Ατομική Διπλωματική Εργασία

Μείωση του Κόστους Μεταφοράς Δεδομένων στον Επεξεργαστή Cell/BE

Ανδρέας Διαβαστός

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2010

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μείωση του Κόστους Μεταφοράς Δεδομένων στον Επεξεργαστή Cell/BE

Ανδρέας Διαβαστός

Επιβλέπων Καθηγητής

Pedro Trancoso

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

Μάιος 2010

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή που κ. Pedro Trancoso για την υποστήριξη και καθοδήγηση που μου προσέφερε κατά την διάρκεια της υλοποίησης της παρούσας εργασίας, καθώς και την επίτευξη μιας άψογης συνεργασίας. Επίσης θερμές ευχαριστίες προς τον διδάκτορα Κυριάκο Σταύρου και τους υποψήφιους διδάκτορες Παναγιώτη Πετρίδη και Frederico Pratas για την υπερπολύτιμη βοήθεια τους καθ'όλην την διάρκεια της εκπόνησης αυτής της εργασίας.

Τέλος στην οικογένεια μου και την παρέα όπως επίσης και στους φίλους και συμφοιτητές μου για την στήριξη που μου έδιναν καθ'όλη τη διάρκεια αυτής της χρονίας.

Περίληψη

Ο επεξεργαστής Cell Broadband Engine είναι επεξεργαστής με κατανεμημένη αρχιτεκτονική μνήμης, πάνω στον οποίο ο προγραμματιστής πρέπει να διαχωρίσει και να κατανέμει τα δεδομένα στους διάφορους πυρήνες που τον αποτελούν. Η λειτουργία αυτή είναι ανάλογη με την κατανομή των δεδομένων σε συστήματα που αποτελούνται από συστάδες υπολογιστών(clusters) και που η επικοινωνία μεταξύ τους επιτυγχάνεται μέσω δικτύων αλληλοσύνδεσης. Αυτό το είδος σχεδιασμού και αρχιτεκτονικής σε ένα πολυπύρηνο επεξεργαστή έρχεται με κάποιο κόστος και το κόστος αυτό είναι η κατανομή και μεταφορά των δεδομένων από την κύρια μνήμη στους πυρήνες επεξεργασίας δεδομένων.

Εδώ και αρκετά χρόνια στα δίκτυα επικοινωνίας και στα δίκτυα αλληλοσύνδεσης χρησιμοποιείται η τεχνική της συμπίεσης δεδομένων(data compression) με αποτέλεσμα την μείωση του όγκου μεταφοράς δεδομένων και συνεπώς την μείωση του χρόνο μεταφοράς των δεδομένων. Αυτή την τεχνική εφαρμόσαμε και στην εργασία αυτή στην προσπάθεια μας να μειώσουμε τον όγκο των δεδομένων που μεταφέρονται από την κύρια μνήμη μέχρι τις μονάδες επεξεργασίας του επεξεργαστή που είναι οι πυρήνες έχοντας δε απώτερο σκοπό να μειώσουμε το κόστος μεταφοράς δεδομένων μέσα στο chip.

Χρησιμοποιώντας την συμπίεση των δεδομένων (data compression) μειώσαμε τον όγκο των δεδομένων κατά την μεταφορά, ακολούθως εφαρμόζαμε την αποσυμπίεση των δεδομένων (data decompression) πάνω στους πυρήνες επεξεργασίας έτσι ώστε να πάρουμε τα αρχικά μας δεδομένα χωρίς να έχουμε οποιαδήποτε απώλεια δεδομένων και να μπορέσουμε να υλοποιήσουμε την εφαρμογή μας. Όπως είναι λογικό έπρεπε να χρησιμοποιήσουμε εφαρμογές οι οποίες χρησιμοποιούν μεγάλο όγκο δεδομένων έτσι ώστε να μπορέσουμε να δούμε σημαντικές αλλαγές στο κόστος. Οι εφαρμογές που χρησιμοποιήσαμε στην εργασία αυτή είναι ο Πολλαπλασιασμός Πινάκων (Matrix Multiplication) και κυρίως εφαρμογές Βάσεων Δεδομένων (Database applications).

Στο έκθεση αυτή θα παρουσιάσουμε σχετική παρόμοια δουλεία που έγινε στο παρελθόν στα δύο κύρια θέματα της δουλείας μας που είναι ο επεξεργαστής Cell/BE(Κεφάλαιο 4) και η τεχνική της συμπίεσης δεδομένων(Κεφάλαιο 3). Στην συνέχεια θα γίνει αναλυτική περιγραφή των αλγορίθμων που χρησιμοποιήσαμε για την εργασία αυτή καθώς επίσης και μια αναλυτική περιγραφή του επεξεργαστή που χρησιμοποιούμε. Στην συνέχεια θα γίνει μικρή περιγραφή των εφαρμογών που χρησιμοποιούμε (Κεφάλαιο 5) και στο τέλος θα παραθέσουμε τα αποτελέσματα με την δική μας αξιολόγηση (Κεφάλαιο 6) αυτών όπως και τα συμπεράσματα που έχουμε εξάγει γενικά από την δουλεία αυτή (Κεφάλαιο 7).

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Τύποι και προγραμματισμός πολυεπεξεργαστών

Οι περισσότεροι επεξεργαστές με πολύ-πυρήνες σήμερα ανήκουν στην οικογένεια της *shared memory* αρχιτεκτονικής, επομένως οι καθυστερήσεις όσων αφορά την αντιγραφή και την μεταφορά δεδομένων μπορούν να θεωρηθούν σαν δεδομένο για ένα προγραμματιστή αφού γνωρίζει ότι δεν μπορεί να τις παρακάμψει. Για να τις παρακάμψει θα πρέπει να μελετήσει και να βρει τα μειονεκτήματα ή τα ελαττώματα του υλικού, ώστε να τα διορθώσει και να αυξήσει την επίδοση του. Παρόλ'αυτά υπάρχουν σήμερα πολύ-πύρηνοι επεξεργαστές οι οποίοι δεν ανήκουν στην ίδια οικογένεια αλλά ανήκουν στην distributed memory οικογένεια επεξεργαστών. Σε αυτούς τους επεξεργαστές η κατανομή των δεδομένων και τελικά ο χρόνος καθυστέρησης για την κατανομή/μεταφορά των δεδομένων στους διάφορους πυρήνες εναπόκειται στην ικανότητα του προγραμματιστή να διαχειριστεί σωστά τα δεδομένα και τους πόρους του συστήματος.

Η επίδοση που θα μπορεί να πετύχει ένας προγραμματιστής θα εξαρτάται από την ικανότητα του να προγραμματίζει σωστά την μεταφορά των δεδομένων αφού όλα εδώ γίνονται σε μορφή λογισμικού(software) κάτι το οποίο ανέκαθεν ήταν πιο αργό παρά μια υλοποίηση σε υλικό(hardware). Σκοπός μας στην εργασία αυτή είναι να δοκιμάσουμε και να υλοποιήσουμε διάφορες τεχνικές μεταφοράς δεδομένων, ή διαχείρισης δεδομένων με σκοπό να μειώσουμε τον συνολικό χρόνο/κόστος μεταφοράς των δεδομένων αυτών. Επιτρέποντας μας έτσι στον τέλος να αυξήσουμε την επίδοση της εφαρμογής μας αφαιρώντας από την εφαρμογή μας όλες τις περιττές καθυστερήσεις που αφορούν την κατανομή των δεδομένων στους διάφορους πυρήνες.

Για την δική μας έρευνα επιλέξαμε τον Cell Broadband Engine(Cell/BE) πολύ-πύρηνο ετερογενή επεξεργαστή. Ένας επεξεργαστής κατασκευασμένος από μια ομάδα εταιρειών, την Sony, την Toshiba και την IBM. Ο Cell/BE διαθέτει συνολικά 9 πυρήνες πάνω στο chip. Εμείς χρησιμοποιούμε το Play Station 3(PS3) για να μπορέσουμε να τρέξουμε εφαρμογές πάνω στον Cell/BE αφού το προϊόν αυτό έχει ενσωματωμένο τον επεξεργαστή αυτό. Στο προϊόν αυτό όμως οι 2 από τους 9 πυρήνες έχουν τεθεί εκτός λειτουργίας για τους προγραμματιστές αφού έχουν κρατηθεί από την κατασκευάστρια εταιρεία για λειτουργίες του συστήματος. Επομένως εμείς χρησιμοποιούμε 7 συνολικά πυρήνες. Στους 7 αυτούς πυρήνες έχουμε 1 τύπου PPE και 6 τύπου SPE. Ο Power Processor Element(PPE) πυρήνας είναι 64-bit PowerPC επεξεργαστής με 2-way multithreading, με 32 KB instructions και 32KB data κρυφή μνήμη επιπέδου 1 και 512KB επιπέδου 2. Οι πυρήνες τύπου Synergistic Processor Element(SPE) είναι RISC επεξεργαστές με 128-bit SIMD οργάνωση για single και double precision εντολές. Όλοι οι SPE's κατέχουν 256KB SRAM για εντολές και δεδομένα και ονομάζεται Local Store(LS). Το LS δεν είναι σαν μια κοινή κρυφή μνήμη ενός CPU αφού δεν παρέχεται καμία βοήθεια υλικού για την πρόβλεψη δεδομένων που θα χρησιμοποιηθούν, ούτε οποιαδήποτε πρωτόκολλα συνοχής δεδομένων. Η μεταφορά, η χρήση και η συνοχή των δεδομένων εναπόκειται αποκλειστικά στον προγραμματιστή.

1.2 Εφαρμογές με εντατική χρήση της μνήμης

Για την πειραματική αξιολόγηση των τεχνικών που αναφέρονται πιο κάτω θα χρησιμοποιήσουμε δύο εφαρμογές. Η πρώτη και κύρια εφαρμογή της εργασίας αυτής είναι οι βάσεις δεδομένων και η γνωστή εφαρμογή του πολλαπλασιασμού πινάκων(Matrix Multiplication). Υλοποιήσαμε επίσης και μια συνθετική εφαρμογή πάνω στην οποία εξετάσαμε πιο αναλυτικά και πιο συγκεκριμένα την τεχνική της συμπίεσης δεδομένων για να δούμε κατά πόσον μπορεί να μας προσφέρει αύξηση στην επίδοση. Σκοπός μας με τις εφαρμογές αυτές είναι να βρούμε τουλάχιστον ένα είδος εφαρμογής που να επιτρέπει την χρήση των τεχνικών αυτών και να μπορεί η εφαρμογή να κερδίσει επίδοση μέσα από αυτές.

Οι εφαρμογές που επιλέξαμε ακολουθούν κάποιο κοινό παρονομαστή. Και ο παρονομαστής αυτός είναι η μεγάλη χρήση της μνήμης. Επιλέξαμε εφαρμογές οι οποίες έχουν μεγάλες απαιτήσεις σε μνήμη έτσι ώστε να έχουμε μεγάλο αριθμό μεταφορών δεδομένων από την κεντρική μνήμη στα Local Stores. Με αυτό τον τρόπο θα έχουμε μια εφαρμογή η οποία θα είναι απαιτητική σε προσβάσεις στην μνήμη και συνεπώς το μειονέκτημα της το οποίο θα μειώνει την επίδοση της θα είναι οι μεταφορές των δεδομένων. Η επιλογή εφαρμογών πρέπει να γίνει με τον τρόπο αυτό έτσι ώστε να βεβαιωθούμε ότι θα δουλέψουμε πάνω σε ένα είδος εφαρμογών οι οποίες μπορούν να μας προσφέρουν περιθώρια αύξησης της επίδοσης με τις δικές μας τεχνικές. Με την μελέτη που έχουμε κάνει βρήκαμε ότι εάν η εφαρμογή δεν είναι απαιτητική σε δεδομένα από την μνήμη τότε οι μεταφορές των δεδομένων θα είναι λιγοστές και συνεπώς δεν θα μας προσφέρεται το απαιτούμενο περιθώριο για βελτίωση.

1.3 Κίνητρο και αναμενόμενα αποτελέσματα

Ο επεξεργαστής Cell/BE είναι επεξεργαστής που για την επικοινωνία μεταξύ των πυρήνων του χρησιμοποιά DMA transfers, δηλαδή μεταφορές δεδομένων απευθείας από την μνήμη. Η μεταφορές αυτές γίνονται μέσω ενός bus, το οποίο όμως έρχεται με ένα περιορισμό. Ο

περιορισμός αυτός είναι στο μέγεθος των δεδομένων που μπορούμε να στείλουμε κάθε μεταφορά (16KB). Το κίνητρο το δικό μας ήταν να βρούμε τρόπο για να μπορούμε να στέλνουμε περισσότερα δεδομένα ανά μεταφορά. Αυτό μας ώθησε να μελετήσουμε διάφορες τεχνικές και να καταλήξουμε στην συμπίεση των δεδομένων πριν την αποστολή καταφέρνοντας έτσι να μειώνουμε τον όγκο τον δεδομένων και σε κάθε μεταφορά να καταφέρνουμε να αποστέλλουμε περισσότερα δεδομένα στον ίδιο όγκο δεδομένων (16KB).

Αφού επιτύχουμε τον σκοπό της εργασίας μας που είναι να μειώσουμε τον όγκο των δεδομένων που αποστέλλουμε πάνω στους πυρήνες επεξεργασίας περιμένουμε να μειωθεί ο χρόνος που χρειάζεται για την μεταφορά των δεδομένων και συνεπώς να μειώσουμε και τον συνολικό χρόνο εκτέλεσης. Ο απώτερος στόχος μας είναι να αυξήσουμε με τις τεχνικές που θα χρησιμοποιήσουμε την επίδοση των εφαρμογών. Δεδομένου του ότι η μεταφορά των δεδομένων αποτελεί μέρος του υπολογισμού πάνω στον Cell/BE τότε καταφέρνοντας να μειώσουμε τις μεταφορές καταφέρνουμε να μειώσουμε και τον χρόνο εκτέλεσης.

1.4 Στόχος ατομικής διπλωματικής εργασίας

Εμείς λοιπόν αυτό που θα κάνουμε στην εργασία αυτή είναι να υλοποιήσουμε και να μελετήσουμε διάφορες προγραμματιστικές τεχνικές για να μειώσουμε τον χρόνο που καταναλώνεται κατά την μεταφορά των δεδομένων από την κεντρική μνήμη του συστήματος μας στα SPE’s και τελικά να αυξήσουμε την επίδοση του συνολικού μας συστήματος. Μερικές από τις τεχνικές που χρησιμοποιήσαμε και μελετήσαμε είναι blocking για τον σωστό καταμερισμό των δεδομένων, Single Instruction Multiple Data(SIMD) εντολές πάνω στα SPE’s για επεξεργασία περισσότερων δεδομένων σε μια εντολή, double-buffering τεχνικές για την μεταφορά των δεδομένων στους πυρήνες εργασίας(SPEs), προεπεξεργασία των δεδομένων εισόδου προτού αποσταλούν στους πυρήνες εργασίας, ομαδοποίηση των δεδομένων πριν την αποστολή.

Η κύρια τεχνική που υλοποιήσαμε, μελετήσαμε και θα αξιολογήσουμε στην εργασία αυτή είναι το Data Compression πάνω στα δεδομένα εισόδου. Δηλαδή με έναν αλγόριθμο συμπίεσης θα προσπαθήσουμε να συμπιέσουμε τα δεδομένα εισόδου από τον PPE σε ένα μικρότερο κομμάτι μνήμης. Καταφέρνοντας έτσι να μειώσουμε και τον συνολικό όγκο των δεδομένων που πρέπει να μεταφερθούν και συνεπώς μειώνοντας τον αριθμό των μεταφορών (transfers) των δεδομένων από την κύρια μνήμη στα Local Stores(LS) των SPEs.

1.5 Μεθοδολογία

Τα πειράματα τα οποία περιγράφουμε στην εργασία αυτή έχουν υλοποιηθεί σε γλώσσα προγραμματισμού C για την μηχανή Cell/BE. Για την εκτέλεση τους χρησιμοποιήσαμε συνολικά 7 επεξεργαστές από το σύστημα αυτό. Έχουμε χρησιμοποιήσει τον πραγματικό επεξεργαστή που βρίσκεται ενσωματωμένος στην παιχνιδοκονσόλα PlayStation 3 της Sony. Όσων αφορά τις εφαρμογές που έχουμε υλοποιήσει και δοκιμάσει τα σενάρια μας είναι οι Βάσεις Δεδομένων, ο Πολλαπλασιασμός Πινάκων και μια Συνθετική Εφαρμογή σχεδιασμένη από εμάς.

Στα πειράματα μας έχουμε εξετάσει κυρίως δύο τεχνικές που αφορούν την μεταφορά μνήμης και φαίνονται και στα αποτελέσματα και είναι το Double Buffering και η συμπίεση δεδομένων. Έχουμε υλοποιήσει σενάρια και με τις δύο αυτές τεχνικές και έχουμε δοκιμάσει επίσης και συνδυασμούς των τεχνικών αυτών. Τα αποτελέσματα των σεναρίων αυτών φαίνονται στο Κεφάλαιο 6 της Πειραματικής Αξιολόγησης.

1.6 Περιγραφή της δομής της διπλωματικής εργασίας

Η εργασία αυτή έχει χωριστεί σε 7 Κεφάλαια και σε κάθε κεφάλαιο αναλύουμε μια διαφορετική πτυχή της εργασίας μας. Στο Κεφάλαιο 1 κάνουμε μια μικρή εισαγωγή στην δουλεία μας και περιγράφουμε το τι θα δούμε στην συνέχεια της εργασίας αυτής καθώς

επίσης και τα κίνητρα και τον στόχο της εργασίας μας. Στο Κεφάλαιο 2 περιγράφουμε σημαντικές δουλείες που έχουν γίνει στο παρελθόν σε παρόμοια θέματα μέσα από επιστημονικά άρθρα και βιβλία που έχουμε μελετήσει κατά την υλοποίηση της εργασίας μας. Στο Κεφάλαιο 3 περιγράφουμε και αναλύουμε τους αλγόριθμους συμπίεσης που έχουμε δοκιμάσει και εξετάσει στην εργασία αυτή. Το κεφάλαιο αυτό περιέχει την σημαντικότερη υλοποίηση της δουλείας μας. Στο Κεφάλαιο 4 περιγράφουμε τον επεξεργαστή Cell/BE και εξηγούμε τα κυριότερα του χαρακτηριστικά. Στο Κεφάλαιο 5 περιγράφουμε τις εφαρμογές που έχουμε επιλέξει για να εξετάσουμε τις τεχνικές μας και αναλύουμε τους παράγοντες που μας έκαναν να επιλέξουμε τις εφαρμογές αυτές. Ακολούθως στο Κεφάλαιο 6 παραθέτουμε την Πειραματική Αξιολόγηση της δουλείας μας, όπου παρουσιάζουμε όλα τα αποτελέσματα που είχαμε και εξηγούμε τους λόγους που παίρνουμε τα αποτελέσματα αυτά. Και τελειώνοντας την εργασία αυτή στο Κεφάλαιο 7 παραθέτουμε τα συμπεράσματα μας βάση των αποτελεσμάτων που έχουμε πάρει. Επίσης στο κεφάλαιο αυτό δίνουμε και κάποιες ιδέες για μελλοντικές δουλείες πάνω στο δικό μας θέμα.

Κεφάλαιο 2

Σχετική Δουλεία

Για αυτή την εργασία έχω μελετήσει διάφορα άρθρα και βιβλία τα οποία αναφέρονται γενικά στο θέμα το δικό μου, άρθρα και βιβλία τα οποία αναφέρονται στον επεξεργαστή Cell/BE και άρθρα και βιβλία τα οποία αναφέρονται σε διάφορες τεχνικές όπως το data compression. Τα άρθρα και τα βιβλία αυτά αναφέρονται στο τελευταίο τμήμα της έκθεσης αυτής, στην βιβλιογραφία. Στο κομμάτι αυτό αναφέρω και σχολιάζω μερικές σημαντικές προηγούμενες εργασίες που έχουν γίνει σχετικά με το θέμα αυτό.

Για τον επεξεργαστή Cell/BE έχω μελετήσει εργασίες οι οποίες μελετούν την επίδοση του επεξεργαστή και προγραμματιστικά βιβλία που εξηγούν και αναλύουν προγραμματιστικές τεχνικές για τον επεξεργαστή [11],[12],[13],[14], [24], [34]. Κυρίως επικεντρώθηκα σε διάφορες τεχνικές προγραμματισμού, διαφορετικές από κοινούς επεξεργαστές, οι οποίες μπορούν να εφαρμοστούν πάνω στον Cell/BE. Τεχνικές όπως το SIMD [2],[12] κατά το οποίο οι vector processors SPEs μπορούν να επεξεργαστούν δεδομένα ομαδικά με την χρήση vector μεταβλητών. Οι vector μεταβλητές είναι μεταβλητές σταθερού μεγέθους για όλους τους τύπους(integers, floating point and character). Το μέγεθος των μεταβλητών αυτών είναι 16 Bytes και ανάλογα με το μέγεθος κάθε τύπου δεδομένων, αναλογούν και οι αντίστοιχοι αριθμοί μεταβλητών σε κάθε vector. Για παράδειγμα ένας vector float περιέχει 4 floating point αριθμούς. Με την χρήση των vector μεταβλητών και των συναρτήσεων που παρέχουν οι βιβλιοθήκες του SPE μπορούσαμε να επεξεργαστούμε περισσότερα δεδομένα ανά κύκλο. Αυτό μας έδωσε μια μεγάλη αύξηση στην επίδοση των εφαρμογών μας.

Άλλες τεχνικές προγραμματισμού που μελέτησα από τα βιβλία και τα άρθρα που αφορούν τον Cell/BE είναι οι διάφορες τεχνικές και οι διάφορες συναρτήσεις που υπάρχουν για την μεταφορά των δεδομένων από την κύρια μνήμη στα LS’s και αντίστροφα [11], [12]. Έπρεπε να βρούμε την καλύτερη επιλογή και η καλύτερη επιλογή ήταν αυτή που

εκτελούσε μια μεταφορά δεδομένων ταχύτερα από τις άλλες. Μέσα από την μελέτη αυτή βρήκαμε σημαντικές τεχνικές όπως το Double Buffering [11],[12] που στην έρευνα μας είναι ίσως ο σημαντικότερος ανταγωνιστής μας σε σχέση με την τεχνική της συμπίεσης δεδομένων που μελετούμε. Με την τεχνική αυτή μπορούμε να στέλνουμε δεδομένα στους πυρήνες επεξεργασίας παράλληλα με την εκτέλεση των πυρήνων αυτών. Με τον τρόπο αυτό μπορούσαμε να εκτελούμε δύο διεργασίες παράλληλα και να αποκρύπτουμε στην ουσία τον χρόνο μεταφοράς των δεδομένων [15].

Αφού αποφασίσαμε την μηχανή την οποία θα χρησιμοποιούσαμε για την έρευνα μας, ακολούθως ψάξαμε για εφαρμογές που θα μπορούσαμε να χρησιμοποιήσουμε. Για να έχουμε μια καλή και ολοκληρωμένη δουλεία μελετήσαμε εφαρμογές ευρέως αναγνωρισμένες που θα μπορούσαν να παρέχουν σωστά και καλά αποτελέσματα. Εφαρμογές όπως οι Βάσεις Δεδομένων [1], [3], [5], [7], [8] και Πολλαπλασιασμός Πινάκων [30] ήταν ιδανικές για την δική μας δουλεία. Επίσης υπήρχαν πολλές μελέτες πάνω σε αυτές τις εφαρμογές γενικά αλλά και πάνω στον Cell/BE κάτι που θα μας βοηθούσε στην δική μας δουλεία. Βρήκαμε ότι οι εφαρμογές αυτές απαιτούν μεγάλη ποσότητες μνήμης και αυτό είναι ένας παράγοντας που επηρεάζει την δική μας δουλεία. Οι εφαρμογές που θέλαμε να χρησιμοποιήσουμε έπρεπε να είναι απαιτητικές σε μνήμη για να μπορούμε να πάρουμε σωστά και ολοκληρωμένα αποτελέσματα.

Και τέλος έχω μελετήσει αρκετές δουλείες σε διάφορες τεχνικές όπως αυτές που θα χρησιμοποιήσουμε εμείς, όπως για παράδειγμα το Data Compression [16], [17], [18], [19], [20], [21], [22]. Μέσα από πολλά άρθρα αλλά και δύο βιβλία [28], [29] έχω μελετήσει την γενική ιδέα του compression, έχω κατανοήσει τον λόγο που γίνεται το compression, έχω μελετήσει διάφορους αλγόριθμους για compression και έχω επίσης μελετήσει συγκεκριμένους αλγόριθμους για data compression ανάλογα με το είδος των δεδομένων αφού στις διάφορες εφαρμογές που θα δοκιμάσουμε τα δεδομένα εισόδου είναι συγκεκριμένου τύπου. Υπάρχουν πάρα πολλές δουλείες και μελέτες πάνω σε compression δεδομένων που ανήκουν σε συγκεκριμένη ομάδα και τύπο δεδομένων. Όπως για παράδειγμα compression δεδομένων πάνω σε floating point δεδομένα [18][19][21][22]. Πιο γενικά έχω μελετήσει και διαφορετικές υλοποιήσεις για συμπίεση δεδομένων όπως το

hardware compression που χρησιμοποιείται σε cache μνήμες [23], [25] όπως και διάφορους αλγορίθμους που χρησιμοποιήσαν πολλές υλοποιήσεις για την εκτέλεση της συμπίεσης. Ο κυριότερος αλγόριθμος που χρησιμοποιείται και έχω μελετήσει είναι οι κώδικες Huffman [26], [27]. Οι κώδικες αυτοί χρησιμοποιούνται στον έτοιμο αλγόριθμο που έχουμε χρησιμοποιήσει και στην δουλεία μας, τον Zlib [31], [32].

Από την μελέτη που έκανα έχω βρει ότι δεν υπάρχει αρκετή προηγούμενη μελέτη και έρευνα στο θέμα αυτό. Δεν υπάρχει κάποια συγκεκριμένη μελέτη παρά μόνο γενικές μελέτες γύρω από την επεξεργαστή Cell και θεωρητικές αναφορές σε διάφορες υλοποιήσεις για επίδοση πάνω στον Cell. Στις περισσότερες έρευνες που έχουν γίνει δεν έχουν καν χρησιμοποιηθεί συγκεκριμένες εφαρμογές, απλώς μερικές από αυτές αναλύουν και αξιολογούν διάφορες τεχνικές προγραμματισμού. Υπάρχουν κάποιες δουλείες αναφορικά με βάσεις δεδομένων και οικονομικές εφαρμογές στις οποίες δεν χρησιμοποιήσαν όμως τεχνικές βελτίωσης της μεταφοράς των δεδομένων, όπως προσπαθούμε να κάνουμε εμείς.

Η δική μας δουλεία θα μπορούσαμε να πούμε ότι συνδυάζει περισσότερο λίγο από όλες τις δουλείες και έρευνες που έχουν προαναφερθεί. Συνδυάζοντας τον παράλληλο επεξεργαστή Cell/BE, με βασικές εφαρμογές απαιτητικές σε μνήμη και τεχνικές για μείωση του χρόνου μεταφοράς όπως η συμπίεση δεδομένων, έχουμε δημιουργήσει την δική μας δουλεία. Προσπαθώντας έτσι να δημιουργήσουμε μια διαφορετική δουλεία με ένα δυνατό βάθος (background) από προηγούμενες δουλείες.

Κεφάλαιο 3

Υλοποίηση Αλγορίθμων Συμπίεσης Δεδομένων

3.1 Εισαγωγή

Η συμπίεση δεδομένων (Data Compression) έχει γίνει αναγκαία την τελευταία δεκαετία στον κόσμο της πληροφορικής. Η ανάπτυξη του διαδικτύου και η μεταφορές δεδομένων μέσω αυτού έχει εξελίξει την συμπίεση δεδομένων σε μεγάλο βαθμό και σήμερα μπορούμε να βρούμε την τεχνική αυτή σχεδόν παντού. Από το διαδίκτυο, τα κινητά τηλέφωνα, τις μηχανές fax μέχρι και τα Mp3 players. Από το είδος των εφαρμογών που συναντούμε την συμπίεση δεδομένων μπορούμε εύκολα να δούμε πόσο σημαντική τεχνική είναι. Αυτό που θα δούμε στην συνέχεια είναι τι στην πραγματικότητα είναι η συμπίεση δεδομένων και τους λόγους που τόσο είναι σημαντική. Ακολούθως θα αναλύσουμε λεπτομερώς τους αλγορίθμους συμπίεσης που έχουμε χρησιμοποιήσει για την υλοποίηση της δικής μας δουλείας. Ο Zlib αλγόριθμος[31] είναι αλγόριθμος που έχουμε εντοπίσει και έχουμε προσαρμόσει στην δική μας δουλεία. Ο Αλγόριθμος 1 (AS1) είναι μια απλή υλοποίηση που

έχουμε φτιάξει εμείς για να συμπιέσουμε τα δικά μας δεδομένα, έχει σχεδιαστεί βάση του τύπου δεδομένων που χρησιμοποιούμε από την βάση δεδομένων.

Έχουμε χρησιμοποιήσει δύο αλγόριθμους στην δουλείας μας για να μπορέσουμε να συγκρίνουμε και αυτά τα σενάρια. Να δούμε τυχόν διαφορές που μπορεί να προκύψουν και να μπορούμε με ασφάλεια να πούμε ότι οι διαφορές αυτές οφείλονται στους αλγόριθμους. Έτσι αποφασίσαμε να χρησιμοποιήσουμε ένα ήδη υπάρχον αλγόριθμο συμπίεσης και ακολούθως να υλοποιήσουμε και ένα δικό μας αλγόριθμο. Ο πρώτος αλγόριθμος θα μας δώσει αποτελέσματα συμπίεσης από ένα πιο γενικό αλγόριθμο ενώ ο δεύτερος αλγόριθμος θα είναι συγκεκριμένος αλγόριθμος για την δική μας δουλεία και τα δικά μας δεδομένα και εφαρμογές.

3.2 Συμπίεση Δεδομένων

Σαν ορισμό για την συμπίεση δεδομένων μπορούμε να πούμε ότι οι αλγόριθμοι συμπίεσης δεδομένων χρησιμοποιούνται για να μειώσουν των αριθμό των bits που χρειάζονται για να αναπαραστήσουμε ένα κομμάτι δεδομένων που μπορεί να είναι είτε εικόνα, είτε βίντεο, είτε μουσική, είτε οτιδήποτε άλλο είδος που αναπαριστάται στην μνήμη με bits[28]. Τα δεδομένα μπορεί να είναι οτιδήποτε, από χαρακτήρες μέχρι αριθμούς. Ο λόγος που χρειαζόμαστε την συμπίεση δεδομένων σήμερα είναι το γεγονός ότι κάθε μέρα που περνάει τα δεδομένα που αναπαριστώνται σε ψηφιακή μορφή αυξάνονται με μεγαλύτερο ρυθμό από ότι αυξάνεται ο διαθέσιμος χώρος. Κυριότερος λόγος όμως είναι ότι αν δεν χρησιμοποιηθεί η συμπίεση πάνω σε κάτι το οποίο θέλουμε να μετακινήσουμε μέσα από κάποιο δίκτυο για παράδειγμα τότε ο χρόνος μεταφοράς του θα είναι πολύ μεγάλος. Χρησιμοποιώντας την συμπίεση καταφέρνουμε να μειώσουμε τον όγκο των δεδομένων και συνεπώς μειώνουμε και τον χρόνο μεταφοράς.

Η συμπίεση δεδομένων δεν είναι κάτι καινούργιο όχι μόνο στον κόσμο της πληροφορικής αλλά γενικά. Ένα δείγμα συμπίεσης δεδομένων εμφανίστηκε με τα σήματα Morse, όπου χρησιμοποιούνταν για επικοινωνία από τον $19^{ο}$ αιώνα. Στον κώδικα Morse, ο Samuel

Morse [33] έχει αναθέσει γράμματα του αλφαβήτου που χρησιμοποιούνται πιο συχνά σε μια λέξη ή πρόταση σε πιο μικρές αναπαραστάσεις και γράμματα που χρησιμοποιούνται λιγότερο σε μεγαλύτερες αναπαραστάσεις. Αυτό κατάφερε να μειώσει τον όγκο δεδομένων που μεταφέρεται με την αναπαράσταση του Morse. Αυτό ήταν μόνο ένα μικρό δείγμα της συμπίεσης δεδομένων, σήμερα η τεχνική αυτή μπορεί να θεωρηθεί ακόμα και ξεχωριστή επιστήμη.

3.2.1 Τεχνικές Συμπίεσης

Όταν αναφερόμαστε σε ένα αλγόριθμο συμπίεσης στην πραγματικότητα μιλάμε για δύο αλγορίθμους. Όταν συμπιέσουμε κάποια δεδομένα έτσι ώστε να μπορέσουμε να τα στείλουμε κάπου ή να τα μεταφέρουμε από κάπου τότε όταν τελειώσει η μεταφορά θα θέλουμε να τα επαναφέρουμε στην αρχική τους μορφή έτσι ώστε να μπορέσουμε να τα επεξεργαστούμε. Συνεπώς χρειαζόμαστε και έναν δεύτερο αλγόριθμο ο οποίος θα κάνει ακριβώς την αντίθετη δουλεία που κάνει ο πρώτος όταν τα συμπιέζει. Ο αλγόριθμος αυτός που χρησιμοποιείται για την επαναφορά/αποσυμπίεση των συμπιεσμένων δεδομένων ονομάζεται αλγόριθμος αποσυμπίεσης (Decompression algorithm). Ένας αλγόριθμος συμπίεσης μπορεί να έχει περισσότερους από ένα αλγόριθμους αποσυμπίεσης οι οποίοι κατατάσσονται σε δύο κατηγορίες. Το είδος που ανήκει κάθε αλγόριθμος καθορίζεται από τις απαιτήσεις που έχει η αποσυμπίεση των δεδομένων. Οι κατηγορίες αυτές είναι η συμπίεση χωρίς απώλειες (lossless compression) κατά την οποία στην αποσυμπίεση δεν επιτρέπεται να έχουμε οποιεσδήποτε απώλειες, συνεπώς θέλουμε κατά την αποσυμπίεση να επαναφέρουμε τα δεδομένα που συμπιέσαμε στην απόλυτη αρχική τους μορφή. Η δεύτερη κατηγορία συμπίεσης λειτουργεί με ανοχή σφαλμάτων (lossy compression) και σε αυτή την περίπτωση μπορούμε να ανεχθούμε κάποια σφάλματα κατά την αποσυμπίεση και να έχουμε ουσιαστικά διαφορετικά δεδομένα από ότι είχαμε πριν την συμπίεση.

3.2.1.1 Lossless Compression

Η κατηγορία του *Lossless Compression* δεν επιτρέπει να έχουμε απώλειες δεδομένων κατά την αποσυμπίεση. Θέλουμε το αποτέλεσμα της αποσυμπίεσης να είναι με απόλυτη

ακρίβεια το ίδιο με τα δεδομένα πριν λάβει μέρος η συμπίεση. Υπάρχουν πάρα πολλές εφαρμογές στις οποίες δεν θέλουμε να έχουμε οποιαδήποτε απώλεια πληροφορίας όπως σε οικονομικές εφαρμογές ή μεταφορές οικονομικών δεδομένων, σε ιατρικές εφαρμογές και ιατρικά αρχεία και κυρίως δεν θέλουμε να έχουμε οποιαδήποτε απώλεια στην συμπίεση δεδομένων τύπου κειμένου. Είναι πολύ σημαντικό σε τέτοιου είδους εφαρμογές, όπου η παραμικρή αλλαγή στα δεδομένα μπορεί να επιφέρει καταστροφικά αποτελέσματα, να μην υπάρχει οποιαδήποτε απώλεια και κατά την αποσυμπίεση.

Αυτή είναι και η κατηγορία που χρειαζόμαστε εμείς για την εργασία αυτή γιατί ασχοληθήκαμε με εφαρμογές τύπου βάσεων δεδομένων όπου χρειαζόμαστε άθικτα τα δεδομένα μας κατά την αποσυμπίεση έτσι ώστε να μπορούμε να επεξεργαστούμε οποιοδήποτε query χρειαστεί. Με οποιαδήποτε αλλαγή στα δεδομένα θα είχαμε διαφορετικά αποτελέσματα κατά την αποπεράτωση των επερωτήσεων κάτι το οποίο θα σήμαινε λάθος εκτέλεση της εφαρμογής. Οι βάσεις δεδομένων ανήκουν στις κατηγορίες των εφαρμογών οι οποίες δεν μπορούν να ανεχθούν οποιαδήποτε απώλεια πληροφορίας.

Αυτή η κατηγορία συμπίεσης έρχεται και με ένα μειονέκτημα. Αναγκαστικά αφού δεν μπορούμε να έχουμε οποιαδήποτε απώλεια πληροφορίας τότε οι αλγόριθμοι οι οποίοι υλοποιούνται για την κατηγορία αυτή πετυχαίνουν μικρότερη αναλογία συμπίεσης παρά αν επιτρέπαμε απώλεια πληροφορίας. Το μειονέκτημα αυτό καταργείται στην επόμενη κατηγορία.

3.2.1.2 Lossy Compression

Στην κατηγορία αυτή ανήκουν εφαρμογές οι οποίες δεν επηρεάζονται από την απώλεια πληροφορίας. Εφαρμογές όπως ομιλία σε τηλεφωνικές συσκευές ή αναπαραγωγή βίντεο. Σε τέτοιου είδους εφαρμογές όταν υπάρχει απώλεια πληροφορίας κατά την συμπίεση και την αποσυμπίεση ο βαθμός που επηρεάζεται το αποτέλεσμα είναι πολύ μικρός και οι διαφορές δεν είναι καθόλου εμφανής σε αυτό που θα χρησιμοποιήσει την πληροφορία αυτή. Συνεπώς επιτρέπουμε να έχουμε κάποια απώλεια πληροφορίας έτσι ώστε να μπορέσουμε να επιτύχουμε μεγαλύτερο βαθμό συμπίεσης και να μειώσουμε ακόμα

περισσότερο τον όγκο των δεδομένων από ότι με την χρήση Lossless Compression αλγορίθμου.

3.3 Συμπίεση Δεδομένων στον Cell/BE

Όλες οι πληροφορίες που έχουμε αναφέρει πιο πάνω εφαρμόζονται σήμερα σε εφαρμογές διαδικτύου για μεταφορά δεδομένων, σε συστάδες υπολογιστών όπου πρέπει να γίνει μεταφορά δεδομένων για να γνωστοποιηθούν σε όλους τους υπολογιστές αφού τα συστήματα αυτά δεν διαθέτουν κοινή μνήμη. Συνήθως οι υπολογιστές που απαρτίζουν τα συστήματα αυτά βρίσκονται σε διαφορετικό γεωγραφικό μέρος, έτσι η επικοινωνία μεταξύ τους απαρτίζει ένα μεγάλο μέγεθος του υπολογισμού τους.

Στην δική μας περίπτωση δεν έχουμε συστάδες υπολογιστών, ούτε οι πυρήνες του συστήματος βρίσκονται σε διαφορετικό γεωγραφικό μέρος αλλά έχουμε τα άλλα δύο κοινά χαρακτηριστικά αυτών των συστημάτων τα οποία είναι η μη κοινή μνήμη αφού κάθε πυρήνας του συστήματος μας έχει δική του μνήμη (Local Store) και παίρνει τα δεδομένα του από την ‘κοινή’ κεντρική μνήμη, από την οποία όμως δεν έχει απευθείας πρόσβαση. Η μεταφορά των δεδομένων πρέπει να γίνει μέσω κάποιου άλλου μέσου που στον Cell/BE είναι το Element Interconnect Bus (EIB). Η δεύτερη ομοιότητα με τα συστήματα που αναφέρουμε πιο πάνω είναι βεβαίως ότι και στον Cell/BE η επικοινωνία απαρτίζει ένα μεγάλο κομμάτι του υπολογισμού. Για τον λόγο αυτό αποφασίσαμε να δοκιμάσουμε να συμπιέσουμε τα δεδομένα εισόδου των εφαρμογών που θα χρειαζόταν να αποστείλουμε πάνω στα SPE’s με κυριότερο σκοπό να μειώσουμε τον όγκο των δεδομένων που μεταφέρουμε, μειώνοντας έτσι τον αριθμό μεταφορών δεδομένων και συνεπώς να μειώσουμε τον συνολικό χρόνο μεταφοράς των δεδομένων. Απώτερος μας σκοπός ήταν να μειώσουμε τον χρόνο εκτέλεσης των εφαρμογών, μειώνοντας τον χρόνο μεταφοράς δεδομένων.

Παρόλο που έχουμε κάποια κοινά χαρακτηριστικά που φαινομενικά μας επιτρέπει να σκεφτόμαστε θετικά για την εφαρμογή της συμπίεσης δεδομένων πάνω στον Cell/BE,

υπάρχουν και κάποια χαρακτηριστικά της μηχανής τα οποία επιδρούν μειονεκτικά στην όλη μας προσπάθεια. Στο δικό μας σύστημα κατά την μεταφορά των δεδομένων υπάρχει περιορισμός στο μέγεθος των δεδομένων που μπορούμε να αποστείλουμε (16 Kbytes), γεγονός που μας περιορίζει στον τρόπο συμπίεσης των δεδομένων. Για παράδειγμα αναγκαστήκαμε να σπάσουμε τα δεδομένα μας σε πολλά μικρά κομμάτια (chunks) δεδομένων τα οποία συμπιέζαμε ξεχωριστά για τον λόγο ότι δεν θα μπορούσαμε να αποσυμπιέσουμε τα δεδομένα μετά αφού θα έλειπα σημαντικές πληροφορίες από τα δεδομένα που αποστείλαμε μέσα στα 16KB, πληροφορίες οι οποίες είναι σημαντικές για την αποσυμπίεση. Αυτό μας ανάγκαζε να εκτελούμε περισσότερες πράξεις και να έχουμε περισσότερο προγραμματιστικό κόπο έτσι ώστε να μπορέσουμε να ολοκληρώσουμε την εργασία. Επίσης μετά από μερικές παρατηρήσεις πάνω στις αναλογίες συμπίεσης των δεδομένων είδαμε ότι η αναλογία συμπίεσης ήταν μικρότερη σε μικρότερο μέγεθος δεδομένων για τον *αλγόριθμο zlib.* Πρακτικά αυτό σημαίνει ότι χάναμε από την συμπίεση που θα μπορούσαμε να έχουμε.

Ακόμα ένα σημαντικό μειονέκτημα που ήρθαμε αντιμέτωποι είναι το μικρό μέγεθος του Local Store των SPE’s (256KB). Το μικρό αυτό μέγεθος μας περιόριζε στον αριθμό των δεδομένων που μπορούσαμε να αποστείλουμε πάνω σε ένα πυρήνα. Πιο συγκεκριμένα σε ορισμένα από τα σενάρια μας που εκτελέσαμε βρήκαμε ότι λόγο της μεγάλης συμπίεσης που είχαμε σε συνδυασμό με την τεχνική του Double-Buffering δεν μπορούσαμε να αποστείλουμε 16KB πάνω σε κάποιο SPE για τον λόγο ότι κατά την αποσυμπίεση το μέγεθος των δεδομένων που υπήρχαν μέσα στο Local Store ήταν μεγαλύτερο από το μέγεθος του LS.

Ένα άλλο μειονέκτημα που βρήκαμε στην πορεία της υλοποίησης μας ήταν ο μεγάλος χρόνος που χρειάζεται για την αποσυμπίεση των δεδομένων πάνω στα SPE’s. Παρόλο που μειώναμε τις μεταφορές των δεδομένων πάνω στα SPE’s, δημιουργήσαμε ένα άλλο πρόβλημα που αναγκαστικά μας καταναλώνει ένα μεγάλο μέρος του χρόνο του υπολογισμού μας. Προσπαθώντας να επιλύσουμε το πρόβλημα αυτό δοκιμάσαμε αρκετές διαφοροποιήσεις, όπως την εκτέλεση πάνω σε συμπιεσμένα δεδομένα για τον AS1. Οι διαφοροποιήσεις αυτές εξηγούνται πιο αναλυτικά στις Υποενότητες 3.4 και 3.5 αυτού του

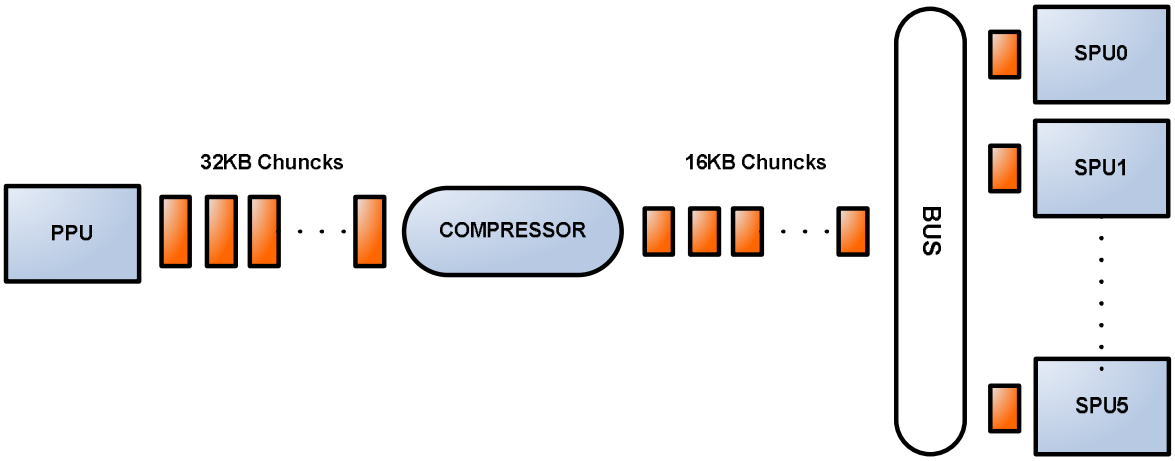
κεφαλαίου και τα αποτελέσματα τους φαίνονται στο Κεφάλαιο 6 της Πειραματικής Αξιολόγησης.

3.4 Zlib Αλγόριθμος Συμπίεσης

Όπως αναφέραμε και πιο πάνω στην εργασία αυτή χρησιμοποιήσαμε δύο αλγόριθμους συμπίεσης. Ο πρώτος λοιπόν αλγόριθμος που χρησιμοποιήσαμε είναι ο Zlib αλγόριθμος[31]. Ο αλγόριθμος συμπίεσης που χρησιμοποιείται εδώ είναι ο ίδιος με τους γνωστούς αλγόριθμους συμπίεσης αρχείων *gzip* και *Zip*, αφού έχουν γραφτεί από τους ίδιους προγραμματιστές. Όπως αναφέρουν και οι συγγραφείς του αλγορίθμου αυτού, ο Zlib είναι μια παραλλαγή του LZ77 (Lempel-Ziv 1977)[32], [35].

Παρόλο που ο αλγόριθμος αυτός είναι ένας καλός αλγόριθμος συμπίεσης αντιμετωπίσαμε σοβαρά προβλήματα κατά την λειτουργία του που μας έδιναν αρνητικά αποτελέσματα. Το κυριότερο πρόβλημα που αντιμετωπίσαμε ήταν ο μεγάλος χρόνος αποσυμπίεσης που χρειάζεται για την αποσυμπίεση των δεδομένων πάνω στα SPE's. Ό χρόνος αυτός ξεπερνά τον χρόνο αποστολής δεδομένων και έτσι δεν μπορούσαμε να τον 'κρύψουμε' με αποτέλεσμα να φαίνεται ότι η συμπίεση των δεδομένων με τον αλγόριθμο αυτό ήταν αχρείαστη και καθόλου αποδοτική.

Λόγω της ιδιομορφίας που εξηγούμε πιο κάτω του αλγορίθμου αυτού έπρεπε να σπάσουμε τα δεδομένα μας από την αρχή σε κομμάτια έτσι ώστε να τα συμπιέσουμε ξεχωριστά γιατί εάν συμπιέζαμε ολόκληρη την βάση μας από την αρχή τότε δεν θα μπορούσαμε να μοιράσουμε τα δεδομένα μας στους διάφορους πυρήνες επεξεργασίας. Η διαδικασία που ακολουθήσαμε ήταν να χωρίσουμε τα δεδομένα μας σε κομμάτια των 32KB έτσι ώστε μετά την συμπίεση να φτάνουν στα 16KB για να μπορούν να αποσταλούν μέσα από το bus, αφού το bus μπορεί να μεταφέρει μόνο 16KB ανά μεταφορά (Σχήμα 3.1).

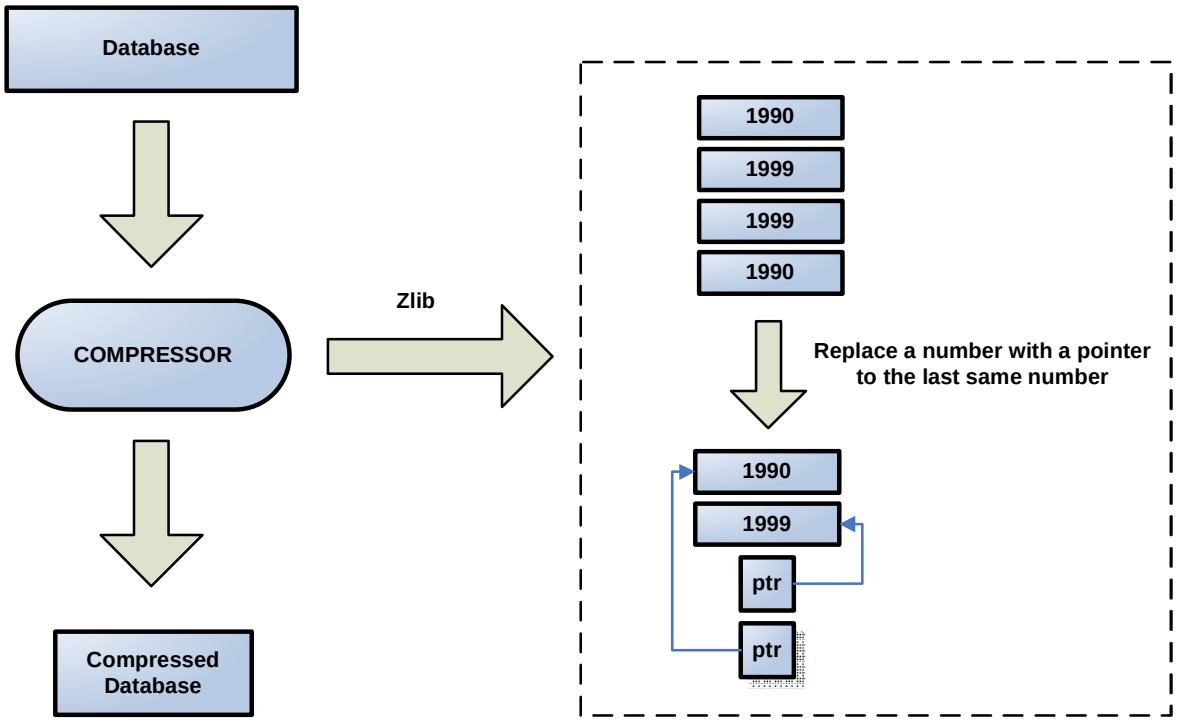


Σχήμα 3.1: *Χωρισμός δεδομένων σε κομμάτια, συμπίεση και αποστολή*

Αλγόριθμος Συμπίεσης

Αυτό που κάνει ο αλγόριθμος αυτός είναι να βρίσκει διπλές (όμοιες) σειρές χαρακτήρων (strings) μέσα στα δεδομένα εισόδου και τα οποία θα πάνε για συμπίεση. Η δεύτερη εμφάνιση της σειράς αυτής από χαρακτήρες την αντικαθιστά με ένα δείκτη προς την πρώτη εμφάνιση αυτής. Ο δείκτης αυτός εμφανίζεται σαν ένα ζευγάρι τιμών, (distance, length). Το ζευγάρι αυτό μπορεί με απλά λόγια να ερμηνευτεί και σαν 'Καθεμία από τους επόμενους *length* χαρακτήρες είναι ίσες με τον χαρακτήρα ακριβώς *distance* χαρακτήρες πιο πίσω μέσα στο ασυμπίεστο ρεύμα δεδομένων'[35]. Η τιμή του *distance* περιορίζεται μέχρι τα 32KB και η τιμή του *length* μέχρι τα 258 bytes. Όταν ένα string δεν παρουσιάζεται για $2^{η}$ φορά μέσα στα 32KB τότε αυτό το string συνεχίζει και αποστέλλεται στο επόμενο βήμα σαν μια ακολουθία από bytes. Οι σειρές από bytes ή τα μήκη (lengths) συμπιέζονται με ένα Huffman δέντρο[31], [32], και οι αποστάσεις (distances) συμπιέζονται με ένα άλλο. Τα δέντρα είναι αποθηκευμένα στην αρχή του κάθε Block των συμπιεσμένων δεδομένων. Ένα block συμπιεσμένων δεδομένων μπορεί να έχει οποιοδήποτε μέγεθος φτάνει να μπορεί να χωρέσει ολόκληρο μέσα στην διαθέσιμη μνήμη. Τα block αυτά καθορίζονται από την συνάρτηση deflate () η οποία υπάρχει μέσα στην μηχανή συμπίεσης Zlib και είναι υπεύθυνη για την συμπίεση των δεδομένων. Επομένως το πότε θα τελειώσει

ένα block και θα αρχίσει η δημιουργία καινούργιων δέντρων για να παράξουν ένα καινούργιο block εναπόκειται σε αυτήν.



Σχήμα 3.2: *Παράδειγμα εκτέλεσης συμπίεσης δεδομένων Zlib αλγορίθμου*

Όλα τα διπλά strings βρίσκονται με την βοήθεια ενός hash table. Και για να κερδίσει χρόνο και να εκμεταλλευτεί την κωδικοποίηση Huffman [26], [27], κατά την αναζήτηση μέσα στον πίνακα αυτό, ψάχνει πρώτα τα stings τα οποία έχουν αποθηκευτεί μέσα πιο πρόσφατα.

Αλγόριθμος αποσυμπίεσης

Κατά την αποσυμπίεση ο αλγόριθμος εκμεταλλεύεται την κωδικοποίηση Huffman για να μπορεί να εκτελεστεί γρήγορα. Το πιο σημαντικό χαρακτηριστικό είναι το γεγονός ότι μικρότεροι κώδικες (codes) εμφανίζονται πιο συχνά παρά μεγαλύτεροι σε μήκος κώδικες.

Για τον λόγο αυτό ο αλγόριθμος προσπαθεί να αποκωδικοποιήσει μικρότερους κώδικες πιο γρήγορα και αφήνει τα μεγαλύτερους κώδικες να πάρουν περισσότερο χρόνο για αποκωδικοποίηση.

Η αποκωδικοποίηση και κατά συνέπεια η αποσυμπίεση γίνεται από την συνάρτηση inflate(). Η συνάρτηση αυτή ξεκινά και κατασκευάζει ένα πίνακα ο οποίος καλύπτει ορισμένο αριθμό bits δεδομένων εισόδου για αποσυμπίεση. Ο αριθμός αυτός των bits πρέπει να είναι μικρότερος από το μεγαλύτερος σε μήκος κώδικα που υπάρχει. Παίρνει τον κώδικα αυτό και αρχίζει αναζήτηση του μέσα στον πίνακα. Ο πίνακας θα μας πει εάν ο επόμενος κώδικας είναι τόσα bits ή λιγότερα και πόσα. Εάν είναι τόσα τότε θα μας επιστρέψει την τιμή που αντιπροσωπεύει ο κώδικας αυτός και συνεπώς τα αποσυμπιεσμένα δεδομένα αλλιώς θα μας πάει στο επόμενο επίπεδο του πίνακα όπου η συνάρτηση Inflate() θα αναγκαστεί να διαβάσει περισσότερα bits, δηλαδή ένα μεγαλύτερο κώδικα και θα προσπαθήσει να τον αποκωδικοποιήσει.

3.5 Αλγόριθμος Συμπίεσης 1 – AS1

Κατά την μεταφορά των δεδομένων από την κεντρική μνήμη στο Local Store των SPE's τα δεδομένα που χρειάζονται είναι 4 αριθμοί μονής κινητής υποδιαστολής (floating point numbers). Βασιζόμενοι σε αυτό το στοιχείο σχεδιάσαμε τον AS1.

Τα δεδομένα που περιέχει κάθε record και που μεταφέρονται από την βάση δεδομένων είναι τα εξής (περισσότερες πληροφορίες για την εφαρμογή στο Κεφάλαιο 5):

1. Extended Price
2. Discount
3. Year
4. Quantity

Μελετώντας την βάση δεδομένων[36] που χρησιμοποιήσαμε και τον τύπο των δεδομένων που κρατά η βάση ανακαλύψαμε τα εξής για κάθε ένα από τα πιο πάνω δεδομένα:

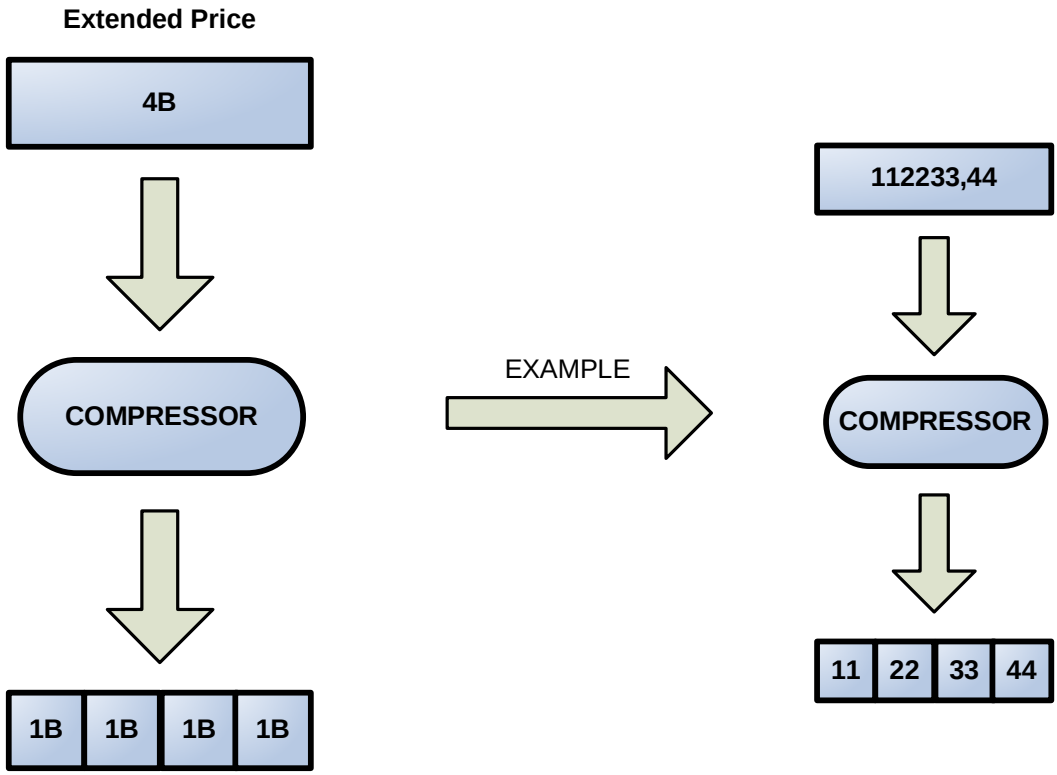
1. ***Extended Price:*** Μπορεί στο μέγιστο να είναι εξαψήφιος δεκαδικός αριθμός με δύο δεκαδικά ψηφία δεξιά της υποδιαστολής. (παράδειγμα: xx-xx-xx,yy)
2. ***Discount:*** Είναι δεκαδικός αριθμός μεταξύ των 0 – 1 και δεξιά της υποδιαστολής κρατά δύο ψηφία. (παράδειγμα: x,yy όπου x = 0 | 1)
3. ***Year:*** είναι ένας τετραψήφιος αριθμός από το 1990 – 1999 χωρίς δεκαδικά ψηφία
4. ***Quantity:*** Είναι ένας διψήφιος αριθμός από το 0 – 99

Ο δικός μας ο αλγόριθμος (AS1) μετατρέπει τα στοιχεία αυτά από αριθμούς των 4 Byte σε χαρακτήρες του 1 Byte. Ο αλγόριθμος αυτός υλοποιείται σε δύο στάδια. Κατά το $1^{ο}$ στάδιο αλλάζουμε την μορφή του αριθμού ***Extended Price*** και κατά το $2^{ο}$ στάδιο συμπιέζουμε τους υπόλοιπους τρεις αριθμούς που διαβάζουμε από την βάση δεδομένων. Πιο κάτω εξηγούμε αναλυτικά την λειτουργία των δύο αυτών σταδίων και στο Παράρτημα Α παραθέτουμε τον κώδικα C που υλοποιήσαμε για την συμπίεση και την αποσυμπίεση.

Συμπίεση

$1^{ο}$ Στάδιο

Το στοιχείο αυτό (extended price) μπορεί να είναι αριθμός με 0 μέχρι και 6 ψηφία. Για τον λόγο αυτό δεν μπορούμε να εφαρμόσουμε απευθείας τον αλγόριθμο μας όπως κάνουμε και στα άλλα στοιχεία. Έτσι ο αριθμός αυτός έπρεπε να σταλεί ανέπαφος έτσι ώστε να μπορέσουμε να εκτελέσουμε τις πράξεις του query. Αλλά επειδή τα υπόλοιπα στοιχεία που θα αποστείλουμε μετά την συμπίεση θα είναι τύπου χαρακτήρα και ενός byte έπρεπε να βρούμε τρόπο να μετατρέψουμε το στοιχείο αυτό σε τύπο χαρακτήρα χωρίς όμως να χρησιμοποιήσουμε περισσότερα bytes από ότι αρχικά χρησιμοποιούσε το στοιχείο αυτό (δηλαδή 4 Bytes). Αυτό που κάνουμε είναι να σπάζουμε τον αριθμό αυτό σε 4 κομμάτια (chunks) των δύο ψηφίων έτσι ώστε κάθε κομμάτι να μπορεί να χωρέσει σε 1 Byte (επειδή η αναπαράσταση του byte είναι 8 bits, σημαίνει ότι δέχεται αριθμούς από το 0 – 256). Με αυτό τον τρόπο αφού σπάσουμε το στοιχείο αυτό βάζουμε κάθε κομμάτι σε ένα Byte. Στο στάδιο αυτό δεν συμπιέζουμε κάτι αλλά μετατρέπουμε τον αριθμό που μας εμπόδιζε στην συμπίεση σε κάτι το οποίο μπορούμε να συμπιέσουμε.



Σχήμα 3.3: *Παράδειγμα εκτέλεσης 1ου σταδίου συμπίεσης* AS1

2ο Στάδιο

Στο δεύτερο στάδιο της συμπίεσης ακολουθούμε τα εξής βήματα έτσι ώστε να συμπιέσουμε τα υπόλοιπα τρία στοιχεία του κάθε record:

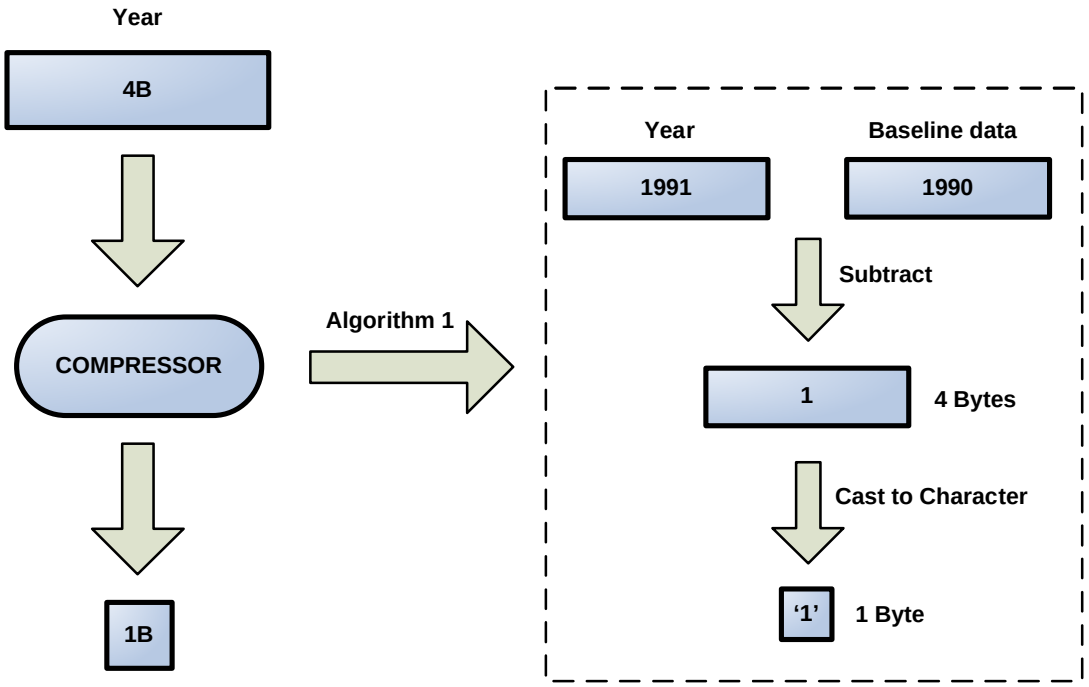
1. Διατρέχουμε ολόκληρη τη βάση δεδομένων για να βρούμε τις μικρότερες τιμές για όλα τα στοιχεία που έχουμε (discount, year and quantity).

2. Αποθηκεύουμε αυτές τις τιμές σε ένα πίνακα σαν αρχικές τιμές και αργότερα τις αποστέλλουμε στα SPE's για να μπορέσει να γίνει η αποσυμπίεση (decompression).

3. Διατρέχουμε ξανά την βάση δεδομένων και για κάθε τιμή στο record αφαιρούμε από το μικρότερο στοιχείο που είχαμε προηγουμένως βρει. Όλες οι τιμές τώρα είναι αριθμοί μεταξύ του 0 – 256 και μπορούν να αναπαρασταθούν σε 8 bits/1 byte.

3.1 Για το στοιχείο *discount* πολλαπλασιάζουμε επίσης με το 100 έστω ώστε να γίνει ακέραιος ο αριθμός μας.

4. Αποθηκεύουμε όλα αυτά τα καινούργια στοιχεία σε ένα νέο πίνακα (βάση δεδομένων) και ακολούθως τον αποστέλλουμε στα SPE's για τον υπολογισμό.



Σχήμα 3.4: *Παράδειγμα εκτέλεσης 2^{ου} σταδίου συμπίεσης* AS1 *για ένα στοιχείο*

Ο αλγόριθμος αυτός έχει βαθμό συμπίεσης 2.28 γιατί αρχικά 1 record είχε 4 στοιχεία των 4 bytes που μας κάνουν 16 bytes συνολικά. Μετά την συμπίεση το *extended price* παραμένει 4 bytes αλλά τα άλλα 3 στοιχεία γίνονται από 4 bytes το καθένα σε 1 byte το καθένα. Συνεπώς έχουμε 3 bytes από εδώ και 4 από το *extended price*, συνολικά 7 από τα 16 που είχαμε αρχικά. Συνεπώς ο βαθμός συμπίεσης ανεβαίνει στα 2.28.

Λόγω της ιδιαιτερότητας του EIB του Cell/BE που κατά την μεταφορά δεδομένων τα δεδομένα αυτά πρέπει να έχουνε μέγεθος πολλαπλάσιο του 16 αναγκαστήκαμε να προσθέσουμε ακόμα 1 byte (dummy byte) σε κάθε συμπιεσμένο record έτσι ώστε να

έχουμε δεδομένα πολλαπλάσια του 16. Αυτό έχει σαν αποτέλεσμα να μειώνεται από 2.28 σε 2 ο βαθμός συμπίεσης. Με άλλα λογία μειώνουμε το μέγεθος των δεδομένων σε ακριβώς τα μισά. Και όπως φαίνεται και στο Κεφάλαιο 6 των αποτελεσμάτων μειώνονται και οι μεταφορές των δεδομένων (DMA transfers) στις μισές.

Όταν συμπιεστούν και αποσταλούν στα SPE's αυτά τα στοιχεία θα πρέπει να αποσυμπιεστούν έτσι ώστε να μπορούμε να τα επεξεργαστούμε στην συνέχεια. Εδώ μπαίνει ο 2^{ος} αλγόριθμος, ο αλγόριθμος αποσυμπίεσης. Για την αποσυμπίεση έχουμε δύο παραλλαγές του αλγορίθμου. Για πρακτικούς λόγους τους ονομάσαμε *Αλγόριθμος Αποσυμπίεσης 1.1* και *Αλγόριθμος Αποσυμπίεσης 1.2*. Στον αλγόριθμος αποσυμπίεσης 1.1 αποσυμπιέζουμε τα δεδομένα και δημιουργούμε ακριβώς τα ίδια δεδομένα με αυτά που είχαμε αρχικά πριν την συμπίεση. Αυτή η διαδικασία όμως είναι αρκετά χρονοβόρα όπως εξηγούμε και πιο κάτω, για το λόγο αυτό δημιουργήσαμε τον Αλγόριθμο αποσυμπίεσης 1.2, όπου ακολουθούμε μια διαφορετική προσέγγιση και δεν επαναφέρουμε πλήρως τα δεδομένα στην αρχική τους μορφή.

Αλγόριθμος αποσυμπίεσης 1.1

Στον αλγόριθμο αυτό όπως αναφέρουμε και πιο πάνω αποσυμπιέζουμε πλήρως τα δεδομένα μας. Για την επίτευξη αυτού ακολουθούμε την ακριβώς αντίθετη διαδικασία που ακολουθήσαμε κατά την συμπίεση. Τα βήματα της αποσυμπίεσης είναι τα ακόλουθα:

1. Κάθε SPE παραλαμβάνει τα baseline δεδομένα που προηγουμένως υπολογίσαμε πάνω στον PPE.

2. Προσθέτουμε πάνω σε κάθε συμπιεσμένο δεδομένο που παραλάβαμε την αντίστοιχη τιμή από τα baseline δεδομένα

2.1 Διαιρούμε την τιμή στο *Discount* πεδίο με το 100 για να παράξουμε την αρχική τιμή αφού κατά την συμπίεση είχαμε πολλαπλασιάσει το πεδίο αυτό με 100.

3. Συνδυάζουμε τους 4 διψήφιους αριθμούς του *Extended Price* για να δημιουργήσουμε την αρχική τιμή

4. Αποθηκεύουμε τις αποσυμπιεσμένες τιμές σε διανύσματα (floating point vectors)

Αλγόριθμος αποσυμπίεσης 1.2

Ο προηγούμενος αλγόριθμος αναφέραμε και προηγουμένως ότι λόγω της πλήρους μετατροπής των δεδομένων στα αρχικά είναι πολύ χρονοβόρος όπως μπορούμε να δούμε και από τα αποτελέσματα στο Κεφάλαιο 6. Για τον λόγο αυτό δημιουργήσαμε και μια $2^{η}$ παραλλαγή του αλγορίθμου αυτού έτσι ώστε να κερδίσουμε χρόνο από την αποσυμπίεση. Τα βήματα και οι διαφορές από τον προηγούμενο αλγόριθμο αποσυμπίεσης φαίνονται πιο κάτω:

1. Κάθε SPE παραλαμβάνει τα baseline δεδομένα που προηγουμένως υπολογίσαμε πάνω στον PPE

2. Μετατρέπουμε (Cast) τα συμπιεσμένα δεδομένα σε floating point numbers και έτσι γλιτώνουμε τις πράξεις της πρόσθεσης που πραγματοποιούνται κατά την αποσυμπίεση του προηγούμενου αλγορίθμου. Η μετατροπή είναι γρηγορότερη από ότι η πρόσθεση και για το λόγο αυτό η αποσυμπίεση γίνεται πιο γρήγορα

3. Συνδυάζουμε τους 4 διψήφιους αριθμούς του *Extended Price* για να δημιουργήσουμε την αρχική τιμή

4. Αποθηκεύουμε τις αποσυμπιεσμένες τιμές σε διανύσματα (floating point vectors)

Υλοποιώντας τους αλγόριθμος συμπίεσης και αποσυμπίεσης κερδίσαμε χρόνο στην μεταφορά των δεδομένων, αφού καταφέραμε να μειώσουμε τις μεταφορές δεδομένων στις μισές. Αλλά δεν καταφέραμε να μειώσουμε τον χρόνο εκτέλεσης της εφαρμογής που ήταν και ο κυριότερος στόχος μας. Ο λόγος που δεν καταφέραμε αυτή την μείωση είναι γιατί η αποσυμπίεση των δεδομένων ακόμα και με τον $2^{ο}$ αλγόριθμο που είναι πιο γρήγορη είναι ακόμα αρκετά αργή. Ο χρόνος που κερδίζουμε από την μείωση των δεδομένων που μεταφέρουμε δεν είναι αρκετός έτσι ώστε να ξεπεράσει τον χρόνο που χρειαζόμαστε για την αποσυμπίεση, γεγονός που μας αυξάνει τον χρόνο εκτέλεσης.

Προσπαθώντας να ξεπεράσουμε τον χρόνο της αποσυμπίεσης μετατρέψαμε τον κώδικα μας έτσι ώστε να μπορεί να εκτελεί τις πράξεις πάνω σε συμπιεσμένα δεδομένα. Οι χρόνοι που πετύχαμε στο σενάριο αυτό ήταν αρκετά καλοί αλλά ακόμα όχι πιο μικροί από το

ταχύτερο σενάριο χωρίς την συμπίεση δεδομένων. Ο λόγος που συμβαίνει αυτό είναι γιατί στο σενάριο αυτό δεν μπορέσαμε να εφαρμόσουμε την τεχνική προγραμματισμού SIMD λόγο του τύπου των δεδομένων. Ένας άλλος λόγος που φαίνεται ότι η τεχνική της συμπίεσης των δεδομένων δεν κερδίζει χρόνο εκτέλεσης είναι η τεχνική του double-buffering που χρησιμοποιείται στον επεξεργαστή Cell/BE. Επειδή η μεταφορά των δεδομένων από την κύρια μνήμη στα Local Stores γίνεται από ένα δεύτερο μηχανισμό δεν επηρεάζει την εκτέλεση των SPE's και συνεπώς οι δύο αυτές ενέργειες (υπολογισμός και αποστολή δεδομένων) μπορούν να γίνουν παράλληλα. Έτσι αποκρύπτεται ο χρόνος μεταφοράς των δεδομένων από τον συνολικό χρόνο εκτέλεσης. Φαίνεται όμως ότι η τεχνική της συμπίεσης μπορεί να λειτουργήσει με θετικά αποτελέσματα εάν η τεχνική του double-buffering δεν χρησιμοποιηθεί. Περισσότερες πληροφορίες σχετικά με τα αποτελέσματα υπάρχουν στο Κεφάλαιο 6.

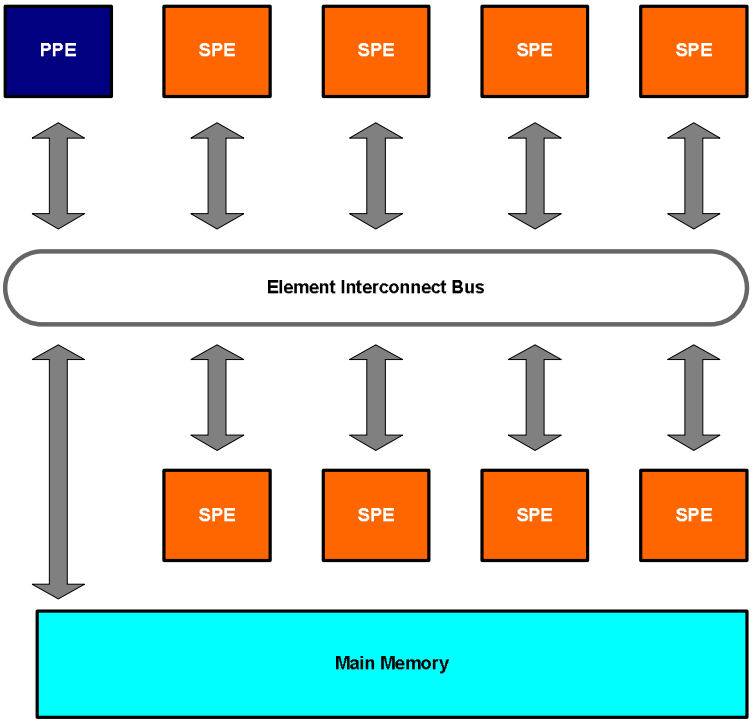
Θετικά αποτελέσματα με αυτή την τεχνική είχαμε όμως για την $3^{η}$ εφαρμογή που εμείς υλοποιήσαμε. Στην εφαρμογή αυτή συμπιέζαμε τα δεδομένα με τον AS1 αλλά μετά την μεταφορά δεν αποσυμπιέσαμε τα δεδομένα. Οι πράξεις που χρειάστηκε να εκτελεστούν πάνω σε αυτά τα δεδομένα μπορούσαν να εκτελεστούν και πάνω σε συμπιεσμένα δεδομένα. Έτσι καταφέραμε να αφαιρέσουμε τον χρόνο αποσυμπίεση και να έχουμε μια πολύ καλή αύξηση της επίδοσης εξαιτίας της τεχνικής της συμπίεσης δεδομένων.

Κεφάλαιο 4

Επεξεργαστής Cell Broadband Engine

Ο Cell Broadband Engine (Cell/BE) είναι ένας ετερογενής πολυπύρηνος επεξεργαστής σχεδιασμένος από μια τριμερή συμμαχία μεγάλων κολοσσών στην κατασκευή υπολογιστών. Οι εταιρίες Sony, Toshiba και IBM, γνωστές και από την συνεργασία τους ως 'STI' συνεργάστηκαν το 2000 και σχεδίασαν και κατασκεύασαν τον επεξεργαστή αυτό. Η πρώτη κυκλοφορία του Cell/BE έγινε τον Μάρτιο του 2007 και ακολούθως ενσωματώθηκε στον παιχνιδοκονσόλα PlayStation 3(PS3) της Sony.

Ο Cell συνδυάζει ένα γενικής χρήσης (general-purpose) Power Architecture πυρήνα με μέτρια επίδοση σε συνεργασία με 8 streaming πυρήνες οι οποίοι επιταχύνουν με επιτυχία εφαρμογές πολυμέσων και εφαρμογές που μπορούν να εκτελεστούν με την χρήση διανυσμάτων(vectors), όπως επίσης και πολλές άλλες εφαρμογές διαφορετικού τύπου.



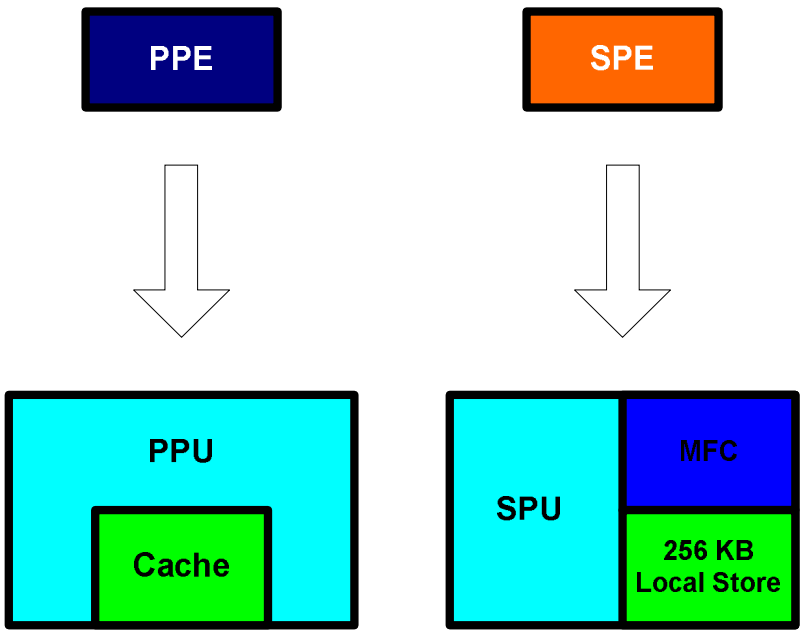
Σχήμα 4.1: *Αρχιτεκτονικός σχεδιασμός Cell/BE*

Ο κεντρικός πυρήνας ονομάζεται Power Processing Element(PPE) και βασίζεται στην 64-bit PowerPC αρχιτεκτονική επεξεργαστών με 2-way Simultaneous Multithreading(SMT). Δηλαδή έχει την ικανότητα εκτέλεσης δύο νημάτων(threads) ταυτόχρονα. Ο πυρήνας αυτός εκτελεί χρέη συντονιστή πάνω στο σύστημα του επεξεργαστή Cell/BE και κυρίως για τους υπόλοιπους 8 πυρήνες που λειτουργούν στο σύστημα και χειρίζονται τον περισσότερο φόρτο εργασίας. Ο PPE περιέχει 32 KB L1 μνήμη εντολών, 32 KB L1 μνήμη δεδομένων και 512 KB L2 κρυφή μνήμη. Το μέγεθος του block στην κρυφή μνήμη είναι 128 bytes και βγάζει επίδοση 6.4 GFLOPS στην συχνότητα των 3.2 GHz και μπορεί να φτάσει στην θεωρητική επίδοση των 25.6 GFLOPS την χρήση εντολών διανυσμάτων(vector).

Όπως αναφέρουμε και πιο πάνω ο PowerPC πυρήνας πάνω στο chip συνοδεύεται και από ακόμα 8 άλλα στοιχεία τα οποία ονομάζονται Synergistic Processor Elements(SPE's). Τα κάθε στοιχείο από αυτά αποτελείται από μια μονάδα υπολογισμού, την Synergistic Processor Unit(SPU) μια προσωπική μνήμη μεγέθους 256 KB που λόγω των διαφόρων

ιδιομορφιών που έχει –αναφέρονται στην συνέχεια - ονομάζεται Local Store και τέλος διαθέτει και μια μονάδα υπεύθυνη για την μεταφορά δεδομένων από και προς τον πυρήνα και ονομάζεται Memory Flow Controller(MFC) που μπορεί να λειτουργεί παράλληλα με την μονάδα επεξεργασίας SPU.

Ένας SPE είναι ένας επεξεργαστής RISC αρχιτεκτονικής με σχεδιασμό 128-bit SIMD για πράξεις μονής και διπλή υποδιαστολής(single and double precision instructions). Όπως αναφέρουμε και πιο πάνω, κάθε SPE διαθέτει 256 KB SRAM για εντολές και δεδομένα και ονομάζεται Local Store. Έχει ονομαστεί έτσι και όχι κρυφή μνήμη(cache memory) όπως σε άλλα CPUs για τον λόγο ότι δεν λειτουργεί όπως τις άλλες κρυφές μνήμες αφού δεν έχει υλική υλοποίηση που να την διαχειρίζεται ή που να προβλέπει ποια δεδομένα θα πρέπει να μπουν μέσα στον επόμενο κύκλο. Αντίθετα το Local Store(LS) ελέγχεται πλήρως από τον προγραμματιστή μέσω λογισμικού. Ο επεξεργαστής αυτός στην συχνότητα των 3.2 GHz μπορεί να βγάλει θεωρητική επίδοση των 25.6 GFLOPS.



Σχήμα 4.2: *Ανάλυση υλικού (α) PPE (β) SPE*

Το τελευταίο κομμάτι του Cell/BE που θα αναλύσουμε είναι το Element Interconnect Bus(EIB). Το EIB είναι ένα κανάλι επικοινωνίας μέσα στο chip το οποίο επιτρέπει την ανταλλαγή μηνυμάτων και την μεταφορά δεδομένων ανάμεσα στις διάφορες μονάδες που υπάρχουν στο σύστημα. Συνολικά οι μονάδες αυτές είναι 12 και αυτές είναι ο PPE, ο Memory Controller (MIC), οι οκτώ SPE's, και δύο μονάδες εισόδου/εξόδου(I/O) εκτός του chip. Λόγω του ότι υπάρχουν πολλές μονάδες που επικοινωνούν μέσω του EIB, σε αυτό περιέχεται και ένας μηχανισμός συντονισμού που λειτουργεί σαν 'φώτα τροχαίας' έτσι ώστε να συντονίζονται όλες οι μεταφορές δεδομένων και μηνυμάτων χωρίς να υπάρχουν οποιεσδήποτε συγκρούσεις. Όσων αφορά την δυνατότητα μεταφοράς δεδομένων (bandwidth) που προσφέρει το κανάλι υπάρχουν αρκετές εκδοχές με το καλύτερο να ξεπερνά τα 300 GB/s και το πραγματικό που αναφέρει και το IBM Systems Performance group να φτάνει τα 197 GB/s με τον επεξεργαστή να τρέχει στην συχνότητα των 3.2 GHz.

Ο σχεδιασμός και ο τρόπος κατασκευής του Cell/BE τον καθιστά ένα υπολογιστικό σύστημα με τεράστιες δυνατότητες. Οι SPE's έχουν κατασκευαστεί ώστε να μπορούν να διαχειρίζονται με μεγάλη απόδοση εφαρμογές τύπου μονής και διπλής υποδιαστολής (single and double precision). Αυτό τον καθιστά υπερδύναμη στην κατηγορία των εφαρμογών πολυμέσων. Για τον λόγο αυτό η Sony τον έχει ενσωματώσει μέσα στην τελευταία της παιχνιδοκονσόλα. Ο σχεδιασμός του μπορεί να ευνοεί εφαρμογές πολυμέσων αλλά δεν αφήνει περιορισμούς σε άλλου είδους εφαρμογές όπως εφαρμογές Βάσεων Δεδομένων ή επιστημονικές εφαρμογές. Αντίθετα πολλοί είναι οι οργανισμοί που χρησιμοποιούν τον Cell/BE για επίλυση μεγάλων και χρονοβόρων προβλημάτων λόγω της μεγάλης θεωρητικής επίδοσης που μπορεί να δώσει.

Όλα τα πιο πάνω φαντάζουν εξαιρετικά για ένα πολύ-επεξεργαστή στην κοινωνία της πληροφορικής αλλά οι επιδόσεις του Cell/BE έρχονται και με αρνητικά στοιχεία. Για να μπορέσει να προσφέρει αυτή την μεγάλη υπολογιστική ισχύ θυσιάζει κάποια από τα προνόμια κυρίως του προγραμματιστή που κατέχει σε διαφορετικούς επεξεργαστές και πολύ-επεξεργαστές. Όπως αναφέρεται και πιο πάνω το Local Store δεν έχει καθόλου μηχανισμούς για να προσφέρει συνέπεια στην μνήμη αλλά ούτε και να προβλέπει πια δεδομένα θα χρησιμοποιηθούν και θα πρέπει να φορτωθούν στην μνήμη. Συνεπώς αυτή η

διαδικασία αφήνεται στον προγραμματιστή. Δηλαδή ο ίδιος ο προγραμματιστής θα πρέπει να διαχειριστεί την μνήμη και τα δεδομένα που χρειάζεται το πρόγραμμα για εκτέλεση. Ο ίδιος ο προγραμματιστής είναι υπεύθυνος για να φορτώνει τα σωστά δεδομένα μέσα στα LS των SPE's και είναι επίσης υπεύθυνος να προσέχει το που γράφει στα LS's και από που διαβάζει από αυτά. Είναι επίσης υπεύθυνος να μοιράσει τα δεδομένα στα διάφορα SPE's. Αυτό προσθέτει μια μεγάλη δυσκολία και περισσότερο χρόνο στην διαδικασία συγγραφής των προγραμμάτων για τον Cell/BE.

Αυτό που εμείς κρατήσαμε από αυτό είναι το ότι όλες οι μεταφορές και η διαχείριση δεδομένων μεταξύ της κύρια μνήμης και των LS υλοποιούνται σε λογισμικό (software) και όχι σε υλικό (hardware) κάτι που κάνει την όλη διαδικασία πιο χρονοβόρα. Πάνω σε αυτό βασιστήκαμε εμείς για να υλοποιήσουμε την δουλειά αυτή, προσπαθώντας να κρύψουμε τον χρόνο αυτό και να κερδίσουμε από τον χρόνο εκτέλεσης και κατά συνέπεια να αυξήσουμε την επίδοση στην εκτέλεση των εφαρμογών μας.

Για την δική μας δουλεία χρησιμοποιήσαμε τον επεξεργαστή Cell/BE που παρέχεται πάνω στην παιχνιδοκονσόλα PlayStation 3 της Sony. Αυτό που είναι διαφορετικό σε αυτό το σύστημα σε σχέση με ένα κανονικό επεξεργαστή Cell/BE είναι ότι εδώ μπορείς να χρησιμοποιήσεις μόνο τα 6 από τα 8 SPE's που διαθέτει το σύστημα. Ο λόγος αυτού είναι ότι η ίδια η εταιρία κατασκευής έχει δεσμεύσει του 2 άλλους πυρήνες για λειτουργίες του συστήματος και κυρίως όπως αναφέρουν για διάφορους ελέγχους ασφαλείας του συστήματος[11]. Αυτό δεν εμπόδισε καθόλου την δική μας δουλεία απλά έπρεπε να λειτουργήσουμε με λιγότερους επεξεργαστές στη διάθεση μας.

Κεφάλαιο 5

Ανάλυση Εφαρμογών

5.1 Απαιτούμενα Χαρακτηριστικά Εφαρμογών

Για να εφαρμόσουμε την τεχνική της συμπίεσης δεδομένων που εξηγούμε στο Κεφάλαιο 3 χρειάστηκε να συγκεντρώσουμε διάφορα χαρακτηριστικά τα οποία έπρεπε να διαθέτουν οι εφαρμογές μας. Χαρακτηριστικά τα οποία θα ευνοούσαν την λειτουργία της τεχνικής της συμπίεσης έτσι ώστε να μπορέσουμε να δούμε διαφορά στην επίδοση. Αφού μιλάμε για συμπίεση δεδομένων και ο σκοπός μας είναι να μειώσουμε τον όγκο δεδομένων που μεταφέρονται και συνεπώς τον αριθμό μεταφορών που λαμβάνουν χώρο μέσα στο σύστημα τότε οι εφαρμογές μας πρέπει να καταναλώνουν μεγάλο όγκο δεδομένων. Με αυτό τον τρόπο θα έχουμε πολλές μεταφορές δεδομένων και συνεπώς θα καταναλώνεται αρκετός χρόνος από την εκτέλεση των εφαρμογών μας στην μεταφορά δεδομένων.

Εφαρμογές του τύπου αυτού υπάρχουν πολλές και εμείς επικεντρωθήκαμε σε δύο γνωστές εφαρμογές που έχουν ευρεία χρήση στην πληροφορική. Η κύρια εφαρμογή που χρησιμοποιήσαμε ήταν οι βάσεις δεδομένων και σαν δεύτερη εφαρμογή που δοκιμάσαμε

αλλά δεν μας ευνόησε – όπως περιγράφουμε και εξηγούμε στο Κεφάλαιο 3 – είναι ο Πολλαπλασιασμός Πινάκων (Matrix Multiplication).

5.2 Synthetic Application

Όπως θα δούμε και στο Κεφάλαιο 6 της Τελικής Αξιολόγησης όπου παραθέτουμε τα αποτελέσματα μας, οι προηγούμενες εφαρμογές δεν είχαν θετικά αποτελέσματα και η τεχνική της συμπίεσης δεδομένων δεν μας προσέφερε οποιαδήποτε αύξηση στην επίδοση. Για τον λόγο αυτό δημιουργήσαμε μια εφαρμογή που να μπορεί να εκμεταλλευτεί την τεχνική αυτή σε συνδυασμό με τον Αλγόριθμος συμπίεσης 1 – AS1. Για την επίτευξη αυτού πήραμε όλα τα απαραίτητα χαρακτηριστικά που πρέπει να έχει μια εφαρμογή για να μπορέσει να εκμεταλλευτεί την συμπίεση. Χαρακτηριστικά τα οποία εκμεταλλεύονται όχι μονό την συμπίεση δεδομένων αλλά και τα ειδικά χαρακτηριστικά του επεξεργαστή μας. Πιο κάτω αναφέρουμε αναλυτικά τα χαρακτηριστικά αυτά και τι χρησιμοποιήσαμε εμείς για να δημιουργήσουμε την εφαρμογή μας.

Ο μεγάλος όγκος δεδομένων είναι αναγκαίος έτσι ώστε να πετύχουμε καλή επίδοση με την συμπίεσης δεδομένων, έτσι πήραμε σαν είσοδο στην εφαρμογή μας τα δεδομένα εισόδου της Βάσης Δεδομένων που χρησιμοποιήσαμε και προηγουμένως. Αυτό που αλλάξαμε στα δεδομένα αυτά είναι τα στοιχεία που θα χρησιμοποιεί η εφαρμογή μας από τα records της Βάσης. Η αρχική εφαρμογή χρησιμοποιούσε 4 στοιχεία από το κάθε record έτσι στέλναμε 4 στοιχεία από το κάθε record. Η φύση των στοιχείων αυτών όμως περιόρισαν την ποσοστό (ratio) συμπίεσης που μπορούσαμε να πετύχουμε σε 2. Στην εφαρμογή αυτή χρησιμοποιήσαμε μόνο ένα στοιχείο από το κάθε record και καταφέραμε να αυξήσουμε το ποσοστό (ratio) συμπίεσης σε 4. Αυτό που μας αύξανε το ποσοστό συμπίεσης ήταν ότι λόγο της φύσης των νέων δεδομένων δεν χρειαζόμασταν να εκτελέσουμε το $1^{ο}$ στάδιο του αλγορίθμου μας (AS1, Κεφάλαιο 3) και εκτελούσαμε μόνο το $2^{ο}$, όπου εκεί το ποσοστό συμπίεσης μπορούσε να φτάσει μέχρι και το 4.

Το 2^{ο} χαρακτηριστικό που έπρεπε να έχει η εφαρμογή μας προέρχεται από πολλές παρατηρήσεις που κάναμε κατά την διεκπεραίωση της εργασίας αυτής. Παρατηρήσαμε ότι η εκτέλεση οποιασδήποτε εφαρμογής πάνω στα SPE's ήταν πάρα πολύ αργή εάν δεν χρησιμοποιούσαμε SIMD εντολές επεξεργασίας. Χρησιμοποιώντας όμως το προγραμματιστικό αυτό μοντέλο δεν μπορούσαμε να υλοποιήσουμε την εφαρμογή μας χωρίς την χρήση συνάρτησης αποσυμπίεσης δεδομένων. Έτσι προσπαθώντας να αποφύγουμε και αυτό το εμπόδιο έπρεπε να χρησιμοποιήσουμε απλές πράξεις εκτέλεσης στα SPE's για να μπορέσουμε να αποφύγουμε την χρήση της συνάρτησης αποσυμπίεσης και να μην δημιουργούμε επιπλέον καθυστερήσεις στην εκτέλεση.

Συνοπτικά στην σενάριο αυτό χρησιμοποιήσαμε τον AS1 για συμπίεση και την επεξεργασία πάνω σε συμπιεσμένα δεδομένα. Χρησιμοποιήσαμε επίσης και την τεχνική του Double Buffering αλλά δεν υλοποιήσαμε την εφαρμογή μας σε SIMD μοντέλο γιατί με το μοντέλο αυτό θα έπρεπε να χρησιμοποιήσουμε συνάρτηση που να μετατρέπει τα δεδομένα μας σε διανύσματα, κάτι που αύξανε τον χρόνο επεξεργασίας και τον συνολικό χρόνο εκτέλεσης. Όσον αφορά το baseline σενάριο για την εφαρμογή αυτή χρησιμοποιήσαμε και τις δύο τεχνικές, δηλαδή και το Double Buffering και το SIMD μοντέλο για να έχουμε το γρηγορότερο δυνατό σενάριο.

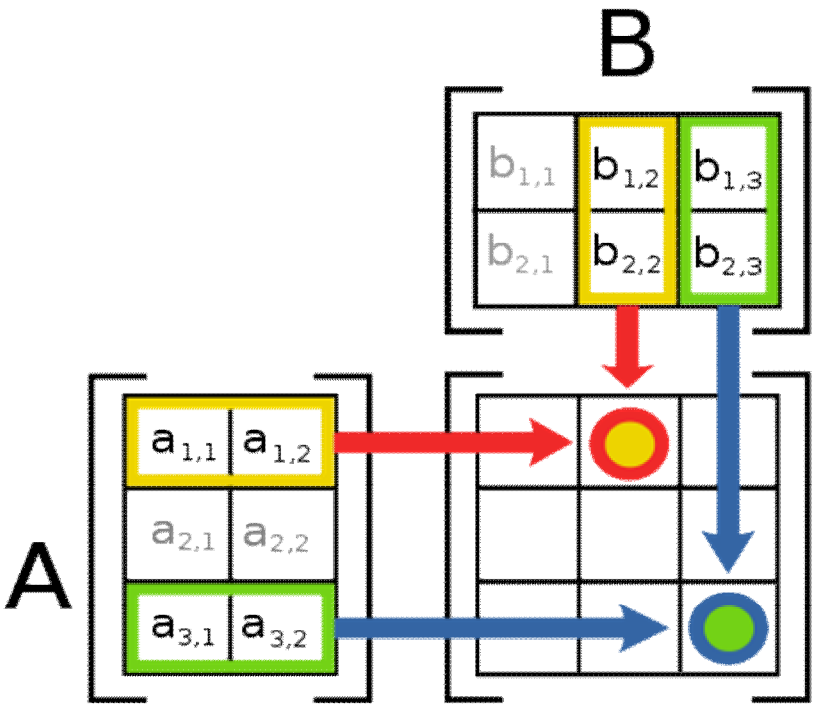
Συνδυάζοντας τα χαρακτηριστικά αυτά δημιουργήσαμε την Συνθετική εφαρμογή η οποία στέλνει όλα τα records πάνω στα SPE's και τα αριθμά έτσι ώστε στο τέλος να πάρουμε από κάθε SPE τον αριθμό των συνολικών records που υπάρχουν μέσα στην Βάση Δεδομένων μας. Δημιουργήσαμε μια πάρα πολύ απλή εφαρμογή η οποία όμως αποδεικνύει ότι η τεχνική της συμπίεσης δεδομένων είναι χρήσιμη και μπορεί να προσφέρει αύξηση στην επίδοση κάποιων εφαρμογών πάνω στον Cell/Be ή και άλλων επεξεργαστών ή συστημάτων παρόμοιου τύπου και αρχιτεκτονικής.

5.3 Πολλαπλασιασμός Πινάκων

Μια δεύτερη εφαρμογή που δοκιμάσαμε στην εργασία αυτή είναι οι Πολλαπλασιασμός Πινάκων, ένα πολύ γνωστό και χρονοβόρο πρόβλημα στην Πληροφορική. Ένα πρόβλημα που χρησιμοποιείται ίσως περισσότερο από κάθε άλλο για την αξιολόγηση νέων συστημάτων. Ο λόγος είναι οι μεγάλες απαιτήσεις σε μνήμη που χρειάζεται η εφαρμογή όπως επίσης και η υπολογιστική ισχύ που χρειάζεται κυρίως για μεγάλο όγκο δεδομένων. Έχοντας τον Cell/BE στην διάθεση μας, μια μηχανή με μεγάλη θεωρητική υπολογιστική ισχύ και το ότι εξετάζουμε συμπίεση δεδομένων χρειαζόμαστε μεγάλο όγκο δεδομένων, κάτι που μπορεί να μας το προσφέρει η εφαρμογή αυτή.

Έχοντας τον μεγάλο αυτό όγκο δεδομένων μπορούμε τότε να εφαρμόσουμε την τεχνική συμπίεσης έτσι ώστε να μπορέσουμε να μειώσουμε αυτό τον όγκο δεδομένων με σκοπό να καταφέρουμε να μειώσουμε και τον αριθμό των μεταφορών των δεδομένων από την κεντρική μνήμη στα SPE’s.

Η λειτουργία της εφαρμογής αυτής είναι αρκετά απλή. Όπως φαίνεται και από το Σχήμα 5.1 πιο κάτω έχουμε σαν δεδομένα δύο πίνακες Α και Β. Για να μπορεί να λειτουργήσει το πρόβλημα αυτό πρέπει οι γραμμές του Α πίνακα να είναι ίσες με τις στήλες του Β πίνακα. Σαν έξοδο/αποτέλεσμα έχουμε ένα τρίτο πίνακα ΑΒ ο οποίος έχει τόσες γραμμές όσες ο πίνακας Α και τόσες στήλες όσες ο πίνακας Β. Για να υπολογίσουμε κάθε στοιχείο του πίνακα ΑΒ χρειαζόμαστε μια ολόκληρη γραμμή από τον πίνακα Α και μια ολόκληρη στήλη από τον πίνακα Β. Συνεπώς για να υπολογίσουμε το πρώτο στοιχείο της πρώτης στήλης και πρώτης γραμμής του πίνακα ΑΒ χρειαζόμαστε την πρώτη γραμμή του πίνακα Α και την πρώτη στήλη του πίνακα Β, κ.ο.κ.



Σχήμα 5.1: *Πολλαπλασιασμός Πινάκων*

Μαθηματικά μπορούμε να αναπαραστήσουμε το πρόβλημα αυτό σαν ένα άθροισμα όπως φαίνεται πιο κάτω στο Σχήμα 5.2, που αναπαριστά μαθηματικά αυτό που έχουμε εξηγήσει πιο πάνω.

$$(AB)_{i,j} = \sum_{r=1}^{n} A_{i,r} B_{r,j}$$

Σχήμα 5.2: *Μαθηματική αναπαράσταση του Πολλαπλασιασμού Πινάκων*

Για την δική μας δουλεία χρησιμοποιήσαμε έτοιμη εφαρμογή[30] υλοποιημένη για τον Cell/BE για γρήγορο πολλαπλασιασμό πινάκων. Η εφαρμογή έχει υλοποιηθεί με σκοπό να πετυχαίνει την μεγαλύτερη επίδοση και για τον σκοπό αυτό οι κατασκευαστές της υλοποιήσαν τον κώδικα για τα SPE's σε κώδικα assembly για SIMD. Οι πίνακες που δημιουργεί η εφαρμογή είναι μεγέθους πάντα πολλαπλάσιου του 128 κάτι που διευκολύνει την μεταφορά των δεδομένων από την κεντρική μνήμη στα Local Stores των SPE's αφού για να γίνει η μεταφορά πρέπει τα δεδομένα να είναι πολλαπλάσια του 128. Σημαντικό επίσης να αναφέρουμε είναι ότι ο τύπος δεδομένων που χρησιμοποιήθηκε για τα δεδομένα εισόδου ήταν μονής κινητής υποδιαστολής (floating point numbers) και οι αριθμοί αυτοί επιλέγονταν τυχαία για την αρχικοποίηση των πινάκων.

Παρόλα αυτά έχουμε εγκαταλείψει την εφαρμογή αυτή και δεν παραθέτουμε οποιαδήποτε αποτελέσματα λόγω του ότι η συμπίεση (compression) των δεδομένων ήταν σε πολύ χαμηλό βαθμό και δεν μπορούσαμε να εκμεταλλευτούμε κάτι έτσι ώστε να κερδίσουμε χρόνο στην μεταφορά των δεδομένων.

5.4 Βάσεις Δεδομένων

Οι Βάσεις Δεδομένων είναι γενικά πολύ γνωστές εφαρμογές και βρίσκονται σχεδόν παντού, και το σημαντικότερο για την δική μας έρευνα είναι η μεγάλη απαίτηση τους σε μνήμη. Συνεπώς σε ένα κατανεμημένο σύστημα (distributed memory system) όπως μπορεί να χαρακτηριστεί και ο Cell/BE βάση της αρχιτεκτονικής του θα καταναλώνεται μεγάλος χρόνος στην μεταφορά των δεδομένων.

Για την δική μας μελέτη χρησιμοποιήσαμε ένα Query, το Q6, από το TPC-H Benchmark suite [36]. Το Query αυτό ανήκει σε μια μεγάλη οικογένεια δοκιμαστικών query βάσεων δεδομένων τα οποία χρησιμοποιούνται από διάφορες εταιρίες και οργανισμούς για την αξιολόγηση υπολογιστικών συστημάτων. Για τον λόγο αυτό αποφασίσαμε και εμείς να το χρησιμοποιήσουμε αφού είναι ένα έγκυρο και αξιόπιστο μέσο για να μπορέσουμε να πάρουμε σωστά και αξιόπιστα αποτελέσματα.

Πιο κάτω παραθέτουμε το αρχικό (Original) query σε SQL και βάση αυτού εμείς κάναμε την υλοποίηση σε C κώδικα για να μπορέσουμε να το τρέξουμε πάνω στον Cell/BE.

```
select
      sum(l_extendedprice * l_discount) as revenue
from
      lineitem
where
      l_shipdate >= date '[DATE]'
      and l_shipdate < date '[DATE]' + interval '1' year
      and l_discount between [DISCOUNT] - 0.01 and [DISCOUNT] + 0.01
      and l_quantity < [QUANTITY]
```

Σχήμα 5.3: *TPC-H Query 6 σε SQL*

Η βάση δεδομένων με τα στοιχεία (records) που χρειάζεται το Query για να εκτελεστεί είναι αποθηκευμένη σε αρχεία κειμένου και σε αναπαράσταση χαρακτήρων. Κάθε record αποτελείται από 9 συνολικά στοιχεία και από αυτά εμείς χρειαζόμαστε τα 4. Χρειαζόμαστε το extended price, το discount, την χρονολογία (year) και την ποσότητα (quantity). Όλα τα στοιχεία της βάσης μας τα οποία το query χρησιμοποιά είναι αριθμοί τύπου κινητής υποδιαστολής (floating point numbers), κάτι που μας βοήθησε να δημιουργήσουμε τον δικό μας αλγόριθμο συμπίεσης δεδομένων.

Μετατρέψαμε το selection statement που φαίνεται πιο πάνω σε C κώδικα και αντικαταστήσαμε την συνθήκη του SQL query με μια συνθήκη επιλογής στην C (if statement). Στο σειριακό κομμάτι διατρέχαμε ολόκληρη την βάση δεδομένων, στοιχείο προς στοιχείο, ελέγχαμε εάν ισχύει η συνθήκη και ακολούθως εκτελούσαμε τον πολλαπλασιασμό που υπάρχει στο query. Στο παράλληλη παραλλαγή του query μοιράζουμε τα δεδομένα σε 6 κομμάτια (όσοι και οι επεξεργαστές μας) και ο κάθε ένας εκτελεί το query πάνω στα δικά του δεδομένα. Το μέγεθος δεδομένων που θα πάρει ο κάθε επεξεργαστής είναι σταθερό και ανάλογο του μεγέθους της Βάσης Δεδομένων που

χρησιμοποιούμε. Για την δική μας δουλεία χρησιμοποιούμε 7 διαφορετικά μεγέθη αρχείων ως είσοδο στο πρόγραμμα μας.

5.5 Σύνοψη Εφαρμογών

Συνοψίζοντας τις πιο πάνω εφαρμογές που έχουμε περιγράψει παραθέτουμε ένα πίνακα με τα διάφορα χαρακτηριστικά που έχουν οι εφαρμογές μας. Για το μέγεθος δεδομένων έχουμε 3 διαφορετικά μεγέθη (low, medium, large) όπου low σημαίνει ότι το μέγεθος του αρχείου είναι πολύ μικρό και large είναι αρκετά μεγάλο. Ενώ για τον αριθμό των πράξεων τα τρία επίπεδα (low, medium, high) συμβολίζουν τον αριθμό των πράξεων.

Πίνακας 5.1: *Χαρακτηριστικά εφαρμογών*

	Μέγεθος Δεδομένων			Αρ. Πράξεων		
	Low	Medium	Large	Low	Medium	High
Βάσεις Δεδομένων	✓	✓	✓		✓	
Πολλ/σμός Πινάκων		✓	✓			✓
Synthetic	✓	✓	✓	✓		

Κεφάλαιο 6

Πειραματική Αξιολόγηση

6.1 Διαδικασία αξιολόγησης

Στην εργασία αυτή μιλούσαμε για επίδοση των εφαρμογών και προσπαθήσαμε να αυξήσουμε την επίδοση των εφαρμογών. Για να προσδιορίσουμε την επίδοση μετρούσαμε χρόνο εκτέλεσης της εκάστοτε εφαρμογής. Πιο συγκεκριμένα μετρούσαμε τον χρόνο εκτέλεσης του query, δηλαδή από την στιγμή που ξεκινά η εκτέλεση πάνω στα δεδομένα μας, είτε αυτά ήταν σε κανονική μορφή είτε ήταν συμπιεσμένα. Οι συναρτήσεις που χρησιμοποιήσαμε για να μετρήσουμε τον χρόνο είναι συναρτήσεις από την βιβλιοθήκη time.h και η μέτρηση γινόταν στο αρχείο του PPU. Δηλαδή μετρούσαμε τον χρόνο από την στιγμή που ο PPU έδινε την εντολή για να αρχίσουν την εκτέλεση τους τα SPU's και τερματίζαμε την μέτρηση αυτή την στιγμή που όλα τα SPU's έστελναν πίσω το αποτέλεσμα τους. Αφήσαμε εκτός μέτρησης το χρόνο που χρειαζόταν η εφαρμογή να

διαβάσει τα δεδομένα από τον δίσκο και στα σενάρια τα οποία είχαμε συμπίεση δεδομένων αφήναμε εκτός και την συμπίεση των δεδομένων.
Ο λόγος που επιλέξαμε να αφήσουμε τα κομμάτια αυτά εκτός του χρόνου εκτέλεσης μας ήταν γιατί αυτό που θέλαμε να δούμε ήταν μόνο ο χρόνος επεξεργασίας και μεταφοράς των δεδομένων. Και στις περιπτώσεις που είχαμε την συμπίεση των δεδομένων θέλαμε να δούμε κατά πόσον με την χρήση της τεχνικής αυτής μειώνεται είτε ο χρόνος επεξεργασίας, είτε ο χρόνος μεταφοράς των δεδομένων μας.

Όπως αναφέρουμε και στο κεφάλαιο των εφαρμογών (Κεφάλαιο 5) αποτελέσματα παρουσιάζουμε μόνο για τις Βάσεις Δεδομένων και για το Synthetic Application. Για τον Πολλαπλασιασμό Πινάκων δεν έχουμε αποτελέσματα γιατί κατά την εφαρμογή της συμπίεσης των δεδομένων πάνω στην συγκεκριμένη εφαρμογή η αναλογία συμπίεσης που βγάζαμε ήταν πάρα πολύ χαμηλή με τον Zlib Αλγόριθμο και αποφασίσαμε να μην προχωρήσουμε περισσότερο. Το ποσοστό της συμπίεσης δεν ήταν αρκετό έτσι ώστε να μας επιτρέψει να έχουμε κέρδος στην μεταφορά των δεδομένων και συνεπώς δεν θα είχαμε κέρδος ούτε και στον χρόνο εκτέλεσης.

6.2 Πειραματική Διάταξη

Σαν κύρια εφαρμογή αξιολόγησης για την τεχνική της συμπίεσης έχουμε χρησιμοποιήσει το Q6 query από το TPC-H Benchmark suite. Πρώτο, πρέπει το Benchmark να έχει πρακτικό ενδιαφέρον και να χρησιμοποιείται για επίλυση πραγματικών προβλημάτων. Επομένως επιλέξαμε το query αυτό το οποίο είναι ευρέως αποδεκτό και χρησιμοποιείται σε επιστημονικές εφαρμογές. Δεύτερο πρέπει το Benchmark που θα χρησιμοποιήσουμε να είναι πολύ απαιτητικό σε μνήμη και να χρησιμοποιεί μεγάλο όγκο δεδομένων για την επεξεργασία του. Αυτό το χαρακτηριστικό ήταν σχεδόν επιτακτικό αφού η δικής μας τεχνική εφαρμόζεται πάνω σε δεδομένα που υπάρχουν στη μνήμη. Έτσι μια εφαρμογή με μεγάλο όγκο δεδομένων θα μας έδινε μεγάλο αριθμό μεταφορών δεδομένων πάνω στον Cell/BE και θα ήταν η καλύτερη επιλογή έτσι ώστε να μπορέσουμε να πάρουμε τους καλύτερους χρόνους εκτέλεσης για τα πειράματά μας.

Στην εφαρμογή αυτή χρησιμοποιήσαμε διάφορα μεγέθη δεδομένων εισόδου. Στον Πίνακα 6.1(α) φαίνονται τα μεγέθη αυτά των δεδομένων και στον Πίνακα 6.1(β) φαίνεται παράδειγμα ενός record από την βάση δεδομένων μας και ο τύπος των δεδομένων αυτών. Επίσης με μπλε χρώμα φαίνονται και τα στοιχεία του record τα οποία χρησιμοποιούνται μέσα στο Q6.

Πίνακας 6.1(α): *Μεγέθη δεδομένων εισόδου*

Scaling Factor	test	1	2	3	4	5	6	7
Records	14	60175	120515	299814	600572	1199969	2999671	6001215
Bytes	224	10^3	$1x10^6$	$4x10^6$	$9x10^6$	$2x10^7$	$5x10^7$	$9x10^7$

Πίνακας 6.1(β): *Παράδειγμα ενός record του Πίνακα Lineitem από την Βάση Δεδομένων*

Order Key	Quantity	Price	Discount	Ship Date	Commit Date	Receipt Date	Ship Mode
1	17	24710.35	0.04	1996-03-13	1996-02-12	1996-03-22	TRUCK

Πάρακάτω παραθέτουμε και το TPC-H Q6 query αυτούσιο σε SQL γλώσσα:

```
select
     sum(l_extendedprice*l_discount) as revenue
from
     lineitem
where
     l_shipdate >= date '[DATE]'
and          l_shipdate < date '[DATE]' + interval '1' year
and          l_discount between [DISCOUNT] - 0.01
and          [DISCOUNT] + 0.01
and          l_quantity < [QUANTITY]
```

Στον Πίνακα 6.1(γ) δείχνουμε όλα τα σενάρια που έχουμε υλοποιήσει και δείχνουμε ποιες από τις τεχνικές που έχουμε εξηγήσει χρησιμοποιά το κάθε σενάριο.

Πίνακας 6.1(γ): *Προγραμματιστικά χαρακτηριστικά για την κάθε εφαρμογή*

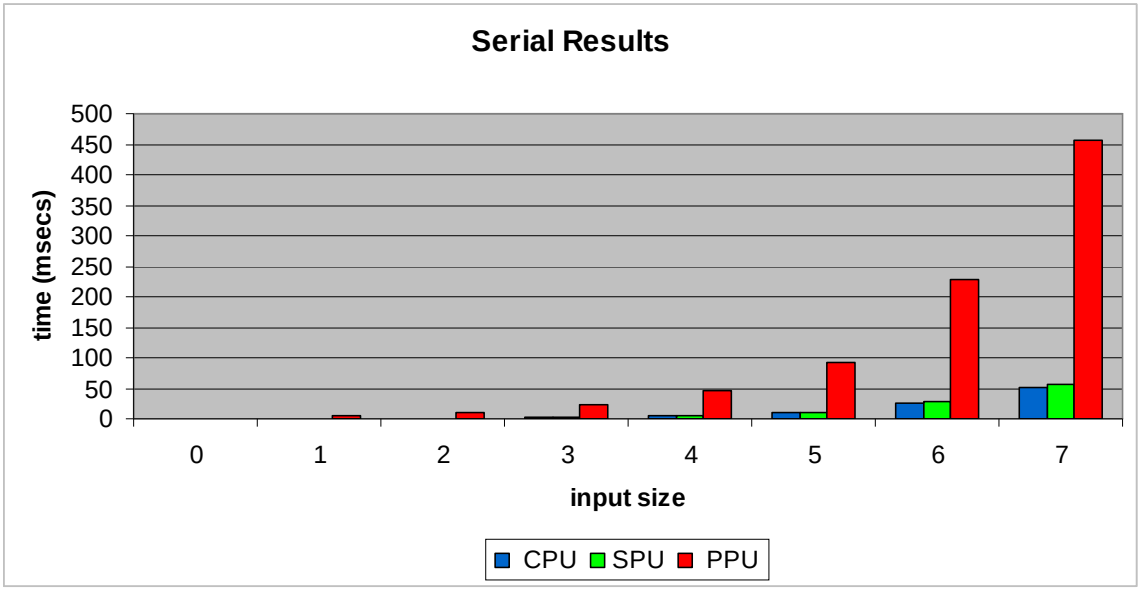
	SIMD	Double Buffering	Decompression	Decompression Function	
				Full	Casting
Baselines	✓	✓			
AS1 Decompression 1.1	✓	✓	✓	✓	
AS1 Decompression 1.2	✓	✓	✓		✓
AS1 No Decompression		✓			
Zlib	✓	✓	✓	✓	

Για την δική μας υλοποίηση και επειδή ο προγραμματισμός του Cell/BE γίνεται σε γλώσσα προγραμματισμού C/C++ μετατρέψαμε το SQL query αυτό σε γλώσσα C. Για την εκτέλεση των προγραμμάτων χρησιμοποιήσαμε πραγματικό σύστημα και τον επεξεργαστή Cell/BE που υπάρχει στην παιχνιδοκονσόλα της Sony, το Sony PlayStation. Περισσότερες λεπτομέρειες για την λειτουργία και την αρχιτεκτονική του συστήματος αυτού υπάρχουν στο Κεφάλαιο 4 της εργασίας αυτής. Τα σειριακά σενάρια τα έχουμε εκτέλεση πάνω σε προσωπικό υπολογιστή με επεξεργαστή Intel(R) Core(TM) 2 Duo CPU P8600 με συχνότητα λειτουργίας στα 2.40GHz, μνήμη 4GBytes. Τα υπόλοιπα σειριακά σενάρια, η εκτέλεση των οποίων αναφέρονται στον PPU έχουν εκτελεστή πάνω στον κεντρικό επεξεργαστή του Cell/BE, τον PPU(τα χαρακτηριστικά του περιγράφονται στο Κεφάλαιο 4).

6.3 Παρουσίαση και Ανάλυση Αποτελεσμάτων

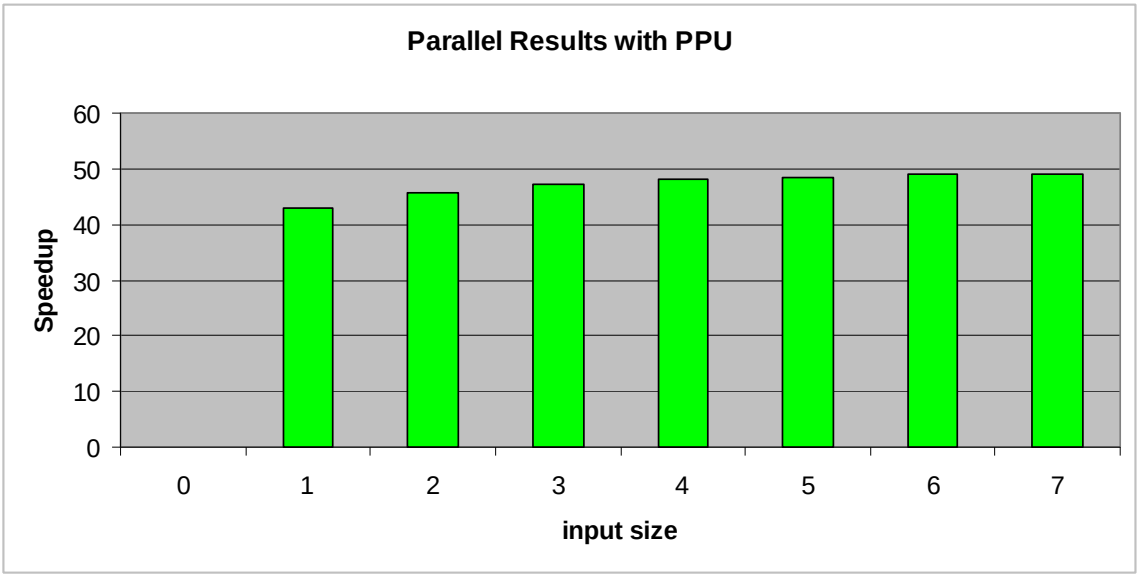
Αυτό που θέλαμε εμείς ήταν να κερδίσουμε επίδοση πάνω στην μηχανή του Cell/BE κάνοντας το query μας παράλληλο και χρησιμοποιώντας επίσης την τεχνική της συμπίεσης δεδομένων. Η παρουσίαση των αποτελεσμάτων όμως θα ξεκινήσει από την αρχή της ανάλυσης της δικής μας δουλείας που περιελάμβανε επίσης και την ανάλυση των σειριακών αποτελεσμάτων. Θα δείξουμε από την αρχή την έρευνα που έχουμε κάνει για να καταλήξουμε στο συμπέρασμα να χρησιμοποιήσουμε την τεχνική της συμπίεσης έτσι ώστε να κερδίσουμε επίδοση από τις εφαρμογές μας.

Στο Κεφάλαιο αυτό θα δείξουμε τα πιο σημαντικά αποτελέσματα που έχουμε βρει και αυτά τα οποία χρησιμοποιήσαμε σαν βάση για να προχωρήσουμε αλλά και για να βγάλουμε τα συμπεράσματα μας.

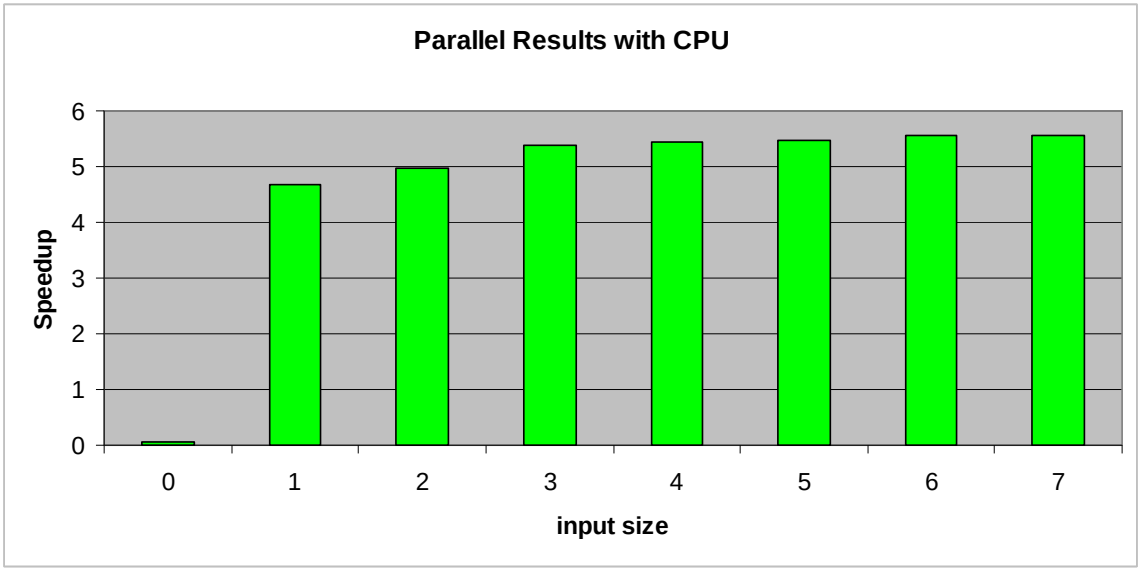


Σχήμα 6.1: *Αποτελέσματα πειραμάτων σειριακού υπολογισμού πάνω σε CPU, πάνω σε ένα SPU και πάνω στον PPU.*

Η εκτέλεση πάνω σε CPU είναι σε πολύ υψηλά επίπεδα επίδοσης, όπως επίσης και σε ένα SPU σε σχέση με τον PPU. Αυτό συμβαίνει για το λόγο ότι ο PPU είναι μια απλή κεντρική μονάδα επεξεργασίας χωρίς ιδιαίτερες δυνατότητες επεξεργασίας αφού στον Cell/BE χρησιμοποιείται όχι για τον υπολογισμό ενός προβλήματος αλλά περισσότερο για τον συντονισμό και τον διαμοιρασμό των δεδομένων στα SPE's. Εδώ όμως εμείς θέλουμε να δείξουμε απλώς ότι η εκτέλεση για τον σειριακό υπολογισμό είναι καλύτερη πάνω σε ένα CPU αλλά εμείς για να μπορέσουμε να βγάλουμε σωστά συμπεράσματα πρέπει να συγκρίνουμε τα παράλληλα σενάρια μας με τον σειριακό υπολογισμό στον PPU.



Σχήμα 6.2(α): *Αποτελέσματα παράλληλης εκτέλεσης Query Q6 χρησιμοποιώντας 6 SPE's και συγκρίνοντας με την σειριακή εκτέλεση πάνω στον PPU*



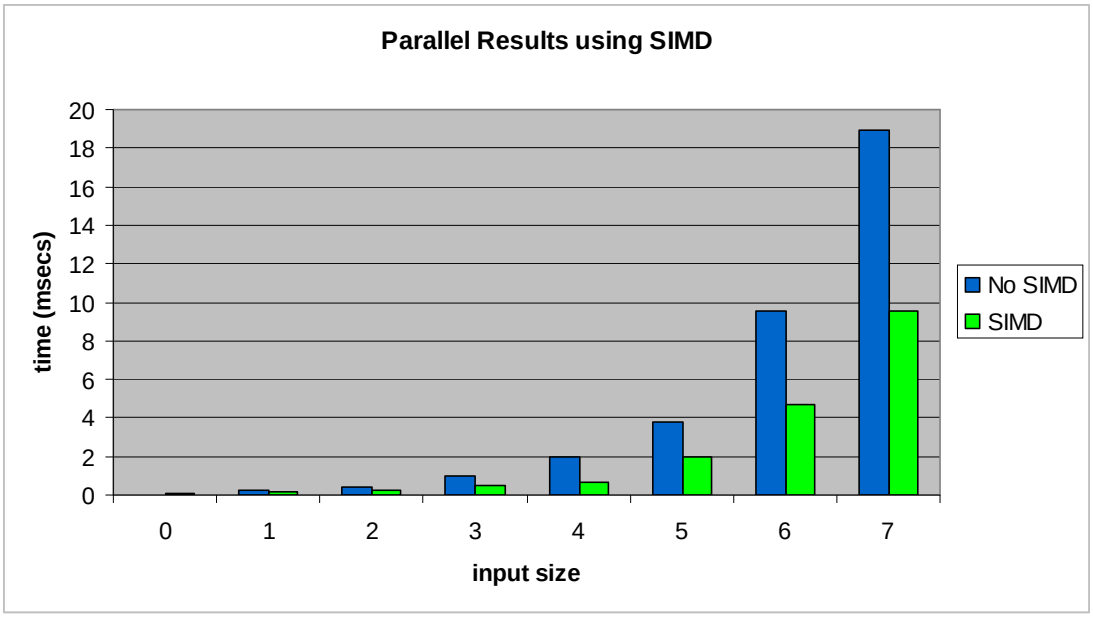
Σχήμα 6.2(β): *Αποτελέσματα παράλληλης εκτέλεσης Query Q6 χρησιμοποιώντας 6 SPE's και συγκρίνοντας με την σειριακή εκτέλεση πάνω στον CPU*

Η παράλληλη εκτέλεση του Q6 πάνω στον Cell/BE βλέπουμε να πετυχαίνει μια μεγάλη αύξηση της επίδοσης που ξεπερνά τα θεωρητικά επίπεδα και φτάνει στα επίπεδα του Super Linear Speedup(σχήμα 6.2(α)). Αυτό οφείλεται στην μικρή υπολογιστική ισχύ που έχει ο PPU αλλά και στην μεγάλη υπολογιστική ισχύ που έχουν τα SPE's. Αυτό αποδεικνύεται και από την $2^{η}$ γραφική παράσταση στο Σχήμα 6.2(β) όπου συγκρίνουμε και με την σειριακή εκτέλεση πάνω σε CPU. Στο σενάριο αυτό φαίνεται ότι τα SPE's φτάνουν απλώς κοντά στο θεωρητικό μέγιστο Speedup που είναι 6 αφού χρησιμοποιούμε 6 SPE's. Παρόλ'αυτά η σωστή σύγκριση που πρέπει να γίνει είναι με την σειριακή εκτέλεση πάνω στον PPU.

6.3.1 Baseline Scenario

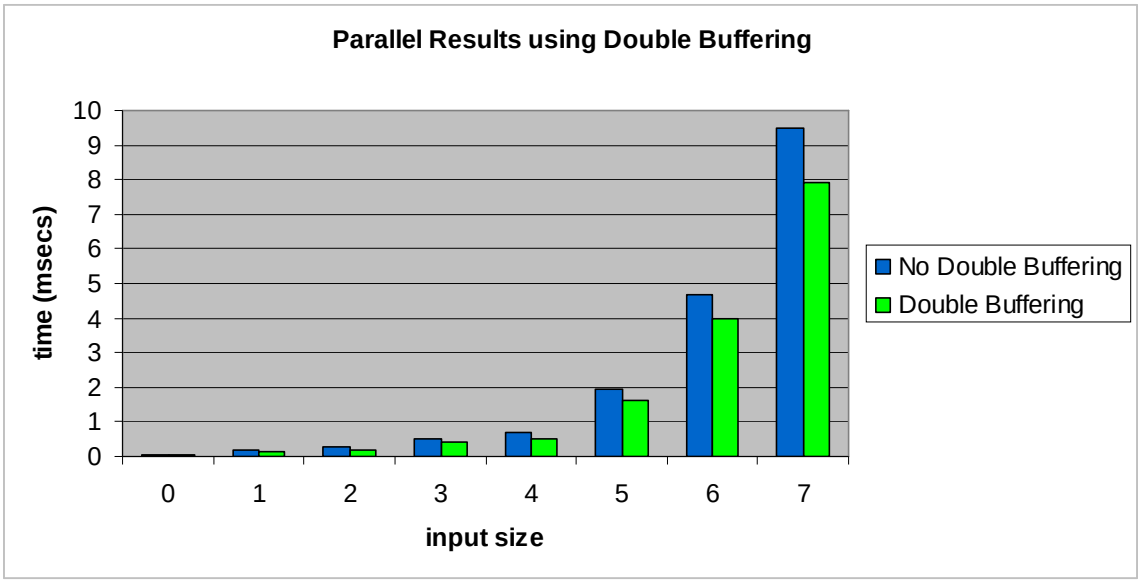
Την δική μας υλοποίηση με την συμπίεση των δεδομένων θα την αξιολογήσουμε χρησιμοποιώντας ένα παράλληλο σενάριο (baseline scenario), το οποίο θα πρέπει να είναι το ταχύτερο (μικρότερος χρόνος εκτέλεσης) παράλληλο σενάριο χωρίς συμπίεση

δεδομένων. Έτσι για να το δημιουργήσουμε αυτό χρησιμοποιήσαμε διάφορες τεχνικές προγραμματισμού του Cell/BE, οι οποίες μας έδιναν κάθε φορά και καλύτερους χρόνους.



Σχήμα 6.3: *Αποτελέσματα συνολικού χρόνου εκτέλεσης χρησιμοποιώντας την τεχνική του SIMD πάνω στα SPE's*

Η προγραμματιστική τεχνική του SIMD (Single Instruction Multiple Data) αποτελεί τεχνική βελτίωσης της επίδοσης οποιασδήποτε εφαρμογής που μπορεί να την εφαρμόσει. Με αυτή την τεχνική μπορεί να γίνει εκτέλεση της ίδιας εντολής σε περισσότερα από 1 δεδομένα σε κάθε κύκλο. Στην δική μας περίπτωση οι SPE's είναι SIMD επεξεργαστές και μπορέσαμε να εφαρμόσουμε την τεχνική αυτή πάνω στην δική μας εφαρμογή. Βλέπουμε ότι κερδίζουμε σε χρόνο εκτέλεσης περίπου 50% που σε συνδυασμό με άλλες τεχνικές που δείχνουμε πιο κάτω είναι αποφέρει μεγάλο κέρδος χρόνου εκτέλεσης. Συνεπώς εφαρμόσαμε την τεχνική αυτή και στα δικά μας σενάρια προτού εφαρμόσουμε και την τεχνική της συμπίεσης δεδομένων.



Σχήμα 6.4: *Αποτελέσματα συνολικού χρόνου εκτέλεσης χρησιμοποιώντας την τεχνική του double buffering πάνω στα SPE's*

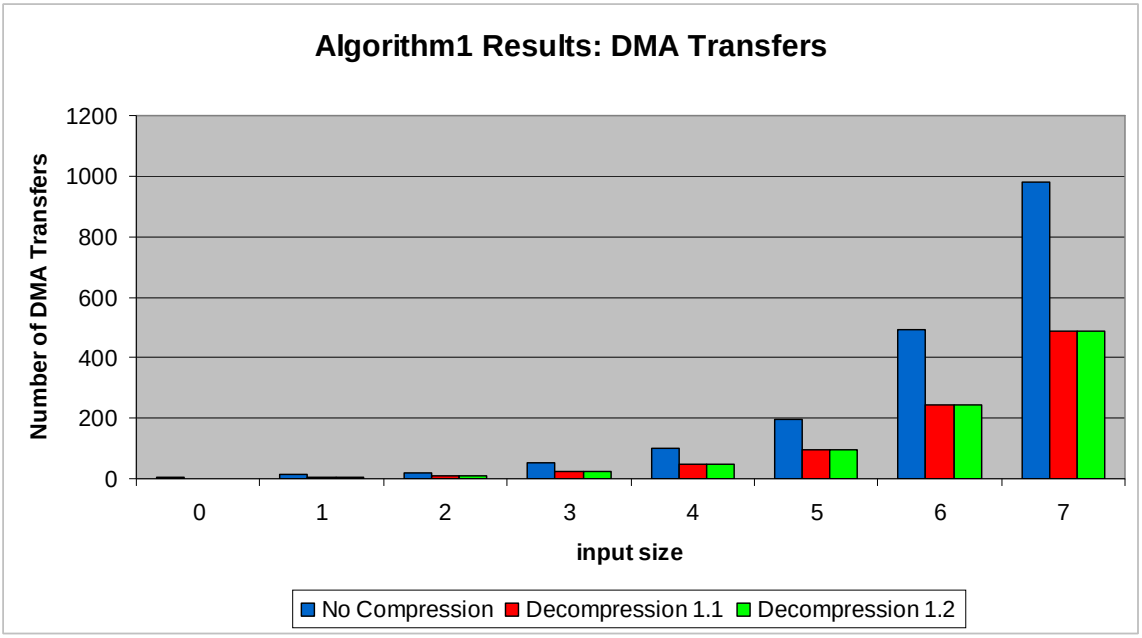
Μια δεύτερη τεχνική που χρησιμοποιήσαμε για να πετύχουμε καλύτερους χρόνους στο Baseline Scenario μας ήταν η τεχνική του Double Buffering που σε συνδυασμό με την τεχνική του SIMD που αναφέρουμε πιο πάνω κερδίσαμε ακόμα περισσότερο χρόνο, περίπου 17%.

Με αυτές τις τεχνικές ολοκληρώσαμε το Baseline σενάριο μας και καταφέραμε να μειώσουμε αρκετά τον χρόνο εκτέλεσης και κατά συνέπεια να αυξήσουμε την επίδοση του Baseline σεναρίου μας.

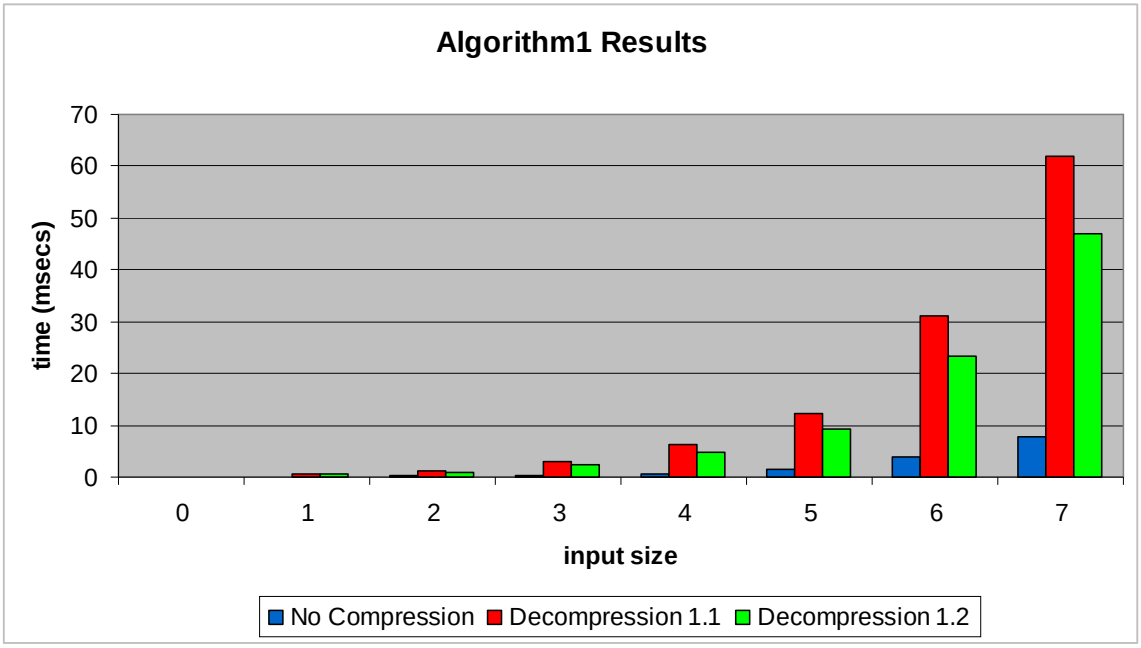
6.3.2 Αλγόριθμος Συμπίεσης 1 – AS1

Ο AS1 έχει αναλογία συμπίεσης 2, που σημαίνει ότι μετά το τέλος της συμπίεσης ο όγκος των δεδομένων μας μειώνεται στον μισό. Αυτό φαίνεται ότι συμβαίνει και με τον αριθμό των μεταφορών των δεδομένων. Η μείωση των μεταφορών των δεδομένων φτάνει στο 50% και αυτό το ποσοστό είναι σταθερό αφού και η αναλογία συμπίεσης του AS1 είναι σταθερή. Όπως βλέπουμε και στο σχήμα 6.5 ο αριθμός των μεταφορών των δεδομένων για

τους δύο αλγόριθμους αποσυμπίεσης είναι σταθερός γιατί ο αλγόριθμος συμπίεσης είναι ο ίδιος και στις δύο περιπτώσεις.



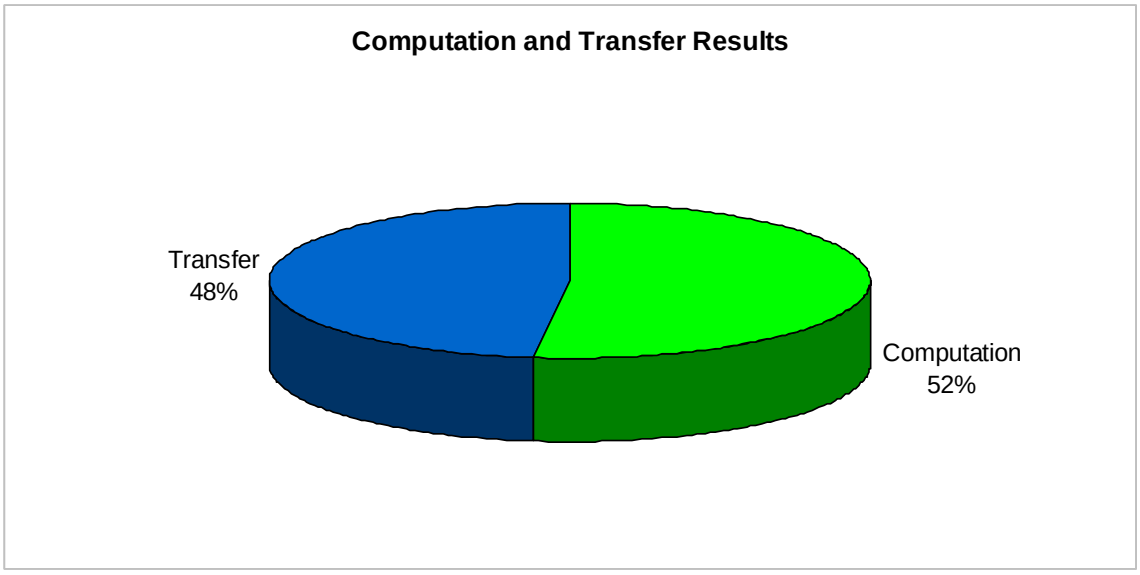
Σχήμα 6.5: *Αποτελέσματα μέτρησης μεταφοράς δεδομένων ανά SPE με τον AS1*



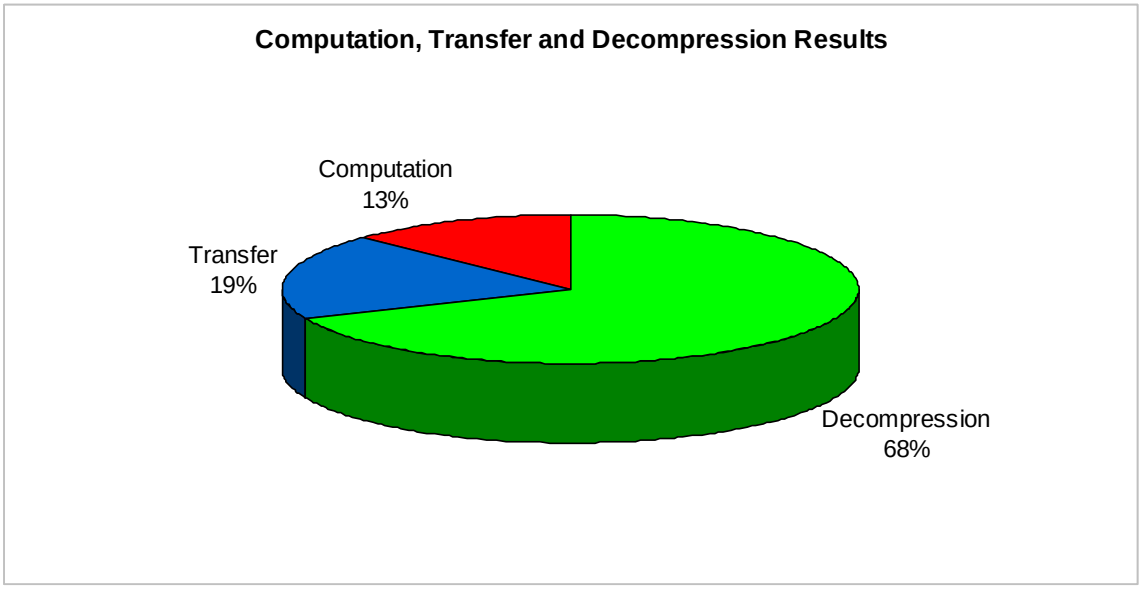
Σχήμα 6.6: *Αποτελέσματα συνολικού χρόνου εκτέλεσης χρησιμοποιώντας τον AS1 για συμπίεση και τους δύο αλγόριθμους αποσυμπίεσης*

Τα αποτελέσματα που είχαμε μετά την εφαρμογή του αλγορίθμου συμπίεσης δεν ήταν καθόλου ενθαρρυντικά, αφού οι χρόνοι εκτέλεσης και για τους δύο αλγόριθμους αποσυμπίεσης ήταν αρκετά μεγαλύτεροι από το baseline σενάριο μας. Στο Σχήμα 6.6 βλέπουμε την τεράστια διαφορά στον χρόνο εκτέλεσης των τριών αυτών σεναρίων. Αρχικά με τον Αλγόριθμο αποσυμπίεσης 1.1 βλέπουμε μια τεράστια αύξηση του χρόνο εκτέλεσης, αργότερα με τον Αλγόριθμο αποσυμπίεσης 1.2 βλέπουμε να μειώνεται ο χρόνος αυτός κατά περίπου 24% αλλά παραμένει αρκετά πιο ψηλός από τον χρόνο εκτέλεσης του baseline σεναρίου μας. Η αύξηση του χρόνου εκτέλεσης στο σενάριο με τον Αλγόριθμο αποσυμπίεσης 1.1 φτάνει στα 800%, ενώ η αύξηση στο σενάριο του Αλγορίθμου αποσυμπίεσης 1.2 φτάνει μέχρι και το 590%.

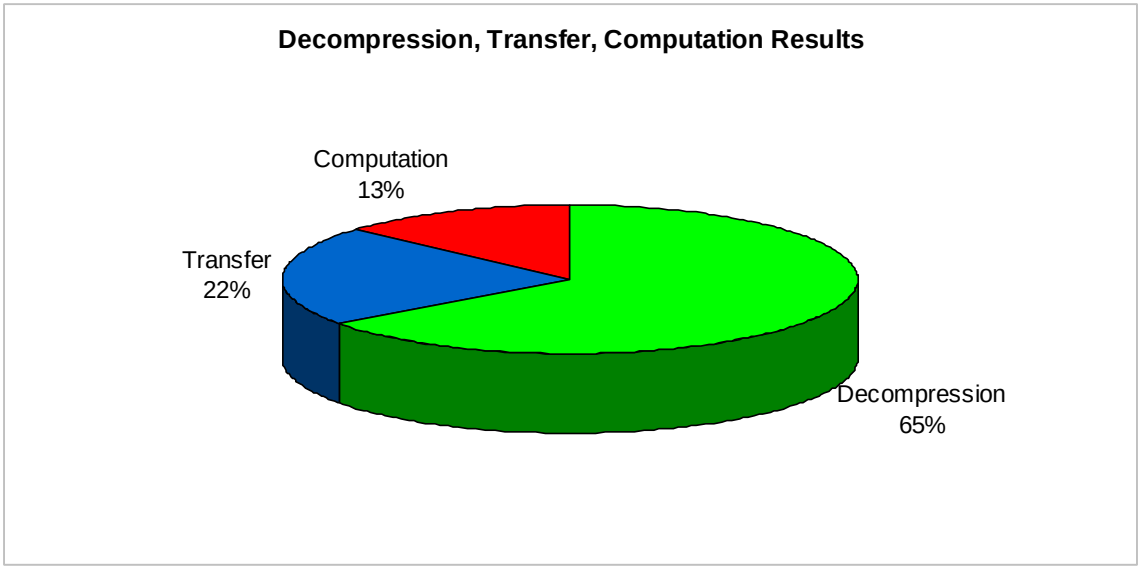
Παρόλο που οι μεταφορές των δεδομένων μειώνονται κατά 50% φαίνεται ότι όχι απλός δεν μειώνει τον χρόνο εκτέλεσης αλλά τον αυξάνει. Αυτό οφείλεται σε 3 παράγοντες. Ο πρώτος παράγοντας είναι το ότι ο χρόνος υπολογισμού πάνω σε ένα αριθμό δεδομένων (κυρίως 16KB) είναι περίπου ο ίδιος με τον χρόνο μεταφοράς των δεδομένων αυτών πάνω στα SPE's(Σχήμα 6.7(α)). Ο χρόνος μεταφοράς μειώνεται με την συμπίεση στο μισό αλλά ο χρόνος υπολογισμού δεν αλλάζει για τον λόγο ότι μετά την αποσυμπίεση, ο υπολογισμός γίνεται σε όλα τα δεδομένα που υπήρχαν στη Βάση Δεδομένων.



Σχήμα 6.7(α): *Ποσοστιαία αναλογία χρόνο υπολογισμού και χρόνου μεταφοράς δεδομένων για το baseline σενάριο*



Σχήμα 6.7(β): *Ποσοστιαία αναλογία χρόνου μεταφοράς, αποσυμπίεσης και υπολογισμού των δεδομένων για τον Αλγόριθμο Αποσυμπίεσης 1.1*



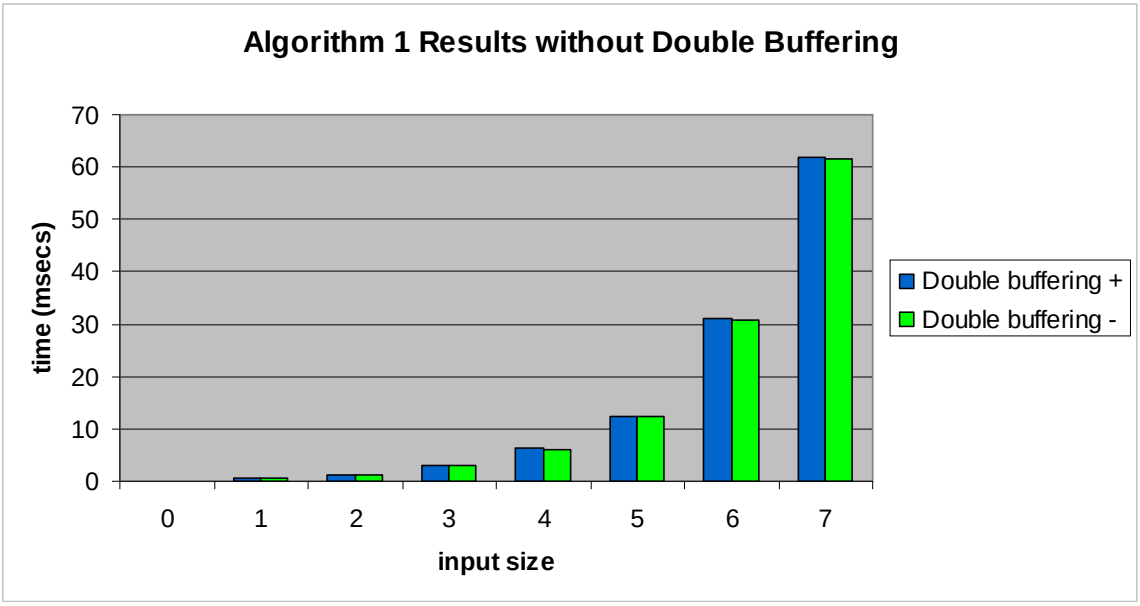
Σχήμα 6.7(γ): *Ποσοστιαία αναλογία χρόνου μεταφοράς, αποσυμπίεσης και υπολογισμού των δεδομένων για τον Αλγόριθμο Αποσυμπίεσης 1.2*

Ο δεύτερος παράγοντας που συνδυάζεται με τον πρώτο είναι η χρήση της τεχνικής του Double Buffering. Με την τεχνική αυτή και σε συνδυασμό με αυτό που αναφέρουμε πιο πάνω ($1^{ος}$ παράγοντας) αποκρύπτεται σχεδόν ολόκληρος ο χρόνος μεταφοράς των δεδομένων (εκτός βέβαια από την πρώτη μεταφορά δεδομένων). Αφού ο χρόνος μεταφοράς είναι περίπου ο ίδιος με τον χρόνο υπολογισμού είναι επόμενο με την χρήση του Double Buffering να αποκρύπτεται εντελώς αφού τα επόμενα δεδομένα μεταφέρονται στα LS's στη διάρκεια του υπολογισμού του αποτελέσματος στα προηγούμενα δεδομένα. Συνεπώς ο χρόνος που βλέπουμε στην εκτέλεση για το baseline (Σχήμα 6.6) είναι ο συνολικός χρόνος υπολογισμού του αποτελέσματος και ο χρόνος μεταφοράς της πρώτης σειράς δεδομένων.

Οι δύο πιο πάνω παράγοντες που περιγράψαμε επηρεάζουν την εκτέλεση αρνητικά και ουσιαστικά δεν αυξάνουν τον χρόνο εκτέλεσης αλλά τον εμποδίζουν από το να μειωθεί. Ο παράγοντας που λειτουργεί σαν επιπλέον υπολογισμός και αυξάνει τον χρόνο εκτέλεσης είναι ο χρόνος αποσυμπίεσης. Λόγω του ότι χρησιμοποιούμε συνάρτηση αποσυμπίεσης και στους δύο αλγόριθμους αποσυμπίεσης για να μπορούμε να εκτελέσουμε πράξεις πάνω στα

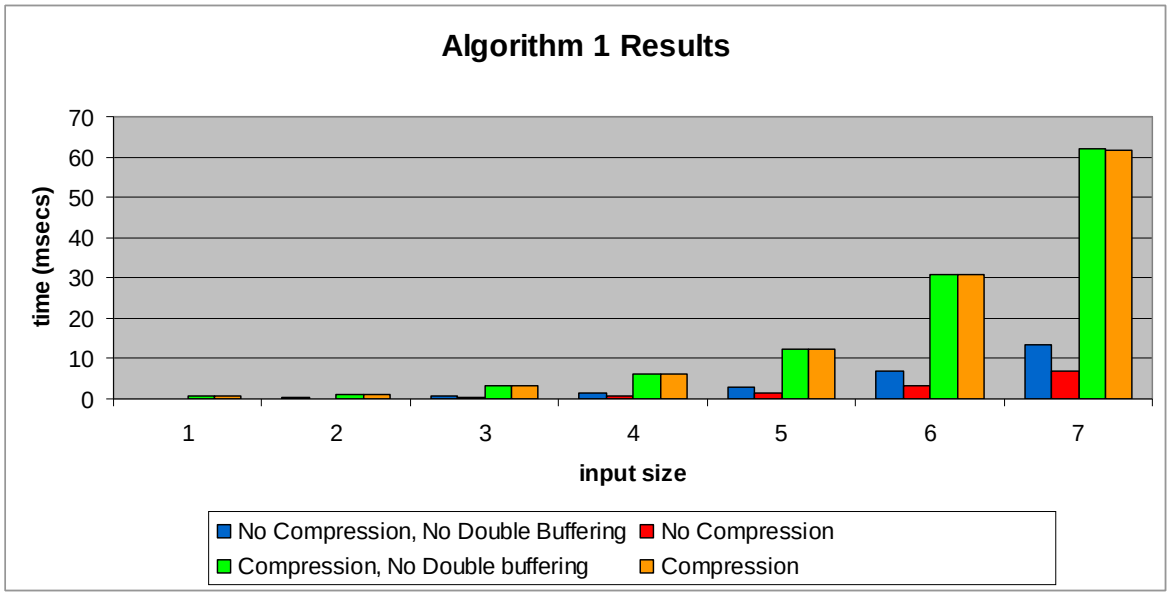
δεδομένα μας χρησιμοποιούμε και επιπλέον υπολογιστικό χρόνο που μας αυξάνει και τον συνολικό χρόνο εκτέλεσης. Αυτό μας δείχνει και το γράφημα στο Σχήμα 6.7(β) για τον Αλγόριθμο Αποσυμπίεσης 1.1 και στο Σχήμα 6.7(γ) για τον Αλγόριθμο Αποσυμπίεσης 1.2.

Όσον αφορά τον πρώτο παράγοντα δεν μπορούσαμε να κάνουμε κάτι έτσι ώστε να το αντιμετωπίσουμε γιατί ούτως η αλλιώς δεν μπορούμε να αλλάξουμε τον χρόνο μεταφοράς των δεδομένων αλλά ούτε και το χρόνο επεξεργασίας αφού η υλοποίηση μας όπως δείξαμε με τα πειράματα στο Υποκεφάλαιο 1.3.1 έχει τον χαμηλότερο χρόνο εκτέλεσης στο μοντέλο του Cell/BE. Για τον παράγοντα Double Buffering τρέξαμε σενάρια χωρίς την χρήση αυτού για να δούμε εάν έχουμε οποιαδήποτε αλλαγή στον χρόνο εκτέλεσης. Περιμέναμε να δούμε αύξηση του χρόνου εκτέλεσης όταν αφαιρούσαμε την τεχνική αυτή αφού εξ'αρχής την χρησιμοποιήσαμε για να μειώσουμε τον χρόνο εκτέλεσης αλλά στην περίπτωση με την συμπίεση των δεδομένων βλέπουμε ότι ο χρόνο δεν αυξάνεται αλλά παραμένει ασυμπτωτικά ο ίδιος (Σχήμα 6.8). Αφού τον επιπρόσθετο χρόνο στην εκτέλεση τον προσθέτει η αποσυμπίεση των δεδομένων, στο σενάριο μας χωρίς την τεχνική του Double Buffering φαίνεται ότι η τεχνική της συμπίεσης κερδίζει χρόνο γι'αυτό και ο χρόνος εκτέλεσης δεν αυξάνεται όταν αφαιρέσουμε το Double Buffering.

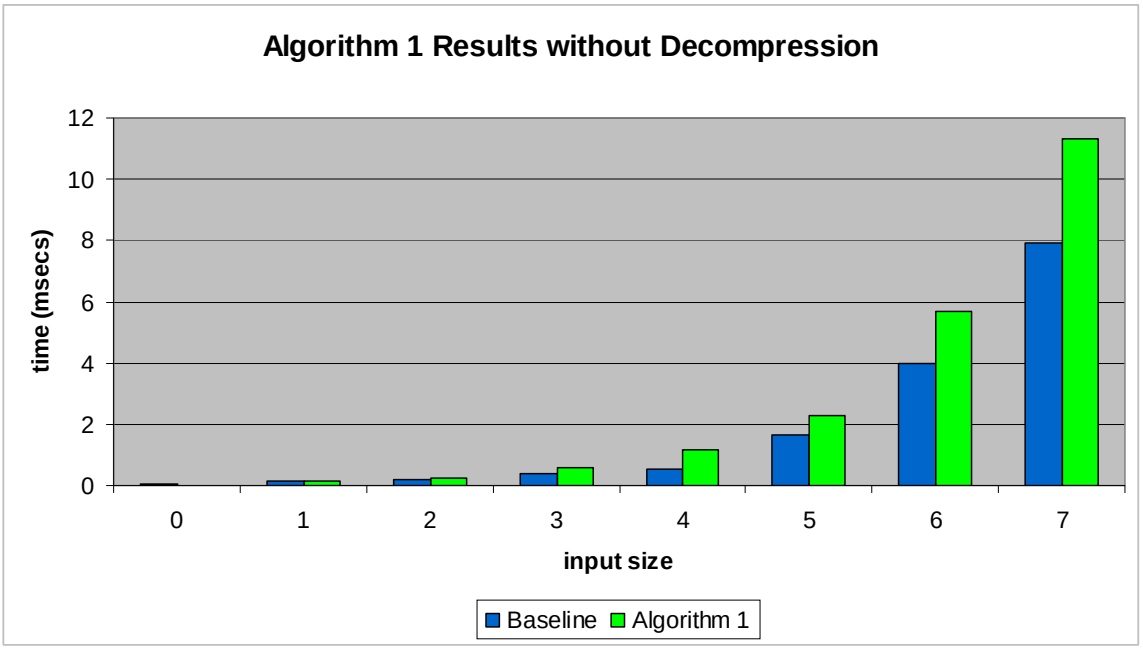


Σχήμα 6.8: *Αποτελέσματα χρόνου εκτέλεσης της εφαρμογής με τον AS1 συμπίεσης χωρίς την χρήση της τεχνικής του Double Buffering*

Αφού είδαμε ότι με τους δύο παράγοντες που δεν μας επιτρέπουν να κερδίσουμε χρόνο κατά την εκτέλεση επικεντρωθήκαμε στον τρίτο παράγοντα για να βρούμε τρόπο να μειώσουμε τον χρόνο αποσυμπίεσης και να καταφέρουμε να πάρουμε καλύτερους χρόνους. Από τα Σχήματα 6.7(β) και 6.7(γ) βλέπουμε ότι ο περισσότερος χρόνος κατά την εκτέλεση μας σπαταλάτε στην αποσυμπίεση. Έτσι αποφασίσαμε να μετατρέψουμε τον κώδικα μας με τέτοιο τρόπο ώστε να μπορεί να εκτελείται το Query πάνω σε συμπιεσμένα δεδομένα και να αποσυμπιέζουμε μόνο τα δεδομένα που χρειαζόταν να αποσυμπιεστούν έτσι ώστε να μπορεί να εκτελεστεί πλήρως το Query. Δηλαδή αποσυμπιέζουμε μόνο τα στοιχεία (records) τα οποία πληρούν την συνθήκη του selection statement στο Query και ακολούθως εκτελούμε πάνω τους τις πράξεις που χρειάζονται. Με αυτό τον τρόπο κερδίζουμε χρόνο από τον χρόνο αποσυμπίεσης των δεδομένων αφού αποσυμπιέζουμε μόνο ένα μικρό ποσοστό των δεδομένων που μεταφέρουμε.



Σχήμα 6.8(β): *Συνοπτικά αποτελέσματα σεναρίου Αλγορίθμου Συμπίεσης 1 με και χωρίς την χρήση του Double Buffering*



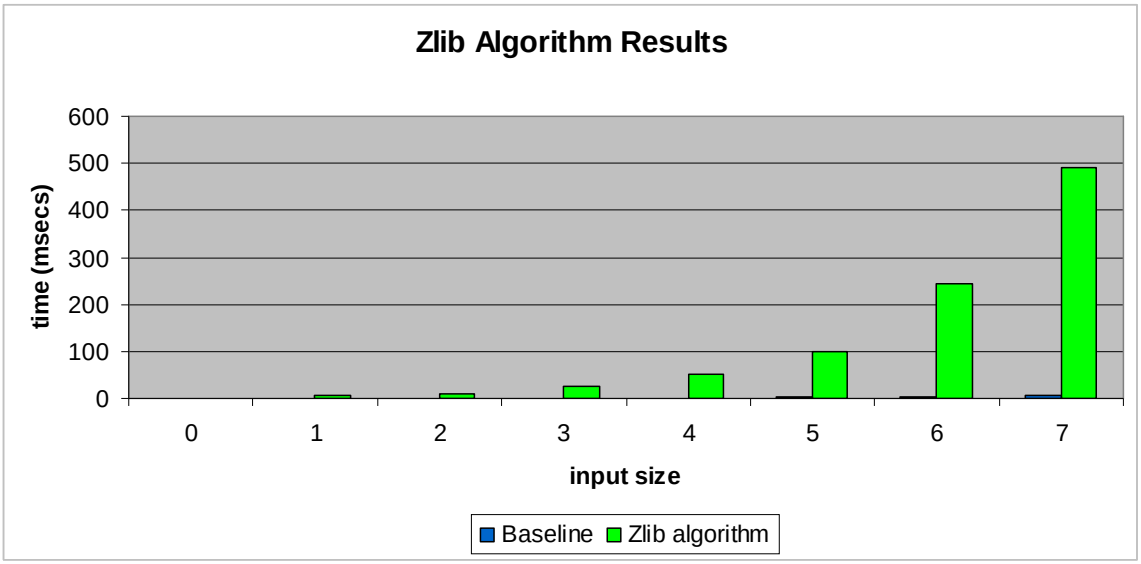
Σχήμα 6.9: *Αποτελέσματα χρόνου εκτέλεσης της εφαρμογής με τον AS1 συμπίεσης χωρίς την χρήση της αποσυμπίεσης δεδομένων*

Χρησιμοποιώντας τον τρόπο αυτό παρατηρούμε (Σχήμα 6.9) ότι υπάρχει μεγάλη μείωση του χρόνο από τα προηγούμενα μας σενάρια αλλά ακόμα παραμένει μεγαλύτερη από το baseline σενάριο μας. Υπάρχει ένα ποσοστό 30% διαφοράς με το βασικό μας σενάριο. Ένα ποσοστό που δεν μας επιτρέπει να μιλάμε για καλούς χρόνους εκτέλεσης και μας κάνει να ψάξουμε να βρούμε μια διαφορετική εφαρμογή όπου θα μας επιτρέπει την αύξηση της επίδοσης.

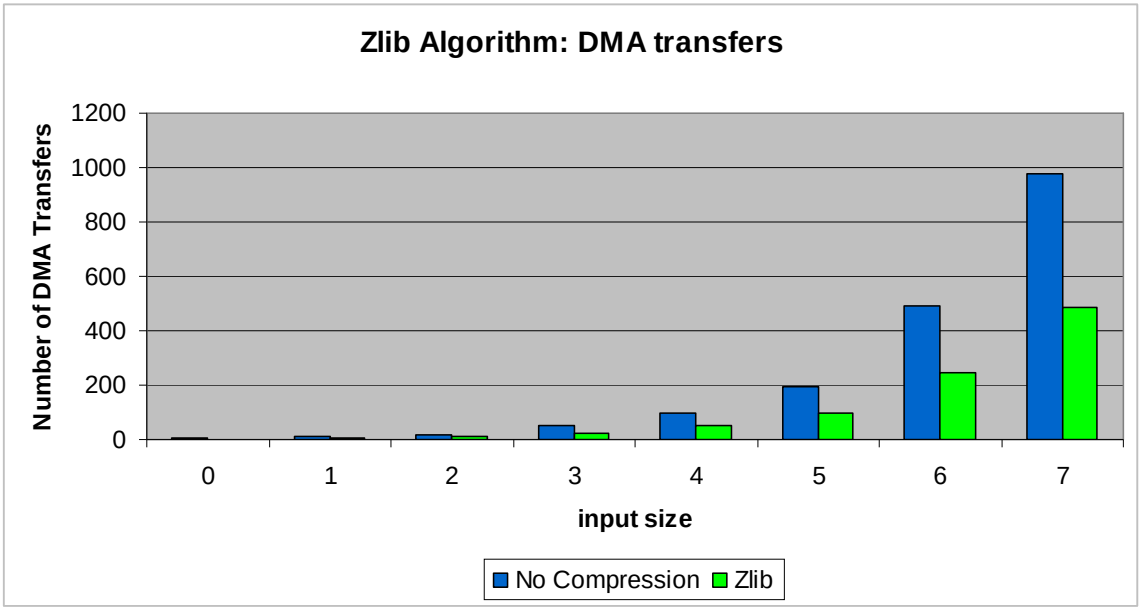
Τέλος στο Σχήμα 6.13 παραθέτουμε τα συνοπτικά αποτελέσματα του σεναρίου μας για να παρατηρήσουμε ότι το Double Buffering είναι ο κύριος ανταγωνιστής μας στην εργασία αυτή, αφού παρατηρούμε ότι στο αρχικό σενάριο χωρίς την συμπίεση ο χρόνος μειώνεται όταν προσθέσουμε και το Double Buffering. Αντίθετα στο σενάριο με την συμπίεση δεν παρατηρούμε οποιαδήποτε διαφορά. Αυτό σημαίνει ότι η συμπίεση δεδομένων μας κερδίζει τον χρόνο που προηγουμένως μας κέρδιζε το Double Buffering.

6.3.3 Zlib Αλγόριθμος Συμπίεσης

Στο τελευταίο σενάριο μας είχαμε δοκιμάσει την ίδια τεχνική με διαφορετικό αλγόριθμο συμπίεσης. Ένα έτοιμο αλγόριθμο, τον Zlib (Κεφάλαιο 3). Στο σενάριο αυτό είχαμε κατακόρυφη αύξηση του χρόνου εκτέλεσης (Σχήμα 6.10). Παρόλο που η αναλογία της συμπίεσης ήταν περίπου στα ίδια επίπεδα με του δικού μας αλγορίθμου συμπίεσης, περίπου στο 2.25 (Σχήμα 6.11) δεν ήταν αρκετό για να μας μειώσει το χρόνο. Ο χρόνος όμως που χρειαζόταν για να γίνει η αποσυμπίεση ήταν πάρα πολύ μεγάλος και τέτοιος ώστε να μας αυξήσει τον συνολικό χρόνο εκτέλεσης κατακόρυφα (Σχήμα 6.12).



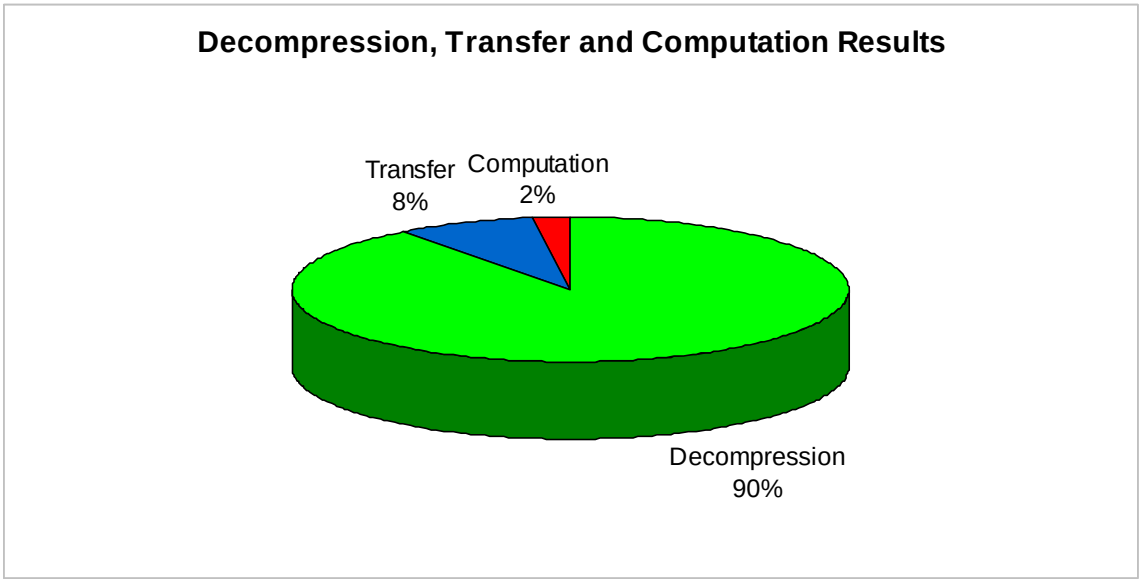
Σχήμα 6.10: *Αποτελέσματα χρόνου εκτέλεσης της εφαρμογής με τον Zlib Αλγόριθμο*



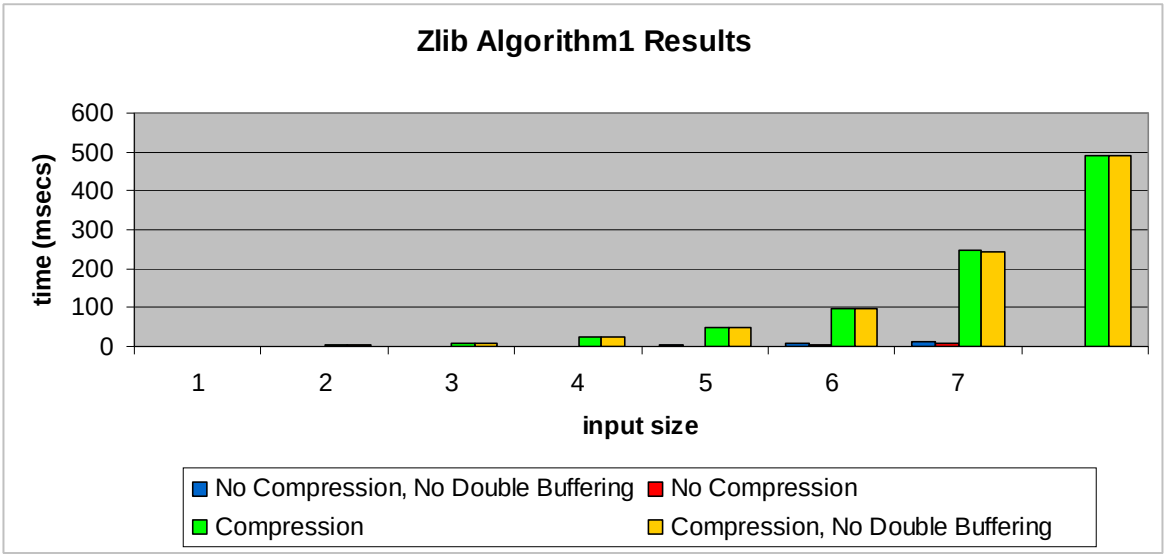
Σχήμα 6.11: *Αριθμός μεταφορών δεδομένων (DMA transfers) χρησιμοποιώντας τον αλγόριθμο Zlib*

Προσπαθήσαμε να χρησιμοποιήσουμε ή να μην χρησιμοποιήσουμε κάποιες από τις υπόλοιπες τεχνικές για να δούμε εάν κερδίζουμε χρόνο αλλά οι χρόνοι εκτέλεσης ήταν οι ίδιοι. Και επειδή δεν μπορούσαμε να αλλάξουμε την συνάρτηση αποσυμπίεσης για τον

αλγόριθμο αυτό αναγκαστήκαμε να περιοριστούμε σε αυτά τα αποτελέσματα και να καταλήξουμε ότι με τον συγκεκριμένο αλγόριθμο και την συγκεκριμένη εφαρμογή δεν μπορούσαμε να κερδίσουμε περισσότερο χρόνο και να αυξήσουμε την επίδοση της εφαρμογής μας.



Σχήμα 6.12 *Ποσοστιαία αναλογία χρόνου μεταφοράς, αποσυμπίεσης και υπολογισμού των δεδομένων για τον Zlib Αλγόριθμο*



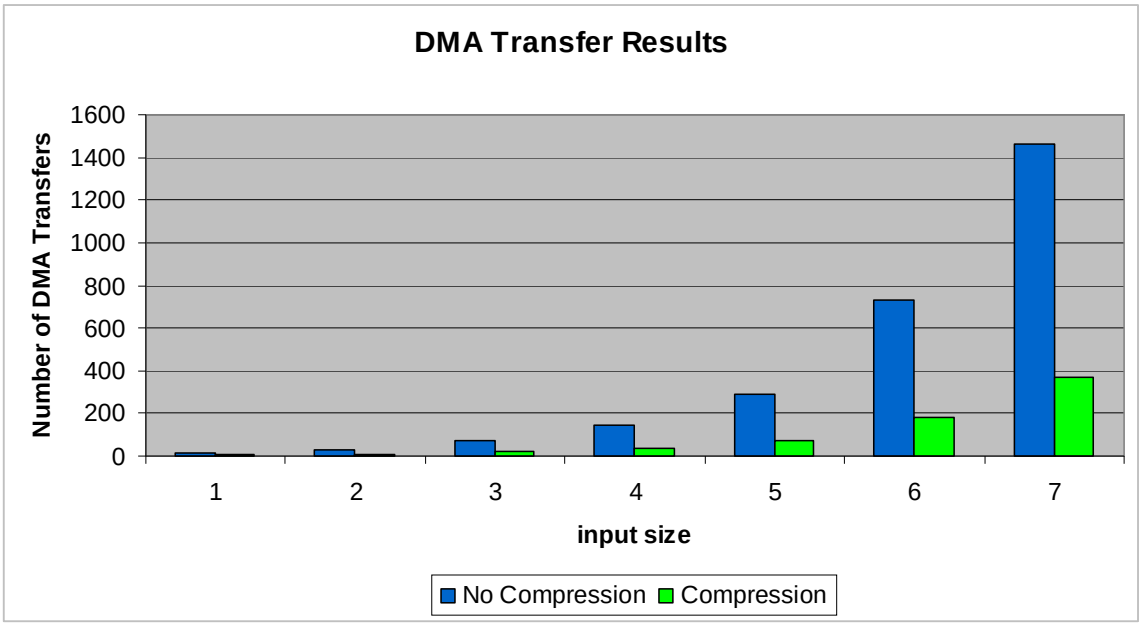
Σχήμα 6.13 *Συνοπτική παρουσίαση αποτελεσμάτων* Zlib *Αλγορίθμου με την χρήση και χωρίς του Double Buffering*

Τέλος στο Σχήμα 6.13 παραθέτουμε τα συνοπτικά αποτελέσματα του σεναρίου μας για να παρατηρήσουμε ότι το Double Buffering είναι ο κύριος ανταγωνιστής μας στην εργασία αυτή, αφού παρατηρούμε ότι στο αρχικό σενάριο χωρίς την συμπίεση ο χρόνος μειώνεται όταν προσθέσουμε και το Double Buffering. Αντίθετα στο σενάριο με την συμπίεση δεν παρατηρούμε οποιαδήποτε διαφορά. Αυτό σημαίνει ότι η συμπίεση δεδομένων μας κερδίζει τον χρόνο που προηγουμένως μας κέρδιζε το Double Buffering.

6.4 Παρουσίαση Αποτελεσμάτων Synthetic Application

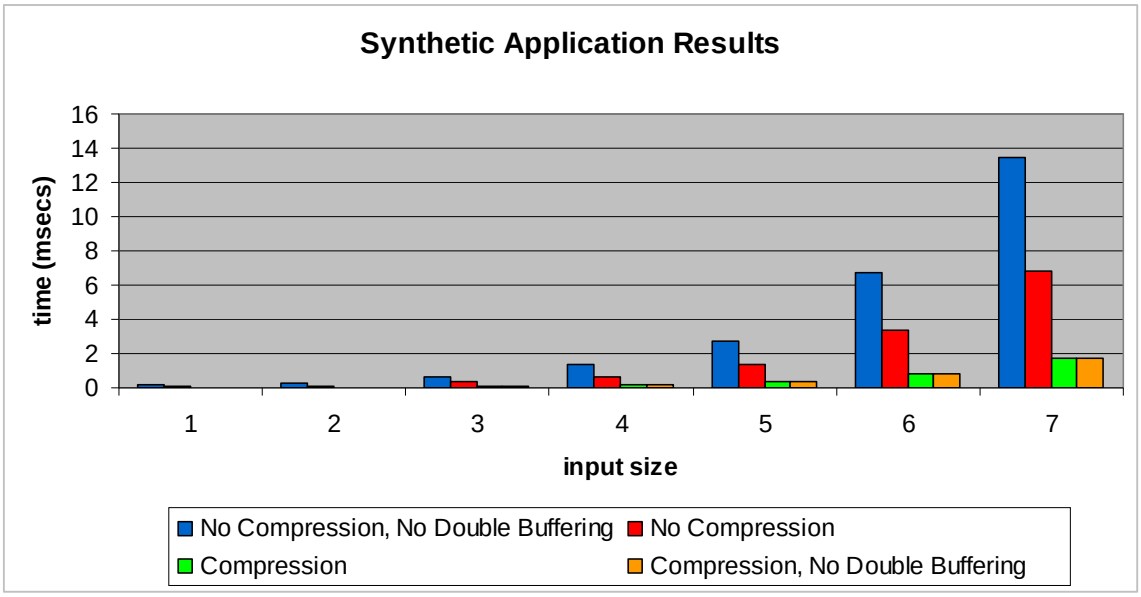
Όπως δείξαμε και στα αποτελέσματα πιο πάνω, δεν έχουμε πάρει θετικά στοιχεία ούτε και οποιαδήποτε αύξηση στην επίδοση της εφαρμογής που χρησιμοποιήσαμε. Αυτό που εξηγούμε όμως και στο Κεφάλαιο 5 είναι ότι έπρεπε να δημιουργήσουμε μια εφαρμογή η οποία θα μας αυξάνει την επίδοση η τεχνική της συμπίεσης δεδομένων. Έτσι δημιουργήσαμε την Συνθετική Εφαρμογή (Synthetic Application) και δείχνουμε πιο κάτω τα αποτελέσματα που έχουμε πάρει. Περισσότερες πληροφορίες όσων αφορά την δομή της εφαρμογής αυτής περιγράφονται στο Κεφάλαιο 5.

Με την νέα μετατροπή δημιουργήσαμε μια εφαρμογή όπου σαν είσοδο τα δεδομένα της προηγούμενης μας εφαρμογής παίρναμε με την συμπίεση δεδομένων 4x λιγότερες μεταφορές δεδομένων. Στο Σχήμα 6.13 φαίνεται γραφικά ο αριθμός των μεταφορών των δεδομένων από την κεντρική μνήμη στα SPE's ανά SPE.

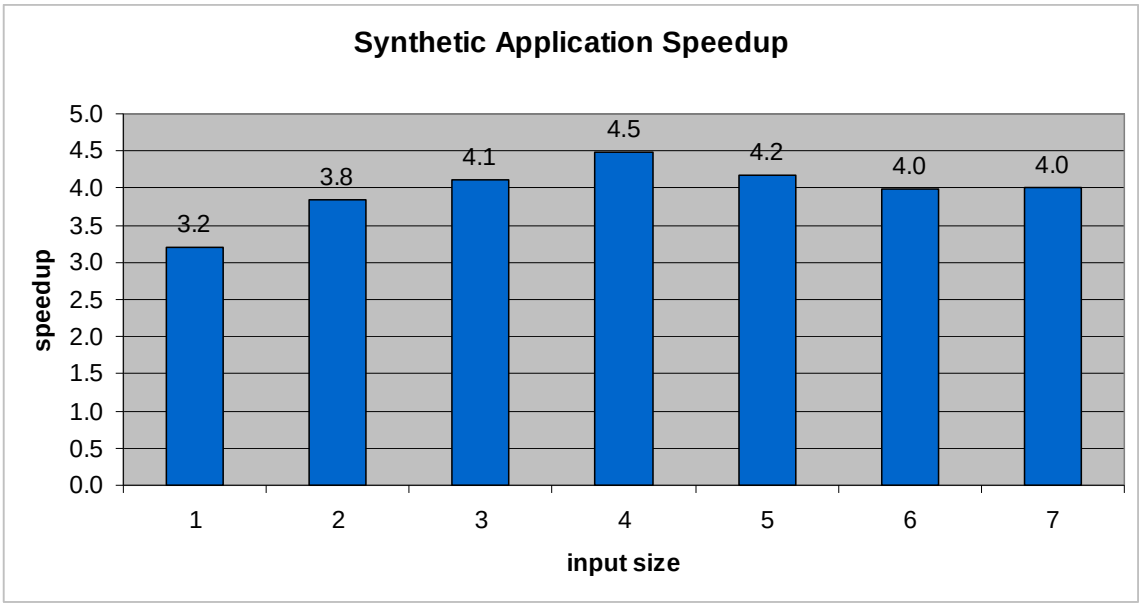


Σχήμα 6.13 *Αποτελέσματα αριθμού μεταφοράς δεδομένων ανά SPE της Συνθετικής Εφαρμογής έναντι του καινούργιου μας* baseline *σεναρίου*

Μειώσαμε ακόμα περισσότερο τις μεταφορές δεδομένων σε σχέση με τα προηγούμενα σενάρια και σε συνδυασμό με τις αλλαγές που έχουμε κάνει στην εφαρμογή αυτή έχουμε κερδίσει πάρα πολύ χρόνο εκτέλεσης(Σχήμα 6.14(α)). Η επίδοση που κερδίσαμε είναι σχεδόν ανάλογη με το ποσοστό/αναλογία συμπίεσης των δεδομένων μας(Σχήμα 6.14(β)). Με αυτά τα αποτελέσματα μπορούμε ασφαλισμένα να πούμε ότι η τεχνική της συμπίεσης δεδομένων πάνω σε επεξεργαστή με κατανεμημένη την μνήμη του στους διάφορους επεξεργαστές (Cell/BE) μπορεί να λειτουργήσει θετικά και να κερδίσει χρόνο εκτέλεσης. Στο σχήμα 6.14(α) βλέπουμε επίσης σενάρια τα οποία αφαιρέσαμε και το Double Buffering. Παρατηρούμε ότι στο σενάριο χωρίς την συμπίεση δεδομένων η τεχνική αυτή κερδίζει μεγάλο ποσοστό χρόνου, ενώ στα σενάρια που χρησιμοποιούμε την συμπίεση δεν βλέπουμε καμία διαφορά είτε χρησιμοποιήσουμε double buffering, είτε όχι.



Σχήμα 6.14(α) *Αποτελέσματα χρόνου εκτέλεσης της Συνθετικής μας εφαρμογής έναντι του καινούργιου μας baseline σεναρίου, με και χωρίς double buffering*



Σχήμα 6.14(β) *Επίδοση της Συνθετικής μας εφαρμογής έναντι του καινούργιου μας* baseline *σεναρίου*

6.5 Περίληψη Αποτελεσμάτων

Συνοψίζοντας σε αυτό το σημείο της εργασίας θα αναλύσουμε την σημασία των αποτελεσμάτων για την δική μας δουλεία αλλά και γενικά τι σημαίνουν τα αποτελέσματα που πήραμε. Υλοποιώντας την παράλληλη εφαρμογή χωρίς την συμπίεση δεδομένων είδαμε ότι παίρνουμε μεγάλη επίδοση (κοντά στην γραμμική αύξηση της επίδοσης) σε σχέση με ένα σειριακό επεξεργαστή. Προσθέτοντας μερικές ακόμα τεχνικές προγραμματισμού όπως το SIMD και το Double Buffering αυξάνουμε ακόμα περισσότερο την επίδοση στον παράλληλο υπολογισμό.

Προχωρώντας στην εφαρμογή της συμπίεσης των δεδομένων παρατηρούμε πολλές αλλαγές στην συμπεριφορά των παράλληλων υπολογισμών. Το πρώτο που είδαμε και αναμέναμε να δούμε ήταν μείωση των μεταφορών των δεδομένων. Η μείωση αυτή είναι πάντα ανάλογη της αναλογίας συμπίεσης των δεδομένων. Δηλαδή (όπως είδαμε και στα δικά μας σενάρια) αν η αναλογία συμπίεσης ήταν 2, τότε οι μεταφορές των δεδομένων θα μειωθούν στις μισές. Παρόλα αυτά είχαμε την αντίθετη αντίδραση στον χρόνο εκτέλεσης. Δηλαδή παρόλο που καταφέραμε να μειώσουμε σημαντικά τις μεταφορές των δεδομένων παρατηρούμε ότι ο χρόνος εκτέλεσης αυξάνεται. Αυτό βέβαια οφείλεται στο ότι τα δεδομένα που συμπιέζονταν έπρεπε πριν την επεξεργασία τους να αποσυμπιεστούν. Ο χρόνος αποσυμπίεσης όπως είδαμε ήταν πολύ μεγαλύτερος από τον χρόνο μεταφοράς, συνεπώς αυτό μας αύξησε τον συνολικό χρόνο επεξεργασίας.

Αυτό που παρατηρήσαμε σε όλα τα σενάρια και φαίνεται και στις γραφικές μας παραστάσεις (Σχήμα 6.8 και Σχήμα 6.14(α)) είναι ότι στα σενάρια όπου χρησιμοποιούμε την συμπίεση αλλά αφαιρούμε το double buffering δεν έχουμε αύξηση του χρόνου εκτέλεσης σε σχέση με το σενάριο όπου έχουμε το double buffering. Αυτό που περιμέναμε ήταν να έχουμε μια μικρή αύξηση στον χρόνο εκτέλεσης αλλά παρατηρούμε το αντίθετο. Από τα αποτελέσματα μας στο Synthetic Application όπου καταφέραμε να αυξήσουμε την επίδοση της εφαρμογής μας με την συμπίεση των δεδομένων μπορούμε με ασφάλεια να πούμε ότι το double buffering ήταν ίσως ο κυριότερος αντίπαλος της τεχνικής της συμπίεσης δεδομένων (αγνοώντας φυσικά τον χρόνο αποσυμπίεσης των δεδομένων). Στο

Σχήμα 6.14(α) βλέπουμε ότι η συμπίεση των δεδομένων καταφέρνει να νικήσει το double buffering. Καταλήγοντας έτσι στο συμπέρασμα ότι η τεχνική της συμπίεσης των δεδομένων μπορεί να αντικαταστήσει την τεχνική του double buffering σε περιπτώσεις που δεν μπορεί να χρησιμοποιηθεί ή σε μηχανές που δεν υποστηρίζουν την τεχνική αυτή.

Τελειώνοντας την ανάλυση των αποτελεσμάτων πρέπει να επισημάνουμε ότι αν καταφέρουμε να ξεπεράσουμε ή να αποκρύψουμε τον χρόνο της αποσυμπίεσης των δεδομένων τότε μπορούμε να έχουμε μεγάλο όφελος στην επίδοση μας, όπως φαίνεται και στο Synthetic Application όπου επεξεργαζόμασταν συμπιεσμένα δεδομένα. Επίσης σε μηχανές που δεν υποστηρίζουν το double buffering, μπορούμε να χρησιμοποιήσουμε την συμπίεση των δεδομένων επιτυχημένα.

Κεφάλαιο 7

Συμπεράσματα

7.1 Συμπεράσματα

Στην εργασία αυτή παρουσιάσαμε την τεχνική της συμπίεσης δεδομένων πάνω στον επεξεργαστή Cell/BE. Προσπαθήσαμε να εφαρμόσουμε μια τεχνική η οποία εφαρμόζεται σε μεγάλα συστήματα υπολογισμού τα οποία όμως ακολουθούν την αρχιτεκτονική της κατανεμημένης επεξεργασίας. Λόγω του ότι και ο επεξεργαστής Cell/Be ακολουθεί αυτή την τεχνική, με την διαφορά ότι η επεξεργασία είναι κατανεμημένη πάνω στο ίδιο κύκλωμα (chip), μας ώθησε στο να προσπαθήσουμε να την εφαρμόσουμε και εδώ και να καταφέρουμε να πετύχουμε αύξηση στην επίδοση.

Η προσπάθεια αυτή, παρόλο που τα αποτελέσματα για την κύρια εφαρμογή μας δεν ήταν θετικά, μας έδωσε σημαντικές παρατηρήσεις και σημαντικά συμπεράσματα τα οποία μπορούν αξιολογηθούν και να αξιοποιηθούν ανάλογα και σε άλλες εφαρμογές. Πιο κάτω παραθέτουμε και εξηγούμε τα σημαντικότερα συμπεράσματα που έχουμε πάρει με την διεκπεραίωση της εργασίας αυτής.

Έχουμε παρατηρήσει ότι ο χρόνος μεταφοράς δεδομένων μέσω το EIB πάνω στον Cell/BE είναι αρκετά γρήγορη και πολλές φορές ακόμα και γρηγορότερη από τον χρόνο

επεξεργασίας των δεδομένων. Λόγω της φύσης του επεξεργαστή που επιτρέπει την χρήση της τεχνικής του Double Buffering ο χρόνος μεταφοράς καλύπτεται από τον χρόνο επεξεργασίας δεδομένων. Έτσι αυτό που εμείς έχουμε να κάνουμε είναι να εκμεταλλευτούμε στο έπακρων τις δυνατότητες του επεξεργαστή στην επεξεργασία έτσι ώστε να μπορέσουμε να κάνουμε την επεξεργασία δεδομένων όσο πιο γρήγορη γίνεται. Με αυτό τον τρόπο ο χρόνος μεταφοράς δεδομένων θα μείνει εκτεθειμένος και μεγαλύτερος από τον χρόνο επεξεργασίας και ακόμα και η τεχνική του Double Buffering δεν θα μπορεί να υπερκαλύψει τον χρόνο αυτό. Έτσι θα αφήσει χώρο για την συμπίεση δεδομένων να δράση και να μειώσει τον χρόνο μεταφοράς δεδομένων και συνεπώς να μειώσει και τον συνολικό χρόνο εκτέλεσης.

Η συμπίεση δεδομένων μπορεί επίσης να δράσει και σαν αντικαταστάτης άλλων τεχνικών που ήδη χρησιμοποιά ο Cell/BE. Μπορεί να αντικαταστήσει το Double Buffering εφόσον ο χρόνος αποσυμπίεσης είναι αρκετά μικρός ή αν η επεξεργασία των δεδομένων μπορεί να γίνει πάνω σε συμπιεσμένα δεδομένα. Μπορεί επίσης να αντικαταστήσει και τον SIMD προγραμματισμό. Παρόλο που ο Cell/BE μπορεί να υποστηρίξει πάρα πολλές λειτουργίες και υπολογισμού σε SIMD μοντέλο, εάν μια εφαρμογή δεν μπορεί να χρησιμοποιήσει το μοντέλο αυτό τότε ο χρόνος εκτέλεσης θα αυξηθεί. Επομένως με την χρήση της συμπίεσης των δεδομένων μπορούμε να μειώσουμε τον συνολικό χρόνο όπως συμβαίνει και στην δική μας περίπτωση με το Synthetic Application που υλοποιήσαμε.

Τελειώνοντας για να χρησιμοποιήσουμε την τεχνική της συμπίεσης δεδομένων θα πρέπει να γίνει αναλυτική έρευνα στο σύστημα που θα γίνει η χρήση της για να μπορέσουμε να την εκμεταλλευτούμε πλήρως αλλά και να δούμε κατά πόσον η τεχνική αυτή μπορεί να εφαρμοστεί θετικά πάνω στο σύστημα μας. Επίσης σημαντικό είναι να γίνει και ανάλυση της εφαρμογής που θα εκτελέσουμε έτσι ώστε να δούμε ότι μπορούμε να εφαρμόσουμε την τεχνική αυτή και να έχουμε αύξηση της επίδοσης και όχι μείωση. Όπως έχουμε δει και στην δική μας έρευνα η ανάλυση αυτή είναι εξαιρετικά σημαντική γιατί η συμπίεση δεδομένων είναι μια χρονοβόρα διαδικασία η οποία σε πολλές περιπτώσεις μπορεί να είναι καταστροφική για την επίδοση μιας εφαρμογής. Συνεπώς η σωστή και πλήρης ανάλυση

όλων των εμπλεκόμενων κομματιών, έτσι ώστε να μπορέσει να επιφέρει θετικά αποτελέσματα, είναι αναγκαία.

7.2 Μελλοντική Δουλεία

Κατά την διάρκεια της εργασίας αυτής και καθώς παρατηρούσα τα αποτελέσματα αλλά και τις υλοποιήσεις, δημιουργήθηκαν πολλά ερωτηματικά και πολλές απορίες. Τα περισσότερα από αυτά τα απαντούμε μέσα από την δουλεία αυτή αλλά υπάρχουν αρκετά άλλα που δημιουργήθηκαν και η εργασία αυτή δεν ήταν αρκετή για την απάντηση τους. Τα ερωτήματα αυτά δεν ήταν μέσα στα όρια της δικής μας δουλεία για τον λόγο αυτό δεν απαντήσαμε αφού δεν μπορούσαμε να επεκτείνουμε την δουλεία εκτός κάποιων ορίων. Για τον λόγο αυτό παραθέτουμε εδώ μερικά από αυτά τα ερωτήματα σαν ιδέες για μελλοντικές δουλείες πάνω στην ιδέα την δική μας.

Μέσα από την δική μας την δουλεία μπορούμε να δούμε καθαρά ότι η τεχνική αυτή μπορεί να χρησιμοποιηθεί με θετικά αποτελέσματα σε ορισμένες εφαρμογές. Αυτό μας δίνει την δυνατότητα να ψάξουμε για εφαρμογές αυτού του τύπου. Δηλαδή εφαρμογές που θα μπορούν να εκμεταλλευτούν σωστά και αποδοτικά την τεχνική της συμπίεσης των δεδομένων. Σε μια μελλοντική δουλεία θα μπορούσαμε να ψάξουμε για άλλες εφαρμογές, είτε επιστημονικές, είτε διάφορα άλλα benchmarks, τα οποία θα καταφέρναμε να δουλέψουμε με την τεχνική της συμπίεσης των δεδομένων με τέτοιο τρόπο που θα καταφέρναμε να αυξήσουμε την επίδοση της εφαρμογής. Επίσης με αυτό το σκεπτικό θα μπορούσαμε να αλλάξουμε ακόμα και την μηχανή πάνω στην οποία δουλεύουμε και να την αντικαταστήσουμε με κάποια άλλη με παρόμοια αρχιτεκτονική. Για παράδειγμα θα μπορούσαμε να το δοκιμάσουμε και πάνω σε κάρτες γραφικών και να χρησιμοποιούμε την συμπίεση των δεδομένων προτού αποστείλουμε τα δεδομένα στην μνήμη της κάρτας γραφικών, έτσι ώστε να μειώσουμε τον χρόνο μεταφοράς δεδομένων από την κεντρική μνήμη στην μνήμη της κάρτας γραφικών.

Θα μπορούσαμε επίσης σε μια άλλη δουλεία να εφαρμόσουμε την τεχνική της συμπίεσης πάνω σε συστάδες επεξεργαστών αυτού του τύπου. Μεγάλοι υπολογιστές σε διάφορα πανεπιστήμια και ιδιωτικούς οργανισμούς χρησιμοποιούν την Cell/BE σε συστάδες υπολογιστών για να μπορέσουν να κτίσουν μια σημαντικά μεγάλη επίδοση στο σύστημα τους. Έτσι θα ήταν σοφό να δοκιμάσουμε να χρησιμοποιήσουμε την τεχνική αυτή για την μεταφορά των δεδομένων από συστάδα σε συστάδα μέσα σε ένα μεγάλο σύστημα. Ακόμα θα μπορούσαμε να το έχουμε και πάνω σε κάθε chip επεξεργαστή Cell/BE.

Βιβλιογραφία

[1] Pedro Trancoso, Despo Othonos, Artemakis Artemiou, "Data Parallel Acceleration of Decision Support Queries Using Cell/BE and GPU's," in *Proceedings of the 6th ACM conference on Computing frontiers,* 2009, pp. 117-126.

[2] Sandor Heman, Niels Nes, Marcin Zukowski, Peter Boncz, "Vectorized Data Processing on the Cell Broadband Engine," in *Proceedings of the 3rd international workshop on Data management on new hardware,* 2007, Article No. 4.

[3] Burga Gedik, Philip S. Yu, Rajesh R. Bordawekar, Thomas J. Watson, "Executing Stream Joins on the Cell Processor," in *Proceedings of the 33rd international conference on Very large data bases,* 2007, pp. 363-374.

[4] Huiliang Zhang, Bertil Schmidt and Wolfgang Muller-Witting, "Accelerating BLASTP on the Cell Broadband Engine," in *Proceedings of the Third IAPR International Conference on Pattern Recognition in Bioinformatics,* 2008, pp. 460-470.

[5] Luke J. Gosink, Kesheng Wu, E.Wes Bethel, John D. Owens and Kenneth I. Joy, "Data Parallel Bin-Based Indexing for Answering Queries on Multi-core Architectures", in *Proceedings of the 21st International Conference on Scientific and Statistical Database Management,* 2009, pp. 110-129.

[6] Werner Mach, Erich Schikute, "Parallel algorothms for execution of relational database operations revisited on Grids," in *International Journal of High Performance Computing Applications,* 2009, pp. 152-170.

[7] Rajesh Bordwekarm Lipyeow, Oded Shmueli, "Parallelization of XPath Queries using Multi-core Processors: Challenges and Experiences," in *Proceedings of the 13th International Conference on Extending Database Technology,* 2010, pp. 159-170.

[8] S. Chattopadhyay, S. Mitra, P. Pal Chaudhuri, "Cellular Automata Based Architecture of a Database Query Processor," in *Proceedings of the 9th International Conference on VLSI Desing: VLSI in Mobile Communication,* 1996, pp. 320.

[9] W. Richard Stevens, Stephen A. Rago, *Advanced Programming in the Unix environment* (Second Edition).

[10] Calvin Lin and Lawrence Snyder, *Princeiples of Parallel Programming.*

[11] Ibm.com/redbooks(RedBooks), *Programming the Cell Broadband Engine™ Architectures, Examples and Best Practices.*

[12] CBEA JSRE Series (IBM), *C/C++ Language Extensions for Cell Broadband Engine Architecture.*

[13] IBM, *Cell Broadband Engine Programming Handbook,* Version 1.1.

[14] IBM, *Security Software Development Kit 3.0. Installation and User's Guide,* Version 3.01.

[15] Farshad Khunjush and Nikitas J. Dimopoulos, "Extended Characterization of DMA Transfers on the Cell BE Processor," in *IEEE International Symposium on Parallel and Distributed Processing,* 2008, pp. 1-8.

[16] An Oracle White Paper, “Oracle Advanced Compression”, April 2008.

[17] Goetz Graefe, Leonard D. Shapiro, “Data Compression and Database Performance”, in *ACM/IEEE-CS Symposium on Apllied Computing,* 1991.

[18] Ajith Padyana, T.V. Sira Kumar, P.K. Baruah, “Fast Floating Point Compression on the Cell BE Processor”.

[19] Martin Burtscher and Paraj Ratanaworabhan, “High Troughput Compression of Double-Precision Floating Point Data”.

[20] Gay E. Blelloch, “Introduction to Data Compression”.

[21] Peter Lindstrom, Martin Isenburg, “Fast and Efficient Compression of Floating-point Data, ” in *IEEE Transactions on Visualization and Computer Graphics,* 2006, pp. 1245-1250.

[22] Manuel N. Gamito and Miguel Salles Dias, “Lossless Coding of Floating-Point Data with JPEG 2000 Part 10”.

[23] Alaa R. Alameldeen and David A. Wood, “Adaptive Cache Compression for High-Performance Processors,” in *Proceedings of the 31st annual international symposium on computer architecture,* 2004, pp. 212.

[24] David A. Bader, Virat Agaiwal, Kamesh Madduri, Seunghwa Kang, “High Performance Combinational algorithm design on the Cell Broadband Engine processor”.

[25] Georgios Keramidas, Konstantinos Aisopos and Stefanos Kaxiras, “Dynamic Dictionary-Based Data Compression for Level-1 Caches,” Architecture of

Computing Systems, 2006, pp. 114-129.

[26] Jeffrey Scott Vitter, ALGORITHM 673 Dynamic Huffman Coding.

[27] Jeffrey Scott Vitter, Design and Analysis of Dynamic Huffman Coding.

[28] Khalid Sayood, *Introduction to Data Compression*, 3rd Edition.

[29] David Salomon, *Data Compression, the complete Reference*, 4th Edition.

[30] Daniel Hackenberg, ZIH, TU-Dresden, "Fast Matrix Multiplication for Cell BE processors".

[31] Zlib Compression Code. [Online]. Available: www.zlib.net

[32] Lz77 Compression Algorithm. [Online]. Available: http://oldwww.rasip.fer.hr/research/compress/algorithms/fund/lz/lz77.html

[33] General Information for Wikipedia. [Online]. Available: www.wikipedia.org

[34] Cell/BE Information and subjects. [Online]. Available: www.ibm.com/developerworks/power/cell/

[35] Zlib Compression Algorithm. [Online]. Available: http://en.wikipedia.org/wiki/LZ77_and_LZ78

[36] TPC-H Queries and Q6 Query information. [Online]. Available: http://www.tpc.org/

Παράρτημα Α

Το παράρτημα αυτό αναφέρεται στο Κεφάλαιο 3 της Υλοποίησης των Αλγορίθμων Συμπίεσης και Αποσυμπίεσης. Εδώ θα δείξουμε τις συναρτήσεις συμπίεσης του Αλγορίθμου 1 και τις συναρτήσεις αποσυμπίεσης 1.1 και 1.2.

Συνάρτηση Συμπίεσης Αλγορίθμου 1 – AS1

```
int compress(int LINEITEM)
{

int i = 0, j = 0;
float extendedPrice, discount, year, quantity;
float tempPrice, tempDiscount, tempYear, tempQuantity;
int tempPriceInt, tempPrice1, tempPrice2, tempPrice3, tempPrice4;


//copy the first value of each different type of data
      //in line_item[x][y] .. x == 1 because the count variable begins
with 1 when reading from the file
extendedPrice = line_item[1][0];
discount = line_item[1][1];
year = line_item[1][2];
quantity = line_item[1][3];


//find the smallest values for each type in all the database
for(i = 1; i < LINEITEM; i++)
{

      if(line_item[i][1] < discount)
            discount = line_item[i][1];
      if(line_item[i][2] < year)
            year = line_item[i][2];
      if(line_item[i][3] < quantity)
            quantity = line_item[i][3];
}

//keep the smallest values which we will need later to send to the SPEs
baseline_data[0][0] = extendedPrice;
baseline_data[0][1] = discount;
baseline_data[0][2] = year;
baseline_data[0][3] = quantity;
```

```
//compress the data
for(i = 0, j = 1; j < LINEITEM ; i++, j++)
{

//extra operations needed for the extended price value
tempPrice = line_item[j][0];
tempPriceInt = (int)tempPrice;
tempPrice4 = (int)(((float)(tempPrice - tempPriceInt)) * 100);

tempPrice3 = (int)tempPriceInt % 100;

tempPriceInt = (int)(tempPriceInt / 100);
tempPrice2 = (int)tempPriceInt % 100;

tempPriceInt = (int)(tempPriceInt / 100);
tempPrice1 = (int)tempPriceInt % 100;

//extra operations needed for the discount value
tempDiscount =  (line_item[j][1] - discount);
tempDiscount = tempDiscount * 100;

tempYear     =    (line_item[j][2] - year);
tempQuantity =  (line_item[j][3] - quantity);

line_item_compressed[j][0] = (char)tempPrice1;
line_item_compressed[j][1] = (char)tempDiscount;
line_item_compressed[j][2] = (char)tempYear;
line_item_compressed[j][3] = (char)tempQuantity;
line_item_compressed[j][4] = (char)tempPrice2;
line_item_compressed[j][5] = (char)tempPrice3;
line_item_compressed[j][6] = (char)tempPrice4;


}

return CORRECT;

}
```

Συνάρτηση Αποσυμπίεσης Αλγορίθμου 1.1

```
int decompression(int buf_idx, int iterations)
{
      int i = 0;

      for(i = 0; i < iterations; i++)
            {

                  data_buffer[i][2] =
(float)((int)data_buffer_compressed[buf_idx][i][2] +
(float)baseline_data[0][2]);
```

```
                  data_buffer[i][1] =
(float)(((int)data_buffer_compressed[buf_idx][i][1] +
(float)baseline_data[0][1]) / 100);

                  data_buffer[i][3] =
(float)((int)data_buffer_compressed[buf_idx][i][3] +
(float)baseline_data[0][3]);

                  data_buffer[i][0] =
(float)(((float)data_buffer_compressed[buf_idx][i][0]*10000) +
((float)data_buffer_compressed[buf_idx][i][4]*100) +
((float)data_buffer_compressed[buf_idx][i][5]) +
((float)data_buffer_compressed[buf_idx][i][6]/100));


            }

return CORRECT;
}
```

Συνάρτηση Αποσυμπίεσης Αλγορίθμου 1.2

```
void decompression(int temp_partition, int idx)
{

      int i = 0, k = 0, j = 0;


for(i = 0, j = 0; i < temp_partition; i++, j++)
      {
            for(k = 0; k < 4; k++, i++)
                  {

            F(data_buffer_decompressed[0][j])[k] =
(float)(((float)data_buffer[idx][i][0]*10000) +
((float)data_buffer[idx][i][4]*100) + ((float)data_buffer[idx][i][5]) +
((float)data_buffer[idx][i][6]/100));

            F(data_buffer_decompressed[1][j])[k] =
(float)((int)data_buffer[idx][i][1]);
            F(data_buffer_decompressed[2][j])[k] =
(float)((int)data_buffer[idx][i][2]);
            F(data_buffer_decompressed[3][j])[k] =
(float)((int)data_buffer[idx][i][3]);
            //fprintf(stderr, "data_buffer_decompressed == %.2f\n",
F(data_buffer_decompressed[2][j])[k]);
                  }
i--;

      }}
```