει περιορισμένο πεδίο εφαρμογής.
- **Multiple Instruction Multiple Data (MIMD)**
  Δυνατότητα εκτέλεσης διαφορετικών υπολογισμών από διαφορετικές πηγές δεδομένων. Μπορεί να επιτευχθεί με την χρήση σύγχρονων, superscalar CPUs.

Επιπλέον, τα παράλληλα συστήματα θα πρέπει να μοιράζονται ένα μέρος της μνήμης, έτσι ώστε να επιτυγχάνεται η επικοινωνία μεταξύ των επεξεργαστικών μονάδων και της κύριας μνήμης. Έτσι, διακρίνονται ανάλογα με τον τρόπο και την ταχύτητα πρόσβασης στην μοιραζόμενη μνήμη στις δύο παρακάτω κατηγορίες:

- **Uniform Memory Access (UMA)**
  Όλοι οι επεξεργαστές μοιράζονται την μνήμη με τον ίδιο τρόπο και ο χρόνος πρόσβασης σε μία θέση της μνήμης είναι ίδιος, ανεξάρτητα από τον επεξεργαστή που εκτελεί την εντολή.
- **Non Uniform Memory Access (NUMA)**
  Ο κάθε επεξεργαστής έχει ιδιωτική μνήμη, αλλά έχει πρόσβαση και στην μνήμη των άλλων επεξεργαστών. Ο χρόνος πρόσβασης σε κάθε θέση της μνήμης όμως εξαρτάται από τον επεξεργαστή. Το μοντέλο αυτό βοηθάει στην επιτάχυνση των παράλληλων προγραμμάτων, αφού ο κάθε επεξεργαστής χρησιμοποιεί την ιδιωτική του μνήμη για το μεγαλύτερο μέρος των υπολογισμών. Ο δίαυλος επικοινωνίας χρησιμοποιείται μόνο όταν είναι απαραίτητο [16].

2.1.3 Συνεπεξεργαστές

Ένας άλλος τρόπος επιτάχυνσης των υπολογιστών είναι με την χρήση συνεπεξεργαστών (coprocessors). Οι συνεπεξεργαστές είναι εξειδικευμένοι επεξεργαστές που μπορούν να εκτελέσουν ταχύτερα συγκεκριμένες λειτουργίες, όπως επεξεργασία γραφικών ή σημάτων, κρυπτογράφηση κ.λ.π.

Ο κύριος επεξεργαστής μπορεί να αναθέσει μία τέτοια εργασία στον κατάλληλο συνεπεξεργαστή με σκοπό την επιτάχυνση του συστήματος.

2.1.4 Ετερογενή Συστήματα

Ετερογενές ονομάζεται το σύστημα που διαθέτει παραπάνω από ένα είδη επεξεργαστικών μονάδων. Κάθε τέτοια μονάδα μπορεί να εκτελέσει την πιο κατάλληλη γι' αυτήν εργασία, επιταχύνοντας κατά πολύ την εκτέλεση της [21].

Αυτή η ετερογένεια μπορεί να εκφραστεί με διαφορετικούς τρόπους όπως:

- Χρήση επεξεργαστών με διαφορετικά χαρακτηριστικά (όπως συχνότητα, τάση). Ένα τέτοιο παράδειγμα είναι το big.LITTLE της ARM.
- Χρήση κάποιου συνεπεξεργαστή σε συνδυασμό με την κύρια επεξεργαστική μονάδα. Για παράδειγμα, πολλές φορές χρησιμοποιούνται General Purpose GPUs (GPGPUs) για την επιτάχυνση χρονοβόρων, παραλληλοποιήσιμων εργασιών.
- Χρήση FPGAs σε συνδυασμό με την κύρια επεξεργαστική μονάδα.

Στην συγκεκριμένη εργασία εξετάζεται κυρίως το πρότυπο big.LITTLE.

2.1.5 big.LITTLE

Η τεχνολογία big.LITTLE, που σχεδιάστηκε από την ARM, περιλαμβάνει δύο είδη πυρήνων στο ίδιο socket, τους big και τους LITTLE. Οι πρώτοι έχουν υψηλή συχνότητα λειτουργίας και υψηλότερη απόδοση, ενώ οι δεύτεροι έχουν χαμηλότερη συχνότητα και καλύτερη ενεργειακή απόδοση [2]. Κατά τ' άλλα, τα δύο είδη επεξεργαστών έχουν πανομοιότυπη αρχιτεκτονική και σετ εντολών, με αποτέλεσμα το ίδιο πρόγραμμα να μπορεί να εκτελεστεί σε big ή LITTLE πυρήνες χωρίς να χρειάζεται να ξαναγίνει η μεταγλώττιση [2], [14]. Μάλιστα, ο πυρήνας γνωρίζει την ύπαρξη της big.LITTLE αρχιτεκτονικής, και μπορεί να αποφασίσει δυναμικά σε ποιά κλάση είναι καλύτερο να εκτελεστεί μια διεργασία [2].

Η τεχνολογία αυτή μπορεί εξοικονόμηση ενέργειας μέχρι και κατά 75% σε στιγμές χαμηλού φόρτου εργασίας, και βελτίωση της ταχύτητας μέχρι και 40% όταν υπάρχει υψηλός φόρτος [2].

2.2 Εικονικοποίηση (Virtualization)

Εικονικοποίηση ονομάζεται μια μεθοδολογία αναπαράστασης του υλικού ενός υπολογιστή με την χρήση λογισμικού [5], [22]. Παρέχεται έτσι ένα ειδικό περιβάλλον εκτέλεσης, μέσα στο οποίο αποκρύπτεται το φυσικό υλικό του υπολογιστή, και εμφανίζεται μόνο το εξομοιωμένο υλικό [5]. Το περιβάλλον αυτό αποτελεί μια **εικονική μηχανή** (virtual machine). Μέσα σε μια εικονική μηχανή μπορεί να εκτελεστεί ένα λειτουργικό σύστημα ή εφαρμογή, το οποίο είναι ανεξάρτητο από το κύριο λειτουργικό.

Το λογισμικό που παρέχει το εξομοιωμένο υλικό, ονομάζεται **Hypervisor** ή **Virtual Machine Monitor (VMM)**. Οι hypervisors χωρίζονται σε δύο κατηγορίες [20]:

- **Type-1, Native or Bare-Metal Hypervisors**

  Αυτού του είδους οι hypervisors τρέχουν απευθείας στο υλικό του συστήματος, χωρίς να παρεμβάλλεται λειτουργικό σύστημα. Παραδείγματα τέτοιων VMMs είναι ο Xen, ο Oracle VM Server, ο Microsoft Hyper-V, και ο VMware ESX.

- **Type-2, Hosted Hypervisors**

  Αυτού του είδους οι hypervisors τρέχουν πάνω σε κάποιο λειτουργικό σύστημα, όπως τα κανονικά προγράμματα. Παραδείγματα τέτοιων VMMs είναι ο VMWare Workstation, ο Oracle Virtualbox, και ο QEMU.

Στην βιβλιογραφία, καθώς και στην παρούσα εργασία, το φυσικό μηχάνημα ή λειτουργικό σύστημα αποκαλείται **host**, ενώ τα εικονικά μηχανήματα αποκαλούνται **guests**.

2.2.1 Γιατί;

Η εικονικοποίηση προσφέρει πολλαπλά ωφέλη όπως:

- Καλύτερη αξιοποίηση και διαμοιρασμός των πόρων ενός μηχανήματος. Σε έναν υπολογιστή μπορούν να τρέχουν ταυτόχρονα πολλαπλές εικονικές μηχανές, οι οποίες προσφέρουν πολλά ανεξάρτητα λειτουργικά συστήματα για πολλούς χρήστες. Έτσι, για παράδειγμα στην παροχή μία υπηρεσίας cloud, μπορούμε να έχουμε μόνο ένα φυσικό μηχάνημα, το οποίο παρέχει N εικονικές μηχανές στους N χρήστες του, αντί για N μηχανήματα [22].

- Δυνατότητα χρήσης μιας εικονικής μηχανής ως απομονωμένο περιβάλλον εκτέλεσης (sandbox) για μία μη αξιόπιστη εφαρμογή ή έναν μη αξιόπιστο χρήστη [22].

- Δυνατότητα προσομοίωσης υλικού που δεν διαθέτουμε στην πραγματικότητα, όπως ενός επεξεργαστή διαφορετικής αρχιτεκτονικής από αυτόν του host μηχανήματος. Επιπλέον, παρέχεται και η δυνατότητα εξομοίωσης δικτύων και διαφορετικών δικτυακών τοπολογιών.
- Δυνατότητα εύκολου περιορισμού των πόρων που χρησιμοποιεί μια εφαρμογή.
- Εύκολη εγκατάσταση ενός λειτουργικού συστήματος ή άλλου περιβάλλοντος σε μια εικονική μηχανή, χωρίς να υπάρχει ανάγκη διαγραφής του υπάρχοντος λειτουργικού συστήματος.
- Δυνατότητα εύκολης αποθήκευσης, μεταφοράς, και επαναφοράς της κατάστασης ενός λειτουργικού συστήματος, με την χρήση εικονικών δίσκων [22].

2.2.2 Εικονικοποίηση και επιδόσεις

Φυσικά, η πλήρης προσομοίωση του υλικού ενός υπολογιστή από τον hypervisor είναι αρκετά πολύπλοκη, και έτσι το εικονικό υλικό που προκύπτει μπορεί να είναι μέχρι και εκατοντάδες φορές πιο αργό από το φυσικό υλικό. Γι' αυτόν τον λόγο έχουν αναπτυχθεί τεχνολογίες που βοηθούν στην επιτάχυνση των εικονικών μηχανών. Αυτές είναι:

- **Binary Translation**

  Οι εντολές των εφαρμογών ξαναγράφονται από τον hypervisor (μεταφρασμένες στο σύνολο εντολών του host), και εισάγονται traps πριν από διάφορα προβληματικά σημεία. Η διαδικασία αυτή μπορεί να γίνει στατικά ή δυναμικά. Η δυναμική μετάφραση χρησιμοποιείται στην just-in-time compilation (JIT), όπως για παράδειγμα σε μία Java Virtual Machine (JVM) [11].

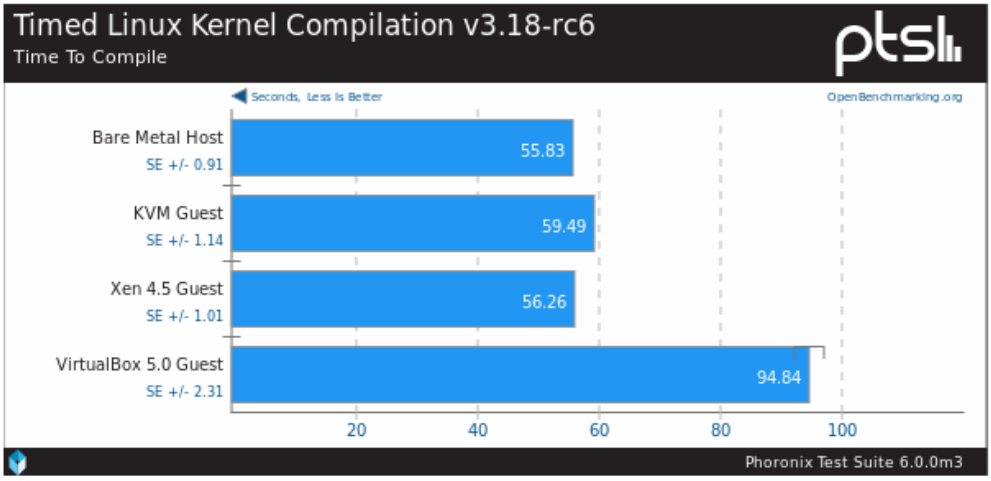
- **Hardware Assisted Virtualization**

  Οι σύγχρονοι επεξεργαστές διαθέτουν χαρακτηριστικά που υποβοηθούν την υλοποίηση ενός VMM, τα οποία ονομάζονται virtualization extensions. Τα virtualization extensions προσφέρουν διαφορετικά επίπεδα εκτέλεσης για τον hypervisor και τις εικονικές μηχανές. Έτσι, ο hypervisor μπορεί να τρέχει με περισσότερα προνόμια, ενώ οι εικονικές μηχανές δεν μπορούν να επηρεάσουν το λειτουργικό σύστημα του host [11]. Επιπρόσθετα, επιτρέπεται το ταχύτατο context switch από την εικονική μηχανή στον hyervisor και αντίστροφα.

- **Paravirtualization**

  Με τον όρο paravirtualization, υπονοείται η διαδικασία τροποποίησης του πυρήνα του guest λειτουργικού συστήματος, έτσι ώστε να αντικατασταθούν διάφορες κλήσεις συστήματος, οι οποίες δεν μπορούν να προσομοιωθούν εύκολα [23]. Όταν γίνεται μια τέτοια κλήση συστήματος, ο τροποποιημένος πυρήνας επικοινωνεί με τον hypervisor, και πραγματοποιεί την κλήση ταχύτατα. Η συγκεκριμένη τεχνική χρησιμοποιείται για την προσομοίωση συσκευών εισόδου/εξόδου, όπως καρτών δικτύου.

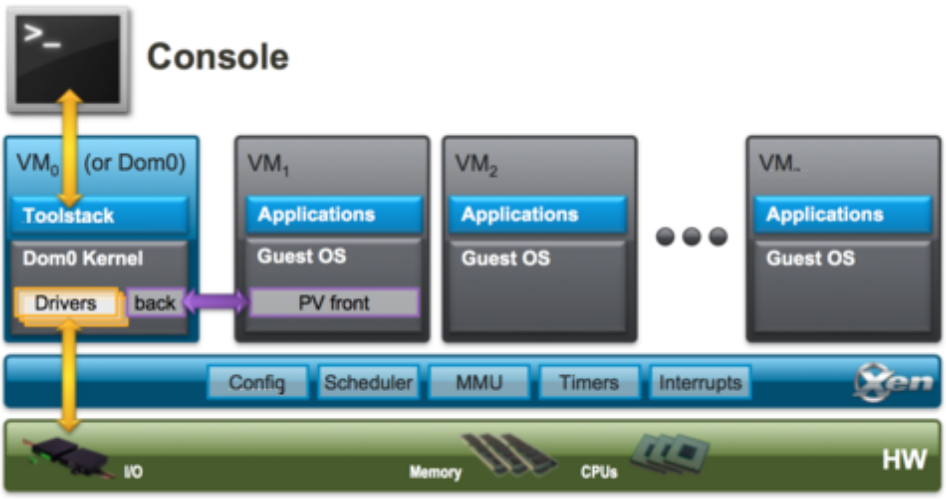
Τελικά, χάρη στις παραπάνω τεχνολογίες, οι επιδόσεις των σύγχρονων εικονικών μηχανών πλησιάζουν αυτές του host συστήματος. Ενδεικτικά, παρουσιάζεται ο χρόνος μεταγλώττισης του πυρήνα του Linux σε 3 δημοφιλή VMMs ανοικτού κώδικα:



Σχήμα 2.1: Χρόνος μεταγλώττισης Linux Kernel v3.18 [15]

2.2.3 Xen Project

Στην παρούσα εργασία, χρησιμοποιείται ο Xen Project Hypervisor, ένας δημοφιλής ανοικτού κώδικα hypervisor τύπου 1. Παρακάτω παρουσιάζεται η αρχιτεκτονική του Xen:



Σχήμα 2.2: Αρχιτεκτονική του Xen [9]

Επεξήγηση των συστατικών του Xen

- **Xen Project Hypervisor**

  Το κύριο λογισμικό του hypervisor, που τρέχει στο υλικό του υπολογιστή και είναι υπεύθυνο για τον έλεγχο της CPU, της μνήμης και των διακοπών, αλλά δεν έχει επίγνωση των συσκευών εισόδου/εξόδου. Είναι το αρχικό πρόγραμμα, το οποίο ξεκινάει από τον bootloader.

- **Guest Domains - Εικονικές Μηχανές**

  Τα εικονικά περιβάλλοντα, καθένα από τα οποία είναι ανεξάρτητο και τρέχει το δικό του λειτουργικό σύστημα. Μπορούν να είναι paravirtualized ή fully virtualized. Οι εικονικές μηχανές δεν έχουν δικαίωμα πρόσβασης στο υλικό του υπολογιστή, γι' αυτό ονομάζονται unprivileged domain (DomU).

- **Domain 0**

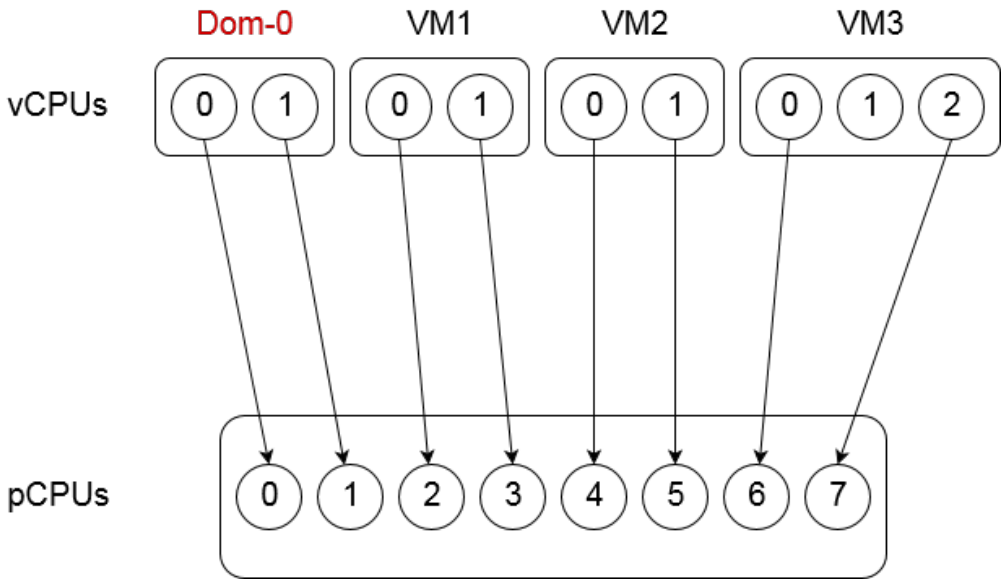
  Μία ειδική εικονική μηχανή, η οποία ξεκινάει αυτόματα από τον Xen Hypervisor. Έχει δικαίωμα πρόσβασης και ελέγχου του υλικού, και χειρίζεται τις συσκευές εισόδου/εξόδου. Επιπλέον, διαχειρίζεται και τις υπόλοιπες εικονικές μηχανές. Το domain 0 τρέχει συνήθως λειτουργικό σύστημα Linux, με ειδικές επεκτάσεις στον πυρήνα, που προσφέρουν την δυνατότητα paravirtualization.

- **Toolstack - Κονσόλα**

  Μία σειρά εργαλείων, που μπορούν να τρέξουν στο domain 0 και αναλαμβάνουν την δημιουργία, έλεγχο, και καταστροφή των εικονικών μηχανών [9].

Περισσότερα για τον Scheduler του Xen

Κάθε εικονική μηχανή διαθέτει έναν αριθμό εικονικών ΚΜΕ (Virtual CPU, vCPU). Ο αριθμός αυτός καθορίζεται από το αρχείο ελέγχου της εικονικής μηχανής όπως θα δούμε στην επόμενη παράγραφο. Οι εικονικές ΚΜΕ αρχικοποιούνται κατά την εκκίνηση της αντίστοιχης εικονικής μηχανής. Για να μπορέσει όμως να λειτουργήσει μία εικονική ΚΜΕ, θα πρέπει να αντιστοιχιστεί σε μία από τις φυσικές ΚΜΕ (Physical CPU, pCPU) του μηχανήματος. Εάν ο αριθμός των ενεργών εικονικών ΚΜΕ είναι μεγαλύτερος από αυτόν των φυσικών ΚΜΕ, τότε η αντιστοίχηση αυτή θα πρέπει να αλλάζει δυναμικά, έτσι ώστε να υπάρχει δικαιοσύνη μεταξύ του φόρτου εργασίας των διαφορετικών εικονικών μηχανών. Η εργασία αυτή, δηλαδή η δυναμική αντιστοίχιση των εικονικών ΚΜΕ σε φυσικές αναλαμβάνεται από τον scheduler του Xen Hypervisor [8]. Παρακάτω βλέπουμε ένα παράδειγμα τέτοιας αντιστοίχησης. Κάθε εικονική ΚΜΕ έχει αντιστοιχιστεί σε μία φυσική ΚΜΕ, εκτός από την εικονική ΚΜΕ 1 της VM3, η οποία, στο συγκεκριμένο στιγμιότυπο κοιμάται.



Σχήμα 2.3: Εικονικές και φυσικές ΚΜΕ

Ο Xen υποστηρίζει πολλαπλούς schedulers με διαφορετικά χαρακτηριστικά. Στον Xen 4.9 υπάρχουν 4 διφορετικοί schedulers, ο Credit, ο Credit2, ο RTDS, και ο ARINC653. Στην συγκεκριμένη εργασία χρησιμοποιούμε τον προεπιλεγμένο, Credit Scheduler. Ο Credit Scheduler είναι ένας δίκαιος χρονοδρομολογητής με βάρη. Λειτουργεί με κβάντα χρόνου, καθένα από τα οποία διαρκεί 30ms. Διαθέτει δύο χαρακτηριστικά για κάθε εικονική μηχανή, το weight και το cap. Η μεταβλητή weight ορίζει τον χρόνο των KME που θα καταλάβει μία εικονική μηχανή, σχετικά με τις υπόλοιπες. Έτσι, μία εικονική μηχανή με weight 256 θα τρέξει τον μισό χρόνο από μία εικονική μηχανή με weight 512. Το Cap ορίζει το μέγιστο μέρος των KME που μπορεί να διαθέτει μια εικονική μηχανή, σε ποσοστό των φυσικών KME. Έτσι, 100 σημαίνει 1 εικονική KME, 50 σημαίνει μισή εικονική KME κ.λ.π. Η προεπιλεγμένη τιμή 0 υπονοεί ότι δεν υπάρχει όριο [6].

Έπιπλέον, υποστηρίζεται και «κάρφωμα» (pinning) των εικονικών KME σε ένα υποσύνολο των φυσικών, χρησιμοποιώντας τις ιδιότητες των KME hard_affinity και soft_affinity. Η hard affinity ορίζει απευθείας τις επιτρεπόμενες φυσικές KME για μία εικονική KME, ενώ η soft affinity ορίζει τις επιθυμητές. Όλες αυτές οι ιδιότητες ελέγχονται από το toolstack, όπως θα δούμε στην επόμενη παράγραφο.

Περισσότερα για το toolstack

Το κύριο εργαλείο ελέγχου του Xen Hypervisor από τον χώρο χρήστη είναι το εκτελέσιμο xl. Το εργαλείο αυτό είναι διαθέσιμο στο Domain 0, και χρησιμεύει για την άντληση πληροφοριών του συστήματος, για τον έλεγχο του Hypervisor, για την δημιουργία, έλεγχο και καταστροφή των εικονικών μηχανών [10]. Οι βασικές εντολές του xl, οι οποίες χρησιμοποιούνται στα πλαίσια αυτής της εργασίας παρουσιάζονται παρακάτω:

- **create <configfile>**

  Δημιουργεί μια νέα εικονική μηχανή. Δέχεται ως παράμετρο ένα config αρχείο το οποίο καθορίζει τις ιδιότητές της. Ένα παράδειγμα τέτοιου αρχείου βλέπουμε παρακάτω.

simple.cfg

```
name = vm1
kernel = "/boot/linux-2.6-xen"
vcpus = 2
memory = 512
disk = [ 'phy:/root/domU.img,xvda,w' ]
root = "/dev/sda ro"
```

- **console <domain-id>**

  Συνδέει την τρέχουσα κονσόλα με την κονσόλα την εικονική μηχανή που καθορίζεται από το domain-id.

- **destroy <domain-id>**

  Τερματίζει στιγμιαία την εικονική μηχανή που καθορίζεται από το domain-id.

- **shutdown <domain-id>**

  Στέλνει σήμα τερματισμού στην εικονική μηχανή που καθορίζεται από το domain-id.

- **list**

  Εμφανίζει πληροφορίες για όλες τις ενεργές εικονικές μηχανές - domains. Παρακάτω βλέπουμε ένα παράδειγμα εκτέλεσης της εντολής.

xl list

```
# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0   750       4     r-----  11794.3
win                                          1  1019       1     r-----      0.3
linux                                        2  2048       2     r-----   5624.2
```

- **vcpu-list**

  Εμφανίζει πληροφορίες για όλες τις εικονικές ΚΜΕ, ταξινομημένες ανά domain. Παρακάτω βλέπουμε ένα παράδειγμα εκτέλεσης της εντολής.

xl vcpu-list

```
# xl vcpu-list
Name                                ID  VCPU   CPU State   Time(s)  Affinity (Hard
    / Soft)
Domain-0                             0     0     0   r--     554.1     0 / all
domU-ananos                          2     0     2   -b-      85.4   0-3 / all
domU-ananos                          2     1     1   -b-      48.9   0-3 / all
domU-ananos                          2     2     3   -b-      24.4   0-3 / all
domU-ananos                          2     3     1   -b-      47.7   0-3 / all
domU-ananos                          2     4     7   -b-      14.1   4-7 / all
domU-ananos                          2     5     4   -b-      12.9   4-7 / all
domU-ananos                          2     6     5   -b-      13.3   4-7 / all
domU-ananos                          2     7     6   -b-       8.7   4-7 / all
```

- **info**

  Εμφανίζει γενικές πληροφορίες του συστήματος και του Hypervisor. Αν χρησιμοποιηθεί και η παράμετρος -n, τότε εμφανίζονται και επιπλέον πληροφορίες για την τοπολογία NUMA του μηχανήματος. Παρακάτω βλέπουμε ένα παράδειγμα εκτέλεσης της εντολής.

xl info

```
# xl info
host                   : scarlett
release                : 3.1.0-rc4+
version                : #1001 SMP Wed Oct 19 11:09:54 UTC
    2011
machine                : x86_64
nr_cpus                : 4
nr_nodes               : 1
cores_per_socket       : 4
threads_per_core       : 1
cpu_mhz                : 2266
hw_caps                : bfebfbff:28100800:00000000:00003
    b40:009ce3bd:00000000:00000001:00000000
virt_caps              : hvm hvm_directio
total_memory           : 6141
free_memory            : 4274
free_cpus              : 0
outstanding_claims     : 0
xen_major              : 4
xen_minor              : 2
xen_extra              : -unstable
xen_caps               : xen-3.0-x86_64 xen-3.0-x86_32p hvm
    -3.0-x86_32 hvm-3.0-x86_32p hvm-3.0-x86_64
xen_scheduler          : credit
xen_pagesize           : 4096
platform_params        : virt_start=0xffff800000000000
xen_changeset          : Wed Nov 02 17:09:09 2011 +0000
    24066:54a5e994a241
xen_commandline        : com1=115200,8n1 guest_loglvl=all
    dom0_mem=750M console=com1
cc_compiler            : gcc version 4.4.5 (Debian 4.4.5-8)
cc_compile_by          : sstabellini
cc_compile_domain      : uk.xensource.com
cc_compile_date        : Tue Nov  8 12:03:05 UTC 2011
xend_config_format     : 4
```

3 Σχεδίαση και Υλοποίηση

Για τον σχεδιασμό της εργασίας ακολουθήθηκε το RFC Heterogeneous Multi Processing Support in Xen [7], από τον Dario Faggioli, που είναι προγραμματιστής στο Xen Project. Στην συνέχεια του κεφαλαίου παρουσιάζεται το περιεχόμενο αυτού του RFC, και πως αυτό υλοποιήθηκε στην πράξη.

Το συγκεκριμένο κεφάλαιο χωρίζεται σε 4 παραγράφους ανάλογα με την λειτουργία του κώδικα:

- Το κύριο μέρος (backend), στο οποίο υλοποιείται η δομή και λειτουργία των κλάσεων.
- Το μέρος του backend το οποίο είναι διαφορετικό για τις δύο υποστηριζόμενες αρχιτεκτονικές, ARM και x86.
- Το μέρος στο οποίο υλοποιούνται οι υπερκλήσεις, με τις οποίες επιτυγχάνεται η επικοινωνία του Xen (backend) με τα εργαλεία (frontend).
- Το μέρος όπου περιγράφονται αυτά τα εργαλεία.

3.1 Γενική Περιγραφή – Σχεδίαση

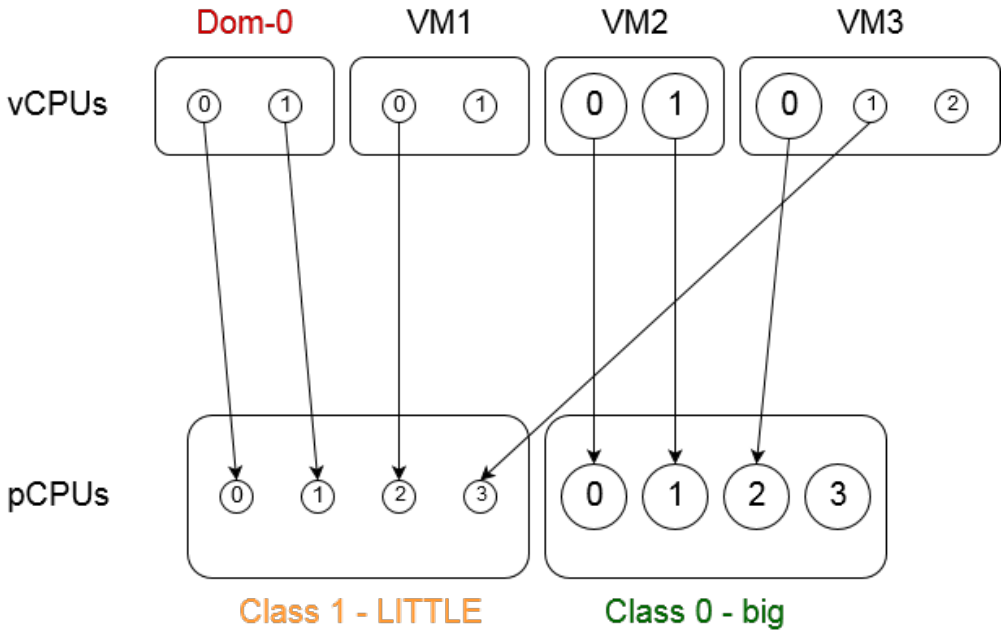
Το πρώτο βήμα για τον σχεδιασμό της παρούσας εργασίας είναι ο ορισμός μίας νέας για τον Xen έννοιας, της κλάσης ΚΜΕ (cpu_class). Μια cpu_class ορίζεται από τα παρακάτω χαρακτηριστικά:

1. Κάθε φυσική ΚΜΕ ανήκει σε μία κλάση.
2. Κάθε κλάση θα πρέπει να έχει τουλάχιστον μία φυσική ΚΜΕ.
3. Κάθε φυσική ΚΜΕ μπορεί να ανήκει μόνο σε μία κλάση.
4. Οι ΚΜΕ που ανήκουν στην ίδια κλάση είναι αρκετά ομογενείς, ώστε μία εικονική ΚΜΕ μπορεί να μεταβεί από μία φυσική ΚΜΕ της κλάσης σε μία άλλη χωρίς επιπλοκές.
5. Όταν μία εικονική ΚΜΕ συσχετιστεί με μία ή περισσότερες κλάσεις, τότε μπορεί να δρομολογηθεί σε όλες τις φυσικές ΚΜΕ που ανήκουν σε αυτές τις κλάσεις.
6. Μπορούμε να αναφερθούμε στις κλάσεις χρησιμοποιώντας τον αναγνωριστικό τους αριθμό (π.χ. κλάση 0, κλάση 1), αλλά και με πιο φιλικές ετικέτες, π.χ. big, LITTLE.

Για παράδειγμα, σε μία υποθετική αρχιτεκτονική AB, οι φυσικές ΚΜΕ 0, 2 ανήκουν στην κλάση A, ενώ οι ΚΜΕ 1, 3 ανήκουν στην κλάση B. Έτσι, αν μία εικονική ΚΜΕ συσχετιστεί με την κλάση A, τότε μπορεί να δρομολογηθεί στις φυσικές ΚΜΕ 0, 2, αλλά ποτέ στις 1, 3.

Το πως ακριβώς ορίζεται μια κλάση, δηλαδή ποια είναι τα χαρακτηριστικά της και ποιες φυσικές ΚΜΕ ανήκουν σε αυτήν, εξαρτάται από την αρχιτεκτονική. Για την αρχιτεκτονική ARM big.LITTLE οι κλάσεις αναγνωρίζονται αυτόματα. Οι επεξέργαστες που διαθέτουν τα ίδια χαρακτηριστικά (αριθμός μοντέλου, συχνότητα λειτουργίας, τάση λειτουργίας) θα πρέπει να τοποθετούνται στην ίδια κλάση. Αρκετά εύκολα θα μπορούσε να υλοποιηθεί παρόμοια λειτουργία και για άλλες αρχιτεκτονικές, όπως την x86.

Παρακάτω βλέπουμε μία παραλλαγή του σχήματος 2.3 για αρχιτεκτονική ARM με κλάσεις big, LITTLE. Με μεγάλο κύκλο συμβολίζονται οι big φυσικές και εικονικές ΚΜΕ, ενώ με μικρό κύκλο οι LITTLE. Το Domain 0 και το VM1 είναι τύπου LITTLE, το VM2 είναι τύπου big, ενώ το VM3 είναι μεικτού τύπου, δηλαδή διαθέτει 2 LITTLE πυρήνες και έναν big. Όπως βλέπουμε, χάρη στην λειτουργία των κλάσεων, 4 LITTLE εικονικές ΚΜΕ αντιστοιχίζονται σε LITTLE φυσικές ΚΜΕ, ενώ 2 εικονικές ΚΜΕ κοιμούνται. Ταυτόχρονα, οι 3 big εικονικές ΚΜΕ αντιστοιχίζονται σε big φυσικές ΚΜΕ, ενώ 1 φυσική ΚΜΕ κοιμάται. Οι LITTLE εικονικές ΚΜΕ δεν επιτρέπεται να χρονοδρομολογηθούν σε big εικονικές ΚΜΕ, ή αντίστροφα.



Σχήμα 3.1: Εικονικές και φυσικές ΚΜΕ σε big.LITTLE

Ταυτόχρονα, θα πρέπει να δίνεται στον χρήστη η δυνατότητα παρακολούθησης και ελέγχου της νέας λειτουργίας. Για να επιτευχθεί αυτό, τροποποιούνται οι εντολές *xl info* και *xl vcpu-list*, έτσι ώστε να τυπώνουν πληροφορίες και για τις κλάσεις των φυσικών ΚΜΕ και εικονικών ΚΜΕ αντίστοιχα. Επιπλέον, υλοποιήθηκε η νέα εντολή *xl vcpu-class <Domain>< VCPU|all><Class affinity|all>*, η οποία επιτρέπει τον ορισμό των κλάσεων για τις εικονικές ΚΜΕ ενός Domain. Τέλος, ο ορισμός των κλάσεων μπορεί να γίνει και στο config αρχείο της εικονικής μηχανής, χρησιμοποιώντας την νέα ιδιότητα vcpuclass.

3.2 Το κύριο μέρος

Στην συγκεκριμένη υποενότητα θα αναλύσουμε το κύριο μέρος της υλοποίησης. Ο πηγαίος κώδικας αυτής της παραγράφου περιέχεται στους φάκελους /xen/include/xen (αρχεία *.h) και /xen/common (αρχεία *.c) του Xen Project, και επηρεάζει την λειτουργία του για όλες τις αρχιτεκτονικές.

Άρχικά παρουσιάζεται η δήλωση και η υλοποίηση της δομής cpu_class, της οποίας ο ορισμός αναφέρθηκε στην προηγούμενη υποενότητα:

cpu_class.h

```
#ifndef __XEN_CPU_CLASS_H__
#define __XEN_CPU_CLASS_H__

#include <xen/cpumask.h>

extern uint16_t NR_CLASSES __read_mostly;
extern uint16_t *cpu_to_class __read_mostly;
extern cpumask_t *class_to_cpumask __read_mostly;


int cpu_class_init(void);
int cpu_class_classify_cpus(int cpus);
void cpu_class_set_all_classes(cpumask_t *dstp);
void cpumask_to_cpu_classes(cpumask_t *classes, const cpumask_t
    *cpus);
void cpu_classes_to_cpumask(cpumask_t *cpus, const cpumask_t *
    classes);

#endif
```

cpu_class.c

```
#include <xen/cpu_class.h>

uint16_t NR_CLASSES __read_mostly;
```

```
uint16_t *cpu_to_class __read_mostly;
cpumask_t *class_to_cpumask __read_mostly;

void cpu_class_set_all_classes(cpumask_t *dstp)
{
    int cpu;
    for (cpu = 0; cpu < NR_CLASSES; cpu++)
        cpumask_set_cpu(cpu, dstp);
}

void cpumask_to_cpu_classes(cpumask_t *classes, const cpumask_t
   *cpus)
{
    int cpu;

    cpumask_clear(classes);
    for_each_cpu(cpu, cpus)
        cpumask_set_cpu(cpu_to_class[cpu], classes);
}

void cpu_classes_to_cpumask(cpumask_t *cpus, const cpumask_t *
   classes)
{
    int cpu_class;

    cpumask_clear(cpus);
    for_each_cpu(cpu_class, classes)
        cpumask_or(cpus, cpus, &class_to_cpumask[cpu_class]);
}
```

Όπως βλέπουμε παραπάνω, οι κλάσεις ορίζονται με την μορφή 2 πινάκων *uint16_t *cpu_to_class* και *cpumask_t *class_to_cpumask*, οι οποίοι περιέχουν την αντιστοίχιση μεταξύ κλάσεων και φυσικών ΚΜΕ. Οι πίνακες αυτοί δημιουργούνται δυναμικά από την συνάρτηση *cpu_class_init*, και αρχικοποιούνται από την συνάρτηση *cpu_class_classify_cpus* οι οποίες είναι διαφορετικές για κάθε αρχιτεκτονική, και θα παρουσιαστούν σε επόμενη υποενότητα.

Επιπλέον, στην δομή της εικονικής ΚΜΕ προστέθηκε ένα επιπλέον πεδίο *classes* τύπου cpumask, στο οποίο αποθηκεύονται οι κλάσεις στις οποίες επιτρέπεται να τρέξει η εικονική ΚΜΕ, όπως φαίνεται παρακάτω.

sched_part.h

```
struct vcpu
{
    int              vcpu_id;
    int              processor;
    vcpu_info_t     *vcpu_info;
```

```
    /* ... */

    /* Bitmask of CPUs on which this VCPU may run. */
    cpumask_var_t    cpu_hard_affinity;
    /* Used to change affinity temporarily. */
    cpumask_var_t    cpu_hard_affinity_tmp;
    /* Used to restore affinity across S3. */
    cpumask_var_t    cpu_hard_affinity_saved;
    /* Bitmask of CPUs on which this VCPU prefers to run. */
    cpumask_var_t    cpu_soft_affinity;
    /* Bitmask of CPUs which are holding onto this VCPU's state.
     */
    cpumask_var_t    vcpu_dirty_cpumask;
    /* Bitmask of classes where the vcpu can run */
    cpumask_var_t classes;

    /* ... */
};
```

Εσωτερικά, για την τήρηση των περιορισμών που ορίζουν οι κλάσεις, χρησιμοποιείται η ιδιότητα hard affinity, η οποία περιορίζει τις φυσικές ΚΜΕ στις οποίες μπορεί να τρέξει μια εικονική ΚΜΕ. Έτσι, όταν αλλάζει η τιμή hard affinity για μια εικονική ΚΜΕ, ελέγχουμε ότι τηρούνται και οι περιορισμοί των κλάσεων, όπως φαίνεται παρακάτω.

schedule_part.c

```
/* ... */

static inline int is_affinity_class_compatible(struct vcpu *v,
    const cpumask_t *affinity)
{
    cpumask_t classes, tmp;
    cpumask_to_cpu_classes(&classes, affinity);
    cpumask_or(&tmp, &classes, v->classes);
    return cpumask_equal(&tmp, v->classes);
}

int vcpu_set_hard_affinity(struct vcpu *v, const cpumask_t *
    affinity)
{
    cpumask_t online_affinity;
    cpumask_t *online;

    if ( v->domain->is_pinned )
        return -EINVAL;

    online = VCPU2ONLINE(v);
```

```
    cpumask_and(&online_affinity, affinity, online);
    if ( cpumask_empty(&online_affinity) )
        return -EINVAL;

    if ( !is_affinity_class_compatible(v, &online_affinity) )
        return -EINVAL;

    return vcpu_set_affinity(v, affinity, v->cpu_hard_affinity);
}

/* ... */
```

Επιπροσθέτως, καθε εικονική ΚΜΕ που δημιουργείται θα πρέπει να ανήκει σε κάποιες κλάσεις. Στα πλαίσια αυτής της εργασίας αποφασίστηκε να αρχικοποιείται σε όλες τις διαθέσιμες κλάσεις, αφού δίνεται η δυνατότητα αλλαγής των κλάσεων αργότερα, από το toolkit. Η υλοποίηση της λειτουργίας φαίνεται παρακάτω.

domain_part.c

```
struct vcpu *alloc_vcpu(
    struct domain *d, unsigned int vcpu_id, unsigned int cpu_id)
{
    struct vcpu *v;

    BUG_ON((!is_idle_domain(d) || vcpu_id) && d->vcpu[vcpu_id]);

    if ( (v = alloc_vcpu_struct()) == NULL )
        return NULL;

    /* ... */

    if ( !zalloc_cpumask_var(&v->cpu_hard_affinity) ||
         !zalloc_cpumask_var(&v->cpu_hard_affinity_tmp) ||
         !zalloc_cpumask_var(&v->cpu_hard_affinity_saved) ||
         !zalloc_cpumask_var(&v->cpu_soft_affinity) ||
         !zalloc_cpumask_var(&v->classes) ||
         !zalloc_cpumask_var(&v->vcpu_dirty_cpumask) )
        goto fail_free;

    cpu_class_set_all_classes(v->classes);

    /* ... */
}
```

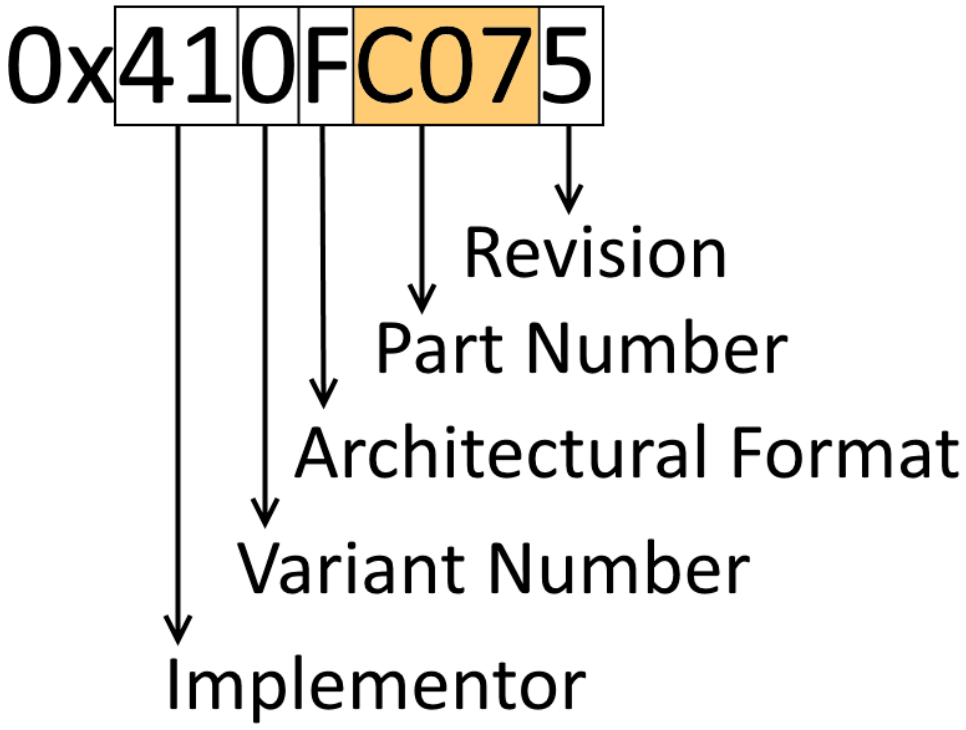
3.3 Architecture Specific

Σε αυτήν την υποενότητα αναλύεται το μέρος της υλοποίησης που είναι διαφορετικό για κάθε μία από τις δύο υποστηριζόμενες αρχιτεκτονικές, arm και x86. Το μέρος του κώδικα που χτίζεται μόνο σε arm περιέχεται στον φάκελο /xen/arch/arm του πηγαίου κώδικα του Xen Project, ενώ το μέρος του κώδικα που χτίζεται μόνο σε x86 περιέχεται στον φάκελο /xen/arch/x86.

Αρχικά, ας αναλύσουμε τον τρόπο με τον οποίο επιτυγχάνεται ο διαχωρισμός των KME στις δύο κλάσεις big και LITTLE, για την αρχιτεκτονική arm. Πρώτα, ας τονίσουμε ότι οι πυρήνες big και LITTLE διαθέτουν πανομοιότυπη αρχιτεκτονική, και ως αποτέλεσμα, δεν είναι δυνατόν να αναγνωριστεί αν ένας μεμονωμένος πυρήνας είναι τύπου big ή LITTLE. Για παράδειγμα, το μοντέλο Samsung Exynos 7 5433 διαθέτει 4 Cortex-A53 πυρήνες ως LITTLE, ενώ το Samsung Exynos 7 7580 διαθέτει 4 Cortex-A53 πυρήνες ως big. Επομένως, ο μόνος τρόπος διαχωρισμού των πυρήνων, είναι εαν εξεταστούν συλλογικά. Αυτό μπορεί να επιτευχθεί με το διάβασμα του ειδικού καταχωρητή της αρχιτεκτονικής ARM, MIDR. Ο MIDR περιέχει κάποια πεδία που παρέχουν χρήσιμες πληροφορίες για τον εξεταζόμενο επεξεργαστή / πυρήνα. Τα πεδία αυτά παρουσιάζονται στον πίνακα 3.1, καθώς και στο σχήμα 3.2.

[31:24]	Implementor
[23:20]	Variant Number
[19:16]	Architectural format description
[15:4]	Part number
[3:0]	Revision

Πίνακας 3.1: Τα πεδία του MIDR [3]



Σχήμα 3.2: Η τιμή του MIDR για το Cortex A7 Revision r0p5

Το πεδίο που μας ενδιαφέρει είναι το Part number, το οποίο είναι διαφορετικό για κάθε τύπο core. Για παράδειγμα, για τους Cortex-A7 είναι 0xC07 [4], ενώ για τους A15 είναι 0xC0F [1]. Χρησιμοποιώντας το πεδίο αυτό, μπορούμε να διαχωρίσουμε τα cores ανάλογα με το μοντέλο τους. Τα cores με το μεγαλύτερο part number κατηγοριοποιούνται στην κλάση 0 (big), ενώ τα υπόλοιπα στην 1 (LITTLE). Παρακάτω φαίνεται η υλοποίηση αυτού του τρόπου διαχωρισμού, με την δημιουργία και αρχικοποίηση των δύο πινάκων που αναφέρθηκαν στο προηγούμενο κεφάλαιο, *uint16_t *cpu_to_class* και *cpumask_t *class_to_cpumask*.

cpu_class.c

```
#include <xen/cpu_class.h>
#include <xen/xmalloc.h>
#include <xen/errno.h>
#include <asm/processor.h>

/*
 * Finds the greatest part_number in all the processors
 * This part_number will be considered the 'big' one,
 * and all big processors should have the same
 */
```

```
static unsigned long find_big_part_number(int cpus)
{
    int cpu;
    unsigned long big = 0;
    for (cpu = 0; cpu < cpus; cpu++)
    {
        unsigned long part_number = cpu_data[cpu].midr.
   part_number;
        if (part_number > big)
            big = part_number;
    }
    return big;
}

int cpu_class_init(void)
{
    /* Initialize global variables */
    NR_CLASSES = 2;

    if ( (cpu_to_class = xzalloc_array(uint16_t, NR_CPUS)) ==
   NULL )
        return -ENOMEM;
    if ( !zalloc_cpumask_arr(&class_to_cpumask, NR_CLASSES) )
    {
        xfree(cpu_to_class);
        return -ENOMEM;
    }
    return 0;
}

int cpu_class_classify_cpus(int cpus)
{
    unsigned long big;
    int cpu;

    cpus = cpus < NR_CPUS ? cpus : NR_CPUS;
    big = find_big_part_number(cpus);
    /*
    * Will loop through all cpus and classify all who have
    * part_number = big as `big`. The rest will be classified as
    `LITTLE`.
    * So, for now only 2 classes will be auto-detected.
    */
    for (cpu = 0; cpu < cpus; cpu++)
    {
        unsigned long part_number = cpu_data[cpu].midr.
   part_number;
        if ( part_number == big )
        {
```

```
            cpumask_set_cpu(cpu, &class_to_cpumask[0]);
            cpu_to_class[cpu] = 0;
        }
        else
        {
            cpumask_set_cpu(cpu, &class_to_cpumask[1]);
            cpu_to_class[cpu] = 1;
        }
    }
    return 0;
}
```

Όπως φαίνεται παραπάνω, η *cpu_class_init* δημιουργεί δυναμικά δύο κλάσεις για την αρχιτεκτονική ARM. Στην συνέχεια, γινέται η κατηγοριοποίηση από την συνάρτηση *cpu_class_classify_cpus*. Η συγκεκριμένη συνάρτηση καλεί αρχικά την *find_big_part_number*, η οποία διατρέχει όλες τις ΚΜΕ και βρίσκει το μέγιστο part number. Έπειτα, διατρέχει ξανά όλες τις ΚΜΕ, και κατηγοριοποιεί όλες τις ΚΜΕ με το μέγιστο part number ως class 0 (big), ενώ τις υπόλοιπες ως class 1 (LITTLE). Σημειώνεται ότι ο πυρήνας του Xen Project έχει ήδη διαβάσει τις τιμές των MIDR και τις έχει αποθηκεύσει στον πίνακα *cpu_data*. Επομένως, δεν χρειάζεται να ξαναδιαβαστούν, και αρκεί η χρήση του συγκεκριμένου πίνακα. Επιπροσθέτως, η παραπάνω υλοποίηση έχει νόημα μόνο για δύο διαφορετικές κλάσεις πυρήνων (big.LITTLE), και όχι παραπάνω. Την περίοδο συγγραφής της συγκεκριμένης εργασίας δεν υπάρχει επεξεργαστής ARM με παραπάνω από δύο κλάσεις πυρήνων.

Ενδιαφέρον έχει και η συνάρτηση *start_xen* όπου φαίνεται η σειρά εκκίνησης των διαφόρων λειτουργιών του Xen και των πυρήνων της ΚΜΕ. Όπως μπορούμε να δούμε παρακάτω, πρώτα καλείται η συνάρτηση *init_idle_domain*, η οποία αρχικοποιεί τον scheduler, όπου καλείται και η *cpu_class_init* που δημιουργεί τους πίνακες. Στην συνέχεια, εκκινούν όλες οι ΚΜΕ και αρχικοποιείται ο πίνακας *cpu_data* από την συνάρτηση *cpu_up*. Έπειτα, αφού έχουν γίνει τα προηγούμενα βήματα, η συνάρτηση *cpu_class_classify_cpus* μπορεί να κληθεί με ασφάλεια.

setup_part.c

```
/*
 * xen/arch/arm/setup.c
 *
 * Early bringup code for an ARMv7-A with virt extensions.
 */

/* ... */

static void __init init_idle_domain(void)
```

```
{
    scheduler_init();
    set_current(idle_vcpu[0]);
    /* TODO: setup_idle_pagetable(); */
}

/* ... */

/* C entry point for boot CPU */
void __init start_xen(unsigned long boot_phys_offset,
                      unsigned long fdt_paddr,
                      unsigned long cpuid)
{
    size_t fdt_size;
    int cpus, i;
    paddr_t xen_paddr;
    const char *cmdline;
    struct bootmodule *xen_bootmodule;
    struct domain *dom0;
    struct xen_arch_domainconfig config;

    setup_cache();

    percpu_init_areas();
    set_processor_id(0); /* needed early, for smp_processor_id()
    */

    set_current((struct vcpu *)0xfffff000); /* debug sanity */
    idle_vcpu[0] = current;

    setup_virtual_regions(NULL, NULL);
    /* Initialize traps early allow us to get backtrace when an
   error occurred */
    init_traps();

    smp_clear_cpu_maps();

    /* This is mapped by head.S */
    device_tree_flattened = (void *)BOOT_FDT_VIRT_START
        + (fdt_paddr & ((1 << SECOND_SHIFT) - 1));
    fdt_size = boot_fdt_info(device_tree_flattened, fdt_paddr);

    cmdline = boot_fdt_cmdline(device_tree_flattened);
    printk("Command line: %s\n", cmdline);
    cmdline_parse(cmdline);

    /* Register Xen's load address as a boot module. */
    xen_bootmodule = add_boot_module(BOOTMOD_XEN,
                                (paddr_t)(uintptr_t)(_start +
```

```
boot_phys_offset),
                        (paddr_t)(uintptr_t)(_end - _start
+ 1), NULL);
 BUG_ON(!xen_bootmodule);

 xen_paddr = get_xen_paddr();
 setup_pagetables(boot_phys_offset, xen_paddr);

 /* Update Xen's address now that we have relocated. */
 printk("Update BOOTMOD_XEN from %"PRIpaddr"-%"PRIpaddr" => %
"PRIpaddr"-%"PRIpaddr"\n",
        xen_bootmodule->start, xen_bootmodule->start +
xen_bootmodule->size,
        xen_paddr, xen_paddr + xen_bootmodule->size);
 xen_bootmodule->start = xen_paddr;

 setup_mm(fdt_paddr, fdt_size);

 /* Parse the ACPI tables for possible boot-time
configuration */
 acpi_boot_table_init();

 end_boot_allocator();

 vm_init();
 dt_unflatten_host_device_tree();

 init_IRQ();

 platform_init();

 preinit_xen_time();

 gic_preinit();

 arm_uart_init();
 console_init_preirq();
 console_init_ring();

 system_state = SYS_STATE_boot;

 processor_id();

 smp_init_cpus();
 cpus = smp_get_max_cpus();

 init_xen_time();

 gic_init();
```

```
    softirq_init();

    tasklet_subsys_init();


    xsm_dt_init();

    init_maintenance_interrupt();
    init_timer_interrupt();

    timer_init();

    /* creates the cpu_class tables */
    init_idle_domain();

    rcu_init();

    arch_init_memory();

    local_irq_enable();
    local_abort_enable();

    smp_prepare_cpus(cpus);

    initialize_keytable();

    console_init_postirq();

    do_presmp_initcalls();

    for_each_present_cpu ( i )
    {
        if ( (num_online_cpus() < cpus) && !cpu_online(i) )
        {
            int ret = cpu_up(i);
            if ( ret != 0 )
                printk("Failed to bring up CPU %u (error %d)\n",
    i, ret);
        }
    }

    printk("Brought up %ld CPUs\n", (long)num_online_cpus());

    /* classifies cpus */
    cpu_class_classify_cpus(cpus);
    /* TODO: smp_cpus_done(); */

    setup_virt_paging();
```

```
    iommu_setup();

    do_initcalls();

    /*
     * It needs to be called after do_initcalls to be able to
    use
     * stop_machine (tasklets initialized via an initcall).
     */
    apply_alternatives_all();

    /* Create initial domain 0. */
    /* The vGIC for DOM0 is exactly emulating the hardware GIC
    */
    config.gic_version = XEN_DOMCTL_CONFIG_GIC_NATIVE;
    config.nr_spis = gic_number_lines() - 32;

    dom0 = domain_create(0, 0, 0, &config);
    if ( IS_ERR(dom0) || (alloc_dom0_vcpu0(dom0) == NULL) )
            panic("Error creating domain 0");

    dom0->is_privileged = 1;
    dom0->target = NULL;

    if ( construct_dom0(dom0) != 0)
            panic("Could not set up DOM0 guest OS");

    /* Scrub RAM that is still free and so may go to an
    unprivileged domain. */
    scrub_heap_pages();

    init_constructors();

    console_endboot();

    /* Hide UART from DOM0 if we're using it */
    serial_endboot();

    system_state = SYS_STATE_active;

    /* Must be done past setting system_state. */
    unregister_init_virtual_region();

    domain_unpause_by_systemcontroller(dom0);

    /* Switch on to the dynamically allocated stack for the idle
    vcpu
     * since the static one we're running on is about to be
```

```
    freed. */
    memcpy(idle_vcpu[0]->arch.cpu_info, get_cpu_info(),
           sizeof(struct cpu_info));
    switch_stack_and_jump(idle_vcpu[0]->arch.cpu_info, init_done
    );
}

/* ... */
```

Στην αρχιτεκτονική x86, δεν υπάρχει ακόμα ανάγκη για παρόμοια κατηγοριοποίηση των πυρήνων. Έτσι, το αντίστοιχο αρχείο δημιουργεί μόνο μία γενική κλάση, την class 0 και προσθέτει όλους τους πυρήνες σε αυτήν την κλάση. Η εργασία αυτή γίνεται στην συνάρτηση *cpu_class_init*, ενώ *cpu_class_classify_cpus* δεν χρησιμοποιείται.

cpu_class.c

```
#include <xen/cpu_class.h>
#include <xen/xmalloc.h>
#include <xen/errno.h>

int cpu_class_init(void)
{
    unsigned int cpu;

    /* Initialize global variables */
    NR_CLASSES = 1;

    if ( (cpu_to_class = xzalloc_array(uint16_t, NR_CPUS)) ==
   NULL )
        return -ENOMEM;
    if ( !zalloc_cpumask_arr(&class_to_cpumask, NR_CLASSES) )
    {
        xfree(cpu_to_class);
        return -ENOMEM;
    }

    for (cpu = 0; cpu < nr_cpu_ids; cpu ++ )
    {
        cpumask_set_cpu(cpu, &class_to_cpumask[0]);
        cpu_to_class[cpu] = 0;
    }

    return 0;
}

int cpu_class_classify_cpus(int cpus)
{
    /* Nothing to do */
```

```
    return 0;
}
```

3.4 Επικοινωνία με το Userspace

Στην συγκεκριμένη υποενότητα θα δούμε πως επιτυγχάνεται η επικοινωνία του Xen Hypervisor (kernelspace) με το Xen Toolstack (userspace), έτσι ώστε να επιτρέπεται η παρακολούθηση και η παραμετροποίηση της νέας λειτουργίας, δηλαδή των κλάσεων. Θα δούμε δηλαδή ποιές υπερκλήσεις υλοποιήθηκαν, ενώ στην επόμενη υποενότητα θα δούμε πως αυτές οι υπερκλήσεις χρησιμοποιούνται από το toolstack. Ο πηγαίος κώδικας αυτής της παραγράφου περιέχεται στους φάκελους /xen/include/public (αρχεία *.h) και /xen/common (αρχεία *.c) του Xen Project. Τα αρχεία *.c χτίζονται για όλες τις αρχιτεκτονικές, ενώ τα αρχεία *.h αυτής της παραγράφου χρησιμοποιούνται από τον Hypervisor, αλλά και το Toolstack.

Αρχικά, υλοποιούμε δύο νέες υπερκλήσεις, τις *XEN_DOMCTL_setvcpuclass* και *XEN_DOMCTL_getvcpuclass*. Η πρώτη χρησιμοποιείται για να θέσουμε τις κλάσεις στις οποίες ανήκει μια εικονική ΚΜΕ, ενώ η δεύτερη για να διαβάσουμε τις κλάσεις αυτές. Η δήλωση και η υλοποίηση αυτών των υπερκλήσεων φαίνεται παρακάτω.

domctl_part.h

```
/******************************************************************
 * domctl.h
 *
 * Domain management operations. For use by node control stack.
 *
 */

 /* ... */

/* Get/set which classes a vcpu belongs in. */
/* XEN_DOMCTL_setvcpuaffinity */
/* XEN_DOMCTL_getvcpuaffinity */
struct xen_domctl_classaffinity {
    uint32_t  vcpu;
    struct xenctl_bitmap classmap;
};
typedef struct xen_domctl_classaffinity
    xen_domctl_classaffinity_t;


/* ... */
```

```
struct xen_domctl {
    uint32_t cmd;
#define XEN_DOMCTL_createdomain                   1
#define XEN_DOMCTL_destroydomain                  2
/* ... */
#define XEN_DOMCTL_setvcpuclass                  80
#define XEN_DOMCTL_getvcpuclass                  81
#define XEN_DOMCTL_gdbsx_guestmemio            1000
#define XEN_DOMCTL_gdbsx_pausevcpu             1001
#define XEN_DOMCTL_gdbsx_unpausevcpu           1002
#define XEN_DOMCTL_gdbsx_domstatus             1003
    uint32_t interface_version; /* XEN_DOMCTL_INTERFACE_VERSION
    */
    domid_t  domain;
    union {
        struct xen_domctl_createdomain       createdomain;
        struct xen_domctl_getdomaininfo      getdomaininfo;
        struct xen_domctl_getmemlist         getmemlist;
        struct xen_domctl_getpageframeinfo3  getpageframeinfo3;
        struct xen_domctl_nodeaffinity       nodeaffinity;
        struct xen_domctl_vcpuaffinity       vcpuaffinity;
        struct xen_domctl_classaffinity      classaffinity;
        /* ... */
    } u;
};
typedef struct xen_domctl xen_domctl_t;
DEFINE_XEN_GUEST_HANDLE(xen_domctl_t);

#endif /* __XEN_PUBLIC_DOMCTL_H__ */
```

domctl_part.c

```
/*****************************************************************
 * domctl.c
 *
 * Domain management operations. For use by node control stack.
 *
 * Copyright (c) 2002-2006, K A Fraser
 */

 /* ... */

long do_domctl(XEN_GUEST_HANDLE_PARAM(xen_domctl_t) u_domctl)
{
    long ret = 0;
    bool_t copyback = 0;
    struct xen_domctl curop, *op = &curop;
    struct domain *d;
```

```
    if ( copy_from_guest(op, u_domctl, 1) )
        return -EFAULT;

    if ( op->interface_version != XEN_DOMCTL_INTERFACE_VERSION )
        return -EACCES;

    switch ( op->cmd )
    {

    /* ... */

    case XEN_DOMCTL_setvcpuclass:
    case XEN_DOMCTL_getvcpuclass:
    {
        struct vcpu *v;
        xen_domctl_classaffinity_t *classaff = &op->u.
   classaffinity;
        ret = -ESRCH;
        if ( (v = d->vcpu[classaff->vcpu]) == NULL )
            break;

        if ( op->cmd == XEN_DOMCTL_setvcpuclass )
        {
            cpumask_var_t classes;
            if ( !alloc_cpumask_var(&classes) )
            {
                ret = -ENOMEM;
                break;
            }

            ret = xenctl_bitmap_to_cpumask(&classes, &classaff->
   classmap);
            if (ret)
                goto setvcpuclass_out;

            ret = vcpu_set_classes(v, classes);

setvcpuclass_out:
            free_cpumask_var(classes);
            break;
        }

        ret = cpumask_to_xenctl_bitmap(&classaff->classmap, v->
   classes);
        break;
    }

    /* ... */
```

```
    default:
        ret = arch_do_domctl(op, d, u_domctl);
        break;
    }

    domctl_lock_release();

 domctl_out_unlock_domonly:
    if ( d )
        rcu_unlock_domain(d);

    if ( copyback && __copy_to_guest(u_domctl, op, 1) )
        ret = -EFAULT;

    return ret;
}
```

Όπως βλέπουμε, δηλώνεται μια νέα δομή, η *xen_domctl_classaffinity_t*, η οποία περιέχει την σύνδεση μεταξύ εικονικής ΚΜΕ και κλάσεων. Η δομή αυτή προστέθηκε στο union *xen_domctl.u* και χρησιμοποιείται ως παράμετρος στις δύο νέες υπερκλήσεις. Η *XEN_DOMCTL_setvcpuclass* διαβάζει τα *xen_domctl_classaffinity_t.vcpu*, *xen_domctl_classaffinity_t.classmap* και θέτει τις αντίστοιχες κλάσεις καλώντας την συνάρτηση *vcpu_set_classes*. Η *XEN_DOMCTL_getvcpuclass* διαβάζει το *xen_domctl_classaffinity_t.vcpu*, και επιστρέφει το ζητούμενο classmap στο πεδίο *xen_domctl_classaffinity_t.classmap*.

Εκτός από τα παραπάνω, τροποποιήθηκαν και οι sysctl κλήσεις *XEN_SYSCTL_physinfo* και *XEN_SYSCTL_cputopoinfo*. Η πρώτη κλήση επιστρέφει διάφορες πληροφορίες για το σύστημα, όπως ο αριθμός των φυσικών ΚΜΕ, και τροποποιήθηκε ώστε να επιστρέφει και τον αριθμό των ενεργών κλάσεων, στο καινούριο πεδίο *xen_sysctl_physinfo_t.nr_classes*. Η δεύτερη επιστρέφει πληροφορίες για όλες τις ενεργές φυσικές ΚΜΕ, και τροποποιήθηκε ώστε να επιστρέφει και την κλάση στην οποία ανήκει η ΚΜΕ, στο καινούριο πεδίο *xen_sysctl_cputopo_t.nr_cpu_class*. Η δήλωση των νέων πεδίων, καθώς και η υλοποίηση των τροποποιημένων κλήσεων παρουσιάζεται παρακάτω.

sysctl_part.h

```
/*
 * Get physical information about the host machine
 */
/* XEN_SYSCTL_physinfo */
 /* (x86) The platform supports HVM guests. */
#define _XEN_SYSCTL_PHYSCAP_hvm          0
```

```
#define XEN_SYSCTL_PHYSCAP_hvm                  (1u<<
    _XEN_SYSCTL_PHYSCAP_hvm)
 /* (x86) The platform supports HVM-guest direct access to I/O
    devices. */
#define _XEN_SYSCTL_PHYSCAP_hvm_directio 1
#define XEN_SYSCTL_PHYSCAP_hvm_directio  (1u<<
    _XEN_SYSCTL_PHYSCAP_hvm_directio)
struct xen_sysctl_physinfo {
    uint32_t threads_per_core;
    uint32_t cores_per_socket;
    uint32_t nr_cpus;     /* # CPUs currently online */
    uint32_t max_cpu_id;  /* Largest possible CPU ID on this
    host */
    uint32_t nr_nodes;    /* # nodes currently online */
    uint32_t max_node_id; /* Largest possible node ID on this
    host */
    uint32_t nr_classes;    /* # classes currently online */
    uint32_t max_class_id; /* Largest possible class ID on this
    host */
    uint32_t cpu_khz;
    uint64_aligned_t total_pages;
    uint64_aligned_t free_pages;
    uint64_aligned_t scrub_pages;
    uint64_aligned_t outstanding_pages;
    uint32_t hw_cap[8];

    /* XEN_SYSCTL_PHYSCAP_??? */
    uint32_t capabilities;
};
typedef struct xen_sysctl_physinfo xen_sysctl_physinfo_t;
DEFINE_XEN_GUEST_HANDLE(xen_sysctl_physinfo_t);

/* ... */

/* XEN_SYSCTL_cputopoinfo */
#define XEN_INVALID_CORE_ID     (~0U)
#define XEN_INVALID_SOCKET_ID   (~0U)
#define XEN_INVALID_NODE_ID     (~0U)
#define XEN_INVALID_CPU_CLASS_ID    (~0U)

struct xen_sysctl_cputopo {
    uint32_t core;
    uint32_t socket;
    uint32_t node;
    uint32_t cpu_class;
};
typedef struct xen_sysctl_cputopo xen_sysctl_cputopo_t;
DEFINE_XEN_GUEST_HANDLE(xen_sysctl_cputopo_t);
```

```
/* ... */
```

sysctl_part.c

```
/* ... */
long do_sysctl(XEN_GUEST_HANDLE_PARAM(xen_sysctl_t) u_sysctl)
{
    long ret = 0;
    int copyback = -1;
    struct xen_sysctl curop, *op = &curop;

    /* ... */
    switch ( op->cmd )
    {
    /* ... */
    case XEN_SYSCTL_physinfo:
    {
        xen_sysctl_physinfo_t *pi = &op->u.physinfo;

        memset(pi, 0, sizeof(*pi));
        pi->threads_per_core =
            cpumask_weight(per_cpu(cpu_sibling_mask, 0));
        pi->cores_per_socket =
            cpumask_weight(per_cpu(cpu_core_mask, 0)) / pi->
    threads_per_core;
        pi->nr_cpus = num_online_cpus();
        pi->nr_nodes = num_online_nodes();
        pi->nr_classes = NR_CLASSES;
        pi->max_node_id = MAX_NUMNODES-1;
        pi->max_cpu_id = nr_cpu_ids - 1;
        pi->max_class_id = nr_cpu_ids - 1;
        pi->total_pages = total_pages;
        /* Protected by lock */
        get_outstanding_claims(&pi->free_pages, &pi->
    outstanding_pages);
        pi->scrub_pages = 0;
        pi->cpu_khz = cpu_khz;
        arch_do_physinfo(pi);

        if ( copy_to_guest(u_sysctl, op, 1) )
            ret = -EFAULT;
    }
    break;
    /* ... */
    case XEN_SYSCTL_cputopoinfo:
    {
        unsigned int i, num_cpus;
        xen_sysctl_cputopoinfo_t *ti = &op->u.cputopoinfo;
```

```
        num_cpus = cpumask_last(&cpu_online_map) + 1;
        if ( !guest_handle_is_null(ti->cputopo) )
        {
            xen_sysctl_cputopo_t cputopo = { 0 };

            if ( num_cpus > ti->num_cpus )
                num_cpus = ti->num_cpus;
            for ( i = 0; i < num_cpus; ++i )
            {
                if ( cpu_present(i) )
                {
                    cputopo.core = cpu_to_core(i);
                    cputopo.socket = cpu_to_socket(i);
                    cputopo.node = cpu_to_node(i);
                    if ( cputopo.node == NUMA_NO_NODE )
                        cputopo.node = XEN_INVALID_NODE_ID;
                    cputopo.cpu_class = cpu_to_class[i];
                }
                else
                {
                    cputopo.core = XEN_INVALID_CORE_ID;
                    cputopo.socket = XEN_INVALID_SOCKET_ID;
                    cputopo.node = XEN_INVALID_NODE_ID;
                    cputopo.cpu_class = XEN_INVALID_CPU_CLASS_ID
;
                }

                if ( copy_to_guest_offset(ti->cputopo, i, &
cputopo, 1) )
                {
                    ret = -EFAULT;
                    break;
                }
            }
        }
        else
            i = num_cpus;

        if ( !ret && (ti->num_cpus != i) )
        {
            ti->num_cpus = i;
            if ( __copy_field_to_guest(u_sysctl, op,
                                       u.cputopoinfo.num_cpus) )
            {
                ret = -EFAULT;
                break;
            }
        }
    }
```

```
    break;
    /* ... */
    }

 out:
    if ( copyback && (!ret || copyback > 0) &&
         __copy_to_guest(u_sysctl, op, 1) )
        ret = -EFAULT;

    return ret;
}
```

3.5 To Toolstack

Σε αυτήν την υποενότητα θα δούμε πως χρησιμοποιούνται από τα εργαλεία οι κλήσεις που παρουσιάστηκαν στην προηγούμενη υποενότητα, καθώς και πως μπορούμε να παρακολουθήσουμε και να παραμετροποιήσουμε την νέα λειτουργία των κλάσεων. Ο κώδικας αυτής της παραγράφου βρίσκεται στους φακέλους /tools/libxc και /tools/libxl.

Το libxc περιέχει τις συναρτήσεις βιβλιοθήκης του Toolstack. Σε αυτόν τον φάκελο υλοποιούμε τις νέες συναρτήσεις *xc_vcpu_setclass* και *xc_vcpu_getclass* που καλούν τις αντίστοιχες domctl, καθώς και τις συναρτήσεις *xc_get_nr_classes* και *xc_get_max_classes* που καλούν τις αντίστοιχες sysctl.

Στο libxl χρησιμοποιούμε αυτές τις συναρτήσεις του libxc για να υλοποιήσουμε τις εντολές που αναφέρθηκαν στην υποενότητα 3.1. Επιπλέον, σε αυτό το επίπεδο γίνεται και η αντιστοίχηση μεταξύ των αναγνωριστικών των κλάσεων και των ετικετών τους, όπως φαίνεται παρακάτω.

libxl_utils_part.c

```
const char *libxl_class_to_string(uint32_t cpu_class) {
    /* Might become more complex at some point */
    static int nr_valid_labels = 2;
    static const char *cpu_class_labels[] = {"big", "LITTLE"};
    return cpu_class >= nr_valid_labels ? NULL :
    cpu_class_labels[cpu_class];
}

uint32_t libxl_string_to_class(const char *str) {
    int i = -1;
    const char *label;

    while ((label = libxl_class_to_string(++i)) != NULL)
        if (strcmp(str, label) == 0)
```

```
            return i;

    return -1;
}
```

Ας δούμε τώρα παραδείγματα εκτέλεσης των τροποποιημένων εντολών. Στην εντολή *xl info* προστέθηκαν δύο νέα flags, το -c (--classes) που δείχνει τις κλάσεις των φυσικών ΚΜΕ σε αριθμητική μορφή, καθώς και το -l (--labels) που δείχνει τις κλάσεις στη μορφή ετικετών. Για παράδειγμα:

xl info -cl

```
# xl info -cl
host                   : odroid-server
release                : 3.10.82+
version                : #1 SMP PREEMPT Thu Jan 26 17:02:43 EET
    2017
machine                : armv7l
nr_cpus                : 8
max_cpu_id             : 127
nr_nodes               : 1
cores_per_socket       : 1
threads_per_core       : 1
cpu_mhz                : 24
hw_caps                : 00000000:00000000:00000000:00000000
    :00000000:00000000:00000000:00000000
virt_caps              :
total_memory           : 2026
free_memory            : 616
sharing_freed_memory   : 0
sharing_used_memory    : 0
outstanding_claims     : 0
free_cpus              : 0
cpu_topology           :
cpu:    core    socket     class
  0:       0         0    LITTLE
  1:       0         0    LITTLE
  2:       0         0    LITTLE
  3:       0         0    LITTLE
  4:       0         0       big
  5:       0         0       big
  6:       0         0       big
  7:       0         0       big
xen_major              : 4
xen_minor              : 6
xen_extra              : .0
xen_version            : 4.6.0
xen_caps               : xen-3.0-armv7l
xen_scheduler          : credit
```

```
xen_pagesize            : 4096
platform_params         : virt_start=0x200000
xen_changeset           : Fri Jan 27 18:28:07 2017 +0200 git:
    d0fc885-dirty
xen_commandline         : sync_console console=dtuart dtuart=/
    serial@12C20000 dom0_mem=800M dom0_max_vcpus=1 dom0_vcpus_pin
     guest_loglvl=all loglvl=all
cc_compiler             : arm-eabi-gcc (Linaro GCC 4.9-2016.02)
    4.9.4 20151028 (prereleas
cc_compile_by           : thanasis
cc_compile_domain       :
cc_compile_date         : Wed Feb  1 17:16:34 EET 2017
xend_config_format      : 4
```

Για να μπορούμε να δημιουργήσουμε εικονικές μηχανές που χρησιμοποιούν την λειτουργία των κλάσεων, θα πρέπει να προσθέσουμε στα config αρχεία τους την νέα ιδιότητα *vcpuclass*, η οποία δέχεται την αντιστοίχιση εικονικών ΚΜΕ σε μία ή περισσότερες κλάσεις με μορφή λίστας, καθώς και τις ειδικές λέξεις *all*, *mixed*. Για παράδειγμα:

domU–big.cfg

```
name = "domU-big"
# various other options which are skipped
vcpus = 4
vcpuclass = big
```

domU–LITTLE.cfg

```
name = "domU-LITTLE"
# various other options which are skipped
vcpus = 4
vcpuclass = LITTLE
```

domU.cfg

```
name = "domU"
# various other options which are skipped
vcpus = 8
vcpuclass = ["0-3:LITTLE", "4-7:big"]
```

Στην εντολή *xl vcpu-list* προστέθηκαν τα ίδια flags -c και -l με την ίδια συμπεριφορά. Για παράδειγμα, μπορούμε να δημιουργήσουμε τις εικονικές μηχανές και να δούμε τις κλάσεις των ΚΜΕ όπως φαίνεται παρακάτω:

xl vcpu-list -cl

```
# xl create domU-big.cfg
# xl create domU-LITTLE.cfg
```

```
# xl create domU.cfg
# xl vcpu-list -cl
Name              ID  VCPU   CPU State   Time(s) Affinity (Hard / Soft
    / Class)
Domain-0           0     0    1   -b-      19.2  0-3 / all / LITTLE
Domain-0           0     1    2   -b-      74.7  0-3 / all / LITTLE
Domain-0           0     2    3   -b-      12.2  0-3 / all / LITTLE
Domain-0           0     3    2   -b-      11.2  0-3 / all / LITTLE
Domain-0           0     4    4   -b-      10.3  4-7 / all / big
Domain-0           0     5    7   -b-      12.6  4-7 / all / big
Domain-0           0     6    6   r--      16.1  4-7 / all / big
Domain-0           0     7    7   -b-       9.3  4-7 / all / big
domU-big           1     0    4   -b-       0.1  4-7 / all / big
domU-big           1     1    5   -b-       0.3  4-7 / all / big
domU-big           1     2    6   -b-       0.1  4-7 / all / big
domU-big           1     3    7   -b-       0.8  4-7 / all / big
domU-LITTLE        2     0    3   -b-       0.1  0-3 / all / LITTLE
domU-LITTLE        2     1    2   -b-       0.6  0-3 / all / LITTLE
domU-LITTLE        2     2    1   -b-       0.5  0-3 / all / LITTLE
domU-LITTLE        2     3    0   -b-       0.1  0-3 / all / LITTLE
domU               3     0    4   -b-       0.1  4-7 / all / big
domU               3     1    5   -b-       0.9  4-7 / all / big
domU               3     2    4   -b-       0.0  4-7 / all / big
domU               3     3    7   -b-       0.1  4-7 / all / big
domU               3     4    2   -b-       0.0  0-3 / all / LITTLE
domU               3     5    3   -b-       0.0  0-3 / all / LITTLE
domU               3     6    1   -b-       0.1  0-3 / all / LITTLE
domU               3     7    0   -b-       0.1  0-3 / all / LITTLE
```

Για να σιγουρευτούμε ότι όλοι οι πυρήνες αναγνωρίζονται σωστά από τους guests, μπορούμε να συνδεθούμε σε έναν από αυτούς και να δούμε το cpuinfo. Παρακάτω παρατηρούμε ότι για την domU έχουν αναγνωριστεί σωστά 4 A53 πυρήνες (CPU part: 0xd03) και 4 A57 (CPU part: 0xd07):

Guest cpuinfo

```
# xl console domU
root@domU:~# cat /proc/cpuinfo
processor       : 0
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 2

processor       : 1
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer : 0x41
```

```
CPU architecture : 8
CPU variant      : 0x0
CPU part         : 0xd03
CPU revision     : 2

processor        : 2
Features         : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer  : 0x41
CPU architecture : 8
CPU variant      : 0x0
CPU part         : 0xd03
CPU revision     : 2

processor        : 3
Features         : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer  : 0x41
CPU architecture : 8
CPU variant      : 0x0
CPU part         : 0xd03
CPU revision     : 2

processor        : 4
Features         : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer  : 0x41
CPU architecture : 8
CPU variant      : 0x1
CPU part         : 0xd07
CPU revision     : 0

processor        : 5
Features         : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer  : 0x41
CPU architecture : 8
CPU variant      : 0x1
CPU part         : 0xd07
CPU revision     : 0

processor        : 6
Features         : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer  : 0x41
CPU architecture : 8
CPU variant      : 0x1
CPU part         : 0xd07
CPU revision     : 0

processor        : 7
Features         : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer  : 0x41
CPU architecture : 8
```

```
CPU variant     : 0x1
CPU part        : 0xd07
CPU revision    : 0
```

Τέλος, ας χρησιμοποιήσουμε την *xl vcpu-class*, για να αλλάξουμε την κλάση όλων των εικονικών ΚΜΕ της domU-big:

xl vcpu-class

```
# xl vcpu-list -cl domU-big
Name          ID  VCPU   CPU State   Time(s) Affinity (Hard / Soft
    / Class)
domU-big       1     0     4   -b-       0.1  4-7 / all / big
domU-big       1     1     5   -b-       0.3  4-7 / all / big
domU-big       1     2     6   -b-       0.1  4-7 / all / big
domU-big       1     3     7   -b-       0.8  4-7 / all / big
# xl vcpu-class domU-big all LITTLE
# xl vcpu-list -cl domU-big
Name          ID  VCPU   CPU State   Time(s) Affinity (Hard / Soft
    / Class)
domU-big       1     0     0   -b-       0.1  0-3 / all / LITTLE
domU-big       1     1     0   -b-       0.3  0-3 / all / LITTLE
domU-big       1     2     0   -b-       0.1  0-3 / all / LITTLE
domU-big       1     3     0   -b-       0.8  0-3 / all / LITTLE
```

4 Πειραματική Αποτίμηση

Σε αυτήν την ενότητα περιγράφονται τα πειράματα που εκτελέστηκαν για την μέτρηση της συμπεριφοράς του τροποποιημένου Xen σε περιβάλλον big.LITTLE. Επιπλέον συγκρίνουμε την επίδοση των εικονικών μηχανών big.LITTLE με την επίδοση του τυπικού Linux.

4.1 Πειραματική Διάταξη

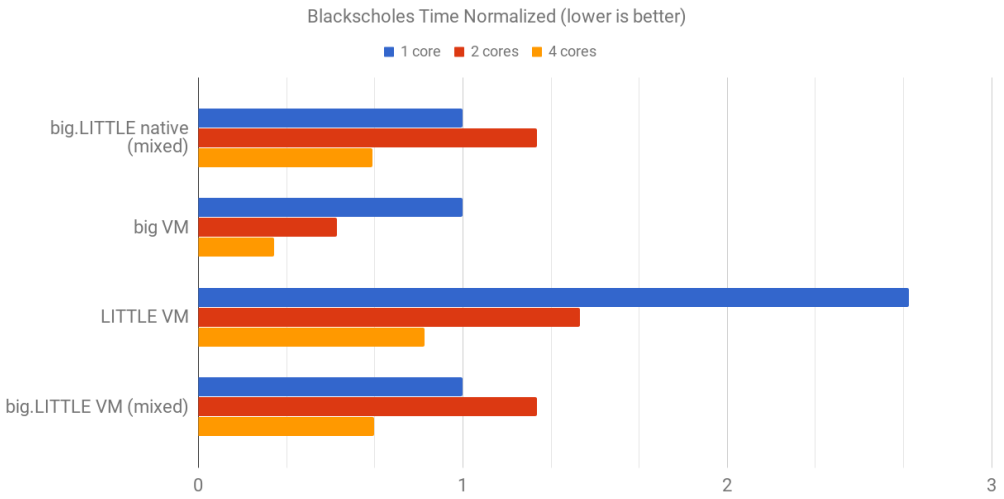
Για όλα τα πειράματα χρησιμοποιήσαμε την πλακέτα ODROID-XU4, τα τεχνικά χαρακτηριστικά της οποίας φαίνονται στον πίνακα 4.1. Το λογισμικό που χρησιμοποιήθηκε ήταν Xen 4.9 Unstable, με την προσθήκη των αλλαγών που υλοποιήσαμε στα πλαίσια της εργασίας, καθώς και πυρήνας Linux v3.13. Ως μετροπρογράμματα χρησιμοποιήσαμε το blackscholes της σουίτας parsec, καθώς και το sysbench v1.0.8.

SoC	Samsung Exynos 5422
Επεξεργαστές	4x Cortex-A15 & 4x Cortex-A7
Κάρτα Γραφικών	Mali-T628 MP6
Κεντρική Μνήμη	2Gbyte LPDDR3

Πίνακας 4.1: Τα τεχνικά χαρακτηριστικά της πλακέτας ODROID-XU4

4.2 Πειραματική Αποτίμηση

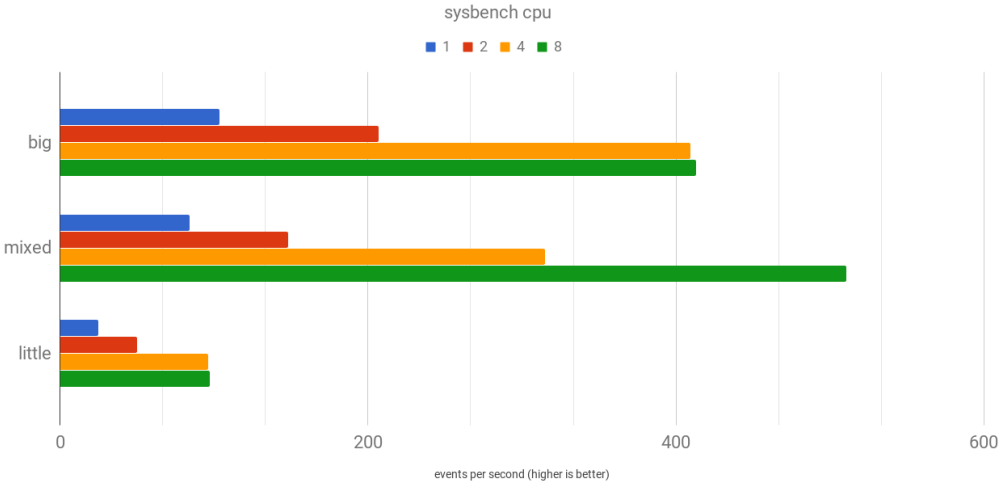
Αρχικά εκτελέσαμε το μετροπρογράμμα blackscholes της σουίτας parsec, σε 4 διαφορετικά περιβάλλοντα: εικονική μηχανή με αποκλειστικά big πυρήνες (*vcpuclass=big*), εικονική μηχανή με αποκλειστικά LITTLE πυρήνες (*vcpuclass=LITTLE*), εικονική μηχανή με big και LITTLE πυρήνες (*vcpuclass=mixed*), και τυπικό Linux (bare-metal). Στην συνέχεια δημιουργήσαμε το γράφημα του χρόνου εκτέλεσης, κανονικοποιημένο στον χρόνο εκτέλεσης ενός big CPU. Το συγκεκριμένο διάγραμμα παρουσιάζεται στο σχήμα 4.1.



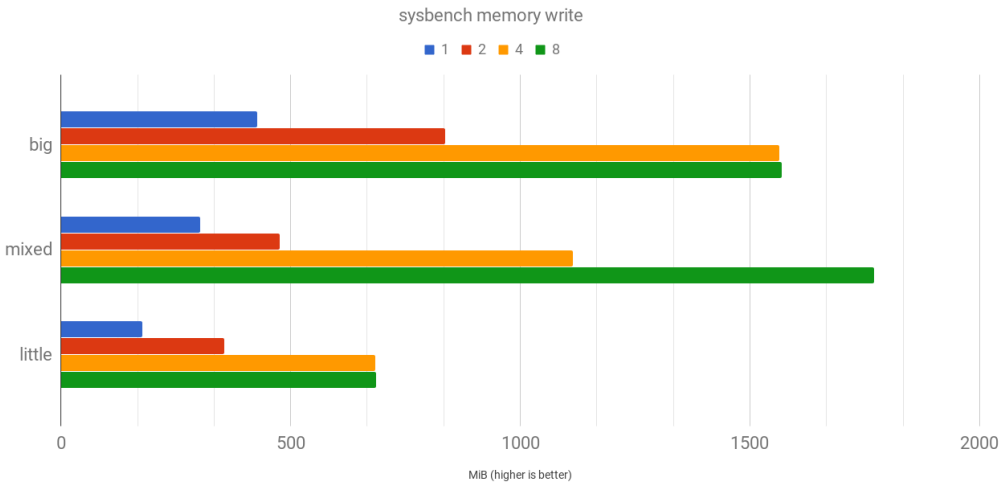
Σχήμα 4.1: Κανονικοποιημένος χρόνος εκτέλεσης του blackscholes

Παρατηρούμε ότι ο χρόνος εκτέλεσης του προγράμματος σε εικονική μηχανή big.LITTLE είναι ίσος με τον αντίστοιχο χρόνο εκτέλεσης του bare-metal Linux. Έτσι, αποδεικνύουμε ότι τα χαρακτηριστικά big.LITTLE αποδίδονται στη εικονική μηχανή με τον σωστό τρόπο, καθώς και ότι η προσέγγιση μας για την εικονικοποίηση του big.LITTLE έχει μηδενική επιβάρυνση στον χρόνο εκτέλεσης. Επιπλέον παρατηρούμε ότι ο χρόνος εκτέλεσης του προγράμματος σε big πυρήνες είναι πολύ μικρότερος από τον αντίστοιχο σε LITTLE πυρήνες, όπως περιμέναμε.

Επιπλέον, εκτέλεσαμε τα μετροπρογράμματα cpu, memory της σουίτας sysbench. Τα αποτελέσματα παρουσιάζονται στα σχήματα 4.2 και 4.3.



Σχήμα 4.2: sysbench cpu



Σχήμα 4.3: sysbench memory

Εδώ εκτελέσαμε 3 εικονικές μηχανές, μία big, μία LITTLE και μία mixed, όλες με 8 vcpus. Στην συνέχεια εκτελέσαμε τα μετροπρογράμματα sysbench cpu και sybench memory με 1, 2, 4 και 8 threads. Και πάλι παρατηρούμε ότι η LITTLE εικονική μηχανή είναι πολύ πιο αργή από την big, ενώ η big.LITTLE βρίσκεται κάπου ενδιάμεσα για 1, 2 και 4 threads. Για 8 threads, η mixed εικονική μηχανή έχει καλύτερη επίδοση αφού είναι η μόνη που αξιοποιεί και τις 8 pCPUs του μηχανήματος.

5 Σύνοψη

Στην συγκεκριμένη εργασία μελετήσαμε τις ετερογενείς αρχιτεκτονικές και ειδικότερα την αρχιτεκτονική big.LITTLE της εταιρείας ARM. Μελετήσαμε τις μεταβολές που έγιναν στον πυρήνα του Linux για την υποστηρίξη των ετερογενών αρχιτεκτονικών, και παρουσιάσαμε τα προβλήματα που προκύποτουν κατά την απόπειρα εικονικοποίησης τους. Στην συνέχεια σχεδιάσαμε και υλοποιήσαμε μία ολοκληρωμένη λύση για την εικονικοποίηση των ετερογενών αρχιτεκτονικών στο Xen Project.

Έτσι, υλοποιήσαμε τη νέα λειτουργία των κλάσεων επεξεργαστικών μονάδων χάρη στην οποία μπορούμε να χωρίζουμε τις ΚΜΕ σε διαφορετικές κλάσεις ανάλογα με τα τεχνικά τους χαρακτηριστικά, όπως αριθμός μοντέλου, συχνότητα και τάση λειτουργίας. Επιπλέον, σχεδιάσαμε και υλοποιήσαμε τα κατάλληλα εργαλεία για τον χειρισμό αυτών των κλάσεων. Χάρη στις παραπάνω προσθήκες, καθιστάται δυνατή η δημιουργία και η χρήση ετερογενών εικονικών μηχανών.

Τέλος, χρησιμοποιήσαμε μετροπρογράμματα με την βοήθεια των οποίων αποδείξαμε την ορθή λειτουργία του νέου μοντέλου στην πλακέτα big.LITTLE ODROID-XU4. Ταυτόχρονα, αποδείξαμε ότι η επιβάρυνση στην ταχύτητα εκτέλεσης που δημιουργείται από τις εικονικές μηχανές του Xen είναι πρακτικά μηδενική.

Στο μέλλον, μπορούμε να επεκτείνουμε το μοντέλο έτσι ώστε να έχει καλύτερη υποστήριξη και για άλλες ετερογενείς αρχιτεκτονικές εκτός από την big.LITTLE. Αυτό μπορεί να επιτευχθεί με την τροποποίηση του αλγόριθμου αναγνώρισης των κλάσεων ώστε να λειτουργεί σε περισσότερες αρχιτεκτονικές και να χρησιμοποιεί περισσότερα χαρακτηριστικά των πυρήνων, όπως την τάση λειτουργίας, για την κατηγοριοποίηση τους. Επιπροσθέτως, θα μπορούσαμε να αναπτύξουμε έναν μηχανισμό για τον χειροκίνητο ορισμό των κλάσεων. Τέλος, σχεδιάζουμε να κάνουμε τις κατάλληλες τροποποιήσεις στον πηγαίο κώδικα, έτσι ώστε να συγχωνευτεί με την κύρια έκδοση του Xen Project.

Αναφορές

[1] Arm cortex-a15 mpcore processor technical reference manual. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0438i/CHDGGCFH.html. Ημερομηνία Πρόσβασης: 11/7/2017.

[2] big.LITTLE technology. https://www.arm.com/products/processors/technologies/biglittleprocessing.php. Ημερομηνία Πρόσβασης: 7/5/2017.

[3] Cortex-a5 technical reference manual. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0433c/BABGDDJH.html. Ημερομηνία Πρόσβασης: 11/7/2017.

[4] Cortex-a7 mpcore technical reference manual. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0464f/BABIGAED.html. Ημερομηνία Πρόσβασης: 11/7/2017.

[5] Virtualization in education. http://www-07.ibm.com/solutions/in/education/download/Virtualization%20in%20Education.pdf, 2007. Ημερομηνία Πρόσβασης: 7/5/2017.

[6] Credit scheduler. https://wiki.xen.org/wiki/Credit_Scheduler, 2016. Ημερομηνία Πρόσβασης:28/5/2017.

[7] Heterogeneous multi processing support in xen. https://lists.xenproject.org/archives/html/xen-devel/2016-12/msg00826.html, 2016. Ημερομηνία Πρόσβασης:1/7/2017.

[8] Xen project schedulers. https://wiki.xen.org/wiki/Xen_Project_Schedulers, 2016. Ημερομηνία Πρόσβασης:28/5/2017.

[9] Xen project software overview. https://wiki.xen.org/wiki/Xen_Project_Software_Overview, 2016. Ημερομηνία Πρόσβασης:28/5/2017.

[10] Xl. https://wiki.xen.org/wiki/XL, 2016. Ημερομηνία Πρόσβασης:28/5/2017.

[11] John Fischer-Ogden. Hardware support for efficient virtualization. http://cseweb.ucsd.edu/~jfisherogden/hardwareVirt.pdf. Ημερομηνία Πρόσβασης: 7/5/2017.

[12] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 1966.

[13] George Grey. big.little software update. https://www.linaro.org/blog/hardware-update/big-little-software-update/, 2013. Ημερομηνία Πρόσβασης:11/8/2017.

[14] Brian Jeff. Ten things to know about big.LITTLE. https://community.arm.com/processors/b/blog/posts/ten-things-to-know-about-big-little, 2013. Ημερομηνία Πρόσβασης: 20/5/2017.

[15] Michael Larabel. Ubuntu 15.10: Kvm vs. xen vs. virtualbox virtualization performance. http://www.phoronix.com/scan.php?page=article&item=ubuntu-1510-virt, 2015. Ημερομηνία Πρόσβασης:28/5/2017.

[16] Nakul Manchanda and Karan Anand. Non uniform memory access (numa). http://cs.nyu.edu/~lerner/spring10/projects/NUMA.pdf, 2010. Ημερομηνία Πρόσβασης: 7/5/2017.

[17] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 1965.

[18] Gordon E. Moore. Progress in digital integrated electronics. *International Electron Devices Meeting*, 1975.

[19] Andy Phillips. Arm unveils its most energy efficient application processor ever; redefines traditional power and performance relationship with big.little processing. https://www.arm.com/about/newsroom/arm-unveils-its-most-energy-efficient-application-processor-ever-with-bigli
php, 2011. Ημερομηνία Πρόσβασης: 13/8/2017.

[20] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM. 17*, 1974.

[21] Amar Shan. Heterogeneous processing: a strategy for augmenting moore's law. *Linux Journal*, 2006.

[22] Amit Singh. An introduction to virtualization. http://www.kernelthread.com/publications/virtualization, 2004. Ημερομηνία Πρόσβασης: 7/5/2017.

[23] VMware. Understanding full virtualization, paravirtualization, and hardware assist. https://www.vmware.com/content/

dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf, 2008. Ημερομηνία Πρόσβασης: 28/5/2017.