ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΤΟΜΙΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ατομική Διπλωματική Εργασία

A study of the benefits of automatic parallelization

Παναγιώτα Παναγίδου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

A study of the benefits of automatic parallelization

Παναγιώτα Παναγίδου

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 2015

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλα αυτά τα άτομα που ήταν δίπλα μου σε όλη αυτή την διαδρομή στο τέλος του πτυχίου μου και κατανοούσαν πραγματικά τι περνούσα λόγω κάποιων προβλημάτων υγείας.

- Πρώτα απ'όλα ευχαριστώ τους γονείς μου για την απεριόριστη πίστη και στήριξη τους σε μένα όλα αυτά τα χρόνια , την αδελφή μου και τον αδελφό μου για το κουράγιο και την αγάπη που μου έδιναν έστω και μακριά, τις φίλες μου για την θετική ενέργεια που μου έστελναν και προπάντων ευχαριστώ τον σύντροφο μου που ήταν ο φύλακας άγγελος μου αυτά τα χρόνια και σε κάθε δυσκολία ήταν εκεί για να μου δίνει θάρρος να συνεχίσω.

- Επίσης θα ήθελα να ευχαριστήσω τους καθηγητές μου κ. Χρύση Γεωργίου και κ.Χρίστο Χριστοδούλου για την κατανόηση και συμπαράσταση τους για όλες τις δυσκολίες που περνούσα στο τελευταίο εξάμηνο. Εκτός από εξαιρετικοί καθηγητές, αποδείχθηκαν και σπουδαίοι άνθρωποι!

- Και ένα μεγάλο ευχαριστώ και στην ψυχολόγο του Πανεπιστημίου κ.Χρυστάλλα Κουτσογιώργη που σε μία περίοδο όπου δεν είχα κάποιο σύμβουλο να με βοηθήσει είτε για ακαδημαϊκά θέματα είτε για άλλα προβλήματα, ήταν εκεί σε κάθε στιγμή και με καθοδηγούσε και με συμβούλευε.

Και τέλος, ευχαριστώ τον κύριο Trancoso και τον κύριο Διαβαστό που μου έδωσαν την ευκαιρία να ολοκληρώσω και να παραδώσω την διπλωματική μου.

Περίληψη

Οδηγούμενοι από την ανάγκη δημιουργίας υψηλού επιπέδου επίδοσης για μια ποικιλία εφαρμογών που πρέπει να καλύπτουν τόσο τις ανάγκες των απλών καταναλωτών αλλά και των επιχειρήσεων κα11ι του επιστημονικού τομέα περάσαμε στη γενιά των παράλληλων υπολογιστών. Ο παραλληλισμός είναι μια γνωστή έννοια που προκύπτει συχνά στη καθημερινή μας ζωή. Το σημαντικό όμως είναι ότι ο παραλληλισμός μπορεί να συμβάλει από πολλές απόψεις στη σταθερή βελτίωση της απόδοσης των υπολογιστών. Με βάση αυτό, άρχισαν να δημιουργούνται καινούργιοι ορίζοντες στο χώρο της πληροφορικής

Παρουσιάστηκε η ανάγκη για νέους τρόπους προσέγγισης του παράλληλου υπολογισμού. Οι προσεγγίσεις αυτές είναι ο παράλληλος προγραμματισμός γραμμένος στο χέρι (hand-coded) και η αυτόματη παραλληλοποίηση από τον μεταγλωττιστή. Κάποιος που θέλει να εμπλακεί με τον παραλληλισμό πρέπει με βάση κάποια κριτήρια είναι σε θέση να αποφασίσει ποια προσέγγιση είναι πιο αποδοτική με βάση τις ανάγκες του.

Όταν λέμε αυτόματη παραλληλοποίηση , εννοούμε στην μετατροπή σειριακού κώδικα σε multi-threaded ή vectorized κώδικα προκειμένου να χρησιμοποιούνται πολλαπλοί επεξεργαστές ταυτόχρονα σε μια multiprocessor μηχανή με κοινόχρηστη μνήμη. Ο στόχος της αυτόματης παραλληλοποίησης είναι να ανακουφίσει τους προγραμματιστές από την διαδικασία να κάνουν οι ίδιοι στο χέρι παράλληλο τον κώδικα..

Ο στόχος όμως αυτής της διπλωματικής εργασίας είναι ο έλεγχος και η σύγκριση του χρόνου εκτέλεσης σε ένα κομμάτι σειριακού κώδικα που του γίνεται μετατροπή σε παράλληλο από το χέρι ή αυτόματα από παράλληλο μεταγλωττιστή.

Επίσης στην Διπλωματική αυτή γίνεται μία ανάλυση των διάφορων μεταγλωττιστών που μετατρέπουν σειριακό κώδικα σε παράλληλο.

Table of contents

Table of Figures

Κεφάλαιο 1: Εισαγωγικά

1.1 Εισαγωγή

Ο στόχος όμως αυτής της διπλωματικής εργασίας είναι ο έλεγχος και η σύγκριση του χρόνου εκτέλεσης σε ένα κομμάτι σειριακού κώδικα που του γίνεται μετατροπή σε παράλληλο από το χέρι ή αυτόματα από παράλληλο μεταγλωττιστή. Έχει γίνει εκτενής μελέτη διάφορων εργαλείων που βοηθούν στην αυτόματη δημιουργία παράλληλου κώδικα καθώς και διαφόρων πειραμάτων που έχουν εκτελεστεί με αυτά. Κάποια από αυτά είναι δωρεάν και ανοικτού κώδικα και κάποια από αυτά παρέχονται μόνο για εμπορικές εφαρμογές με συγκεκριμένο κόστος.

Στη συνέχεια παρουσιάζονται τα αποτελέσματα από πειράματα που έχω εκτελέσει με τον GCC και τα NAS Parallel Benchmarks. Ένα σετ από benchmarks που δίνεται ελεύθερα και δωρεάν και περιλαμβάνει σειριακές και παράλληλες υλοποιήσεις προγραμμάτων.

1.2 Βασικές έννοιες

1.2.1 Parallel Code

Με τον όρο παράλληλο κώδικα εννοούμε κώδικα ο οποίος μπορεί να εκμεταλλευτεί πλήρως τις σύγχρονες τεχνολογίες επεξεργαστών που περιλαμβάνουν περισσότερους από ένα πυρήνες οι οποίοι μπορούν να τρέξουν παράλληλα.

Κάτω από το μικροσκόπιο της παράλληλης επεξεργασίας υπάρχουν τέσσερα είδη υπολογιστικών συστημάτων (ταξινόμηση Φλυν)[5][6], τα οποία διαχωρίζονται με βάση το πόσες ταυτόχρονες ροές επεξεργασίας εντολών και δεδομένων υποστηρίζουν:

- Οι υπολογιστές SISD (Μία Εντολή - Ένα Δεδομένο): οι συνήθεις σειριακοί υπολογιστές Φον Νόιμαν, όπου κάθε στιγμή ένας μοναδικός επεξεργαστής εκτελεί μία εντολή σε ένα δεδομένο το οποίο έχει προσκομίσει από την κύρια μνήμη.

- Οι υπολογιστές SIMD (Μία Εντολή - Πολλαπλά Δεδομένα): μία σπάνια κατηγορία, όπου κάθε στιγμή πολλαπλοί επεξεργαστές εκτελούν μία μόνο κοινή εντολή σε πολλαπλά διαφορετικά δεδομένα. Ο όρος σήμερα περιγράφει σε σημαντικό βαθμό τη λειτουργία των σύγχρονων επεξεργαστών γραφικών.

- Οι υπολογιστές MISD (Πολλαπλές Εντολές - Ένα Δεδομένο): ασυνήθιστη αρχιτεκτονική, όπου κάθε στιγμή πολλαπλοί επεξεργαστές τροποποιούν με διαφορετικό τρόπο την ίδια ροή δεδομένων· συνήθως αξιοποιείται με γνώμονα την αύξηση της ανοχής σφαλμάτων μίας εφαρμογής και όχι την αύξηση των υπολογιστικών επιδόσεων.

- Οι υπολογιστές MIMD (Πολλαπλές Εντολές - Πολλαπλά Δεδομένα): οι πλήρως παράλληλοι υπολογιστές, όπου πολλαπλοί αυτόνομοι επεξεργαστές εκτελούν ταυτοχρόνως πολλαπλές διαφορετικές ροές εντολών σε διαφορετικά δεδομένα.

1.2.2 Automatic Parallelization

Με τον όρο αυτόματη παραλληλοποίηση, εννοούμε στην μετατροπή διαδοχικού κώδικα σε multi-threaded ή vectorized κώδικα προκειμένου να χρησιμοποιούνται

πολλαπλοί επεξεργαστές ταυτόχρονα σε μια multiprocessor μηχανή με κοινόχρηστη μνήμη. Ο στόχος της αυτόματης παραλληλοποίησης είναι να ανακουφίσει τους προγραμματιστές από την χειροκίνητη διαδικασία παραλληλισμού. [1]

Η αυτόματη παραλληλοποίηση εστιάζει στους βρόχους γιατί σε γενικές γραμμές, το μεγαλύτερο μέρος του χρόνου εκτέλεσης ενός προγράμματος γίνεται εντός κάποιας μορφής βρόχου. Μια σημαντική προσέγγιση για την παραλληλοποίηση των βρόχων είναι το pipeline multi-threading. Ένας παράλληλος compiler που χρησιμοποιεί αυτή την προσέγγιση προσπαθεί να σπάσει την ακολουθία των πράξεων μέσα σε ένα βρόχο σε μια σειρά από μπλοκ κώδικα, έτσι ώστε κάθε μπλοκ κώδικα να μπορεί να εκτελεστεί σε ξεχωριστούς επεξεργαστές ταυτόχρονα. [2]

1.2.3 Vectorization

Μια άλλη έννοια που έχω μελετήσει είναι η προσέγγιση του vectorization. [3,4]

Το vectorization είναι μια ειδική περίπτωση αυτόματης παραλληλοποίησης όπου ο compiler παίρνει ένα scalar κώδικα και τον κάνει σε πίνακα. (τον κάνει vectorize).

Για παράδειγμα:

C1=a1+b1

C2=a2+b2

C3=a3+b3

C4=a4+b4

Το πρώτο πράγμα που θα κάνει ο compiler είναι να ελέγξει τις εξαρτήσεις μεταξύ των μεταβλητών. Αν δεν υπάρχουν εξαρτήσεις τότε ο compiler μετατρέπει τις μεταβλητές του κώδικα σε πίνακα, και μετά αυτός ο πίνακας σπάζει σε κομμάτια στους πυρήνες του επεξεργαστή.

```
For(i=0;i<n;i++)
        C[i]=a[i]+b[i];
```

Το ότι γίνεται αυτή η διαδικασία είναι βοήθεια για μετά που θα γίνει παράλληλο. Έτσι που έγινε σε πίνακα ο κώδικας με της μεταβλητές μας, π.χ. αν ήταν 100 στην σειριακή εκτέλεση θα γινόταν ένα-ένα η εκτέλεση των πράξεων, ενώ στην περίπτωση του vectorization ο πίνακας θα έσπαζε σε κομμάτια όπως για παράδειγμα στους 4 πυρήνες του επεξεργαστή και θα εκτελούνταν παράλληλα 4-4 οι πράξεις.

1.3 Προεπισκόπηση των υπόλοιπων κεφαλαίων

Στα υπόλοιπα κεφάλαια θα δοθεί τα κατάλληλο τεχνικό υπόβαθρο για κατανόηση της υπόλοιπης διπλωματικής. Επίσης θα αναλυθούν διάφοροι compilers και πειράματα που έχουν εκτελεστεί με αυτούς, καθώς επίσης και πειράματα που έχω εκτελέσει με τον GCC.

Θα δούμε ότι σε πολλές περιπτώσεις ο αυτόματα δημιουργημένος παράλληλος κώδικας έχει αρκετά καλύτερο χρόνο εκτέλεσης από τον σειριακό και σε κάποιες περιπτώσεις πολύ κοντά στον χρόνο εκτέλεσης του hand-coded παράλληλου κώδικα.

Υπάρχουν άλλες περιπτώσεις όμως που είτε λόγο μικρού dataset όπου η αλλαγή σε παράλληλο δεν προσφέρει ιδιαίτερο κέρδος σε χρόνο και σε κάποιες άλλες περιπτώσεις η σύγκριση τείνει υπέρ του hand-coded παράλληλου κώδικα.

Κεφάλαιο 2: Τεχνικό Υπόβαθρο

2.1 Why go parallel

Δεδομένων των νέων τεχνολογιών που υπάρχουν στον τομέα της Αρχιτεκτονικής Υπολογιστών πρέπει να προσπαθήσουμε να τις εκμεταλλευτούμε δημιουργώντας προγράμματα που να αξιοποιούν περισσότερους από ένα πυρήνα.

Σκοπός λοιπόν είναι με την αξιοποίηση της εξελισσόμενης τεχνολογίας της Αρχιτεκτονικής υπολογιστών να αυξήσουμε την απόδοση των εφαρμογών μας. Με την χρήση περισσότερων από ένα πυρήνες μπορούμε να έχουμε μία θετική αύξηση στο Speedup

2.1.1 Speedup

Με τον όρο Speedup[7] εννοούμε την αύξηση (ή μείωση) της απόδοσης ενός προγράμματος με βάση τις αλλαγές που εφαρμόσαμε. Για να συγκρίνουμε το πόσο Speedup κερδίζουμε με την μετατροπή ενός σειριακού προγράμματος σε παράλληλο πρέπει να χρησιμοποιήσουμε τη συνάρτηση

Speedup=Χρόνος εκτέλεσης σειριακού προγράμματος / Χρόνος εκτέλεσης παράλληλου προγράμματος.

Μεγαλύτερο Speedup = περισσότερο Κέρδος.

2.1.2 Actual Speedup

Σε θεωρητικό επίπεδο χρησιμοποιώντας 2 επεξεργαστές αυτομάτως ο χρόνος εκτέλεσης μειώνεται στα δύο. Συνεπώς όσο αυξάνουμε τους πυρήνες/επεξεργαστές τόσο περισσότερο χρόνο κερδίζουμε. Στην πραγματικότητα όμως η αύξηση των πυρήνων συνήθως έχει την μορφή που παρουσιάζετε στην πιο κάτω γραφική παράσταση.

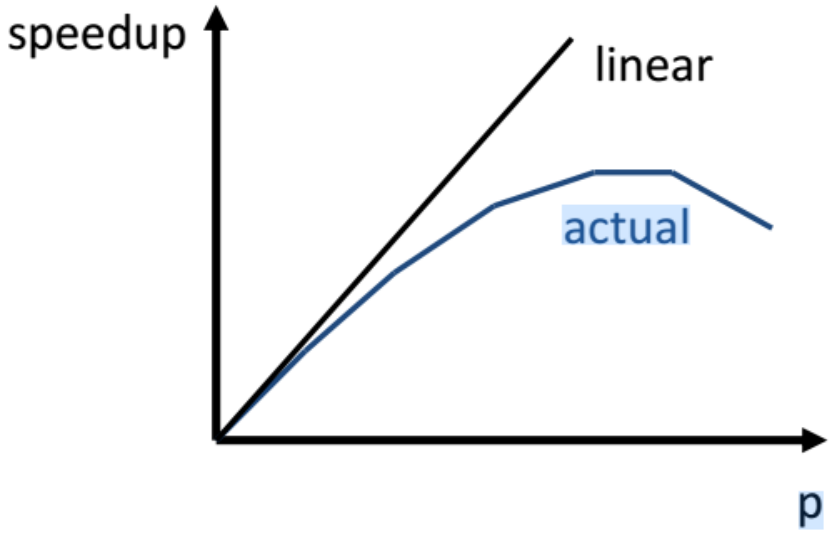


Figure 1: Sub-Linear vs Linear Speedup

2.1 Τι οδήγησε στον παράλληλο προγραμματισμό

Η εξέλιξη της αρχιτεκτονικής μικροεπεξεργαστών εξαρτάται από τις μεταβαλλόμενες πτυχές της τεχνολογίας. Η αύξηση της πυκνότητας και της ταχύτητας των transistors ανά επεξεργαστή, η μνήμη και η συμπεριφορά προγράμματος γίνονται όλο και περισσότερο σημαντικά στην βιομηχανία αρχιτεκτονικής Η/Υ. Ενώ η τεχνολογία επιτρέπει περισσότερο σύνθετες σχεδιάσεις επεξεργαστών, υπάρχουν φυσικά και προγραμματιστικά όρια στη χρησιμότητα αυτής της πολυπλοκότητας. Τα φυσικά όρια περιλαμβάνουν τα όρια συσκευών/επεξεργαστών καθώς επίσης και τα πρακτικά όρια που αφορούν τη δύναμη και το κόστος. Τα όρια συμπεριφοράς προγράμματος προκύπτουν από απρόβλεπτα γεγονότα που εμφανίζονται κατά τη διάρκεια μιας εκτέλεσης. Οι αρχιτεκτονικές και οι εφαρμογές που επεκτείνουν αυτά τα όρια είναι ζωτικής σημασίας στη συνεχή εξέλιξη των επεξεργαστών.

2.2 Προσεγγίσεις παραλληλισμού

Υπάρχουν δύο κύριες προσεγγίσεις για να παραλληλίσουμε τις εφαρμογές [8,9]: ο παράλληλος προγραμματισμός και ο αυτόματος παραλληλισμός τα οποία διαφέρουν από την άποψη του αν θέλει κάποιος να πετύχει το μέγιστο βαθμό επίδοσης (maximum speedup) της εφαρμογής του ή αν θέλει να ευνοηθεί της ευκολίας του παραλληλισμού αντίστοιχα.

Η προσέγγιση του αυτόματου παραλληλισμού ή οι παράλληλοι μεταγλωττιστές, παραλληλίζουν αυτόματα τις εφαρμογές οι οποίες έχουν αναπτυχθεί χρησιμοποιώντας σειριακά πρότυπα προγραμματισμού. Το πλεονέκτημα αυτής της προσέγγισης είναι ότι οι ήδη υπάρχουσες εφαρμογές δεν χρειάζονται να τροποποιηθούν, π.χ. οι εφαρμογές πρέπει απλά να μεταφραστούν με έναν παράλληλο μεταγλωττιστή. Επομένως, οι προγραμματιστές δεν χρειάζεται να μάθουν τα νέα πρότυπα του παράλληλου προγραμματισμού. Εντούτοις, αυτό γίνεται και ένας περιοριστικός παράγοντας στην εκμετάλλευση ενός υψηλότερου βαθμού παραλληλισμού: είναι μια εξαιρετικά μεγάλη πρόκληση ο αυτόματος μετασχηματισμός των αλγόριθμων από τη σειριακή τους φύση στην παράλληλη.

Σε αντίθεση με τον αυτόματο παραλληλισμό, στην παράλληλη προσέγγιση προγραμματισμού, οι εφαρμογές αναπτύσσονται συγκεκριμένα για να εκμεταλλευτούν όσο το δυνατόν σε μεγαλύτερο βαθμό τον παραλληλισμό με σκοπό να φτάσουν την μέγιστη απόδοση που μπορούν να πετύχουν στη συγκεκριμένη αρχιτεκτονική που θα τρέξουν. Γενικά, η ανάπτυξη μιας παράλληλης εφαρμογής περιλαμβάνει καταμερισμό του φόρτου εργασίας σε τμήματα και ανάθεση των τμημάτων αυτών προς εκτέλεση. Γίνεται αντιληπτό ότι ο παράλληλος προγραμματισμός οδηγεί σε αποτελέσματα υψηλότερης επίδοσης σε σχέση με τον αυτόματο παραλληλισμό, αλλά με την προϋπόθεση ότι ο προγραμματιστής θα πρέπει να καταβάλει περισσότερο κόπο και προσπάθεια για να πετύχει το στόχο του.

Τόσο στον αυτόματο παραλληλισμό όσο και στην παράλληλη προσέγγιση προγραμματισμού η διαδικασία που πρέπει να ακολουθηθεί για να παραλληλιστεί μια εφαρμογή είναι κοινή και για τις δυο περιπτώσεις. Σε γενικές γραμμές η όλη διαδικασία μπορεί να χωριστεί σε 4 βήματα [figure 2] που είναι τα εξής:

- **Διάσπαση – Decomposition** Προσδιορισμός του τι μπορεί να εκτελεστεί παράλληλα και διάσπαση του σειριακού προγράμματος/υπολογισμού σε μικρότερες εργασίες/στόχους

- **Ανάθεση – Assignment** Καταμερισμός των εργασιών/στόχων που δημιουργήθηκαν από το προηγούμενο βήμα στις διάφορες μονάδες εκτέλεσης

- **Ενορχήστρωση -Orchestration** Διαχείριση της πρόσβασης στα δεδομένα, της επικοινωνίας και του συγχρονισμού μεταξύ των μονάδων εργασίας

- **Δέσμευση – Mapping** Δέσμευση των διαδικασιών και των μονάδων εκτέλεσης στους διαθέσιμους επεξεργαστές

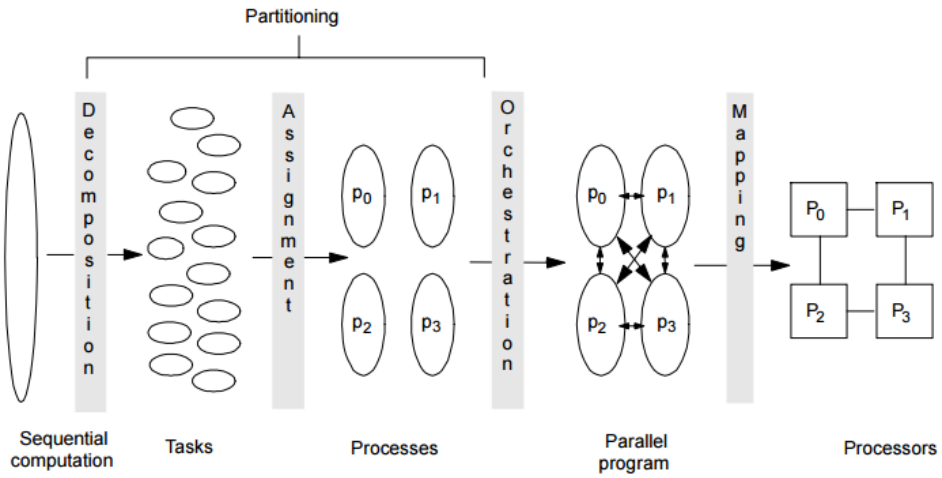


Figure 2: Steps in creating a Parallel Program[43]

2.3 Εργαλεία για αυτόματη παραλληλοποίηση κώδικα

Οι περισσότεροι από τους σημερινούς αυτόματους μεταφραστές για παράλληλα συστήματα κάνουν source to source μετάφραση χρησιμοποιώντας οδηγίες OpenMP ή MPI ενώ μερικοί άλλοι μεταφράζουν άμεσα σε κώδικα μηχανών. Αυτό μπορεί να γίνει είτε εντελώς αυτόματα, είτε με τη βοήθεια του προγραμματιστή χρησιμοποιώντας ένα γραφικό εργαλείο περιβάλλοντος (GUI). Η πιο συνηθισμένη αυτόματα παραλληλοποιήσιμη γλώσσα προγραμματισμού είναι η Fortran και αυτό οφείλεται κυρίως στην έλλειψη δεικτών.

Πιο κάτω παρουσιάζονται εν συντομία οι πιο γνωστοί αυτόματοι μεταφραστές για παράλληλα συστήματα και ο κάθε ένας θα αναλυθεί σε λεπτομέρεια σε δικό του κεφάλαιο.

2.3.1 Intel Parallel Studio

Το Intel Parallel Studio [10,11,12]είναι ένα προϊόν ανάπτυξης λογισμικού που αναπτύχθηκε από την Intel και διευκολύνει την φυσική ανάπτυξη κώδικα για τα Windows και τα Linux σε C++, C και Fortran για Parallel Computing. Το Parallel Studio αποτελείται από πολλά συστατικά μέρη, καθένα από τα οποία είναι μια συλλογή από δυνατότητες, αλλά οι συνθέσεις που μας ενδιαφέρουν είναι ο μεταγλωττιστής για Intel C++ και Intel Fortran.[13]

Οι μεταγλωττιστές της Intel μπορούν να δημιουργήσουν αυτόματα multithreaded εκδοχή για εφαρμογές όπου το μεγαλύτερο μέρος του υπολογισμού γίνεται σε απλά loops. Γενικά οι Intel C++ και Intel Fortran μεταγλωττιστές έχουν την δυνατότητα να αναλύουν τα dataflow σε βρόγχους, ώστε να καθορίσουν ποιοι βρόγχοι μπορούν με ασφάλεια και αποτελεσματικότητα να εκτελεστούν αυτόματα. Στα πολυπύρηνα συστήματα, η αυτόματη παραλληλοποιήση μπορεί μερικές φορές να οδηγήσει σε μικρότερους χρόνους εκτέλεσης.

Και τέλος, ανακουφίζει τους προγραμματιστές από την αναζήτηση βρόγχων που είναι καλοί για παράλληλη εκτέλεση και προσθέτει manually οδηγίες για τον παράλληλο μεταγλωττιστή.

2.3.2 SUIF Compiler

Το σύστημα μεταγλωττιστών SUIF, που έχει αναπτυχθεί στο Πανεπιστήμιο του Stanford, υποστήριξε την πειραματική έρευνα για τις πρώτες νέες τεχνικές παράλληλων μεταγλωττιστών. [14,15,16]Αποτελείται από τα διάφορα εργαλεία με τυποποιημένα περάσματα βελτιστοποίησης, τα οποία λειτουργούν πάνω από ένα κοινό πλαίσιο (framework). Υπάρχουν δύο σημαντικές εκδόσεις του SUIF, ο SUIF 1.0 και μια αναθεωρημένη έκδοση του, ο SUIF 2.0. Ο SUIF 2.0 μοιράζεται τις βασικές αρχές και λειτουργίες του SUIF 1.0, αλλά ο σχεδιασμός και ο τρόπος εφαρμογής του είναι εντελώς διαφορετικά. Και οι δύο εκδοχές είναι παίρνουν σαν είσοδο γλώσσα C και Fortran. Επιπλέον, ο SUIF 2.0 έχει ένα καθαρότερο σχέδιο και μια πιο συγυρισμένη δομή.

2.3.3 PLUTO

Το PLUTO [17] είναι ένα αυτόματο εργαλείο παραλληλισμού με βάση το πολυεδρικό μοντέλο. Το πολυεδρικό μοντέλο για τη βελτιστοποίηση του μεταγλωττιστή είναι μια αναπαράσταση για τα προγράμματα που το καθιστά κατάλληλο για την εκτέλεση μετασχηματισμών υψηλού επιπέδου, όπως βελτιστοποιήσεις των βρόγχων και παραλληλισμό των βρόχων. Το PLUTO μετατρέπει τα προγράμματα C από πηγή σε πηγή για παραλληλισμό και ταυτόχρονη τοποθεσία των δεδομένων. Το βασικό πλαίσιο μετασχηματισμού λειτουργεί κυρίως με την εύρεση συσχετισμένων μετασχηματισμών για αποτελεσματική συνένωση, αλλά δεν περιορίζονται σε αυτές. Ο OpenMP παράλληλος κώδικας για πολυπυρήνες μπορεί να παραχθεί αυτόματα από σειριακά τμήματα του προγράμματος C.

2.3.4 Par4all

Το Par4All [18]είναι ένα workbench για αυτόματη παραλληλοποίηση και βελτιστοποίηση από τον μεταγλωττιστή σε σειριακά προγράμματα C και Fortran. Ο σκοπός αυτού του source-to-source μεταγλωττιστή είναι η προσαρμογή των υφιστάμενων εφαρμογών σε διάφορα hardware, όπως σε συστήματα πολλαπλών πυρήνων, υπολογιστές υψηλών επιδόσεων και GPUs. Δημιουργεί ένα νέο πηγαίο κώδικα και έτσι επιτρέπει στο αρχικό πηγαίο κώδικα της εφαρμογής να παραμείνει αμετάβλητος.

2.3.5 GCC

Ο GCC είναι ένα σύστημα compiler που παράγεται από το GNU Project και υποστηρίζει διάφορες γλώσσες προγραμματισμού. Ο GCC είναι ένα βασικό συστατικό του GNU toolchain. Το Ίδρυμα Ελεύθερου Λογισμικού (FSF) προσφέρει το GCC κάτω από το GNU General Public License (GNU GPL). Ο GCC έχει διαδραματίσει σημαντικό ρόλο στην ανάπτυξη του ελεύθερου λογισμικού, τόσο ως εργαλείο όσο και ώς παράδειγμα. Αρχικά το όνομα του ήταν GNU C Compiler, αφού χειριζόταν μόνο τη γλώσσα προγραμματισμού C. όταν κυκλοφόρησε το 1987. [19] Επεκτάθηκε για να υποστηρίζει τη C ++, τον Δεκέμβριο του ίδιου έτους και αργότερα επεκτάθηκε για να υποστηρίζει μεταξύ άλλων Objective-C, Objective-C ++, Fortran, Java, Ada, και Go. [20]

2.3.6 Polaris

Ο compiler Polaris είναι ένας source to source compiler που παίρνει ένα πρόγραμμα FORTRAN77 ως input, μετατρέπει αυτό το πρόγραμμα έτσι ώστε να εκτελείται αποτελεσματικά σε ένα παράλληλο υπολογιστή, και εξάγει αυτήν την έκδοση του προγράμματος σε μία από τις πολλές πιθανές παράλληλες διαλέκτους της Fortran. Η input γλώσσα περιλαμβάνει διάφορες οδηγίες που επιτρέπουν στο χρήστη του Polaris για να καθορίσει τον παραλληλισμό ρητά στον πηγαίο κώδικα. Η γλώσσα output του Polaris είναι συνήθως με τη μορφή της FORTRAN77 μαζί με parallel directives. Για παράδειγμα, ένα γενικό σύνολο parallel directives περιλαμβάνει τις οδηγίες "CSRD $ Parallel" και "CSRD $ Private a, b", διευκρινίζοντας ότι οι επαναλήψεις του loop, που ακολουθεί των εντολών, θα εκτελείται παράλληλα και ότι οι μεταβλητές A και B χαρακτηρίζονται ως «ιδιωτικές στο τρέχων loop», αντίστοιχα. Ο Polaris εκτελεί τους μετασχηματισμούς του σε διάφορα

περάσματα από τον κώδικα. Εκτός από τα πολλά στάδια που έχουν άλλοι ο compilers, ο Polaris περιλαμβάνει προηγμένες δυνατότητες που εκτελεί τις ακόλουθες εργασίες:

- Array privatization
- Data dependence testing
- Induction variable recognition
- Interprocedural analysis
- Symbolic program analysis.

2.4 Επισκόπηση των επόμενων κεφαλαίων

Στα υπόλοιπα κεφάλαια της διπλωματικής αυτής θα γίνει μία πιο αναλυτική περιγραφή των εργαλείων που έχουν αναφερθεί πιο πάνω, των πλεονεκτημάτων που προσφέρουν καθώς και μία ανάλυση πειραμάτων που έχω εκτελέσει με τον GCC.

Κεφάλαιο 3: Polaris

3.1 Λεπτομέρειες για τον Polaris

Ο Polaris compiler[24] παίρνει σαν είσοδο ένα πρόγραμμα FORTRAN77 και το μετατρέπει ώστε να τρέχει αποτελεσματικά σε μια παράλληλη μηχανή. Σαν έξοδο εξάγει αυτήν την έκδοση του προγράμματος σε μια από τι πολλές παράλληλες Fortran διαλέκτους.

Δεν είναι automatic parallelization compiler με την κλασσική έννοια καθώς η έξοδος του δεν είναι ένα εκτελέσιμο αρχείο που τρέχει σε παράλληλους επεξεργαστές αλλά η έξοδος του είναι παράλληλος κώδικας. Ο προγραμματιστής αναγνωρίζοντας ότι κάποια σημεία του κώδικα του χρειάζονται περισσότερο χρόνο για να ολοκληρωθούν, μπορεί να δώσει κάποια directives στον Polaris για να τα μετατρέψει σε παράλληλο κώδικα.

Η γλώσσα εισαγωγής περιλαμβάνει διάφορες οδηγίες (directives) που επιτρέπουν στον χρήστη του Polaris να καθορίσει ρητά τον παραλληλισμό μέσα στον πηγαίο κώδικα. Αλλά μπορεί να δώσει και εντολές την στιγμή της εκτέλεσης του compilation για να αναζητήσει μόνος του ο compiler για σημεία τα οποία μπορούν να παραλληλοποιηθούν. Η γλώσσα που βγάζει σαν έξοδο το Polaris είναι συνήθως με την μορφή του FORTRAN77 συν κάποιες παράλληλες οδηγίες.

Για παράδειγμα, ένα γενικό σύνολο από παράλληλες οδηγίες περιλαμβάνει την οδηγία «CSRD$ PARALLEL» και «CSRD$ PRIVATE a,b», διευκρινίζει ότι οι επαναλήψεις των μεταγενέστερων βρόγχων θα εκτελούνται ταυτόχρονα και ότι οι μεταβλητές a και b θα δηλώνονται private στο τρέχον βρόγχο.

Άλλη γλώσσα που βγάζει σαν έξοδο ο Polaris είναι η Fortran με οδηγίες που είναι διαθέσιμες στην μηχανή SG Challenge.

Ο Polaris εκτελεί τους μετασχηματισμούς του σε διάφορα περάσματα από τον κώδικα. Ο Polaris περιλαμβάνει προηγμένες δυνατότητες στο να εκτελείς τα ακόλουθα parallelization tasks : ιδιωτικοποίηση πινάκων, έλεγχο για εξάρτηση μεταξύ των δεδομένων induction variable recognition και συμβολική ανάλυση του προγράμματος.

Ένα εκτεταμένο σύνολο από επιλογές, επιτρέπουν στον χρήστη και τον προγραμματιστή του Polaris να πειραματιστούν με το εργαλείο με ευέλικτο τρόπο. Η εφαρμογή του Polaris αποτελείται από 170.000 γραμμές C++ κώδικα. Μια βασική υποδομή που παρέχει μια ιεραρχία από C++ κλάσεις, που οι προγραμματιστές των διαφόρων περασμάτων, κατά την διάρκεια του compilation, μπορούν να χρησιμοποιήσουν για να χειριστούν και να αναλύσουν τον input κώδικα.

3.2 Benchmark Results

Έχει εκτελεστεί ένα σύνολο από benchmark προγράμματα[24] (σε λειτουργία πραγματικού χρόνου για την ακρίβεια της χρονομέτρησης) σε 8 επεξεργαστές της μηχανής SGI Challenge με 150 MHz R4400 επεξεργαστές στο Εθνικό κέντρο για εφαρμογές υπερυπολογιστών. (NCSA).

Έτρεξαν κάθε κώδικα σειριακά και κατέγραψαν τους χρόνους εκτέλεσης. Μετά έτρεξαν αυτούς τους κώδικες παράλληλα, αφού τους έκαναν παράλληλους με το Silicon Graphics Power Fortran Analyzer. Και στο τέλος τους έτρεξαν παράλληλα, αφού τους μετέτρεψαν με το Polaris.

3.2.1 Brief Benchmarks Description

Program Name	Brief Description	Benchmark Suite	Lines of Code
ARC2D	Fluid dynamics (2D inviscid flows)	Perfect	4000
BDNA	Nucleic acid simulation	Perfect	4000
FLO52	Fluid dynamics (2D inviscid flow)	Perfect	2368
MDG	Liquid water simulation	Perfect	1200
OCEAN	Fluid dynamics (2D Boussinesq fluid layer)	Perfect	3285
TRFD	Quantum mechanics	Perfect	634
TURB3D	3D turbulent fluid flow	Perfect	1400
HYDRO2D	Galactical jets simulation	SPEC	4289
SU2COR	Elementary particle mass computation	SPEC	2333
SWIM	Shallow water equations	SPEC	426
TFFT2	Fourier transforms	SPEC	642
TOMCATV	Mesh generator	SPEC	190
CLOUD3D	Weather simulation,	NCSA	14438
CMHOG	Astrophysical simulation	NCSA	11826

3.2.2 General Speedup Benefits

Η αύξηση της ταχύτητας για κάθε κώδικα σε σύγκριση με την σειριακή ταχύτητα, που παράχθηκε από κάθε compiler, φαίνεται στο πιο κάτω διάγραμμα :

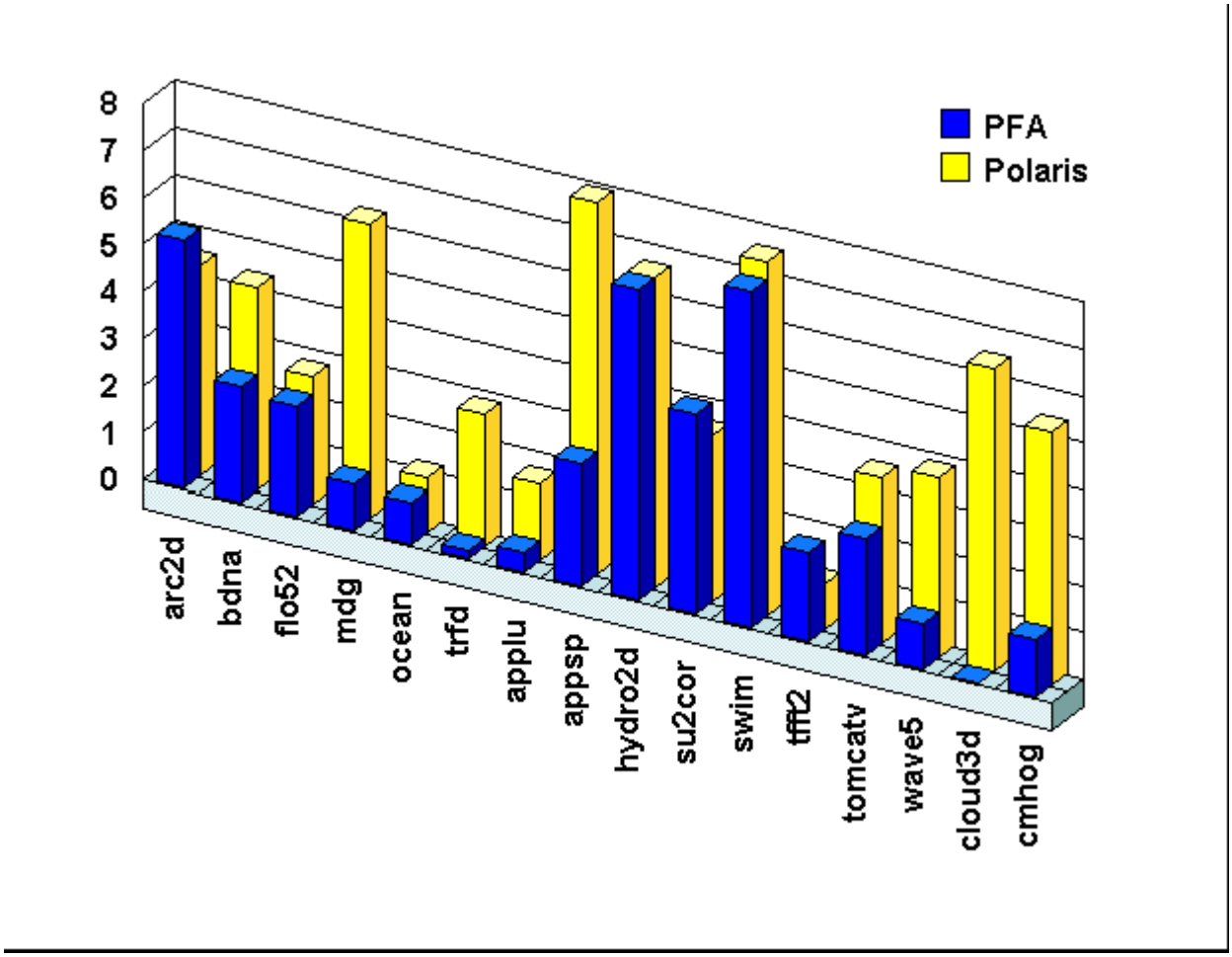


Figure 3: Comparison of PFA and Solaris speedup benefits[24]

3.2.3 Συμπεράσματα

Σε γενικές γραμμές το speedup που έχει επιτευχθεί με την χρήση του Polaris compiler είναι αρκετά καλό, χωρίς όμως να είναι εξωπραγματικό.

3.3 Parallelization techniques comparison

Στη συνέχεια, σε μια προσπάθεια να καθοριστεί η σημαντικότητα των έξι από τις παράλληλες τεχνικές που χρησιμοποιούνται στο Polaris[24], μεταγλώττισαν τον κάθε κώδικα ακόμα 6 φορές. Απενεργοποιούσαν μια από τις έξι τεχνικές κάθε φορά και μετρούσαν τί ποσοστό των βρόγχων στον κώδικα ήταν σειριακοί. Τα αποτελέσματα φαίνονται στο παρακάτω διάγραμμα.

3.3.1 Αποτελέσματα

Ο κάθετος άξονας αναπαριστά το ποσοστό των βρόγχων που ήταν σειριακοί από την απενεργοποίηση μιας δεδομένης τεχνικής.

Οι έξι τεχνικές ήταν:

- Advanced induction variable analysis
- Automatic inlining of subroutines
- Interprocedural value propagation (IPVP)
- Array privatization
- A dependence test called the Range Test
- Advanced reduction analysis.

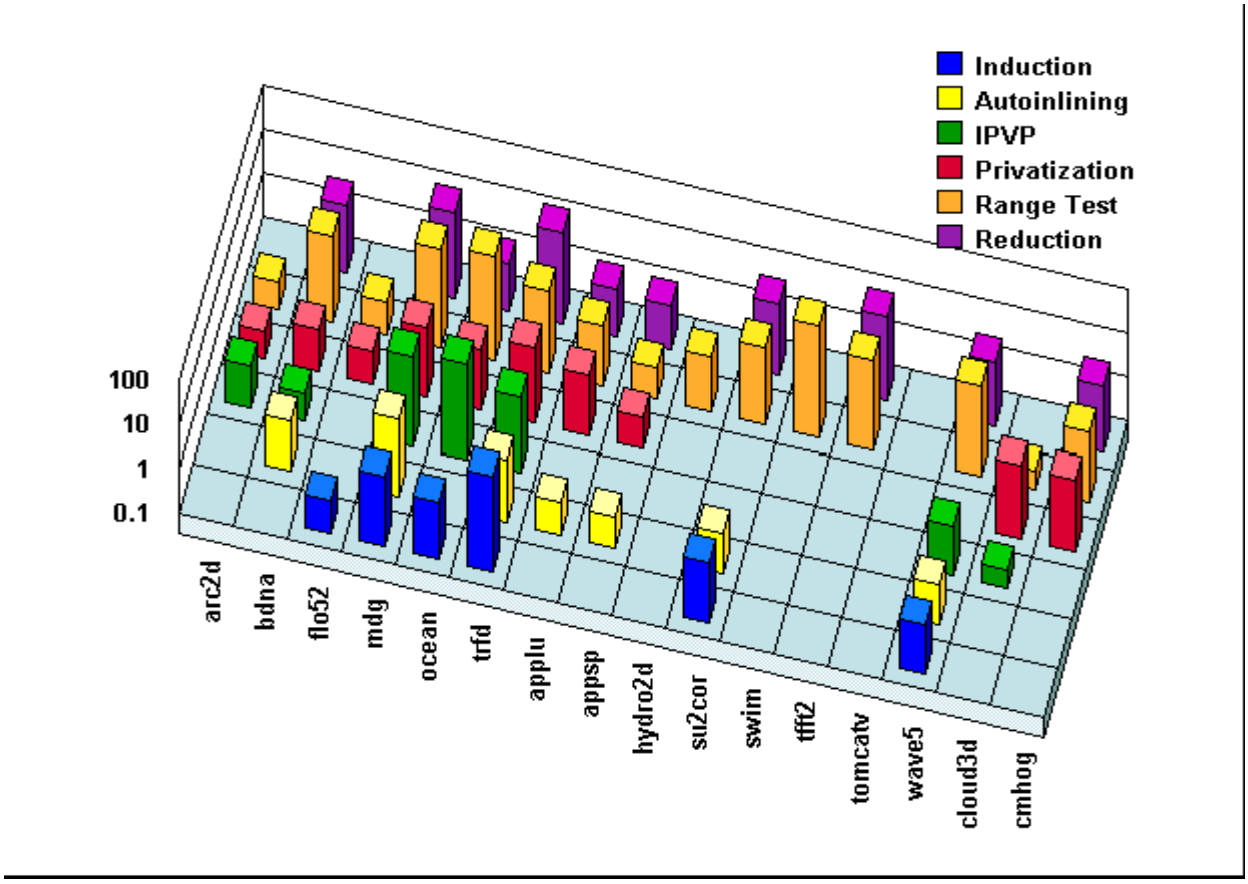


Figure 4: Comparison of the benefits of different parallelization techniques [24]

3.3.2 Συμπεράσματα

Βλέπουμε ότι αν αφαιρέσουμε την τεχνική Advanced Reduction Analysis και την τεχνική Range test τότε μένουν τα περισσότερα loops σειριακά, κάτι που δείχνει την σημαντικότητα των συγκεκριμένων τεχνικών

Κεφάλαιο 4: Par4All

4.1 Λεπτομέρειες του Par4All

Το Par4All[18] είναι ένας compiler αυτόματης παραλληλοποίησης και βελτιστοποίησης για σειριακά προγράμματα στην γλώσσα C και Fortran.

Ο σκοπός αυτού του source-to-source compiler είναι να προσαρμοστούν οι υπάρχουσες εφαρμογές για διάφορα hardware, όπως multicore συστήματα, υψηλής απόδοσης υπολογιστές και GPUs ή κάποια παράλληλα embedded ετερογενή συστήματα. Δημιουργεί νέους πηγαίους κώδικες χρησιμοποιώντας OpenMP, CUDA ή OpenCL και αφήνει τους αρχικούς πηγαίους κώδικες να παραμείνουν σχεδόν αμετάβλητοι για καλά δομημένα προγράμματα.

Ο Par4All μπορεί να είναι source-to-source compiler αλλά μπορεί να δημιουργήσει και εκτελέσιμο αρχείο καλώντας τον gcc στον κώδικα που δημιούργησε.

Το Par4All είναι ένα project ανοικτού πηγαίου κώδικα που συνδυάζει διάφορες εξελίξεις ανοιχτού κώδικα. Το Par4All δεν υποστηρίζεται πλέον από την ομάδα που το δημιούργησε, αλλά μπορεί όποιος ενδιαφέρεται να το κατεβάσει και να το χρησιμοποιήσει.

4.2 Stars-PM

4.2.1 Περιγραφή

Το Stars-PM είναι μία κοσμολογική προσομοίωση γραμμένη σε C κώδικα γραμμένη από το Αστρονομικό παρατηρητήριο του Στρασβούργου[21]. Είναι μία N-Body-Particle-Mesh προσομοίωση που μεταξύ άλλων χρησιμοποιεί και τον μετασχηματισμό Fourier σε τρεις διαστάσεις.

4.2.2 Αποτελέσματα

Η σειριακή έκδοση του Stars-PM γράφτηκε σε κώδικα C και με την βοήθεια του Par4All δημιουργήθηκε παράλληλος πηγαίος κώδικας ο οποίος συγκρίθηκε με το αρχικό σε χρόνο εκτέλεσης[25]. Το πρόγραμμα εκτελέστηκε σε επεξεργαστή με 12 πυρήνες και στην nVidia Tesla C2050 GPU.

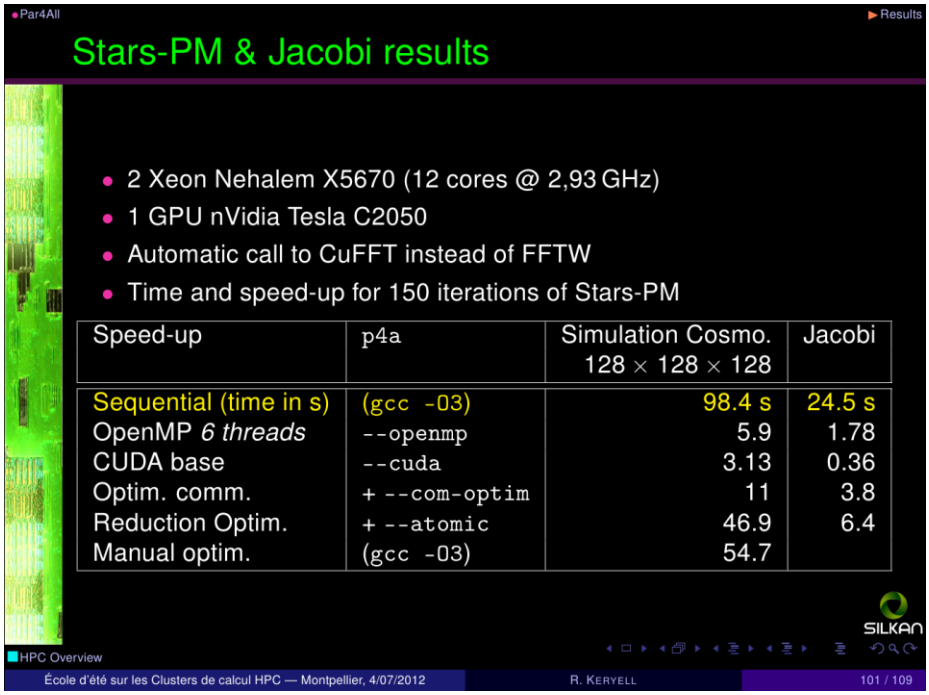


Figure 5: Par4All Benchmark Results [29]

4.3 Hyantes

4.3.1 Περιγραφή

Το Hyantes[22] είναι μια βιβλιοθήκη για τον υπολογισμό τον πιθανό μελλοντικό πληθυσμό μιας γειτονιάς με έλεγχο κλίμακας στις προσομοιώσεις. Έχει αναπτυχθεί από την ομάδα Mescal από το Laboratoire Informatique de Grenoble (Γαλλία), ως μέρος του σχεδίου HyperCarte. Το έργο HyperCarte[23] στοχεύει στην ανάπτυξη νέων μεθόδων για τη χαρτογραφική απεικόνιση της διανομής των ανθρώπων (πυκνότητα του πληθυσμού, αύξηση του πληθυσμού, κ.λπ.) με διάφορες λειτουργίες smoothing και ευκαιρίες για χρονική κλίμακα animations των χαρτών. Το Hyantes παρέχει μία από τις μεθόδους smoothing που σχετίζονται με multiscalar εκτίμησης της πυκνότητας μιας γειτονιά. Πρόκειται για μια βιβλιοθήκη C που παίρνει σύνολα γεωγραφικά δεδομένα ως είσοδο, και υπολογίζει μια ομαλοποιημένη εκπροσώπηση των εν λόγω δεδομένων, λαμβάνοντας υπόψη την επιρροή της γειτονιάς.

4.3.2 Αποτελέσματα

Χρονομετρήθηκε ο συνολικός χρόνος του προγράμματος[29,41] (χρόνος εκκίνησης, χρόνο φόρτωσης δεδομένων, χρόνος πυρήνα, χρόνος output δεδομένων) διότι αυτός είναι ο πραγματικός χρόνος που βλέπει ο χρήστης. Σε ένα υπολογιστή με δύο επεξεργαστές Intel Xeon X5670 @ 2.93GHZ (σύνολο 12 πυρήνες) και μία nVidia Tesla C2050 με Linux Debian χρονομετρήθηκε το πρόγραμμα με τα εξής αποτελέσματα:

Σειριακός χρόνος εκτέλεσης: 30.355s

Παράλληλος με OpenMP χρόνος εκτέλεσης: 3.859s, δηλαδή 7,87 speed-up

Παράλληλος με CUDA χρόνος εκτέλεσης στην GPU: 0.441s, δηλαδή 68.8 speed-up

4.4 Συμπέρασμα

Έχει αποδειχθεί λοιπόν ότι στα πειράματα με τα οποία έχει αξιολογηθεί ο Par4All ότι ο αυτόματος παραλληλισμός που προσφέρει μπορεί να δώσει αρκετό speed-up σε σχέση με το σειριακό πρόγραμμα κυρίως όταν δημιουργεί κώδικα που θα τρέξει σε GPU.

Κεφάλαιο 5: SUIF

5.1 Λεπτομέρειες του SUIF

Ο SUIF compiler [40] μεταφράζει σειριακά προγράμματα σε παράλληλο κώδικα για shared address space machines. Ο compiler δημιουργεί ένα ενιαίο πρόγραμμα, ένα πρόγραμμα πολλαπλών δεδομένων (SPMD) που περιέχει κλήσεις σε μια portable run-time βιβλιοθήκη. Μέχρι στιγμής υπάρχουν εκδόσεις της βιβλιοθήκης αυτής που για μηχανές SGI, για τον πολυεπεξεργαστή DASH multiprocessor και για την μηχανή Kendall Square Research KSR1.

Στη συνέχεια χρησιμοποιούμε τον SUIF-to-C μεταφραστή για να μετατρέψει τον SUIF κώδικα σε ANSI C. Αυτή η μετάφραση μας δίνει τη δυνατότητα να εφαρμόσουμε υψηλού επιπέδου μετασχηματισμούς και τεχνικές παραλληλισμού σε μηχανές για τις οποίες δεν μπορούμε να παράγουμε άμεσα τον τελικό κώδικα μηχανής.

Ο SUIF αποτελείται από πολλά διαφορετικά περάσματα compiler.

Πρώτον, ένας αριθμός από scalar βελτιστοποιήσεις βοηθούν στο να βγάζουν προς τα έξω τον παραλληλισμό. Αυτά περιλαμβάνουν constant propagation, forward propagation, induction variable detection, constant folding και την scalar ανάλυση των ιδιωτικοποιήσεων. Στη συνέχεια, unimodular μετατροπές των βρόγχων που καθοδηγούνται από την ανάλυση της εξαρτήσεων των πινάκων, αναδιαρθρώνουν τον κώδικα για τη βελτιστοποίηση τόσο για τον παραλληλισμό όσο και για την τοπικότητα (locality). Τέλος, ο generator παράλληλου κώδικα παράγει ένα παράλληλο κώδικα με κλήσεις στην παράλληλη run-time βιβλιοθήκη.

5.2 Experimental Evaluation

Διεξήγαγαν μια σειρά από αξιολογήσεις απόδοσης[39] για να αποδείξουν την επίδραση των αναλύσεων και βελτιστοποιήσεων του SUIF. Πήραν μετρήσεις από τον Digital AlphaServer 8400 με οκτώ 21164 επεξεργαστές, κάθε ο καθένας με δύο επίπεδα on-chip cache μνήμης και μιας 4-Mbyte cache εξωτερική μνήμης. Επειδή τα speed-ups είναι δύσκολο να επιτευχθούν σε μηχανήματα με γρήγορους επεξεργαστές, η χρήση που έκαναν σε μηχανές τελευταίας τεχνολογίας κάνει τα αποτελέσματα να είναι πιο ουσιαστικά και εφαρμόσιμα για τα μελλοντικά συστήματα.

Χρησιμοποιήσαν δύο ολοκληρωμένα σετ από benchmarks για την αξιολόγηση του compiler. Πιο κάτω παρουσιάζονται τα αποτελέσματα για τα 10 προγράμματα των SPECfp95 benchmarks, που χρησιμοποιούνται συνήθως για benchmarking πολυεπεξεργαστές. Επίσης χρησιμοποιήθηκαν τα οκτώ επίσημα benchmarks από το NAS parallel-system, εκτός από το embar. Εδώ χρησιμοποιήσαν μια ελαφρώς τροποποιημένη έκδοση από την έρευνα Applied Parallel.

Το πάρακατω σχήμα δείχνει τα speed ups των SPECfp95 και NAS, που μετρήθηκαν σε μέχρι οκτώ επεξεργαστές σε ένα AlphaServer 300 MHz. Υπολόγισαν τα speed ups σε σχέση με τον καλύτερο διαδοχικό χρόνο εκτέλεσης είτε από επίσημα δημοσιευμένα αποτελέσματα ή τις δικές τους μετρήσεις. Σημειώστε ότι τα projects mgrid και aplu εμφανίζονται και στα δύο σετ από benchmarks. Ο πηγαίος κώδικας και τα μεγέθη δεδομένων διαφέρουν ελαφρώς.

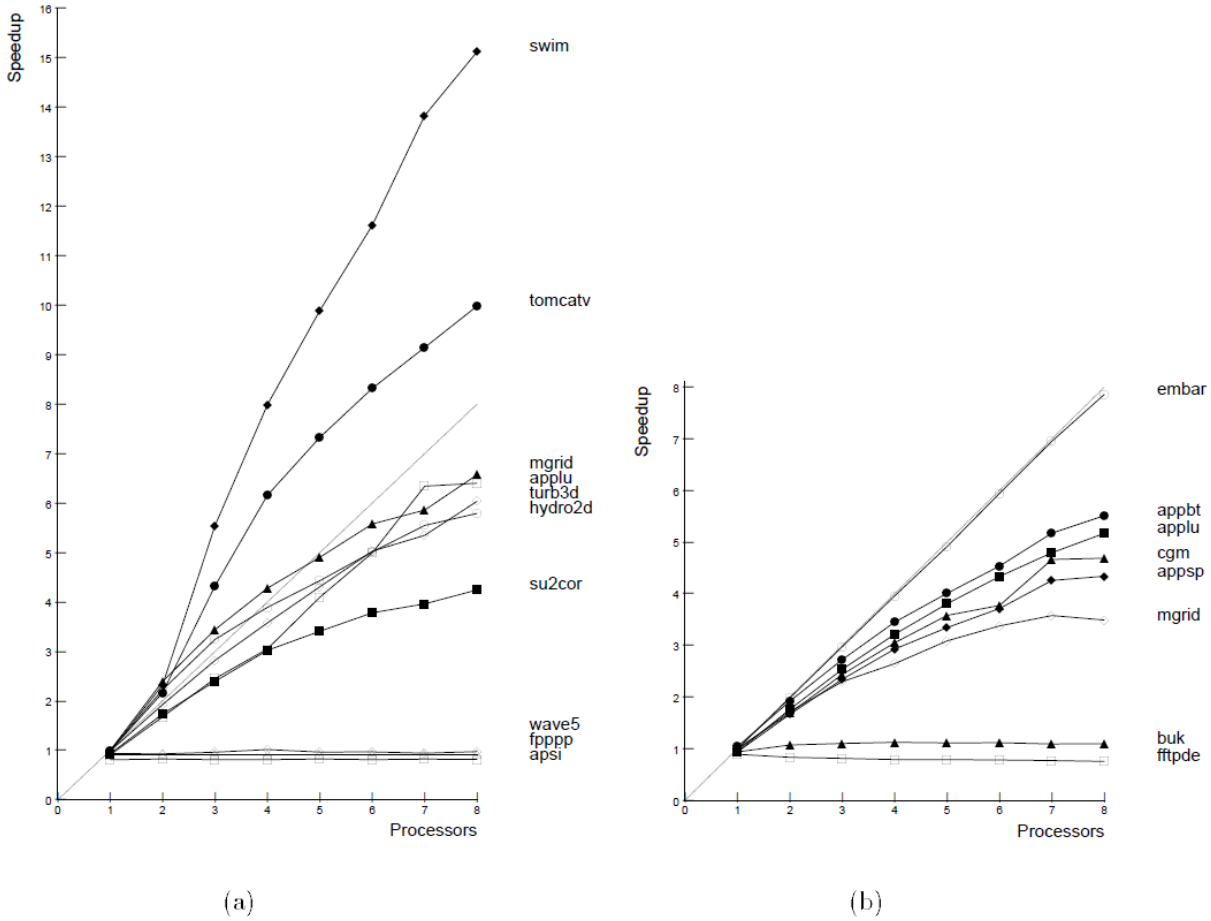


Figure 6: Speedups using SUIF[39]

Για τη μέτρηση των αποτελεσμάτων των διαφόρων τεχνικών μεταγλώττισης, έσπασε σε κομμάτια, σε τρεις συνιστώσες (components), η απόδοση που γίνεται από οκτώ επεξεργαστές. Στο figure 7 , η αρχική τιμή (baseline) δείχνει την επιτάχυνση που γίνεται με παραλληλοποίηση χρησιμοποιώντας μόνο ενδο-διαδικαστική ανάλυση εξάρτησης δεδομένων, βαθμωτή ιδιωτικοποίηση(scalar privatization), και βαθμωτή μείωση μετασχηματισμών(scalar reduction transformations) . To Coarse grain paralelism περιλαμβάνει τις τεχνικές του baseline, καθώς και τεχνικές για τον εντοπισμό coarse grain παράλληλων βρόγχων. Για παράδειγμα, η ιδιωτικοποίηση πινάκων και η μείωση των μετασχηματισμών, και η πλήρη ανάλυση μεταξύ των διαδικασιών και των 2 βαθμωτών μεταβλητών και μεταβλητών πινάκων.

Στο Figure 6 η γραφική παράσταση δείχνει ότι από τα 18 προγράμματα, 13 δείχνουν καλό παράλληλο speedup και μπορούν έτσι να επωφεληθούν από τους επιπλέον επεξεργαστές. Οι coarse grain τεχνικές του SUIF και οι βελτιστοποιήσεις στην μνήμη επηρεάζουν σημαντικά την απόδοση των μισών από τα προγραμμάτα. Τα προγράμματα swim και tomcatv δείχνουν εξαιρετικά γραμμικό speedup γιατί ο compiler εξαλειφθεί σχεδόν όλα τα cache misses και τα 14 Mbyte Datasets χοράνε πιο εύκολα στην συνολική cache των πλυπήρηνων επεξεργαστών.

Για τα περισσότερα που δεν είχαν θετικά speed up, ο compiler βρήκε ένα μεγάλο μέρος του υπολογισμού τους παραλληλοποιήσιμο, αλλά η διακριτότητα είναι πάρα πολύ καλή για να δώσει καλή multiprocessor απόδοση σε μηχανές με γρήγορους επεξεργαστές. Μόνο δύο εφαρμογές, η fpppp και η buuuk, δεν έχουν στατικά αναλύσιμο παραλληλισμό σε επίπεδο βρόγχων, κι έτσι δεν μπορούν να επηρεαστούν από τις τεχνικές του SUIF.

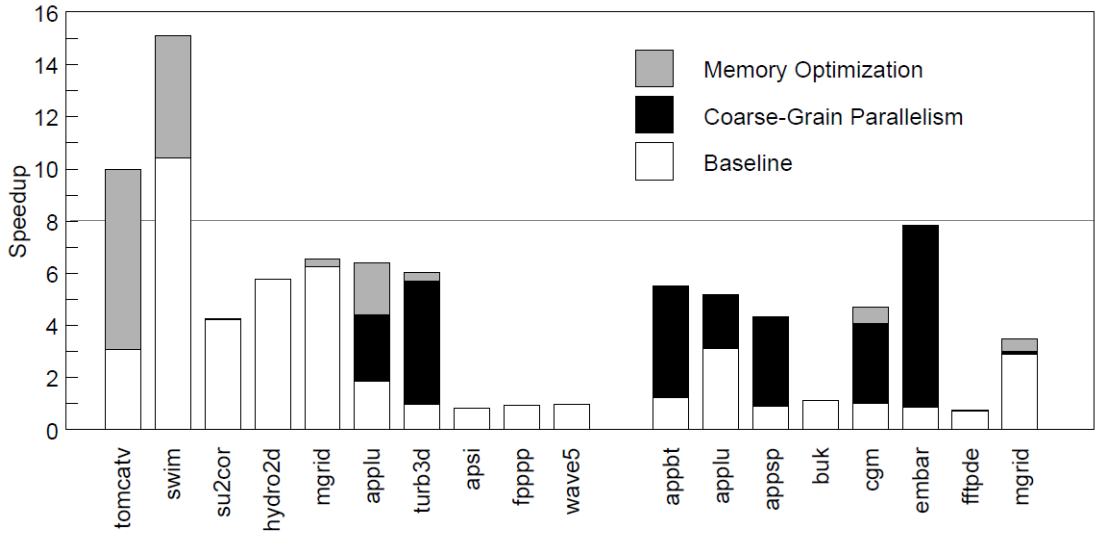


Figure 7: Impact of SUIF optimizations shown as speedup on eight processors[39]

Στο Figure 8 δείχνει τους χρόνους εκτέλεσης και τα SPEC ratios που λήφθηκαν στον επεξεργαστή AlphaServer 8400, και πιστοποιεί την υψηλή απόλυτη απόδοση του

compiler. Τα SPEC ratios συγκρίνουν την επίδοση της μηχανής με εκείνη ενός μηχανήματος αναφοράς. Το συνολικό SPEC ratio είναι ο γεωμετρικός μέσος των ratios που λαμβάνονται για μεμονωμένα προγράμματα. Ο γεωμετρικός μέσος των SPEC ratio βελτιώνεται κατά τη διάρκεια της uniprocessor εκτέλεσης κατά ένα συντελεστή τρεις με τέσσερις επεξεργαστές. Το eight-processor ratio του 63,9 αντιπροσωπεύει μια βελτίωση 50 τοις εκατό κατά το μεγαλύτερο αριθμό που έχει αναφερθεί μέχρι σήμερα.

	Execution Time (sec)			SPEC ratio		
Benchmark	1P	4P	8P	1P	4P	8P
tomcatv	219.1	30.3	18.5	16.9	122.1	200.0
swim	297.9	33.5	17.2	28.9	256.7	500.0
su2cor	155.0	44.9	31.0	9.0	31.2	45.2
hydro2d	249.4	61.1	40.7	9.6	39.3	59.0
mgrid	185.3	42.0	27.0	13.5	59.5	92.6
applu	296.1	85.5	39.5	7.4	25.7	55.7
turb3d	267.7	73.6	43.5	15.3	55.7	94.3
apsi	137.5	141.2	143.2	15.3	14.9	14.7
fpppp	331.6	331.6	331.6	29.0	29.0	29.0
wave5	151.8	141.9	147.4	19.8	21.1	20.4
SPEC ratio				**15.0**	**44.4**	**63.9**

Figure 8: Absolute Performance for the SPECfp95 benchmarks measured on a 440Mhz Digital AlphaServer[39]

5.3 Συμπέρασμα

Βλέπουμε από τα πειράματα που έχουν εκτελεστεί ότι ο SUIF compiler μπορεί να προσφέρει μεγάλα οφέλη σε θέματα speedup. Όσο πιο πολύπλοκο το project τόσο περισσότερο το όφελος της αυτόματης παραλληλοποίησης. Με την προϋπόθεση όμως ότι υπάρχει στατικά αναλύσιμοι βρόγχοι για να μπορεί να τους παραλληλοποιήσει.

Κεφάλαιο 6: Intel Parallel Studio

6.1 Εισαγωγή

Το Intel Parallel Studio είναι μία σουίτα προγραμμάτων που υποβοηθούν τους προγραμματιστές να παράγουν πιο εύκολα παράλληλα προγράμματα. Κυκλοφορεί σε student και full version και αντιθέτως με τα υπόλοιπα εργαλεία που θα περιγραφούν δεν είναι δωρεάν και έχει περισσότερο εμπορική χρήση παρά ερευνητική. Περιλαμβάνει διάφορες εφαρμογές που μπορούν να βοηθήσουν τον προγραμματιστή στο να παράγει παράλληλο κώδικα είναι:

- Intel Parallel Advisor
- Intel Vtune Amplifier (προηγουμένως γνωστό ως VTune Performance Analyzer)
- Intel Parallel Inspector
- Intel Parallel Composer

Η πιο πρόσφατη έκδοση του Intel Parallel Studio κυκλοφόρησε 26 Αυγούστου του 2014 και περιλαμβάνει plug-ins για να συνδέετε και να αποτελεί μέρος των Microsoft Visual Studio

και Eclipse έτσι ώστε ο χρήστης να μην χρειάζεται να αποχωριστεί το IDE προσωπικής του προτίμησης.

6.2 Intel Parallel Advisor

Το Intel Parallel Advisor[30] είναι ένα εργαλείο που βοηθά αρχιτέκτονες λογισμικού στην προτυποποίηση multi-threading εφαρμογών για C, C++, C# και Fortran.

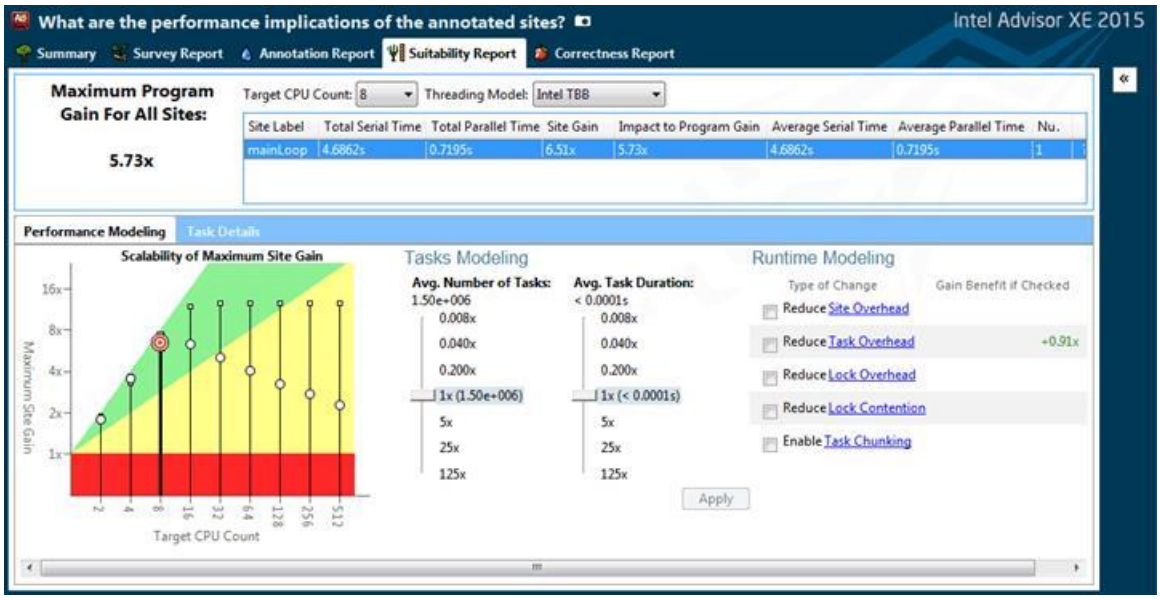


Figure 9: Intel Parallel Advisor – Suitability Report[30]

- Βοηθά στην γρήγορη μοντελοποίηση και σύγκριση των διαφόρων threading σχεδίων ως προς το performance scaling χωρίς όμως το κόστος και την ταλαιπωρία της υλοποίησης της τελικής εφαρμογής.

- Βοηθά στον βρεθούν διάφορα προβλήματα κοινόχρηστων δεδομένων κατά τη διάρκεια της σχεδίασης του λογισμικού, φάση στην οποία κοστίζει λιγότερο.

- Μοντελοποιά την επίδραση στην επίδοση κάθε επιπέδου παραλληλισμού που θέλουμε να προσθέσουμε καθώς επίσης προβλέπει την κλιμάκωση της απόδοσης σε συστήματα με μεγαλύτερο αριθμό πυρήνων.

- Μπορεί να δημιουργήσει ένα profile του κώδικα και υποδείξει στον προγραμματιστή ποια σημεία θα πάρουν περισσότερο όφελος από τη παραλληλοποιήση τους.

- Βοηθά στην σύγκριση μεταξύ κόστους παραλληλοποίησης και τελικού κέρδους στην επίδοση.

6.3 Intel VTune Amplifier

Το Intel VTune Amplifier[32] είναι μία εφαρμογή που βοηθά στην ανάλυση της απόδοσης 32-bit και 64-bit εφαρμογών. Παρέχει πολλές πληροφορίες ενός προγράμματος όπως π.χ. εύρεση hotspots του κώδικα και δημιουργία ενός profile για τον κώδικα της μηχανής.

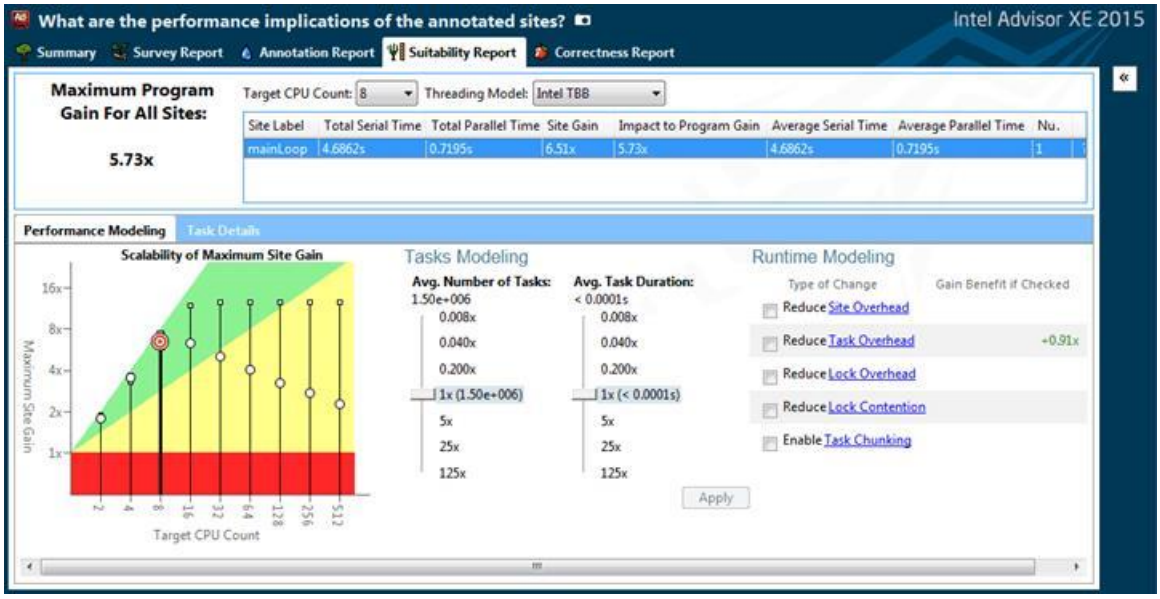


Figure 10: Intel VTune Amplifier[32]

Συγκεκριμένα παρέχει τις πιο κάτω λειτουργίες:

- Software sampling

  Δουλεύει κυρίως σε X86 επεξεργαστές και δίνει σαν αποτέλεσμα τα σημεία του κώδικα καθώς και το call stack για τα σημεία αυτά όπου ξοδεύετε ο περισσότερος χρόνος του προγράμματος.

- JIT profiling support

  Μπορεί να δημιουργήσει profile για δυναμικά δημιουργημένο κώδικα, δηλαδή κώδικα που δημιουργήθηκε με τον Just In Time compiler (π.χ. κώδικα που δημιουργείται από το JVM ή ο .NET Framework Runtime).

- Locks and waits analysis

  Εντοπίζει σημεία όπου για υπάρχουν μεγάλες καθυστερήσεις σε θέματα συγχρονισμού που συμβαίνει όταν οι πυρήνες υποχρησιμοποιούνται.

- Threading timeline

  Δείχνει τις συσχετίσεις μεταξύ των threads και βοηθά στον εντοπισμό της ισορροπίας του φορτίου σε κάθε thread καθώς προβλήματα συγχρονισμού.

- Source view

  Τα αποτελέσματα του sampling μπορούν να παρουσιαστούν δίπλα και γραμμή γραμμή με τον πηγαίο κώδικα.

- Hardware event sampling

  Χρησιμοποιώντας την μονάδα παρακολούθησης επίδοσης που υπάρχει στους Intel επεξεργαστές το εργαλείο αυτό μπορεί να εντοπίσει cache misses και branch mispredictions.

6.4 Intel Parallel Inspector

Ο Intel® Inspector XE[31] είναι ένας εύκολος στην χρήση debugger μνήμης και εντοπισμού threading errors σε εφαρμογές C, C++ και Fortran που τρέχουν σε Windows και Linux. Δεν απαιτούνται ειδικοί compilers και builds. Είναι συμβατός με compilers από vendors που ακολουθούν τα πρότυπα της πλατφόρμας. Απλά χρησιμοποιείς έναν απλό debugger ή production build. Χρησιμοποιείς την γραφική διεπαφή του χρήστη για ένα αυτοματοποιημένο regression test με τη γραμμή εντολών. Αυτή η διεπαφή του χρήστη μπορεί να χρησιμοποιηθεί αυτόνομα και στα Windows και στα Linux ή να ενσωματωθεί με

το Microsoft Visual Studio. Το Intel® Inspector XE βελτιώνει την παραγωγικότητα, μειώνει το κόστος και τον χρόνο εξόδου στην αγορά.

Ο Intel Parallel Inspector μόλις εντοπίσει ένα λάθος δείχνει ακριβώς το σημείο στον κώδικα όπου το λάθος συμβαίνει παρέχοντας ταυτόχρονα και ένα call stack για να δούμε πώς καταλήξαμε εδώ.

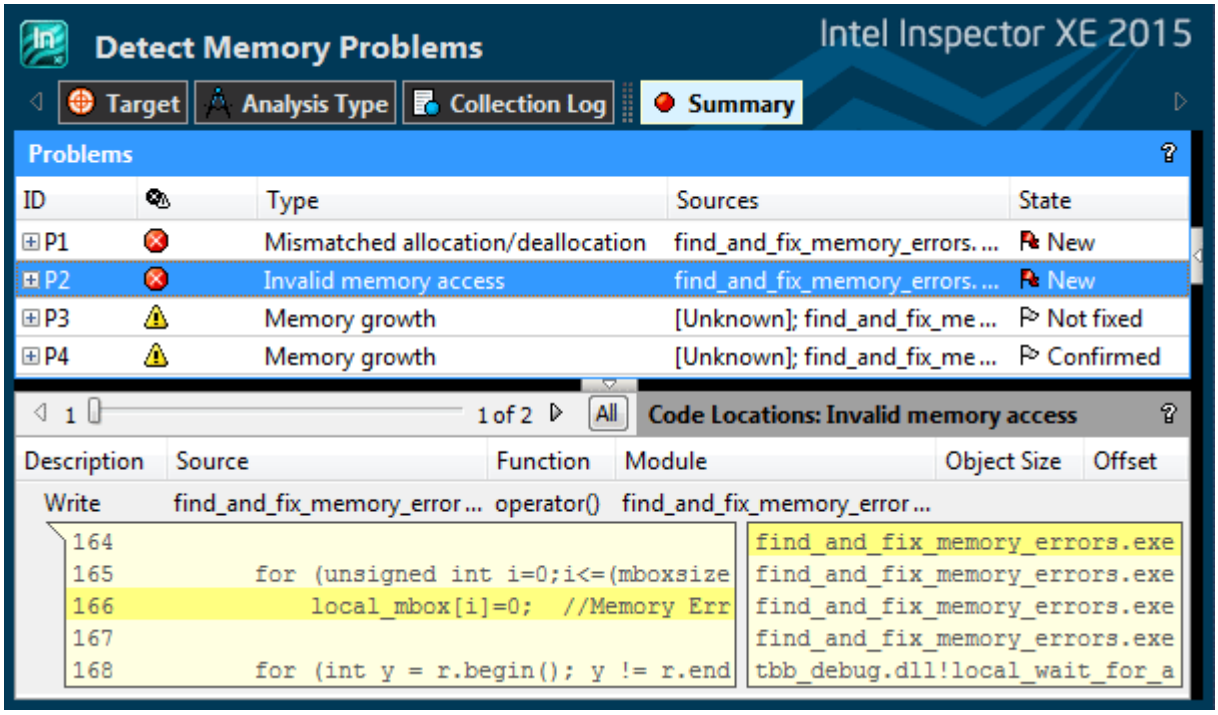


Figure 11: Intel Parallel Inspector[31]

Η δυναμική ανάλυση που κάνει, αποκαλύπτει λεπτές αδυναμίες και ελαττώματα στα οποία η αιτία είναι πολύ περίπλοκη για να εντοπιστεί από στατική ανάλυση. Σε αντίθεση με την στατική ανάλυση, η ολοκλήρωση εντοπισμού σφαλμάτων (debugger integration) σου επιτρέπει να διαγνώσεις το πρόβλημα και να βρεις και την αιτία. Ο Intel Inspector XE βρίσκει κρυμμένα σφάλματα στο μονοπάτι του κώδικα που εκτελείται, όπως επίσης βρίσκει και εντοπίζει σφάλματα μη ντετερμινιστικά ακόμα κι αν το σενάριο χρονοδιαγράμματος που προκαλεί το σφάλμα δεν έχει συμβεί.

Memory errors που μπορεί να εντοπίσει

- Διαρροές μνήμης (memory leaks)
- Καταστροφή της μνήμης (Memory corruption)

- Δέσμευση και αποδέσμευση των API mismatches (Allocation / de-allocation API mismatches)
- Ασυνεπής χρήση της μνήμης του API (Inconsistent memory API usage)
- Παράνομη πρόσβαση στην μνήμη (Illegal memory access)
- Μη αρχικοποιημένη ανάγνωση μνήμης (Uninitialized memory read)

Threading errors που μπορεί να εντοπίσει

- Αδιέξοδα (Deadlocks)
- Αγώνες δεδομένων (Data races)
  - Heap Races
  - Stack Races

6.5 Intel Parallel Composer

Ο Intel Parallel Composer είναι το τελικό σημείο της δημιουργίας ενός παράλληλου προγράμματος. Ένας προγραμματιστής με την βοήθεια όλων των πιο πάνω ή ακόμα και χωρίς αυτά μπορεί με την χρήση του Intel Parallel Composer να δημιουργήσει ένα παράλληλο πρόγραμμα που να εκμεταλλεύεται πλήρως τους πολυπύρηνους επεξεργαστές. Μπορεί να γράψει παράλληλο κώδικα, μπορεί να δώσει συγκεκριμένα σημεία στον compiler για να μετατρέψει σε παράλληλα ή απλά να ζητήσει από τον compiler να δημιουργήσει παράλληλο πρόγραμμα από τον σειριακό του κώδικα.

Αποτελείται από δύο βασικά κομμάτια τον Intel Fortran Compiler και τον Intel C/C++ Compiler.

6.5.1 Intel Fortan Compiler

Ο Intel Fortran Compiler[28], γνωστός και ως IFORT, είναι μια ομάδα από μεταγλωττιστές Fortran από την Intel. Οι μεταγλωττιστές παράγουν κώδικα για τις αρχιτεκτονικές IA-32 και Intel 64 και για ορισμένους μη-Intel, αλλά συμβατούς επεξεργαστές, όπως ορισμένους

επεξεργαστές AMD. Ο Intel Fortran Compiler υποστηρίζει auto-vectorization και μπορεί να δημιουργήσει SSE, SSE2, SSE3, SSSE3, SSE4 και AVX SIMD οδηγίες. Η χρήση αυτών των οδηγιών μέσω του compiler μπορεί να οδηγήσει σε βελτιωμένη απόδοση των εφαρμογών που τρέχουν στις αρχιτεκτονικές IA-32 και Intel 6, σε σύγκριση με εφαρμογές που έχουν δημιουργηθεί με τους compilers που δεν υποστηρίζουν αυτές τις οδηγίες. Επίσης, υποστηρίζει OpenMP 4.0, αυτόματη παραλληλοποίηση για συμμετρική πολυεπεξεργασία, το σύνολο σχεδόν της Fortran 2003 Standard και το μεγαλύτερο μέρος του Fortran 2008 standard, συμπεριλαμβανομένων Coarray Fortran.

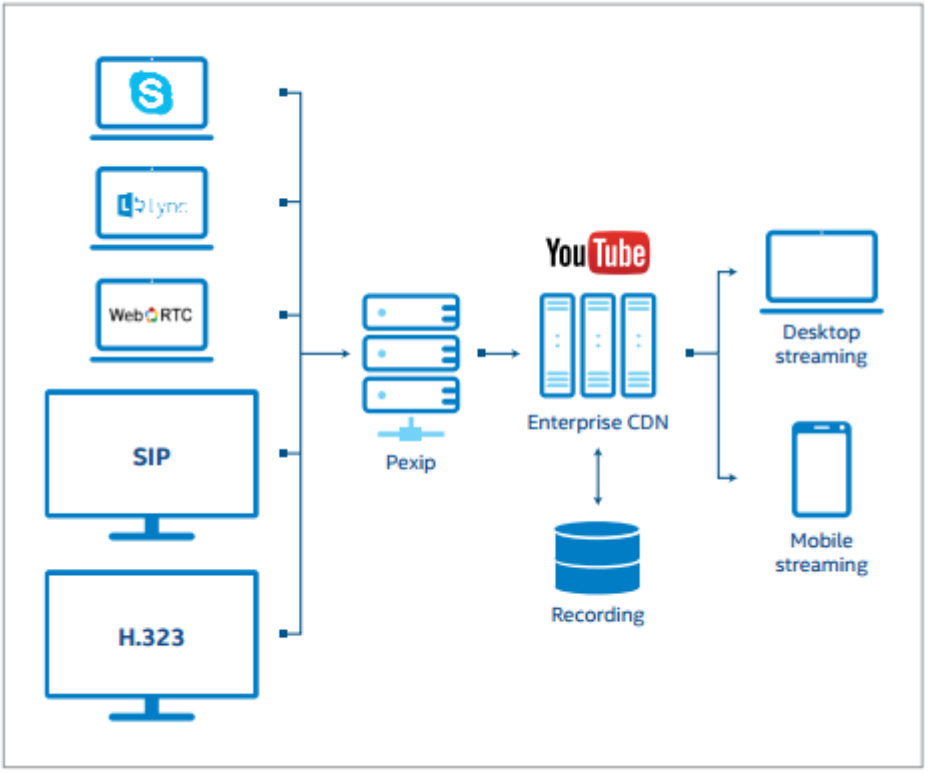
6.5.2 Intel C/C++ Compiler

Ο Intel C/C ++ Compiler[44], γνωστός και ως ICC ή ICL, είναι μία ομάδα από C και C ++ compilers από την Intel διατίθενται για OS X, Linux, Windows και Intel-based Android συσκευές. Αναγνωρίζει OpenMP και όλες τις βιβλιοθήκες που μπορεί να αναγνωρίσει ο gcc καθώς επίσης auto-vectorization και μπορεί να δημιουργήσει SSE, SSE2, SSE3, SSSE3, SSE4 και AVX SIMD οδηγίες όπως και ο Intel Fortran Compiler. Μπορεί αυτόματα να δημιουργήσει παράλληλο κώδικα ιδανικό για να εκτελεστεί σε Intel επεξεργαστές (IA-32 και Intel 6) και συμβατούς AMD.

6.6 Case Study: Pexip

6.6.1 Pexip Details

Η Pexip είναι μια ιδιωτική εταιρεία που εδρεύει στο Όσλο της Νορβηγίας που αναπτύσσει software τεχνολογίες για βίντεο, φωνής και content collaboration.



Έχει κυκλοφορήσει το Pexip Infinity το οποίο το οποίο είναι ένα cloud-based software για videoconferencing χρησιμοποιόντας πάρα πολλά standards όπως π.χ. Microsoft Lync, Skype, WebRTC, H323. Στην προσπάθεια της να βελτιώσει την ποιότητα του encoded video σήματος που μεταδίδει αποφάσισε να κάνει τους encoders της παράλληλους για να τρέχουν πολύ πιο γρήγορα και σε πραγματικό χρόνο να μπορούν να κάνουν encode βίντεο πολύ υψηλής ανάλυσης

6.6.2. Improvements using Intel Parallel Studio

Ακολουθήθηκε η πιο κάτω διαδικασία για βελτίωση του video encoder[33]:

1. Ανάλυση του κώδικα για hotspots με τη χρήση του Intel VTune Amplifier.

2. Εξέταση του κώδικα στα hotspots και αναζήτηση τρόπων για τη βελτίωση των αλγορίθμων.

3. Ανάλυση των vectorized βρόχων και SIMD (Single Instruction Multiple Data) οδηγίες χρησιμοποιώντας το Intel Advisor.

4. Δημιουργία τροποποιημένο κώδικα και αυτόματη παραλληλοποίηση του με το Intel C/C ++ Compiler.

5. Σύγκριση της βελτιστοποιημένη έκδοση με το πρωτότυπο.

Με τη χρήση λοιπόν του όλου συστήματος του Intel Parallel Studio και, σύμφωνα με τα λεγόμενα των εκπροσώπων της ίδιας της Pexip, έχουν καταφέρει να πετύχουν speedup μέχρι και 2.5 φορές σε σχέση με το αρχικό.

6.6.3 Συμπέρασμα

Το Intel Parallel Studio μπορεί να βοηθήσει σε διάφορους τομείς της παραλληλοποίησης ενός προγράμματος. Μπορεί να εντοπίσει σημεία που χρειάζονται παραλληλοποίηση, μπορεί να εντοπίσει σφάλματα σε παράλληλο πρόγραμμα και με την χρήση των Intel Compilers και συγκεκριμένα directives να δημιουργήσει αυτόματα παράλληλο κώδικα. Μπορεί όμως να βοηθήσει και άτομα που έχουν την ικανότητα και την εμπειρία στο να γράφουν μόνοι τους παράλληλο κώδικα καθώς μπορεί να τους βοηθήσει στα σημεία που πρέπει να εισάγουν παραλληλοποίηση, στο debugging του παράλληλου προγράμματος καθώς και στον εντοπισμών διάφορων errors είτε αυτά έχουν να κάνουν με memory faults, threading faults κλπ.

Κεφάλαιο 7: GCC

7.1 Λεπτομέρειες του GCC

Ο GCC είναι μία συλλογή από compilers ανοιχτού κώδικα και ήταν η βάση για την προώθηση και εγκαθίδρυση της κοινωνίας του ανοικτού κώδικα. Αρχικά όταν πρωτοκυκλοφόρησε αναγνώριζε μόνο κώδικα γραμμένο σε C αλλά στην πορεία εξελίκτηκε και αναγνωρίζει περισσότερες γλώσσες καθώς έχει γίνει και port σε διάφορα συστήματα και αρχιτεκτονικές όπως Symbian[25], Playstation 2, Dreamcast[27] και σε chips βασισμένα στην Freescale Power Architecture[26]. Παρέχει ένα μεγάλο εύρος από optimizations τα οποία σε κάθε νέα έκδοση αυξάνονται και βελτιώνονται. Επίσης μπορεί να αναγνωρίσει και να μεταγλωττίσει παράλληλο κώδικα αλλά έχει και την δυνατότητα να μετατρέψει σειριακό κώδικα σε παράλληλο.

7.2 SPEC 2006

7.2.1 Λεπτομέρειες για το SPEC2006

Το SPEC2006[34] είναι μία ομάδα από benchmarks για την αξιολόγηση διαφόρων CPU σε διαφορετικά είδη εκτέλεσης (integer operations, floating point operations κλπ).

Αυτή η ομάδα από benchmarks σχεδιάστηκε για να παρέχουν ένα συγκριτικό μέτρο των computer intensive software σε όλη την ευρύτερη φάσμα hardware χρησιμοποιώντας φόρτο εργασίας που αναπτύχθηκε από πραγματικές εφαρμογές χρήστη. Αυτά τα benchmarks παρέχονται ως πηγαίος κώδικας και απαιτούν από το χρήστη να δημιουργήσει τα εκτελέσιμα με την χρήση του compiler της επιλογής του.

Τα CPU SPEC ™ 2006 benchmarks έχουν πολλούς διαφορετικούς τρόπους για να μετρήσουν τις επιδόσεις του υπολογιστή. Ένας τρόπος είναι να μετρήσει πόσο γρήγορα ο υπολογιστής ολοκληρώνει ένα ενιαίο έργο. Ένας άλλος τρόπος είναι να μετρηθεί το πόσες πολλές διεργασίες μπορεί να ολοκληρώσει ένας υπολογιστής σε ένα ορισμένο χρονικό διάστημα.

7.2.2 Results

Χρησιμοποιώντας λοιπόν τον GCC και κάποια projects του SPEC2006 έχουν εκτελεστεί κάποια benchmarks[35] για σύγκριση εκτέλεσης σειριακού κώδικα και κώδικα που έχει παραλληλοποιηθεί αυτόματα από τον compiler. Συγκεκριμένα σε ένα υπολογιστή με 6 πυρήνες και 4 threads στον κάθε πυρήνα έχει εκτελεστεί το σειριακό project με όλα τα optimizations (-O3) εκτός αφαιρώντας όμως το automatic vectorization. Στη συνέχεια στον ίδιο υπολογιστή εκτελέστηκε το ίδιο project προσθέτοντας στο optimizations και αυτόματο παραλληλισμό σε 6 threads.

Τα αποτελέσματα του Speedup είχαν ως εξής:

- Project: 462.libquantum Speedup: **2.5X**
- Project: 410.bwaves Speedup: **3.3X**
- Project: 436.cactusADM Speedup: **4.5X**
- Project: 459.GemsFDTD Speedup: **1.27X**
- Project: 481.wrf **1.25X**

7.3 Συμπέρασμα

Βλέπουμε ότι και με την βοήθεια του GCC που είναι ένας αρκετά διαδεδομένος compiler, προφέρεται δωρεάν και δεν χρειάζεται κάποιο ειδικό τρόπο εγκατάστασης (πλεονεκτήματα σε σχέση με όλα τα άλλα εργαλεία που βοηθούν στην αυτόματη παραλληλοποίηση) μπορούμε να πετύχουμε αρκετά θετικά αποτελέσματα. Σε πολύ πολύπλοκα projects όπως αυτά του CPU SPEC 2006 καταφέρνουμε να πετύχουμε περισσότερο και από 4 φορές αύξηση στον χρόνο εκτέλεσης του λογισμικού μας. Ένας προγραμματιστής λοιπόν που δεν είναι τόσο έμπειρος στη συγγραφή παράλληλου κώδικα μπορεί να επωφελθεί και να αξιοποιήσει τις νέες τεχνολογίες των πολυπύρηνων CPU.

Κεφάλαιο 8: Σύγκριση χρόνου εκτέλεσης αυτόματα παράλληλου κώδικα με hand-code παράλληλου κώδικα

Χρησιμοποιώντας τον GCC και το project NPB του NASA Advanced Supercomputer Division έτρεξα πειράματα σύγκρισης παράλληλου κώδικα που δημιουργήθηκε αυτόματα από τον compiler και παράλληλου κώδικα που δημιουργήθηκε hand-coded από έμπειρους προγραμματιστές.

8.1 NPB – NAS Parallel Benchmarks

Τα NAS Παράλληλη Benchmarks (NPB)[36] είναι μια σουίτα παράλληλων benchmarks για αξιολόγηση ενός υπολογιστή. Αρχικά αναπτύχθηκαν στο NASA Ames Research Center το 1991 για την αξιολόγηση high-end παράλληλων υπερυπολογιστών. Παρόλο που δεν χρησιμοποιούνται πλέον τόσο ευρέως όσο παλιά για τη σύγκριση τις απόδοσης των high-end συστημάτων, συνεχίζουν να μελετιούνται και να αναλύονται σε μεγάλο βαθμό στην κοινότητα του high performance computing. Το ακρωνύμιο "NAS" δήλωνε αρχικά για το Αριθμητικό Πρόγραμμα Προσομοίωσης Αεροναυπηγών (Numerical Aeronautical Simulation) στη NASA Ames. Το όνομα αυτού του οργανισμού στη συνέχεια άλλαξε σε Αριθμητικό Πρόγραμμα Προσομοίωσης Αεροδιαστημικής, και πιο πρόσφατα σε NASA Advanced Supercomputing Center, αν και το ακρωνύμιο παραμένει "NAS." Οι προγραμματιστές του της πρώτης έκδοσης των NPB ήταν οι David H. Bailey, Eric Barszcz, John Barton, David Browning, Russell Carter, LeoDagum, Rod Fatoohi, Samuel Fineberg, Paul Frederickson, Thomas Lasinski, Rob Schreiber, Horst Simon, V. Venkatakrishnan and Sisira Weeratunga.

Για τα πειράματα μου χρησιμοποίησα δύο projects του NPB. Το IS και το DC.

8.1.1 IS Project

Αυτό το project εκτελεί sorting σε ένα μεγάλο σετ από πολύ μεγάλους Integer αριθμούς που είναι σημαντικό σε ορισμένους "particle methods" κώδικες. Ελέγχει την ταχύτητα υπολογισμού καθώς και την ταχύτητα επικοινωνίας. Σκοπός του συγκεκριμένο προβλήματος[37] είναι να δημιουργήσει ένα μεγάλο φάσμα από ένα ορισμένο σχήμα και στη συνέχεια να το λύσει. Δημιουργεί N ακέραιους αριθμούς οι οποίοι πρέπει να είναι ομοιόμορφα κατανεμημένοι στη μνήμη και στη συνέχεια τους ταξινομεί χρησιμοποιώντας bucket sort.

8.1.2 DC Project

Το project αυτό προσπαθεί να αναλύσει ένα Data cube γνωστό και ως OLAP cube [36]. Τα αρχικά OLAP αντιστοιχούν στο online analytical processing[38]. Ένας OLAP cube είναι ένα πολυδιάστατο σετ από δεδομένα. Για παράδειγμα μια εταιρεία θέλει να αναλύσει οικονομικά δεδομένα βάση προϊόντος, χρονικής περιόδου και πόλης για να συγκρίνει τα έξοδα της βάση προϋπολογισμού. Οι παράγοντες είναι περισσότεροι από δύο και έτσι χρειάζονται περισσότερες από δύο διαστάσεις στην ανάλυση του προβλήματος.

8.1.3 Datasets

Το κάθε ένα από τα projects του NPB τρέχει με συγκεκριμένους αριθμούς από data τα οποία πρέπει να καθοριστούν κατά την διάρκεια του compilation. Ο αριθμός αυτός ουσιαστικά καθορίζει το μέγεθος του προβλήματος που προσπαθεί το project να επιλύσει.

Συγκεκριμένα το project IS μπορεί να δουλέψει με τα datasets:

- S

  Πλήθος αριθμών: 2^{16}

Μέγιστος αριθμός: 2^{11}

- W

Πλήθος αριθμών: 2^{20}

Μέγιστος αριθμός: 2^{16}

- A

Πλήθος αριθμών: 2^{23}

Μέγιστος αριθμός: 2^{19}

- B

Πλήθος αριθμών: 2^{25}

Μέγιστος αριθμός: 2^{21}

- C

Πλήθος αριθμών: 2^{27}

Μέγιστος αριθμός: 2^{23}

και το project DC μπορεί να δουλέψει με τα πιο κάτω Datasets:

- S

Μέγεθος εισόδου: 10^3

Πλήθος διαστάσεων: 5

- W

Μέγεθος εισόδου: 10^5

Πλήθος διαστάσεων: 10

- A

Μέγεθος εισόδου: 10^6

Πλήθος διαστάσεων: 15

- B

Μέγεθος εισόδου: 10^7

Πλήθος διαστάσεων: 20

8.2 Procedure

Από τα NAS Parallel Benchmarks χρησιμοποίησα το IS και το DC και για compiler χρησιμοποίησα τον GCC. Εκτέλεσα πρώτα το κάθε ένα με όλα τα datasets τα οποία δέχεται σε σειριακό με όλα τα optimizations που περιλαμβάνει το flag O3. Ακολούθως τα εκτέλεσα με την χρήση automatic parallelization flags και τέλος εκτέλεσα την παράλληλη έκδοση του κάθε project.

Το κάθε project καταγράφει από μόνο του τον χρόνο εκτέλεσης του αλλά αυτό καταγράφει μόνο τον χρόνο εκτέλεσης του αλγόριθμου.

8.3 Benchmark Results

Μονάδα μέτρησης των πειραμάτων είναι σε δευτερόλεπτα

8.3.1 IS project

Χρόνοι εκτέλεσης του benchmark με το κάθε dataset:

	O3	O3 AUTOMATIC	O3

		PARALLEL	OPENMP
S	0,01	0,01	0
W	0,12	0,12	0,06
A	0,99	0,98	0,13
B	3,97	3,37	0,52
C	23,09	22,06	2,42

Από τους χρόνους εκτέλεσης είναι εμφανές ότι με το Dataset S δεν μπορούμε να έχουμε αξιόλογες μετρήσεις καθώς ολοκληρώνετε η εκτέλεση σε πολύ μικρό χρονικό διάστημα στη σειριακή εκτέλεση κι έτσι δεν υπάρχει κάποιο όφελος, αλλά ούτε και μείον.

Θα αναλύσουμε όμως καλύτερα με τα speedups

Τα speedups του projects έχουν ως εξής

	Speedup serial/automatic parallel	Speedup automatic parallel/openmp
S	1	infinity
W	1	2
A	1,01	7,53
B	1,17	6,48
C	1,001	9,52

Συγκρίνοντας τα speedups που παίρνουμε από τον αυτόματο παραλληλισμό βλέπουμε καθαρά ότι ο αυτόματος παραλληλισμός στο συγκεκριμένο project δεν προσφέρει και κάτι ιδιαίτερο. Το κέρδος είναι αρκετά μικρό. Ενώ αντιθέτως αν πάμε από το αυτόματα παράλληλο κώδικα στον hand-coded παράλληλο τότε έχουμε πάρα πολλά να κερδίσουμε καθώς όσο μεγαλώνει το dataset τόσο αυξάνεται και το speedup.

8.3.2 DC project

Χρόνοι εκτέλεσης του benchmark με το κάθε dataset:

	O3	O3 AUTOMATIC PARALLEL	O3 OPENMP
S	0,01	0,01	0,01
W	50,96	50,06	5,35
A	412,37	371,86	132,2
B	938,02	892,84	365,6

Από τους χρόνους εκτέλεσης είναι εμφανές ότι με το Dataset S δεν μπορούμε να έχουμε αξιόλογες μετρήσεις καθώς ολοκληρώνετε η εκτέλεση σε πολύ μικρό χρονικό διάστημα στη σειριακή εκτέλεση κι έτσι δεν υπάρχει κάποιο όφελος, αλλά ούτε και μείον.

Θα αναλύσουμε όμως καλύτερα με τα speedups.

Τα speedups του projects έχουν ως εξής

	Speedup serial/automatic parallel	Speedup automatic parallel/openmp
S	1	1
W	1,017	9,35
A	1,108	2,81
B	1,05	2,44

Βλέπουμε για ακόμα μια φορά ότι ο αυτόματος παραλληλισμός προσφέρει μεν ένα speedup στον τελικό χρόνο εκτέλεσης αλλά αυτό το speedup δεν είναι και τόσο μεγάλο. Όμως, στο συγκεκριμένο project επειδή είναι πολύ μεγάλοι γενικά οι χρόνοι εκτέλεσης ένα speedup της τάξης του 1.288 είναι αρκετά σημαντικό και κερδοφόρο.

Επίσης βλέπουμε ότι το speedup στη μετακίνηση από αυτόματα παράλληλο σε hand-coded παράλληλο κώδικα δεν είναι τόσο μεγάλο ώστε να δικαιολογεί τον κόπο και το κόστος μετατροπής της εφαρμογής σε παράλληλη.

Κεφάλαιο 9: Συγκρίσεις και τελικό συμπέρασμα

9.1 Σύγκριση όλων των compilers

Πιο κάτω βλέπουμε τις διάφορες τεχνικές που μπορούν να χρησιμοποιηθούν για αυτόματη δημιουργία παράλληλου κώδικα. Κάθε compiler υποστηρίζει διαφορετικές τεχνικές και αυτό επηρεάζει τα αποτελέσματα που μπορεί να παράξει.

	SUIF	par4all	Intel Parallel Studio	gcc	polaris
Advanced induction variable analysis		✓		✓	✓
Automatic inlining of subroutines					✓
Array privatization	✓	✓			✓
Advanced reduction analysis.	✓	✓		✓	✓
Loop Parallelization	✓		✓	✓	
Vectorization	✓		✓	✓	
coarse grain parallelism	✓				

Όλοι όμως μπορούν να δώσουν θετικά αποτελέσματα. Κάποια μικρά και κάποια μεγαλύτερα.

9.2 Τελικό Συμπέρασμα

Είναι εμφανές λοιπόν ότι πρέπει να αρχίσουμε να εκμεταλλευόμαστε περισσότερο τις νέες τεχνολογίες παράλληλων επεξεργαστών καθώς μπορούν να μας βοηθήσουν πάρα πολύ στους χρόνους εκτέλεσης των προγραμμάτων μας. Σε πολλές περιπτώσεις το κόστος μετατροπής αλλά και οι γνώσεις και η εμπειρία που χρειάζεται ένας προγραμματιστής για να μετατρέψει ένα σειριακό πρόγραμμα σε παράλληλο χωρίς λάθη στον κώδικα ή στα αποτελέσματα φαντάζει απαγορευτικό.

Γι' αυτό λοιπόν το λόγο μπορούμε να αξιοποιήσουμε τις τεχνολογίες που μας παρέχουν οι διάφοροι μεταγλωττιστές στο να δημιουργήσουν από μόνοι τους παράλληλο κώδικα ή κατευθείαν μία εφαρμογή που να εκμεταλλεύεται τις παράλληλες τεχνολογίες.

Το κέρδος σε πολλές από αυτές τις περιπτώσεις μπορεί να μην συγκρίνεται με τον hand-coded παράλληλο κώδικα αλλά πάντα υπάρχει ένα speedup σε σύγκριση με τον σειριακό κώδικα. Και δεδομένου ότι το κόστος αυτόματης παραλληλοποίησης είναι μηδαμινό σε σχέση με τον hand-coded παράλληλο κώδικα του οποίου το κόστος είναι τεράστιο, τότε κάλο είναι σε όλες μας τις εφαρμογές να χρησιμοποιούμε αυτόματο παραλληλισμό αφού σχεδόν όλοι οι σύγχρονοι compilers το παρέχουν.

References

1. **Fox, Geoffrey; Roy Williams; Paul Messina** *(1994). Parallel Computing Works!. Morgan Kaufmann. pp. 575, 593*

2. **Simone Campanoni, Timothy Jones, Glenn Holloway, Gu-Yeon Wei, David Brooks.** "The HELIX Project: Overview and Directions". 2012.

3. https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/GUID-32ED933F-5E8A-4909-A581-4E9DB59A6933.htm

4. https://gcc.gnu.org/projects/tree-ssa/vectorization.html

5. **Flynn, M. J**. (September 1972). "Some Computer Organizations and Their Effectiveness". IEEE Trans. Comput. C–21 (9): 948–960

6. **Duncan, R.** (February 1990). "A survey of parallel computer architectures".

7. **Martin, Milo,** Performance and Benchmarking http://www.cis.upenn.edu/~milom/cis501-Fall12/lectures/04_performance.pdf

8. **Henry Kasim, Verdi March, Rita Zhang, Simon See,** Survey on Parallel Programming Model, J. Cao et al. (Eds.): NPC 2008, LNCS 5245, pp. 266– 275, 2008, IFIP International Federation for Information Processing

9. **Krste Asanovíc, Rastislav Bodik, Bryan Catanzaro, Joseph Gebis, Parry Husbands, Kurt Keutzer, David Patterson, William Plishker, John Shalf, Samuel Williams, and Katherine Yelick,** The Landscape of Parallel Computing Research: A View from Berkeley, University of California, Berkeley, No. UCB/EECS-2006-183, December 18, 2006.

10. *"Intel® Parallel Studio XE 2015 Professional Edition Release Notes"*
11. https://software.intel.com/en-us/articles/automatic-parallelization-with-intel-compilers

12. *Intel Corporation. Intel Parallel Studio.* http://software.intel.com/en-us/intel-parallel-studio/home
13. *Intel Corporation. Intel Compilers. http://www.intel.com/software/products*

14. ***Hall, M.W. USC Inf. Scis. Inst., Marina del Rey, CA, USA Anderson, J.M. ; Amarasinghe, S.P. ; Murphy, B.R. ; Shih-Wei Liao ; Bugnion, E. ; Lam, M.S.*** *Maximizing multiprocessor performance with the SUIF compiler*
15. ***Robert P. Wilson, Robert S. French, Christopher S. Wilson, Saman P. Amarasinghe, Jennifer M. Anderson, Steve W. K. Tjiang, Shih-Wei Liao, Chau-Wen Tseng, Mary W. Hall, Monica S. Lam, John L. Hennessy*** *SUIF: an infrastructure for research on parallelizing and optimizing compilers. ACM SIGPLAN Notices, v.29 n.12, p.31-37, Dec. 1994.*
16. *SUIF Compiler Infrastructure, http:/suif.stanford.edu/, Accessed: February 2010.*
17. *http://pluto-compiler.sourceforge.net/*
18. http://www.par4all.org/features
19. https://www.gnu.org/software/gcc/releases.html *"GCC Releases". GNU Project. Retrieved 2006-12-27.*
20. http://gcc.gnu.org/frontends.html *Programming Languages Supported by GCC". GNU Project. Retrieved 2014-06-23.*
21. http://astro.unistra.fr/
22. http://hyantes.gforge.inria.fr/
23. http://hypercarte.imag.fr/
24. *http://polaris.cs.uiuc.edu/polaris/polaris-old.html*
25. *Symbian Improvement project http://www.inf.u-szeged.hu/projectdirs/symbian-gcc/*
26. *Linux Board Support Packages http://www.freescale.com/webapp/sps/site/overview.jsp?code=CW_BSP&fsrch=1*
27. *Sh4-g++ guide http://web.archive.org/web/20021220025554/http://www.ngine.de/gccguide.html*
28. *https://software.intel.com/en-us/articles/intel-parallel-studio-xe-2015-composer-edition-fortran-release-notes*
29. *http://www.par4all.org/benchmarks.html*
30. *https://software.intel.com/en-us/intel-advisor-xe*
31. *https://software.intel.com/en-us/intel-inspector-xe*
32. *https://software.intel.com/en-us/intel-vtune-amplifier-xe*
33. *https://software.intel.com/sites/default/files/managed/77/ea/pexip-case-study.pdf*
34. *https://www.spec.org/cpu2006/*
35. *https://gcc.gnu.org/wiki/AutoParInGCC*
36. **David H Bailey** The NAS Parallel Benchmarks
37. **D Bailey, E Barszcz, L Dagum, P Fredericksony, R Schreibery and H Simon** The Nas Parallel Benchmarks: The Kernel Benchmarks
38. ***"Just What Are Cubes Anyway? (A Painless Introduction to OLAP Technology)" http://msdn.microsoft.com/en-us/library/aa140038%28v=office.10%29.aspx#odc_da_whatrcubes_topic2***
39. **Anderson, J.M., Murphy, B.R., Shih-Wei Liao , Bugnion, E. , Lam, M.S.** Maximizing multiprocessor performance with the SUIF compiler
40. **Robert P. Wilson, Robert S. French, Christopher S. Wilson, Saman P. Amarasinghe, Jennifer M. Anderson, Steve W. K. Tjiang, Shih-Wei Liao, Chau-**

Wen Tseng, Mary W. Hall, Monica S. Lam, and John L. Hennessy SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers

41. https://github.com/Par4All/par4all/tree/p4a/examples/P4A/Hyantes
42. https://www.stroustrup.com/~rwerger/Courses/654/ch2.pdf
43. https://software.intel.com/en-us/c-compilers