ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

PERFORMANCE OVERHEAD AND ISOLATION OF EXECUTING APPLICATIONS ON VIRTUAL MACHINES

Μάιος 2011

Ατομική Διπλωματική Εργασία

PERFORMANCE OVERHEAD AND ISOLATION OF EXECUTING APPLICATIONS ON VIRTUAL MACHINES

Νικολαΐδης Γιώργος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

PERFORMANCE OVERHEAD AND ISOLATION OF EXECUTING APPLICATIONS ON VIRTUAL MACHINES

Νικολαΐδης Γιώργος

Επιβλέπων Καθηγητής

Trancoso Pedro

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2011

Ευχαριστίες

Αρχικά θα ήθελα να εκφράσω τις ευχαριστίες μου στον κύριο Trancoso Pedro για όλη τη βοήθεια, καθοδήγηση και υποστήριξη που μου παρείχε όχι μόνο για αυτή τη διπλωματική εργασία αλλά και για τις σπουδές μου. Θα ήθελα ακόμη να τον ευχαριστήσω για το θέμα που μου πρόσφερε όπου μέσα από τη συνεργασία αυτή κατάφερα να πάρω αρκετές γνώσεις και σημαντικά πράγματα στον τομέα των VM, καθώς επίσης και στο να γνωρίσω καλύτερα τον τομέα της έρευνας. Μέσα από τις συμβουλές του καθώς και όλες τις γνώσεις του σαν εκπαιδευτικός κατάφερα να ολοκληρώσω την παρούσα διπλωματική εργασία.

Ακόμη ένα μεγάλο ευχαριστώ στον Πετρίδη Παναγιώτη τόσο για τη βοήθεια όσο και για την υποστήριξη του καθώς και για τις γνώσεις του πάνω σε θέματα που αφορούν τα VM.

Τέλος να εκφράσω ένα ευχαριστώ στους φίλους μου για όλη την ψυχολογική υποστήριξη και βοήθεια που παρείχαν καθώς επίσης και στην οικογένεια μου για όλη την οικονομική υποστήριξη αλλά και φροντίδα που έπαιρνα κατά την περίοδο των σπουδών μου.

Περίληψη

Οι υπολογιστές σήμερα είναι αρκετά ισχυροί για να χρησιμοποιήσουν το virtualization με τρόπο τον οποίο να αξιοποιούν καλύτερα τους πόρους του υλικού, έτσι ώστε να μπορεί να επιτευχθεί η καλύτερη δυνατή ολική απόδοση. Είναι σημαντικό κάποιος να γνωρίζει όλα τα οφέλη που μπορεί «κερδίσει» χρησιμοποιώντας virtual machines και σε desktops αλλά και στους server, με ένα αμελητέο κόστος στην επίδοση τους. Μερικά από τα οφέλη που προσφέρει το Virtualization είναι το program isolation, τη δυνατότητα να δημιουργεί κάποιος διάφορα λειτουργικά συστήματα πάνω στην ίδια μηχανή, τη δημιουργία διαφόρων virtual machine clones για διάφορα setups καθώς και διαφόρων σημείων επαναφοράς όπου είναι μια αρκετά εύκολη διαδικασία, να μειώνει τον αριθμό των φυσικών servers που χρειάζονται από ένα οργανισμό, τη δυνατότητα back-up που επίσης είναι εύκολη, τόσο όσο είναι το copy-paste και άλλα πολλά οφέλη.

Στην εργασία αυτή παρουσιάζεται αναλυτική απόδοση μερικών εφαρμογών, ενώ αυτές εκτελούνται σε διάφορα virtual machines. Σκοπός μας είναι να μελετήσουμε την επίδοση εφαρμογών χρησιμοποιώντας το isolation που παρέχουν τα virtual machines έτσι ώστε να δούμε αν μπορούμε να πετύχουμε καλύτερους ή τουλάχιστο αποδεκτούς χρόνους εκτέλεσης από την περίπτωση που δεν χρησιμοποιούνται καθόλου τα virtual machines.

Πιο συγκεκριμένα χρησιμοποιούμε διάφορα σενάρια όπου δύο ή περισσότερα virtual machines τρέχουν την ίδια ώρα εκτελώντας παράλληλα κάποιες εφαρμογές πάνω σε πολυπύρηνους επεξεργαστές. Τα πειράματα περιέχουν την εκτέλεση εφαρμογών από το PARSEC benchmark suite πάνω στο Oracle VirtualBox σε 64-bit αρχιτεκτονική.

Στα πλαίσια της μελέτης αυτής καταφέραμε να δείξουμε ότι το να έχει κανείς πολλά virtual machines πάνω στην ίδια μηχανή, παρόλο που θα πρέπει να αποδεχτεί το overhead που προσθέτουν, θα του προσφέρουν μια σταθερότητα όσο αφορά την εκτέλεση των εφαρμογών του ακόμη και στις περιπτώσεις όπου ο αριθμός των threads αυξομειώνεται συνεχώς. Αυτό οφείλεται κυρίως στο isolation που παρέχουν τα virtual machines, αφού για κάθε εφαρμογή δημιουργείται το δικό της εικονικό περιβάλλον με τους απόλυτα δικούς της πόρους.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Υποκίνηση Εργασίας

Η μεγάλη ετερογένεια των σημερινών υπολογιστικών συστημάτων αυξάνει την πολυπλοκότητα διαχείρισης τους. Για παράδειγμα ένας οργανισμός μπορεί να χρειάζεται να υποστηρίζει ένα μεγάλο αριθμό λειτουργικών συστημάτων σε πολλαπλές μηχανές. Επιπλέον «τα παρεπόμενα των σημερινών λειτουργικών συστημάτων και η αρχιτεκτονική των λογισμικών συστήματος έχουν ως αποτέλεσμα οι πλείστοι εξυπηρετητές να αξιοποιούν κάτω από το 10% της λειτουργικότητας τους», [5]. Το virtualization είναι μια μέθοδος που μπορεί να επιφέρει σημαντικά οφέλη σε αυτό τον τομέα. Η μέθοδος αυτή επιτρέπει τη δυνατότητα ταυτόχρονης εκτέλεσης πολλών λειτουργικών συστημάτων πάνω στον ίδιο εξυπηρετητή, περιορίζοντας τον αριθμό των αναγκαίων εξυπηρετητών και κατά συνέπεια το κόστος αγοράς και συντήρησης. Σύμφωνα με το [5] το κόστος μπορεί να μειωθεί μέχρι και στο δεκαπλάσιο.

Το virtualization έχει τις ρίζες του στους μηχανισμούς απομόνωσης παραδοσιακών λειτουργικών συστημάτων της δεκαετίας του ‘70. Για παράδειγμα η χρήση της εικονικής μνήμης επιτρέπει το φυσικό διαχωρισμό εφαρμογών. Στην περίπτωση του virtualization γίνεται απομόνωση των συνυπαρχόντων λειτουργικών συστημάτων. Αυτό επιταχύνεται με τη χρήση των Virtual Machine Monitors (VMM), τα οποία υλοποιούνται ως hypervisors ή σαν συνηθισμένες εφαρμογές. Τα hypervisors είναι ένα στρώμα λογισμικού το οποίο παρεμβάλλεται μεταξύ του υλικού και των λειτουργικών συστημάτων που φιλοξενούνται στον εξυπηρετητή. Η επικοινωνία με τα λειτουργικά συστήματα που φιλοξενεί μπορεί να γίνει μέσω paravirtualization, το οποίο είναι ένα Application Interface (API) που παρέχει ο hypervisor μέσω του οποίου οι αιτήσεις για εκτέλεση των privileged operations

διαβιβάζονται από το λειτουργικό σύστημα. Με άλλα λόγια ο hypervisor και το λειτουργικό που φιλοξενεί συνεργάζονται για την επίτευξη της βέλτιστης επίδοσης. Σε αντίθεση η υλοποίηση με συνηθισμένες εφαρμογές τοποθετεί τα VMM ένα επίπεδο πιο πάνω μιας και χρειάζονται ένα πρωτεύων λειτουργικό σύστημα μέσα στο οποίο να τρέχουν. Ένα παράδειγμα hypervisor είναι το XEN [22] ενώ ένα παράδειγμα VMM υλοποιημένου ως συνηθισμένη εφαρμογή είναι το VirtualBox [55].

Το XEN υποστηρίζει τις ακόλουθες λειτουργίες: διαχείριση πολλαπλών πυρήνων και επεξεργαστών, κατανομή μνήμης, επικοινωνία με συσκευές εισόδου/εξόδου και asynchronous events. Στην περίπτωση διαχείρισης πολλαπλών πυρήνων/επεξεργαστών και της μνήμης όταν το guest operating system ανανεώνει δομές δεδομένων του υλικού όπως το Page Table ή διεκπεραιώνει μια DMA λειτουργία τότε επικοινωνεί με το XEN μέσω του ειδικευμένου API του για paravirtualization. Αυτό επιτρέπει στο XEN να διατηρεί μια συνεπή κατάσταση των δομών δεδομένων του υλικού καθώς λαμβάνει χώρα η εναλλαγή μεταξύ των λειτουργικών συστημάτων που φιλοξενούνται. Από την άλλη μέσω του paravirtualization API το XEN πληροφορεί τα guest operating systems για την κατάσταση διαφόρων φυσικών συμβάντων, όπως για την πάροδο του χρόνου ενόσω δεν ήταν η σειρά του για εκτέλεση. Ως αποτέλεσμα τα guest operating systems πρέπει να τροποποιηθούν με τέτοιο τρόπο έτσι ώστε να κάνουν χρήση του paravirtualization API. Το πλείστο μέρος του λειτουργικού συστήματος καθώς και η ολότητα τον λογισμικών που τρέχουν σε αυτό δεν χρειάζεται να τροποποιηθούν. Για την επικοινωνία με συσκευές εισόδου εξόδου το ίδιο το XEN χρησιμοποιεί μη τροποποιημένα drivers του εμπορίου ενώ παρέχει στα guest operating systems τη δυνατότητα χρήσης αυτών των συσκευών μέσο generic drivers για κάθε τύπο συσκευής.

Το VirtualBox τρέχει σαν μια συνηθισμένη εφαρμογή πάνω από ένα λειτουργικό σύστημα. Αυτή εμπερικλείει μια εφαρμογή διαχείρισης των VMMs και ένα driver. Ως αποτέλεσμα αυτό το driver τρέχει στο ring 0 της αρχιτεκτονικής x86. Το Virtual Box τρέχει τα guest operating systems στο ring 1. Όλα τα αιτήματα εκτέλεσης κάποιου privileged instruction που προέρχονται από αυτά «παγιδεύονται» από το Virtual Box και προσομοιώνονται από το driver [6]. Τα λογισμικά προγράμματα που τρέχουν εντός των guest operating systems εκτελούνται κανονικά στο ring 3.

Λαμβάνοντας λοιπόν υπόψη αυτή την εξέγερση στην τεχνολογία των υπολογιστών σήμερα και κάποια βασικά χαρακτηριστικά των virtual machines αυτό που θέλουμε να μελετήσουμε

είναι αν, αφού όπως αναφέρεται από πολλούς αρκετά από τα cores μιας μηχανής αφιερώνουν τον περισσότερο τους χρόνο στο να μην κάνουν κάτι, με το να προσθέσουμε στη μηχανή πολλά virtual machines, όπου για ένα οργανισμό αυτό θα σήμαινε ότι θα μπορούσε να έχει πάνω σε ένα φυσικό μηχάνημα τον webserver, email server, file server πληρώνοντας μεν κάποιο κόστος στην επίδοση των εφαρμογών, θα μπορούσε αυτό να φέρει κάποια ωφελήματα.

Πιο συγκεκριμένα παρατηρώντας τη συμπεριφορά μιας μηχανής με πολλούς πυρήνες στη οποία υπάρχουν περισσότερα από ένα virtual machines και μεταβάλλοντας τον αριθμό των threads κάθε εφαρμογής που εκτελείται ξεχωριστά στα VMs θα μπορέσουμε να συμπεράνουμε αν πράγματι το να απομονώνει κανείς τις εφαρμογές με αυτό τον τρόπο θα είχε κάποια ωφελήματα τα οποία αξίζουν το κόστος επίδοσης που επιφέρεται χρησιμοποιώντας τα VMs.

1.2 Περίγραμμα Εργασίας

Στο πρώτο κεφάλαιο παρουσιάσαμε το βασικό υπόβαθρο και την υποκίνηση που υπάρχει για τη χρήση του virtualization και πιο συγκεκριμένα των virtual machines γενικά. Στο δεύτερο κεφάλαιο της εργασίας αυτής θα παρουσιάσουμε τη σχετική δουλεία που αφορά μελέτες για τα κόστη επίδοσης εφαρμογών όταν χρησιμοποιούνται τα virtual machines. Στο τρίτο κεφάλαιο θα γίνει μια ιστορικοί αναδρομή καθώς και θα επεξηγηθούν διάφοροι ορισμοί όσον αφορά το virtualization. Στο τέταρτο κεφάλαιο θα γίνει μια παρουσίαση της πειραματικής μεθοδολογίας που ακολουθήσαμε για την σύγκριση των αποτελεσμάτων ενώ στο έκτο θα παρουσιάζουμε και θα αναλύσουμε τα πειραματικά αποτελέσματα αυτής της μελέτης. Στο έβδομο και τελευταίο κεφάλαιο θα παραθέσουμε τα τελικά συμπεράσματα και τις επιπλέον μελλοντικές εργασίες που μπορούν να γίνουν.

Κεφάλαιο 2

Σχετική Δουλειά

Όπως αναφέρεται στο [1], μετά από μια ποσοτική ανάλυση δύο αρκετά γνωστών hypervisors, του XEN και του KVM όσον αφορά επίδοση, performance isolation και το scalability των virtual machines κατέληξαν στα πιο κάτω συμπεράσματα. Όσο αφορά επίδοση στις δοκιμές CPU-intensive και kernel compilation παρατήρησαν ότι το XEN είναι πολύ κοντά με το Native Linux και με ελάχιστα μικρή διαφορά να ακολουθεί το KVM, όσον αφορά όμως δοκιμές που αφορούσαν I/O εντολές (κυρίως read&write) φάνηκε ότι το KVM είχε καλύτερη επίδοση. Στα πειράματα που αφορούν το performance isolation αναφέρουν ότι και τα δύο virtual machines παρουσιάζουν αρκετά καλό performance isolation, το καθένα βέβαια με τα υπέρ και τα κατά του. Τέλος, όταν εξέτασαν το scalability των virtual machines αυτό που παρατήρησαν στην περίπτωση του XEN είναι ότι αυξάνοντας εκθετικά τον αριθμό των guests στη μηχανή ο χρόνος που χρειαζόταν το kernel για να γίνει compile αυξανόταν γραμμικά, κάτι που δείχνει αρκετά καλό scalability για το XEN. Όσο αφορά την περίπτωση του KVM δεν κατάφεραν να δείξουν κάτι αφού κάθε φορά μετά από 4 guests που δημιουργούσαν όλο και κάποιο δεν ανταποκρινόταν.

Το σημαντικό κόστος που επιφέρουν στην επίδοση του συστήματος τα virtual machines είναι γενικά αποδεκτό από όλους. Είναι όμως σημαντικά και τα πλεονεκτήματα που παίρνει κανείς (flexibility, security, ease of configuration and management, reduction of cost). Στο [2] γίνεται μια μέτρηση και να ανάλυση στις επιδόσεις σε δύο virtual machine monitors ανοικτού πηγαίου κώδικα το XEN και το KVM χρησιμοποιώντας τις εφαρμογές Linpack, LMbench και IOzone, και παρέχουν μια ποσοτική και ποιοτική σύγκριση των δύο. Αυτό που απέδειξαν είναι ότι το context switching, η δημιουργία διεργασιών και οι λειτουργίες I/O είναι τα κύρια στοιχεία που συμβάλουν στο virtualization overhead.

Στο [60] μετά από διάφορα πειράματα που έγιναν αναφέρεται ότι σχετικά με το performance isolation ότι θα μπορούσε κανείς να πετύχει strong resource isolation σε οποιοδήποτε από τα 3 virtual machines που αναφέρουν (XEN, KVM, VMware) με πολλούς τρόπους, αλλά με ένα βαθμό δυσκολίας. Αν και οποιοδήποτε virtual machine από αυτά τα τρία θα μπορούσε να παρέχει το αναγκαίο isolation των πόρων, είναι και πάλι πολύ πιο δύσκολο να τροποποιήσει

κανείς ένα λειτουργικό γενικής χρήσης που να καθορίζει τα virtualization layers. Όσο αφορά το scalability τα virtual machines βοηθούν στην αύξηση της συνολικής μέγιστης απόδοσης ενός συγκεκριμένου server απαιτώντας όμως σημαντικές αλλαγές στο guest OS ή τη χρήση των virtualization instructions VT.

Οι Hai Jin, Wenzhi Cao, Pingpeng Yuan και Xia Xie έχουν παρατηρήσει χρησιμοποιώντας τα VSCBenchmark στο [61] ότι στην αρχή των εφαρμογών υπάρχει μια περίοδος κατά την οποία η απόδοση των virtual machine αλλάζει συνεχώς και είναι ασταθής. Αυξάνοντας όμως τον αριθμό των virtual machine στη μηχανή, παρατήρησαν ότι παρόλο που η συνολική απόδοση μειώνεται σε ρυθμούς ανάλογους του αριθμού των virtual machines που προσθέτουν κρατάει κάπως σταθερούς ρυθμούς. Σίγουρα όμως έρχεται ένα σημείο όπου για παράδειγμα η συνολική απόδοση 12 virtual machine μαζί είναι σχεδόν η ίδια με την περίπτωση που έχουν 2 virtual machine. Από την άλλη πλευρά όμως, όταν λειτουργούν δύο ή περισσότερα virtual machines, το ποσοστό επίδοσης του κάθε VM είναι σχεδόν το ίδιο όσο αφορά τη συνολική απόδοση της μηχανής.

Στο [62], ομάδα του VMware παρουσιάζει το VMmark, ένα σύνολο από benchmarks τα οποία αποτελούνται από βασικές εφαρμογές για ένα data center όπως ένας database server, ένας web server, ένας Java server κ.α. Σκοπός του είναι να τρέχει κάθε εφαρμογή παράλληλα σε διάφορα virtual machines έτσι ώστε να μπορεί κανείς να αξιολογήσει την απόδοση των virtual machine σε κάποια μηχανή. Χρησιμοποιώντας το VMmark έχουν αποδείξει τη σταθερότητα που υπάρχει στα virtual machines όπου έτρεχαν αντίστοιχα ένας mail server, ένας java server, ένας web server, ένας database server και ένας file server.

Η επίδοση του XEN εξετάστηκε διεξοδικά στο [7] με τη χρήση benchmarks, όπως τα SPEC INT2000 και SPEC WEB99, αλλά και τα OSDB-IR/OSDB-OLTOP (όπου OSDB το Open Source Database Benchmark suite, και τα Information Retrieval (IR) και On-Line Transaction Processing (OLTP) workloads αντίστοιχα). Στην περίπτωση του SPEC INT2000 παρατηρήθηκε ότι το Linux ως guest operating system στο XEN (XENOLINUX) είχε την ίδια επίδοση με το native Linux. Στα δύο database benchmarks παρατηρήθηκε μια μικρή μείωση της απόδοσης της τάξης του 8% στο πρώτο benchmark (OSDB-IR) σε σχέση με το native και μια σχετικά μικρή μείωση της τάξης του 4.7% στο δεύτερο benchmark (OSDB-OLTOP) σε σύγκριση πάντα με τις εγγραφές στο native. Στα δύο τελευταία benchmark, τα οποία εξετάζουν την επίδοση (throughput) δικτύου βλέπουμε μια αποδεκτή μείωση του 4.3% ενώ στην περίπτωση του SPEC WEB99 φαίνεται μια ελάχιστη μείωση του 1%. Αυτά

επιδεικνύουν το σχετικά χαμηλό overhead του XEN πράγμα το οποίο επιτρέπει σε κάποιο να υποστηρίξει ότι το XENOLINUX και το native Linux έχουν «πρακτικά ισοδύναμη απόδοση» [7].

Σε αντιδιαστολή πειραματική αξιολόγηση που έχει γίνει στο [8] και συμπεριέλαβε όλα τα VMMs που τρέχουν ως συνηθισμένη εφαρμογή (όπως το VirtualBox, VMWare Player), έχουν συμπεράνει ότι το overhead αυτών των VMMs κυμαίνεται από 10-35%. Η αξιολόγηση αυτή έγινε μετρώντας την επίδοση του CPU, του file I/O και του network bandwidth ενός Linux guest operating system σε σύγκριση με ένα native Linux. Τα πειράματα αυτά εκτελέστηκαν πάνω σε μία dual-core μηχανή.

Η εργασία αυτή έχει σαν στόχο της την πειραματική αξιολόγηση του συνολικού overhead των VMM κατά την εκτέλεση παράλληλων εφαρμογών καθώς μεταβάλλεται ο αριθμός των threads. Αυτό το επιτυγχάνουμε χρησιμοποιώντας το Oracle VirtualBox και μερικές από τις εφαρμογές του PARSEC Benchmark Suite. Σκοπός μας ήταν να καταφέρουμε να δείξουμε ότι τα virtual machines μέσο του program isolation που παρέχουν μπορούν, μειώνοντας κυρίως το context switching να επιφέρουν καλύτερους χρόνους εκτέλεσης στις εφαρμογές πετυχαίνοντας ψηλό CPU utilization.

Κεφάλαιο 3

Virtualization

3.1 Ορισμοί του Virtualization

Το Virtualization είναι μια απλουστευμένη λύση η οποία λειτουργεί σαν ένα αφηρημένο στρώμα πάνω από το υλικό του Η.Υ. το οποίο κάνει πιο εύκολη τη διαχείριση και την αλληλεπίδραση των πόρων της μηχανής [9]. Το Virtualization μπορεί επίσης να οριστεί σαν η δημιουργία ενός εικονικού αντίγραφου μιας συσκευής ή ενός πόρου μέσω του οποίου να μπορούν να δημιουργούνται διάφορα execution environments [10]. Σκοπός της δημιουργίας των execution environments είναι να μπορούν στη συνέχεια να αντιμετωπίζονται σαν με τον συνηθισμένο τρόπο. Σκεφτείτε για παράδειγμα το partitioning ενός δίσκου, όπου αποτελεί μια μορφή virtualization, αφού μπορούμε και δημιουργούμε πολλά partitions από έναν και μοναδικό πόρο.

Στο [11], το virtualization είναι μία από τις πολλές, και ίσως η πλέον πρόσφατη τεχνολογική πρόοδος που προσθέτει ένα υψηλότερο επίπεδο εφευρετικότητας στα συστήματα και δίνει την ευκαιρία στους IT να επιτύχουν στην εργασία τους ακόμη πιο υψηλή παραγωγικότητα όσον αφορά τους υπολογιστές.

Όπως αναφέρεται στο [12], το virtualization είναι η τέλεια λύση προκειμένου να αντικατασταθεί μεγάλος αριθμός των serves που χρησιμοποιείται για τις απαιτήσεις των data centers, δεδομένου ότι για κάθε διαφορετική εφαρμογή χρειάζεται και ένας server. Αξιοποιώντας τη δυνατότητα που προσφέρει το virtualization να τρέχει πολλαπλά λειτουργικά συστήματα σε ένα μόνο μηχάνημα, τα κέντρα δεδομένων μπορούν να έχουν πιο λίγους servers και να ικανοποιούν τις ανάγκες τους όπως και προηγουμένως. Αυτό κατέστη δυνατό λόγω του ότι με το virtualization είναι δυνατόν να τρέχουν διάφορα OS και διαφορετικές εφαρμογές στον ίδιο server, το καθένα μέσα στο δικό του απομονωμένο εικονικό περιβάλλον, δηλαδή χωρίς να επηρεάζεται από τα υπόλοιπα.

3.2 Ιστορία του Virtualization

Η πρώτη εφαρμογή του Virtualization έλαβε χώρα στα τέλη της δεκαετίας του 1960 από την IBM [13, 14, 15]. Η IBM ήθελε να «διαμελίσει» τους κεντρικούς υπολογιστές (mainframe computers) σε ξεχωριστά virtual machines. Με το διαμελισμό αυτό, ήθελαν να επιτύχουν καλύτερο utilization από πλευράς του υλικού και να κάνουν τα mainframes πιο αποτελεσματικά δημιουργώντας πολλά λογικά partitions. Σε κάθε partition θα υπήρχε ένα λειτουργικό σύστημα και όλα τα partitions θα μπορούσαν θα λειτουργούν ταυτόχρονα καθώς θα βρίσκονταν πάνω στο ίδιο mainframe. Αν λάβουμε υπόψη το τεράστιο κόστος των mainframes τότε, είναι εύκολο να συνειδητοποιήσει κανείς ότι αξιοποιώντας όσο το δυνατό περισσότερο τους φυσικούς πόρους του συστήματος, μπορούσαν να εξοικονομήσουν αρκετά χρήματα.

Μεταξύ το 1980 και το 1990, το virtualization στα mainframes άρχισε να χάνει έδαφος, και ο κύριος λόγος ήταν η εμφάνιση των κατανεμημένων υπολογιστών. Στα κατανεμημένα συστήματα οι υπολογιστικές μονάδες που χρησιμοποιούνταν ήταν φθηνοί servers και desktop computers, αντί των ακριβών mainframes. Όμως ενώ η τεχνολογία στους servers και στα desktop εξελισσόταν, υπήρχαν πολλά προβλήματα που έπρεπε να λυθούν, αφού αυτές οι μηχανές δεν είχαν σχεδιαστεί για να υποστηρίζουν πλήρως το virtualization, σε αντίθεση με τα mainframes. Παρουσίαζαν προβλήματα που σχετίζονται με τη χαμηλή αξιοποίηση του υλικού καθώς και την αύξηση στο κόστους του, αύξηση σε κόστη που είχαν να κάνουν με τη διαχείριση τους από τους ITs, την ανεπαρκής στήριξη για προστασία μετά από ένα failover ή ένα disaster, την υψηλή συντήρηση τους και πολλά άλλα που έπρεπε όμως να ξεπεραστούν [16].

3.3 Hardware Virtualization

Το 1974 ο Gerald J.Popek και ο Robert P.Goldberg δημιούργησαν μια προδιαγραφή για το virtualization που ονόμασαν 'Formal Requirements for Virtualizable Third Generation Architectures' [17]. Σύμφωνα με το εν λόγω εγχειρίδιο, η x86 αρχιτεκτονική του επεξεργαστή δεν πληρούσε όλες τις απαιτήσεις. Γι' αυτό και οι προγραμματιστές είχαν δυσκολίες στο να υλοποιήσουν ένα virtual machine platform στην αρχιτεκτονική x86 και αντιμετώπιζαν σημαντική επιβάρυνση στην επίδοση σε σχέση με την τις εκτελέσεις που γίνονταν χωρίς τα virtual machine.

Το 2005 και το 2006 η Intel [18, 19], καθώς και η AMD [20], έδωσαν την απάντηση σε αυτό το πρόβλημα με τη δημιουργία των 'Processor Extension' στην αρχιτεκτονική x86. Παρά το γεγονός ότι τα 'Processor Extension' της υλοποίησης της AMD και αυτή της Intel διέφεραν, και οι δύο οργανισμοί κατάφεραν να πετύχουν τον ίδιο στόχο. Επιτρέπουν σε ένα virtual machine hypervisor να τρέχει ένα λειτουργικό σύστημα χωρίς τροποποιήσεις και το πιο σημαντικό χωρίς να προσθέτουν διάφορες κυρώσεις σε ότι αφορά την επίδοση του emulation.

*(**A**) AMD Virtualization (AMD-V)*

Η AMD στρέφει τις επεκτάσεις για το virtualization της στην αρχιτεκτονική x86 64-bit, που ονομάζει AMD Virtualization (AMD-V) [20]. Το 2006, η AMD κυκλοφόρησε τον επεξεργαστή Athlon 64 ("Orleans"), τον Athlon 64 X2 ("Windsor") και τον Athlon 64 FX ("Windsor") σαν τους πρώτους επεξεργαστές της AMD οι οποίοι υποστήριζαν αυτή τη τεχνολογία. Οι AMD Opteron και PhenomII επεξεργαστές, υποστηρίζουν την τεχνολογία του hardware virtualization που ονομάζεται Rapid Virtualization Indexing, όπου αργότερα υιοθετήθηκε από την Intel ως Extended Page Tables (EPT).

*(**B**) Intel Virtualization Technology for x86 (Intel VT-x)*

Αρχικά ονομαζόμενη σαν "Vanderpool", το VT-x αντιπροσωπεύει την τεχνολογία της Intel για το virtualization στην πλατφόρμα x86 [18, 19]. Η Intel περιλαμβάνει την Extended Page Tables (EPT), μια τεχνολογία για page-table virtualization, στην αρχιτεκτονική Nehalem. Από το 2009 το VT-x δεν υποστηρίζεται από όλους τους τελευταίους επεξεργαστές της Intel.

Ορισμένοι επεξεργαστές της Intel που υποστηρίζουν το VT-x είναι: Pentium 4, Xeon 3300 και +, 5000, 7000 series, Pentium Dual-Core E6300, E6500, E6600, Celeron SU2300, E3200, E3300, E3400.

Ορισμένα λογισμικά που χρησιμοποιούν αυτά που προσφέρουν οι AMD-V ή/και Intel VT-x είναι τα ακόλουθα: VirtualBox [21], XEN [22], VMware ESX Server [23], Hyper-V [24], Microsoft Virtual Server [25], VM Oracle [26], xVM Hypervisor [27], Windows Virtual PC [28].

3.4 Software Virtualization

Το 1999, η εταιρία VMware παρουσίασε την ιδέα του full virtualization για την αρχιτεκτονική x86 [29], προκειμένου να επιλύσει τα προαναφερθέντα προβλήματα. Με αυτή της την εισαγωγή κατάφερε να μεταφέρει τα συστήματα αρχιτεκτονικής x86 σε ένα πιο υψηλό επίπεδο επιτρέποντας έτσι στο υλικό των υπολογιστών να είναι shared σε ένα πλήρως απομονωμένο περιβάλλον, παρέχοντας mobility και δίνοντας την δυνατότατα επιλογής κάποιου λειτουργικού συστήματος για το περιβάλλον κάποιας εφαρμογής. Η VMware παρείχε μια πλατφόρμα όπου είναι δυνατόν να έχουμε high-performance virtual machines χωρίς να χάνουμε τη συμβατότητα με το υλικό καθώς επίσης και με το λογισμικό.

Άλλα παραδείγματα [30] του x86 virtualization software περιλαμβάνουν:

- Προϊόντα της Microsoft για Windows, όπως Microsoft Virtual PC [31], Hyper-V [32], και Microsoft Virtual Server [33], τα οποία βασίζονται σε τεχνολογίες του Connectix [34]
- Προϊόντα ανοικτού πηγαίου κώδικα, όπως το QEMU [48], Kernel-based Virtual Machine (KVM) [35] και VirtualBox [36]

Ένα άλλο παράδειγμα του software virtualization προέρχεται από τα συστήματα έρευνας Denali [37], L4 [38], και XEN [22], τα οποία παρέχουν high-performance virtual machines για τα συστήματα αρχιτεκτονικής x86 υλοποιώντας ένα virtual machine το οποίο δεν είναι ‘συμβατό’ με το υλικό. Αυτό το είδος του virtualization ονομάζεται paravirtualization. Τα virtual machines που δημιουργηθήκαν, δεν περιελάμβαναν υλοποίηση του instruction set της αρχιτεκτονικής x86, το οποίο είναι και δύσκολο να γίνει virtualize. Αυτή η μέθοδος όμως προϋποθέτει ότι το host-system(το σύστημα που θα εγκατασταθεί το virtual machine) υποστηρίζει hardware-assisted virtualization, όπως το Intel VT [39] και το AMD-V [40].

3.5 Hardware Virtualization VS Software Virtualization

Στο [41], οι Adams και Agesen από την ομάδα του VMware παρουσίασαν μια σύγκριση από τεχνικές software και hardware που σχετίζονται με το x86 virtualization. Μετά που η AMD και η Intel έβαλαν τις προεκτάσεις στην αρχιτεκτονική τους, προκειμένου να υποστηρίζουν άμεσα το virtualization στο hardware, οι συγγραφείς ήθελαν να δουν πιο από τα δύο δίνει καλύτερη απόδοση, ένα software ή ένα hardware VMM. Στη μελέτη τους περιελάβανε architectural-level events, όπως page table updates, context switches και διεργασίες που έχουν να κάνουν με I/O. Τα αποτελέσματα εξέπληξαν τους συγγραφείς όταν διαπίστωσαν ότι στην περίπτωση που υπάρχει φόρτος εργασίας όταν εκτελούνται εντολές I/O, η δημιουργία διαδικασιών ή το γρήγορο context switching, το software VMM αποδίδει καλύτερα από το hardware VMM. Στην περίπτωση όμως που η εφαρμογή έχει πολλά system calls τότε αυτό που έχει την καλύτερη απόδοση είναι το hardware VMM. Τέλος σε περιπτώσεις όπου οι εφαρμογές είναι compute-bound, και τα software αλλά και τα hardware VMMs αποδίδουν πολύ καλά. Όπως λένε οι συγγραφείς, ο λόγος που το hardware VMM δεν έχει καλύτερες επιδόσεις από το software σε όλες τις περιπτώσεις, οφείλεται στο γεγονός ότι δεν υποστηρίζει MMU virtualization από μόνο του και επειδή δεν μπορεί να συνυπάρχει με τεχνικές λογισμικού που υποστηρίζουν το MMU virtualization.

3.6 Τύποι Virtualization

Μετά από μια γρήγορη ανασκόπηση στο διαδίκτυο, μπορούμε να δούμε ότι υπάρχουν πολλά είδη virtualization τα οποία είναι αρκετά για να μας προκαλέσουν σύγχυση σχετικά με το ποια είναι πιο κατάλληλα για εμάς. Οι πιο κοινοί τύποι virtualization που εφαρμόζονται στα data centers είναι το *Server* και *Storage Virtualization*.

3.6.1 Server Virtualization

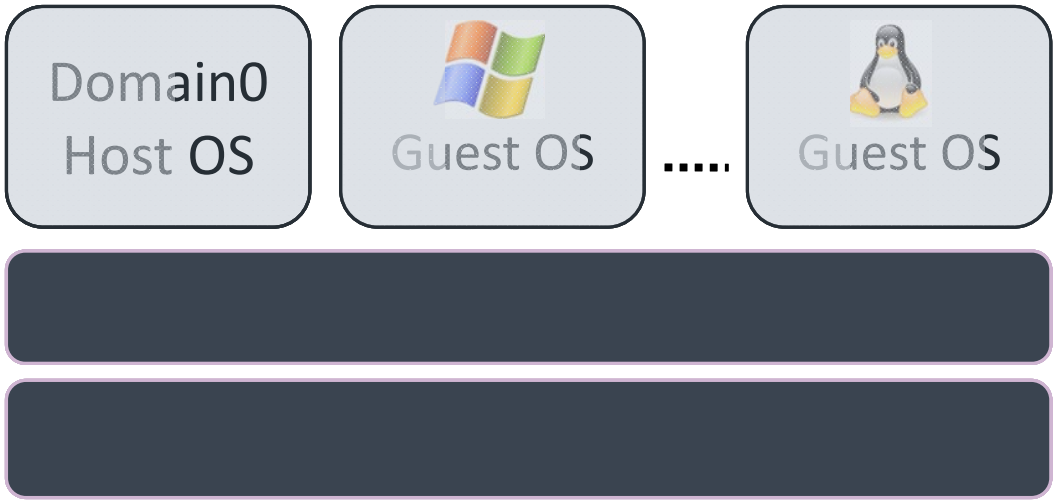
Αυτός ο τύπος virtualization έχει ως στόχο να κρύβει τους πόρους κάποιου server από τους χρήστες του έτσι ώστε να μην χρειάζεται να κατανοήσουν και να διαχειρίζονται τις πολύπλοκες λεπτομέρειες της λειτουργίας των πόρων του. Με αυτό τον τρόπο γίνεται δυνατή η αύξηση του διαμοιρασμού των πόρων και καλύτερη αξιοποίηση τους. Στο server virtualization διακρίνουμε τέσσερις τύπους του:

i. Operating System Virtualization

Το Operating System (OS) virtualization είναι αυτό που τρέχει πάνω από ένα υπάρχον λειτουργικό σύστημα, το σύστημα υποδοχής (host OS), και παρέχει ένα σύνολο βιβλιοθηκών που αλληλεπιδρούν με τις εφαρμογές, δίνοντας τους την ψευδαίσθηση ότι τρέχουν σε μια μηχανή η οποία είναι αφιερωμένη γι' αυτές. Από την πλευρά των εφαρμογών, μπορούν να βλέπουν και να αλληλεπιδρούν μόνο με τις εφαρμογές που τρέχουν στο ίδιο virtual OS. Φυσικά ο τρόπος που αλληλεπιδρούν είναι σαν να έχουν τον αποκλειστικό έλεγχο των πόρων μιας 'κανονικής' μηχανής αλλά βέβαια χωρίς να μπορούν να δουν τις εφαρμογές ή τους πόρους κάποιου άλλου λειτουργικού συστήματος που βρίσκεται σε ένα άλλο virtual machine. Μερικές από τις εταιρίες που προσφέρουν operating system virtualization είναι η SWSoft [42], η οποία προσφέρει το εμπορικό προϊόν Virtuozzo [43], καθώς και το λειτουργικό σύστημα ανοικτού πηγαίου κώδικα OpenVZ [44].

ii. Hardware Emulation (Hypervisors)

Στην τεχνική αυτή, έχουμε ένα εικονικό περιβάλλον υλικού το οποίο παρουσιάζεται από το virtualization software, τον hypervisor, και το guest OS τρέχει στην κορυφή του εν λόγω περιβάλλοντος. Το εικονικό αυτό περιβάλλον ονομάζεται VMM. Το guest OS τοποθετείται πάνω από το VMM και αλληλεπιδρά με αυτό. Και τα δύο μαζί, ως μια ολότητα, μπορούν να μετακινηθούν από μια φυσική μηχανή σε μια άλλη. Το hypervisor βρίσκεται μεταξύ του VMM και του υλικού και βοηθά στην μεταξύ τους επικοινωνία. Όταν το VMM στέλνει ένα system call, μεταφράζετε από τον hypervisor που το μεταφέρει σε συγκεκριμένο πόρο στο υλικό της μηχανής.



Σχήμα.1. Virtual Machine Monitor - Hypervisor (Layers)

Αυτός ο τύπος virtualization προσφέρει isolation σε κάθε guest OS ακόμα και στην περίπτωση όπου έχουμε πολλά λειτουργικά συστήματα να τρέχουν, το καθένα στο δικό του VMM. Με τη μέθοδο αυτή μπορούμε να έχουμε πολλά λειτουργικά συστήματα να τρέχουν την ίδια ώρα και παράλληλα να είναι εντελώς διαφορετικά μεταξύ τους. Η VMware (VMware Server και ESX Server) και η Microsoft (Virtual Server) είναι οι εταιρείες που προσφέρουν τα αυτοαποκαλούμενα hardware emulation virtualization software. Το XEN είναι μια hypervisor-based εφαρμογή ανοικτού πηγαίου κώδικα.

iii. Paravirtualization

Στο paravirtualization αντί να έχουμε το emulation ολόκληρου του υλικού, έχουμε ένα λεπτό στρώμα το οποίο συνδέει το φιλοξενούμενο λειτουργικό σύστημα με το υλικό της μηχανής από κάτω. Σε αντίθεση το Hardware Emulation εισάγει ένα ολόκληρο επίπεδο προσομοίωσης του υλικού που επιτρέπει σε ένα φιλοξενούμενο λειτουργικό σύστημα να έχει πρόσβαση στους πόρους του υλικού μιας μηχανής, αποτρέποντας όμως σε όλα τα άλλα λειτουργικά συστήματα να έχουν πρόσβαση στους ίδιους πόρους την ίδια στιγμή. Εδώ να αναφέρουμε ακόμη ότι στο paravirtualization το λειτουργικό σύστημα που φιλοξενείται πρέπει να έχει τη δυνατότητα να αναγνωρίζει ότι τρέχει πάνω από ένα virtual machine και να το διαχειρίζεται ανάλογα.

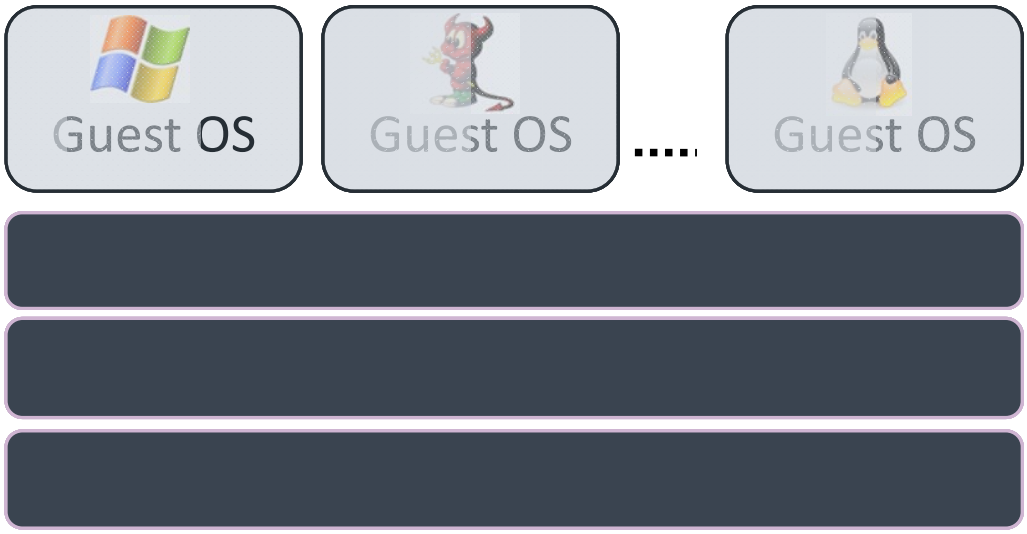
Ένα παράδειγμα εφαρμογής paravirtualization είναι το XEN, το οποίο χρηματοδοτείται από μια εμπορική εταιρεία με την επωνυμία XenSource. Ένα άλλο παράδειγμα είναι το Virtual Irons [84], μια εφαρμογή βασισμένη στο XEN.

iv. Virtual Machines

Η χρήση των Virtual Machines είναι αυτό που έρχεται αμέσως στο μυαλό μας μόλις ακούμε τον όρο ‘virtualization’. Μέσα σε κάθε Virtual Machine να υπάρξει ένα οποιοδήποτε λειτουργικό σύστημα, άσχετο με αυτό της μηχανής και το καθένα με τις δικές του εφαρμογές.

Δύο παραδείγματα είναι το VMware ESX και το xVM Server τα οποία τρέχουν ως η κύρια εφαρμογή (αποτελούν δηλαδή ένα τροποποιημένο kernel) σε ένα σύστημα, με διάφορα λειτουργικά συστήματα να τρέχουν από πάνω τους. Το xVM VirtualBox παρέχει στους προγραμματιστές έναν τρόπο να δημιουργούν διάφορα guest OS στα laptop ή τα workstation τους.

Ο πρώτος ορισμός για το virtual machine δόθηκε από τους Popek και Goldberg [17], ως "ένα αποτελεσματικό, απομονωμένο αντίγραφο μιας πραγματικής μηχανής". Όπως αναφέρεται στο [29], ένα virtual machine είναι ένα πακέτο λογισμικού που είναι καλά απομονωμένο και μπορεί να τρέξει ένα λειτουργικό σύστημα με τον ίδιο τρόπο που το κάνει μια φυσική μηχανή. Κάθε virtual machine έχει τους δικούς της εικονικούς πόρους, όπως το CPU, τη μνήμη RAM, το σκληρό δίσκο και κάρτα δικτύου, και συμπεριφέρεται ακριβώς όπως μια κανονική μηχανή - ένας υπολογιστής. Με βάση το γεγονός ότι ένα λειτουργικό σύστημα δεν μπορεί να ξεχωρίσει αν τρέχει πάνω σε ένα virtual machine ή σε ένα κανονικό μηχάνημα, και σε συνδυασμό με το γεγονός ότι το virtual machine είναι εξ ολοκλήρου κατασκευασμένο από λογισμικό χωρίς καθόλου υλικό, μπορούμε να πούμε ότι τα virtual machines έχουν πολλά πλεονεκτήματα σε σχέση με τις κανονικές μηχανές.



Σχήμα.2. Normal Virtual Machine Monitor (Layers)

Σε γενικές γραμμές, ένα virtual machine (VM) είναι ένα περιβάλλον, συνήθως ένα πρόγραμμα ή ένα λειτουργικό, το οποίο δεν υπάρχει στην πραγματικότητα αλλά δημιουργείται μέσα σε ένα άλλο φυσικό περιβάλλον. Σε αυτή την περίπτωση, το πρόγραμμα ή το λειτουργικό που τρέχει μέσα στο VM ονομάζεται «guest», ενώ το περιβάλλον που τρέχει το VM ονομάζεται «host». Συνήθως τα virtual machines δημιουργούνται συχνά για να εκτελέσουν ένα σύνολο από εντολές που διαφέρουν από εκείνες του περιβάλλοντος υποδοχής (host). Ένα περιβάλλον υποδοχής είναι πολύ πιθανό να τρέχει διάφορα VMs ταυτόχρονα. Επειδή τα VMs είναι διαχωρισμένα από τους φυσικούς πόρους μιας μηχανής που χρησιμοποιούν, το περιβάλλον υποδοχής είναι σε θέση να αναθέτει δυναμικά αυτούς τους πόρους μεταξύ τους.

Υπάρχουν δύο τύποι virtual machines ανάλογα με τη χρήση τους και τον τρόπο που αλληλεπιδρούν με την πραγματική μηχανή:

1. System Virtual Machine:

Το System VM, παρέχει μια πλήρη πλατφόρμα η οποία υποστηρίζει την εκτέλεση ενός ολοκληρωμένου λειτουργικού συστήματος (OS). Επιτρέπει την από κοινού χρήση των πόρων της φυσικής μηχανής που βρίσκεται από κάτω ανάμεσα στα διάφορα virtual machines, όπου το καθένα τρέχει το δικό του λειτουργικό σύστημα. Το στρώμα του λογισμικού που παρέχετε από το virtualization ονομάζεται virtual machine monitor ή hypervisor. Μερικά παραδείγματα των system virtual machines είναι: KVM [35], xVM [27], VirtualBox [36], VMware [29], XEN [22].

2. Process Virtual Machine:

Το Process VM, είναι σχεδιασμένο για να τρέχει ένα πρόγραμμα μόνο, πράγμα που σημαίνει ότι υποστηρίζει μόνο μία διαδικασία. Τρέχει σαν μια κανονική εφαρμογή μέσα σε ένα λειτουργικό σύστημα. Δημιουργείται όταν η διαδικασία που το αφορά ξεκινά και καταστρέφεται όταν η διαδικασία αυτή τερματίζει. Παρέχει ένα ανεξάρτητο περιβάλλον προγραμματισμού εντελώς αφαιρετικό από τις λεπτομέρειες του υλικού που βρίσκεται μέσα στη μηχανή και του λειτουργικού συστήματος στο οποίο τρέχει, επιτρέποντας έτσι σε ένα πρόγραμμα να μπορεί να εκτελείται σε οποιαδήποτε πλατφόρμα. Αυτός ο τύπος VM έχει γίνει δημοφιλές με τη γλώσσα προγραμματισμού Java, το οποίο υλοποιείται με το Java Virtual Machine (JVM).
Μερικά παραδείγματα από Process Virtual Machines είναι: Java Virtual Machine [25], το Adobe Flash Player - SWF [26], VX32 virtual machine [47], Common Language Infrastructure - C#, Visual Basic NET, J#, C++/CLI [48].

3.6.2 Storage Virtualization

Αυτός ο τύπος virtualization παίρνει όλους τους φυσικούς χώρους αποθήκευσης που ανήκουν σε διάφορες συσκευές αποθήκευσης μέσα σε ένα δίκτυο και μας δίνει την ψευδαίσθηση ότι είναι μια ενιαία συσκευή αποθήκευσης. Αυτή η συσκευή αποθήκευσης ελέγχεται από μια κεντρική κονσόλα. Το Storage Virtualization χρησιμοποιείται συνήθως σε Storage Area Networks (SAN) και Network Attach Storage (NAS). Μερικοί πωλητές συσκευών Storage Virtualization [49] είναι: 3PAR [50], Arrow ECS HP Group και Intel [51], Dell και VMware [52], IBM [53].

3.7 Τι μπορούμε να πετύχουμε με το Virtualization

Το Virtualization μας επιτρέπει να έχουμε δύο ή περισσότερα images από ένα πλήρες σύστημα που τρέχει δύο ή περισσότερα εντελώς διαφορετικά περιβάλλοντα, πάνω στο ίδιο σύστημα. Για παράδειγμα, με το virtualization, μπορούμε να έχουμε Linux και Windows στον ίδιο υπολογιστή.

Μπορούμε να πούμε ότι το virtualization απαλείφει τους χρήστες και τις εφαρμογές από τα ιδιαίτερα χαρακτηριστικά του υλικού μιας μηχανής που χρησιμοποιεί για την εκτέλεση υπολογιστικών καθηκόντων. Πρόκειται για μια αρκετά υποσχόμενη τεχνολογία και είναι πολύ χρήσιμο όταν πρόκειται για την αναβάθμιση κάποιου συστήματος, δεδομένου ότι μπορεί κανείς να συλλάβει την κατάσταση ενός VM και να τη μεταφέρει από ένα παλαιό σε ένα νέο σύστημα.

Το Virtualization είναι επίσης σχεδιασμένο για να καταστεί δυνατή η εξοικονόμηση ενέργειας στους υπολογιστές. Αν λάβουμε υπόψη το γεγονός ότι το virtualization βοηθά στο να μειώνει τον αριθμό των servers που χρειάζεται από κάθε data center, τότε το συνολικό κόστος της ενέργειας που εξοικονομείται για κάθε εταιρεία θα ήταν αρκετά λιγότερο.

Συνοψίζοντας, σύμφωνα με το [54] το virtualization μας βοηθά στο να δημιουργούμε ένα δυναμικό data center, μας βοηθά στην μείωση της κατανάλωσης ενέργειας, παρέχει καλύτερη ασφάλεια, συμβάλλει στην ανάπτυξη και δοκιμή νέων πραγμάτων εύκολα και γρήγορα, για την εκτέλεση πολλαπλών λειτουργικών συστημάτων στην ίδια μηχανή, για τη βελτίωση του scalability, για όσο το δυνατό καλύτερο utilization του υλικού, και πολλά άλλα.

Κεφάλαιο 4

Πειραματική Μεθοδολογία

4.1 Στόχοι Πειραμάτων

Όπως αναφέραμε και στην αρχή αυτό που θέλαμε να μελετήσουμε είναι η συμπεριφορά εφαρμογών όταν εκτελούνται σε πολλά virtual machines έτσι ώστε να τη συγκρίνουμε με εκτελέσεις χωρίς τα virtual machines. Αυτό που θέλουμε να μελετήσουμε είναι αν το program isolation που μας παρέχουν τα virtual machines μπορεί να βοηθήσει την εκτέλεση παράλληλων εφαρμογών, αφού απομονώνοντας την κάθε εφαρμογή στο δικό της virtual machine δεν έχουμε τόσο πολλή context switching και διάφορες συγκρούσεις με άλλες εφαρμογές. Επίσης με αυτό τον τρόπο μπορούμε να δούμε το scalability των virtual machines και το βαθμό σταθερότητας που μπορούν να μας παρέχουν.

Επιπλέον λόγω του ότι θέλαμε να μελετήσουμε αλλά πιο σημαντικά να κάνουμε μια σύγκριση σχετικά με τα κόστη στους χρόνους εκτέλεσης των εφαρμογών όταν αυτές εκτελούνται μόνες τους, χρειαστήκαμε κάποια πειράματα έτσι ώστε να πάρουμε τα αποτελέσματα τους. Με αυτά θα μπορέσουμε να διαχωρίσουμε τις εφαρμογές σε κατηγορίες αλλά και να δούμε κατά πόσο το overhead που προσθέτουν περισσότερα από ένα virtual machines στην ίδια μηχανή επηρεάζεται.

Για αυτούς τους σκοπούς λοιπόν αποφασίσαμε να εκτελέσουμε διάφορα είδη πειραμάτων. Οι στόχοι των πειραμάτων μπορούν να ομαδοποιηθούν σε τρεις (3) κατηγορίες:

- Cat1: Μελέτη του κόστους στο χρόνο εκτέλεσης των εφαρμογών όταν αυτά τρέχουν παράλληλα σε δύο VMs και σύγκριση τους όταν αυτά εκτελούνται παράλληλα χωρίς VMs σε υπολογιστή με πολλούς πυρήνες με σκοπό τη παρατήρηση αν το isolation που προσφέρουν τα VMs βοηθά, χωρίς να πληρώνουμε αρκετό κόστος επίδοσης.

- Cat2: Μελέτη του κόστους στο χρόνο εκτέλεσης των εφαρμογών όταν αυτά τρέχουν παράλληλα σε δύο VMs και σύγκριση τους όταν αυτά εκτελούνται παράλληλα χωρίς VMs σε υπολογιστή με πολλούς πυρήνες, μειώνοντας σε αυτή τη περίπτωση το συνολικό αριθμό των πυρήνων του συστήματος για να μελετήσουμε κατά πόσο αυτό επηρεάζει στο συνολικό κόστος επίδοσης.

- Cat3: Μελέτη του κόστους στο χρόνο εκτέλεσης των εφαρμογών όταν αυτά τρέχουν παράλληλα σε τέσσερα VMs και σύγκριση τους όταν αυτά εκτελούνται παράλληλα έξω από τα VMs σε υπολογιστή με πολλούς πυρήνες με σκοπό τη παρατήρηση αν το isolation που προσφέρουν τα VMs βοηθά αλλά επίσης και για να δούμε το stability και scalability περισσότερων virtual machines.

	Cat1	Cat2	Cat3
Αριθμός VMs	2	2	4
Cores ανά VM	4	2	2
Cores Μηχανής	8	4	8
Εφαρμογές	PARSEC	PARSEC	PARSEC
Threads	2-4-8	2-4	2-4

Πίνακας 1. Πίνακας περιγραφής κάθε κατηγορίας

4.2 Τρόπος Μέτρησης Του Χρόνου Εκτέλεσης

Για τη μέτρηση του χρόνου εκτέλεσης των εφαρμογών PARSEC στο host OS χρησιμοποιήσαμε την εντολή time του Linux. Η εντολή αυτή επιστρέφει τρείς χρόνους, το real, το user και το system. Ο χρόνος του real αντιστοιχεί στο συνολικό ***πραγματικό*** χρόνο εκτέλεσης, ενώ ο χρόνος user αντιστοιχεί στο συνολικό χρόνο που χρειάστηκαν τα processes/threads για να εκτελεστούν, και τέλος ο χρόνος system αντιστοιχεί στο διάστημα που χρειάστηκε η μηχανή για να ολοκληρώσει τα αιτημάτων της εφαρμογής. Να σημειώσουμε ότι στην περίπτωση που οι εφαρμογές τρέχουν σε πολυπύρηνα συστήματα, τότε τον ακριβή χρόνο εκτέλεσης μιας εφαρμογή μας τον δίνει η ένδειξη 'real'. Για τις μετρήσεις μας επομένως επιλέξαμε να παίρνουμε το αποτέλεσμα που μας επέστρεφε η εντολή time του real.

Τώρα όσο αφορά τη μέτρηση του χρόνου εκτέλεσης των εφαρμογών PARSEC στα VMs δεν είναι και τόσο απλό. Αυτό γιατί λόγω των πολλών στρωμάτων που υπάρχουν μεταξύ host και guest OS, είναι ευρέως αποδεκτό ότι η χρονομέτρηση μέσα στα virtual machines μπορεί να μην είναι ακριβές και ακόμη χειρότερα να είναι παραπλανητικές, αν δεν ληφθούν τα απαραίτητα μέτρα που ελήφθησαν υπόψη. Επειδή ο τρόπος που τα virtual machines δουλεύουν στο θέμα του χρόνου είναι βασισμένος στο να κάνουν share την ώρα που έχει το host OS από το υλικό της μηχανής, δεν μπορούν έτσι απλά να αντιγράφουν ακριβώς τη συμπεριφορά του χρόνου μιας μηχανής. Το VirtualBox παίρνει το local time του host και το χρησιμοποιεί σαν το «hardware-clock» για τα guests του. Σε ένα άρθρο του VMware [59], γίνεται μια πολύ καλή περιγραφή της χρονομέτρησης στο εσωτερικό των virtual machines.

Όπως αναφέρεται στο Κεφάλαιο 4, παράγραφος 3, στην περίπτωση μας με το να εγκαταστήσουμε στα virtual machines τα VirtualBox Guest Additions καταφέραμε να διασφαλίσουμε ότι ο χρόνος των guests είναι καλύτερα συγχρονισμένος με το χρόνο του host. Συνεπώς, τα πειράματα μας βασίζονται στο γεγονός ότι με τα VirtualBox Guest Additions εγκατεστημένα στα virtual machine, μειώνουμε τις πιθανότητες να μην παίρνουμε ακριβής μετρήσεις στους χρόνους εκτέλεσης των εφαρμογών. Έτσι, λαμβάνοντας υπόψη μας τα παραπάνω αποφασίσαμε ότι και στην περίπτωση της μέτρησης του χρόνου εκτέλεσης για κάθε εφαρμογή θα την κάνουμε χρησιμοποιώντας την εντολή κελύφους time του Linux.

4.3 Το Σύστημα που Χρησιμοποιήθηκε

Για τα πειράματα χρησιμοποιήθηκε το πιο κάτω σύστημα με τις ακόλουθες ρυθμίσεις:

IBM x3650 [3]: Ένας πολυπύρηνος υπολογιστής με 8-core, συγκεκριμένα με δύο 4-core Intel (R) Xeon (R) CPU E5320 επεξεργαστές στα 1,86 GHz. Οι επεξεργαστές αυτοί είναι ρυθμισμένοι με 128KB private L1 cache και 8MB shared L2 cache. Στο σύστημα αυτό τρέχει η 64-bit έκδοση του Ubuntu 9.04. Αυτό αντιπροσωπεύει και το σύστημα «υποδοχής» (host) στο οποίο αναφερθήκαμε στο Κεφάλαιο 3.
Και για τις τρείς (3) κατηγορίες χρησιμοποιήθηκε το πιο πάνω σύστημα σαν σύστημα «υποδοχής».

4.4 Εργαλεία που Χρησιμοποιήθηκαν (Virtual Machines)

Για το virtualization σε αυτή τη μελέτη χρησιμοποιήθηκε το VirtualBox από την Oracle [55]. Η έκδοση του VirtualBox που εγκαταστάθηκε είναι η 4.0 πάνω από Ubuntu 9.04 x64.

Το VirtualBox είναι μια συλλογή ισχυρών virtual machine tools. Είναι δωρεάν και μπορεί να χρησιμοποιηθεί για προσωπικούς υπολογιστές ή από τις επιχειρήσεις στους servers ή σε embedded systems. Υποστηρίζει αρκετά από τα γνωστά λειτουργικά που υπάρχουν σήμερα τόσο για σκοπούς host όσο και για guests. Είναι μια επαγγελματική λύση virtualization που έρχεται σε πολλές εκδόσεις ανάλογα για να ικανοποιήσει τις ανάγκες των χρηστών. Χρησιμοποιώντας το VirtualBox, μπορεί κανείς να κάνει virtualize 32-bit και 64-bit λειτουργικά συστήματα σε μηχανήματα με επεξεργαστές Intel και AMD, είτε χρησιμοποιώντας τα hardware virtualization features που παρέχονται από αυτούς τους επεξεργαστές ή μέσο λογισμικού [6]. Ο βασικός λόγος για τον οποίο επιλέξαμε το VirtualBox και όχι κάποιο άλλο εργαλείο για virtualization, οφείλεται στο γεγονός ότι στη συνέχεια θέλουμε να είναι εφικτό το profiling του κώδικα του VM. Για να γίνει αυτό, πρέπει να έχουμε πρόσβαση στον πηγαίο κώδικα από τον οποίο υλοποιείται το VM. Διαλέγοντας ένα δωρεάν εργαλείο, μπορούμε να το επιτύχουμε.

Αφού επιλέξαμε το εργαλείο, το πρώτο βήμα ήταν η δημιουργία των virtual machine μέσω του VirtualBox και στη συνέχεια η ρύθμιση των πόρων που θα μπορούσαν να χρησιμοποιήσουν. Η μνήμη RAM που δώσαμε για τα πειράματα σε κάθε VM ήταν 4096MB.

Η αριθμός των επεξεργαστών που διατέθηκαν σε αυτά ήταν για το Cat1 και Cat3 δύο (2 cores) και για το Cat2 τέσσερα (4 cores).
Σαν δεύτερο βήμα έπρεπε να εγκατασταθεί ένα λειτουργικό σύστημα σε κάθε virtual machine. Για να έχουμε δίκαιη σύγκριση, εγκαταστήσαμε στο VM το ίδιο λειτουργικό σύστημα με αυτό που είχαμε και στη μηχανή του συστήματος «υποδοχής». Έτσι καταλήξαμε να έχουμε Ubuntu 9.04 x64 στο VM. Το λειτουργικό σύστημα που τρέχει σε κάθε VM, είναι γνωστό ως το guest OS.

Το τρίτο βήμα ήταν να εγκατασταθούν τα VirtualBox Guest Additions για το Linux μέσα στο guest OS. Όπως αναφέρεται στο [6] τα VirtualBox Guest Additions κάνουν τη ζωή μας πιο εύκολη, παρέχοντας closer integration μεταξύ host και guest OS και βελτιώνουν την επίδοση των guest OS. Τα additions αυτά είναι σαν ένα σύνολο από device drivers και system applications που μπορούν να εγκατασταθούν σε κάθε guest OS. Μεταξύ άλλων πλεονεκτημάτων, οι προσθήκες παρέχουν τα εξής:

-*Shared Folders*: προσφέρουν την εύκολη ανταλλαγή αρχείων μεταξύ host και guest OS.

-*Mouse Pointer Integration*: με αυτό το driver έχουμε πλέον μόνο ένα mouse pointer και παραλλαζόμαστε από την ανάγκη να πατούμε το Host Key (right Ctrl) κάθε φορά που θέλουμε να ελευθερώσουμε το mouse από το VM.

-*Full Screen Setting*: επιτρέπει την πλήρης οθόνη.

-*Time Synchronization*: Αυτό εξασφαλίζει ότι το system time του guest OS είναι συγχρονισμένο με αυτό του host OS.

Τώρα αφού το virtual machine ετοιμάστηκε, έπρεπε να εγκατασταθούν τα απαραίτητα προγράμματα/βιβλιοθήκες για τις εφαρμογές που τρέχουν στα guest OS, καθώς επίσης και για το host OS. Τέλος, για ευκολία μας αφού τελείωσε η εγκατάσταση στα guest OS κάναμε τρία (3) clone images του υπάρχοντος VM τα οποία θα χρησιμοποιήσουμε για τα υπόλοιπα VMs.

4.5 Εφαρμογές που Χρησιμοποιήθηκαν

Οι εφαρμογές που χρησιμοποιήθηκαν για τα πειράματα μας αποτελούνται από πέντε προγράμματα από το PARSEC benchmark suite [4].

Το PARSEC (Princeton Application Repository for Shared-Memory Computers) είναι ένα δωρεάν benchmark suite ανοικτού πηγαίου κώδικα που αποτελείται από multithreaded προγράμματα με μεγάλο φόρτο εργασίας και είναι σχεδιασμένα με τέτοιο τρόπο ώστε να αντιπροσωπεύουν next-generation shared-memory προγράμματα για chip-multiprocessors [56]. Περισσότερες πληροφορίες για τα PARSEC BENCHMARKS μπορούν να βρεθούν στο [57] και [58]. Αυτό που κάνει τα PARSEC να διαφέρουν από τα άλλα πακέτα είναι ότι είναι παράλληλα, περιλαμβάνει τα αρχεία του φόρτου εργασίας της κάθε εφαρμογής, οι οποίες είναι πιθανό να γίνουν σημαντικές στο εγγύς μέλλον, και προσφέρει μια ευρεία επιλογή των από εφαρμογές. Επικεντρώνεται σε προγράμματα από διάφορους τομείς, όπως εφαρμογές για desktop και για server. Πιο συγκεκριμένα οι εφαρμογές προέρχονται από πολλούς τομείς, όπως η υπολογιστική όραση, κωδικοποίηση video, οικονομικές αναλύσεις, η φυσική κινούμενη εικόνα και την επεξεργασία εικόνας. Κάθε μια από τις εφαρμογές, έρχεται με προ-εγκατεστημένες τις ρυθμίσεις που θα καθορίσουν το συγκεκριμένο τρόπο που θα μεταγλωττιστούν. Εδώ να αναφέρουμε ότι δεν είναι όλες οι εφαρμογές που υποστηρίζουν τις προ-εγκατεστημένες αυτές ρυθμίσεις. Για τη μελέτη αυτή χρειαζόμαστε τις εφαρμογές που υποστηρίζουν το «gcc-pthreads». Έχοντας αυτό ως τη μόνη προϋπόθεση, επιλέξαμε τις πρώτες πέντε εφαρμογές που μπορέσαμε να οικοδομήσουμε με επιτυχία. Αυτές οι εφαρμογές είναι: Blackscholes, Bodytrack, Facesim, Fluidanimate και Raytrace. Σύντομη περιγραφή τους παρουσιάζεται στον πίνακα 2.

Εφαρμογή	***Περιγραφή***	***Δεδομένα Εισόδου***	***Όνομα Δεδομένων Εισόδου***
Blackscholes	Performs option pricing using the Black-Scholes partial differential equation(PDE) method	'native'	in_10M.txt
Bodytrack	Performs the tracking of people in security camera images	'native'	sequenceB_261
Facesim	Simulates the motions of a human face	'native'	Face_Data
Fluidanimate	Models the fluid dynamics for animation purposes using the Smoothed Partical Hydrodynamics (SPH) method	'native'	in_500K.fluid
Raytrace	Renders a 2D image out of a 3D model using the ray-tracing method	'native'	thai_statue.obj

Πίνακας 2.*PARSEC* Application Description

Αυτό που χρειαζόμασταν για τα πειράματα ήταν ένα σύνολο εφαρμογών το οποίο να αντιπροσωπεύει διάφορες κατηγορίες, όπως φαίνεται και στον πίνακα 3, αλλά και αντιπροσωπεύουν εφαρμογές οι οποίες είναι ρεαλιστικές και χρησιμοποιούνται σήμερα. Έτσι επιλέξαμε τις εφαρμογές που μας προσφέρει το PARSEC benchmark suite.

Εφαρμογή	Είδος Εφαρμογής	Σύνολο Δεομένων Εισόδου	Locks	Barriers	Conditions
Blackscholes	Financial Analysis	small	0	8	0
Bodytrack	Computer Vision	medium	114,621	619	2,042
Facesim	Animation	large	14,541	0	3,137
Fluidanimate	Animation	large	17,771,909	0	0
raytrace	Imaging	large	N/A	N/A	N/A

Πίνακας 3. PARSEC Application Characteristics

Οι εφαρμογές του PARSEC (έκδοση 2.1) που χρησιμοποιήθηκαν έχουν παραλληλοποιηθεί με POSIX threads (pthreads). Για τα πειράματα μας μεταγλωττίσαμε τις εφαρμογές χρησιμοποιώντας το parsecmgmt το οποίο είναι το βασικό εργαλείο που έρχεται με το PARSEC για την κατασκευή και εκτέλεση των εφαρμογών. Επίσης Χρησιμοποιήσαμε το gcc-pthreads για τη παράλληλη μεταγλώττιση των εφαρμογών. Ο μεταγλωττιστής ήταν ο gcc-3.4 και ο g++. Κατά την εκτέλεση των εφαρμογών χρησιμοποιήσαμε native δεδομένα εισόδου τα οποία αποτελούν μεγάλης κλίμακας πειράματα για μετρήσεις επίδοσης και έρευνας.

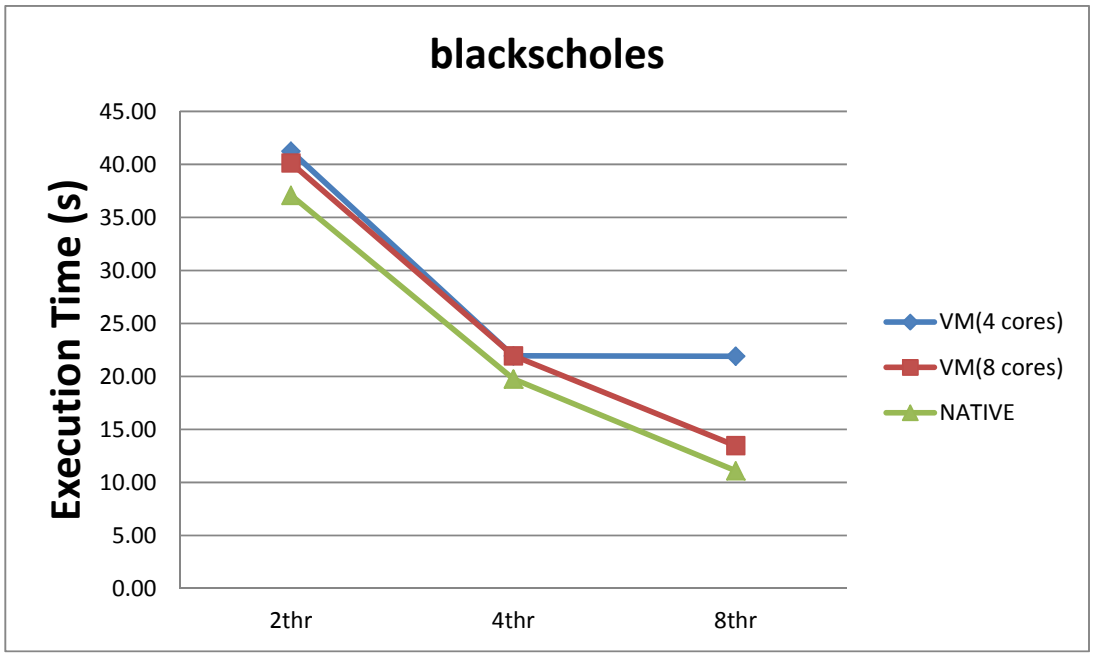
Κεφάλαιο 5

Πειραματικά Αποτελέσματα

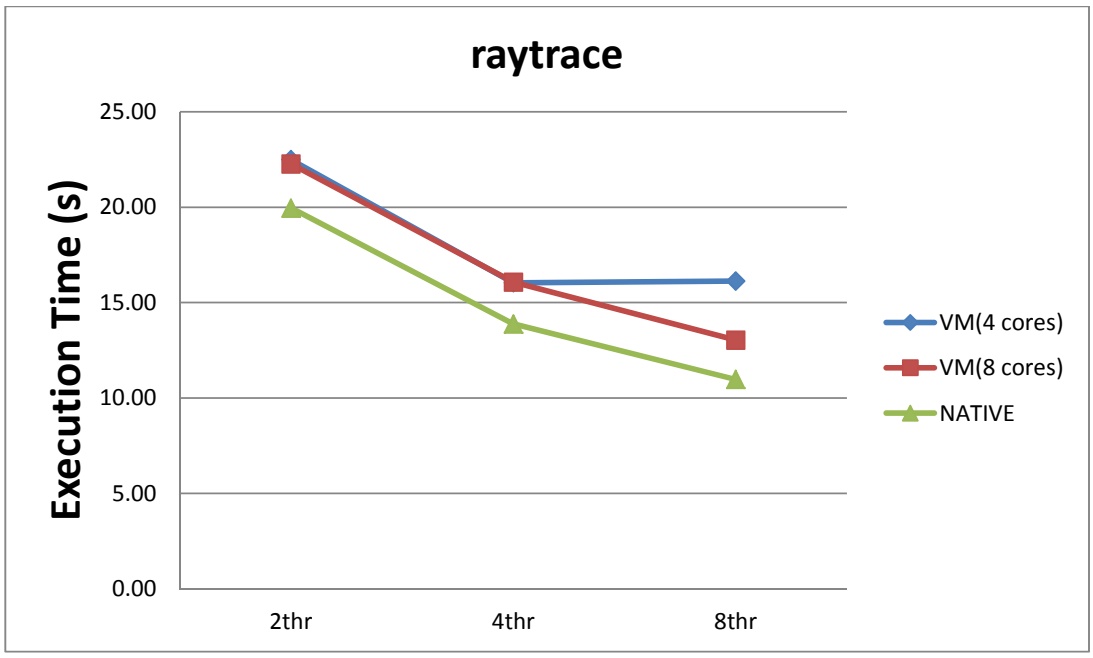
5.1 Αποτελέσματα Εφαρμογών σε ένα Virtual Machine

Πιο κάτω ακολουθούν τα αποτελέσματα που προαναφέραμε και αφορούν την εκτέλεση των εφημεριών που επιλέξαμε χρησιμοποιώντας μόνο ένα virtual machine στη μηχανή. Συγκεκριμένα θα παρουσιάσουμε τα αποτελέσματα των εκτελέσεων από virtual machine με 4 και 8 cores καθώς ο αριθμός των threads της εφαρμογής αλλάζει από δύο (2) μέχρι και οκτώ (8) threads και τα αντίστοιχα αποτελέσματα σε native περιβάλλον, χωρίς δηλαδή τα virtual machines. Κάθε φορά υπήρχε ένα VM ενεργοποιημένο το όποιο έτρεχε αποκλειστικά μια εφαρμογή από τα PARSEC benchmarks. Σκοπός μας είναι να δείξουμε το overhead που προκαλείται από τα VMs έτσι ώστε να συμπεράνουμε αν μπορούμε χρησιμοποιώντας περισσότερα από ένα virtual machine στην ίδια μηχανή να πετύχουμε καλύτερα αποτελέσματα.

Εδώ να αναφέρουμε ότι σε οποιαδήποτε από τις περιπτώσεις όπου μιλούμε για overhead ο τρόπος μέτρησης του γίνεται βάση του χρόνου εκτέλεσης που χρειάστηκε κάθε εφαρμογή σε κάθε περίπτωση ξεχωριστά. Αυτό ισχύει και για τις περιπτώσεις που έχουμε περισσότερα από ένα VMs χρησιμοποιώντας όμως το συνολικό χρόνο εκτέλεσης.



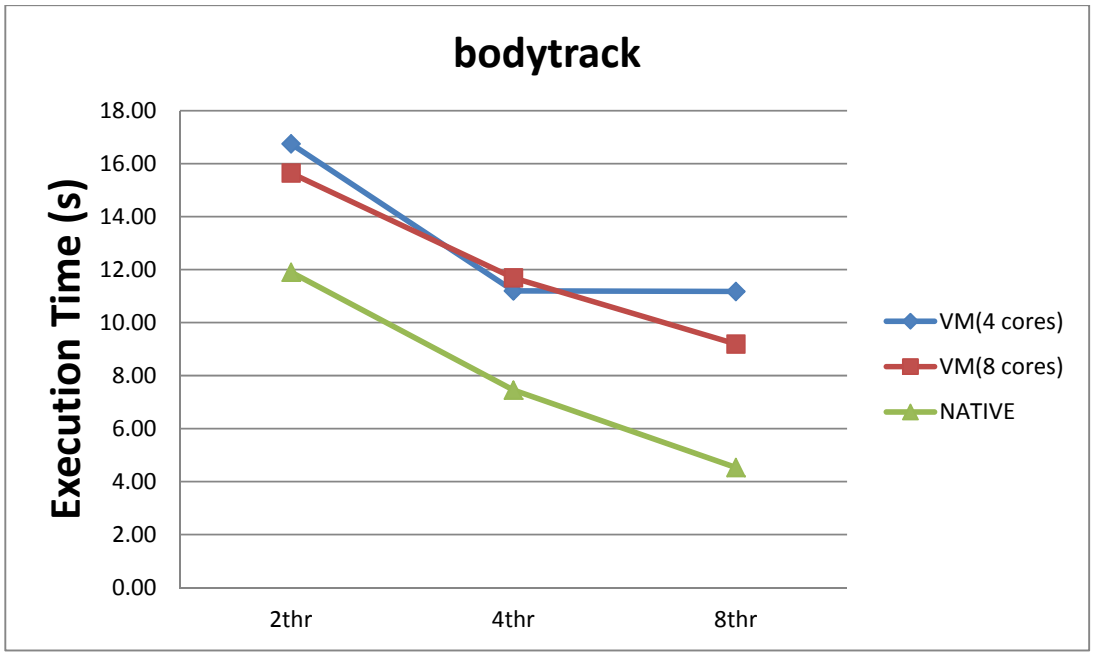
(a)



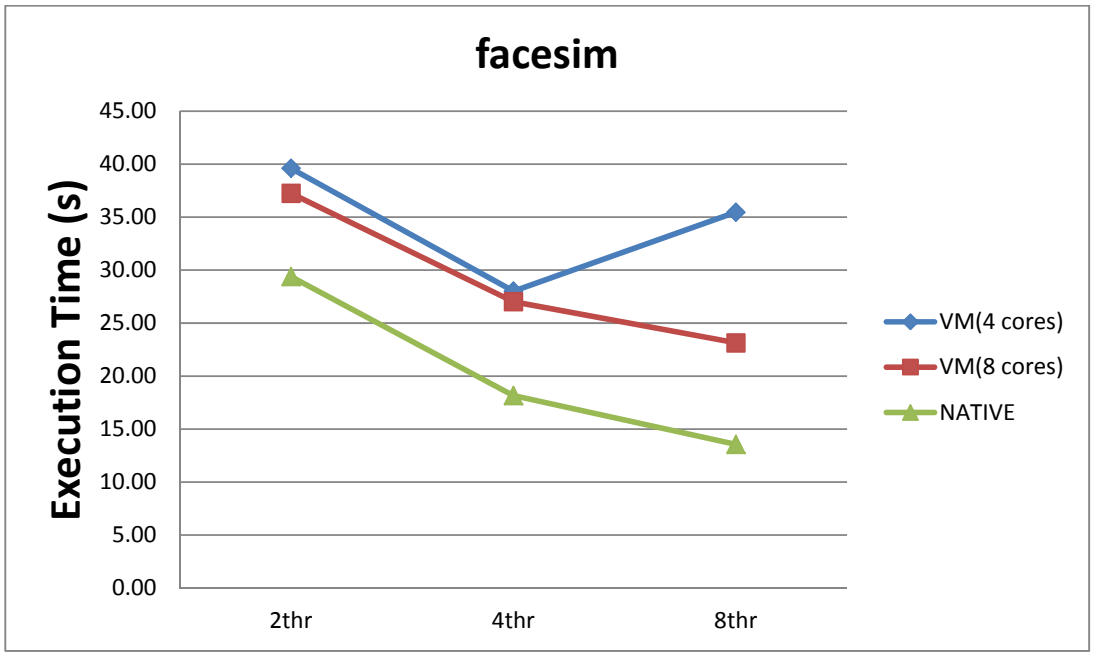
(b)

Σχήμα.3. execution times of blackscholes and raytrace

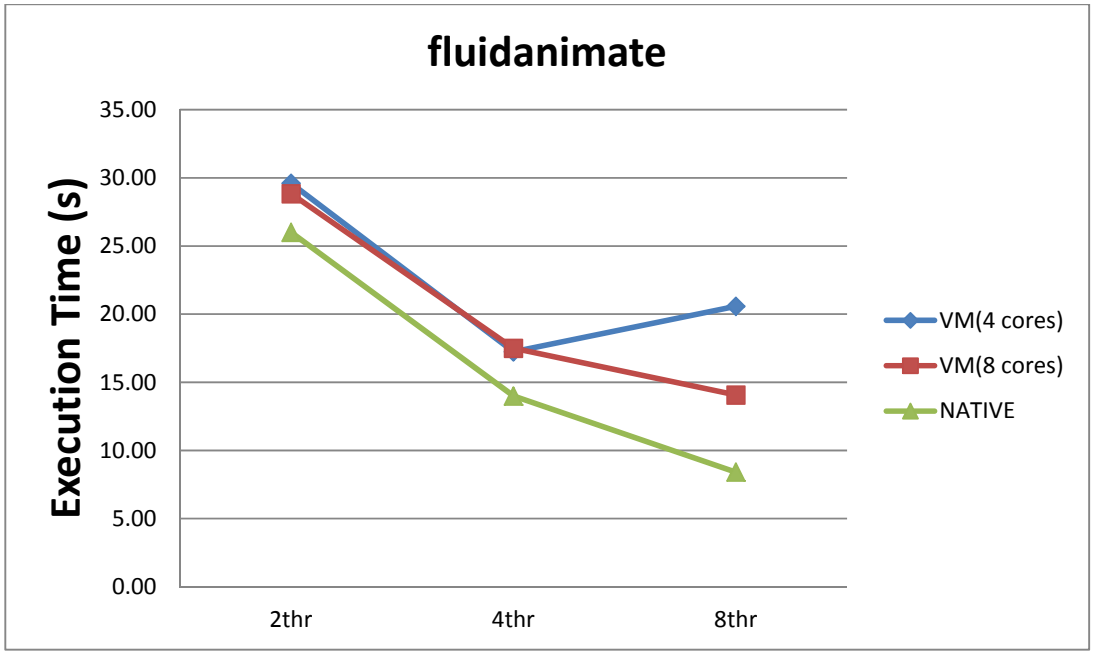
Αυτό που παρουσιάζει ενδιαφέρον από τις γραφικές παραστάσεις του Σχήμα.3.(a) & (b) είναι ότι οι χρόνοι εκτέλεσης χωρίς VM και αυτοί με virtual machine με 8 cores είναι πολύ κοντά καθώς ο αριθμός των threads αλλάζει. Επίσης σημαντικό είναι να προσέξουμε ότι στην περίπτωση του virtual machine με τα 4 cores ο χρόνος εκτέλεσης καθώς αλλάζουμε τον αριθμό των threads από 4 σε 8 παραμένει σχεδόν σταθερός. Αυτό είναι κάτι που περιμένουμε αφού το περιορίζει ο αριθμός των cores που είναι διαθέσιμα.



(a)



(b)



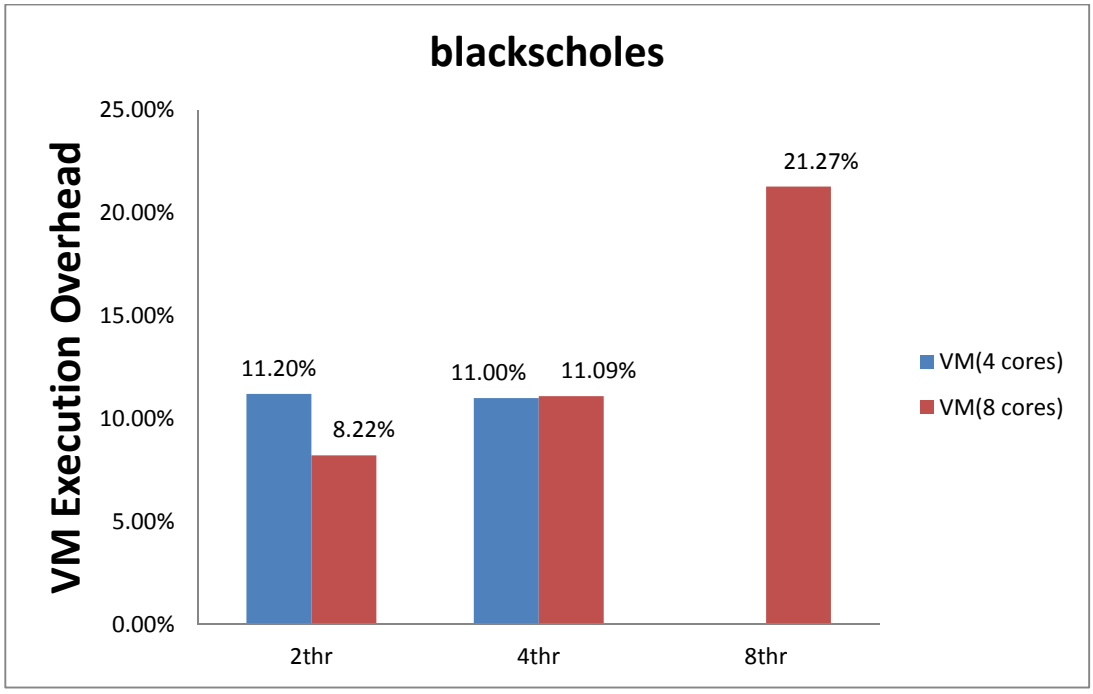
(c)

Σχήμα.4. execution times of bodytrack, facesim and fluidanimate

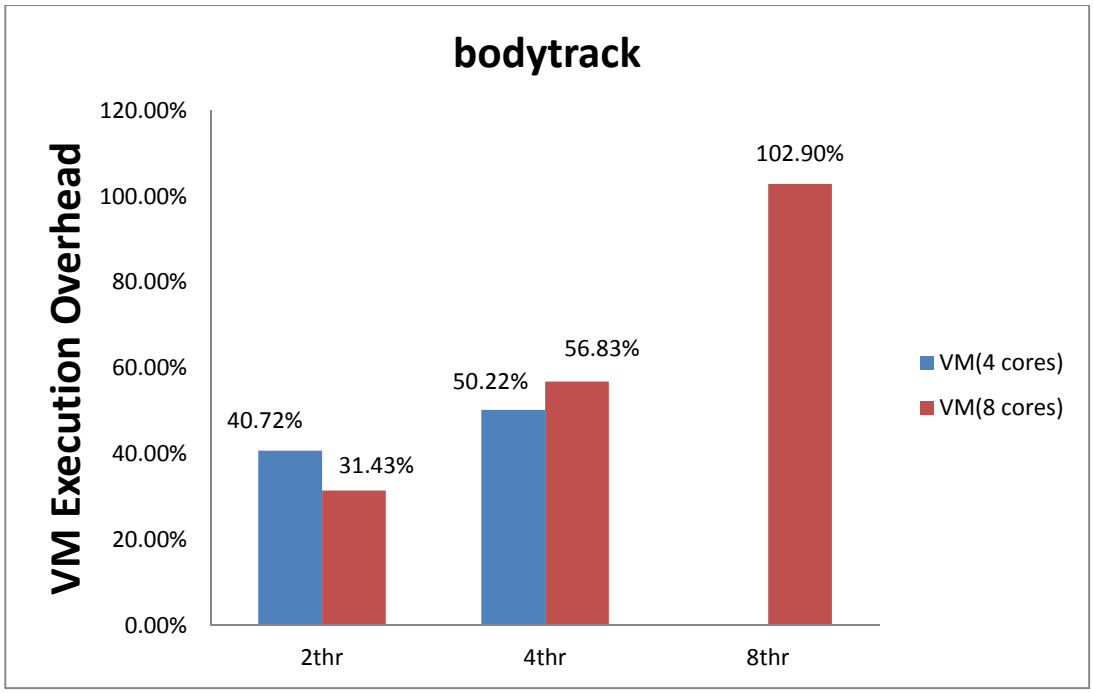
Από την γραφική παράσταση Σχήμα.4. (a) βλέπουμε ότι οι χρόνοι εκτέλεσης και των δύο virtual machines είναι παρόμοιοι ενώ και οι δύο δεν κοντεύουν καθόλου στους native χρόνους εκτέλεσης της εφαρμογής. Αυτό οφείλεται το πιο πιθανό στον τύπο τις εφαρμογής.

Από τις γραφικές παραστάσεις Σχήμα.4. (b) & (c) βλέπουμε ότι ο χρόνος εκτέλεσης στην περίπτωση του virtual machine με τα 4 core όταν ο αριθμός των threads από 4 γίνεται 8 αυξάνεται, γεγονός που μας ανησυχεί αφού σε προηγούμενες περιπτώσεις παρέμενε σταθερός. Επίσης στην περίπτωση της (b) δεν βλέπουμε καθόλου μείωση στο χρόνο εκτέλεσης του virtual machine με τα 8 cores.

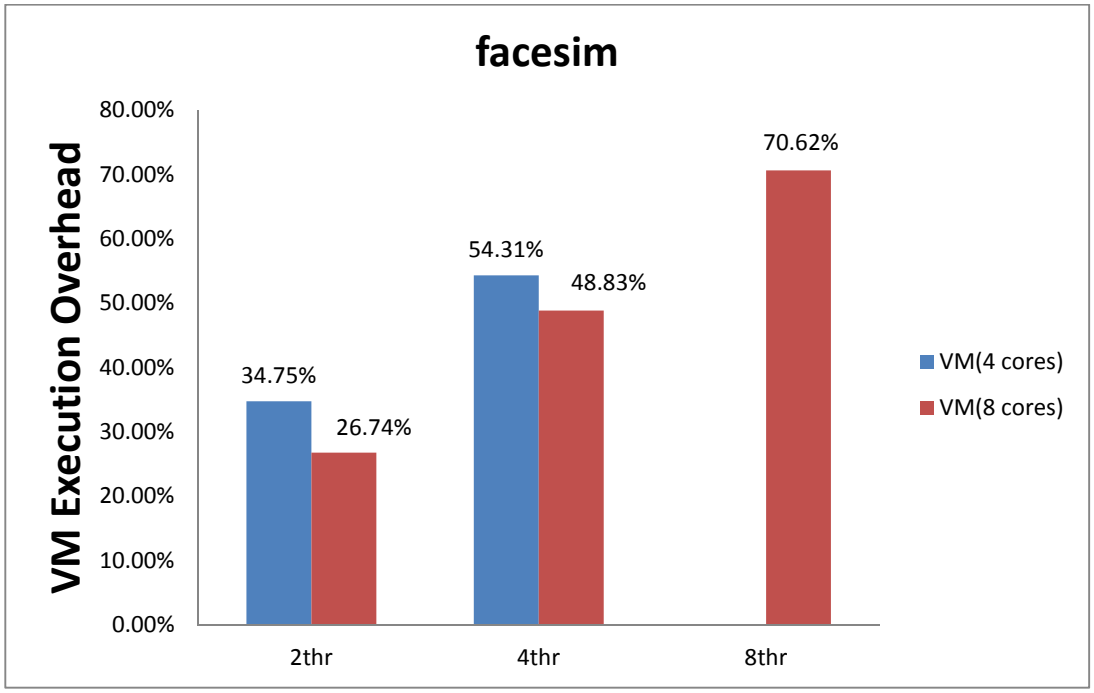
Αν κοιτάξουμε πιο αναλυτικά τα αποτελέσματα αυτά και δούμε τα overheads που προκαλούν τα virtual machine για κάθε εφαρμογή θα πάρουμε τις πιο κάτω γραφικές.


blackscholes
VM Execution Overhead
25.00%
20.00%
15.00%
10.00%
5.00%
0.00%
11.20%
8.22%
11.00%
11.09%
21.27%
2thr
4thr
8thr
VM(4 cores)
VM(8 cores)

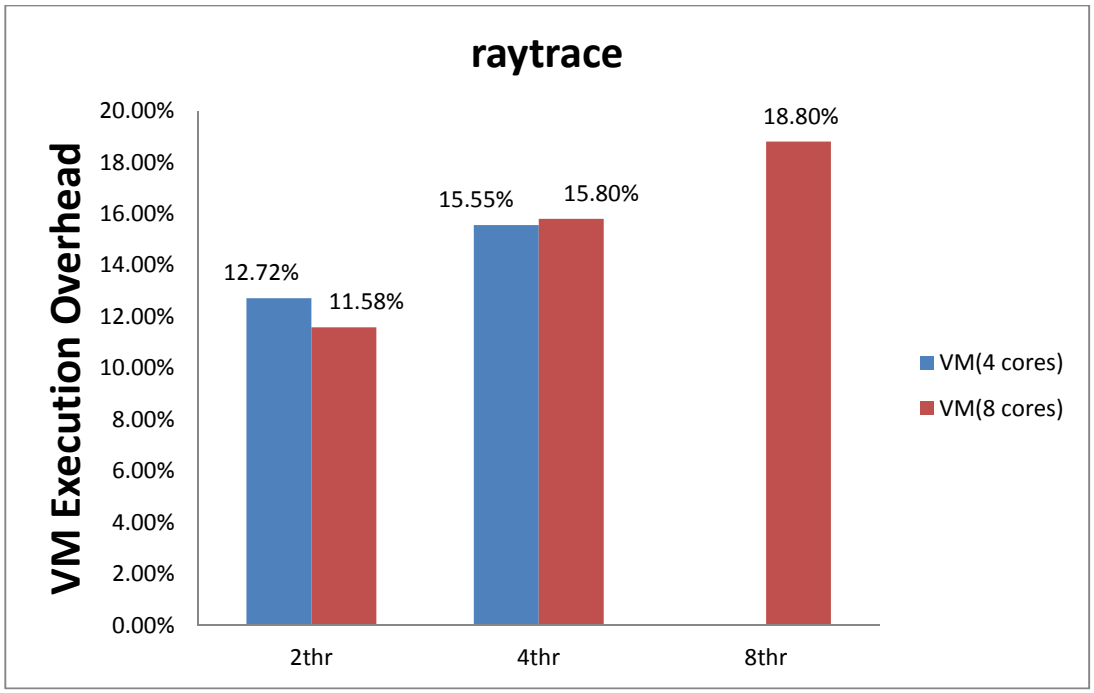
(a)


bodytrack
VM Execution Overhead
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
40.72%
31.43%
50.22%
56.83%
102.90%
2thr
4thr
8thr
VM(4 cores)
VM(8 cores)

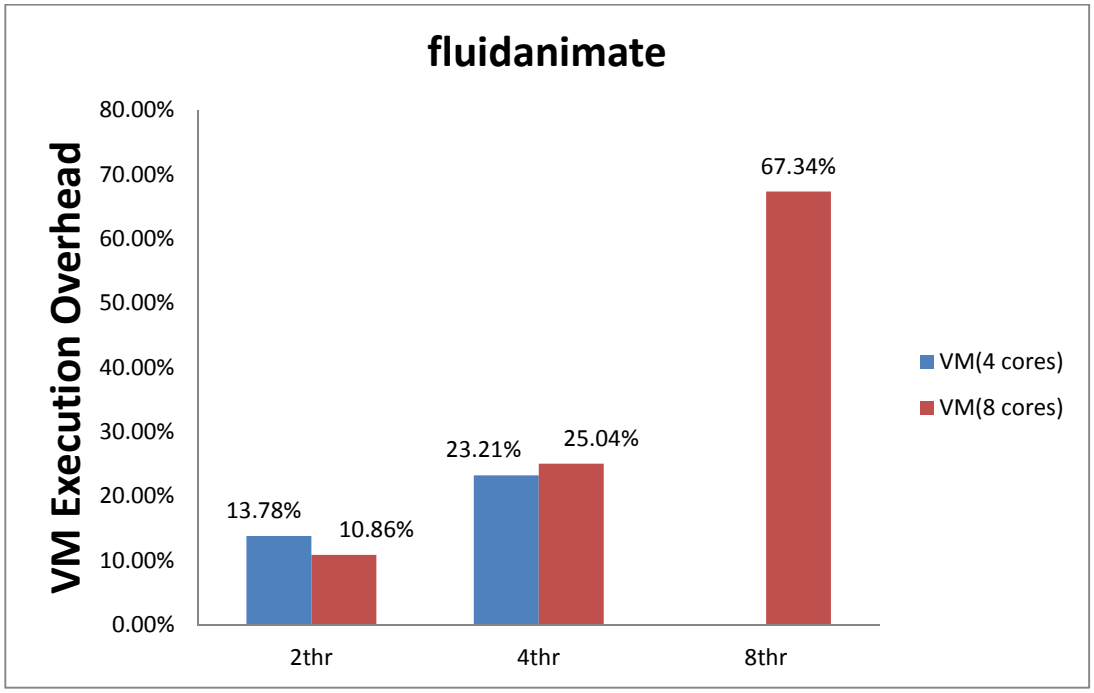
(b)


facesim
VM Execution Overhead
80.00%
70.00%
60.00%
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
34.75%
26.74%
54.31%
48.83%
70.62%
2thr
4thr
8thr
VM(4 cores)
VM(8 cores)

(c)


raytrace
VM Execution Overhead
20.00%
18.00%
16.00%
14.00%
12.00%
10.00%
8.00%
6.00%
4.00%
2.00%
0.00%
12.72%
11.58%
15.55%
15.80%
18.80%
2thr
4thr
8thr
VM(4 cores)
VM(8 cores)

(d)



(e)

Σχήμα.5. VM overheads for PARSEC benchmarks

Καταρχήν πρέπει να αναφέρουμε ότι το overhead στην περίπτωση του virtual machine με τα 4 cores όταν ο αριθμός των threads αυξάνεται από 4 σε 8 δεν αναγράφεται λόγω του ότι είναι κάτι που δεν παρουσιάζει ενδιαφέρον αφού δεν υπάρχουν αρκετά cores. Έτσι οποιαδήποτε σύγκριση του δεν είναι σημαντική.

Αυτό που μπορούμε ακόμη να παρατηρήσουμε από τις γραφικές παραστάσεις του Σχήμα.5. και διαβάζοντας το [56], είναι ότι υπάρχουν τέσσερις διαφορετικές κατηγορίες εφαρμογών στα PARSEC benchmarks.

Στην πρώτη κατηγορία κατατάσσονται οι εφαρμογές που είναι βαριά υπολογιστικές και τα δεδομένα εισόδου που χρησιμοποιούν είναι λίγα (μέγεθος δεδομένων = 2MB). Δεδομένου ότι η εκτέλεση του virtualized κώδικα της εφαρμογής εκτελείται στο φυσικό σύστημα και ότι δεν έχει πολλές I/O εντολές, οι εφαρμογές αυτές αναμένεται να έχουν μικρό virtualization overhead. Μια τέτοια εφαρμογή είναι το blackscholes με μέγιστο overhead 21%.
Η δεύτερη κατηγορία περιλαμβάνει εφαρμογές που χειρίζονται μεγάλα σύνολα δεδομένων εισόδου (μέγεθος δεδομένων = 128MB) και γι' αυτό το overhead που παρουσιάζουν είναι

μεταξύ 16% και 25%. Στην κατηγορία αυτή μπορούμε να κατατάξουμε το fluidanimate και το raytrace.

Η τρίτη κατηγορία περιλαμβάνει εφαρμογές με ακόμη μεγαλύτερο συνόλων δεδομένων εισόδου (μέγεθος δεδομένων = 256MB) που οδηγούν σε ακόμα μεγαλύτερα overheads, μέχρι και 50%. Στην κατηγορία αυτή ανήκει το facesim. Αυτό δικαιολογείται από το γεγονός ότι στο facesim το 33% των εντολών του αποτελείται από reads και το 14% αποτελείται από writes, σε αντίθεση με τις άλλες εφαρμογές, όπου οι αντίστοιχες εντολές read και write είναι 25% και 7% αντίστοιχα [56].

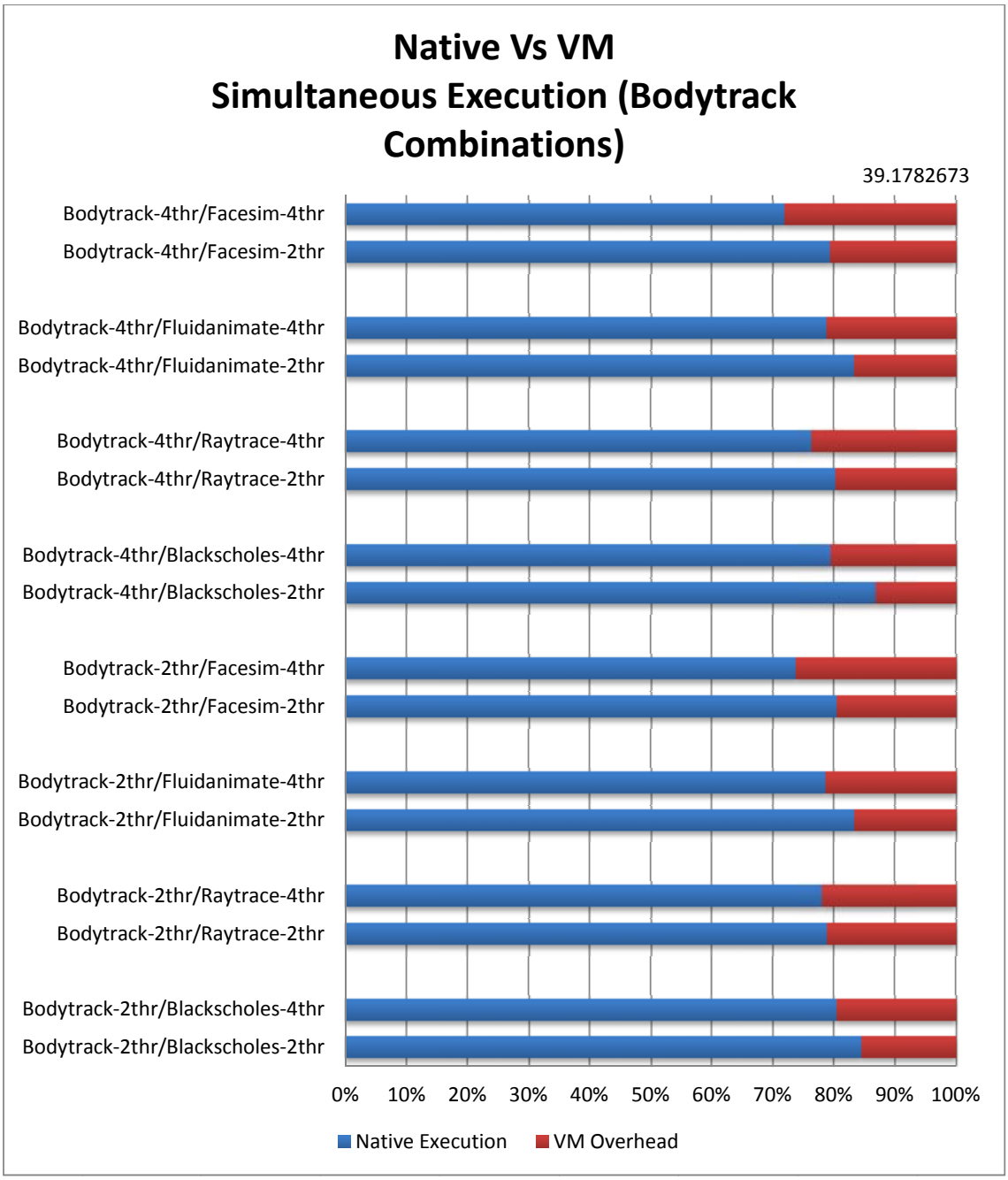
Τέλος, στην τελευταία κατηγορία κατατάσσουμε τις εφαρμογές που έχουν πολύ μεγάλο overhead. Στην περίπτωση του bodytrack φαίνεται να παρουσιάζει ένα αρκετά μεγάλο overhead έως και 102% (Σχήμα.5. (b)). Το μέγεθος των δεδομένων εισόδου του είναι 8MB. Όπως διαβάσαμε από το [56] το ψηλό αυτό overhead που παρουσιάζεται οφείλεται μάλλον στα πολλά barriers (lock & barrier-based synchronizations) και τον μεγάλο αριθμό των condition variables (waits), σε αντίθεση με τις άλλες εφαρμογές. Από το overhead που παρουσιάζεται από την εν λόγο εφαρμογή, και από εφαρμογές που ανήκουν στην κατηγορία αυτή, φαίνεται να μην είναι καλή ιδέα να εκτελούνται σε virtual machines.

Όπως αναφέραμε όμως και στην αρχή του υποκεφαλαίου αυτού κύριος στόχος μας είναι να εξετάσουμε αν με το να χρησιμοποιήσουμε περισσότερα virtual machines, εκμεταλλευόμενοι έτσι του isolation που μας παρέχουν μπορούμε να πετύχουμε καλύτερους χρόνους εκτέλεσης στις εφαρμογές μας και όσο το δυνατό λιγότερα overheads.

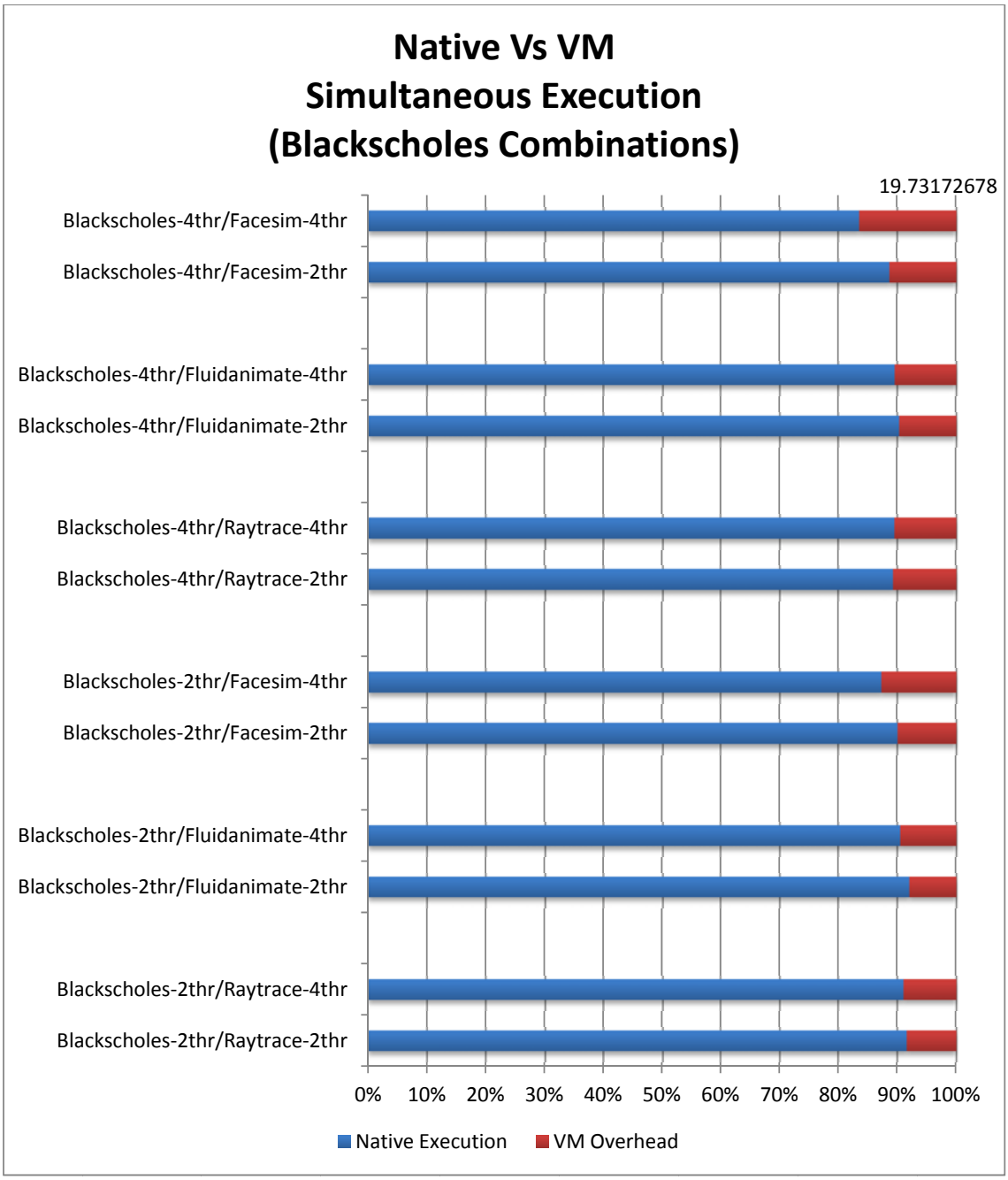
5.2 Παρουσίαση Αποτελεσμάτων Πρώτης και Δεύτερης Κατηγορίας

Για την πρώτη κατηγορία πειραμάτων διαλέξαμε να δημιουργήσουμε όλα τα πιθανά ζευγάρια από τα PARSEC benchmarks που έχουμε αναφέρει και για κάθε ζευγάρι όλους του πιθανούς συνδυασμούς από threads που θα μπορούσαν να έχουν. Ο λόγο που το αλλάζουμε τον αριθμό τον threads είναι για να δούμε κατά πόσο αυτό επηρεάζει τις εκτελέσεις με virtual machines αλλά και χωρίς. Αυτά τα πειράματα έγιναν σε δύο VM που το καθένα αποτελείται από 4 cores και αντίστοιχα runs έγιναν και για τη native τους εκτέλεση.

Αρχικά παρουσιάζουμε ένα γενικό πίνακα που περιέχει όλα τα ζευγάρια και το αναλογιζόμενο overhead τους σε σχέση με τις native εκτελέσεις τους. Αυτό γίνεται για να γίνει μια σύγκριση με τα αποτελέσματα του υποκεφαλαίου 5.1 στη συνέχεια.


Native Vs VM
Simultaneous Execution (Bodytrack Combinations)
39.1782673
Bodytrack-4thr/Facesim-4thr
Bodytrack-4thr/Facesim-2thr
Bodytrack-4thr/Fluidanimate-4thr
Bodytrack-4thr/Fluidanimate-2thr
Bodytrack-4thr/Raytrace-4thr
Bodytrack-4thr/Raytrace-2thr
Bodytrack-4thr/Blackscholes-4thr
Bodytrack-4thr/Blackscholes-2thr
Bodytrack-2thr/Facesim-4thr
Bodytrack-2thr/Facesim-2thr
Bodytrack-2thr/Fluidanimate-4thr
Bodytrack-2thr/Fluidanimate-2thr
Bodytrack-2thr/Raytrace-4thr
Bodytrack-2thr/Raytrace-2thr
Bodytrack-2thr/Blackscholes-4thr
Bodytrack-2thr/Blackscholes-2thr
0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
Native Execution
VM Overhead

(a)



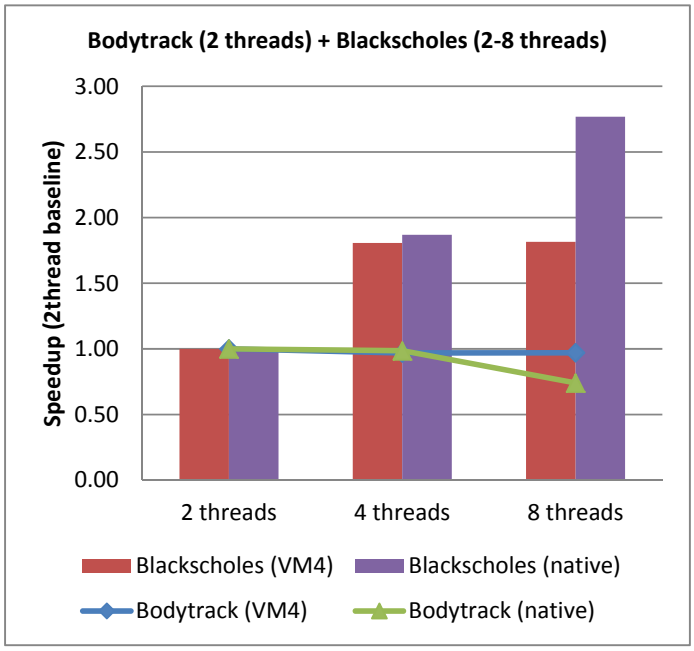
(b)

Σχήμα.6. PARSEC benchmark overheads on parallel execution with 2 VMs

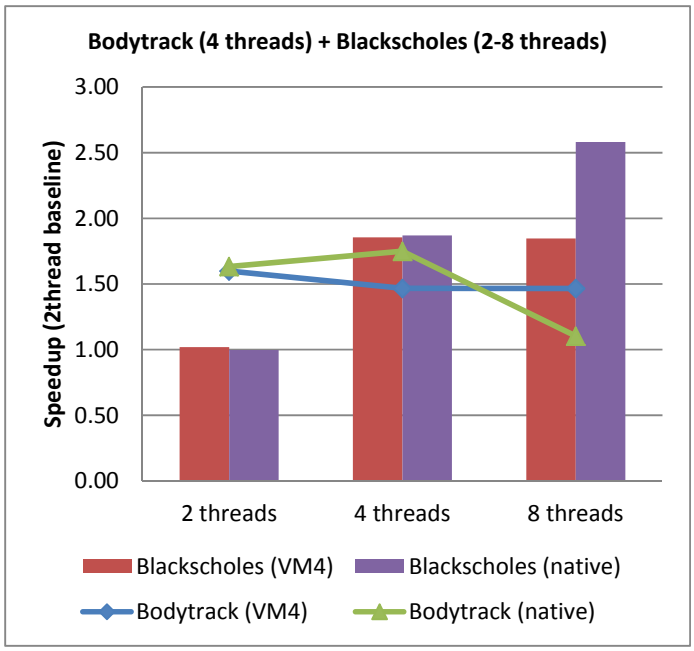
Αυτό που είναι σημαντικό να παρατηρήσουμε είναι ότι στην περίπτωση του fluidanimate με το bodytrack τα overheads που παρουσιάζονται είναι αρκετά χαμηλά σε όλες τις περιπτώσεις αν τα συγκρίνουμε με αυτά του υποκεφαλαίου 5.1 παρόλο που οι δύο εφαρμογές αυτές σε πολλές περιπτώσεις παρουσίαζαν μεγάλο overhead. Από αυτό μπορούμε να συμπεράνουμε ότι λόγο του context switching που γίνεται κατά την εκτέλεση του native προσθέτει αρκετό κόστος στους χρόνους εκτέλεσης κάθε εφαρμογής. Επομένως το isolation που προσφέρουν τα VM φαίνεται να είναι πράγματι ένας παράγοντας που βοηθά στις εφαρμογές.

Συγκρίνοντας τα overheads των συνδυασμών του bodytrack και του blackscholes βλέπουμε ότι στις περιπτώσεις όπου οι εφαρμογές δεν τρέχουν μόνες τους παρουσιάζουν λιγότερο συνολικό overhead από τις περιπτώσεις που έτρεχαν αποκλειστικά μόνες τους. Στις γραφικές παραστάσεις (a) και (b) ο αριθμός που αναγράφεται είναι το μέγιστο overhead που παρουσιάζουν το bodytrack και blackscholes αντίστοιχα. Αυτό που μπορούμε να δούμε είναι ότι το μέγιστο συνολικό overhead είναι αρκετά λιγότερο και στις δύο περιπτώσεις. Αυτό οφείλεται στο γεγονός ότι όταν το λειτουργικό σύστημα έχει να εκτελέσει περισσότερες εφαρμογές, τότε τα κόστη που προσθέτει στους χρόνους εκτέλεσης χωρίς virtual machine είναι αρκετά ενώ οι εκτελέσεις μέσα στα virtual machine λόγω του ότι έχουμε λιγότερο context switching και λόγο του program isolation δεν επηρεάζονται σημαντικά.

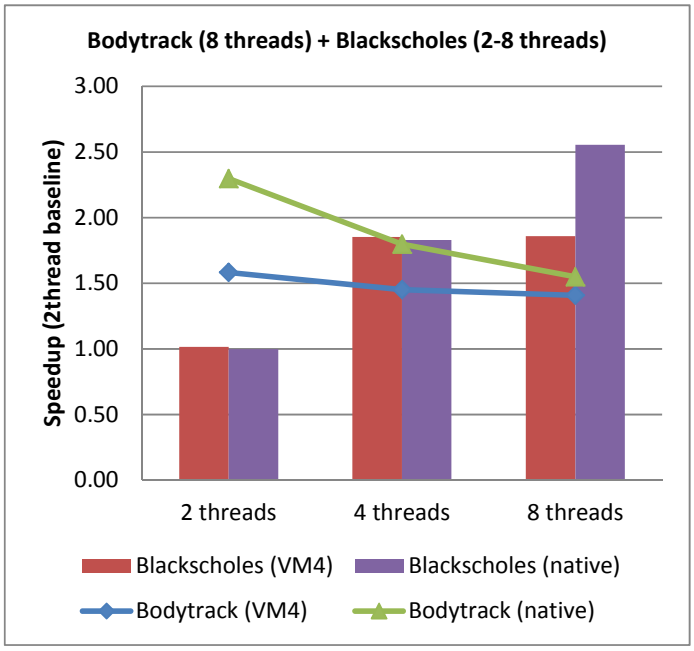
Χρησιμοποιώντας τα αποτελέσματα θα παρουσιάσουμε μια σειρά από γραφικές παραστάσεις που δείχνουν το speedup για τους συνδυασμούς ζευγαριών με σκοπό να δούμε κατά πόσο αυτό επηρεάζεται από τα virtual machines.



(a)



(b)

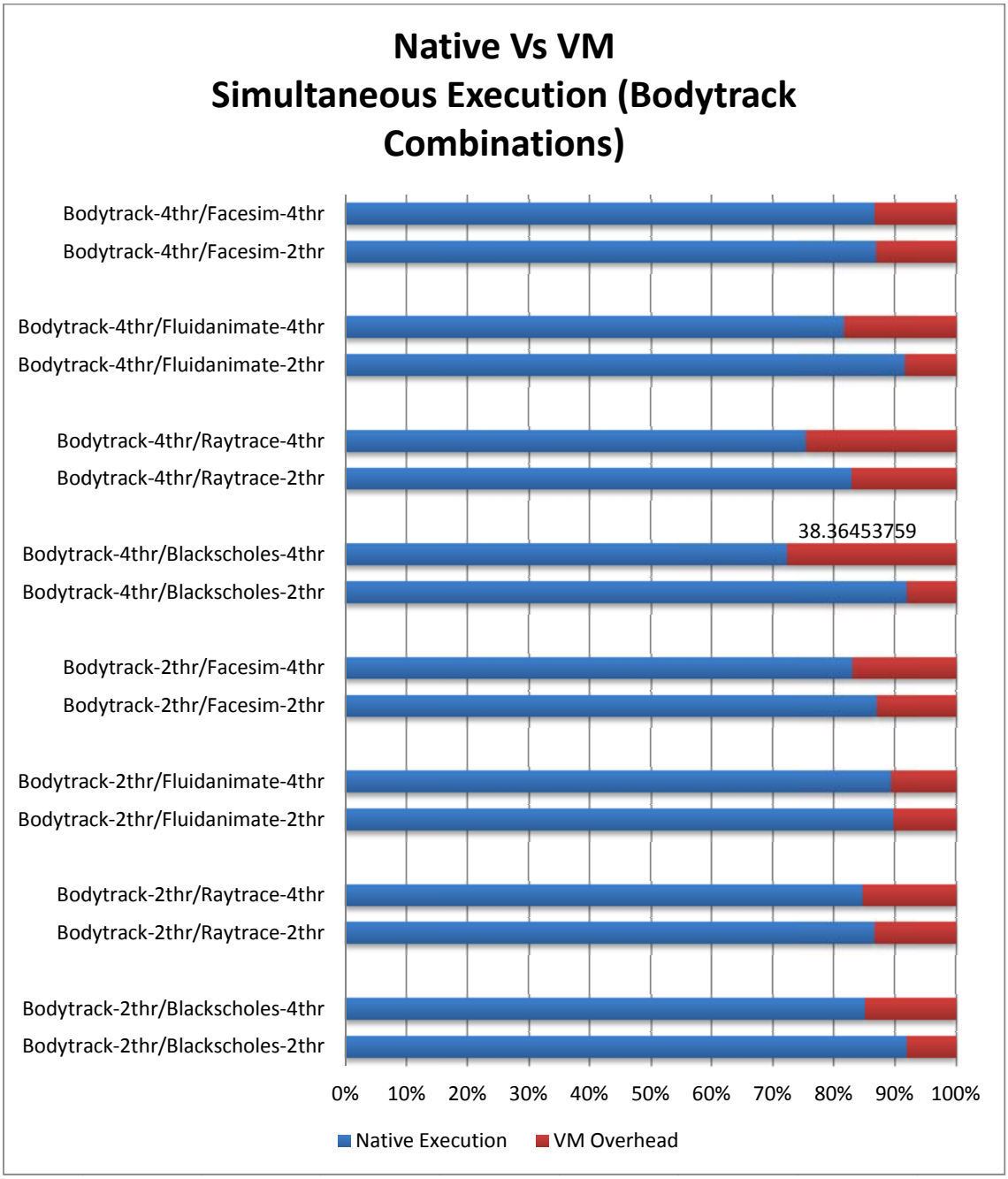


(c)

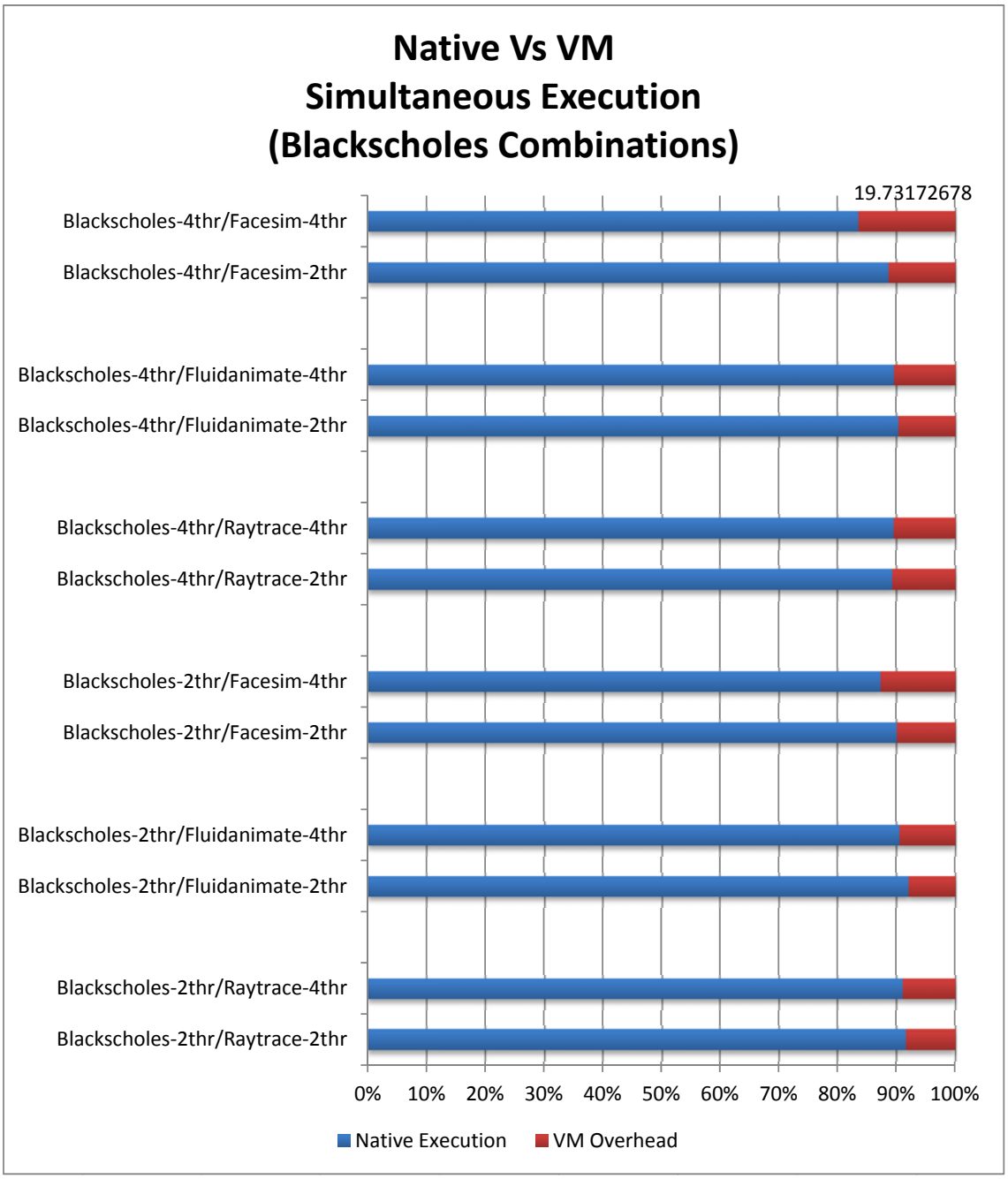
Σχήμα.7. bodytrack-blackscholes speedup

Κρατώντας σταθερό σε κάθε περίπτωση (Σχήμα.7. (a) (b) (c)) τον αριθμό των threads της μιας εφαρμογής αλλάζαμε τον αριθμό των threads της άλλης. Στις γραφικές παραστάσεις του Σχήμα.7. παρουσιάζεται η σύγκριση του speedup μεταξύ native και VM εκτέλεσης. Αυτό που παρατηρούμε για την εφαρμογή όπου ο αριθμός των threads παραμένει σταθερός κάθε φορά είναι ότι στο VM πετυχαίνουμε μια σταθερότητα όσο αφορά το speedup σε αντίθεση με τις native εκτελέσεις όπου έχουμε συνεχώς μια μείωση η οποία μάλιστα σε αρκετές περιπτώσεις φαίνεται να είναι χειρότερη από αυτή στα virtual machines. Όσον αφορά την εφαρμογή της οποίας ο αριθμός των threads αλλάζει κάθε φορά, βλέπουμε ότι στο VM πετυχαίνουμε το ίδιο speedup με τις native εκτελέσεις έκτος βέβαια από την περίπτωση των 8 threads, γεγονός το οποίο περιμέναμε αφού το virtual machine έχει μόνο 4 cores. Παρόλα αυτά όμως το speedup σε αυτή την περίπτωση παραμένει σχετικά σταθερό. Με αυτά τα αποτελέσματα μπορούμε να εξακριβώσουμε ότι το program isolation είναι αυτό που οφείλεται στην σταθερότητα του speedup. Για το λόγο αυτό έχουμε λιγότερα context switching, αφού έχουμε λιγότερα threads σε κάθε περίπτωση, και επομένως λιγότερος χρόνος σπαταλείται από το λειτουργικό για την εργασία αυτή.

Στη συνέχεια θέλαμε να παρατηρήσουμε τη διαφορά που θα είχαμε στο overhead των virtual machines σε περίπτωση που είχαμε μια μηχανή με 4 αντί 8 cores. Για να το πετύχουμε αυτό απενεργοποιήσαμε τα 4 cores της μηχανής μέσω του GRUB. Συγκεκριμένα πήραμε αποτελέσματα από native εκτελέσεις για όλους τους συνδυασμούς εφαρμογών με 2 και 4 threads ανάλογα, και με το ίδιο σκεπτικό μετρήσεις για 2 virtual machine με 2 cores στο καθένα.



(a)



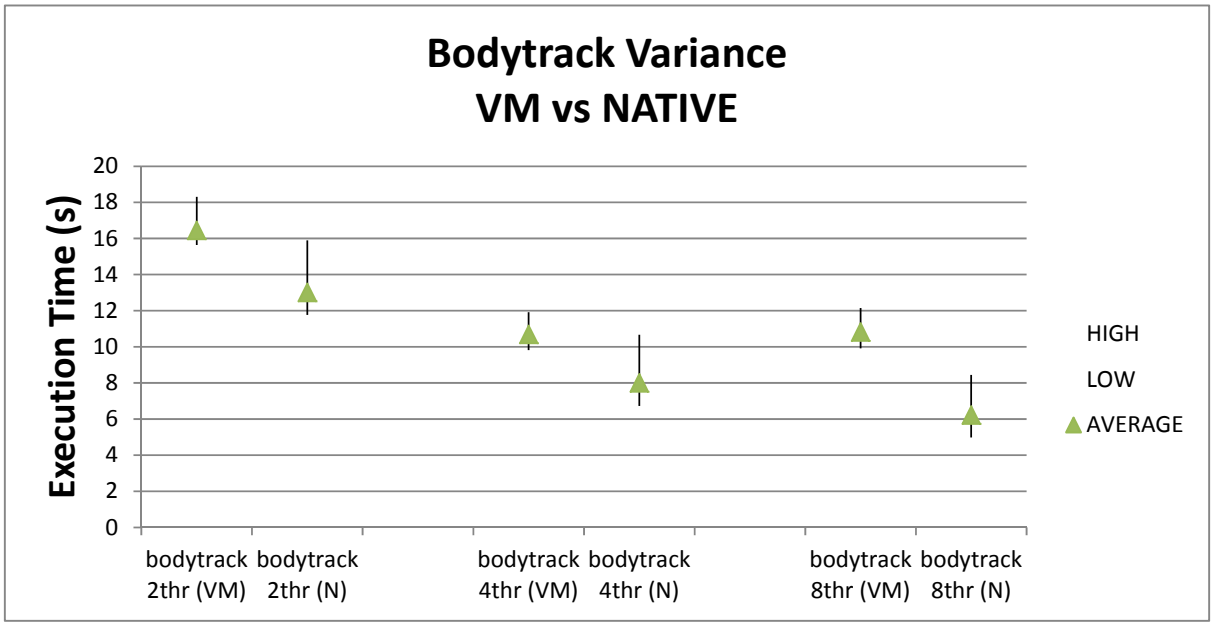
(b)

Σχήμα.8. VM overheads with 4 physical cores

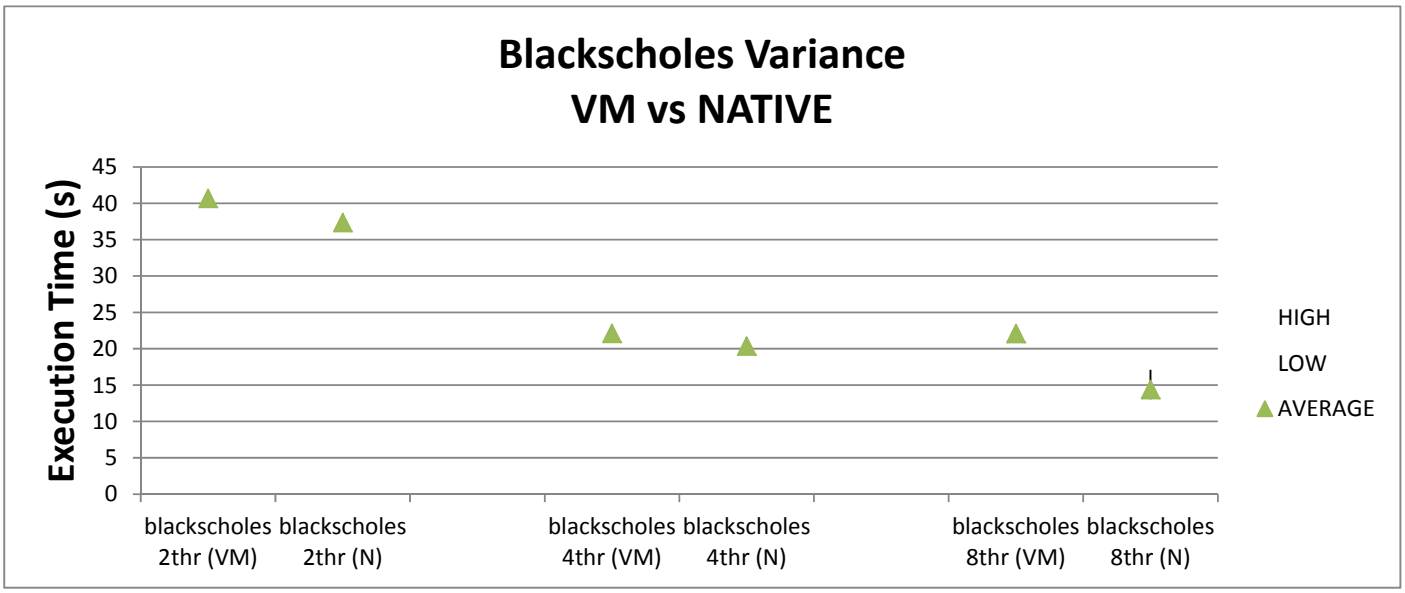
Συγκρίνοντας τις γραφικές παραστάσεις του Σχήμα.6. με αυτές του Σχήμα.8. μπορούμε να δούμε ότι μειώνοντας τον αριθμό των cores στη μηχανή, και κατά συνέπεια τον αριθμό που αναθέτουμε σε κάθε VM παρατηρούμε ότι η εφαρμογή blackscholes δεν επηρεάζεται σε κανένα από τους συνδυασμούς, ενώ αντίθετα η εφαρμογή bodytrack επηρεάζεται. Και στις δύο περιπτώσεις το μέγιστο συνολικό overhead παραμένει το ίδιο. Αυτό συμβαίνει λόγω του

ότι κάθε εφαρμογή είναι διαφορετική, και επομένως επηρεάζεται μεταβάλλοντας τον αριθμό των cores της μηχανής ανάλογα.

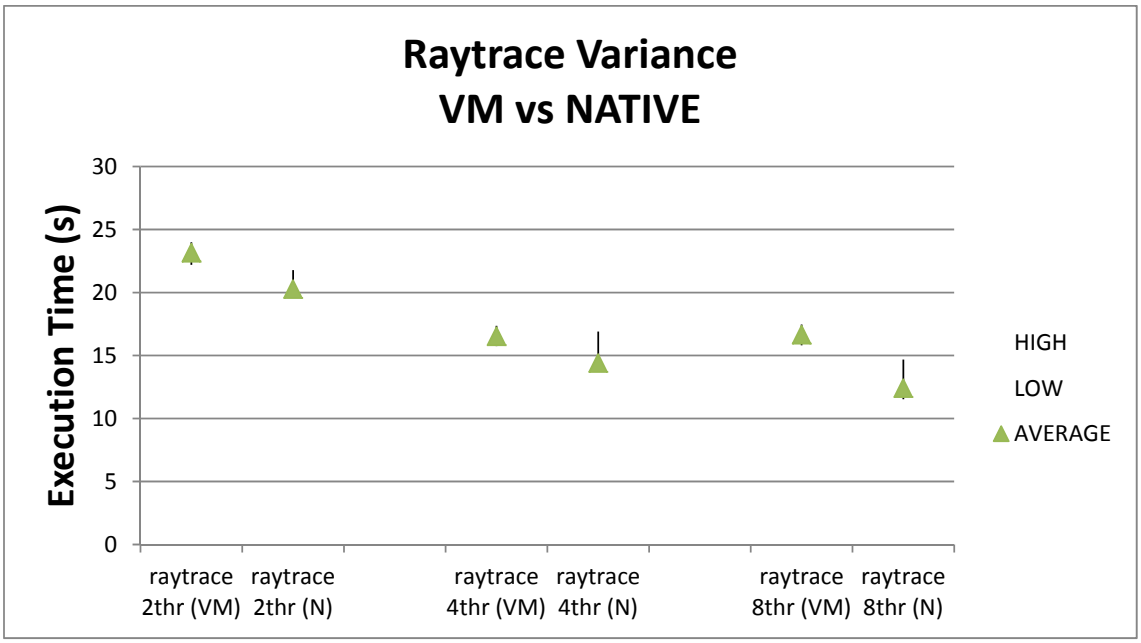
Για να μελετήσουμε τη σταθερότητα των δύο virtual machines χρησιμοποιήσαμε τις πιο κάτω γραφικές παραστάσεις όπου δείχνουν τη διακύμανση στους χρόνους εκτέλεσης των εφαρμογών στα δύο VM με 4 cores. Η γραμμή αναπαριστά την μέγιστη και ελάχιστη τιμή ενώ το τρίγωνο μας δείχνει την μέση τιμή.



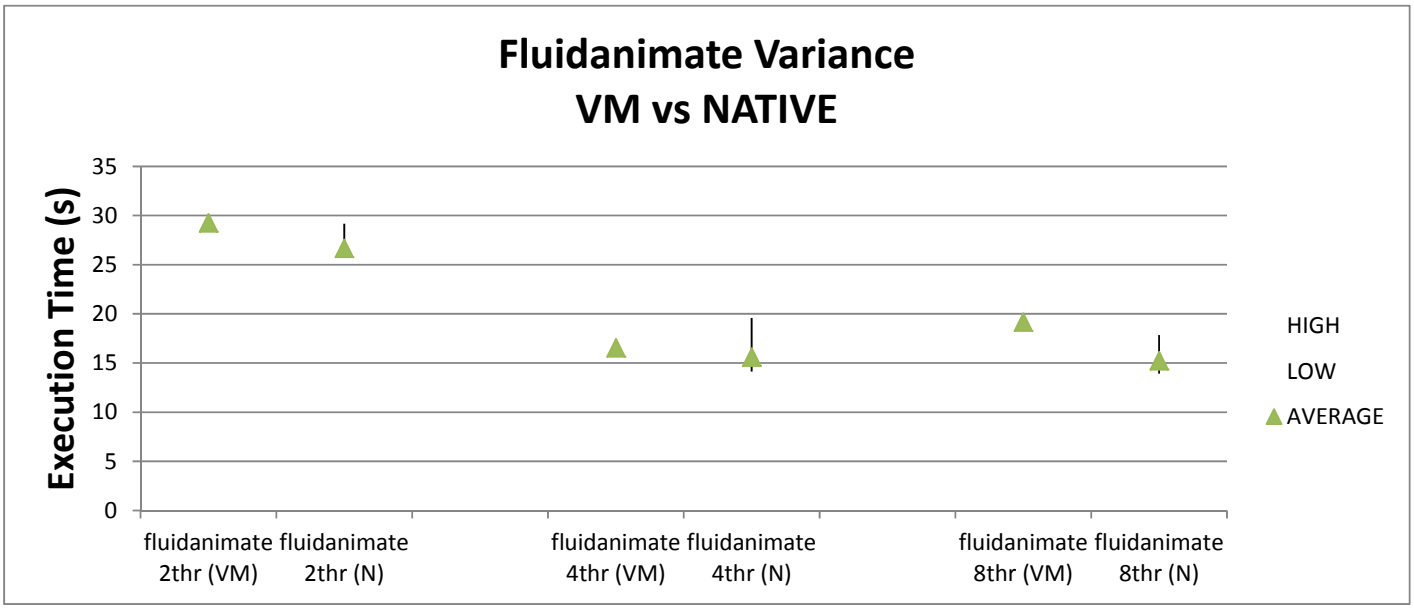
(a)



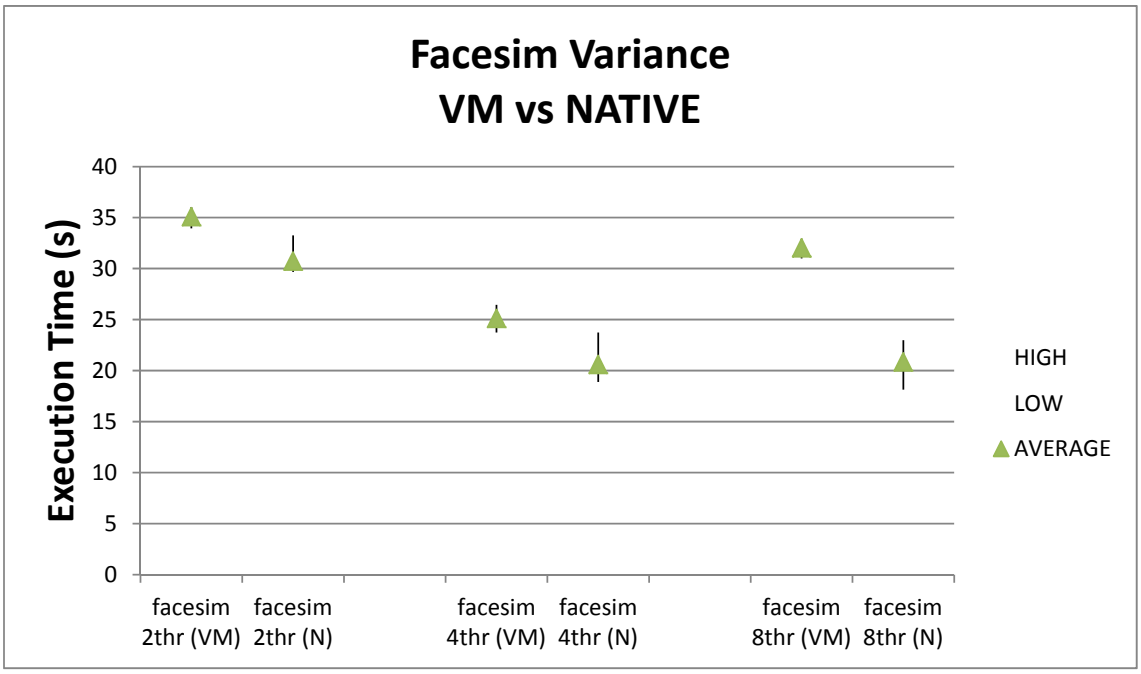
(b)


Raytrace Variance
VM vs NATIVE
Execution Time (s)
30
25
20
15
10
5
0
raytrace 2thr (VM)
raytrace 2thr (N)
raytrace 4thr (VM)
raytrace 4thr (N)
raytrace 8thr (VM)
raytrace 8thr (N)
HIGH
LOW
AVERAGE

(c)


Fluidanimate Variance
VM vs NATIVE
Execution Time (s)
35
30
25
20
15
10
5
0
fluidanimate 2thr (VM)
fluidanimate 2thr (N)
fluidanimate 4thr (VM)
fluidanimate 4thr (N)
fluidanimate 8thr (VM)
fluidanimate 8thr (N)
HIGH
LOW
AVERAGE

(d)



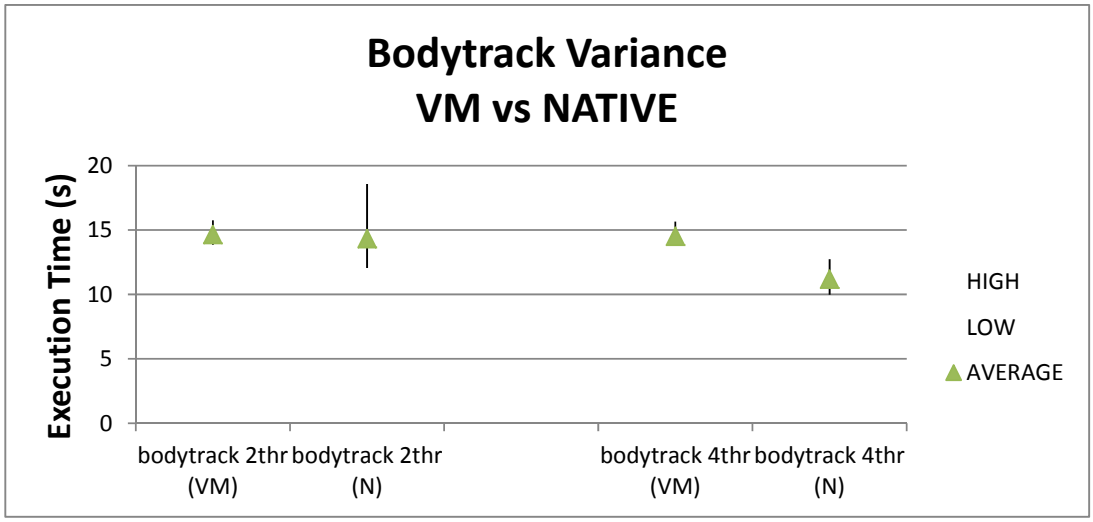
(e)

Σχήμα.9. VM vs Native Variance (8 physical cores)

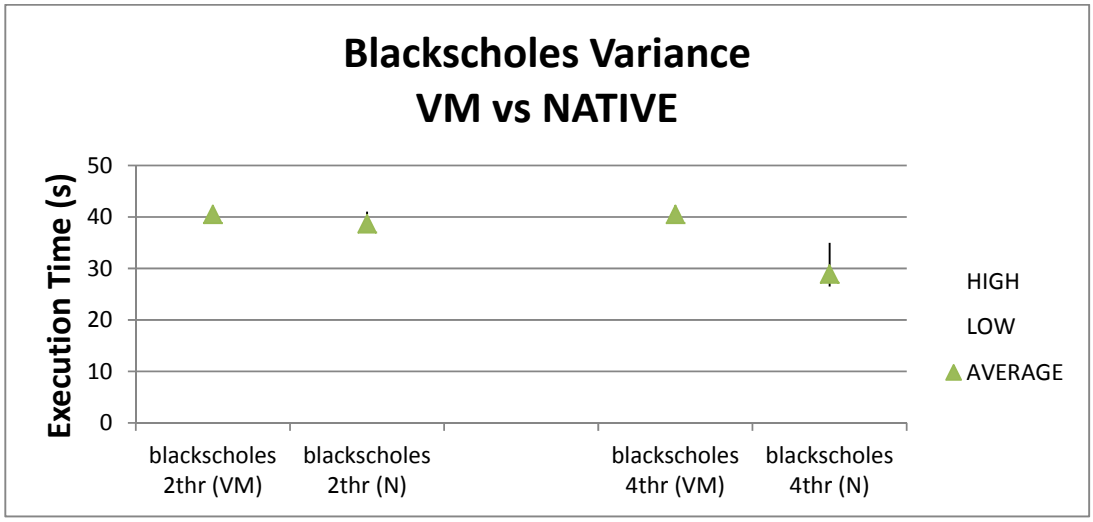
Από τις γραφικές παραστάσεις (b) (c) και (d) βλέπουμε ότι οι χρόνοι εκτέλεσης των εφαρμογών στα virtual machines είναι αρκετά σταθεροί σε αντίθεση με τους χρόνους εκτέλεσης χωρίς virtual machine. Παρόλο που οι χρόνοι εκτέλεσης χωρίς τα VM είναι καλύτεροι, ακόμη και στη χειρότερη περίπτωση, η συμπεριφορά αυτή των VM είναι ενδιαφέρουσα. Αυτό σημαίνει ότι το program isolation αλλά και αυτή η δέσμευση των πόρων που παρέχει το κάθε VM βοηθά στην πιο ομαλή/σταθερή εκτέλεση των εφαρμογών μας.

Στις περιπτώσεις (a) και (e) όπου και το VM παρουσιάζει κάποια διακύμανση θεωρούμε στην πρώτη περίπτωση ότι οφείλεται στο είδος της εφαρμογής (πολλά condition statement, locks και barriers) ενώ στη δεύτερη λόγο του μεγάλου συνόλου δεδομένων εισόδου σε συνδυασμό με τα locks και τα condition statements που υπάρχουν.

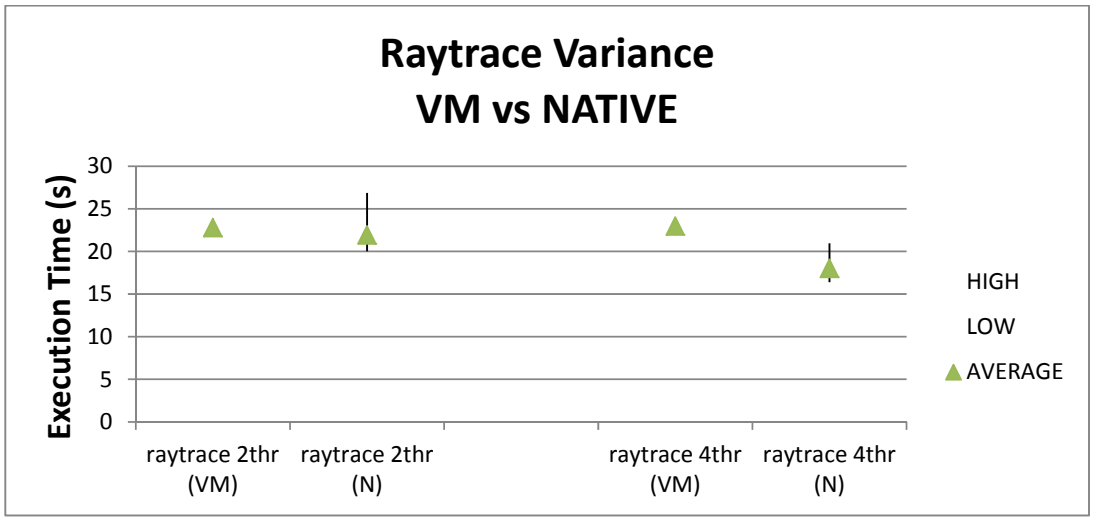
Παίρνοντας τις ίδιες γραφικές παραστάσεις για την περίπτωση όπου ο συνολικός αριθμός των cores της μηχανής μειώθηκε στα 4, και σε κάθε virtual machine είχαμε αναθέσει 2 core παρατηρήσαμε κάτι αρκετά ενδιαφέρον.


Bodytrack Variance
VM vs NATIVE
Execution Time (s)
20
15
10
5
0
bodytrack 2thr (VM)
bodytrack 2thr (N)
bodytrack 4thr (VM)
bodytrack 4thr (N)
HIGH
LOW
AVERAGE

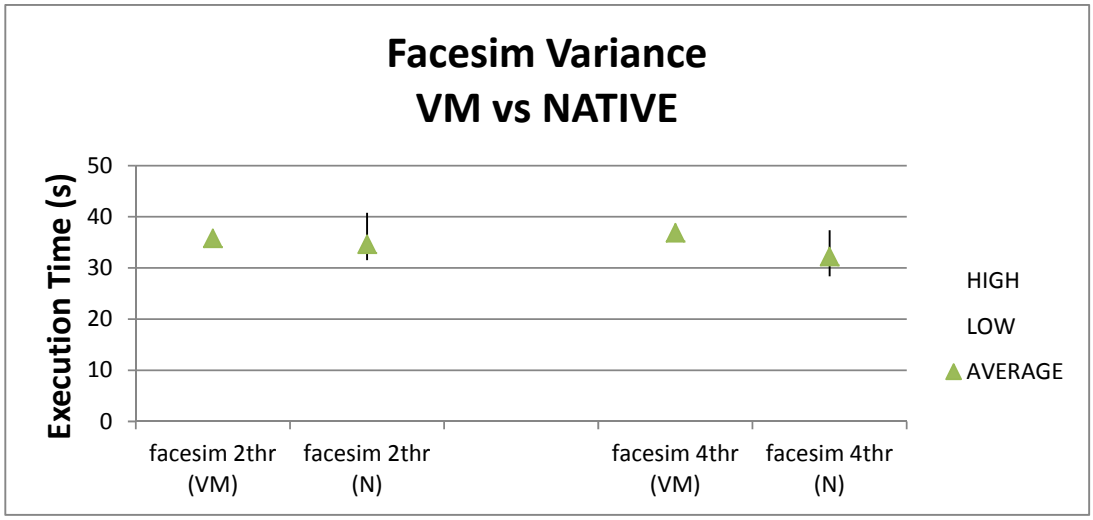
(a)


Blackscholes Variance
VM vs NATIVE
Execution Time (s)
50
40
30
20
10
0
blackscholes 2thr (VM)
blackscholes 2thr (N)
blackscholes 4thr (VM)
blackscholes 4thr (N)
HIGH
LOW
AVERAGE

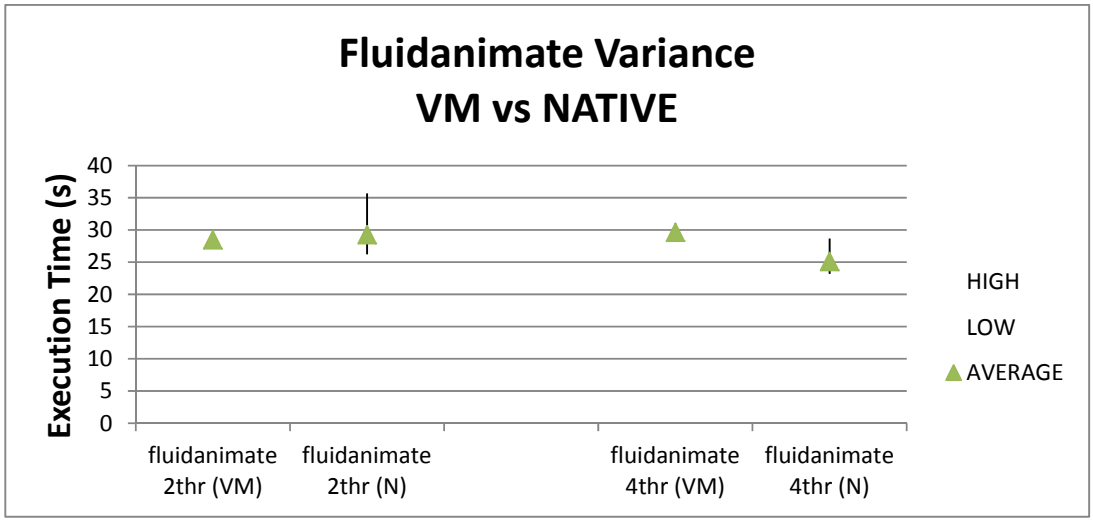
(b)


Raytrace Variance
VM vs NATIVE
Execution Time (s)
30
25
20
15
10
5
0
raytrace 2thr (VM)
raytrace 2thr (N)
raytrace 4thr (VM)
raytrace 4thr (N)
HIGH
LOW
AVERAGE

(c)



(d)



(e)

Σχήμα.10. VM vs Native Variance (4 physical cores)

Από τις γραφικές παραστάσεις του Σχήμα.10. μπορούμε να παρατηρήσουμε ότι σε όλες τις περιπτώσεις η εκτέλεση των εφαρμογών με δύο threads είχε αρκετή διακύμανση, δηλαδή επηρεαζόταν πολύ από τις εφαρμογές που έτρεχαν παράλληλα με αυτή με διαφορετικό αριθμό thread κάθε φορά ενώ σε αντίθεση οι χρόνοι εκτέλεσης μέσα στα VMs παρουσιάζουν μια σταθερότητα, πράγμα αρκετά σημαντικό. Αυτό μας δείχνει ότι το περιβάλλον μέσα στα virtual machine είναι αρκετά απομονωμένο έτσι ώστε να μην επηρεάζει η μια εφαρμογή την άλλη. Επίσης αρκετές είναι οι περιπτώσεις όπου ο μέσος χρόνος εκτέλεσης των εφαρμογών μέσα στο virtual machine είναι περίπου ο ίδιος με αυτό χωρίς τα virtual machine.

Έτσι έχοντας υπόψη τα αποτελέσματα που πήραμε μέχρι τώρα θα παρουσιάσουμε τα αποτελέσματα της επόμενης κατηγορίας όπου αυξάνουμε τον αριθμό των virtual machines στη μηχανή.

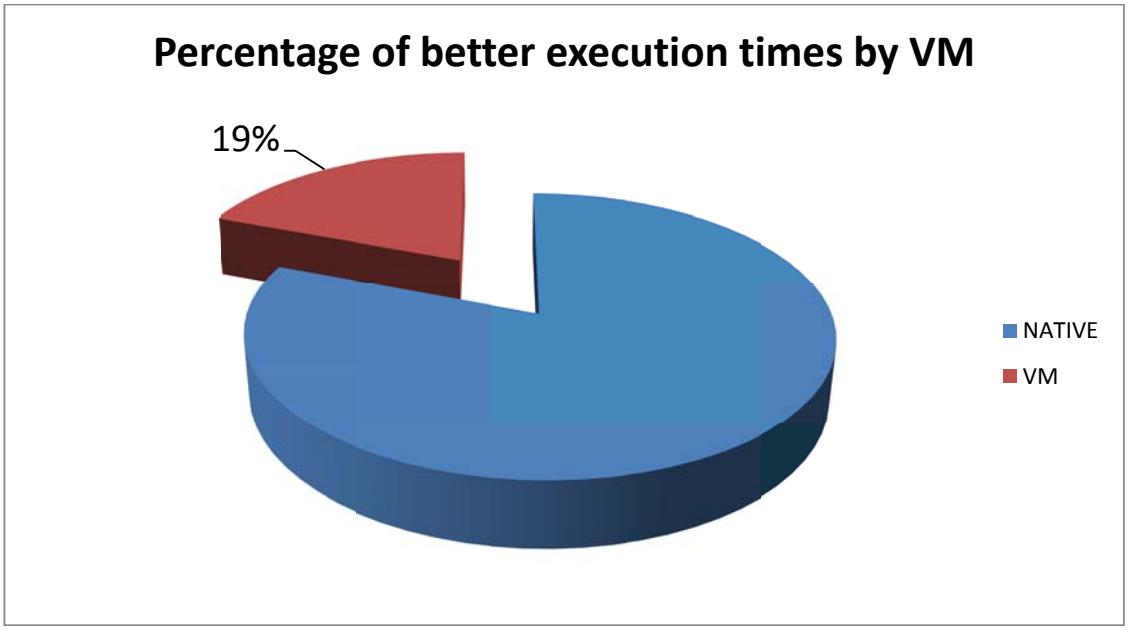
5.3 Παρουσίαση Αποτελεσμάτων Τρίτης Κατηγορίας

Έχοντας στη διάθεση μας 5 εφαρμογές από τα PARSEC benchmarks και συνολικά αυτή τη φορά 4 virtual machines καταλήξαμε να έχουμε τα πιο κάτω σενάρια για τους συνδυασμούς των διαφόρων εκτελέσεων:

Scenario1: bodytrack – blackscholes – raytrace – fluidanimate
Scenario2: bodytrack – blackscholes – raytrace – facesim
Scenario3: bodytrack – blackscholes – fluidanimate – facesim
Scenario4: bodytrack – raytrace – fluidanimate – facesim

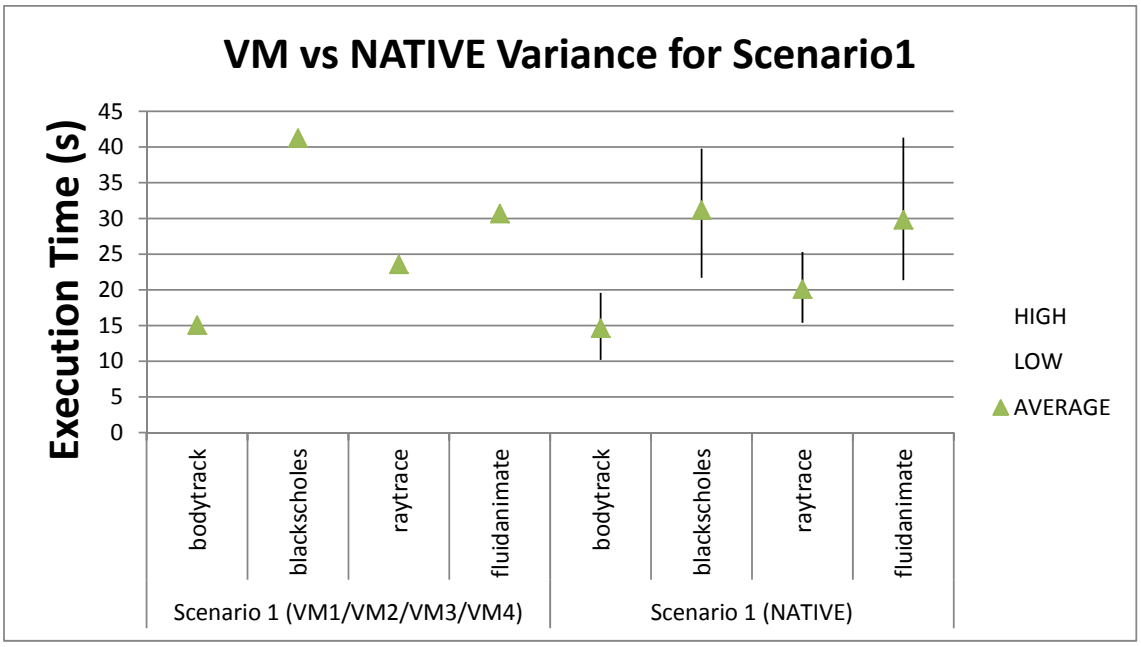
Λόγω του ότι σε κάθε ένα από τα 4 virtual machines που είχαμε αναθέσαμε 2 cores ο αριθμός τον threads έπρεπε να ήταν 2 και 4. Συνολικά υπήρχαν 256 διαφορετικοί συνδυασμοί.

Πιο κάτω παρουσιάζουμε τις περιπτώσεις όπου είχαμε καλύτερους χρόνους εκτέλεσης στο virtual machine από αυτούς που είχαμε στο native.

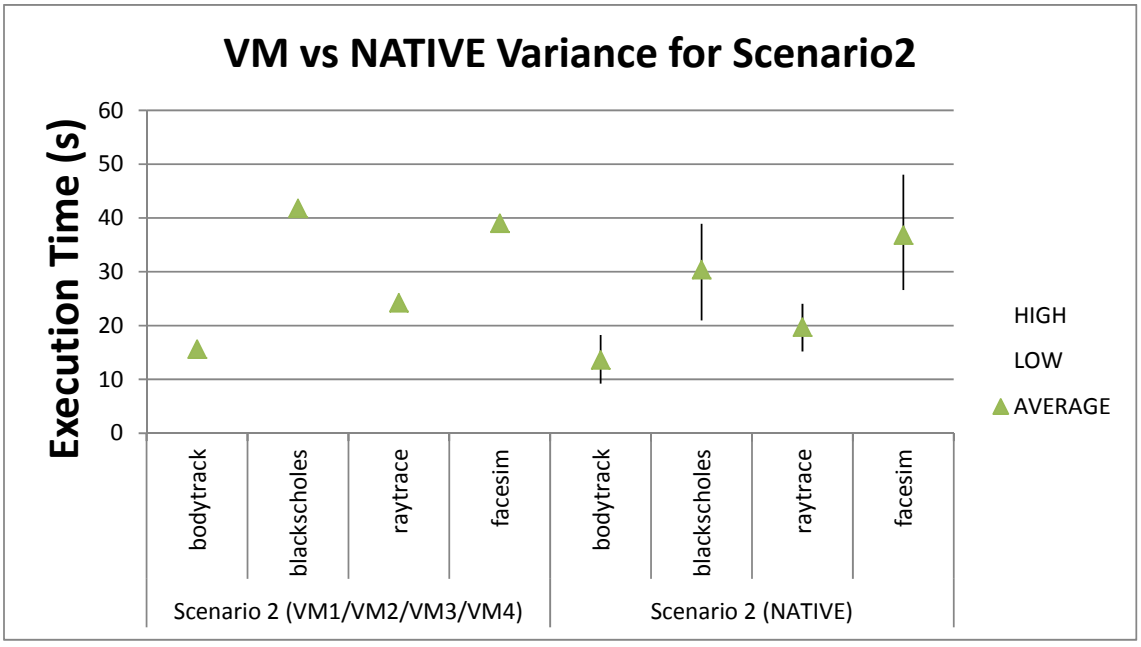


Σχήμα.11. Percentage where virtual machine execution time was better than native

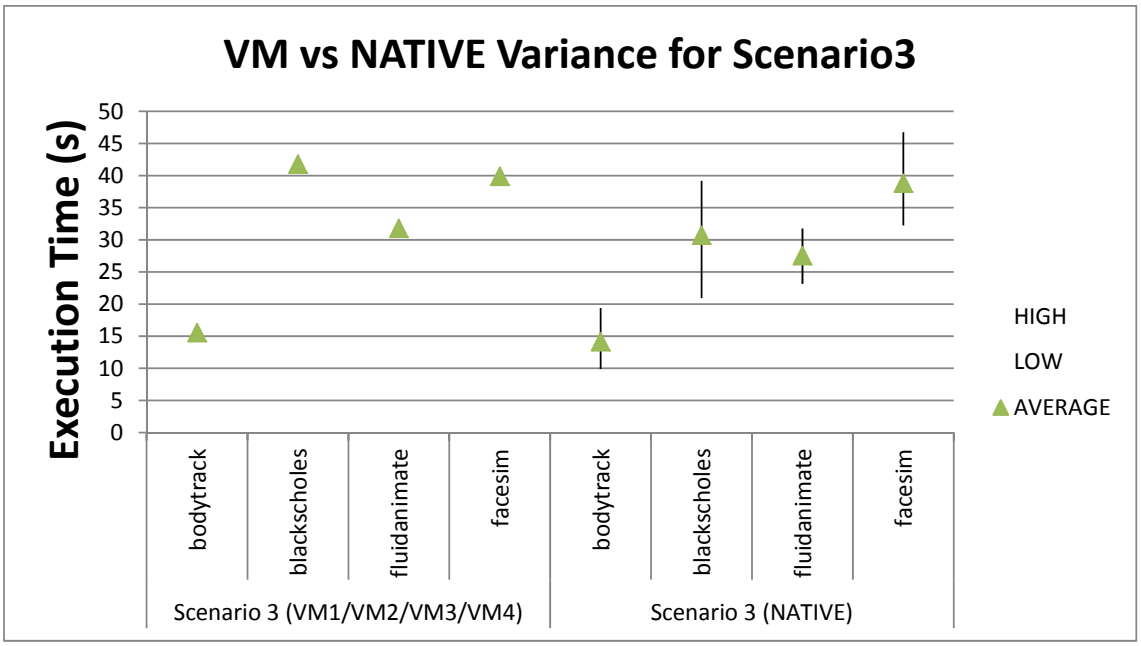
Όπως παρατηρούμε έχουμε ένα σημαντικό ποσοστό του 19% όπου το virtual machine έχει πετύχει καλύτερους χρόνους εκτέλεσης από αυτούς στο native σύστημα.


VM vs NATIVE Variance for Scenario1
Execution Time (s)
45
40
35
30
25
20
15
10
5
0
bodytrack
blackscholes
raytrace
fluidanimate
bodytrack
blackscholes
raytrace
fluidanimate
Scenario 1 (VM1/VM2/VM3/VM4)
Scenario 1 (NATIVE)
HIGH
LOW
AVERAGE

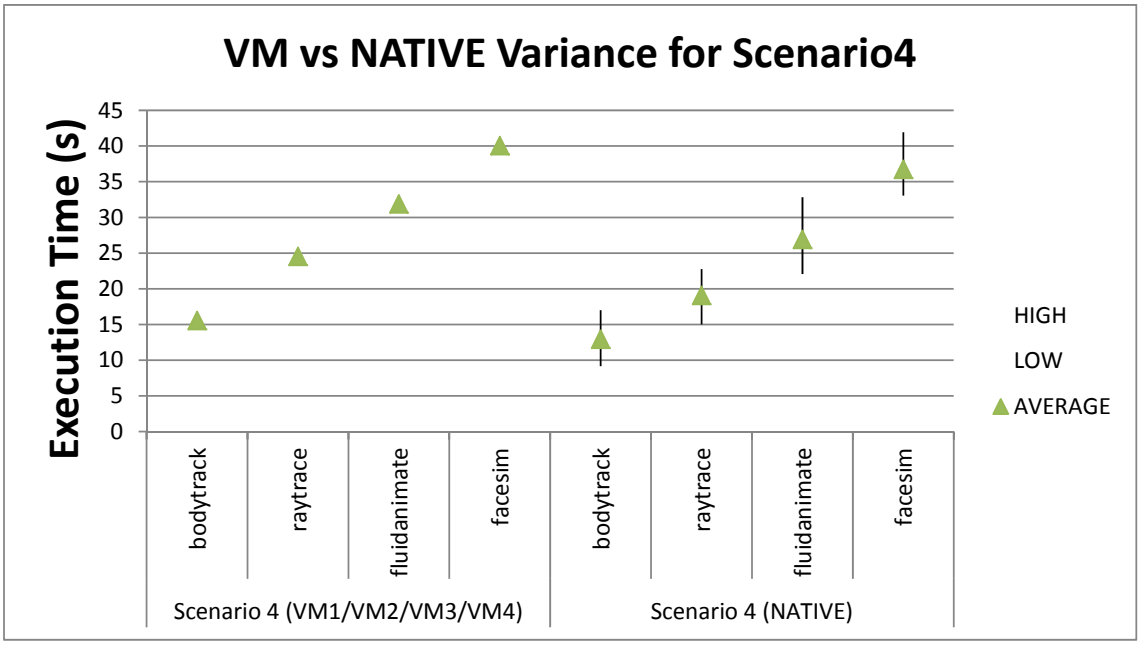
(a)


VM vs NATIVE Variance for Scenario2
Execution Time (s)
60
50
40
30
20
10
0
bodytrack
blackscholes
raytrace
facesim
bodytrack
blackscholes
raytrace
facesim
Scenario 2 (VM1/VM2/VM3/VM4)
Scenario 2 (NATIVE)
HIGH
LOW
AVERAGE

(b)



(c)



(d)

Σχήμα.12. VM vs. Native Variance → (execution times)

Στις γραφικές παραστάσεις του Σχήμα.12. παρουσιάζεται η διακύμανση μεταξύ του καλύτερου και του χειρότερου χρόνου εκτέλεσης καθώς επίσης και ο μέσος όρος των δύο.

Συγκρίνοντας σε κάθε γραφική παράσταση το κάθε σενάριο ξεχωριστά μπορούμε να παρατηρήσουμε κάτι που ισχύει για όλα τα σενάρια. Αυτό είναι ότι οι χρόνοι εκτέλεσης για τις εφαρμογές μέσα στα virtual machines παραμένουν σταθεροί ενώ σε αντίθεση με τους χρόνους εκτέλεσης έξω από τα virtual machines παρουσιάζουν μεγάλη διακύμανση. Αυτό συμβαίνει διότι βάζοντας τις 4 εφαρμογές να τρέχουν natively με τη σύγχυση και το μεγάλο φόρτο εργασίας από τα threads καθώς επίσης και του context switching στους επεξεργαστές δεν επιτρέπουν στο σύστημα να αποδώσει σωστά, με αποτέλεσμα να παρατηρείται αυτή η αστάθεια.

Ακόμη αν πάρουμε για παράδειγμα την περίπτωση της εφαρμογής facesim στο σενάριο 2 μπορούμε με ευκολία να παρατηρήσουμε ότι οι χρόνοι εκτέλεσης με virtual machine είναι καλύτεροι από τους μισούς χρόνους εκτέλεσης χωρίς virtual machine. Και πάλι αυτό συμβαίνει λόγο του program isolation που παρέχεται από τα virtual machines.

5.4 Προβλήματα που Παρουσιάστηκαν

Όταν προσπαθήσαμε να δημιουργήσουμε virtual machines με συνολικό αριθμό των virtual cores περισσότερο από τα physical cores, δηλαδή 8 cores σε κάθε VM οπόταν συνολικό αριθμό cores 16, παρουσιάστηκε πρόβλημα στην ακριβής μέτρηση του χρόνου εκτέλεσης κάθε εφαρμογής και στη συνέχεια το συγκεκριμένο virtual machine δεν ανταποκρινόταν. Μετά από πολλές προσπάθειες και αρκετό χρόνο αποφασίσαμε ότι το πρόβλημα δεν μπορεί να λυθεί και αναβάλαμε την εκτέλεση αυτών των πειραμάτων.

Κεφάλαιο 6

Συμπεράσματα και Μελλοντική Εργασία

6.1 Συμπεράσματα

Λαμβάνοντας υπόψη όλη την πειραματική μελέτη που έχει γίνει πάνω στα virtual machines καθώς και τα πειράματα που έγιναν στα πλαίσια αυτής της εργασίας μπορούμε να πούμε ότι τα virtual machine monitors τα οποία τρέχουν πάνω από ένα λειτουργικό σύστημα, παρόλο που αυτό προσθέτει κάποιο κόστος στην επίδοση που θα πρέπει να το αποδεχτούμε, παρέχοντας το program isolation μπορούν να προσφέρουν αρκετά καλούς χρόνους εκτέλεσης για εφαρμογές διαφόρων τύπων. Στις περισσότερες περιπτώσεις μπορούμε να προβλέψουμε το πόσο speedup θα έχουμε καθώς αλλάζουμε τον αριθμό τον threads σε μια εφαρμογή. Αυτή η σταθερότητα του speedup στους χρόνους εκτέλεσης των εφαρμογών είναι ένα επιπλέον πλεονέκτημα που διασφαλίζουν τα virtual machines.

Στις περιπτώσεις όπου έχουμε servers με πολλούς επεξεργαστές διάφοροι οργανισμοί θα μπορούσαν να επωφεληθούν από το virtualization αφού για μικρό σχετικά κόστος επίδοσης θα μπορούσαν για παράδειγμα να έχουν πάνω σε μία μηχανή μαζί τον web server, mail server και file server τους. Με αυτό τον τρόπο δεν εξοικονομούν μόνο ενέργεια και χώρο αλλά καταφέρνουν να εκμεταλλεύονται τους πόρους της μηχανής τους καλύτερα.

Σύμφωνα με τα πειράματα που έχουν γίνει σε αυτή την έρευνα μπορούμε να διασφαλίσουμε ότι για παράδειγμα σε ένα web server όπου ο αριθμός των threads τυγχάνει ραγδαίας διακύμανσης σύμφωνα με τον αριθμό των χρηστών χρησιμοποιώντας ένα virtual machine θα μπορούσε ο διαχειριστής της μηχανής να έχει μια πιο ομαλή εξυπηρέτηση των χρηστών. Η ομαλότητα στην εξυπηρέτηση των χρηστών προκύπτει από την σχετικά σταθερή και

προβλέψιμη επίδοση των εφαρμογών σε ένα virtual machine καθώς αυξομειώνεται ο αριθμός των threads τους σε αντίθεση με τα προγράμματα που τρέχουν έξω από το virtual machine. Ως γνωστό πολλοί servers αυξομειώνουν τον αριθμό των threads τους ανάλογα με τον αριθμό των αιτήσεων που παίρνουν ανά μονάδα χρόνου (thread pool).

Πιο συγκεκριμένα τα πειράματα που έγιναν έχουν δήξει ότι καταρχήν το κόστος επίδοσης των εφαρμογών εξαρτάται σίγουρα από το είδος της κάθε εφαρμογής. Επίσης συγκρίνοντας τα κόστη στους χρόνους εκτέλεσης στην περίπτωση που χρησιμοποιούμε μόνο ένα VM και όταν χρησιμοποιούμε περισσότερα παρατηρήσαμε ότι οι χρόνοι εκτέλεσης είναι αρκετά κοντά αναμεταξύ τους όσο αφορά τις εκτελέσεις στα virtual machine, ενώ σε αντίθεση οι χρόνοι εκτέλεσης χωρίς virtual machine παρουσιάζουν κάποια σημαντική αύξηση, αφού τώρα εμπλέκονται περισσότερες εφαρμογές για εκτέλεση στην ίδια μηχανή. Αυτό έχει σαν αποτέλεσμα, αφού οι χρόνοι εκτέλεσης με virtual machine παραμένουν σχετικά σταθεροί, να μην παρατηρείται τόσο υψηλό κόστος στα virtual machine. Τέλος δείξαμε ότι η σταθερότητα στους χρόνους εκτέλεσης των εφαρμογών που παρουσιάζουν τα τέσσερα (4) virtual machines είναι αρκετά πιο καλή από αυτή που παρουσιάζουν τα δύο (2).

6.2 Μελλοντική Εργασία

Όσο αφορά την περαιτέρω ερεύνα για μελλοντικές επεκτάσεις το πιο σημαντικό στάδιο θα είναι να γίνουν τα πειράματα αυτά πάνω σε hypervisor αντί των συνηθισμένων virtual machine monitors. Όπως προαναφέρθηκε λόγω του ότι σε αυτή την περίπτωση δεν έχουμε το επιπλέον στρώμα του host OS αλλά το virtual machine monitor επικοινωνεί κατευθείαν με το υλικό της μηχανής είναι αναμενόμενο να περιμένουμε να επιτύχουμε καλύτερους χρόνους εκτέλεσης από αυτούς που έχουμε τώρα, και ίσως καλύτερα ποσοστά όπου το virtual machine θα επιτυγχάνει καλύτερους χρόνους εκτέλεσης από το native.

Περεταίρω θα μπορούσε να εξεταστεί η συμπεριφορά των virtual machines σε περιπτώσεις όπου ο αριθμός των thread είναι αρκετά μεγάλος.

Επιπλέον, σαν συνέχεια της εργασίας αυτής, θα μπορούσαμε να μελετήσουμε τις εφαρμογές ως προς τα χαρακτηριστικά τους (χρήση μνήμης, χρήση υπολογιστικής μονάδας, κ.α.) έτσι ώστε να διαχωρίσουμε σε κατηγορίες τις εφαρμογές και να μπορούμε να καθορίσουμε σε ποιες κατηγορίες εφαρμογών μπορεί το virtualization να μας ωφελήσει.

Τέλος θα μπορούσε να γίνει μια πειραματική μελέτη με θέμα την κατανάλωση ενέργειας σε μηχανή όπου τρέχουν n virtual machines και να συγκριθεί με την κατανάλωση ενέργειας που έχουν n φυσικές μηχανές έτσι ώστε να δείξει πόσο όφελος μπορεί να έχει ένας οργανισμός.

Βιβλιογραφία

[1] Deshane, T. and Shepherd, Z. and Matthews, J.N. and Ben-Yehuda, M. and Shah, A. and Rao, B., Quantitative comparison of Xen and KVM at Xen Summit, Boston, MA, USA, 2008

[2] Che, J. and He, Q. and Gao, Q. and Huang, D., Performance Measuring and Comparing of Virtual Machine Monitors, Embedded and Ubiquitous Computing, IEEE/IFIP International Conference 2008. EUC'08

[3] IBM x3650. In http://www-03.ibm.com/systems/x/hardware/rack/x3650m3/specs.html

[4] PARSEC benchmark suite. In http://parsec.cs.princeton.edu/

[5] Simon Crosby and David Brown, The Virtualization Reality at ACM QUEUE December/January 2006-2007

[6] Oracle VM VirtualBox User Manual, http://www.cemeb.net/vitualbox.pdf

[7] Paul Barham∗, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer†, Ian Pratt, Andrew Warfield, XEN and the Art of Virtualization at SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA

[8] Patricio Domingues, Filipe Araujo, Luis Silva, "Evaluating the performance and intrusiveness of virtual machines for desktop grid computing," ipdps, pp.1-8, 2009 IEEE International Symposium on Parallel&Distributed Processing, 2009

[9] Definition of Virtualization. In http://cplus.about.com/od/glossar1/g/virtualization.htm

[10] Webopedia Computer Dictionary. In http://www.webopedia.com/TERM/V/virtualization.html

[11] Golden, B. and Scheffy, C. Virtualization for Dummies – SUN AND AMD SPECIAL EDITION (Understand why virtualization is so important). Published by Wiley Publishing, Inc.

[12] Chutneytech, UK Technology News. In http://www.chutneytech.com/virtualization-a-brief-history/

[13] IBP Virtualization. In http://www-03.ibm.com/systems/z/advantages/virtualization/features.html

[14] IBM Virtualization. In http://www-03.ibm.com/systems/z/advantages/virtualization/index.html

[15] Adair, R.J., R.U. Bayles, L.W. Comeau, and R.J. Creasy, A Virtual Machine System for the 360/40, Report 320-2007, May 1966, IBM Cambridge Scientific Center: Cambridge,MA

[16] VMware Virtualization Basics. In http://www.vmware.com/virtualization/history.html

[17] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. In Proc. of the Fourth Symposium on Operating System Principles, Yorktown Heights, New York, October 1973

[18] Neiger, Gil; A. Santoni, F. Leung, D. Rodgers, R. Uhlig. "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization". *Intel Technology Journal* (Intel) 10 (3): 167–178. doi:10.1535/itj.1003.01. http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art01.pdf. Retrieved 2008-07-06

[19] Gillespie, Matt (2007-11-12). "Best Practices for Paravirtualization Enhancements from Intel Virtualization Technology: EPT and VT-d". Intel Software Network. Intel. http://software.intel.com/en-us/articles/best-practices-for-paravirtualizationenhancements-from-intel-virtualization-technology-ept-and-vt-d. Retrieved 2008-07-06

[20] Advanced Micro Devices. AMD64 Virtualization Codenamed \Pacifica" Technology, Secure Virtual Machine Architecture Reference Manual, May 2005

[21] Sun: Sun Microsystems VirtualBox. http://www.virtualbox.org/ (2011)

[22] XEN Hypervisor. In http://xen.org/

[23] VMWARE vSphere 4. In http://www.vmware.com/products/vsphere/

[24] Windows Server 2008 R2. In http://www.microsoft.com/windowsserver2008/en/us/hypervfaqaspx#SetupandRequirements

[25] Virtual Server 2005 R2. In http://blogs.technet.com/jhoward/archive/2006/04/28/426703.aspx

[26] Oracle VM. In http://www.oracle.com/us/technologies/virtualization/024974.htm

[27] Sun xVM Hypervisor. In http://hub.opensolaris.org/bin/view/Community+Group+xen/sunxvmfaq

[28] Windows Virtual PC. In http://www.microsoft.com/windows/virtual-pc/default.aspx

[29] VMWARE official website. In http://www.vmware.com/

[30] x86 virtualization. In http://en.wikipedia.org/wiki/X86_virtualization

[31] Windows Virtual PC. In http://www.microsoft.com/windows/virtual-pc/

[32] Windows Server 2008 R2 Virtualization with Hyper-V. In http://www.microsoft.com/windowsserver2008/en/us/hyperv-main.aspx

[33] Microsoft Virtual Server 2005 R2. In http://www.microsoft.com/windowsserversystem/virtualserver/

[34] Connectix Home. In http://www.connectix.co.uk/

[35] QEMU Open Source Processor Emulator. In http://wiki.qemu.org/Main_Page

[36] KVM Kernel-based Virtual Machine. In http://www.linux-kvm.org/page/Main_Page

[37] Denali software. In https://www.denali.com/en/

[38] L4 Microkernel Family. In http://en.wikipedia.org/wiki/L4_microkernel_family

[39] Virtualization Technologies from Intel. In http://www.intel.com/technology/virtualization/

[40] AMD Virtualization Technology. In http://sites.amd.com/us/business/itsolutions/usage-models/virtualization/Pages/amd-v.asp

[41] Adams, K. and Agesen, O. 2006. A comparison of software and hardware techniques for x86 virtualization. In Proceedings of the 12th international Conference on Architectural Support For Programming Languages and Operating Systems (San Jose, California, USA, October 21 - 25, 2006). ASPLOS-XII. ACM, New York, NY, 2-13. DOI= http://doi.acm.org/10.1145/1168857.1168860

[42] Swsoft Virtualization and Automation Software. In http://www.swsoft.co.uk/

[43] Parallels Virtuozzo Containers. In http://www.parallels.com/eu/products/pvc45/

[44] OpenVZ Wiki. In http://wiki.openvz.org/Main_Page

[45] Java Virtual Machine Wikipedia. In http://en.wikipedia.org/wiki/Java_Virtual_Machine

[46] Adobe Flash Player. In http://get.adobe.com/flashplayer/

[47] Vx32 Wikipedia. In http://en.wikipedia.org/wiki/Vx32

[48] Common Language Infrastructure. In http://en.wikipedia.org/wiki/Common_Language_Infrastructure

[49] Storage Virtualization Companies. In http://www.bitpipe.com/olist/Storage-Virtualization.html

[50] 3PAR Products, services and technologies. In http://www.bitpipe.com/detail/ORG/1115668099_139.html

[51] Arrow ECS HP Group and Intel. In http://www.bitpipe.com/rlist/org/1252584980_528.html

[52] Dell and VMWare Software Products. In http://www.bitpipe.com/rlist/org/1243608377_78.html

[53] IBP Software Products. In http://www.bitpipe.com/rlist/org/1033409397_523.html

[54] Scheffy, C. Virtualization for Dummies – AMD SPECIAL EDITION (Find out how virtualization can benefit your organization). Published by Wiley Publishing, Inc

[55] Oracle VM Virtual Box downloads. In http://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html

[56] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, October 2008.

[57] Christian Bienia and Kai Li. PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors. In Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation, June 2009

[58] Christian Bienia, Sanjeev Kumar and Kai Li. PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors. In Proceedings of the IEEE International Symposium on Workload Characterization, September 2008

[59] VMware. Timekeeping in VMware Virtual Machines, VMware® ESX 3.5/ESXi 3.5, VMware Workstation 6.5, VMware 2008

[60] Xu, X. and Zhou, F. and Wan, J. and Jiang, Y., Quantifying performance properties of virtual machine at Information Science and Engineering, 2008. ISISE'08. International Symposium in 2008

[61] Jin, H. and Cao, W. and Yuan, P. and Xie, X., Benchmark for dynamic server performance of virtualization technology at Proceedings of the 1st international forum on Next-generation multicore/manycore technologies in 2008

[62] Makhija, V. and Herndon, B. and Smith, P. and Roderick, L. and Zamost, E. and Anderson, J., VMmark: A scalable benchmark for virtualized systems at VMware Inc, CA, Tech. Rep. VMware-TR-2006-002, September