Μετροπρογράμματα ελέγχου και επιδόσεων για συστήματα OpenMP

Άγγελος Κοσιντζής

Διπλωματική Εργασία

Επιβλέπων: Βασίλειος Δημακόπουλος

Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Πολυτεχνική Σχολή
Πανεπιστήμιο Ιωαννίνων

Ιωάννινα, Σεπτέμβριος 2019

Περίληψη

Η έρευνα γύρω από τα παράλληλα συστήματα σε κάθε τομέα, από τη σχεδίαση καινούργιων αρχιτεκτονικών, σύμφωνα με τις ανάγκες της εποχής, μέχρι την ανάπτυξη ενός μεταφραστή για κάποιο μοντέλο προγραμματισμού συστημάτων, στηρίζεται πάνω σε πειράματα και δοκιμές μετροπρογραμμάτων (benchmarks). Τα προγράμματα αυτά, είτε έχουν αναπτυχθεί αποκλειστικά για τον σκοπό αυτό είτε έχουν τροποποιηθεί τα ήδη υπάρχοντα με κάποιες επιθυμητές ιδιότητες για να αντιπροσωπεύσουν διαφορετικούς τομείς της επιστήμης. Η διπλωματική εργασία αυτή αποτελεί μία προσπάθεια να συλλεχθούν όλα τα μετροπρογράμματα που έχουν προταθεί για το προγραμματιστικό μοντέλο OpenMP, να αναλυθεί η λειτουργία τους και να καταγραφούν οι στόχοι του καθενός από αυτά, καταλήγοντας σε μία συνολική κατηγοριοποίησή τους. Ταυτόχρονα, και με βάση αυτά τα προγράμματα, γίνεται μία συγκριτική μελέτη δημοφιλών και ευρέως διαθέσιμων μεταφραστών OpenMP σε διαφορετικές παράλληλες αρχιτεκτονικές.

Πίνακας Περιεχομένων

Κατάλογος Πινάκων

Κατάλογος Σχημάτων

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΗ

1.1 ΕΞΕΛΙΞΗ ΤΩΝ Η/Υ

Εδώ και χιλιάδες χρόνια, χρησιμοποιούνται συσκευές για να δώσουν βοήθεια στους υπολογισμούς. Από απλές συσκευές, όπως ο άβακας, μέχρι μηχανικές συσκευές τον $19^{ο}$ αιώνα και αναλογικούς υπολογιστές στο πρώτο μισό του $20^{ου}$. Οι υπολογιστές λοιπόν, κατά μία έννοια ήταν πάντα ένα βοηθητικό εργαλείο για τον άνθρωπο. Από την εμφάνιση του πρώτου ηλεκτρονικού υπολογιστή ξεκίνησε μια ραγδαία εξέλιξη σε όλους τους τομείς. Η κατανάλωση ενέργειας περιορίστηκε, νέες ιδέες εφαρμόστηκαν, όπως η αποθήκευση του προγράμματος, οι αποστάσεις μειώθηκαν, οι μνήμες μεγάλωσαν και το πιο σημαντικό: η ταχύτητα του επεξεργαστή αυξήθηκε. Όλα αυτά είχαν ως αποτέλεσμα πολλά προβλήματα να λυθούν και να προοδεύσουν άλλες επιστήμες, για τις οποίες πολλά προβλήματα δεν ήταν δυνατόν παλαιότερα να επιλυθούν. Επιπλέον, η εξέλιξη των υπολογιστών ικανοποίησε ανάγκες της εποχής και πιθανόν να έφερε καινούργιες. Ένας υπολογιστής έπρεπε να ανταποκριθεί σε, ολοένα και περισσότερες, απαιτήσεις και να καλύψει ολοένα και περισσότερες ανάγκες. Όσον αφορά τον επεξεργαστή, η εξέλιξή του ήταν ραγδαία, με την εμφάνιση των μικροεπεξεργαστών και την μείωση του μεγέθους των τρανζίστορ, ακολουθώντας το νόμο του Moore. Ενώ όμως τα μεγέθη, η πολυπλοκότητα και η κατασκευή των CPUs άλλαξαν τόσο πολύ, οι βασικές ιδέες και η λειτουργία τους παρέμειναν ίδιες. Πλέον, ο νόμος του Moore δεν ακολουθείται επακριβώς, καθώς τα μεγέθη δεν μπορούν να μικρύνουν άλλο χωρίς να αυξηθεί υπερβολικά η κατανάλωση ενέργειας. Καθώς όμως οι ανάγκες του ανθρώπου μεγαλώνουν ασταμάτητα και συνεχώς γεννιούνται προβλήματα που επιζητούν λύση, είναι επόμενο να πρέπει να εφαρμοστούν νέες ιδέες για την αύξηση της ταχύτητας. Οι νέες ιδέες που προκύπτουν και οι έρευνες που γίνονται αφορούν τους κβαντικούς υπολογιστές, όπως επίσης και την επέκταση της χρήσης μεθόδων παραλληλισμού που εκμεταλλεύονται την χρησιμότητα του κλασσικού Von Neumann μοντέλου.

1.2 Παραλληλα Συστηματα

Ο παράλληλος υπολογισμός είναι μια μορφή υπολογισμού κατά την οποία κάποια τμήματα αυτού εκτελούνται ανεξάρτητα. Ορισμένα μεγάλα προβλήματα μπορούν να διαιρεθούν σε μικρότερα, τα οποία μπορούν να επιλυθούν ταυτόχρονα. Τα τελευταία χρόνια, ο παραλληλισμός αποτελεί το βασικό στοιχείο των σύγχρονων αρχιτεκτονικών, κυρίως με τη μορφή των πολυπύρηνων επεξεργαστών.

1.2.2 Αρχιτεκτονικες Παραλληλων Συστηματων

Ένα παράλληλο σύστημα αποτελείται από διαφορετικές μονάδες. Αυτές οι μονάδες μπορεί να είναι είτε πυρήνες σε ένα chip, είτε διαφορετικοί υπολογιστές. Στην πρώτη περίπτωση, οι πυρήνες μοιράζονται κάποιο κοινό χώρο διευθύνσεων. Αυτά είναι τα παράλληλα συστήματα κοινόχρηστης μνήμης. Οι διαφορετικοί επεξεργαστές επικοινωνούν μέσω της μνήμης, τροποποιώντας μεταβλητές. Η μνήμη συνήθως τοποθετείται σε κεντρική θέση, όπως στα συστήματα SMP (symmetric multiprocessors), στα οποία κάθε επεξεργαστής είναι συνδεδεμένος σε έναν κοινό δίαυλο μέσω του οποίου έχει πρόσβαση σε αυτήν. Όλοι οι επεξεργαστές έχουν τον ίδιο χρόνο προσπέλασης στην κύρια μνήμη (Uniform Memory Access) και για το λόγο αυτό τα συστήματα αυτά αναφέρονται ως UMA. Κάθε επεξεργαστής επίσης έχει μια τοπική cache, με αποτέλεσμα να προκύπτουν προβλήματα όσον αφορά τη συνέπειά της. Η άλλη εναλλακτική, όσον αφορά τα συστήματα κοινόχρηστης μνήμης, είναι το NUMA (Non-uniform Memory Access) μοντέλο, όπου κάθε επεξεργαστής είναι συνδεδεμένος σε μια αφοσιωμένη σε αυτόν μνήμη. Ωστόσο, αυτά τα κομμάτια μνήμης συνδυάζονται, ώστε να προκύπτει ένας κοινός χώρος διευθύνσεων. Σε αντίθεση με το UMA μοντέλο, ο χρόνος προσπέλασης της μνήμης για κάθε επεξεργαστή διαφέρει, καθώς εξαρτάται από την τοποθέτησή του. Οι πυρήνες ομαδοποιούνται σε κόμβους (nodes) στους οποίους κάθε πυρήνας έχει το δικό του ελεγκτή μνήμης.

Η άλλη κατηγορία είναι τα συστήματα κατανεμημένης μνήμης, τα οποία αποτελούνται από μονάδες που διαθέτουν δικιά τους μνήμη, συνήθως διαφορετικοί υπολογιστές. Τέτοια συστήματα είναι τα clusters, συστάδες υπολογιστών που δουλεύουν ακατάπαυστα για την επίλυση ενός προβλήματος και υπάρχουν εδώ και αρκετά χρόνια. Το πλεονέκτημα αυτής της οργάνωσης είναι ότι κάθε μονάδα μπορεί να χρησιμοποιήσει ολόκληρο

το διαθέσιμο εύρος ζώνης της τοπικής μνήμης χωρίς να παρεμβάλλεται κάποιος άλλος. Επιπλέον, εξαιτίας της απουσίας διαύλου δεν υπάρχει περιορισμός στο πλήθος των μονάδων που μπορούν να συνδεθούν, με αποτέλεσμα το μέγεθος του συστήματος να περιορίζεται μόνο από το δίκτυο που χρησιμοποιείται για τη σύνδεση των μονάδων. Ένα ακόμα πλεονέκτημα, είναι ότι δεν υπάρχουν προβλήματα συνέπειας της cache καθώς κάθε επεξεργαστής είναι υπεύθυνος για τη συνέπεια της δικιάς του. Όλα αυτά τα πλεονεκτήματα αντικρούονται από τη δυσκολία της επικοινωνίας μεταξύ των μονάδων. Αν ένας επεξεργαστής χρειάζεται δεδομένα από έναν άλλον θα πρέπει να γίνει ανταλλαγή μηνυμάτων, γεγονός το οποίο απαιτεί χρόνο για τη δημιουργία και αποστολή του μηνύματος καθώς και τη λήψη του, η οποία απαιτεί διακοπή της λειτουργίας του αποδέκτη για την επεξεργασία των μηνυμάτων.

1.2.3 Προγραμματιστικα Μοντελα

Σε ένα σύστημα κοινόχρηστης μνήμης, οι παράλληλες διεργασίες ή νήματα μοιράζονται έναν κοινό χώρο διευθύνσεων στον οποίο γράφουν και διαβάζουν μεταβλητές ασύγχρονα. Αυτό μπορεί να οδηγήσει σε συνθήκες ανταγωνισμού για έναν πόρο που τροποποιείται ή διαβάζεται από διαφορετικές διεργασίες. Για την αντιμετώπιση αυτού του φαινομένου υπάρχουν διάφοροι μηχανισμοί, όπως είναι οι κλειδαριές. Η πιο διαδεδομένη βιβλιοθήκη για τον προγραμματισμό συστημάτων κοινόχρηστης μνήμης είναι τα νήματα POSIX (Pthreads). Ο προγραμματιστής είναι υπεύθυνος να διατηρήσει τη συνέπεια των δεδομένων και τη σωστή λειτουργία του προγράμματος χρησιμοποιώντας κλήσεις της βιβλιοθήκης για το συντονισμό των νημάτων. Εξαιτίας αυτής της δυσκολίας, έχουν προταθεί διάφορες ιδέες ώστε να διευκολύνουν τους προγραμματιστές. Εκτός της δημιουργίας μιας νέας γλώσσας, όπως η Ada- η οποία δεν είναι πολύ διαδεδομένη- μια ιδέα είναι η χρήση ρουτινών βιβλιοθήκης σε μια υπάρχουσα ακολουθιακή γλώσσα, όπως συμβαίνει στην περίπτωση της TBB (Threading Building Blocks), μιας βιβλιοθήκης για τη γλώσσα C++ που έχει αναπτυχθεί από την Intel. Μια άλλη ιδέα είναι η τροποποίηση της σύνταξης μιας ακολουθιακής γλώσσας, όπως συμβαίνει στην περίπτωση του UPC (unified parallel C), το οποίο είναι ένα προγραμματιστικό μοντέλο τόσο για συστήματα κοινόχρηστης μνήμης, όσο και για κατανεμημένα. Ο άλλος τρόπος είναι μέσω οδηγιών προς το μεταφραστή, χρησιμοποιώντας μια υπάρχουσα ακολουθιακή γλώσσα, όπως για παράδειγμα το OpenMP. Το OpenMP είναι μια διεπαφή που

περιλαμβάνει ένα σετ από οδηγίες προς το μεταφραστή, ένα σύνολο από συναρτήσεις της βιβλιοθήκης και μεταβλητές περιβάλλοντος που επηρεάζουν τη συμπεριφορά του χρόνου εκτέλεσης. Η ιδέα είναι ο χρήστης να μπορεί εύκολα να αναπτύξει παράλληλες εφαρμογές, ελαφρύνοντας τον από περίπλοκες διαδικασίες που έχουν να κάνουν με το συντονισμό των νημάτων, μέσω των υλοποιήσεων των μεταφραστών για τις γλώσσες C, C++ και Fortran και των λειτουργιών που προσφέρει η διεπαφή. Το OpenMP περιγράφεται λεπτομερώς στο επόμενο κεφάλαιο.

Σε ένα κατανεμημένο σύστημα, παράλληλες διεργασίες ανταλλάσουν δεδομένα μέσω μηνυμάτων η μία στην άλλη. Οι επικοινωνίες αυτές μπορεί να είναι ασύγχρονες, δηλαδή όταν ένα μήνυμα μπορεί να σταλεί πριν ο αποδέκτης να είναι έτοιμος ή σύγχρονες, δηλαδή όταν ο αποδέκτης πρέπει να είναι έτοιμος. Το πιο διαδεδομένο πρότυπο είναι το MPI, το οποίο προσφέρει έτοιμες ρουτίνες για την αποστολή και λήψη μηνυμάτων και άλλων λειτουργιών, χωρίς να αναγκάζει τον προγραμματιστή να γνωρίζει τα πάντα για την αρχιτεκτονική και τη διασύνδεση των υπολογιστών.

Στην πράξη, μπορεί να χρησιμοποιούνται ταυτόχρονα διαφορετικά μοντέλα προγραμματισμού και διαφορετικές παράλληλες αρχιτεκτονικές για την επίλυση ενός προβλήματος και κάθε ένα έχει τα πλεονεκτήματα και τις ιδιαιτερότητες του. Το μόνο σίγουρο είναι ότι τα μοντέλα προσπαθούν να είναι εύκολα και κατανοητά στον προγραμματιστή, καθώς η ανάγκη για υλοποίηση παράλληλων αλγορίθμων είναι πλέον η μοναδική λύση για πολλές ανερχόμενες εφαρμογές.

1.3 Αντικειμενο της Διπλωματικης

Ο κύριος στόχος της διπλωματικής εργασίας αυτής είναι η συλλογή, ο χαρακτηρισμός και η κατηγοριοποίηση όλων των μετροπρογραμμάτων που σχετίζονται με το OpenMP. Τα μετροπρογράμματα που δοκιμάζονται εκτός από έλεγχο ορθότητας και μετρήσεις επίδοσης, αναλύουν ειδικές περιπτώσεις και καταστάσεις δύσκολα προβλέψιμες. Επιπλέον, εξετάζουν διάφορες πτυχές του μοντέλου OpenMP που είναι διφορούμενες και ανοιχτές στον τρόπο υλοποίησης, με αποτέλεσμα διαφορετικές προσεγγίσεις να έχουν σημαντικά διαφορετικά αποτελέσματα. Επιπλέον, σκοπός είναι να υπάρχει ένα σύνολο των μετροπρογραμμάτων που σχετίζονται με το OpenMP, τα οποία περιγράφονται με μια πιο γενική σκοπιά. Αυτό βοηθά, όχι μόνο να έχουν μια εικόνα οι προγραμματιστές που υλοποιούν έναν μεταφραστή, αλλά και τους προγραμματιστές

εφαρμογών να γνωρίζουν ποιες αξιοποιήσιμες δυνατότητες υπάρχουν και ποιες τεχνικές είναι αποτελεσματικές για την επίλυση προβλημάτων στα παράλληλα συστήματα. Στο κείμενο αυτό φυσικά, δεν υπάρχουν όλα τα αποτελέσματα των δοκιμών παρά μόνο κάποια αντιπροσωπευτικά για την κάθε σουίτα μετροπρογραμμάτων. Ωστόσο, σκοπός είναι να υπάρξει μια ιστοσελίδα όπου θα είναι συγκεντρωμένα διάφορα αποτελέσματα και μετρήσεις, καθώς και οδηγίες εκτέλεσης των μετροπρογραμμάτων, η οποία ίσως φανεί χρήσιμη σε άλλους.

Ένας ακόμα στόχος είναι η πειραματική μελέτη δημοφιλών και ευρέως διαθέσιμων υλοποιήσεων του OpenMP με βάση τα μετροπρογράμματα αυτά και το πώς αυτοί συμπεριφέρονται σε διαφορετικές αρχιτεκτονικές παράλληλων συστημάτων κοινόχρηστης μνήμης (UMA vs NUMA). Ταυτόχρονα, αποτιμάται και συγκρίνεται ο παραλληλοποιητικός μεταφραστής OMPi, ο οποίος έχει αναπτυχθεί από το Πανεπιστήμιο Ιωαννίνων. Με τη σύγκριση των επιδόσεων μεταξύ των μεταφραστών, μπορούν να βγουν συμπεράσματα σχετικά με τις εκάστοτε υλοποιήσεις και το που χωράν επιπλέον βελτιστοποιήσεις.

1.4 Δομη του κειμενου

Το κείμενο της εργασίας αυτής δομείται ως εξής:

- Στο κεφάλαιο 2 περιγράφεται το μοντέλο OpenMP και κάποιες σημαντικές οδηγίες. Λεπτομέρειες και συγκεκριμένες ιδιότητες αναλύονται κατά την περιγραφή των αντίστοιχων benchmarks.
- Στο κεφάλαιο 3 γίνεται μια περιγραφή του OMPi και των μεταφραστών που χρησιμοποιήθηκαν για την εκτέλεση των μετρήσεων καθώς και των πλατφόρμων εκτέλεσης.
- Στο κεφάλαιο 4 γίνεται μια κατηγοριοποίηση των μετροπρογραμμάτων.
- Στο κεφάλαιο 5 περιγράφεται ο έλεγχος ορθότητας και τα Validation and Verification tests.
- Στο κεφάλαιο 6 τα Microbenchmarks του EPCC τα οποία αποτελούν μια ξεχωριστή κατηγορία.
- Στο κεφάλαιο 7 τα benchmarks γενικού σκοπού. Αναφέρονται ως γενικού σκοπού γιατί οι εφαρμογές που δοκιμάζονται αναλύουν πολλά διαφορετικά workloads και λειτουργίες του OpenMP. Αυτά είναι τα NAS, τα Parsec, τα SPEC OMP2012 και τα Ompscr.

- Στο κεφάλαιο 8 τα benchmarks ειδικού σκοπού. Αναφέρονται ως ειδικού σκοπού γιατί είτε αναλύουν συγκεκριμένες λειτουργίες του OpenMP όπως τα tasks, είτε συγκεκριμένες συσκευές όπως οι επιταχυντές, είτε workloads με συγκεκριμένες ιδιότητες. Αυτά είναι τα BOTS, το UTS, τα Rodinia, και τα MineBench.
- Στο κεφάλαιο 9 γίνεται μια επισκόπηση και μια περιγραφή κάποιων γενικών συμπερασμάτων.

ΚΕΦΑΛΑΙΟ 2
ΤΟ ΠΡΟΤΥΠΟ OPENMP

2.1 ΕΙΣΑΓΩΓΗ

Το OpenMP (open multi-programming) είναι μια διεπαφή (API) που υποστηρίζει κοινόχρηστης μνήμης παράλληλης επεξεργασίας προγραμματισμό στις γλώσσες C, C++ και Fortran, στις περισσότερες πλατφόρμες, αρχιτεκτονικές και λειτουργικά συστήματα. Αποτελείται από οδηγίες προς το μεταφραστή, ρουτίνες βιβλιοθήκης και μεταβλητές περιβάλλοντος που επηρεάζουν τη συμπεριφορά χρόνου εκτέλεσης. Χρησιμοποιεί ένα φορητό, κλιμακώσιμο μοντέλο το οποίο δίνει στους προγραμματιστές μια απλή και ευέλικτη διεπαφή για την ανάπτυξη παράλληλων εφαρμογών.

Οι πρώτες προδιαγραφές εκδόθηκαν τον Οκτώβρη του 1997 για Fortran και τον επόμενο χρόνο για C και C++, ενώ το 2005 ενώθηκαν για τις γλώσσες Fortran, C και C++ με την έκδοση 2.5.

Μέχρι τότε, το OpenMP όριζε απλούς τρόπους παραλληλοποίησης συχνών βρόγχων στους οποίους το πλήθος των επαναλήψεων είναι γνωστό εκ των προτέρων. Αυτός ο περιορισμός οδήγησε σε μια προσπάθεια να εισαχθεί ο παραλληλισμός με *tasks*, εμπνευσμένη από τα αντίστοιχα χαρακτηριστικά των Cilk, X10 και Chapel. Έτσι το 2008 η έκδοση 3.0 έγινε διαθέσιμη και άνοιξε τις δυνατότητες πέρα από παραλληλοποίηση βασικών βρόγχων.

Η έκδοση 4.0 των προδιαγραφών παρουσιάστηκε τον Ιούλιο του 2013 και περιλαμβάνει εκτός των άλλων, υποστήριξη για επιταχυντές και νέες ιδιότητες όπως το *thread affinity*. Η τρέχουσα έκδοση είναι η 5.0 που παρουσιάστηκε το 2018, ενώ να σημειωθεί ότι οι μεταφραστές και τα λειτουργικά ίσως να μην υποστηρίζουν ολόκληρο το σετ των χαρακτηριστικών όπως αυτά περιγράφονται στις τελευταίες εκδόσεις.

2.2 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ OPENMP

Το προγραμματιστικό μοντέλο OpenMP βασίζεται στην παραλληλία με χρήση νημάτων και στηρίζεται στο μοντέλο fork-join. Το κύριο νήμα (master thread) εκτελεί τον κώδικα σειριακά μέχρι να συναντήσει μια παράλληλη περιοχή και να δημιουργήσει μια ομάδα νημάτων, το μέγεθος

της οποίας καθορίζεται από τον προγραμματιστή, είτε δυναμικά, είτε στατικά μέσω μιας μεταβλητής περιβάλλοντος. Η ομάδα των νημάτων εκτελεί παράλληλα την περιοχή και το κύριο νήμα περιμένει την εκτέλεση για να συνεχίσει έπειτα το σειριακό κώδικα. Αυτή η διαδικασία μπορεί να επαναληφθεί και άλλες φορές, όσες και οι περιοχές παραλληλίας που έχουν επισημανθεί από τον προγραμματιστή.

Ορισμένες επιθυμητές ιδιότητες προσφέρονται από το μοντέλο, ενώ κάποιες άλλες οφείλει ο προγραμματιστής να τις ενσωματώσει στο πρόγραμμα μέσω οδηγιών. Αυτές μαζί με τις ρουτίνες βιβλιοθήκης και τις μεταβλητές περιβάλλοντος ορίζουν το προγραμματιστικό μοντέλο του OpenMP και περιγράφονται στη συνέχεια.

2.2.1 Οδηγιες OpenMP για C/C++

Κάθε οδηγία στο OpenMP αποτελείται από 3 μέρη. Αρχικά, πρέπει να ξεκινά υποχρεωτικά με **#pragma omp**. Στη συνέχεια, ακολουθεί το όνομα της οδηγίας, το οποίο μπορεί να δηλώνει για παράδειγμα μια παράλληλη περιοχή, μια υπόδειξη συγχρονισμού κ.α. Το τρίτο μέρος είναι προαιρετικό και αφορά διάφορες *φράσεις* (clauses) στις οποίες ορίζονται κάποιες παράμετροι της εκάστοτε οδηγίας.

2.2.1.1 Περιοχές Παραλληλίας

Η πιο σημαντική οδηγία είναι η **parallel**, η οποία πρέπει να δημιουργήσει μια ομάδα νημάτων για να εκτελέσει μια παράλληλη περιοχή. Όταν το κύριο νήμα συναντήσει μια τέτοια αναθέτει στα νήματα την εργασία και εκείνα την εκτελούν παράλληλα. Εξαιτίας της απρόβλεπτης ταχύτητας εκτέλεσης από τα εκάστοτε νήματα, το OpenMP προσφέρει και συγχρονισμό των νημάτων, υπονοώντας έναν *barrier* στο τέλος της παράλληλης περιοχής. Κάθε νήμα επομένως που φτάνει στο σημείο αυτό οφείλει να περιμένει όλα τα υπόλοιπα νήματα να φτάσουν στο ίδιο σημείο για να καταστραφεί η ομάδα και να συνεχιστεί η εκτέλεση.

Ο χρήστης έχει τη δυνατότητα να ορίσει το μέγεθος της ομάδας, δηλαδή το πλήθος των νημάτων που θα εκτελέσουν μια παράλληλη περιοχή. Μπορεί στατικά να ορίσει το μέγιστο πλήθος νημάτων που θα δημιουργηθούν είτε μέσω της συνάρτησης **omp_set_num_threads()** είτε μέσω της μεταβλητής περιβάλλοντος **OMP_NUM_THREADS**. Μπορεί επίσης δυναμικά να δώσει τιμή στο πλήθος των νημάτων, χρησιμοποιώντας τη φράση **num_threads()** μετά από την οδηγία **parallel**.

2.2.1.2 Περιοχές Διαμοιρασμού Εργασίας

Μια σημαντική δυνατότητα που προσφέρει το OpenMP, είναι ο ορισμός περιοχών διαμερισμού εργασίας για την κατανομή του φόρτου μεταξύ των νημάτων. Εντός μια παράλληλης περιοχής επομένως, ο χρήστης μπορεί να ορίσει τον τρόπο που θα διαμεριστεί η εργασία στα νήματα μέσω τριών οδηγιών:

- **`for`**. Αφορά τον καταμερισμό του βρόγχου επανάληψης for που ακολουθεί. Ο χρήστης έχει τη δυνατότητα να επιλέξει από διάφορους τρόπους κατανομής των επαναλήψεων στα νήματα.
- **`sections`**. Αφορά τον καταμερισμό των τμημάτων **`section`** που δηλώνονται μέσα στην οδηγία, ώστε αυτά να διαμοιραστούν σε διαφορετικά νήματα.
- **`single/master`**. Το πρώτο νήμα (στην περίπτωση single) ή το κύριο νήμα (στην περίπτωση master) που συναντά την οδηγία εκτελεί το τμήμα κώδικα που ακολουθεί με τα υπόλοιπα να την προσπερνούν.

Στο τέλος των περιοχών διαμοιρασμού εργασίας υπονοείται ένα *φράγμα* (barrier), ακριβώς όπως και στην περίπτωση των παράλληλων περιοχών, με σκοπό τον συγχρονισμό των νημάτων. Το φράγμα αυτό μπορεί να παραλειφθεί με την τοποθέτηση της φράσης **`nowait`** στην οδηγία.

2.2.1.3 Οδηγίες συγχρονισμού

Οι οδηγίες συγχρονισμού χρησιμοποιούνται για το συγχρονισμό των νημάτων κατά την παράλληλη εκτέλεση ώστε να εξασφαλίζεται η συνέπεια των δεδομένων και η σωστή λειτουργία του προγράμματος. Οι βασικότερες είναι:

- **`atomic`**. Χρησιμοποιείται για αδιαίρετη πράξη σε μια μεταβλητή. Εξασφαλίζει ότι στη μεταβλητή που πρόκειται να τροποποιηθεί δεν παρεμβάλλεται κάποιο άλλο νήμα.
- **`barrier`**. Ορίζει ένα φράγμα συγχρονισμού των νημάτων στο σημείο που τοποθετείται. Τα νήματα που καταφτάνουν στο σημείο αυτό περιμένουν τα υπόλοιπα της ομάδας, ώστε να συνεχιστεί η εκτέλεση με τις τιμές των κοινόχρηστων μεταβλητών ίδιες για όλα τα νήματα.
- **`critical`**. Παρόμοια με την οδηγία **`atomic`** αλλά για τμήμα κώδικα που πρέπει να εκτελεστεί από ένα νήμα τη φορά. Τα νήματα που φτάνουν στην *κρίσιμη περιοχή* περιμένουν την εκτέλεση από το νήμα που βρίσκεται ήδη σε αυτήν.

- **flush.** Χρησιμοποιείται για να συγχρονίσει τα νήματα της ομάδας και εξασφαλίζει τη συνέπεια της κοινόχρηστης μνήμης, με την έννοια ότι δεν εκκρεμούν εγγραφές σε αυτήν.

2.2.1.4 Φράσεις Οδηγιών

Οι φράσεις οδηγιών τοποθετούνται στο σημείο δήλωσης μιας οδηγίας OpenMP, ακριβώς μετά από το όνομα της οδηγίας και χρησιμοποιούνται για να ρυθμίσουν τον τρόπο εκτέλεσης της. Πιο συγκεκριμένα, ορίζουν τις συνθήκες υπό τις οποίες αυτή θα εκτελεστεί, ή καθορίζουν την συμπεριφορά των νημάτων πριν, κατά τη διάρκεια και μετά την εκτέλεση του τμήματος κώδικα που ακολουθεί την οδηγία. Κάποιες βασικές φράσεις είναι οι εξής:

- **if(***συνθήκη***)**. Εφόσον η συνθήκη είναι αληθής εκτελείται η οδηγία που περιέχει τη φράση.
- **shared/private/firstprivate/lastprivate (***λίστα μεταβλητών***).** Ορίζουν τις μεταβλητές που χρησιμοποιούνται στην παράλληλη περιοχή από τα νήματα ως κοινόχρηστες, ιδιωτικές, ιδιωτικές με αρχική τιμή την τιμή εκτός της παράλληλης περιοχής και ιδιωτικές με την τιμή της μεταβλητής να παραμένει μετά το πέρας της παράλληλης περιοχής.
- **nowait**. Χρησιμοποιείται για να παραλειφθεί το φράγμα που υπονοείται στην αντίστοιχη οδηγία.
- **num_threads()**. Χρησιμοποιείται για να ορίσει το πλήθος των νημάτων που θα συμμετάσχουν σε μια παράλληλη περιοχή.
- **schedule(***τύπος, chunk size***)**. Σε συνδυασμό με την οδηγία **for** ορίζουν την πολιτική διαμοιρασμού των επαναλήψεων του βρόγχου στα νήματα. Έτσι, υπάρχουν πολιτικές **static**, για στατικό διαμοιρασμό του βρόγχου σε τμήματα chunk size ή πλήθους νημάτων, **dynamic** για δυναμικό διαμοιρασμό, με την έννοια ότι το επόμενο τμήμα μεγέθους chunk size θα το αναλάβει όποιο νήμα είναι διαθέσιμο και **guided,** όπου ο διαμοιρασμός γίνεται παρόμοια με το **dynamic** αλλά το μέγεθος του κόκκου μειώνεται εκθετικά. Επίσης, ο προγραμματιστής μπορεί να ελέγξει τις πολιτικές αυτές από τη γραμμή εντολών, δοκιμάζοντας διαφορετικές εκτελέσεις και διαλέγοντας τελικά το καλύτερο schedule.
- **reduction(***πράξη* : *λίστα μεταβλητών).* Χρησιμοποιείται για την εκτέλεση μιας πράξης σε μια κοινόχρηστη μεταβλητή και εξασφαλίζει τη συνέπεια των δεδομένων, χωρίς να απασχολείται με αυτό το κομμάτι ο προγραμματιστής.

2.2.1.5 Tasks

Μια άλλη πολύ σημαντική οδηγία είναι η οδηγία **task**. Αυτή η προσθήκη ενσωματώθηκε στην έκδοση 3.0 για να μην υπάρχει ο περιορισμός της εκτέλεσης παράλληλου κώδικα με στατικό τρόπο. Με τον τρόπο αυτό, ο προγραμματιστής μπορεί να ορίσει μια περιοχή, η οποία μπορεί να εκτελεστεί παράλληλα οποιαδήποτε χρονική στιγμή. Αυτό έχει ως αποτέλεσμα ο κώδικας αυτός να εκτελείται όταν είναι διαθέσιμος κάποιος πυρήνας ή ακόμα να εκτελεστεί παράλληλα με κάποια άλλη παράλληλη περιοχή. Αυτός ο τρόπος παραλληλισμού ταιριάζει σε πολλές εφαρμογές στις οποίες η εξισορρόπηση φόρτου ανάμεσα στα νήματα αποτελεί περιορισμό. Ακόμα, μπορεί να γίνει εναλλαγή των tasks ανάμεσα στα νήματα, δηλαδή ένα νήμα να ξεκίνησε την εκτέλεση και σε κάποιο σημείο να την ανέλαβε κάποιο άλλο. Φυσικά, ο προγραμματιστής μπορεί να ορίσει τις μεταβλητές που χρησιμοποιούνται στο task ως **private, shared** και **firstprivate** σαν clauses στην οδηγία ώστε να επιτυγχάνουν τη σωστή λειτουργία. Η χρήση των **barrier** και **taskwait** βοηθούν το συγχρονισμό των tasks, με την πρώτη να εξασφαλίζει ότι η εκτέλεση όλων των tasks που δημιουργήθηκαν από την τρέχουσα ομάδα νημάτων έχει ολοκληρωθεί και τη δεύτερη να σταματά την εκτέλεση ενός task μέχρι να ολοκληρωθούν τα "παιδιά" που αυτό δημιούργησε.

2.2.1.6 Devices

Με την έκδοση 4.0 και έπειτα υπάρχει η δυνατότητα αποστολής κώδικα για εκτέλεση σε συσκευές εκτός της CPU όπως συνεπεξεργαστές, επιταχυντές και κάρτες γραφικών. Η CPU παίζει το ρόλο του *"host"* και η συσκευή αναφέρεται ως *"device"*. Η οδηγία που χρησιμοποιείται είναι η **#pragma omp target** και μαζί με τις φράσεις **to, from, tofrom,** και **alloc,** ο προγραμματιστής δηλώνει ποια ενέργεια θέλει να πραγματοποιήσει. Με το **to** μπορεί να αντιγραφεί η τιμή μιας μεταβλητής στη συσκευή πριν την εκτέλεση για αρχικοποίηση και με το **from** για να αντιγραφεί το αποτέλεσμα μετά την εκτέλεση πίσω στον host. Το **tofrom** είναι συνδυασμός των δύο, ενώ το **alloc** χρησιμοποιείται για αντιστοίχιση των δεδομένων χωρίς να γίνεται μεταφορά. Για να υπάρχει πρόσβαση σε καθολικές μεταβλητές και συναρτήσεις από τον κώδικα που αποστέλλεται στις συσκευές αυτές, πρέπει να δηλωθούν σε μια περιοχή που ορίζεται από τις **declare/end declare data**. Με την έκδοση 4.5 προστέθηκαν επιπλέον λειτουργίες, όσον αφορά τις συσκευές, και οι δυνατότητες επεκτείνονται πέρα από την απλή περιγραφή που δόθηκε.

2.2.2 Συναρτησεις βιβλιοθηκης OpenMP

Το OpenMP προσφέρει ορισμένες συναρτήσεις που μπορεί να χρησιμοποιήσει ο προγραμματιστής από τη βιβλιοθήκη του χρόνου εκτέλεσης (**omp.h**). Αυτές χρησιμοποιούνται για να ορίσουν δυναμικά το πλήθος των νημάτων με βάση τους διαθέσιμους πόρους, για συγχρονισμό των νημάτων με χρήση κλειδαριών, για ερωτήσεις στο σύστημα, όπως το πλήθος των επεξεργαστών, για χρονομετρήσεις κ.α. Οι πιο βασικές είναι:

- **omp_get_num_threads**: Επιστρέφει το πλήθος των νημάτων που χρησιμοποιούνται.
- **omp_set_num_threads**: Ορίζει το πλήθος των νημάτων που θα εκτελέσουν μια παράλληλη περιοχή (υπερισχύει της μεταβλητής περιβάλλοντος **OMP_NUM_THREADS**).
- **omp_get_thread_num**: Επιστρέφει τον αριθμό του τρέχοντος νήματος (μεταξύ 0 και #νημάτων -1).
- **omp_get_num_procs**: Επιστρέφει το πλήθος των διαθέσιμων πυρήνων που βλέπει το σύστημα.
- **omp_init_lock**: Αρχικοποιεί μια απλή κλειδαριά.
- **omp_destroy_lock**: Αφαιρεί μια κλειδαριά.
- **omp_set_lock**: Το νήμα περιμένει να γίνει διαθέσιμη η κλειδαριά.
- **omp_unset_lock**: Παραχωρεί τη διαθεσιμότητα της κλειδαριάς.
- **omp_get_wtime**: Επιστρέφει το χρόνο που έχει περάσει από κάποια στιγμή στο παρελθόν (δεν αλλάζει κατά το χρόνο εκτέλεσης) και χρησιμοποιείται για χρονομετρήσεις.
- **omp_set_nested**: Ενεργοποιεί *nested* παραλληλισμό.

2.2.3 Μεταβλητεσ περιβαλλοντοσ

Σε συστήματα UNIX ο χρήστης μπορεί να δώσει τιμές σε διάφορες παραμέτρους μέσω των μεταβλητών περιβάλλοντος από το UNIX shell ελέγχοντας διάφορα χαρακτηριστικά της εκτέλεσης. Η πιο χαρακτηριστική είναι η **OMP_NUM_THREADS** μέσω της οποίας δηλώνεται το πλήθος των νημάτων που θα δημιουργηθούν σε κάθε παράλληλη περιοχή, εφόσον δεν δηλώνεται μέσα στο πρόγραμμα. Η μεταβλητή **OMP_NESTED[=TRUE|=FALSE]** ορίζει αν είναι ενεργοποιημένος ο nested παραλληλισμός (εκτός αν χρησιμοποιείται η **omp_set_nested()**). Η **OMP_SCHEDULE[=type[,size]]** τροποποιεί τη συμπεριφορά του **schedule** clause όταν έχει οριστεί ως **runtime** σε ένα βρόγχο for. Άλλες μεταβλητές μπορεί να αφορούν

συσκευές, διάφορες πολιτικές όπως τι κάνουν τα νήματα όταν περιμένουν δουλειά, steal policy κ.α.

Μια σημαντική παράμετρος, η οποία θα παίξει καθοριστικό ρόλο και στο πειραματικό κομμάτι της εργασίας, αφορά την τοποθέτηση των νημάτων. Το OpenMP δίνει τη δυνατότητα στον χρήστη να επιλέξει σε ποιον επεξεργαστικό πυρήνα θα εκτελείται το κάθε νήμα μέσω δύο μεταβλητών: **OMP_PROC_BIND** και **OMP_PLACES**. Μέσω της **OMP_PLACES** προσδιορίζονται οι διαθέσιμοι επεξεργαστικοί πόροι. Έτσι, ανάλογα με την τιμή της μεταβλητής ο χρήστης μπορεί να ομαδοποιήσει τους πόρους. Η τιμή που μπορεί να πάρει η μεταβλητή μπορεί να είναι είτε **threads**, που σημαίνει ότι κάθε *place* αντιστοιχεί σε κάθε νήμα υλικού (hardware thread), είτε **cores**, που σημαίνει ότι κάθε place αντιστοιχεί σε κάθε επεξεργαστικό πυρήνα, είτε **sockets**, που σημαίνει ότι κάθε place αντιστοιχεί σε κάθε socket (αποτελούμενο από ένα ή περισσότερα cores). Επίσης, ο χρήστης μπορεί να ομαδοποιήσει με δικό του τρόπο τους επεξεργαστικούς πόρους μέσω λιστών, κάθε μία από τις οποίες αντιστοιχεί σε ένα place. Η μεταβλητή **OMP_PROC_BIND** ελέγχει πώς τα OpenMP νήματα "δένονται" στους διαθέσιμους πόρους (με βάση τα places) παίρνοντας τον έλεγχο από το λειτουργικό σύστημα. Η τιμή που μπορεί να πάρει η μεταβλητή είναι είτε **spread**, που σημαίνει ότι τα νήματα εκτέλεσης "δένονται" όσο το δυνατόν πιο ομοιόμορφα, είτε **close**, που σημαίνει ότι τα νήματα "δένονται" κοντά στο κύριο (master) νήμα, είτε **master**, που σημαίνει ότι "δένονται" τα νήματα στο ίδιο place με το κύριο νήμα της ομάδας. Για παράδειγμα, σε ένα σύστημα με δύο sockets και 16 cores συνολικά η τοποθέτηση των νημάτων αν η μεταβλητή **OMP_PLACES** πάρει την τιμή **sockets** θα είναι το πρώτο νήμα να τρέξει στο socket 0 το δεύτερο στο socket 1 το τρίτο στο 0 κ.ο.κ. Αν τώρα η **OMP_PLACES** πάρει την τιμή **cores** και η **OMP_PROC_BIND=close** το πρώτο νήμα θα πάει στο core 0 που βρίσκεται στο socket 0, το δεύτερο στο core 1 που βρίσκεται στο socket 0 κ.ο.κ. μέχρι και το έβδομο που θα πάει στο core 7 στο socket 0 και το όγδοο στο core 8 του socket 1 κ.ο.κ.

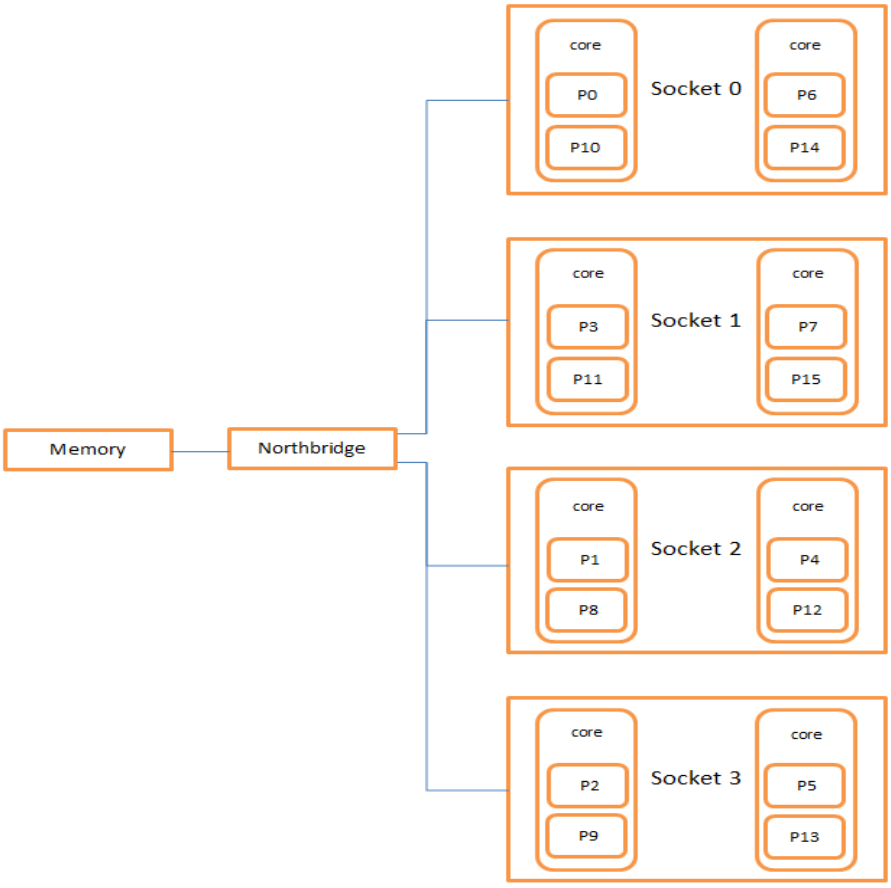
ΚΕΦΑΛΑΙΟ 3
ΠΕΡΙΒΑΛΛΟΝ ΕΚΤΕΛΕΣΗΣ ΠΕΙΡΑΜΑΤΩΝ

3.1 ΠΛΑΤΦΟΡΜΕΣ ΕΚΤΕΛΕΣΗΣ

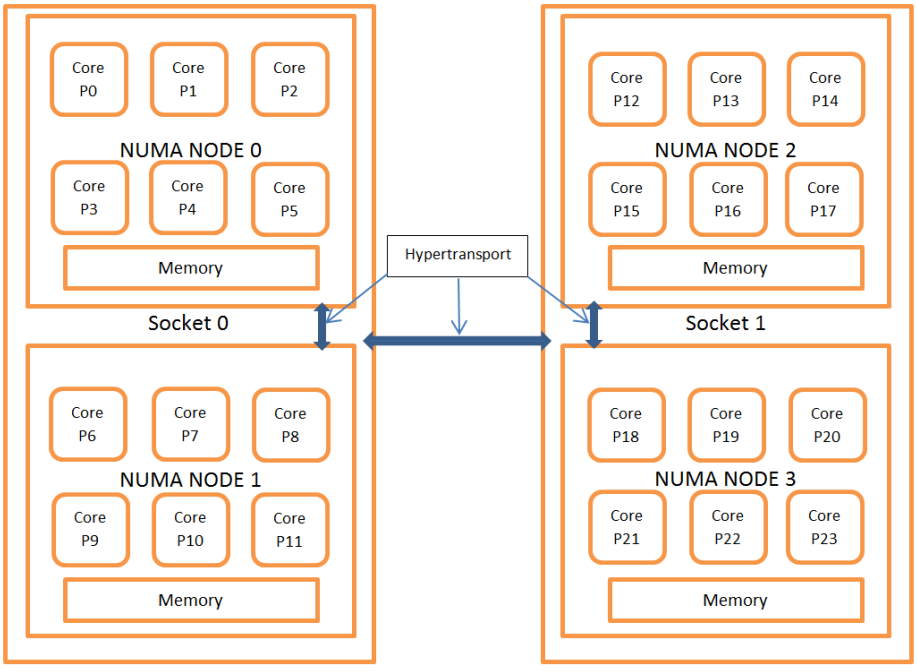
Οι πλατφόρμες εκτέλεσης, στις οποίες έγιναν οι μετρήσεις, είναι δύο πολυπύρηνοι υπολογιστές που ανήκουν στο Parallel Processing Group του Πανεπιστημίου Ιωαννίνων. Ο πρώτος είναι η Paraguay και αποτελεί έναν SMP διακομιστή τύπου SR6850HW4. Διαθέτει τέσσερις θέσεις επεξεργαστών (sockets), οι οποίες φιλοξενούν επεξεργαστές Intel Xeon 7040 Paxville MP, ο καθένας εκ των οποίων περιλαμβάνει δύο hyperthreaded cores των 3GHz. Η προσπέλαση της κύριας μνήμης γίνεται μέσω του ελεγκτή μνήμης (Northbridge), ο οποίος συνδέεται με τα sockets όπως φαίνεται στο Σχήμα 3.1. Κάθε core αποτελείται από δύο νήματα υλικού τα οποία έχουν πρόσβαση σε δύο επίπεδα κρυφής μνήμης (L1d των 16KB και L2 των 2048KB). Η συνολική διαθέσιμη μνήμη είναι 8Gb, και το λειτουργικό σύστημα Debian GNU/Linux 9.7 (stretch). Το δεύτερο μηχάνημα στο οποίο έγιναν οι μετρήσεις είναι ο Paragon. Ο Paragon διαθέτει δύο επεξεργαστές AMD Opteron 6166, καθένας εκ των οποίων αποτελείται από δύο NUMA nodes των 6 πυρήνων στα 1.8GHz. Κάθε core έχει πρόσβαση τοπικά σε δύο επίπεδα κρυφής μνήμης (L1i των 64KB, L1d των 64KB και L2 των 512 KB), ενώ οι 6 πυρήνες που βρίσκονται σε κάθε NUMA node μοιράζονται την τρίτου επιπέδου cache (L3 των 5MB) και το κομμάτι της κύριας μνήμης που είναι αφοσιωμένο σε αυτούς (4GB). Η επικοινωνία μεταξύ των κόμβων μέσα στο chip αλλά και μεταξύ των sockets γίνεται μέσω Hypertrasnport καναλιών επικοινωνίας όπως φαίνεται χονδρικά στο Σχήμα 3.2. Η συνολική διαθέσιμη μνήμη είναι 16Gb και το λειτουργικό σύστημα Debian GNU/Linux 9.9 (stretch).

Τα δύο αυτά μηχανήματα αντιπροσωπεύουν τις δύο πιο σύνηθες κατηγορίες εμπορικών παράλληλων συστημάτων (SMPs vs NUMA). Η τοπολογία των επεξεργαστών παίζει καθοριστικό ρόλο στην επίδοση των προγραμμάτων και διαφορετικές αρχιτεκτονικές είναι κατάλληλες για συγκεκριμένα workloads, εξαιτίας της φύσης των δεδομένων που διαχειρίζονται. Επομένως, η επιλογή της κατάλληλης αρχιτεκτονικής, όσον αφορά την εξέλιξη των πολυπύρηνων συστημάτων κοινόχρηστης μνήμης, εξαρτάται και από τη φύση των σύγχρονων προβλημάτων που καλούνται να φέρουν εις πέρας.

Σχήμα 3.1: Τοπολογία της Paraguay (ως P# αναφέρονται τα νήματα υλικού)



Σχήμα 3.2: Τοπολογία του Paragon (ως P# αναφέρονται τα νήματα υλικού)



3.2 Μεταφραστεσ OpenMP

Το OpenMP έχει πολλές υλοποιήσεις, τόσο εμπορικού όσο και ερευνητικού σκοπού. Οι πιο γνωστοί προμηθευτές, σύμφωνα με την επίσημη ιστοσελίδα του OpenMP (updated 16 September 2019), είναι οι παρακάτω:

- **AMD**. Ο AOMP είναι ένας ανοιχτού κώδικα LLVM/Clang based μεταφραστής, ο οποίος υποστηρίζει το μοντέλο OpenMP καθώς και offloading σε Radeon GPUs.
- **ARM**. Ο εμπορικός μεταφραστής της ARM υποστηρίζει το OpenMP 3.1, καθώς και όλες τις προδιαγραφές - εκτός από offloading σε συσκευές - των εκδόσεων 4.0/4.5 για τις γλώσσες C και C++ για 64-bit Armv8-A αρχιτεκτονική. Για τη γλώσσα Fortran υποστηρίζει την έκδοση 3.1 καθώς και κάποιες προδιαγραφές των 4.0/4.5.
- **Barcelona Supercomputing Center**. Ο Mercurium είναι ένας ερευνητικός μεταφραστής, ελεύθερα διαθέσιμος, που υποστηρίζει το OpenMP 3.1 και όποια χαρακτηριστικά σχετίζονται με tasks των μετέπειτα εκδόσεων, για τις γλώσσες C, C++ και Fortran.
- **Cray**. Ο μεταφραστής CEE είναι εμπορικού σκοπού και υποστηρίζει το OpenMP 4.5 για τις γλώσσες C, C++ και Fortran.
- **Flang**. Ο μεταφραστής Flang είναι ελεύθερα διαθέσιμος και υποστηρίζει το OpenMP 4.5 για τη γλώσσα Fortran, σε συστήματα Linux/x86-64, Linux/ARM, Linux/OpenPOWER.
- **GNU**. Ο μεταφραστής GCC είναι ελεύθερα διαθέσιμος και ανοιχτού κώδικα και υποστηρίζει την έκδοση 4.5 του OpenMP 4.5 (GCC 6.1) για τις γλώσσες C και C++, καθώς και για την γλώσσα Fortran φτάνοντας μέχρι και το OpenMP 4.0 (GCC 4.9.1).
- **IBM**. Οι μεταφραστές XL C/C++ for Linux V16.1.1 και XL Fortran for Linux V16.1.1 είναι εμπορικού σκοπού και υποστηρίζουν το OpenMP 4.5 για POWER9 αρχιτεκτονική και offloading σε NVIDIA GPUs.
- **Intel**. Ο μεταφραστής της Intel υποστηρίζει το OpenMP με τις εκδόσεις 17.0 και έπειτα για τις γλώσσες C, C++ και Fortran και για λειτουργικά συστήματα Windows, Linux και MacOS, καθώς είναι και ελεύθερα διαθέσιμος για ακαδημαϊκή χρήση.
- **LLNL Rose Research Compiler**. Ο Rose είναι ένας ερευνητικός ελεύθερα διαθέσιμος μεταφραστής για τις γλώσσες C, C++ και Fortran, που υποστηρίζει ορισμένα χαρακτηριστικά του OpenMP 4.0, στοχεύοντας σε NVIDIA GPUs.

- **LLVM**. Ο μεταφραστής clang είναι ανοιχτού κώδικα για τις γλώσσες C και C++ και υποστηρίζει με την έκδοση 7.0 το OpenMP 4.5 με τα offloading constructs να τρέχουν στο host.
- **Oracle**. Οι μεταφραστές Oracle Developer Studio 12.6 (C, C++ και Fortran) υποστηρίζουν το OpenMP 4.0 για λειτουργικά συστήματα Oracle Solaris και Linux και είναι ελεύθερα διαθέσιμοι.
- **PGI**. Ο PGI είναι ελεύθερα διαθέσιμος και υποστηρίζει το OpenMP 4.5 (NVIDIA GPUs) για τις γλώσσες C, C++ και Fortran σε συστήματα Linux/x86-64 και Linux/OpenPOWER.

Οι μεταφραστές που επιλέχθηκαν τελικά για τα πειράματα της εργασίας είναι ο μεταφραστής GCC (έκδοση 6.3) της GNU και ο μεταφραστής της Intel (έκδοση 19.0.2). Η επιλογή έγινε με βάση τα παρακάτω χαρακτηριστικά:

- Είναι εξαιρετικά δημοφιλείς και υλοποιούν πλήρως τις προδιαγραφές του OpenMP.
- Είναι ελεύθερα διαθέσιμοι (ο Intel για ακαδημαϊκή χρήση).
- Σημαντικά τμήματα τους είναι ανοιχτού κώδικα, με ολόκληρο τον GCC αλλά και το σύστημα runtime του Intel.
- Υποστηρίζουν τα συνήθη συστήματα –x86, x64 αρχιτεκτονικές και Linux λειτουργικό σύστημα.

Προσπαθήσαμε να είμαστε αντικειμενικοί σε όλα τα πειράματα και τις μετρήσεις ανάμεσα στους μεταφραστές που δοκιμάστηκαν, ώστε να μην αδικήσουμε κάποιον. Ένα παράδειγμα είναι η τοποθέτηση των νημάτων (thread affinity), η οποία έχει σημαντικές επιπτώσεις στο χρόνο εκτέλεσης. Ανάλογα την αρχιτεκτονική αλλά και τη φύση των μετροπρογραμμάτων, διαφορετικές πολιτικές τοποθέτησης των νημάτων έχουν διαφορά στους χρόνους εκτέλεσης. Επίσης, οι υλοποιήσεις παίζουν ρόλο, επομένως σε κάθε μεταφραστή αντιστοιχεί και μια πολιτική κατάλληλη για συγκεκριμένη αρχιτεκτονική και workload. Επομένως λαμβάνοντας υπόψη τα παραπάνω, οι μετρήσεις έγιναν με βάση την καλύτερη πολιτική για τον καθένα.

Επίσης, δεν χρησιμοποιήθηκαν επιπλέον βελτιστοποιήσεις για κάποιον μεταφραστή, εκτός από βασικές όπως η `-O3`, καθώς σκοπός ήταν μια γενική εικόνα των benchmarks και μας ενδιέφερε η εξ ορισμού (default) συμπεριφορά των μεταφραστών στα συστήματα που δοκιμάστηκαν. Οι δοκιμές έγιναν με 1, 2, 4 και 8 νήματα στην Paraguay, καθώς από εκεί και έπειτα δεν περιμένουμε σημαντική βελτίωση στις επιδόσεις αφού τα 16 hyperthreads πολυπλέκονται σε 8 πυρήνες. Στον Paragon οι δοκιμές έγιναν

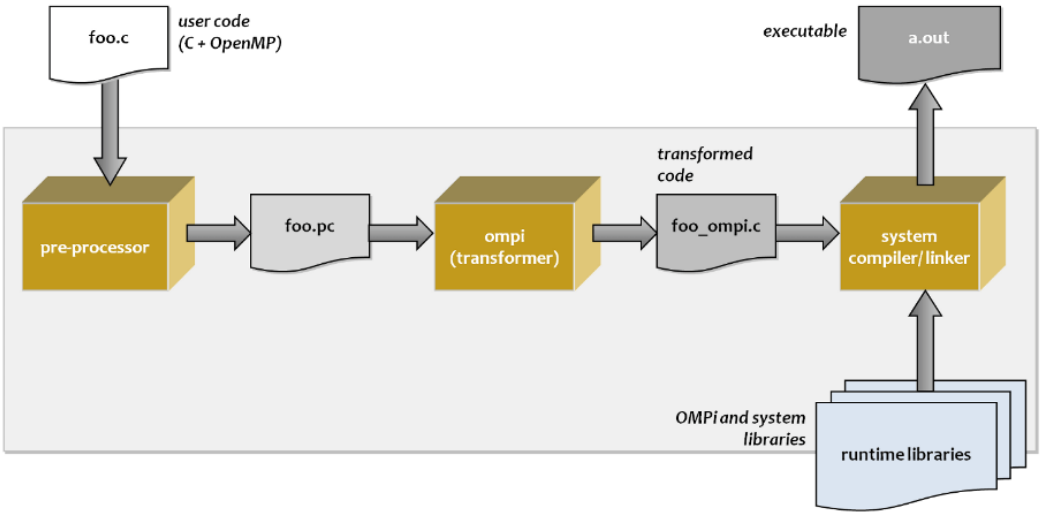
με 1, 2, 6, 12 και 24 νήματα, όσα και τα διαθέσιμα cores. Επομένως σε ένα "embarrassingly parallel" πρόγραμμα περιμένουμε μέγιστο speedup στα 8 νήματα στην Paraguay και στα 24 στον Paragon. Τέλος, όσων αφορά τον OMPi, ο μεταφραστής που χρησιμοποιήθηκε τις περισσότερες φορές ως back-end είναι ο GCC 6.3, ενώ ορισμένες φορές χρησιμοποιήθηκε και ο Intel 19.0.2.

3.3 Ο ΜΕΤΑΦΡΑΣΤΗΣ OMPi

Όπως προαναφέρθηκε, ένας από τους στόχους των πειραμάτων ήταν η δοκιμή και η συγκριτική μελέτη του παραλληλοποιητικού μεταφραστή OMPi. Ο OMPi [1] λαμβάνει ένα πρόγραμμα γραμμένο σε γλώσσα C με οδηγίες OpenMP και με τη βοήθεια του συντακτικού αναλυτή παράγει προγράμματα στα οποία έχουν αντικατασταθεί οι οδηγίες με κλήσεις συναρτήσεων της βιβλιοθήκης του χρόνου εκτέλεσης. Σε επόμενο στάδιο, ο τοπικός μεταφραστής του συστήματος αναλαμβάνει την παραγωγή του τελικού εκτελέσιμου. Η διαδικασία παραγωγής εκτελέσιμου κώδικα φαίνεται στο Σχήμα 3.3.

Η βιβλιοθήκη του χρόνου εκτέλεσης, εκτός των άλλων, προσφέρει συναρτήσεις για τη δημιουργία νημάτων για την εκτέλεση του παραλληλοποιημένου κώδικα, συναρτήσεις που διαχειρίζονται κλειδαριές και βοηθητικές συναρτήσεις συγχρονισμού, όπως επίσης και συναρτήσεις που διαχειρίζονται την επικοινωνία με συσκευές.

Πιο συγκεκριμένα, η διαδικασία της μετάφρασης ξεκινά με τη συντακτική ανάλυση. Κατά την ανάγνωση του προγράμματος δημιουργείται ένα αφηρημένο συντακτικό δέντρο. Στη συνέχεια, το παραγόμενο δέντρο δίνεται σαν είσοδος στο επόμενο στάδιο, κατά το οποίο διασχίζεται κάθε κόμβος του και αντικαθίσταται κάθε οδηγία OpenMP με την αντίστοιχη κλήση συνάρτησης της βιβλιοθήκης του χρόνου εκτέλεσης. Το μετασχηματισμένο πλέον δέντρο χρησιμοποιείται για την παραγωγή του κώδικα σε C, έτοιμο για είσοδο στον τοπικό μεταφραστή του συστήματος, ο οποίος μαζί με τη σύνδεση (linking) με τη βιβλιοθήκη του χρόνου εκτέλεσης ολοκληρώνει τη διαδικασία παραγωγής εκτελέσιμου κώδικα.



Σχήμα 3.3 Διαδικασία παραγωγής εκτελέσιμου κώδικα με τον μεταφραστή OMPi

Το σύστημα του χρόνου εκτέλεσης είναι υπεύθυνο για την επεξεργασία των υπολογιστικών οντοτήτων (τα νήματα για παράδειγμα) που θα εκτελέσουν τις παράλληλες περιοχές. Αποτελείται από δύο τμήματα τα οποία είναι ανεξάρτητα μεταξύ τους, το ORT το οποίο διαχειρίζεται τις υπολογιστικές οντότητες, και το EELIB, το οποίο τις υλοποιεί. Κατά την έναρξη του προγράμματος, εισάγεται η συνάρτηση **`ort_initiallize()`**, μέσω της κλήσης της οποίας το κύριο νήμα αναλαμβάνει την αρχικοποίηση ορισμένων σημαντικών παραμέτρων με βάση και των τιμών των μεταβλητών περιβάλλοντος. Έπειτα, καλείται η αντίστοιχη συνάρτηση αρχικοποίησης του τμήματος EELIB, η **`ee_initialize()`**. Αυτή δημιουργεί τις υπολογιστικές οντότητες. Αν η επιθυμητή λειτουργία είναι η δημιουργία των νημάτων αμέσως και η παραμονή τους σε αδράνεια μέχρι την ανάθεση εργασίας σε αυτά, τότε η υλοποίηση το υποστηρίζει. Στο σημείο αυτό, ας αναφερθεί παρενθετικά ότι υπάρχουν κι άλλες εναλλακτικές, όπως η δημιουργία των νημάτων με το που συναντηθεί μια παράλληλη περιοχή και η καταστροφή τους με το πέρας αυτής, ή η παραμονή τους σε ένα pool, οι οποίες είναι υλοποιημένες στον OMPi. Η καλύτερη επιλογή μιας τέτοιας πολιτικής δεν πρέπει να αφορά τον προγραμματιστή, μπορεί όμως να εξαρτάται από την αρχιτεκτονική του συστήματος και από το λειτουργικό σύστημα που έχει στη διάθεση του. Η τελευταία συζήτηση είναι ένα πολύ καλό παράδειγμα της χρησιμότητας των μετροπρογραμμάτων κατά την ανάπτυξη μεταφραστών, καθώς με την αξιοποίηση τους μπορούν να ληφθούν αποφάσεις τέτοιου είδους. Η διεπαφή των δύο αυτών τμημάτων του χρόνου εκτέλεσης γίνεται μέσω

συναρτήσεων. Το τμήμα ORT δε γνωρίζει τον τύπο και το είδος των υπολογιστικών οντοτήτων, μπορεί όμως να τις διαχειρίζεται μέσω συναρτήσεων που προσφέρει το τμήμα EELIB.

Το τμήμα EELIB του OMPi χρησιμοποιεί έναν αριθμό διαφορετικών βιβλιοθηκών που προσφέρουν υπολογιστικές οντότητες με ποικίλες ιδιότητες. Όταν το τμήμα ORT ζητήσει έναν συγκεκριμένο αριθμό νημάτων για παράδειγμα, το τμήμα EELIB πρέπει να αποφασίσει αν η αίτηση είναι εφικτή, ανάλογα με τις βιβλιοθήκες που έχει στη διάθεση του. Οι βιβλιοθήκης νημάτων POSIX διακρίνονται σε PTHREADS και PTHREADS1 και διαφοροποιούνται στην ιδιότητα υποστήριξης παραλληλισμού πολλαπλών επιπέδων (nested parallelism). Η βιβλιοθήκη νημάτων επιπέδου χρήστη είναι η PSTHREAD, τα νήματα της οποίας τρέχουν σε εικονικούς επεξεργαστές και η αντιστοίχιση τους σε φυσικούς γίνεται μέσω της βιβλιοθήκης. Λόγω της διεπαφής ανάμεσα στα τμήματα του συστήματος runtime και της ανεξαρτησίας τους, είναι εύκολο κάποιος να αναπτύξει και να προσαρτήσει στον OMPi μια δικιά του βιβλιοθήκη οντοτήτων.

ΚΕΦΑΛΑΙΟ 4
ΚΑΤΗΓΟΡΙΕΣ ΜΕΤΡΟΠΡΟΓΡΑΜΜΑΤΩΝ

Τα μετροπρογράμματα που μελετήσαμε στη διπλωματική εργασία αυτή μπορούν να κατηγοριοποιηθούν με πολλούς τρόπους, ανάλογα με τις εφαρμογές που χρησιμοποιούν, τα συστήματα στα οποία είναι προσανατολισμένα κλπ. Εμείς τα κατηγοριοποιήσαμε σε τρεις μεγάλες κατηγορίες: προγράμματα ελέγχου ορθότητας, microbenchmarks και σουίτες εφαρμογών για μέτρηση επιδόσεων.

4.1 ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ

Η πρώτη κατηγορία προγραμμάτων είναι αυτά που στοχεύουν στον έλεγχο ορθότητας. Ο έλεγχος ορθότητας είναι απαραίτητος κατά την ανάπτυξη των μεταφραστών και τον έλεγχο της σωστής λειτουργίας μιας υλοποίησης του OpenMP, σύμφωνα με τις προδιαγραφές που αυτό ορίζει. Εξαιτίας της απρόβλεπτης συμπεριφοράς των νημάτων, είναι δύσκολο να βρεθεί μια συγκεκριμένη μεθοδολογία εύρεσης σφαλμάτων στις υλοποιήσεις. Επομένως, οι συνεχείς δοκιμές διαφόρων τεστ είναι μονόδρομος. Εκτός από τα προγράμματα που έχουν αναπτυχθεί για το σκοπό αυτό και περιγράφονται στο κεφάλαιο 5, οι περισσότερες σουίτες διαθέτουν και κάποιο μηχανισμό επικύρωσης της εξόδου της εκάστοτε εφαρμογής. Συνήθως αυτό γίνεται μέσω δεδομένων που περιλαμβάνουν, τα οποία συγκρίνονται με την έξοδο. Στα πειράματα που έγιναν, όπου ήταν διαθέσιμη κάποιας μορφής επικύρωση, χρησιμοποιήθηκε για τον έλεγχο του αποτελέσματος. Στην κατηγορία προγραμμάτων ελέγχου ορθότητας ανήκουν τα OpenMP Validation Suite V 3.1 και OpenMP Validation and Verification (έκδοση 4.5).

Τα microbenchmarks είναι μικρές εφαρμογές που σκοπό έχουν να φέρουν στην επιφάνεια τυχόν καθυστερήσεις που προκύπτουν από την υλοποίηση των μεταφραστών. Στην ουσία, δεν μετράνε γενική καθυστέρηση, αλλά τεχνητά δοκιμάζουν συγκεκριμένες δομές και λειτουργίες σε ακραίες καταστάσεις, προκειμένου να αποκαλύψουν καθυστερήσεις (*overheads*) που πιθανώς αθροιζόμενες να επηρεάσουν την τελική ταχύτητα των εφαρμογών που τις χρησιμοποιούν. Οι καθυστερήσεις αυτές αποτελούν πολύ σημαντικό παράγοντα για ορισμένες εφαρμογές και θα πρέπει να λαμβάνονται υπόψη τόσο κατά την ανάπτυξη των μεταφραστών, όσο και

από τους προγραμματιστές εφαρμογών, ώστε να διαλέγουν/τροποποιούν τους παράλληλους αλγορίθμους κατάλληλα. Τα μοναδικά διαθέσιμα μετροπρογράμματα αυτής της κατηγορίας που σχετίζονται με το OpenMP περιλαμβάνονται στο EPCC OpenMP microbenchmark suite το οποίο συναντάμε στο κεφάλαιο 6.

Οι σουίτες εφαρμογών για μέτρηση επιδόσεων, εν συντομία benchmarks, αποτελούνται από διάφορες εφαρμογές οι οποίες έχουν επιλεγεί εξαιτίας των χαρακτηριστικών τους, για να ανταποκριθούν στους στόχους της εκάστοτε σουίτας. Διακρίνονται χονδρικά σε γενικού και ειδικού σκοπού. Αυτή η διάκριση γίνεται με τη λογική του ότι κατά την ανάπτυξη συγκεκριμένων προδιαγραφών του OpenMP, είναι χρήσιμο να γνωρίζει ο προγραμματιστής τί μετροπρογράμματα έχει στη διάθεση του για να ελέγξει την υλοποίηση του.

Στα benchmarks γενικού σκοπού, τα οποία περιγράφονται στο κεφάλαιο 7, ανήκουν τα NAS (ενότητα 7.1), τα οποία δοκιμάζουν τη συμπεριφορά ολοκληρωμένων εφαρμογών και συχνών *πυρήνων εφαρμογών*, δηλαδή πολύπλοκων σε υπολογισμούς και χρονοβόρων επαναληπτικών τμημάτων που αποτελούν μέρος πολλών εφαρμογών, και σκοπό έχουν τη σύγκριση της απόδοσης παράλληλων πολυπύρηνων συστημάτων. Τα Parsec (ενότητα 7.2) έχουν αναπτυχθεί με σκοπό να δείξουν πως συμπεριφέρονται γενικά τα παράλληλα συστήματα κοινόχρηστης μνήμης. Δοκιμάζοντας σύγχρονες, αλλά και μελλοντικής σημασίας εφαρμογές, θέλουν να δήξουν την κατεύθυνση προς την οποία χρειάζεται να οδηγηθούν τα παράλληλα συστήματα κοινόχρηστης μνήμης, κυρίως όσον αφορά την αρχιτεκτονική τους, αλλά και τα διαφορετικά μοντέλα προγραμματισμού και τις υλοποιήσεις αυτών, σύμφωνα με τις ανάγκες της εποχής. Τα SPEC έχουν αναπτυχθεί για σύγκριση επιδόσεων σε συστήματα και σκοπό έχουν να αξιολογήσουν γενικά τη συμπεριφορά ενός παράλληλου συστήματος κοινόχρηστης μνήμης και των μεταφραστών του OpenMP, μέσω των SPEC OMP (ενότητα 7.3). Μέσω συγκεκριμένων οδηγιών εκτέλεσης και εργαλείων, παρέχουν ένα αντικειμενικό περιβάλλον για να συγκριθεί η συμπεριφορά διαφορετικών συστημάτων και τμημάτων τους (αρχιτεκτονικές, μεταφραστές, κατανάλωση ενέργειας). Το Ompscr (ενότητα 7.4) έχει αναπτυχθεί ως μια συλλογή μετροπρογραμμάτων κοντά στους προγραμματιστές που αναπτύσσουν μεταφραστές για το OpenMP και σκοπό είχε να εμπλουτίσει τις υπάρχουσες σουίτες με στοιχεία που έλειπαν, όσον αφορά τις προδιαγραφές του OpenMP.

Στην κατηγορία ειδικού σκοπού (κεφάλαιο 8), ανήκουν τα BOTS και το UTS, οι εφαρμογές των οποίων είναι προσανατολισμένες γύρω από τα tasks και το διαμοιρασμό φόρτου εργασίας. Αυτά περιγράφονται λεπτομερώς στις ενότητες 8.1 και 8.2. Τα rodinia (ενότητα 8.3) έχουν αναπτυχθεί για να συγκρίνουν επιδόσεις ανάμεσα σε συμβατικούς πολυπύρηνους επεξεργαστές και επιταχυντές. Σκοπός τους είναι να αναδείξουν πλεονεκτήματα και περιορισμούς της σύγχρονης τάσης στα παράλληλα συστήματα, που είναι η χρήση επιταχυντών (για παράδειγμα GPUs) με τις υλοποιήσεις των εφαρμογών τους να είναι χρήσιμες για την ανάπτυξη υποστήριξης χαρακτηριστικών των πιο πρόσφατων εκδόσεων του OpenMP. Τα minebench (ενότητα 8.4) θίγουν ένα ζήτημα: το κατά πόσο οι τωρινές αρχιτεκτονικές είναι κατάλληλες για εφαρμογές εξόρυξης δεδομένων, ένας τομέας που έχει αποκτήσει ιδιαίτερη σημασία στη σύγχρονη εποχή. Σκοπός είναι να δοκιμαστούν κλασικοί αλγόριθμοι εξόρυξης δεδομένων, παραλληλοποιημένοι με OpenMP και είναι χρήσιμα κυρίως σε προγραμματιστές τέτοιων εφαρμογών που αναζητούν λύσεις όσον αφορά την ταχύτητα εκτέλεσης τους.

4.2 Πληρης Καταλογος Μετροπρογραμματων

Ο Πίνακας 4.1 δίνει μια συγκεντρωτική αναπαράσταση των χαρακτηριστικών των μετροπρογραμμάτων που σχετίζονται με το OpenMP και τα οποία περιγράφονται λεπτομερώς στα επόμενα κεφάλαια. Οι στήλες αφορούν τη γλώσσα στην οποία έχουν αναπτυχθεί οι εφαρμογές, τις πλατφόρμες για τις οποίες αναπτύχθηκαν, τις προδιαγραφές του OpenMP γύρω από τις οποίες είναι προσανατολισμένα και ιδιαίτερες τεχνικές που χρησιμοποιούν, τους κύριους τομείς που αντιπροσωπεύουν οι εφαρμογές τους, τη χρονολογία τελευταίας ενημέρωσης με βάση τυχόν repositories και changelogs των ιστοσελίδων τους και την κατηγορία στην οποία τα εντάσσουμε.

Ορισμένα projects είναι ενεργά, με την έννοια ότι ενημερώνονται ανά διαστήματα και σύμφωνα με αλλαγές και νέες εκδόσεις των προδιαγραφών του OpenMP. Άλλα, όπως τα ompsrc, UTS και Minebench αφορούν παλιές υλοποιήσεις και αναφέρονται σε παλαιότερες εκδόσεις του OpenMP. Ωστόσο μπορεί να φανούν χρήσιμα για μελέτες και για έλεγχο ορισμένων λειτουργιών των υλοποιήσεων των μεταφραστών.

Πίνακας 4.1: Κατηγοριοποίηση των μετροπρογραμμάτων

Όνομα	Γλώσσα	Πλατφόρμα	Λειτουργίες OpenMP / ιδιαίτερες τεχνικές	Κύριο Είδος Εφαρμογών	Τελευτάια Ενημέρωση	Κατηγορία
OpenMP Validation Suite V 3.0	C, Fortran	CPU	OpenMP 3.0	Ελέγχου Ορθότητας	2015	Έλεγχος Ορθότητας
OpenMP Validation and Verification (έκδοση 4.5)	C, C++, Fortran	Επιταχυντές	Devices	Ελέγχου Ορθότητας	2019	Έλεγχος Ορθότητας
EPCC OpenMP microbenchmark suite	C, Fortran	CPU	OpenMP 3.0 (γλώσσα C) OpenMP 2.5 (γλώσσα Fortran)	Microbenchmarks	2015	Microbenchmarks
NAS Parallel Benchmarks	Fortran	CPU	OpenMP 2.5, nested παραλληλισμός, pipelining	Επιστημονικές εφαρμογές, πυρήνες εφαρμογών	2019	Benchmarks γενικού σκοπού
Parsec	C++	CPU	OpenMP 3.0, TBB, Pthreads, nested παραλληλισμός, pipelining	Επιστημονικές και "state of the art" εφαρμογές	unknown	Benchmarks γενικού σκοπού
SPEC OMP2012	C, C++, Fortran	CPU	OpenMP 3.0, nested παραλληλισμός, pipelining	Επιστημονικές εφαρμογές, πυρήνες εφαρμογών	2019	Benchmarks γενικού σκοπού
Ompscr	C, C++	CPU	OpenMP 3.0, on purpose bad solutions	Πυρήνες εφαρμογών	2013	Benchmarks γενικού σκοπού
Bots	C	CPU	OpenMP 3.0, tasks	Πυρήνες εφαρμογών	2019	Benchmarks ειδικού σκοπού
UTS	C	CPU	OpenMP 3.0, tasks, work stealing	Εξερεύνηση σε γραφήματα	2012	Benchmarks ειδικού σκοπού
Rodinia	C, C++	CPU, GPU	OpenMP 4.5, offloading, CUDA, OpenCL	Επιστημονικές εφαρμογές, εξόρυξη δεδομένων	2017	Benchmarks ειδικού σκοπού
MineBench	C, C++	CPU	OpenMP 3.0	Εξόρυξη δεδομένων	2015	Benchmarks ειδικού σκοπού

ΚΕΦΑΛΑΙΟ 5
ΈΛΕΓΧΟΣ ΟΡΘΟΤΗΤΑΣ

Τα Validation and Verification tests έχουν δημιουργηθεί με σκοπό τον έλεγχο της ορθότητας όλων των constructs του OpenMP. Εκτός από τα επίσημα tests που περιγράφονται στις επόμενες ενότητες, ορισμένοι μεταφραστές περιλαμβάνουν δικά τους tests ελέγχου ορθότητας, όπως για παράδειγμα ο GCC με τα tests της βιβλιοθήκης libgomp. Μπορούν να χρησιμοποιηθούν για τον έλεγχο ορθότητας μίας υλοποίησης του OpenMP, και για το σκοπό αυτό δοκιμάζονται από διάφορους μεταφραστές. Πρόκειται για δοκιμαστικά προγράμματα πολύ σημαντικά και χρήσιμα για όσους αναπτύσσουν συστήματα OpenMP, καθώς μπορούν να χρησιμοποιηθούν για να ελέγχει κάποιος κατά πόσο η υλοποίηση συμφωνεί με τη συμπεριφορά που προδιαγράφει το πρότυπο.

Οι επίσημες σουίτες περιλαμβάνουν tests γραμμένα σε γλώσσα C για τον έλεγχο των διάφορων οδηγιών του OpenMP της έκδοσης 3.0 καθώς και της έκδοσης 4.5 που περιλαμβάνει τα devices. Κατά την εκπόνηση της προκείμενης διπλωματικής, χρησιμοποιήθηκαν και οι δύο αυτές σουίτες για τον έλεγχο του OMPi, κατά την ανάπτυξη νέων εκδόσεων και αποτελούν το βασικό εργαλείο για την εύρεση σφαλμάτων.

5.1 OpenMP Validation Suite V 3.1

Το Validation Suite για την έκδοση 3.1 του OpenMP [2] περιλαμβάνει μικρά tests γραμμένα σε γλώσσα C και είναι ελεύθερα διαθέσιμα. Για την ανάπτυξη των tests, καθώς και το σχολιασμό της αποδοτικότητας τους, από τους συγγραφείς χρησιμοποιήθηκαν διάφοροι ελεύθερα διαθέσιμοι μεταφραστές, όπως της OpenUH, της Intel, της Sun/Oracle και της GNU.

Το OpenMP είναι ένα πολύ διαδεδομένο μοντέλο προγραμματισμού, κάτι που δε γίνεται αντιληπτό μόνο από τις πολλές εφαρμογές που το χρησιμοποιούν, αλλά και από τη πληθώρα μεταφραστών που είναι διαθέσιμοι για χρήση και το υποστηρίζουν. Μέχρι και την έκδοση 2.0 το OpenMP standard περιλάμβανε 10 διαφορετικά constructs, 2 directives και 11 clauses ενώ η κύρια διαφορά με τις μετέπειτα εκδόσεις, έως και την 3.0, ήταν η προσθήκη των tasks. Αν και δεν μπορούν όλοι οι συνδυασμοί clauses με directives να χρησιμοποιηθούν, ένας μεγάλος αριθμός αυτών

μπορούν, κάτι που κάνει την ανάπτυξη αυτών των tests μία πολύπλοκη εργασία.

Η ιδέα της υλοποίησης των tests είναι να υπάρχει μία υπορουτίνα για κάθε construct η οποία επιστρέφει true αν δουλεύει κατά το προβλεπόμενο και false σε αντίθετη περίπτωση. Κάθε υπορουτίνα εκτελεί έναν υπολογισμό, το αποτέλεσμα του οποίου θα πρέπει να εξαρτάται από τη σωστή λειτουργία του αντίστοιχου construct. Για παράδειγμα, αν κατά τον έλεγχο της οδηγίας "parallel" και τα clauses που σχετίζονται με αυτήν όπως το "shared" το συγκεκριμένο χαρακτηριστικό δεν έχει υλοποιηθεί σωστά, το αντίστοιχο test θα αποτύχει. Αυτά τα tests αναφέρονται ως "normal tests". Επίσης, αξιολογείται το αποτέλεσμα του υπολογισμού όταν το αντίστοιχο construct λείπει. Τα tests αυτά αναφέρονται ως "crosstests". Επιπλέον, πρέπει να ελεγχθεί αν η συγκεκριμένη οδηγία εξυπηρετεί το σκοπό της. Για παράδειγμα, έστω ότι έχουμε μία μεταβλητή δηλωμένη ως "shared". Αν αντικαταστήσουμε τη μεταβλητή αυτή προσθέτοντας το clause "private", ή όποιο άλλο clause δεν προσφέρει τις ιδιότητες της οδηγίας που ελέγχουμε (shared), το αποτέλεσμα του "crosstest" θα πρέπει να είναι "false" καθώς πλέον η μεταβλητή μας δεν είναι "shared". Επιπλέον για να ελεγχθεί η ορθότητα της οδηγίας όταν αυτή είναι "orphaned from main" αναπτύχθηκαν τα "orphaned tests", τα οποία εκτελούν την οδηγία σε μία συνάρτηση που καλείται από την main.

Για να μεγαλώσει η πιθανότητα να βρεθεί ένα race condition, κάτι που θέλουμε ώστε να πούμε με βεβαιότητα ότι ένα test επιστρέφει true καθώς όφειλε και όχι επειδή έτυχε με τη σειρά που τα νήματα μπήκαν σε μια κοινόχρηστη περιοχή, τα tests επαναλαμβάνονται N φορές. Φυσικά αν ένα test αποτύχει μία φορά το συγκεκριμένο construct δε δουλεύει, το αντίστροφο όμως δεν ισχύει. Για την εκτίμηση της πιθανότητας ένα test να πέρασε τον έλεγχο "κατά λάθος" ακολουθείται η ακόλουθη προσέγγιση. Αν nf είναι ο αριθμός των αποτυχημένων cross tests και M ο συνολικός αριθμός των επαναλήψεων, εκτιμάται η πιθανότητα ένα test να απέτυχε με $p = \frac{nf}{M}$. Η πιθανότητα μία λανθασμένη υλοποίηση να πέρασε το test είναι $pa = (1-p)^M$ και η βεβαιότητα του test $pc = 1 - pa$, δηλαδή η πιθανότητα να επικυρωθεί μια οδηγία.

Η ιδέα της υλοποίησης των tests είναι ότι η οδηγία που ελέγχεται περνάει από το "normal test" και εφόσον είναι επιτυχημένο παράγεται αυτόματα ο κώδικας για τους άλλους δύο τύπους test, "cross test" και "orphaned" και εκτελείται ο κώδικας. Σε αντίθετη περίπτωση το test μαρκάρεται ως failed και δεν παράγεται ο κώδικας για τα άλλα δύο.

Όσον αφορά την έκδοση 3.0, αυτή περιλαμβάνει τα tasks, τα οποία μπορούν να εκτελεστούν κατευθείαν ή μετά από μία καθυστέρηση από τα διάφορα νήματα. Η βασική ιδέα είναι τα tasks να δημιουργηθούν από ένα νήμα και μετά να εκτελεστούν σε μια παράλληλη περιοχή από πολλαπλά νήματα. Κατά το "cross test" η οδηγία task αφαιρείται. Ακολουθεί ο κώδικας του OpenMP Task construct :

```
int test_omp_task(){
      int i, result=0;
      int tids[NUM_TASKS];

      for (i=0; i<NUM_TASKS; i++)
      {
            tids[i]=0;
      }
      #pragma omp parallel
      {
            #pragma omp single
            {
                  for (i=0; i<NUM_TASKS; i++){
                        int myi=i;
                        #pragma omp task
                        {
                              sleep(SLEEPTIME);
                              tids[myi]=omp_get_thread_num();
                        }
                  }
            }
      }
      /*now check for results*/
      for (i=1; i<NUM_TASKS; i++){
            if (tids[0] != tids[i])
                  return (result=1);
      }
      return result;
}
```

Χρησιμοποιώντας στατιστικής ανάλυσης προσέγγιση, τα tests επαναλαμβάνονται πολλές φορές και ο συνολικός αριθμός των fail/pass tests υπολογίζεται, αφήνοντας ανοιχτό το ενδεχόμενο κάποια tests να περνάνε τον έλεγχο αλλά η υλοποίηση των αντίστοιχων μεταφραστών να είναι λανθασμένη. Ας αναφερθεί εδώ, ότι είναι αρκετά δύσκολο να ερμηνεύσει κανείς τον λόγο για τον οποίο ένα test αποτυγχάνει. Παρόλα αυτά, οι συγγραφείς έχουν προσπαθήσει να δώσουν όσο πιο πολλές πληροφορίες μπορούν, όπως επίσης να επισημάνουν και να ξεκαθαρίσουν διάφορες αμφιβολίες που προκύπτουν για κάποιες από τις προδιαγραφές του OpenMP. Στο σημείο αυτό, ας ειπωθεί επίσης, πως μπορεί να ανοίξει

ολόκληρη συζήτηση για το κατά πόσο τα OpenMP standards είναι αρκετά ακριβή ώστε να μπορεί να γίνει επαλήθευση (verification).

5.2 OpenMP Validation and Verification (εκδοση 4.5)

Το συγκεκριμένο project [3] αναφέρεται στην έκδοση 4.5 του OpenMP που περιλαμβάνει οδηγίες για ετερογενείς υπολογισμούς. Περιέχει tests που σκοπό έχουν να ελέγξουν τις συγκεκριμένες οδηγίες και κατά πόσο είναι σωστά υλοποιημένες από μεταφραστές που έχουν τη δυνατότητα να στέλνουν εκτελέσιμο κώδικα σε συσκευές, όπως για παράδειγμα επιταχυντές γραφικών.

Οι συγγραφείς επικεντρώνονται τόσο στην αξιολόγηση της υποστήριξης των οδηγιών target offload από διάφορους μεταφραστές, όσο και στην έκθεση κάποιον αμφιβολιών σχετικά με τις προδιαγραφές της έκδοσης 4.5 του OpenMP. Σκοπεύουν επίσης να δωρίσουν τα tests, όπως επίσης και τα mini apps/kernels που έχουν αναπτύξει, ώστε να ενσωματωθούν σε μεγαλύτερες σουίτες εφαρμογών benchmark, συγκεκριμένα των SPEC –τα οποία θα δούμε σε επόμενο κεφάλαιο-, δηλαδή στην επόμενη έκδοση των SPEC OMP και SPEC ACCEL.

Αυτή τη στιγμή, δύο είναι τα πιο διαδεδομένα directive-based μοντέλα προγραμματισμού, το OpenACC και το OpenMP. Εκτός από αυτές τις προσεγγίσεις, έχουν αναπτυχθεί και άλλες τεχνικές για τον προγραμματισμό επιταχυντών που περιλαμβάνουν τα CUDA, OpenCL, NVIDIA Thrust, kokkos κ.λ.π. Καθώς τα offloading χαρακτηριστικά συνεχίζουν να εξελίσσονται είναι σημαντικό να δούμε κατά πόσο οι υλοποιήσεις ανταποκρίνονται στις προδιαγραφές. Οι προδιαγραφές αυτές, εξαιτίας της αμφιβολίας που τις διέπουν ορισμένες φορές, είναι δύσκολο να ερμηνευτούν και να διατηρηθεί η συνέπεια μεταξύ αυτών και μιας υλοποίησης. Έτσι, οι ερευνητές τείνουν να ερμηνεύσουν τέτοιες προδιαγραφές διαφορετικά και να υπάρχουν διαφορές ανάμεσα στους μεταφραστές που τις υποστηρίζουν, κάποιες από τις οποίες είναι αρκετά λεπτές, αλλά ικανές να προκαλέσουν μία συζήτηση γύρω από την αλλαγή στην περιγραφή των προδιαγραφών.

Το κύριο χαρακτηριστικό της έκδοσης 4.0 είναι να μπορεί να φορτωθεί κώδικας για εκτέλεση σε έναν ή πολλαπλούς συν-επεξεργαστές ενώ ταυτόχρονα εκτελείται ο προ 4.0 έκδοσης παράλληλος κώδικας στον πολυπύρηνο επεξεργαστή. Όλη αυτή η διαδικασία φέρνει αλλαγές στην

εκτέλεση του OpenMP και στο μοντέλο μνήμης. Πιο συγκεκριμένα, η προσθήκη ενός ανεξάρτητου περιβάλλοντος εκτέλεσης για κάθε μια από τις συσκευές, διαφορετικός χώρος διευθύνσεων μνήμης, ιεραρχία μνήμης και η μικροαρχιτεκτονική των πυρήνων. Διάφοροι επιταχυντές, όπως οι GPGPUs και οι Xeon Phi co-processors έχουν επηρεάσει τους ορισμούς του OpenMP σχετικά με τα μοντέλα εκτέλεσης και τα μοντέλα μνήμης.

Όσο προχωράμε προς το μοντέλο CPU+device, γίνονται περισσότερο αναγκαίες οι δυνατότητες εκφόρτωσης σε συσκευές που προσφέρει το OpenMP 4.5. Οι συγγραφείς στο [3] αναλύουν διαφορετικές υλοποιήσεις μεταφραστών OpenMP διαθέσιμων στις εφαρμογές και τεστάρουν την ωριμότητα αυτών των υλοποιήσεων, ώστε να μπορούν να χρησιμοποιηθούν από τις εφαρμογές. Προσπαθούν ακόμα, να ξεκαθαρίσουν αμφιβολίες σχετικά με τις προδιαγραφές του OpenMP, ώστε να γίνουν κατανοητές από όλους. Σκοπός τους επομένως είναι η συλλογή των απαραίτητων tests τόσο ως προς τα χαρακτηριστικά όσο και ως προς την απόδοση των device constructs καθώς και να αναπτυχθούν αυτά τα tests και σε γλώσσα Fortran.

5.3 Αποτελεσματα

Όπως προαναφέρθηκε, τα προγράμματα αυτά χρησιμοποιήθηκαν κατά τον έλεγχο της υλοποίησης κατά την ανάπτυξη του OMPi. Στην έκδοση 3.1 περνάει με επιτυχία ο έλεγχος ορθότητας, ενώ στην έκδοση 4.5 βρέθηκαν σημεία στα οποία η υλοποίηση δεν ανταποκρίνονταν με επιτυχία στα τεστ που αφορούν την εκτέλεση κώδικα σε συσκευές. Φυσικά, σε αυτές τις περιπτώσεις μελετήθηκαν περεταίρω οι προδιαγραφές του μοντέλου και διορθωθήκαν τα σημεία που δε συμφωνούσαν, με σκοπό την επιτυχία σε όλα τα τεστ τα οποία σχετίζονται με τις λειτουργίες που υποστηρίζει ο OMPi. Επίσης, δοκιμάστηκαν και οι μεταφραστές GCC και Intel. Και οι δύο φαίνεται να ανταποκρίνονται με ορθότητα στις προδιαγραφές του OpenMP, περνώντας με επιτυχία όλα τα τεστ των εκδόσεων 3.1 και 4.5.

Τα προγράμματα αυτά είναι πολύ χρήσιμα κατά την ανάπτυξη των υλοποιήσεων των μεταφραστών, καθώς δοκιμάζουν περιπτώσεις δύσκολο να προβλεφτούν ή ακόμη και αμφιβολίες σχετικά με τις προδιαγραφές. Όπως φάνηκε και στη δική μας περίπτωση, μπορούν να επισημάνουν αστοχίες ή ατέλειες στις υλοποιήσεις και είναι το κύριο εργαλείο που έχουν στη διάθεση τους οι ερευνητές για αποσφαλμάτωση.

ΚΕΦΑΛΑΙΟ 6
MICROBENCHMARKS

Το EPCC OpenMP microbenchmark suite [4] περιλαμβάνει μια συλλογή από μετροπρογράμματα, τα οποία υπολογίζουν το overhead διάφορων construct του OpenMP και σχετίζονται με συγχρονισμό, προγραμματισμό των βρόγχων (loop scheduling) και χειρισμό των thread-private data, δηλαδή των δεδομένων που τα νήματα δεν μπορούν να μοιραστούν μεταξύ τους. Στην πιο πρόσφατη έκδοσή τους, η οποία είναι η 3.1, επεκτείνονται ώστε να περιλαμβάνουν και τις οδηγίες της έκδοσης 3.0 του OpenMP που σχετίζονται με tasks. Τα προγράμματα αυτά δε μετράν συνολικά το χρόνο εκτέλεσης των μικρο-εφαρμογών, αλλά προσπαθούν να βγάλουν στην επιφάνεια καθυστερήσεις που υπάρχουν σε κάθε λειτουργία δοκιμάζοντας ακραίες καταστάσεις. Στοχεύουν στο να μετρήσουν τις καθυστερήσεις (overheads) που υπάρχουν σε κάθε δομή/οδηγία του OpenMP, οι οποίες προκύπτουν από τον τρόπο που έχουν υλοποιηθεί αυτές από τον εκάστοτε μεταφραστή.

Η μεθοδολογία που ακολουθείται για τον υπολογισμό του overhead είναι η ίδια για όλα τα προγράμματα και είναι αρκετά απλή. Δεδομένου ενός παράλληλου προγράμματος, έστω ότι Ts είναι ο χρόνος που απαιτείται για την εκτέλεση του αντίστοιχου σειριακού προγράμματος και Tp ο χρόνος για την εκτέλεση σε p επεξεργαστές, όπου κάθε ένας έχει την ίδια ποσότητα έργου να εκτελέσει με το σειριακό πρόγραμμα. Αν δεν υπήρχαν τα overheads, το παράλληλο πρόγραμμα θα εκτελούταν στον ίδιο ακριβώς χρόνο με το σειριακό. Έτσι, το overhead ορίζεται απλά ως $Tp - Ts$, η διαφορά δηλαδή ανάμεσα στο παράλληλο και στο ακολουθιακό πρόγραμμα. Για παράδειγμα, ας αναφέρουμε το "ParallelTasks", στο οποίο κάθε νήμα δημιουργεί και ενδεχομένως εκτελεί ένα task. Υπολογίζεται επομένως ο χρόνος που απαιτείται για την εκτέλεση του

```
#pragma omp parallel{
      for(int j-0; j<innerreps; j++){
            #pragma omp task
            delay(delaylength);
      }
}
```

συγκρινόμενου με το χρόνο εκτέλεσης του

```
for (int j=0; j<innerreps; j++){
      delay(delaylength);
}
```

σε ένα νήμα. Η συνάρτηση **delay** εκτελεί ένα βρόγχο για ένα προκαθορισμένο χρόνο, ο οποίος μπορεί να οριστεί από το χρήστη και δεν καταναλώνει cache ή εύρος ζώνης της μνήμης. Ο αριθμός των επαναλήψεων (**innerepts**) καθορίζεται αυτόματα, ώστε να ικανοποιεί όλα τα tests.

Για να έχουν τα αποτελέσματα των μετρήσεων στατιστική σημασία, ώστε να βγάλουμε ασφαλή συμπεράσματα από αυτά, επαναλαμβάνεται η μέτρηση του overhead 50 φορές για μία εκτέλεση και 20 διαφορετικές εκτελέσεις. Ο λόγος είναι ότι παρατηρείται αρκετά μεγαλύτερη μεταβλητότητα ανάμεσα σε διαφορετικές εκτελέσεις, παρά σε διαφορετικές μετρήσεις στην ίδια εκτέλεση. Σε κάθε run υπολογίζεται ο μέσος όρος και η τυπική απόκλιση σ. Έτσι μετρήσεις οι οποίες δεν είναι "καθαρές" αποκλείονται ελέγχοντας για μεγάλες τιμές του σ και για περιπτώσεις που υπάρχουν υπερβολικά υψηλές τιμές (3σ πάνω από το mean).

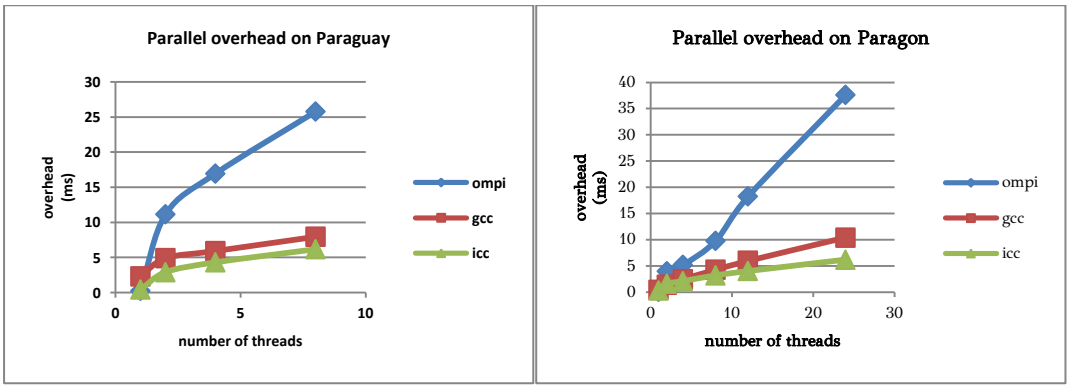
Ο συγχρονισμός των νημάτων και το loop scheduling είναι οι κύριες πηγές overhead στα παράλληλα προγράμματα κοινόχρηστης μνήμης και το βάρος πέφτει στις αντίστοιχες υλοποιήσεις. Συνεπώς, τα tests αυτά, μετρώντας το overhead μπορούν να συγκρίνουν την αποδοτικότητα διαφορετικών υλοποιήσεων του OpenMP και να επισημάνουν αδυναμίες, όπως επίσης και να κατευθύνουν τον προγραμματιστή να διαλέξει ανάμεσα από πλήθος επιλογών την καλύτερη οδηγία για την εφαρμογή του, όπως για παράδειγμα η περίπτωση του **`CRITICAL`** vs **`ATOMIC`** vs lock routines.

Από τη συμπεριφορά των μετρήσεων διαφορετικών constructs, μπορούμε να βγάλουμε επιπλέον συμπεράσματα για την υλοποίηση αυτών. Ένα παράδειγμα είναι το for, στο οποίο υπονοείται ένας barrier στο τέλος. Αν η συμπεριφορά των μετρήσεων στα δύο αυτά προγράμματα είναι παρόμοια, το συμπέρασμα είναι ότι το κύριο μέρος του overhead στο for είναι εξαιτίας του barrier. Αν προστεθεί επιπλέον και reduction λογικό είναι να αυξηθεί το overhead. Ερωτήματα που προκύπτουν είναι κατά πόσο είναι ομαλή η αύξηση αυτή, σε σχέση με το πλήθος των επεξεργαστών, και κατά πόσο είναι αποτελεσματική η συγκεκριμένη υλοποίηση, σε σχέση με μια άλλη. Αυτά τα ερωτήματα δε θα πρέπει να απασχολούν τον προγραμματιστή, παρά μόνο στο βαθμό της επιλογής του μοντέλου παραλληλοποίησης που θα επιλέξει για την εφαρμογή του, ώστε να αποφύγει ορισμένα overheads, κάτι που συζητείται στην περιγραφή κάποιων εφαρμογών στα επόμενα κεφάλαια. Στο [4] συζητούνται ορισμένα πειράματα που έχουν κάνει οι συγγραφείς σε διάφορους

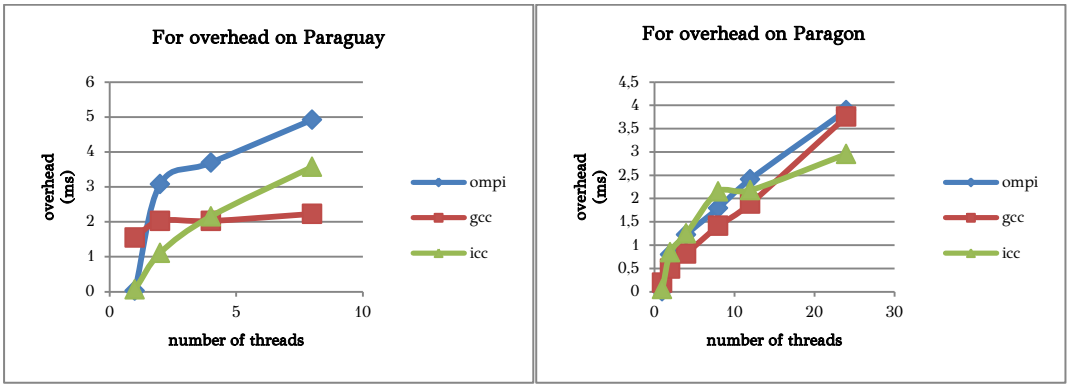
μεταφραστές και συστήματα. Προσπαθούν να ερμηνεύσουν τα αποτελέσματα, καθώς και να προτείνουν εναλλακτικές στην υλοποίηση ορισμένων constructs.

Ακολουθούν μερικά αντιπροσωπευτικά αποτελέσματα των πειραμάτων για το EPCC OpenMP microbenchmark suite. Στα γραφήματα φαίνεται η μεταβολή του overhead σε microseconds (άξονας y) σε σχέση με το πλήθος νημάτων (άξονας χ) με τις κουκίδες να αναπαριστούν το αντίστοιχο πείραμα. Άξια σχολιασμού είναι η συμπεριφορά του OMPi στην οδηγία Parallel (Σχήμα 6.1), οι μέτριες επιδόσεις του οποίου πιστεύουμε ότι εν μέρει μπορεί να αποδίδονται στο γεγονός ότι υποστηρίζει πολλαπλές βιβλιοθήκες νημάτων (EELIBs), αντί να είναι βελτιστοποιημένος για μόνο μία από αυτές. Ο OMPi, γενικά και συγκριτικά με τους άλλους μεταφραστές, φαίνεται να πηγαίνει καλύτερα στον Paragon (For, Barrier, Critical), όπως φαίνεται στα Σχήματα 6.2, 6.3 και 6.5. Επίσης, όπως βλέπουμε στα Σχήματα 6.2 και 6.3, η συμπεριφορά των For και Barrier είναι παρόμοια και στις δύο αρχιτεκτονικές ανεξάρτητα τον μεταφραστή, κάτι που ενισχύει την προηγούμενη τοποθέτηση ότι το κύριο μέρος των καθυστερήσεων του for βρίσκεται στον barrier που υπονοείται στο τέλος. Το Σχήμα 6.4 δείχνει τη μεταβολή του overhead (άξονας y) σε σχέση με το chunk size (άξονας χ) για 8 νήματα στην Paraguay και 24 στον Paragon. Παρατηρούμε ότι η παράμετρος chunk size έχει επίδραση στο overhead στη φράση schedule, τόσο για το Guided όσο και για το Dynamic προσθέτοντας μία ακόμα παράμετρο στην επιλογή της κατάλληλης πολιτικής διαμοιρασμού των επαναλήψεων για τον προγραμματιστή. Όσον αφορά τα tasks (Σχήμα 6.6), παρατηρούνται πολύ καλές επιδόσεις για τον OMPi, με τον GCC να μην έχει την επιθυμητή συμπεριφορά όσο αυξάνει το πλήθος των νημάτων και τον Intel να έχει φτωχές επιδόσεις στο Master Task για μεγάλο πλήθος νημάτων στην Paraguay.

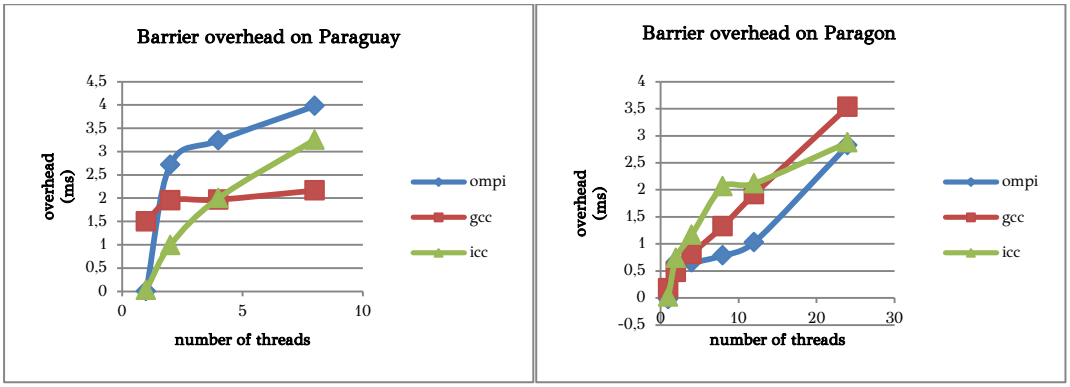
Σχήμα 6.1: Αποτελέσματα πειραμάτων - Parallel



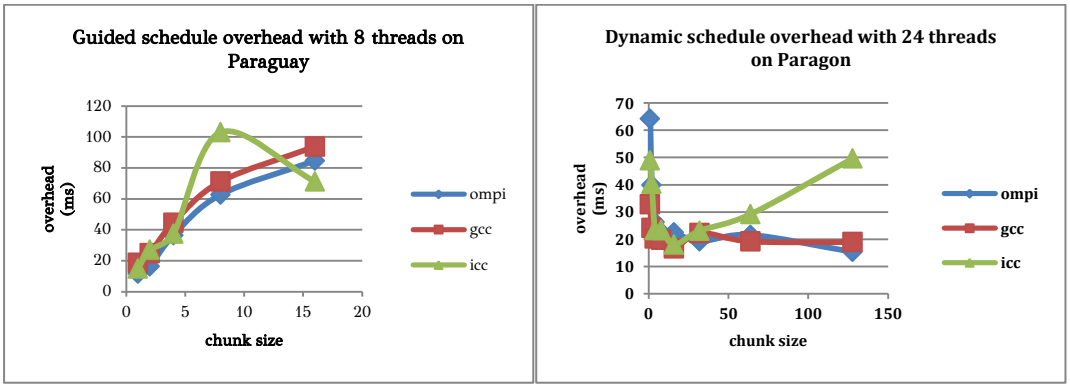
Σχήμα 6.2: Αποτελέσματα πειραμάτων - For



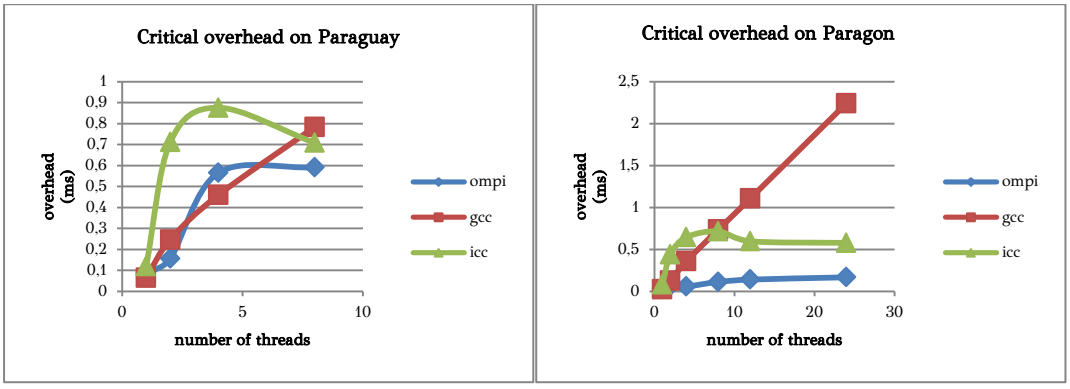
Σχήμα 6.3: Αποτελέσματα πειραμάτων - Barrier



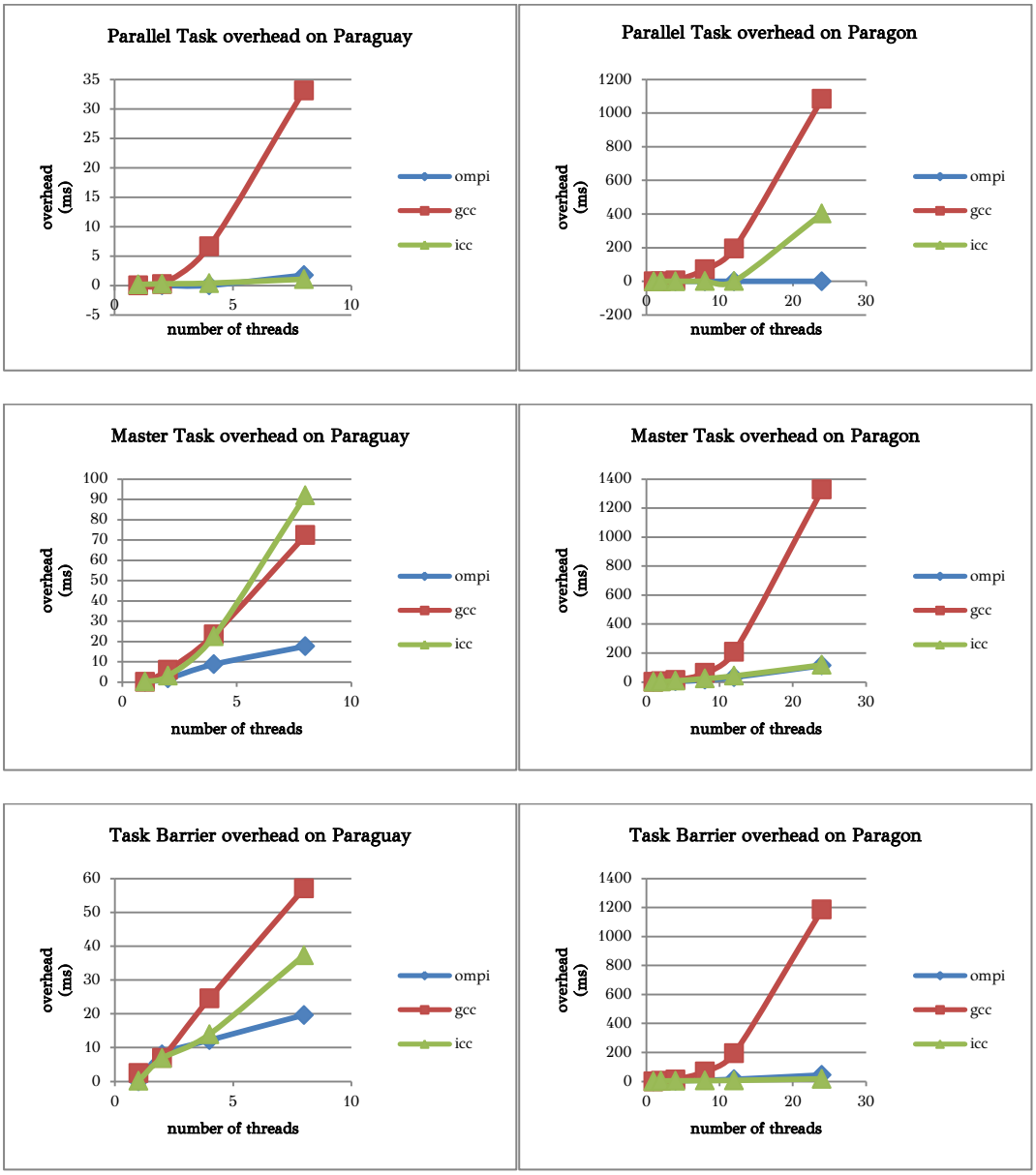
Σχήμα 6.4: Αποτελέσματα πειραμάτων - Schedule



Σχήμα 6.5: Αποτελέσματα πειραμάτων - Critical



Σχήμα 6.6: Αποτελέσματα πειραμάτων -Tasks



ΚΕΦΑΛΑΙΟ 7
ΣΟΥΙΤΕΣ BENCHMARK ΓΕΝΙΚΟΥ ΣΚΟΠΟΥ

7.1 NAS PARALLEL BENCHMARKS

Τα NAS Parallel Benchmarks προέρχονται από κώδικες CFD (computanional fluid dynamics). Έχουν σχεδιαστεί για να συγκρίνουν την απόδοση παράλληλων υπολογιστών και αναγνωρίζονται ευρέως ως ένας δείκτης της απόδοσης ενός παράλληλου συστήματος. Αποτελούνται από 5 εφαρμογές πυρήνα και 3 προσομοιωμένες εφαρμογές CFD που προέρχονται από σημαντικές κλάσεις της αεροφυσικής, της φυσικής δηλαδή που απαιτείται για το σχεδιασμό και την κατασκευή μηχανημάτων αεροδυναμικής. Αυτοί οι 5 πυρήνες μιμούνται τον υπολογιστικό πυρήνα αριθμητικών μεθόδων που χρησιμοποιούνται από τις εφαρμογές CFD. Οι προσομοιωμένες εφαρμογές CFD αναπαράγουν μεγάλο μέρος της κίνησης των δεδομένων και των υπολογισμών που συναντάμε στους κώδικες CFD. Αυτά τα προγράμματα έχουν σχεδιαστεί μόνο αλγοριθμικά στο χαρτί ("pencil and paper" specification) και αναφέρονται ως NPB 1 [5]. Τα benchmarks, τα οποία έχουν τροποποιηθεί και έχουν υλοποιηθεί με OpenMP σε γλώσσα Fortran στην έκδοση 3.0 [6], είναι τα 7 που αναγράφονται παρακάτω, ενώ λεπτομερή περιγραφή γίνεται στο παράρτημα Α. Αυτά είναι όσα αναγράφονται στο NPB 1 και είναι όλα, εκτός του πυρήνα IS (integer sort).

BT: Το BT είναι μια προσομοιωμένη εφαρμογή CFD που χρησιμοποιεί έναν έμμεσο (implicit) αλγόριθμο για την επίλυση 3-διαστάσεων συμπιεσμένων εξισώσεων Navier-Stokes. Η λύση πεπερασμένων διαφορών στο πρόβλημα γίνεται με βάση μια προσέγγιση παραγοντοποίησης εναλλασσόμενης κατεύθυνσης (ADI) που αποσυνδέει τις διαστάσεις x, y, z. Τα συστήματα που προκύπτουν είναι Block-Tridiagonal 5x5 και επιλύονται ακολουθιακά για κάθε μία διάσταση.

SP: Το SP είναι μια προσομοιωμένη εφαρμογή CFD παρόμοια με τη BT. Η λύση πεπερασμένων διαφορών του προβλήματος βασίζεται σε παραγοντοποίηση κατά προσέγγιση με τη Beam-Heating μέθοδο και αποσυνδέει τις διαστάσεις x, y, z. Το προκύπτον σύστημα έχει Scalar-Pentadiagonal ζώνες γραμμικών εξισώσεων, οι οποίες επιλύονται ακολουθιακά για κάθε μια διάσταση.

LU: Το LU είναι μια προσομοιωμένη εφαρμογή CFD που χρησιμοποιεί συμμετρική διαδοχική υπερχαλάρωση (μέθοδος SSOR), για την επίλυση

ενός διαγώνιου συστήματος 7-block που προκύπτει από τη διακριτοποίηση πεπερασμένων διαφορών των εξισώσεων Navier-Stokes σε 3-D, χωρίζοντας το σε block κάτω και άνω τριγωνικού συστήματος.

FT: Το FT περιέχει τον υπολογιστικό πυρήνα μιας φασματικής μεθόδου (spectral method) βασιζόμενη στον 3-D fast Fourier Transform (FFT). Το FT εκτελεί 3 μονοδιάστατους (1-D) FFT, ένα για κάθε διάσταση. Το κύριο χαρακτηριστικό του είναι η ανάγκη για επικοινωνίες από όλους προς όλους.

MG: Το MG χρησιμοποιεί μια V-cycle πολυπλεγματική (MultiGrid) μέθοδο για τον υπολογισμό της λύσης της κλιμακωτής 3-D εξίσωσης Poisson. Ο αλγόριθμος εκτελείται συνεχώς σε ένα σύνολο πλέγματος "that are made between coarse and fine". Η μετακίνηση των δεδομένων τόσο σε μικρές όσο και σε μεγάλες αποστάσεις και οι απαιτήσεις σε μνήμη είναι τα κύρια χαρακτηριστικά του.

CG: Το CG χρησιμοποιεί μια μέθοδο συζυγών κλήσεων (Conjugate Gradient) για να υπολογίσει μια προσέγγιση της μικρότερης ιδιοτιμής ενός μεγάλου, αραιού (sparsed), αδόμητου πίνακα. Ο πυρήνας ελέγχει τους υπολογισμούς και τις επικοινωνίες του αδόμητου πλέγματος, με τη χρήση ενός μητρώου με τυχαία δημιουργούμενες καταχωρήσεις στις θέσεις του. Οι ανομοιόμορφες προσπελάσεις της μνήμης και επικοινωνίες είναι το κύριο χαρακτηριστικό του.

EP: Το EP είναι ένα "Embarrassingly Parallel" benchmark. Δημιουργεί ζεύγη Gaussian τυχαίων αποκλίσεων σύμφωνα με ένα συγκεκριμένο σχήμα. Ο στόχος είναι να προσδιοριστεί το σημείο αναφοράς για το "peak" της απόδοσης μιας συγκεκριμένης πλατφόρμας.

Η υλοποίηση με OpenMP των NAS Parallel Benchmarks ξεκινά με την έκδοση 3.0 και βασίζεται στην βελτιστοποιημένη σειριακή έκδοση NPB2.3-serial. Μαζί με την υλοποίηση HPF σχηματίζουν τη Programming Baseline for NPB (PBN). Στις επόμενες εκδόσεις υλοποιήθηκε η εκδοχή με nested παραλληλισμό με το OpenMP (3.3.1-MZ), καθώς και η υλοποίηση με δυναμική κατανομή μνήμης (NPB 3.4-MZ και NPB 3.4), με την τελευταία να αποτελεί και την τρέχουσα έκδοση των NAS Parallel Benchmarks. Οι εκδόσεις αυτές είναι διαθέσιμες στην επίσημη ιστοσελίδα της NAS και αφορούν υλοποιήσεις σε Fortran.

Για τις ανάγκες των δικών μας μετρήσεων, καθώς ο μεταφραστής OMPi δεν υποστηρίζει προγράμματα γραμμένα σε Fortran, χρησιμοποιήθηκε το ανεπίσημο σετ σε γλώσσα C, παραλληλοποιημένο με OpenMP από το

πανεπιστήμιο της Tsukuba, το οποίο έχει αναπτύξει τον ερευνητικό compiler για OpenMP, Omni [7]. Το σετ αυτό αποτελεί την παράλληλη εκδοχή των σειριακών εκδόσεων NPB2.3-serial, τα οποία είναι γραμμένα σε Fortran. Ο σκοπός της ανάπτυξης της έκδοσης OpenMP σε γλώσσα C ήταν η αξιολόγηση της απόδοσης των αριθμητικών κωδίκων OpenMP σε C και η σύγκριση της απόδοσης με αυτών σε Fortran. Η σουίτα αυτή αναπτύχθηκε από το Real World Partnership ως μέρος του Omni OpenMP compiler project και είναι βασισμένο στο έργο της NAS. Κατά την ανάπτυξη της σουίτας OpenMP σε C από τους ερευνητές του RWP πρώτα μεταφράστηκαν οι κώδικες του NPB2.3-serial σε γλώσσα C και έπειτα παραλληλοποιήθηκαν με το μοντέλο OpenMP. Επιπλέον, το ανεπίσημο αυτό σετ έχει τροποποιηθεί εκ νέου ώστε να ανταποκρίνεται στην έκδοση 3.0 των NPB από το University of Versailles Saint Quentin en Yvlines [8] και αφορά υλοποιήσεις σε C. Το σετ αυτό χρησιμοποιήθηκε επίσης για μέτρηση επιδόσεων και ορισμένα αποτελέσματα φαίνονται παρακάτω.

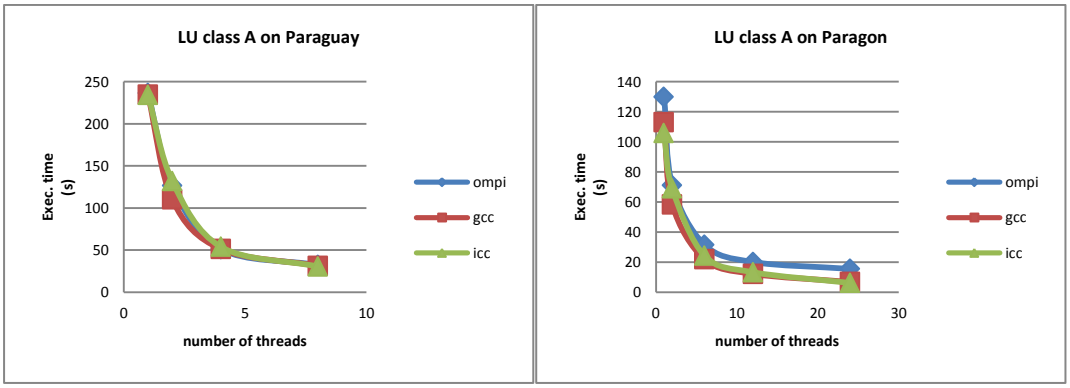
Οι είσοδοι στα benchmarks είναι κλάσεις διαφόρων μεγεθών:

- *Class S:* μικρή είσοδος για γρήγορο έλεγχο.
- *Class W(orkstation):* μικρή είσοδος – αναφέρεται σε 90's workstation.
- *Class A, B, C:* είσοδοι στάνταρντ προβλημάτων – 4x αύξηση μεγέθους από τη μία στην άλλη.
- *Class D, E, F:* είσοδοι μεγάλων προβλημάτων – 16x αύξηση από τις προηγούμενες κλάσεις.

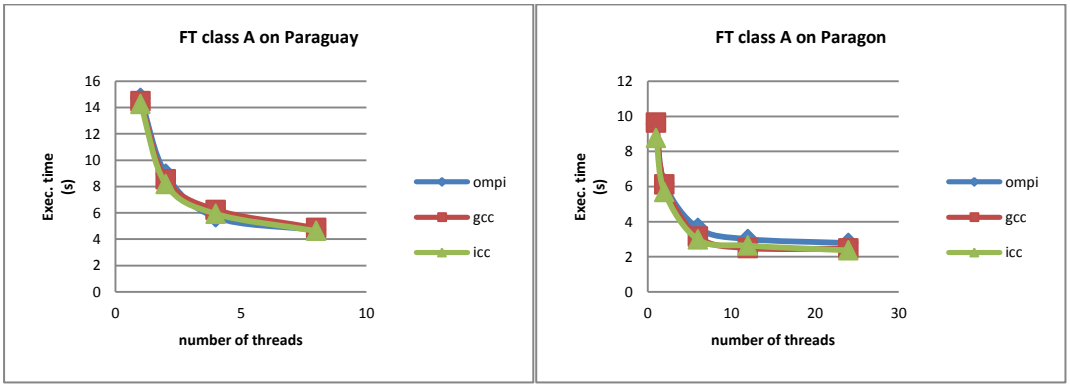
Τα πειράματα έγιναν με είσοδο τα S, W και A, ενώ παρακάτω είναι τα αποτελέσματα τριών benchmark από το σετ του Πανεπιστημίου των Βερσαλλιών, τα οποία είναι αντιπροσωπευτικά της συμπεριφοράς που συναντάμε σε όλες τις εφαρμογές. Τα γραφήματα αφορούν τη μεταβολή του χρόνου εκτέλεσης σε seconds (άξονας y) σε σχέση με το πλήθος νημάτων (άξονας χ), με τις κουκίδες να αφορούν τα αντίστοιχα πειράματα. Γενικά, παρατηρείται και σε ορισμένα από τα δικά μας αποτελέσματα ότι η απόδοση δεν μεταβάλλεται καλά όσο αυξάνει το πλήθος των νημάτων, κάτι που πιθανώς έχει να κάνει με την έλλειψη nested παραλληλισμού και δυναμικής κατανομής μνήμης, δύο στοιχεία που προστέθηκαν σε επόμενες εκδόσεις και παίζουν σημαντικό ρόλο στον περιορισμό των overheads και της επικοινωνίας μεταξύ των νημάτων. Στο Σχήμα 7.1 φαίνεται η συμπεριφορά της εφαρμογής LU να είναι αρκετά καλή, με τους μεταφραστές να έχουν παρόμοιες επιδόσεις και να εκμεταλλεύονται σε μεγάλο βαθμό τον παραλληλισμό που προσφέρουν οι δύο αρχιτεκτονικές.

Για τον πυρήνα FT, στο Σχήμα 7.2, βλέπουμε ότι οι μεταφραστές έχουν παρόμοιες επιδόσεις, η κλιμάκωση όμως του χρόνου εκτέλεσης σε σχέση με το πλήθος των νημάτων δεν είναι η ιδανική, κυρίως στην NUMA αρχιτεκτονική, ενδεχομένως εξαιτίας των επικοινωνιών που είναι απαραίτητες από όλους προς όλους. Τέλος, η εφαρμογή SP (Σχήμα 7.3), δε δείχνει καλή κλιμάκωση του χρόνου εκτέλεσης όσο αυξάνει το πλήθος νημάτων και στις δύο αρχιτεκτονικές, ενώ οι επιδόσεις του OMPi είναι χειρότερες στον Paragon από τα 6 νήματα και έπειτα.

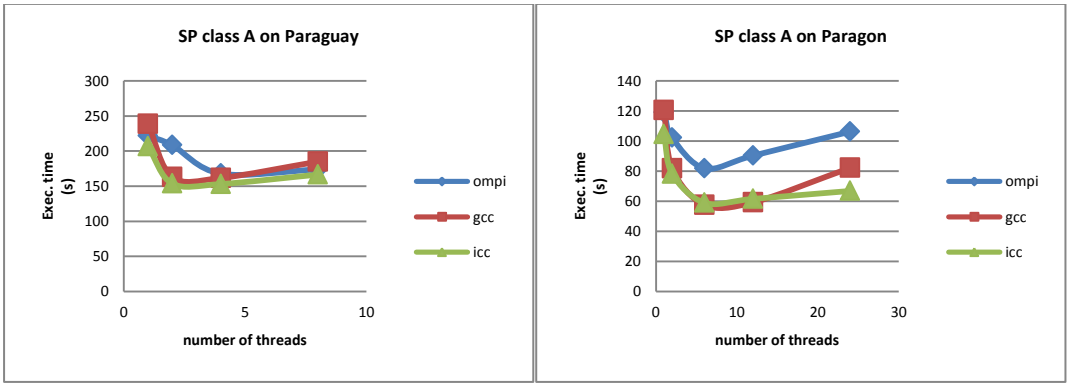
Σχήμα 7.1: Αποτελέσματα πειραμάτων - LU



Σχήμα 7.2: Αποτελέσματα πειραμάτων - FT



Σχήμα 7.3: Αποτελέσματα πειραμάτων - SP



7.2 PARSEC

Το Parsec [9], από τα αρχικά του Princeston Application Repository for Shared-Memory Computers, είναι μια σουίτα εφαρμογών για τη μελέτη των Chip-Multiprocessors (CMPs). Περιλαμβάνει εφαρμογές γύρω από την αναγνώριση, την εξόρυξη και τη σύνθεση (RMS), καθώς και εφαρμογές οι οποίες μιμούνται μεγάλης κλίμακας πολυ-νηματικά εμπορικά προγράμματα και χρησιμοποιούν "state of the art" αλγόριθμους και τεχνικές. Αυτές χαρακτηρίζονται από ποικιλία στη μνήμη που απαιτείται από τους επεξεργαστές, την τοπικότητα, την κοινοχρησία των δεδομένων, το συγχρονισμό και την κίνηση εκτός του chip. Σκοπός του είναι και η υποστήριξη έρευνας, με την έννοια ότι δε στοχεύει σε πραγματικά μηχανήματα μόνο, αλλά παρέχει υποδομές για την ενορχήστρωση, την μεταχείριση και την εκτέλεση λεπτομερών προσομοιώσεων των προγραμμάτων.

Το Parsec αποτελείται από εννιά εφαρμογές και τρεις πυρήνες εφαρμογών σε γλώσσα C++, οι οποίες έχουν επιλεγεί από ένα μεγάλο εύρος τομέων. Όλες οι εφαρμογές έχουν παραλληλοποιηθεί με Pthreads εκτός από την freqmine όπου έχει μόνο υλοποίηση με OpenMP, ενώ για τις Blackscholes και Bodytrack υπάρχει επιπλέον υλοποίηση με OpenMP, και για τις Blackscholes, Bodytrack, Fluidanimate, Streamcluster και Swaptions επιπλέον υλοποίηση με Intel TBB. Η επιλογή έγινε έτσι ώστε να περιλαμβάνονται διαφορετικοί συνδυασμοί μοντέλων παραλληλοποίησης, απαιτήσεων μηχανημάτων και συμπεριφορών χρόνου εκτέλεσης. Ο Πίνακας 7.1 περιγράφει τα χαρακτηριστικά τους.

Οι είσοδοι σε αυτές τις εφαρμογές είναι:

- *test:* πολύ μικρή είσοδος για να ελεγχθεί η βασική λειτουργία του προγράμματος.
- *simdev:* πολύ μικρή είσοδος, η οποία εγγυάται μια βασική συμπεριφορά του προγράμματος πολύ κοντά στην κανονική, με σκοπό την ανάπτυξη test προσομοίωσης.
- *simsmall, simmedium, simlarge:* είσοδοι διαφόρων μεγεθών κατάλληλοι για μελέτη μικροαρχιτεκτονικών με προσομοιωτές.
- *native:* μεγάλη είσοδος για ατόφια (native) εκτέλεση.

Πίνακας 7.1: Σύνοψη χαρακτηριστικών των Parsec εφαρμογών (από το summary paper [10])

Program	Application Domain	Parallelization		Working Set	Data Usage	
		Model	Granularity		Sharing	Exchange
blackscholes	Financial Analysis	data-parallel	coarse	small	low	low
bodytrack	Computer Vision	data-parallel	medium	medium	high	medium
canneal	Engineering	unstructured	fine	unbounded	high	high
dedup	Enterprise Storage	pipeline	medium	unbounded	high	high
facesim	Animation	data-parallel	coarse	large	low	medium
ferret	Similarity Search	pipeline	medium	unbounded	high	high
fluidanimate	Animation	data-parallel	fine	large	low	medium
freqmine	Data Mining	data-parallel	medium	unbounded	high	medium
raytrace	Rendering	data-parallel	medium	unbounded	high	low
streamcluster	Data Mining	data-parallel	medium	medium	low	medium
swaptions	Financial Analysis	data-parallel	coarse	medium	low	low
vips	Media Processing	data-parallel	coarse	medium	low	medium
x264	Media Processing	pipeline	coarse	medium	high	high

Οι είσοδοι *test* και *simdev* έχουν οριστεί με σκοπό τον έλεγχο και την ανάπτυξη και δεν πρέπει να χρησιμοποιούνται για επιστημονικές μελέτες. Οι τρείς είσοδοι προσομοιώσεων για μελέτες ποικίλουν στο μέγεθος, αλλά η γενική εικόνα είναι ότι μεγαλύτερο μέγεθος σημαίνει μεγαλύτερο working set και περισσότερη παραλληλία. Η είσοδος *native* χρησιμοποιείται για μετρήσεις επιδόσεων σε πραγματικά μηχανήματα και, από επιστημονικής σκοπιάς, αποτελεί την πιο ενδιαφέρουσα, καθώς μοιάζει με αληθινά προγράμματα περισσότερο.

Στο [8], για τον χαρακτηρισμό των benchmarks ακολουθείται η τακτική των πολλών, αλλά λιγότερο ακριβών πειραμάτων και μηχανικά μοντέλα που μπορούν να συγκριθούν με αληθινά μηχανήματα, αντί μη ρεαλιστικών μοντέλων τα οποία ενδεχομένως να ταιριάζουν καλύτερα στις ανάγκες των προγραμμάτων. Ορισμένες εφαρμογές, χρησιμοποιούν ιδιαίτερες τεχνικές και λύσεις σε προβλήματα παραλληλοποίησης, που πιθανώς να είναι χρήσιμα στους προγραμματιστές εφαρμογών. Παρακάτω είναι μια περιγραφή των τριών εφαρμογών που έχουν παραλληλοποιηθεί με OpenMP, όπως επίσης και ενδεικτικά γραφήματα της συμπεριφοράς της εφαρμογής με τον GCC.

Blackscholes

Η εφαρμογή blackscholes προέρχεται από το Intel RMS. Υπολογίζει τιμές για το πορτφόλιο των Ευρωπαϊκών οψιόν αναλυτικά μέσω της μερικής διαφορικής εξίσωσης Black-Scholes

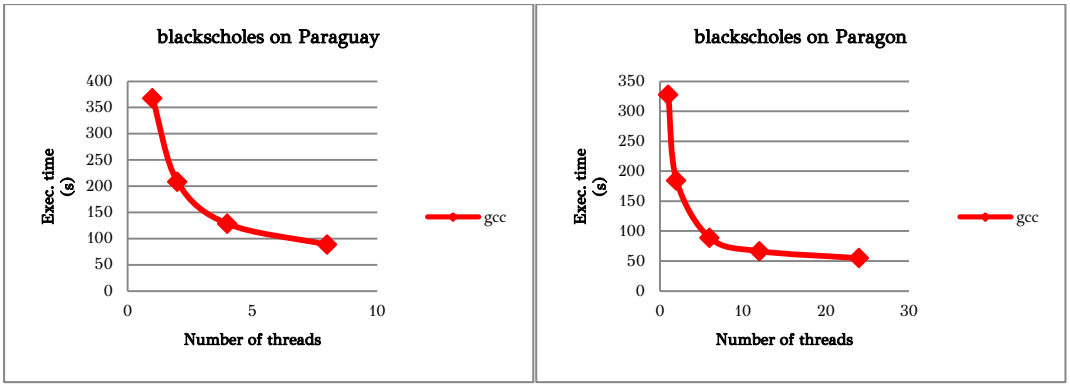
$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

όπου το V είναι μια οψιόν στο υποκείμενο S με μεταβλητότητα σ σε χρόνο t όταν το επιτόκιο είναι η σταθερά r. Δεν υπάρχει κλειστή έκφραση για την εξίσωση Black-Scholes, επομένως η λύση υπολογίζεται αριθμητικά. Το πρόγραμμα περιορίζεται από το πλήθος των υπολογισμών κινητής υποδιαστολής, τους οποίους μπορεί να εκτελέσει ένας επεξεργαστής.

Το πρόγραμμα διαιρεί το πορτφόλιο σε ένα πλήθος εργασιών, ίσο με το πλήθος των νημάτων, και τα επεξεργάζεται παράλληλα. Κάθε νήμα επαναληπτικά καλεί τη συνάρτηση `BlkSchlsEqEuroNoDiv`. Αν ο έλεγχος σφαλμάτων είναι ενεργοποιημένος σε χρόνο μετάφρασης, τότε συγκρίνεται το αποτέλεσμα με μια ενδεικτική τιμή. Η είσοδος για το πρόγραμμα είναι μία οψιόν για την κλάση *test*, 16 για τη *simdev*, 4.096 για τη *simsmall*, 16.384 για τη *simmedium*, 65.536 για τη *simlarge* και 10.000.000 για τη κλάση εισόδου *native*.

Στο Σχήμα 7.4 φαίνεται ενδεικτικά η συμπεριφορά του χρόνου εκτέλεσης σε seconds σε σχέση με το πλήθος νημάτων με το μεταφραστή GCC και την κλάση εισόδου native.

Σχήμα 7.4: Αποτελέσματα πειραμάτων - blackscholes



Bodytrack

Η εφαρμογή υπολογιστικής όρασης bodytrack είναι ένα workload του Intel RMS και καταγράφει μια 3D πόζα ενός ανθρώπινου σώματος με πολλαπλές κάμερες, μέσω μια ακολουθίας εικόνων. Το bodytrack χρησιμοποιεί ένα φίλτρο σωματιδίων (annealed particle filter) για να καταγράψει τη πόζα χρησιμοποιώντας τις ακμές και τη σιλουέτα σε πρώτο πλάνο ως χαρακτηριστικά (featutes) της εικόνας, με βάση ένα χωρισμένο σε 10 τμήματα 3D μοντέλο δέντρου κίνησης του σώματος.

Το πρόγραμμα έχει ένα επίμονο thread pool το οποίο υλοποιείται στην κλάση **`WorkPoolPthread`**. Το κύριο νήμα εκτελεί το πρόγραμμα και στέλνει tasks στο pool μέσω της μεθόδου **`SignalCmd`**, κάθε φορά που συναντά έναν παράλληλο πυρήνα και συνεχίζει την εκτέλεση του προγράμματος μόλις λάβει το αποτέλεσμα από το νήμα εργάτη. Τα πιθανά tasks κωδικοποιούνται με αρίθμηση **`threadCommands`** στην κλάση **`WorkPoolPthread`**. Το πρόγραμμα έχει τους εξής παράλληλους πυρήνες:

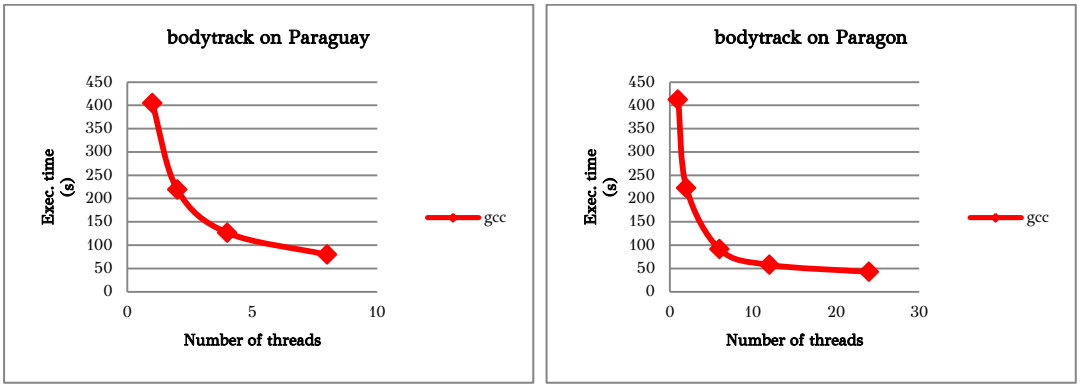
- *Edge detection*, ο οποίος χρησιμοποιεί την βασιζόμενη στην κλίση αναγνώρισης ακμής μάσκα για να βρει τις ακμές.
- *Edge smoothing*, ένα διαχωρίσιμο Gaussian φίλτρο μεγέθους 7x7 για την εξομάλυνση των ακμών στην συνάρτηση **`GaussianBlur`**. Ο πυρήνας αυτός έχει δύο παράλληλες φάσεις, μια για το φιλτράρισμα των γραμμών και μία των στηλών.
- *Calculate particle weighs*. Ο πυρήνας αυτός εκτιμά τη σιλουέτα στο προσκήνιο και τις ακμές στην εικόνα που παρήχθησαν προηγουμένως, για να υπολογίσει τα βάρη των σωματιδίων. Εκτελείται μία φορά για κάθε annealing στρώμα κατά τη διάρκεια κάθε χρονικού βήματος, κάνοντάς τον το πιο απαιτητικό υπολογιστικά σημείο της εφαρμογής.

Οι παράλληλοι πυρήνες χρησιμοποιούν "tickets" για να κατανεμηθεί ο φόρτος σε όλα τα νήματα ομοιόμορφα. Ο μηχανισμός αυτός υλοποιείται στην κλάση **`TicketDispenser`** και συμπεριφέρεται σαν κοινόχρηστος μετρητής.

Η είσοδος για την κλάση εισόδου *test* περιλαμβάνει 4 κάμερες, 1 frame, 5 σωματίδια και 1 στρώμα, για την κλάση *simdev* 4 κάμερες, 1 frame, 100 σωματίδια και 3 στρώματα, για την κλάση *simsmall,* 4 κάμερες, 2 frames, 2.000 σωματίδια και 5 στρώματα, για την *simlarge*, 4 κάμερες, 4 frames, 4.000 σωματίδια και 5 στρώματα, και τέλος για την κλάση *native* 4 κάμερες, 261 frames, 4.000 σωματίδια και 5 annealing layers.

Στο Σχήμα 7.5 φαίνεται ενδεικτικά η συμπεριφορά της εφαρμογής με τον GCC με την κλάση εισόδου native. Το γράφημα δείχνει τη μεταβολή του χρόνου εκτέλεσης σε seconds σε σχέση με το πλήθος νημάτων.

Σχήμα 7.5: Αποτελέσματα πειραμάτων - bodytrack



Freqmine

Το freqmine επιστρατεύει μια array-based εκδοχή της FP-growth μεθόδου για την εξόρυξη συχνών στοιχειοσυνόλων. Συγκαταλέγεται στα benchmarks εξαιτίας της αυξημένης ζήτησης για τεχνικές εξόρυξης δεδομένων, οι οποίες οδηγούνται από την ραγδαία αύξηση του όγκου της πληροφορίας που αποθηκεύεται.

Το πρόγραμμα έχει παραλληλοποιειθεί με OpenMP και χρησιμοποιεί τρεις παράλληλους πυρήνες:

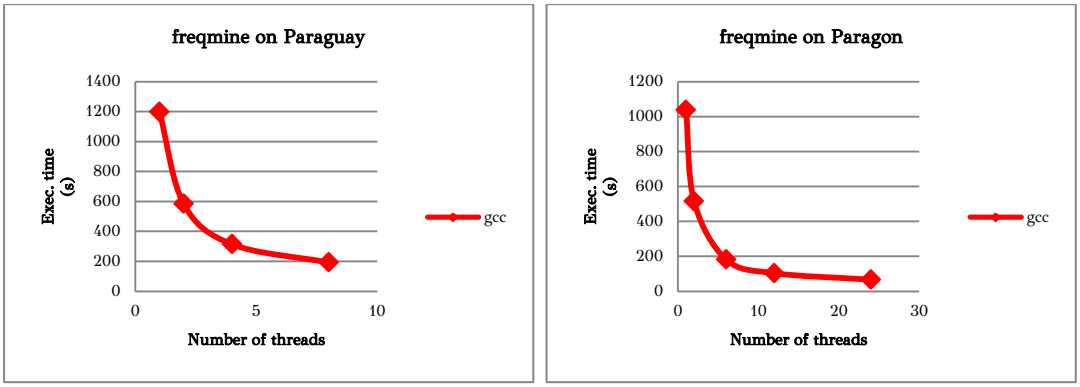
- *Build FP-tree header.* Αυτός ο πυρήνας σαρώνει τη βάση δεδομένων και μετράει το πλήθος των εμφανίσεων για κάθε αντικείμενο. Εκτελεί την πρώτη από τις δυο σαρώσεις και το αποτέλεσμα της είναι η δημιουργία του πίνακα επικεφαλίδων, στον οποίο περιέχεται η πληροφορία για τη συχνότητα των αντικειμένων. Έχει μια παραλληλοποιημένη επανάληψη υλοποιημένη στη συνάρτηση **`scan1_DB`**.
- *Construct prefix tree.* Ο επόμενος πυρήνας κατασκευάζει την αρχική δομή του FP δέντρου. Εκτελεί τη δεύτερη σάρωση της βάσης δεδομένων, απαραίτητη για την κατασκευή των δομών που θα χρησιμοποιηθούν για την εξόρυξη. Ο πυρήνας έχει τέσσερις παράλληλες λούπες, οι οποίες είναι υλοποιημένες στις συναρτήσεις **`scan2_DB`** και **`database_tiling`**.
- *Mine data.* Ο τελευταίος πυρήνας χρησιμοποιεί τις δομές που κατασκευάστηκαν στους προηγούμενους πυρήνες και τις εξορύχει για να λάβει επαναληπτικά την πληροφορία για τα συχνά στοιχειοσύνολα. Είναι υλοποιημένος στη συνάρτηση **`FP_growth_first`**. Πρώτα, εξάγει τον αρχικό πίνακα επικεφαλίδων καλώντας τη συνάρτηση **`first_transform_FPTree_into_FP-Array`**.

Αυτή η συνάρτηση εκτελεί την πρώτη από τις δύο παράλληλες επαναλήψεις, με τη δεύτερη να είναι η **`FP_growth`** ανάλογη της **`FP_growth_first`**.

Τα input sets για το πρόγραμμα freqmine περιέχουν βάση δεδομένων 3 συνθετικών συναλλαγών και ελάχιστο support 1 για την *test*, βάση δεδομένων με 1.000 συνθετικές συναλλαγές και ελάχιστο support 3 για την *simdev*, βάση 250.000 ανώνυμων click streams από ένα ουγγρικό online news portal και ελάχιστο support 220 για την *simsmall*, 500.000 click streams από την ίδια πηγή με 410 ελάχιστο support για την *simmedium*, 990.000 click streams από την ίδια πηγή και 790 ελάχιστο support για την *simlarge* και βάση δεδομένων, συγκροτημένη από συλλογή 250.000 web HTML εγγράφων και ελάχιστο support 11.000 για την *native* είσοδο.

Στο Σχήμα 7.6 φαίνεται ενδεικτικά η συμπεριφορά του χρόνου εκτέλεσης σε seconds σε σχέση με το πλήθος νημάτων με το μεταφραστή GCC και την κλάση εισόδου native.

Σχήμα 7.6: Αποτελέσματα πειραμάτων - freqmine



7.3 SPEC OMP2012

Τα SPEC OMP2012 [11] έχουν αναπτυχθεί από το SPEC High Performance Group και αποτελούνται από 15 παράλληλες εφαρμογές OpenMP από ένα μεγάλο εύρος πεδίων. Οι μετρικές είναι: μία για την απόδοση που βασίζεται στο χρόνο εκτέλεσης της κάθε εφαρμογής και μία επιπλέον για την κατανάλωση ενέργειας.

Με την έκδοση αυτή των SPEC ανανεώνεται η OpenMP σουίτα που είχε εκδοθεί το 2001. Η αρχική σουίτα SPEC OMP2001, συγκροτείται από μια συλλογή εφαρμογών OpenMP και είχε εκδοθεί τον Ιούνιο του 2001. Ακολούθησε μια ενημέρωση τον Ιούνιο του 2002 που περιλάμβανε ένα μεγαλύτερο dataset. Μέχρι το Φεβρουάριο του 2012 περισσότερα από 370 αποτελέσματα δημοσιεύτηκαν συμβάλλοντας στη δημοτικότητα του benchmark. Το SPEC OMP2001 ήταν βασισμένο στις προδιαγραφές της έκδοσης 1.0 του OpenMP που εκδόθηκε το 1998. Οι περισσότερες εφαρμογές ήταν βασισμένες σε κώδικα των SPEC CPU2000 με προσθήκη οδηγιών OpenMP. Στο μεταξύ το OpenMP είχε φτάσει στην έκδοση 3.0, περιλαμβάνοντας νέες οδηγίες και clauses. Η αυξανόμενη χρήση του OpenMP, η εξέλιξη των προδιαγραφών και το γεγονός ότι τυπικές εφαρμογές αλλάζουν με το χρόνο στα πλαίσια αλγορίθμων και προδιαγραφών γλώσσας, έδωσαν το κίνητρο για την ανάπτυξη του SPEC OMP2012.

Η ανάπτυξη της συγκεκριμένης σουίτας περιλαμβάνει την εύρεση υποψήφιων εφαρμογών από διαφορετικά επιστημονικά πεδία, χρησιμοποιώντας ποικιλία οδηγιών OpenMP σε διαφορετικές γλώσσες προγραμματισμού και καταπονώντας διάφορα κομμάτια του υλικού, όπως ο πυρήνας και οι ιεραρχίες μνήμης.

Τα SPEC προσφέρουν μια συλλογή εργαλείων για τον αυτόματο έλεγχο ορθότητας και τον ορισμό των "ασφαλών" βελτιστοποιήσεων που χρησιμοποιούνται από τους προμηθευτές των μεταφραστών, για την δόμηση των αποτελεσμάτων ώστε να είναι δημοσιοποιήσιμα και συγκρίσιμα. Η ολική μετρική της απόδοσης είναι ο γεωμετρικός μέσος όρος των αναλογιών του χρόνου εκτέλεσης του εξεταζόμενου συστήματος προς το χρόνο εκτέλεσης σε ένα σύστημα αναφοράς. Το σύστημα αναφοράς είναι ένα Sun Fire X4140 με δύο AMD Opteron 2384 επεξεργαστές (quad core "Shangai", 2.7 GHz) με 32 GB RAM.

Μια άλλη ενδιαφέρουσα πλευρά είναι η προσθήκη μιας πειραματικής μετρικής της ενέργειας. Η κατανάλωση ενέργειας έχει προστεθεί ως ξεχωριστό και προαιρετικό κομμάτι του benchmark και συγκρίνει την κατανάλωση ενέργειας για κάθε benchmark με την κατανάλωση ενέργειας του μηχανήματος αναφοράς.

Παρακάτω ακολουθεί μια σύντομη περιγραφή των εφαρμογών σύμφωνα με το [11]:

350.md Εκτελεί προσομοιώσεις μοριακών δυνάμεων (molecular dynamics) πυκνής πυρηνικής "ύλης", όπως αυτές συμβαίνουν σε Type II supernovas, στα εξωτερικά στρώματα των αστέρων νετρονίου και σε λευκούς νάνους αστέρες. Ο IU-MD κώδικας προσομοιώνει πλήρως ιονισμένα άτομα μέσω μιας κλασσικής screened Coulomb interaction. Ένας εκθετικός παράγοντας μοντελοποιεί το φαινόμενο screening του αερίου ηλεκτρονίων. Αυτές οι προσομοιώσεις έχουν χρησιμοποιηθεί για να μελετηθούν διάφορες ιδιότητες πυκνής ύλης σε συμπαγή αστρικά αντικείμενα. Το benchmark εκτελεί ένα μικρό run ενός ρεαλιστικού συστήματος 27.648 ιόντων άνθρακα και οξυγόνου.

351.bwaves Προσομοιώνει αριθμητικά, οστικά κύματα σε μια τριών διαστάσεων διηχητική μεταβατική ελασματοειδή ιξώδη ροή (transonic transient laminar viscous flow). Η αρχική ρύθμιση του προβλήματος των οστικών κυμάτων αποτελείται από μια υψηλής πίεσης και πυκνότητας περιοχή στο κέντρο ενός κυβικού κελιού σε ένα περιοδικό πλέγμα, με χαμηλή πυκνότητα και πίεση οπουδήποτε αλλού. Ο αλγόριθμος που υλοποιείται είναι ένας επιλυτής χωρίς παράγοντες για την έμμεση λύση των συμπιεστών Navier-Stokes εξισώσεων χρησιμοποιώντας τον biconjucate gradient stabilized (Bi-CGstab) αλγόριθμο, ο οποίος λύνει συστήματα μη συμμετρικών γραμμικών εξισώσεων επαναληπτικά. Ο κώδικας έχει παραλληλοποιειθεί με 29 οδηγίες parallel do.

352.nab Είναι βασισμένη στο Nucleic Acid Builder (NAB), που είναι μια εφαρμογή molecular dynamics που εκτελεί τους ακριβούς υπολογισμούς κινητής υποδιαστολής που συμβαίνουν συχνά σε επιστημονικούς υπολογισμούς. Οι υπολογισμοί διαφέρουν από σχετικά ανοργάνωτους "molecular dynamics" σε δομημένη γραμμική άλγεβρα.

357.bt331 Πρόκειται για την αντίστοιχη εφαρμογή των NPB 3.3.1 που περιγράφεται στην αντίστοιχη ενότητα.

358.botsalgn πρόκειται για την εφαρμογή που αποτελεί μέρος του Barcelona OpenMP tasks suite που περιγράφεται στην αντίστοιχη ενότητα.

359.botsspar Πρόκειται για την εφαρμογή που εκτελεί παραγοντοποίηση LU και αποτελεί μέρος του Barcelona OpenMP tasks suite που περιγράφεται στην αντίστοιχη ενότητα.

360.ilbdc Ο πυρήνας εφαρμογών αυτός, απευθύνεται στη ρουτίνα διάδοσης συγκρούσεων ενός 3-D lattice Boltzmann flow solver χρησιμοποιώντας έναν TRT-type τελεστή σύγκρουσης για το D3Q19 μοντέλο. Οι επιλυτές ροής lattice Bolzmann χρησιμοποιούν μια διακριτή, με βάση την ταχύτητα,

εξίσωση Boltzmann και διακριτοποιούν το χώρο και το χρόνο με τέτοιο τρόπο ώστε ένα αποκλειστικό (πεπερασμένη διαφορά) αριθμητικό σχήμα με Euler εμπρόσθιο χρονικό βήμα να προκύπτει. Τα αποτελέσματα για τη μηχανική του υγρού ικανοποιούν τις ασυμπίεστες αθερμικές Navier-Stokes εξισώσεις με ακρίβεια 2ης τάξης. Οι δομές δεδομένων του πυρήνα χρησιμοποιούν μια βασισμένη σε λίστα αραιή αναπαράσταση, με αποτέλεσμα μοτίβα έμμεσων προσπελάσεων στα δεδομένα. Ωστόσο, τέτοιες δομές, ειδικά σε προσομοιώσεις ροής υγρών σε πορώδη αντικείμενα ή ροής του αίματος, έχουν αρκετά πλεονεκτήματα στην αποτελεσματική αποκατάσταση της περίπλοκης γεωμετρίας.

362.fma3d Είναι ένα πρόγραμμα για την μέθοδο πεπερασμένων στοιχείων (FEM), σχεδιασμένο να προσομοιώνει την ανελαστική, μεταβατική δυναμική απόκριση των 3-D στερεών και τις δομές που εξαρτώνται από ενστικτωδώς και απότομα προστιθέμενα βάρη. Σε αντίθεση με προγράμματα που χρησιμοποιούν έμμεσης χρονικής ολοκλήρωσης αλγόριθμους, αυτό χρησιμοποιεί ένα μεγάλο αριθμό από σχετικά μικρά χρονικά βήματα, με τη λύση για την επόμενη ρύθμιση του σώματος να είναι άμεση σε κάθε βήμα. Για να μειωθεί περαιτέρω η υπολογιστική πολυπλοκότητα, το πρόγραμμα έχει μια ολοκληρωμένη υλοποίηση του Courant subcycling, στην οποία κάθε στοιχείο ενσωματώνεται με το μέγιστο επιτρεπτό χρονικό βήμα βάσει κριτηρίων τοπικής σταθερότητας. Περισσότερες από 100 parallel do οδηγίες περιλαμβάνονται στον κώδικα με την οδηγία threadprivate να χρησιμοποιείται.

363.swim Το swim είναι ένα benchmark πρόγνωσης καιρού για τη σύγκριση της απόδοσης των τωρινών υπερυπολογιστών. Ο κώδικας είναι μια προσέγγιση πεπερασμένων διαφορών για τις εξισώσεις shallow-water και είναι γνωστός για τους περιορισμούς που έχει σε memory bandwidth. Κάνει υπολογισμούς σε μια 1335x1335 περιοχή ενός πίνακα δεδομένων επαναληπτικά σε 512 χρονικά βήματα.

367.imagick Είναι μια σουίτα λογισμικού για τη δημιουργία, την επεξεργασία, τη σύνθεση και τη μετατροπή εικόνων bitmap. Μπορεί να διαβάσει εικόνες σε διάφορες μορφές (πάνω από 100) όπως DPX, EXR, GIF, JPEG, PDF κ.α. Χρησιμοποιείται για μετασχηματισμούς εικόνας, χρώματα, ειδικά εφέ, για σχεδίαση κειμένου, γραμμών, πολυγώνων, ελλείψεων και καμπυλών Bezier.

370.mgrid331 Προέρχεται από το αντίστοιχο benchmark του NPB 3.3.1. Ο κώδικας χρησιμοποιεί OpenMP οδηγίες για παραλληλοποίηση βρόγχων,

συμπεριλαμβανομένων και collapse clause για την παραλληλοποίηση φωλιασμένων βρόγχων.

371.applu331 Αποτελεί τη λύση πέντε συζευγμένων μη γραμμικών μερικών διαφορικών εξισώσεων, σε ένα τριών διαστάσεων λογικά δομημένο πλέγμα, χρησιμοποιώντας ένα έμμεσο σχήμα ψευδοχρονοπόρευσης, που βασίζεται σε παραγοντοποίηση κατά προσέγγιση με δύο παράγοντες του αραιού πίνακα Jacobi. Αυτό το σχήμα είναι συναρτησιακά αντίστοιχο με ένα μη γραμμικό επαναληπτικό σχήμα μπλοκ SSOR με λεξικογραφική ταξινόμηση. Η χωρική διακριτοποίηση των διαφορικών συντελεστών είναι βασισμένη σε δεύτερης τάξης ακρίβεια της μεθόδου πεπερασμένων όγκων ελέγχου (finite volume method). Ο βαθμός εκμεταλλεύσιμου παραλληλισμού κατά τη διάρκεια αυτής της φάσης είναι της τάξης του $O(N^2)$ ενώ σε άλλες φάσεις είναι $O(N^3)$ και η χωρική κατανομή είναι μη ομοιογενής. Αυτό το γεγονός δημιουργεί προκλήσεις κατά την ανακατανομή των βρόγχων για τη βελτίωση της τοπικότητας της cache. Αυτή η έκδοση προέρχεται από τα NPB 3.3.1.

372.smithwa Το C πρόγραμμα runSequenceAlignment προέρχεται από το πρόγραμμα RUN_secuenceAlignment που είναι γραμμένο σε Matlab από τον Bill Man. Το αρχικό αυτό πρόγραμμα είναι σειριακό και η παράλληλη εκδοχή του με OpenMP αναπτύχθηκε για τα SPEC με τη βοήθεια των υποδείξεων στο αρχείο parallelization.txt στην αρχική έκδοση. Ένας πίνακας ομοιότητας δημιουργείται από το **`genSimMatrix.c`**. Δύο τυχαίες ακολουθίες κωδικωνείων αμινοξέων δημιουργούνται από το **`genScalData.c`** και έξι προκαθορισμένες ακολουθίες για επαλήθευση ενσωματώνονται σε αυτές. Έπειτα στο Kernel 1 κάθε OpenMP νήμα συγκρίνει υποακολουθίες των δύο αρχικών μέσω του αλγορίθμου Smith-Waterman και κατασκευάζει μια λίστα με τις καλύτερες ευθυγραμμίσεις και τα σημεία τέλους τους. Έπειτα, στον Kernel 2A κάθε νήμα OpenMP ή διεργασία MPI ξεκινά σε κάθε σημείο τέλους και ακολουθεί κάθε ευθυγράμμιση πίσω στο σημείο εκκίνησης με αποτέλεσμα στην έξοδο μια λίστα και με τις καλύτερες ευθυγραμμίσεις και τα σημεία εκκίνησης και τέλους και τις ακολουθίες κωδικωνίων. Ο Kernel 2B συγχωνεύει τα αποτελέσματα του 2A από όλα τα νήματα και βγάζει στην έξοδο το αποτέλεσμα.

376.kdtree Το πρόγραμμα αυτό κατασκευάζει ένα k-d δέντρο, χρησιμοποιώντας τυχαία σημεία συντεταγμένων και στη συνέχεια, εκτελεί αναζήτηση σε αυτό, για σημεία τα οποία είναι πλησιέστερα σε κάθε σημείο στο δέντρο. Η φάση της κατασκευής γίνεται από ένα νήμα, αλλά η φάση αναζήτησης γίνεται από πολλά OpenMP νήματα, χρησιμοποιώντας

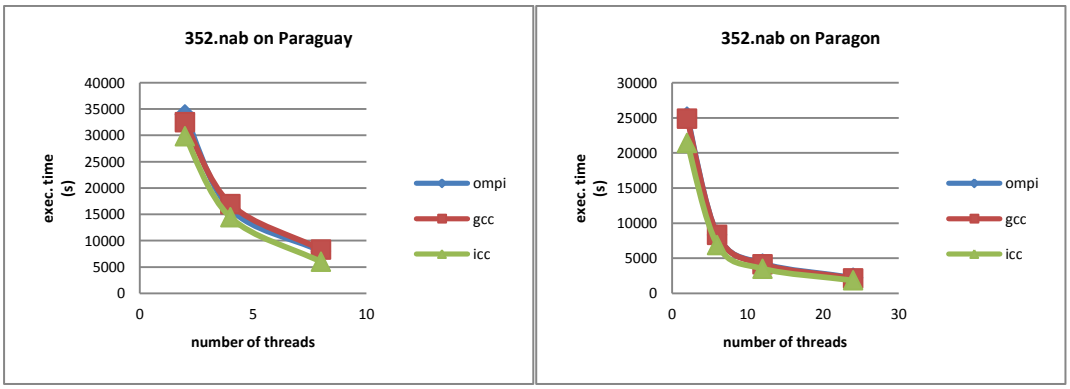
οδηγίες task. Το k-d δέντρο είναι ισορροπημένο και κατασκευάζεται σε $O[n * log(n)]$ χρόνο. Με το που κατασκευάζεται το δέντρο, επισκέπτονται όλα τα σημεία μέσω ενός περιπάτου, και το αντίστοιχο σημείο χρησιμοποιείται για την ερώτηση αναζήτησης για όλα τα σημεία που βρίσκονται μέσα σε μια συγκεκριμένη ακτίνα. Η default τιμή της ακτίνας είναι το 1/10 του εύρους των τυχαίων αριθμών. Στο τέλος, αναφέρονται τα σημεία που βρέθηκαν για κάθε σημείο αναζήτησης, όπως επίσης και ο συνολικός χρόνος εκτέλεσης.

377.DROPS Πρόκειται για έρευνα που στηρίζεται από την Deutsche Forschungsgemeiinschaft (DFG) εντός του SFB 540 (Model-based experimental analysis aims of kinetic phenomena in fluid multi-phase reactive systems). Το λογισμικό στοχεύει στην προσομοίωση ροών αποτελούμενων από δύο φάσεις, για παράδειγμα μια σταγόνα λαδιού στο νερό [Bertakis:2010] ή ένα λεπτό στρώμα υγρού που τρέχει σε έναν τοίχο [Gross:2005]. Ο τομέας των υπολογισμών διαχωρίζεται από μια ιεραρχία τετραεδρικών πλεγμάτων, τα οποία προσαρμοστικά τροποποιούνται ενώ εξελίσσονται κατά το χρόνο της προσομοίωσης. Η παραλληλοποίηση για κοινόχρηστης μνήμης συστήματα είναι βασισμένη στο OpenMP και χρησιμοποιείται για τη μείωση του χρόνου εκτέλεσης των κύριων υπολογιστικά ακριβών σημείων, όπως είναι για παράδειγμα το στήσιμο και η λύση των μη γραμμικών συστημάτων εξισώσεων.

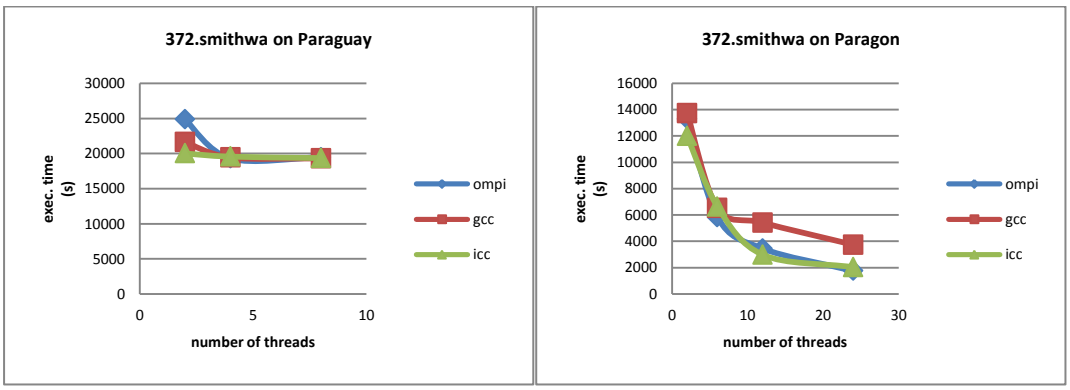
Οι εφαρμογές 352.nab, 367.imagick, 372.smithwa, 358.botsalgn και 359.botsspar είναι γραμμένες σε C, με τις δύο τελευταίες να προέρχονται από τη σουίτα των BOTS με τα αντίστοιχα πειράματα να βρίσκονται στο αντίστοιχο κεφάλαιο. Η εφαρμογή 367.imagick παρουσίασε προβλήματα κατά τη μετάφραση με ορισμένους μεταφραστές. Τα σετ εισόδου είναι τα test (για validation), train και ref, κλιμακωμένου μεγέθους με τα τελευταία να είναι αυτά που χρησιμοποιήθηκαν για τα δικά μας πειράματα. Στα αποτελέσματα δε συμπεριλαμβάνονται οι μετρήσεις για ένα νήμα εξαιτίας του περιορισμού του χρόνου και του πολύ μεγάλου χρόνου εκτέλεσης. Τα γραφήματα δείχνουν τη μεταβολή του χρόνου εκτέλεσης σε seconds (άξονας y) σε σχέση με το πλήθος νημάτων (άξονας χ) με τις κουκίδες να προσδιορίζουν τα αντίστοιχα πειράματα. Στο 352.nab (Σχήμα 7.4) φαίνεται η συμπεριφορά των μεταφραστών να είναι παρόμοια ανεξαρτήτως αρχιτεκτονικής με τις επιδόσεις του Intel ελάχιστα καλύτερες. Η κλιμάκωση της απόδοσης όσο αυξάνουν τα νήματα είναι ένα θέμα στον Paragon. Όσον αφορά το 372.smithwa, ενδιαφέρον παρουσιάζει το γεγονός ότι ενώ στην Paraguay φαίνεται να μην υπάρχει βελτίωση της απόδοσης από 2 νήματα και πάνω για όλους τους μεταφραστές, δε

συμβαίνει το ίδιο στον Paragon με τη συμπεριφορά του προγράμματος να είναι καλύτερη, όπως φαίνεται στο Σχήμα 7.5, δείχνοντας πως η επιλογή NUMA αρχιτεκτονικής μάλλον είναι η κατάλληλη για την εφαρμογή. Η απόδοση του GCC είναι χειρότερη στον Paragon, συγκριτικά με τους άλλους δύο, για πλήθος νημάτων 12 και 24 δείχνοντας βέβαια μια βελτίωση της απόδοσης όσο αυξάνουν τα νήματα.

Σχήμα 7.7: Αποτελέσματα πειραμάτων – 352.nab



Σχήμα 7.8: Αποτελέσματα πειραμάτων – 372.smithwa



7.4 THE OPENMP SOURCE CODE REPOSITORY

Το OpenMP source code repository [12] είναι μια συλλογή εφαρμογών γραμμένων σε C, παραλληλοποιημένων για μέτρηση επιδόσεων στο OpenMP, ενώ περιέχει και κάποιες εφαρμογές γραμμένες σε γλώσσα C++ και Fortran. Ο στόχος της ανάπτυξης αυτών των benchmarks ήταν να υπάρχει κώδικας ελεύθερος στην κοινότητα, όπου εξηγείται λεπτομερώς και είναι κατανοητές όλες οι παράμετροι των προγραμμάτων, όπως για παράδειγμα πως ακριβώς γίνονται οι μετρήσεις και πότε, τι μορφή έχουν τα δεδομένα ή τι στρατηγικές χρησιμοποιούνται για να επιλύσουν το πρόβλημα. Τέτοιες λεπτομέρειες οφείλουν να είναι ξεκάθαρες και κατανοητές στους χρήστες ή ακόμα και σε άτομα που δεν είναι εξοικειωμένα με τη χρήση του μοντέλου OpenMP, ώστε να μπορούν να εξάγουν συμπεράσματα σχετικά με τη σύγκριση διαφορετικών αρχιτεκτονικών σε συστήματα κοινόχρηστης μνήμης και διαφορετικών υλοποιήσεων μεταφραστών. Κάποιες ερωτήσεις για παράδειγμα που πρέπει να έχουν ξεκάθαρη απάντηση στους χρήστες ενός benchmark μπορεί να είναι:

- Τα δεδομένα είναι ακέραιοι ή πραγματικοί;
- Πώς δημιουργούνται τα δεδομένα;
- Έχει υπολογιστεί ο μέσος χρόνος ή ο καλύτερος;
- Όταν μετριέται ο χρόνος έχει υπολογιστεί η δημιουργία του διανύσματος ή μόνο ο χρόνος της ταξινόμησης;
- Ποιος είναι ο αλγόριθμος του σειριακού προγράμματος με βάση τον οποίο υπολογίζεται το speedup;

Ερωτήσεις αυτής της μορφής είναι δύσκολο να απαντηθούν σε μια επιστημονική δημοσίευση εξαιτίας του περιορισμού χώρου, είναι σημαντικό όμως να γνωρίζουν οι χρήστες αυτές τις λεπτομέρειες ώστε να είναι πιο ευκρινή τα συμπεράσματα τους. Παρόμοιες καταστάσεις συναντάμε και σε άλλες δουλειές όπου υπάρχουν υπολογιστικά πειράματα.

Ένα άλλο χαρακτηριστικό αυτών των benchmarks είναι η εις βάθος μελέτη του μοντέλου OpenMP, κάτι που λείπει από benchmarks μέτρησης επιδόσεων, όπως είδαμε στα NAS και SPEC, τα οποία είναι προσανατολισμένα σε ανάλυση της απόδοσης καθαρά και επικεντρώνονται σε αριθμητικές εφαρμογές χρησιμοποιώντας απλά εργαλεία παραλληλισμού, όπως parallel for loops. Είναι σπάνιο να βρεθούν εφαρμογές που χρησιμοποιούν nested παραλληλισμό για παράδειγμα, όπως είδαμε και στα NAS, όπου προστέθηκε σε επόμενες εκδόσεις. Έτσι,

οι συγγραφείς προτείνουν εναλλακτικά benchmarks σε σχέση με τα πιο διαδεδομένα, ώστε να καλύψουν όλες τις ιδιότητες του μοντέλου OpenMP με συγκεκριμένα χαρακτηριστικά.

Δώδεκα είναι τα διαθέσιμα benchmarks, και τα περισσότερα είναι γραμμένα σε C. Αυτό δεν αποτελεί κανόνα, αντιθέτως οι συγγραφείς επιθυμούν κάθε εφαρμογή να είναι γραμμένη σε όλες τις διαθέσιμες γλώσσες, ώστε να συγκριθεί η συμπεριφορά αυτών και των μεταφραστών.

Το είδος των εφαρμογών κυμαίνεται από απλά παραδείγματα χρήσης του OpenMP, σε πολύπλοκο κώδικα με διάφορα παράλληλα loops και παραδείγματα χρήσης σπάνιων και δύσκολων λειτουργιών του OpenMP. Επίσης, για την κατανόηση της συμπεριφοράς διάφορων τρόπων παραλληλισμού, υπάρχουν benchmarks που ο κώδικας τους χαρακτηρίζεται κακός και είναι μαρκαρισμένα ως "wrong". Ακολουθεί μια γενική περιγραφή των εφαρμογών.

Pi. Ο υπολογισμός του π είναι ένας κλασικός αλγόριθμος παράλληλου υπολογισμού. Είναι πολύ απλός κώδικας και πρέπει να έχει σχεδόν γραμμική επιτάχυνση σε οποιοδήποτε παράλληλο σύστημα. Η μόνη παράμετρος του προγράμματος είναι η επιθυμητή ακρίβεια στον υπολογισμό του π.

Mandelbrot set area. Το mandelbrot set area αποτελεί ανοιχτό πρόβλημα. Στατιστικές μέθοδοι έχουν χρησιμοποιηθεί για την προσέγγιση του, καθώς η αναλυτική ακριβής προσέγγιση του είναι αρκετά δύσκολη. Το πρόγραμμα υπολογίζει μια προσέγγιση του προβλήματος χρησιμοποιώντας MonteCarlo sampling. Αυτός ο κώδικας χρησιμοποιείται συχνά για τη μοντελοποίηση καταστάσεων ανισορροπίας φόρτου, καθώς ο αριθμός των επαναλήψεων που απαιτούνται για τον έλεγχο σύγκλισης ενός σημείου στο χώρο των μιγαδικών, αλλάζει ανάλογα με το σημείο. Η παράμετρος εισόδου του προβλήματος είναι το πλήθος των σημείων στο χώρο των μιγαδικών που θέλουμε να εξερευνήσουμε.

Molecular dynamic. Το molecular dynamic είναι μια μέθοδος υπολογιστικής προσομοίωσης για τη μελέτη της φυσικής κίνησης των ατόμων και των μορίων. Το πρόγραμμα είναι μια υλοποίηση του αλγόριθμου Verlet. Δοθέντος των θέσεων, των μαζών και των ταχυτήτων των σωματιδίων, το πρόγραμμα υπολογίζει την ενέργεια του συστήματος και τις δυνάμεις που επιδρούν στα σωματίδια. Μια αριθμητική επαναληπτική διαδικασία χρησιμοποιείται για να υπολογιστεί μια προσέγγιση, η ακρίβεια της οποίας εξαρτάται από τον αριθμό των

βημάτων της προσομοίωσης. Ο αριθμός των σωματιδίων *np* και ο αριθμός των βημάτων είναι οι παράμετροι εισόδου.

Quicksort. Ο αλγόριθμος quicksort είναι από τους πιο διαδεδομένους αλγόριθμους ταξινόμησης. Το πρόγραμμα είναι μια μίξη αναδρομής και παραλληλισμού, ενώ γίνεται χρήση multilevel παραλληλισμού. Ο χρόνος εκτέλεσης εξαρτάται από την εκάστοτε εικόνα του προβλήματος, επομένως υπολογίζεται ο μέσος χρόνος δέκα διαφορετικών στιγμιότυπων τυχαία παραγόμενων. Το μέγεθος του διανύσματος που πρόκειται να ταξινομηθεί είναι και η μόνη παράμετρος εισόδου του προγράμματος.

Divide and conquer fast Fourier transform. Το πρόγραμμα είναι ένας συνδυασμός αναδρομής και παραλληλισμού στο γνωστό αλγόριθμο Fast Fourier Transform (FFT). Υπολογίζει το διακριτό μετασχηματισμό Fourier ενός σήματος, το οποίο αναπαρίσταται από ένα διάνυσμα *N* μιγαδικών αριθμών. Χρησιμοποιεί multilevel παραλληλισμό και η παράμετρος εισόδου είναι το μέγεθος *N* του διανύσματος.

Bailey's 6 "step" FFT. Μια άλλη υλοποίηση του FFT που χρησιμοποιεί παραλληλισμό με tasks. Είναι μέρος του CMU task parallel suite.

Loops with dependencies. Η εφαρμογή αυτή περιλαμβάνει παραδείγματα παραλληλοποίησης βρόγχων με εξαρτήσεις προς τα εμπρός και προς τα πίσω, καθώς επίσης και "κακά" παραλληλοποιημένους κώδικες. Ένας από τους βρόγχους έχει παραλληλοποιηθεί με τη χρήση εικονικού pipelining μεταξύ των νημάτων. Αυτή η εφαρμογή έχει ως σκοπό μια πρώτη επαφή με την παραλληλοποίηση μη τετριμμένων βρόγχων στο OpenMP.

LU decomposition. Στη γραμμική άλγεβρα η LU παραγοντοποίηση είναι μια διαδικασία αποσύνθεσης ενός μητρώου *M* σε γινόμενο ενός κάτω τριγωνικού πίνακα *L*, και ενός άνω τριγωνικού πίνακα *U*. Το πρόγραμμα είναι μα υλοποίηση του αλγόριθμου LU για πυκνούς πίνακες. Είναι ένα παράδειγμα χρήσης του parallel-for με stride schedule, για την εξισορρόπηση φόρτου και ανομοιόμορφης κατανομής φόρτου μεταξύ των επαναλήψεων.

Graph search. Η εφαρμογή αυτή είναι ένα παράδειγμα υλοποίησης master-slave για την παραλληλοποίηση της κατά βάθος αναζήτησης, σε ένα άκυκλο κατευθυνόμενο γράφημα. Χρησιμοποιεί ένα pool από tasks, η πρόσβαση στο οποίο ελέγχεται με συγχρονισμό με critical sections για να αποφευχθεί η διπλή αναζήτηση σε μονοπάτια από διαφορετικούς επεξεργαστές. Επομένως, η κλιμακωσιμότητα ίσως είναι θέμα εξαιτίας του έντονου συγχρονισμού. Η είσοδος είναι έτοιμα δείγματα γραφημάτων.

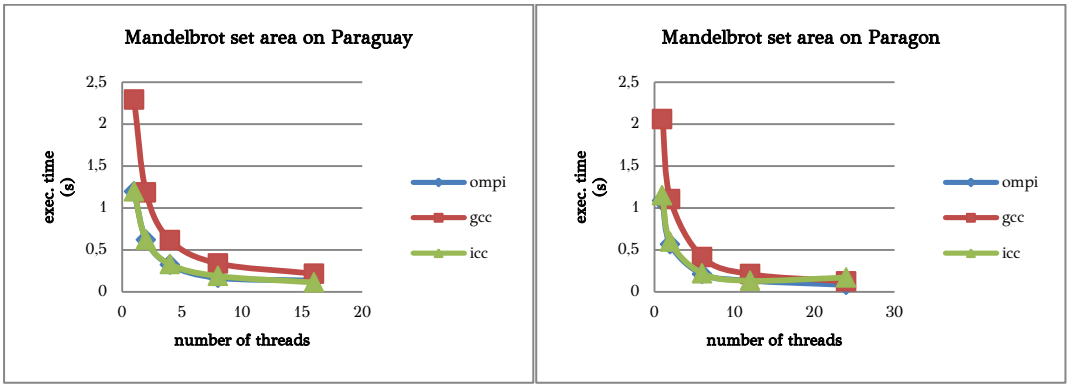
Jacobi. Το πρόγραμμα αυτό περιλαμβάνει τρεις διαφορετικές εκδοχές σε C και δέκα σε Fortran για την επίλυση διακριτοποίησης πεπερασμένων διαφορών της εξίσωσης Helmholtz, χρησιμοποιώντας την επαναληπτική μέθοδο Jacobi. Οι διαφορετικές υλοποιήσεις δείχνουν πώς η επίδοση και το overhead του OpenMP μπορούν να αλλάξουν με χρήση κατάλληλης στρατηγικής, όπως απόσπαση των παράλληλων περιοχών, χρήση barriers, τεχνικές pipelining κλπ.

Οι άλλες δύο εφαρμογές είναι η Cellular Automaton γραμμένη σε Fortran και η SortOpenMP γραμμένη σε C++.

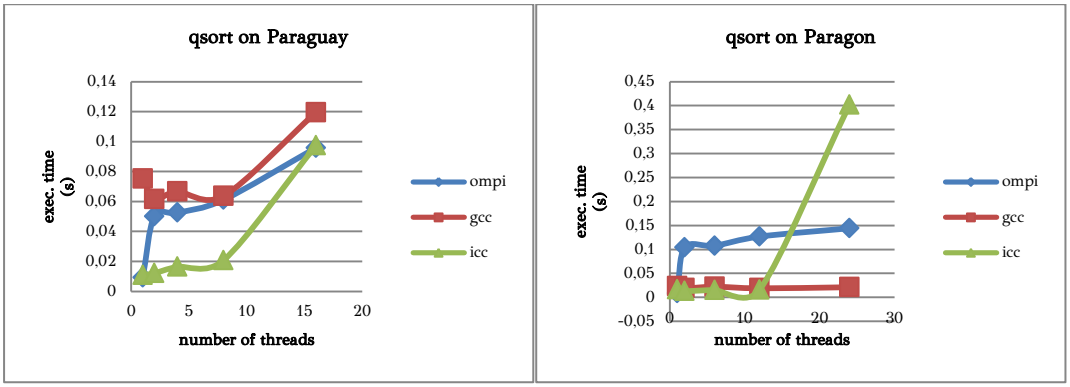
Το project αυτό δεν προτείνεται από μόνο του ως benchmark για το OpenMP από τους συγγραφείς, αλλά αναφέρουν ότι δεν αποκλείουν μια προσπάθεια, ώστε να γίνει κάτι τέτοιο στο μέλλον, πράγμα το οποίο μάλλον δεν έγινε μέχρι σήμερα και πιθανώς εγκαταλείφθηκε η προσπάθεια. Πάντως οι εφαρμογές αποτελούν καλά παραδείγματα χρήσης κάποιων λειτουργιών του OpenMP και θα μπορούσαν να συμπεριληφθούν σε άλλες σουίτες.

Ακολουθούν δύο από τα αποτελέσματα των πειραμάτων, με πολλά από τα πειράματα που δε συγκαταλέγονται να έχουν περίεργη συμπεριφορά και να έχουν μεγάλη απόκλιση στους χρόνους από εκτέλεση σε εκτέλεση. Πάντως, είναι στο χέρι του χρήστη να ερμηνεύσει αυτές τις περίεργες συμπεριφορές και αν μπορεί να βγάλει συμπεράσματα για τη δικιά του υλοποίηση. Στα γραφήματα φαίνεται η μεταβολή του χρόνου εκτέλεσης σε seconds (άξονας y) σε σχέση με το πλήθος νημάτων (άξονας χ) με τις κουκίδες να προσδιορίζουν τα αντίστοιχα πειράματα. Το Σχήμα 7.6 δείχνει τα αποτελέσματα του Mandelbrot set area και φαίνεται η συμπεριφορά των μεταφραστών παρόμοια όσον αφορά τη κλιμάκωση της απόδοσης σε σχέση με το πλήθος των νημάτων, ανεξαρτήτως αρχιτεκτονικής, ενώ οι επιδόσεις του GCC είναι ελαφρώς χειρότερες. Το Σχήμα 7.7 δείχνει τα αποτελέσματα του qsort. Αποτελεί χαρακτηριστικό παράδειγμα της συμπεριφοράς που συναντάμε και σε άλλα πειράματα που δε συγκαταλέγονται και έχει να κάνει με την περίεργη συμπεριφορά της εφαρμογής που δείχνει να μην εκμεταλλεύεται τον παραλληλισμό που προσφέρουν οι δύο αρχιτεκτονικές, ενώ η συμπεριφορά του Intel στον Paragon διαφοροποιείται από τους υπόλοιπους στα 24 νήματα.

Σχήμα 7.9: Αποτελέσματα πειραμάτων – Mandelbrot set area



Σχήμα 7.10: Αποτελέσματα πειραμάτων - qsort



ΚΕΦΑΛΑΙΟ 8
ΣΟΥΙΤΕΣ BENCHMARK ΕΙΔΙΚΟΥ ΣΚΟΠΟΥ

8.1 BARCELONA OPENMP TASKS SUITE

Το Barcelona OpenMP Tasks Suite [13] αποτελεί ένα σύνολο εφαρμογών που εκμεταλλεύονται κανονικό και ακανόνιστο παραλληλισμό και βασίζονται σε tasks. Με την έκδοση 3.0 του OpenMP, τα tasks προστέθηκαν στο μοντέλο και έδωσαν την ευκαιρία σε εφαρμογές να εκμεταλλευτούν τον ακανόνιστο παραλληλισμό, κάτι που γίνεται όλο ένα και πιο σημαντικό με τις σύγχρονες πολυπύρηνες αρχιτεκτονικές που προσφέρουν πολλές εναλλακτικές στην εκτέλεση παράλληλων εφαρμογών, σε σχέση με τις παραδοσιακές SMP αρχιτεκτονικές. Επομένως, είναι πολύ χρήσιμο να υπάρχει ένα σύνολο από benchmarks για την αποτίμηση της ιδιότητας αυτής του μοντέλου.

Σκοπός των BOTS είναι η αξιολόγηση των υλοποιήσεων του OpenMP από ερευνητές και προμηθευτές, όπως επίσης και οι προγραμματιστές να έχουν μια συλλογή παραδειγμάτων που χρησιμοποιούν το μοντέλο των tasks.

Ο ορισμός των tasks από το OpenMP αφήνει μια σχετική ελευθερία στο πώς θα υλοποιηθούν. Για παράδειγμα, βάζει λίγους περιορισμούς στη χρονοδρομολόγιση των tasks, ενώ μια ιδιότητα που δεν προσδιορίζει, είναι αν η εναλλαγή των tasks πρέπει να υποστηρίζεται. Μπορεί δηλαδή η εκτέλεση των tasks -η οποία έχει ανασταλεί- να μπορεί να συνεχιστεί από οποιοδήποτε άλλο νήμα (*untied)* ή όχι (*tied)* ;

Ενώ μερικά από τα benchmarks έχουν αναπτυχθεί από τους ίδιους τους συγγραφείς των BOTS, τα περισσότερα είναι ήδη υπάρχουσες εφαρμογές διαθέσιμες στο κοινό και προερχόμενες από το Cilk project, το Application Kernel Matrix project και το Olden suite και έχουν μεταφερθεί στο OpenMP.

Για το λόγο ότι τη δεδομένη στιγμή τα trade-offs όσον αφορά το μοντέλο tasking του OpenMP δεν είναι σαφή και εξαρτώνται σε μεγάλο βαθμό από την ποιότητα της υλοποίησης, έχουν αναπτυχθεί διαφορετικές εκδοχές για κάθε benchmark με διαφορετικά χαρακτηριστικά. Όλα τα benchmarks έχουν εκδοχές με tied και untied tasks τα οποία επιτρέπουν τον πειραματισμό στο πώς συμπεριφέρεται το κάθε ένα. Πολλά από τα benchmarks δημιουργούν ένα πολύ μεγάλο πλήθος από μικρά tasks. Λόγω

αυτού, έχουν αναπτυχθεί τρείς διαφορετικές εκδοχές αυτών, ανάλογα με το μηχανισμό *cutoff* που χρησιμοποιούν:

- Μια εκδοχή, η οποία δεν βάζει περιορισμό στη δημιουργία των tasks και ρίχνει την ευθύνη στην υλοποίηση. Αυτό είναι το ιδανικό για τον προγραμματιστή, αφού η υλοποίηση η ίδια βάζει το όριο στη δημιουργία των tasks. Ο μηχανισμός αναφέρεται σαν *none*.
- Άλλη εκδοχή στην οποία η εφαρμογή ελέγχει τη δημιουργία των tasks μέσω ενός if clause στην οδηγία task. Η συνθήκη αυτή διαφέρει από benchmark σε benchmark αλλά συνήθως έχει να κάνει με το βάθος του recursive μονοπατιού. Αναφέρεται ως *pragma_if.*
- Άλλη μια εκδοχή στην οποία η εφαρμογή ελέγχει τη δημιουργία των tasks "χειροκίνητα" μέσω κλήσης μιας συνάρτησης με οδηγίες με την ίδια συνθήκη με πριν. Αναφέρεται ως *manual*.

Μερικά benchmarks επιτρέπουν είτε τη δημιουργία των tasks από πολλαπλά νήματα (για παράδειγμα tasks κάτω από for/sections construct), είτε από ένα μόνο νήμα (για παράδειγμα tasks κάτω από single construct). Στις περιπτώσεις αυτές, εκδοχές του ίδιου benchmark και με τις δύο επιλογές έχουν αναπτυχθεί, ώστε να αποτιμήσουν την υποστήριξη και των δύο.

Ο παραλληλισμός με tasks, λόγω της ακανόνιστης φύσης του, περιλαμβάνει μια μορφή τυχαιότητας στην εκτέλεση του (για παράδειγμα το κλάδεμα σε αλγόριθμους αναζήτησης), με την ιδιαιτερότητα αυτή να μην είναι κατάλληλη για benchmarking. Οι εφαρμογές τέτοιου είδους επομένως που συγκαταλέγονται στα BOTS, έχουν τροποποιηθεί από τους συγγραφείς, ώστε να κρατείται υπό έλεγχο η τυχαιότητα αυτή.

Η επαλήθευση του αποτελέσματος των benchmarks είναι ένα άλλο πολύ σημαντικό χαρακτηριστικό, καθώς δίνει τη δυνατότητα να ελεγχθεί το κατά πόσο η υλοποίηση είναι σωστή. Επομένως, μερικά benchmarks, εφαρμόζουν μια μέθοδο επαλήθευσης πριν την έξοδο του αποτελέσματος. Κάποια άλλα τροφοδοτούνται με δεδομένα ελέγχου στην είσοδο, ώστε να ελέγξουν το αποτέλεσμα με βάση αυτά και να γίνει η επαλήθευση. Όταν δεν είναι εφικτό να πραγματοποιηθούν τα παραπάνω, μια σειριακή εκδοχή της εφαρμογής χρησιμοποιείται όταν ο χρήστης το απαιτήσει, ώστε να επικυρωθεί το αποτέλεσμα της παράλληλης εκτέλεσης με βάση το σειριακό.

Για κάθε ένα benchmark, ένα σετ από διαφορετικά δεδομένα για είσοδο ορίζονται, για να τεστάρουν την εφαρμογή κάτω από διαφορετικά σενάρια.

- *Test:* Η κλάση *test* είναι πολύ μικρή και χρησιμοποιείται για έναν γρήγορο έλεγχο ότι το benchmark δουλεύει.
- *Small:* Η κλάση *small* έχει σχεδιαστεί, ώστε και η απαίτηση για μνήμη να μη ξεπερνάει το 1Gb αλλά και ο χρόνος εκτέλεσης του σειριακού προγράμματος να μην ξεπερνάει το ένα λεπτό (SGI Altix 4700 system).
- *Medium:* Η κλάση *medium* έχει σχεδιαστεί ανάλογα, με τις απαιτήσεις για μνήμη να μη ξεπερνάνε τα 4Gb και ο χρόνος εκτέλεσης του σειριακού προγράμματος τα 10 λεπτά.
- *Large:* Ανάλογα με τα προηγούμενα, 10Gb και μισή ώρα.

Ακολουθεί μια σύντομη περιγραφή των εφαρμογών που χρησιμοποιούνται από το Barcelona OpenMP task Suite.

Allignment: Ευθυγραμμίζει κάθε ακολουθία πρωτεϊνών από ένα αρχείο εισόδου με κάθε άλλη ακολουθία σύμφωνα με τον αλγόριθμο των Myers και Miller. Οι ευθυγραμμίσεις έχουν ένα σκορ και το καλύτερο για κάθε ζευγάρι επιστρέφεται στο αποτέλεσμα. Η μέθοδος της βαθμολόγησης είναι ένας δυναμικός αλγόριθμος. Χρησιμοποιεί ένα διάνυσμα βαρών, ώστε να δώσει τιμές σε αναντιστοιχίες και αναθέτει ποινές για ανοίγματα και επέκταση κενών.

Σε αυτή την εφαρμογή έχει παραλληλοποιηθεί η εξωτερική λούπα μέσω της οδηγίας **`omp for`** και δημιουργούνται tasks μέσα στην περιοχή αυτή. Αυτό επιτρέπει στην εφαρμογή να σπάσει την επανάληψη όταν το πλήθος των νημάτων είναι μεγάλο, σε σχέση με το πλήθος των επαναλήψεων, και όταν υπάρχει ανισορροπία.

FFT: Υπολογίζει τον μονοδιάστατο Fast Fourier Transform ενός διανύσματος *n* μιγαδικών τιμών χρησιμοποιώντας τον αλγόριθμο Cooley-Tukey. Αυτός είναι ένας διαίρει και βασίλευε αλγόριθμος ο οποίος αναδρομικά σπάει έναν Discrete Fourier Transform σε πολλούς μικρότερους DFT. Σε κάθε μία διαίρεση πολλά νέα tasks δημιουργούνται.

Fibonacci: Υπολογίζει το ν-οστό αριθμό fibonacci χρησιμοποιώντας έναν αναδρομικό παραλληλισμό. Ενώ δεν είναι αποτελεσματικός τρόπος υπολογισμού αριθμού Fibonacci, είναι χρήσιμος καθώς αποτελεί μια απλή περίπτωση ενός deep δέντρου αποτελούμενου από λεπτόκοκκα tasks.

Περιλαμβάνει εκδοχές που χρησιμοποιούν ένα κατώφλι με βάση το βάθος του δέντρου ώστε να αποφύγει τη δημιουργία πολύ λεπτόκοκκων tasks.

Floorplan: Ο πυρήνας αυτός υπολογίζει την ιδανική floorplan κατανομή ενός αριθμού κελιών. Ο αλγόριθμος παίρνει σαν είσοδο ένα αρχείο με την περιγραφή των κελιών και επιστρέφει την ελάχιστη περιοχή, η οποία περιλαμβάνει όλα τα κελιά. Αυτή η περιοχή βρίσκεται μέσω ενός αναδρομικού κλαδιού και μιας αναζήτησης στα όρια. Τα tasks δημιουργούνται ιεραρχικά για κάθε κλαδί του χώρου της λύσης. Η κατάσταση του αλγορίθμου πρέπει να αντιγραφεί στο νέο task που δημιουργήθηκε. Αυτό υπονοεί επιπλέον συγχρονισμό, ώστε να διατηρηθεί η κατάσταση όπως ήταν.

Η εφαρμογή χρησιμοποιεί ένα μηχανισμό κλαδέματος, ώστε να μειώσει το χώρο αναζήτησης. Αυτό το κλάδεμα είναι ακανόνιστο και πολύ επιθετικό, με αποτέλεσμα το δέντρο να μην είναι καθόλου ισορροπημένο. Το κλάδεμα γίνεται με βάση το καλύτερο αποτέλεσμα που έχει βρεθεί μέχρι εκείνη τη στιγμή, το οποίο δημιουργεί μια μορφή τυχαιότητας. Επειδή όλοι οι κόμβοι του δέντρου έχουν περίπου το ίδιο φόρτο υπολογισμών, υπολογίζεται το πλήθος των κόμβων που έχουν επισκεφτεί για να βρεθεί μια λύση. Όπως και στο fibonacci υπάρχουν δύο εκδοχές, που χρησιμοποιούν ένα μηχανισμό cut-off βασιζόμενο στο βάθος του δέντρου, ώστε να μη δημιουργούνται πολύ λεπτόκοκκα tasks.

Health: Προσομοιώνει το Columbian Health Care System. Χρησιμοποιεί πολυεπίπεδες λίστες, στις οποίες κάθε στοιχείο αντιπροσωπεύει ένα χωριό με μια λίστα πιθανών ασθενών και ένα νοσοκομείο. Κάθε νοσοκομείο έχει πολλαπλές διπλά συνδεδεμένες λίστες, οι οποίες αναπαριστούν την πιθανή κατάσταση του ασθενή (σε αναμονή, σε φάση αξιολόγησης, σε θεραπεία ή αναμένοντας για ανακατανομή). Σε κάθε χρονική στιγμή όλοι οι ασθενείς προσομοιώνονται σύμφωνα με κάποιες πιθανότητες (να αρρωστήσουν, να χρειαστούν φαρμακευτική αγωγή, να ανακατανεμηθούν σε ένα ανωτέρου επιπέδου νοσοκομείο κλπ). Ένα task δημιουργείται για κάθε χωριό που προσομοιώνεται. Όταν τα κατώτερα επίπεδα έχουν προσομοιωθεί, γίνεται συγχρονισμός. Χρησιμοποιείται ένας μηχανισμός cut-off, με βάση το επίπεδο που βρίσκεται το χωριό στην ιεραρχία.

Οι πιθανότητες σε διαφορετικά βήματα της προσομοίωσης εισάγουν κάποια τυχαιότητα. Για να αποφευχθεί αυτό, χρησιμοποιείται αντί για ένα seed για τυχαίους αριθμούς, ξεχωριστά seeds για κάθε χωριό. Με αυτόν τον τρόπο, οι πιθανότητες για κάθε χωριό, οι οποίες υπολογίζονται από

κάθε ένα task, είναι ίδιες ανάμεσα σε κάθε εκτέλεση και δεν επηρεάζονται από τα άλλα tasks.

***N* Queens**: Υπολογίζει όλες τις πιθανές λύσεις στο πρόβλημα των *N* Queens, δηλαδή όλες τις πιθανές θέσεις που μπορούν να τοποθετηθούν οι *N* βασίλισσες σε μια σκακιέρα *NxN* ώστε καμία να μην απειλεί κάποια άλλη. Χρησιμοποιεί έναν backtracking αλγόριθμο αναζήτησης με κλάδεμα. Ένα task δημιουργείται για κάθε βήμα της λύσης. Όπως και στο floorplan, η κατάσταση του αλγορίθμου (parent state) πρέπει να αντιγραφεί στα παιδιά, στα tasks δηλαδή που δημιουργούνται από το parent state. Το κλάδεμα συμβαίνει στα κλαδιά στα οποία δεν περιέχεται η σωστή λύση. Αυτό δημιουργεί μια ανισορροπία στο δέντρο, όπως επίσης και μια τυχαιότητα, όχι όμως τόσο όσο στο floorplan, καθώς δεν εξαρτάται από κάποια τρέχων λύση στον αριθμό των κόμβων που απομένουν. Για να αποφευχθεί αυτό, αντί να βρίσκεται μία λύση στο πρόβλημα, ο πυρήνας βρίσκει όλες τις πιθανές λύσεις. Αυτό εξασφαλίζει ότι η εφαρμογή έχει πάντα το ίδιο υπολογιστικό φόρτο. Για να προσδιοριστούν αυτές οι λύσεις που βρέθηκαν από τα διαφορετικά tasks, μια προσέγγιση θα ήταν να περιβληθεί η συγχώνευση με οδηγίες **`critical`**, όμως αυτό θα είχε ως αποτέλεσμα μεγάλο ανταγωνισμό. Επομένως χρησιμοποιούνται **`threadprivate`** μεταβλητές. Με αυτόν τον τρόπο όλα τα νήματα μπορούν να συγχωνεύσουν τις λύσεις που βρίσκουν.

Sort: Ταξινομεί μια τυχαία μετάθεση 32-bit αριθμών με μια γρήγορη παράλληλη εκδοχή του merge-sort αλγόριθμου. Πρώτα διαιρεί έναν πίνακα με στοιχεία στα δύο, ταξινομώντας το κάθε μισό αναδρομικά και έπειτα συγχωνεύοντας τα ταξινομημένα τμήματα μέσω μιας παράλληλης μεθόδου διαίρει και βασίλευε αντί του κλασικού σειριακού merge. Τα tasks χρησιμοποιούνται για κάθε split και merge. Όταν ο πίνακας γίνει πολύ μικρός, χρησιμοποιείται σειριακά ο quicksort αλγόριθμος για την ταξινόμηση, ώστε να αυξηθεί η λεπτομερειακότητα του task. Για να αποφευχθεί το overhead του quicksort, ο αλγόριθμος insertion sort χρησιμοποιείται για πολύ μικρούς πίνακες (κάτω από το όριο των 20 στοιχείων).

SparseLU: Υπολογίζει μία LU παραγωντοποίηση πάνω σε αραιούς πίνακες. Ένας πρώτου επιπέδου πίνακας συντίθεται με δείκτες προς μικρότερους υποπίνακες, οι οποίοι ίσως να μην κατανεμηθούν. Εξαιτίας της αραιότητας των πινάκων μπορεί να δημιουργηθεί ανισορροπία. Το μέγεθος του πίνακα και των υποπινάκων μπορεί να προσδιοριστεί κατά το χρόνο εκτέλεσης. Ενώ μπορεί να χρησιμοποιηθεί dynamic schedule για να μειωθεί η

ανισορροπία, μια λύση με tasks δείχνει να επιφέρει καλύτερα αποτελέσματα. Σε κάθε φάση του SparseLU, δημιουργείται ένα task για κάθε ένα block του πίνακα το οποίο δεν είναι κενό.

Δύο εκδοχές του benchmark έχουν αναπτυχθεί, μία που δημιουργεί tasks μέσα σε ένα **`single`** και μία που δημιουργεί τα tasks μέσα σε μία **`omp for`** περιοχή, ώστε να επιτρέπει σε πολλαπλά νήματα να δημιουργούν tasks για κάθε φάση.

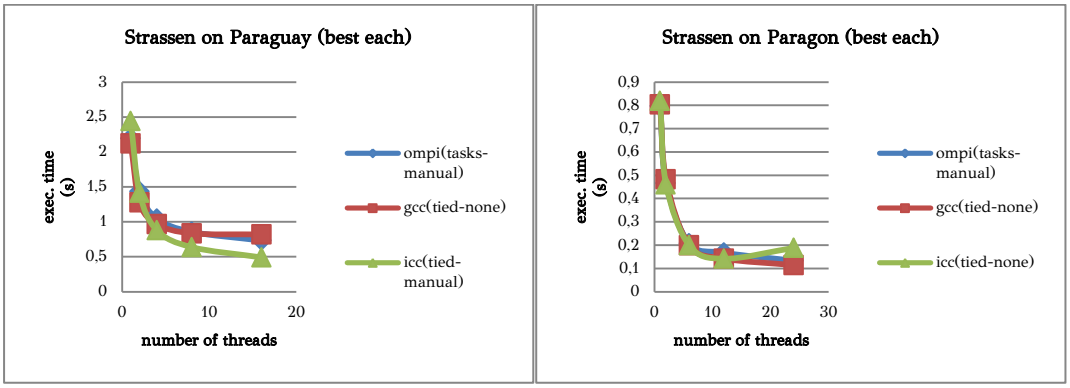
Strassen: Χρησιμοποιεί ιεραρχική αποσύνθεση ενός πίνακα για τον πολλαπλασιασμό μεγάλων πυκνών πινάκων. Η αποσύνθεση γίνεται διαιρώντας κάθε διάσταση του πίνακα σε δύο τμήματα ίσου μεγέθους. Για κάθε αποσύνθεση ένα task δημιουργείται. Για να αποφευχθεί η δημιουργία πολύ μικρών tasks, έχουν αναπτυχθεί εκδοχές που χρησιμοποιούν μηχανισμούς cut-off με βάση το βάθος.

Ορισμένες εφαρμογές, όπως οι Fib, NQueens, Floorplan χαρακτηρίζονται από πολλά λεπτόκοκκα tasks και επομένως η πρόκληση είναι να εκμεταλλευτούν το διαθέσιμο παραλληλισμό ενώ ταυτόχρονα να μειώσουν τα σχετικά overheads. Σε άλλες περιπτώσεις (allignment, sparseLU), οι εφαρμογές δημιουργούν σχετικά χονδρόκοκκα tasks και η πρόκληση είναι να διατηρηθεί μια ισορροπία στο φόρτο του κάθε task.

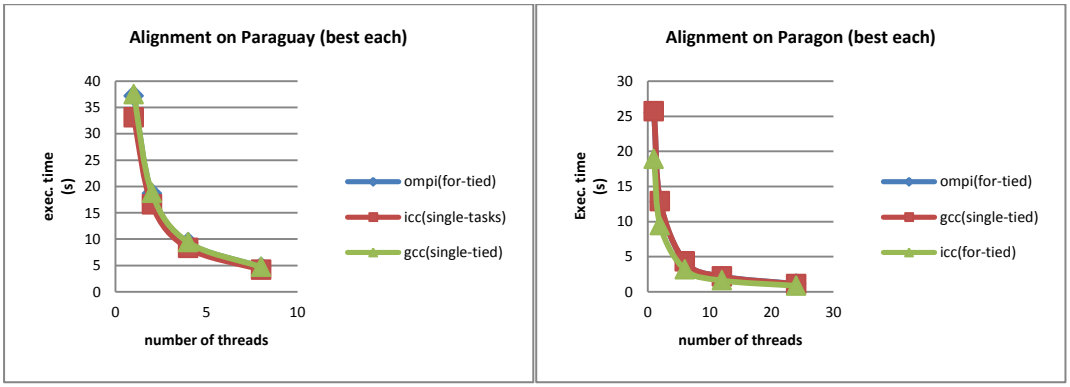
Τα αποτελέσματα αφορούν τις εφαρμογές στις οποίες είναι ευδιάκριτα όλα τα χαρακτηριστικά των συμπεριφορών της εκάστοτε υλοποίησης, με αυτά που δε συγκαταλέγονται να έχουν παρόμοια συμπεριφορά. Σαν γενική παρατήρηση, η υλοποίηση του GCC δεν ανταποκρίνεται στις απαιτήσεις όλων των εφαρμογών, με τους χρόνους σε μερικές εκδοχές των προγραμμάτων, όπως σε αυτές που το cutoff είναι ευθύνη της υλοποίησης, να είναι απαγορευτικά υψηλές και αδύνατον να τοποθετηθούν στο ίδιο γράφημα με τους OMPi και Intel (περιπτώσεις Floorplan και NQueens) ή να μην τερματίζουν καν. Πιθανώς να μην υποστηρίζεται η αντίστοιχη λειτουργία. Τα γραφήματα δείχνουν τη συμπεριφορά του χρόνου εκτέλεσης των εφαρμογών (άξονας y) σε σχέση με το πλήθος νημάτων (άξονας χ) και οι κουκίδες αφορούν τα αντίστοιχα πειράματα. Από τις διαφορετικές εκδοχές των benchmarks συγκαταλέγονται οι καλύτερες για τον καθένα. Το Σχήμα 8.1 δείχνει τη συμπεριφορά του Strassen για τους τρεις μεταφραστές. Οι καλύτεροι χρόνοι για τον OMPi είναι με untied tasks και manual cutoff, για τον GCC με tied και cutoff αυτό που βάζει η υλοποίηση και για τον Intel tied tasks και manual cutoff στην Paraguay και cutoff που ορίζει η υλοποίηση στον Paragon. Η συμπεριφορά της εφαρμογής είναι παρόμοια για τους μεταφραστές ανεξαρτήτως

αρχιτεκτονικής, με τις επιδόσεις του Intel ελαφρώς καλύτερες στην Paraguay συγκριτικά με τους άλλους. Για το Alignment (Σχήμα 8.2), ο OMPi έχει καλύτερες επιδόσεις στην εκδοχή κατά την οποία τα tasks είναι tied και δημιουργούνται κάτω από for sections. Οι καλύτερες επιδόσεις για τον GCC είναι με την εκδοχή κατά την οποία τα νήματα δημιουργούνται κάτω από single sections, όπως και για τον Intel στην Paraguay, ενώ στον Paragon από πολλαπλά νήματα. Οι επιδόσεις των μεταφραστών είναι παρόμοιες. Για το Floorplan (Σχήμα 8.3) και για τους δύο μεταφραστές η καλύτερη εκδοχή είναι με tied tasks και manual cutoff. Οι επιδόσεις είναι παρόμοιες ανεξαρτήτως αρχιτεκτονικής και μεταφραστή, με μόνη διαφορά την καλύτερη κλιμάκωση των επιδόσεων για 8 και 24 νήματα αντίστοιχα του Intel, με τον OMPi να μη διαχειρίζεται τόσο καλά τα πολλά λεπτόκοκκα tasks που δημιουργεί η εφαρμογή. Στο Σχήμα 8.4 φαίνεται η συμπεριφορά των διαφορετικών εκδοχών του benchmark NQueens για τους μεταφραστές OMPi και Intel. Οι καμπύλες αφορούν τις διαφορετικές εκδοχές του benchmark, οι οποίες είναι tasks tied και untied, και οι διαφορετικοί μηχανισμοί cutoff που προαναφέρθηκαν, με το *final* να υποδηλώνει ότι tasks που δημιουργούνται από αυτό που φέρει το clause δεν μπορούν να δημιουργήσουν άλλα. Σε παρένθεση είναι το βάθος του recursive μονοπατιού το οποίο είναι 3. Ο OMPi γενικά έχει καλύτερες επιδόσεις από τον Intel στο συγκεκριμένο benchmark, αλλά δείχνει να υστερεί στην υλοποίηση του μηχανισμού cutoff με if clauses. Ο Intel από την άλλη έχει αισθητά καλύτερες επιδόσεις στις εκδοχές με manual cutoff πράγμα που υπονοεί ότι οι υπόλοιπες υλοποιήσεις υστερούν σε σχέση με τον OMPi. Γενικά συμπεράσματα από όλα τα πειράματα, δείχνουν κακή υλοποίηση του GCC όσον αφορά τα tasks και πολύ καλή του OMPi - αν εξαιρεθεί η περίπτωση του if clause - και ότι η διαφορά ανάμεσα σε tied και untied tasks είναι ελάχιστη για όλους τους μεταφραστές, κάτι που δείχνει ότι πιθανώς να μην υποστηρίζονται untied tasks. Τέλος, η αρχιτεκτονική δεν έχει επίδραση στη συμπεριφορά των εφαρμογών, κάτι που θα περίμενε κανείς λόγω του ανομοιόμορφου χρόνου προσπέλασης μνήμης στο NUMA μοντέλο, σε συνδυασμό με τη φύση των tasks.

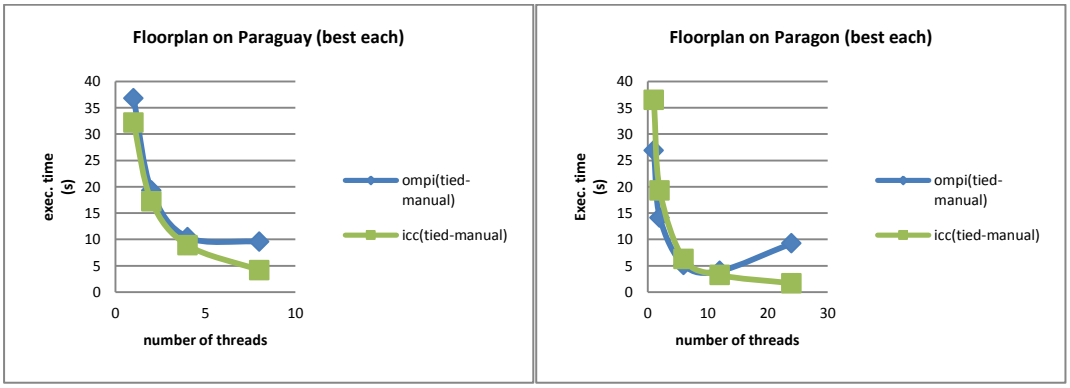
Σχήμα 8.1: Αποτελέσματα πειραμάτων - Strassen



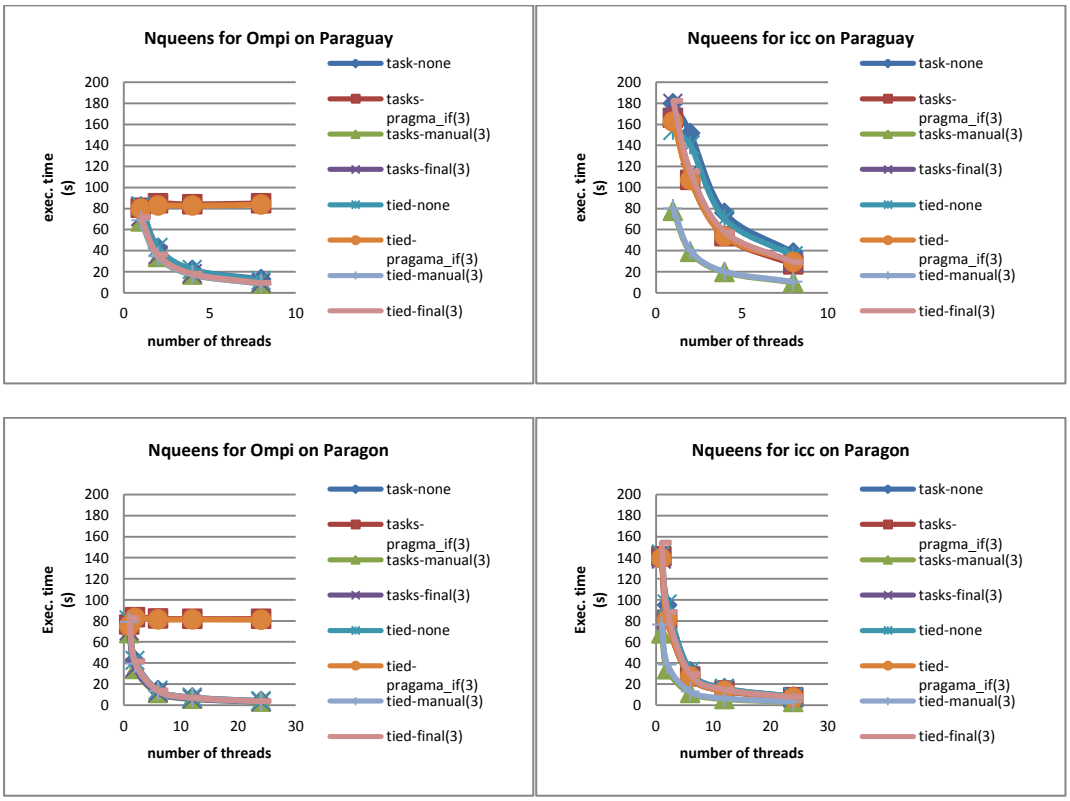
Σχήμα 8.2: Αποτελέσματα πειραμάτων - Alignment



Σχήμα 8.3: Αποτελέσματα πειραμάτων - Floorplan



Σχήμα 8.4: Αποτελέσματα πειραμάτων – Nqueens



8.2 UTS

Το UTS (Unbalanced Tree Search) benchmark [14] εκτελεί μια εξαντλητική (brute force) αναζήτηση σε ένα μη ισορροπημένο δέντρο χρησιμοποιώντας κλοπή εργασίας (work stealing), ως μηχανισμό μείωσης της ανισορροπίας φόρτου.

Το πρόβλημα που εισάγεται έχει να κάνει με την παράλληλη εξερεύνηση ενός μη ισορροπημένου δέντρου και αποτελεί μέσο μελέτης της συμπεριφοράς και της απόδοσης εφαρμογών που απαιτούν συνεχή δυναμική εξισορρόπηση φόρτου. Άλλες εφαρμογές, που ανήκουν στην κλάση αυτή, αφορούν προβλήματα αναζήτησης και βελτιστοποίησης που πρέπει να απαριθμήσουν ένα μεγάλο χώρο καταστάσεων μιας άγνωστης και απρόβλεπτης δομής.

Το πρόβλημα που περιγράφεται από το UTS είναι η καταμέτρηση των κόμβων ενός δέντρου, που έχει κατασκευαστεί με έναν τρόπο ο οποίος δεν έχει εκφραστεί ευθέως και το οποίο είναι παραμετροποιήσημο ως προς το σχήμα, το μέγεθος, το βάθος και την ανισορροπία. Η έμμεση κατασκευή του δέντρου έγκειται στο γεγονός ότι κάθε κόμβος περιέχει όλη την πληροφορία που χρειάζεται για την κατασκευή κόμβων που είναι παιδιά του. Επομένως, ξεκινώντας από τη ρίζα, το δέντρο μπορεί να διασχιστεί παράλληλα με οποιαδήποτε σειρά, αρκεί κάθε γονέας να επισκέπτεται πριν τα παιδιά του. Η ανισορροπία ενός δέντρου είναι μια μετρική της ποικιλίας στο μέγεθος των υπο-δέντρων του. Δέντρα με πολύ μεγάλη ανισορροπία αποτελούν πρόκληση για την παράλληλη διάσχισή τους, διότι το έργο που απαιτείται από τα διαφορετικά υπο-δέντρα μπορεί να διαφέρει σημαντικά και επομένως, μια αποτελεσματική στρατηγική δυναμικής κατανομής φόρτου είναι απαραίτητη, ώστε να επιτευχθεί καλή απόδοση.

Οι υλοποιήσεις του UTS χρησιμοποιούν μια στρατηγική work stealing για δυναμική κατανομή φόρτου σε πολλά παράλληλα μοντέλα προγραμματισμού, με την υλοποίηση σε OpenMP να περιλαμβάνει οδηγίες parallel. Τα νήματα διασχίζουν το τμήμα του δέντρου που βρίσκεται κατά βάθος στην τοπική στοίβα, ενώ η κλοπή μεταξύ των νημάτων προσθέτει μια -κατά πλάτος- διάσταση στην αναζήτηση.

Το γενικό σχήμα του δέντρου καθορίζεται από την παράμετρο tree type. Κάθε μία από αυτές δημιουργεί τα παιδιά των κόμβων με βάση δύο διαφορετικές κατανομές πιθανοτήτων, γεωμετρική ή διωνυμική, οι οποίες επηρεάζουν το μέγεθος και την ανισορροπία του δέντρου.

Η υλοποίηση αφορά την διάσχιση κατά βάθος ενός έμμεσου δέντρου και εφόσον δεν υπάρχει η ανάγκη να συγκρατείται η περιγραφή των παιδιών ενός κόμβου αφού αυτά έχουν δημιουργηθεί, μία στοίβα depth-first μπορεί να χρησιμοποιηθεί. Η εξερεύνηση ενός κόμβου γίνεται με την εξαγωγή του από τη στοίβα και την προσθήκη των παιδιών του σε αυτή. Η διάσχιση μπορεί να γίνει παράλληλα με το να μετακινηθεί ένας η περισσότεροι κόμβοι μιας στοίβας που βρίσκεται σε έναν επεξεργαστή στην άδεια στοίβα κάποιου άλλου που ήταν αδρανής.

Διάφορες στρατηγικές έχουν προταθεί ώστε να εξισορροπηθεί δυναμικά ο φόρτος σε μια τέτοια παράλληλη διάσχιση. Από αυτές, οι στρατηγικές work stealing ρίχνουν το βάρος της εύρεσης και μετακίνησης των tasks σε αδρανείς επεξεργαστές στους ίδιους τους αδρανείς επεξεργαστές, ελαχιστοποιώντας το overhead σε αυτούς που ήδη είναι απασχολημένοι. Οι

στρατηγικές work sharing από την άλλη, επιβαρύνουν τους, ήδη απασχολημένους, επεξεργαστές με τη μετακίνηση έργου στους αδρανείς.

Το πλήθος των κόμβων που μετακινούνται μεταξύ των επεξεργαστών κάθε φορά (k) έχει επίπτωση στην επίδοση των work stealing και work sharing στρατηγικών. Όσο πιο μεγάλη είναι αυτή η παράμετρος, τόσο μικρότερο είναι το overhead και στις δύο. Αυτό υπονοεί την επιλογή μεγάλου k. Ωστόσο, η πιθανότητα μια κατά βάθος αναζήτηση σε ένα από τα δέντρα να περιέχει k κόμβους στη στοίβα σε μια δεδομένη χρονική στιγμή είναι ανάλογη του $1/k$, επομένως, είναι δύσκολο να βρεθεί μεγάλος όγκος έργου, ώστε να μετακινηθεί.

Αντί να προσπαθήσει να γίνει κλοπή των συνεχειών των διαδικασιών, το οποίο χρειάζεται συνεργασία από τον μεταφραστή, στην πράξη ένα νήμα κάνει κλοπή κόμβων από τη depth-first στοίβα ενός άλλου νήματος. Αρχικά, το πρώτο νήμα κρατά τη ρίζα του δέντρου. Όσο αρκετοί κόμβοι παράγονται από τη ρίζα και τους απογόνους της, τα υπόλοιπα νήματα κλέβουν κομμάτια (chunks) για να τα τοποθετήσουν στη στοίβα τους. Κάθε νήμα εκτελεί καθοδική διερεύνηση σε κάποιο μέρος του δέντρου χρησιμοποιώντας τη δική του στοίβα από κόμβους. Ένα νήμα του οποίου η στοίβα αδειάζει, προσπαθεί να κλέψει έναν ή παραπάνω κόμβους από μια non-empty στοίβα κάποιου άλλου. Κατά την ολοκλήρωση, ο συνολικός αριθμός των κόμβων που εισέρχονται σε κάθε νήμα μπορεί να συνδυαστεί με τους υπόλοιπους και να δώσει το συνολικό μέγεθος του δέντρου.

Το UTS αναφέρει λεπτομερή στατιστικά σχετικά με το load balance, την επίδοση και το δέντρο που δημιουργείται. Περιλαμβάνει δείγματα δέντρων για διάφορα συστήματα, με τα πιο πρακτικά να είναι πέντε δέντρα με διαφορετική κατανομή και συνδυασμούς αυτών που χρησιμοποιήθηκαν και στα πειράματα μας, του μεγέθους του 4.1 million nodes περίπου, με μεγάλη απόκλιση όμως στο βάθος του δέντρου.

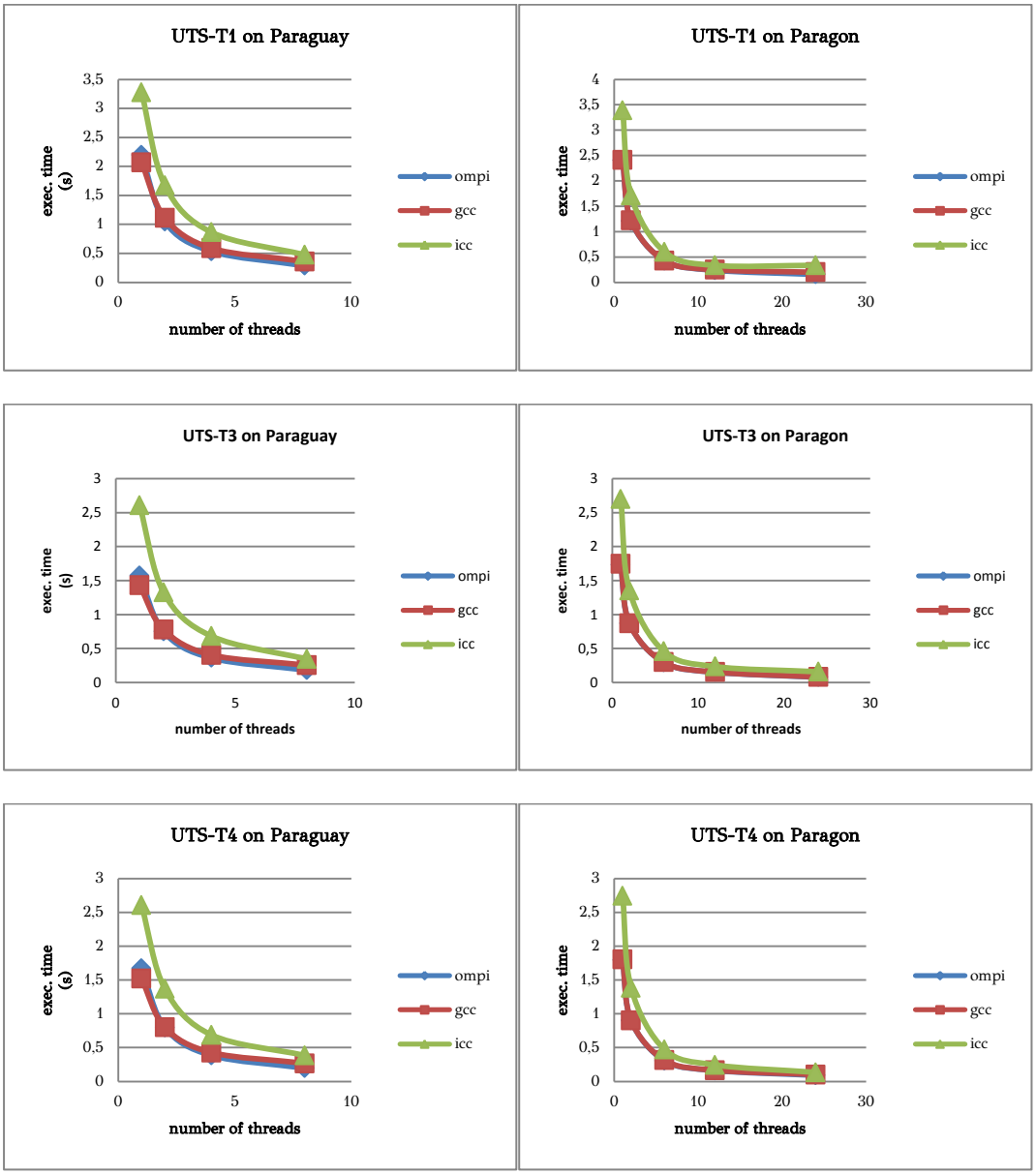
small workloads (~4 million nodes):

1. (T1) Geometric [fixed]: Tree size=4.130.071, tree depth=10 ,num leaves=3.305.118 (80.03%).
2. (T5) Geometric [linear dec.]: Tree size=4.147.582, tree depth=20, num leaves=2.181.218 (52.59%).
3. (T2) Geometric [cyclic]: Tree size=4.117.769, tree depth=81, num leaves=2.342.762 (56.89%).
4. (T3) Binomial: Tree size=4.112.897, tree depth=1.572, num leaves = 3.599.034 (87.51%).

5. (T4) Hybrid: Tree size = 4.132.453, tree depth =134, num leaves = 3.108.986 (75.23%).

Στα αποτελέσματα συμπεριλαμβάνονται οι εκδοχές με τους τύπους δέντρων T1, T3 και T4 για να αντιπροσωπεύσουν τα διαφορετικά ήδη δέντρων βασιζόμενα στη δημιουργία παιδιών με κατανομή γεωμετρική και διωνυμική. Οι δοκιμές έγιναν με default τιμές στις παραμέτρους, με τη πιο σημαντική να είναι το πλήθος κόμβων που μετακινούνται κάθε φορά μεταξύ των επεξεργαστών και η οποία καθορίζει τα ποσοστά ανισορροπίας και καθυστερήσεων (overheads). Αυτή έχει οριστεί από τους συγγραφείς σε τιμή η οποία δίνει φυσιολογικά αποτελέσματα σε διάφορα συστήματα. Τα γραφήματα δείχνουν τη μεταβολή του χρόνου εκτέλεσης σε seconds (άξονας y) σε σχέση με το πλήθος νημάτων (άξονας χ) με τις κουκίδες να αφορούν τα αντίστοιχα πειράματα. Η συμπεριφορά των μεταφραστών δεν εξαρτάται από την κατανομή με βάση την οποία δημιουργούνται τα παιδιά κάθε κόμβου, όπως φαίνεται στο Σχήμα 8.6. Τα benchmarks δείχνουν καλή συμπεριφορά για πλήθος νημάτων μέχρι 4, ενώ από εκεί και πέρα η ανισσοροπία του φόρτου και πιθανώς τα overheads ρίχνουν την απόδοση, ιδιαίτερα στην NUMA αρχιτεκτονική. Οι επιδόσεις του OMPi και του GCC είναι πολύ καλές, με τον Intel να συμπεριφέρεται παρόμοια, αλλά οι επιδόσεις του να είναι χειρότερες, ιδιαίτερα στην Paraguay.

Σχήμα 8.5: Αποτελέσματα πειραμάτων - UTS



8.3 Rodinia

Το Rodinia [15] είναι μια σουίτα εφαρμογών benchmarks για ετερογενείς υπολογισμούς. Έχει αναπτυχθεί με σκοπό να βοηθήσει τους αρχιτέκτονες να μελετήσουν αναδυόμενες πλατφόρμες όπως οι GPUs. Αποτελείται από εφαρμογές και πυρήνες εφαρμογών που στοχεύουν σε πολυπύρηνες CPUs και GPUs. Τα benchmarks αυτά καλύπτουν μεγάλο εύρος μοτίβων

παράλληλων επικοινωνιών, τεχνικών συγχρονισμού και κατανάλωσης ισχύος, και έχουν ήδη οδηγήσει σε συμπεράσματα όπως η αυξανόμενη σημασία των περιορισμών του bandwidth της μνήμης και επομένως της οργάνωσης των δεδομένων (data layout).

Οι εφαρμογές που περιλαμβάνονται στο Rodinia έχουν υλοποίηση για GPUs με τα μοντέλα CUDA και OpenCL και για πολυπύρηνες CPUs χρησιμοποιώντας OpenMP. Σκοπός ήταν να γίνει μια σύγκριση μέσω του Rodinia μεταξύ CPUs και GPUs, επιδεικνύοντας τις βασικές αρχιτεκτονικές διαφορές. Στην εργασία αυτή, δε μας αφορά η σύγκριση αυτών των αρχιτεκτονικών, ούτε των μοντέλων στα οποία έχουν αναπτυχθεί οι εφαρμογές. Έτσι, επικεντρωνόμαστε στις υλοποιήσεις με OpenMP, οι οποίες δε χρησιμοποιούν τις GPUs, καθώς υλοποιήσεις της έκδοσης 4.0 του μοντέλου δεν ήταν διαδεδομένες.

Το Rodinia προσπαθεί να καλύψει τα βασικά χαρακτηριστικά που πρέπει να πληροί μια σουίτα benchmark όπως είναι η ποικιλία στις εφαρμογές, οι "state of the art" αλγόριθμοι και είσοδοι για έλεγχο διαφορετικών καταστάσεων.

Η σουίτα, αρχικά, αποτελούνταν από τέσσερις εφαρμογές και πέντε πυρήνες εφαρμογών, ενώ έπειτα έχουν προστεθεί και άλλες εφαρμογές. Χρησιμοποιούνται διάφορες τεχνικές βελτιστοποίησης και εκμεταλλεύονται διάφοροι υπολογιστικοί πόροι στο chip. Τα προγράμματα εκθέτουν διάφορους τύπους παραλληλισμού, μοτίβα πρόσβασης δεδομένων και χαρακτηριστικά κοινοχρησίας των δεδομένων.

Ακολούθως, περιγράφονται κάποια workloads από το Rodinia benchmarks suite.

Leukocyte Tracking (LC): Εντοπίζει και καταγράφει κυλιόμενα λευκά κύτταρα στο αίμα σε μικροσκοπία βίντεο στα αιμοφόρα αγγεία. Τα κύτταρα εντοπίζονται στο πρώτο frame και έπειτα καταγράφονται στα επόμενα. Οι κύριες διεργασίες περιλαμβάνουν υπολογισμούς για κάθε pixel του ανώτατου Gradient Inverse Coefficient of Variation (GIGOV) σκορ μέσα από ένα εύρος πιθανών ελλείψεων, και υπολογίζει στην περιοχή που περικλείει το κύτταρο ένα Motion Gradient Vector Flow (MGVF) μητρώο.

Speckle Reducing Anisotropic Diffusion (SRAD): Είναι ένας αλγόριθμος διάχυσης (diffusion), βασισμένος σε μερικές διαφορικές εξισώσεις και χρησιμοποιείται για την αφαίρεση των κηλίδων σε μια εικόνα χωρίς να θυσιάζονται σημαντικά χαρακτηριστικά της. Το SRAD χρησιμοποιείται ευρέως σε εφαρμογές απεικόνισης υπερήχων. Οι είσοδοι στο πρόγραμμα

είναι εικόνες υπερήχων και η τιμή κάθε σημείου εξαρτάται από τους τέσσερις γείτονες του.

HotSpot (HS): Είναι ένα εργαλείο θερμικής προσομοίωσης που χρησιμοποιείται για να εκτιμηθεί η θερμοκρασία του επεξεργαστή, με βάση μια αρχιτεκτονική κάτοψη και προσομοιώνοντας μετρικές ισχύος. Η έκδοση αυτή περιλαμβάνει τον 2D πυρήνα προσωρινής θερμικής προσομοίωσης του HotSpot, ο οποίος επαναληπτικά επιλύει μια σειρά από διαφορικές εξισώσεις. Οι είσοδοι στο πρόγραμμα είναι η ισχύς και οι αρχικές θερμοκρασίες.

Back Propagation (BP): Είναι ένας αλγόριθμος μηχανικής μάθησης, ο οποίος μαθαίνει τα βάρη των ακμών συνδεδεμένων κόμβων σε ένα νευρωνικό δίκτυο αποτελούμενο από στρώματα. Η εφαρμογή αποτελείται από δύο φάσεις, την εμπρόσθια φάση στην οποία τα σήματα προωθούνται από την είσοδο στο στρώμα εξόδου, και την οπίσθια φάση κατά την οποία το σφάλμα μεταξύ της παρατηρούμενης και της επιθυμητής τιμής στην έξοδο διαδίδεται προς τα πίσω, ώστε να προσαρμοστούν οι τιμές των βαρών και της πόλωσης.

Needleman-Wunsch (NW): Είναι μια καθολική μέθοδος βελτιστοποίησης για ευθυγράμμιση DNA ακολουθιών. Τα πιθανά ζευγάρια ακολουθιών οργανώνονται σε ένα 2-D μητρώο. Ο αλγόριθμος γεμίζει τον πίνακα με σκορ, τα οποία αντιπροσωπεύουν την τιμή του μέγιστου σε βάρος μονοπατιού που καταλήγει σε αυτό το κύτταρο. Μια αναδρομική διεργασία χρησιμοποιείται για την αναζήτηση της βέλτιστης ευθυγράμμισης. Ο παράλληλος αλγόριθμος επεξεργάζεται τον πίνακα με τα σκορ σε διαγώνιες λωρίδες, από πάνω αριστερά μέχρι κάτω δεξιά.

K-means (KM): Είναι αλγόριθμος ομαδοποίησης που χρησιμοποιείται ευρέος στην εξόρυξη δεδομένων και η υλοποίηση του με OpenMP είναι βασισμένη στην υλοποίηση που περιγράφεται στο αντίστοιχο κεφάλαιο των MineBench.

Stream Cluster (SC): Είναι υλοποίηση με OpenMP και CUDA του αντίστοιχου benchmark του Parsec Benchmark Suite που περιγράφεται στο αντίστοιχο κεφάλαιο.

Breadth-First Search (BFS): Διασχίζει όλα τα συνδεδεμένα στοιχεία ενός γράφου. Μεγάλα γραφήματα που περιέχουν χιλιάδες κορυφές είναι σύνηθες φαινόμενο σε επιστημονικές και μηχανικές εφαρμογές.

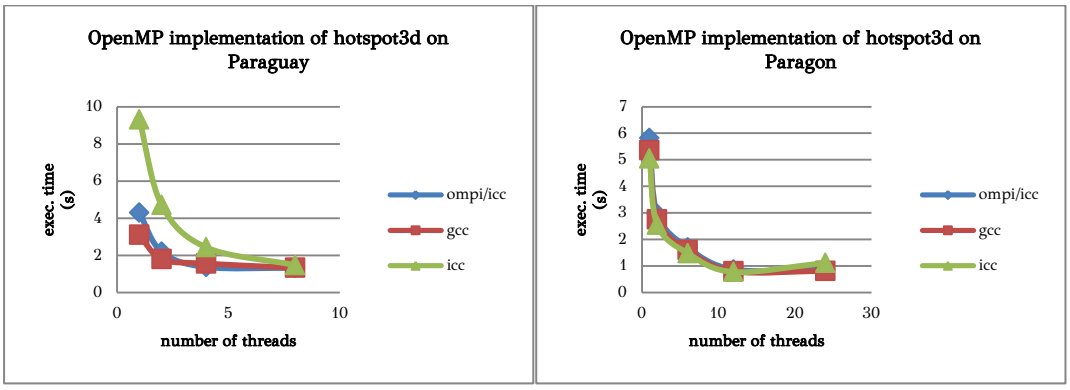
Similarity Score (SS): Χρησιμοποιείται σε ομαδοποίηση εγγράφων από το web για τον υπολογισμό της ομοιότητας ανά ζευγάρι εγγράφων. Ο πηγαίος κώδικας προέρχεται από το Mars project του Hong Kong University of Science and Technology. Το Mars κρύβει την πολυπλοκότητα του προγραμματισμού της GPU πίσω από την απλότητα και την οικειότητα που προσφέρει το MapReduce interface.

Η τρέχουσα έκδοση είναι η 3.1 και έχουν προστεθεί επιπλέον workloads και υλοποιήσεις σε μοντέλα που δεν υπήρχαν. Μερικά από τα καινούρια workloads είναι τα B+tree, cfd, heartwall, hotspot3D, lavaMD, myocyte κ.α.

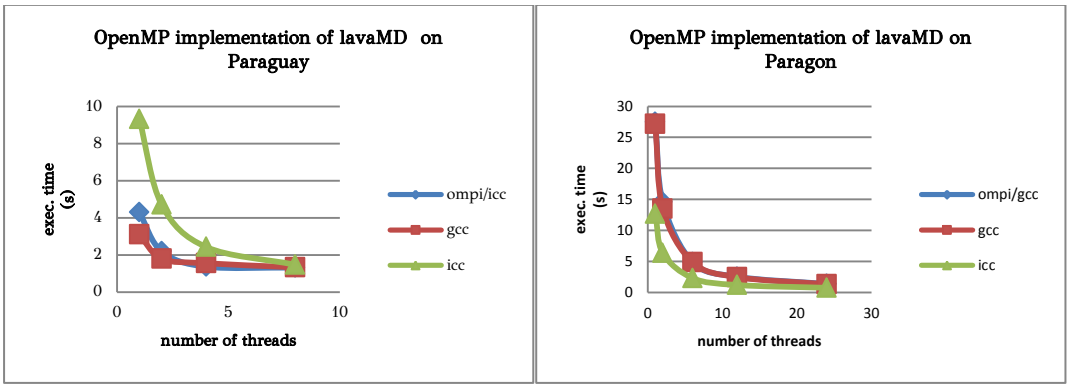
Οι διαφορετικές υλοποιήσεις των εφαρμογών στο Rodinia, είναι πολύ χρήσιμες για τη σύγκριση διαφορετικών αρχιτεκτονικών και μοντέλων παραλληλοποίησης. Επίσης, από τη στιγμή που το OpenMP με τις πρόσφατες εκδόσεις υποστηρίζει και αποστολή κώδικα σε συσκευές όπως επιταχυντές και κάρτες γραφικών, τα προγράμματα αυτά ίσως φανούν χρήσιμα κατά την ανάπτυξη των αντίστοιχων υλοποιήσεων, καθώς και τη συγκριτική μελέτη αυτών.

Για τις ανάγκες των δικών μας μετρήσεων χρησιμοποιήθηκαν υλοποιήσεις με OpenMP σε γλώσσα C. Οι εφαρμογές που δοκιμάστηκαν είναι οι hotspot3d και lavaMD, οι οποίες αποτελούν μέρος των πιο πρόσφατων εκδόσεων. Στα γραφήματα φαίνεται η συμπεριφορά του χρόνου εκτέλεσης των εφαρμογών σε seconds (άξονας y) σε σχέση με το πλήθος νημάτων (άξονας χ) με τις κουκίδες να αφορούν τα αντίστοιχα πειράματα. Στο Σχήμα 8.7 φαίνεται ότι ο χρόνος εκτέλεσης του hotspot3d δεν κλιμακώνεται καλά όσο αυξάνουν τα νήματα και στις δύο αρχιτεκτονικές. Οι επιδόσεις των GCC και OMPi είναι παρόμοιες, ενώ για τον Intel η UMA αρχιτεκτονική δεν είναι η κατάλληλη επιλογή. Παρόμοια, για την εφαρμογή lavaMD. Και εδώ ο χρόνος εκτέλεσης δεν κλιμακώνεται καλά με την αύξηση των νημάτων, όπως φαίνεται στο Σχήμα 8.8. Ο Intel και εδώ φαίνεται να μην πηγαίνει καλά στην Paraguay, ενώ αντιθέτως στον Paragon παρατηρούνται καλύτερες επιδόσεις, σε σχέση με τους GCC και OMPi, οι οποίοι συμπεριφέρονται παρόμοια.

Σχήμα 8.6: Αποτελέσματα πειραμάτων – hotspot3d



Σχήμα 8.7: Αποτελέσματα πειραμάτων – lavaMD



8.4 MineBench

Το MineBench [16] είναι μια σουίτα εφαρμογών, η οποία περιέχει 15 εφαρμογές που αντιπροσωπεύουν την κλάση επιστημονικών και εμπορικών εφαρμογών της εξόρυξης δεδομένων. Αυτές, πιο συγκεκριμένα, ανήκουν σε διάφορες κατηγορίες όπως ομαδοποίηση, κατηγοριοποίηση και εξόρυξη κανόνων συσχέτισης. Οι κώδικες των εφαρμογών έχουν παραλληλοποιηθεί με το μοντέλο OpenMP.

Η εξόρυξη δεδομένων είναι μια τεχνική για την κατανόηση και τη μετατροπή ακατέργαστων δεδομένων σε χρήσιμη πληροφορία και είναι αυξανόμενη η χρήση της σε πεδία όπως το marketing, επιστημονικές ανακαλύψεις, βιοτεχνολογία, αναζητήσεις στον ιστό κ.α. Η διαδικασία της εξόρυξης πληροφορίας σε ακατέργαστα δεδομένα είναι αυτοματοποιημένη και τις περισσότερες φορές με τρόπο πρόβλεψης. Αυτή είναι και η διαφορά με τις τεχνικές βάσεων δεδομένων.

Τα δεδομένα πλέον έχουν γίνει αχανή, και για το λόγο αυτό, οι αλγόριθμοι ολοένα και γίνονται πιο περίπλοκοι για να τα διαχειριστούν. Ωστόσο, περιορισμοί στη γενική απόδοση των συστημάτων έχει ως αποτέλεσμα απαγορευτικούς χρόνους εκτέλεσης για τις εφαρμογές. Έτσι, προκύπτει η ανάγκη σχεδίασης μοντέρνων αρχιτεκτονικών για τη βελτιστοποίηση της απόδοσης τέτοιων εφαρμογών. Αυτή η διαδικασία περιλαμβάνει δύο βήματα: αρχικά την κατανόηση των χαρακτηριστικών απόδοσης διαφορετικών αντιπροσωπευτικών εφαρμογών και δεύτερων υψηλής απόδοσης αρχιτεκτονικές βελτιστοποιημένες για συστήματα εξόρυξης δεδομένων χρειάζεται να αναπτυχθούν και οι επιδόσεις σε συχνά χρησιμοποιούμενα workloads να αποτιμηθεί. Για τους λόγους αυτούς, αναπτύχθηκε το MineBench.

Οι εφαρμογές που έχουν επιλεγεί, αντιπροσωπεύουν την ετερογένεια των αλγορίθμων και μεθόδων που χρησιμοποιούνται στην εξόρυξη δεδομένων. Οι κώδικες είναι πλήρως ολοκληρωμένες υλοποιήσεις εφαρμογών και δεν περιέχουν stand-alone πυρήνες και οι δώδεκα από τις δεκαπέντε εφαρμογές είναι παραλληλοποιημένες με OpenMP. Όσον αφορά τα input sets, τα οποία αποτελούν σημαντική πτυχή των εφαρμογών, υπάρχουν τρεις κατηγορίες, *small*, *medium* και *large*, κλιμακωμένου μεγέθους

Οι εφαρμογές έχουν επιλεγεί για να καλύπτουν τους τομείς της εξόρυξης κανόνων συσχέτισης, κατηγοριοποίησης, ομαδοποίησης, οπτικοποίησης δεδομένων, εξόρυξης ακολουθιών, εύρεσης όμοιων, βελτιστοποίησης, text mining και περιγράφονται παρακάτω.

ScalParc: Είναι μία αποτελεσματική και κλιμακώσιμη παραλλαγή των δέντρων απόφασης. Το μοντέλο αυτό κατασκευάζεται με επαναληπτικό διαχωρισμό των δεδομένων εκπαίδευσης με βάση ένα κριτήριο βελτιστοποίησης, μέχρι όλες οι εγγραφές που ανήκουν σε κάθε διαχωρισμό να φέρουν την ίδια ετικέτα κλάσης. Το πρόγραμμα χρησιμοποιεί ένα πρότυπο παράλληλου κατακερματισμού για να επιτύχει κλιμακωσιμότητα κατά τη φάση του διαχωρισμού. Αυτή η προσέγγιση το κάνει κλιμακώσιμο ως προς το χρόνο εκτέλεσης, αλλά και ως προς τις απαιτήσεις σε μνήμη.

Naïve Bayesian classifier: Είναι ένας απλός κατηγοριοποιητής που χρησιμοποιεί ένα σύνολο δεδομένων εκπαίδευσης για να κατασκευάσει ένα μοντέλο πρόβλεψης που περιέχει κλάσεις των εγγραφών, έτσι ώστε το μοντέλο να μπορεί να χρησιμοποιηθεί εκ νέου για να προβλέψει την κλάση μιας νέας εγγραφής.

Το **SNP**, από τα αρχικά single nucleotide polymorphisms (DNA ακολουθιακές διακυμάνσεις), χρησιμοποιεί τη μέθοδο αναζήτησης hill climbing, η οποία διαλέγει ένα αρχικό σημείο εκκίνησης και αναζητεί τους κοντινότερους γείτονες αυτού. Ο γείτονας με το υψηλότερο σκορ γίνεται έπειτα το νέο σημείο εκκίνησης. Αυτή μέθοδος επαναλαμβάνεται μέχρι να βρεθεί ένα τοπικό μέγιστο.

Το **GeneNet** χρησιμοποιεί έναν παρόμοιο αλγόριθμο με το **SNP** με τη διαφορά ότι τα δεδομένα εισόδου είναι πιο περίπλοκα, απαιτώντας επιπλέον υπολογισμούς κατά τη διαδικασία της εκπαίδευσης. Το πρόγραμμα έχει παραλληλοποιηθεί χρησιμοποιώντας ένα επιπέδου κόμβου πρότυπο, κατά το οποίο σε κάθε βήμα, οι κόμβοι του BN διανέμονται σε διαφορετικούς επεξεργαστές.

Το **SEMPHY** είναι ένας αλγόριθμος structure learning που βασίζεται στα φυλογενετικά δέντρα. Τα φυλογενετικά δέντρα αναπαριστούν τις γενετικές σχέσεις των ειδών, με τα κοντινά είδη να βρίσκονται σε γειτονικά κλαδιά. Το πρόγραμμα χρησιμοποιεί τον αλγόριθμο structural expectation maximization, για να αναζητήσει αποτελεσματικά για μέγιστης πιθανότητας φυλογενετικά δέντρα. Οι υπολογισμοί κλιμακώνονται τετραγωνικά με το μέγεθος της εισόδου, κάτι που απαιτεί παραλληλοποίηση. Οι απαιτητικοί σε υπολογισμούς πυρήνες του αλγορίθμου είναι αυτοί που παραλληλοποιήθηκαν με OpenMP.

Rsearch: Χρησιμοποιεί μια προσέγγιση βασισμένη στη γραμματική για να αναζητήσει στη βάση δεδομένων γονιδίων ομόλογες ακολουθίες RNA. Ο μηχανισμός που χρησιμοποιείται για την παραλληλοποίηση του είναι μια δυναμική εξισορρόπηση φόρτου.

Η **SVM-RFE** ή αλλιώς Support Vector Machines-Recursive Feature Elimination είναι μια μέθοδος επιλογής χαρακτηριστικών. Η επιλογή γίνεται με μια επαναληπτική διαδικασία, κατά την οποία αποκλείονται χαρακτηριστικά. Ο πολλαπλασιασμός διανυσμάτων είναι η ακριβή διαδικασία και ο πυρήνας που την εκτελεί έχει παραλληλοποιηθεί με OpenMP.

K-Means: Είναι ένας αλγόριθμος ομαδοποίησης. Δοθείσας της παραμέτρου k από το χρήστη, οι αρχικές k ομάδες δημιουργούνται από τη βάση. Έπειτα, κάθε αντικείμενο ανατίθεται στην ομάδα, της οποίας το κέντρο είναι πιο κοντά σε αυτό με βάση μια συνάρτηση ομοιότητας. Όταν οι καινούργιες αναθέσεις ολοκληρωθούν, τα νέα κέντρα βρίσκονται μέσω του

μέσου όρου των σημείων που περιέχονται σε αυτό. Αυτή η διαδικασία επαναλαμβάνεται, μέχρι κάποια κριτήρια σύγκλισης ικανοποιούνται.

Οι ομάδες που επιστρέφει ο K-means ορισμένες φορές αναφέρονται ως "hard clusters", εξαιτίας του γεγονότος ότι κάποιο αντικείμενο ανήκει ή όχι σε μια ομάδα. Ο **Fuzzy K-means** αλγόριθμος χαλαρώνει αυτή την ιδιότητα με την υπόθεση ότι ένα αντικείμενο μπορεί να έχει ένα βαθμό συμμετοχής σε κάθε ομάδα. Συγκριτικά με τη συνάρτηση ομοιότητας στον απλό K-means, ο υπολογισμός για το βαθμό συμμετοχής είναι πολύ πιο κοστοβόρος. Ωστόσο, η ευελιξία της ανάθεσης αντικειμένων σε διάφορες ομάδες μπορεί να είναι απαραίτητη για καλύτερα αποτελέσματα. Και τα δύο προγράμματα έχουν παραλληλοποιηθεί με τη διανομή των αντικειμένων εισόδου στους επεξεργαστές. Στο τέλος κάθε επανάληψης, επιπλέον επικοινωνίες είναι απαραίτητες για το συγχρονισμό της διαδικασίας ομαδοποίησης.

Το **HOP**, αρχικά προτάθηκε στην αστροφυσική, και είναι μια τυπική μέθοδος ομαδοποίησης που βασίζεται στην πυκνότητα. Μετά την ανάθεση μιας εκτίμησης για την πυκνότητα κάθε σωματιδίου, το πρόγραμμα συνδέει κάθε σωματίδιο με τον πιο πυκνό γείτονα του. Η διαδικασία αυτή επαναλαμβάνεται, μέχρι ο πιο πυκνός γείτονας του σωματιδίου να είναι το ίδιο το σωματίδιο. Αυτά που οδηγούνται στο ίδιο σωματίδιο ομαδοποιούνται μαζί. Το HOP έχει παραλληλοποιηθεί χρησιμοποιώντας ένα k-D δέντρο τριών διαστάσεων, το οποίο επιτρέπει κάθε νήμα να επεξεργάζεται μόνο ένα υποσύνολο των σωματιδίων, και επομένως, να μειώνει τα κόστη επικοινωνίας σημαντικά.

BIRCH: Είναι ένας αυξητικός και ιεραρχικός αλγόριθμος ομαδοποίησης για μεγάλες βάσεις δεδομένων. Χρησιμοποιεί ένα ιεραρχικό δέντρο για να αναπαραστήσει το πόσο κοντινά είναι τα αντικείμενα μεταξύ τους. Το πρόγραμμα σκανάρει τη βάση δεδομένων για να κατασκευάσει ένα clustering-feature (CF) δέντρο για να συνοψίσει την αναπαράσταση των ομάδων. Για μεγάλες βάσεις δεδομένων, το πρόγραμμα μπορεί να επιτύχει καλή απόδοση και κλιμακωσιμότητα, ενώ είναι αποτελεσματικό για incremental ομαδοποίηση νεοεισερχόμενων αντικειμένων ή για ροές δεδομένων που πρέπει να ομαδοποιηθούν.

Apriori: Είναι ένας ARM (Association Rule Mining) αλγόριθμος που χρησιμοποιεί κλάδεμα με βάση το support για να ελέγξει συστηματικά την εκθετική αύξηση του χώρου αναζήτησης. Χρησιμοποιείται για την εύρεση συχνών στοιχειοσυνόλων με μια στρατηγική generate-and-test. Βασίζεται στην ιδιότητα ότι όλα τα μη κενά υποσύνολα ενός συχνού στοιχειοσυνόλου

πρέπει να είναι συχνά (Apriori property). Για να αποφασιστούν γρήγορα τα συχνά αντικείμενα, ο αλγόριθμος χρησιμοποιεί ένα δέντρο κατακερματισμού για να αποθηκεύσει τους υποψήφιους. Ο αλγόριθμος έχει παραλληλοποιειθεί με διανομή τμημάτων των δεδομένων εισόδου στους επεξεργαστές. Στη συνέχεια, το master νήμα συλλέγει τα τοπικά συχνά στοιχειοσύνολα και δημιουργεί τα καθολικά.

Utility mining: Είναι άλλη μια ARM τεχνική εξόρυξης, όπου η υπόθεση της ομοιομορφίας μεταξύ των αντικειμένων δεν υπάρχει. Ψηλότερα "utility" σύνολα εντοπίζονται από μια βάση δεδομένων λαμβάνοντας υπόψη διαφορετικές τιμές για ξεχωριστά αντικείμενα. Ο σκοπός του utility mining είναι να περιορίσει το μέγεθος του σετ των υποψηφίων για να μειώσει τους υπολογισμούς για την εύρεση της τιμής των αντικειμένων. Το πρότυπο παραλληλοποίησης είναι το ίδιο με αυτό του Apriori.

Eclat: Μπορεί να προσδιορίσει το support ενός k-στοιχειοσυνόλου, μέσω της τομής της λίστας id των πρώτων δύο $(k-1)$-μήκους υποσυνόλων που έχουν κοινό πρόθεμα. Σπάει το χώρο αναζήτησης σε μικρά, ανεξάρτητα και διαχειρίσιμα κομμάτια. Αποτελεσματικές τεχνικές διάσχισης πλέγματος χρησιμοποιούνται για τον εντοπισμό των πραγματικά συχνών στοιχειοσυνόλων.

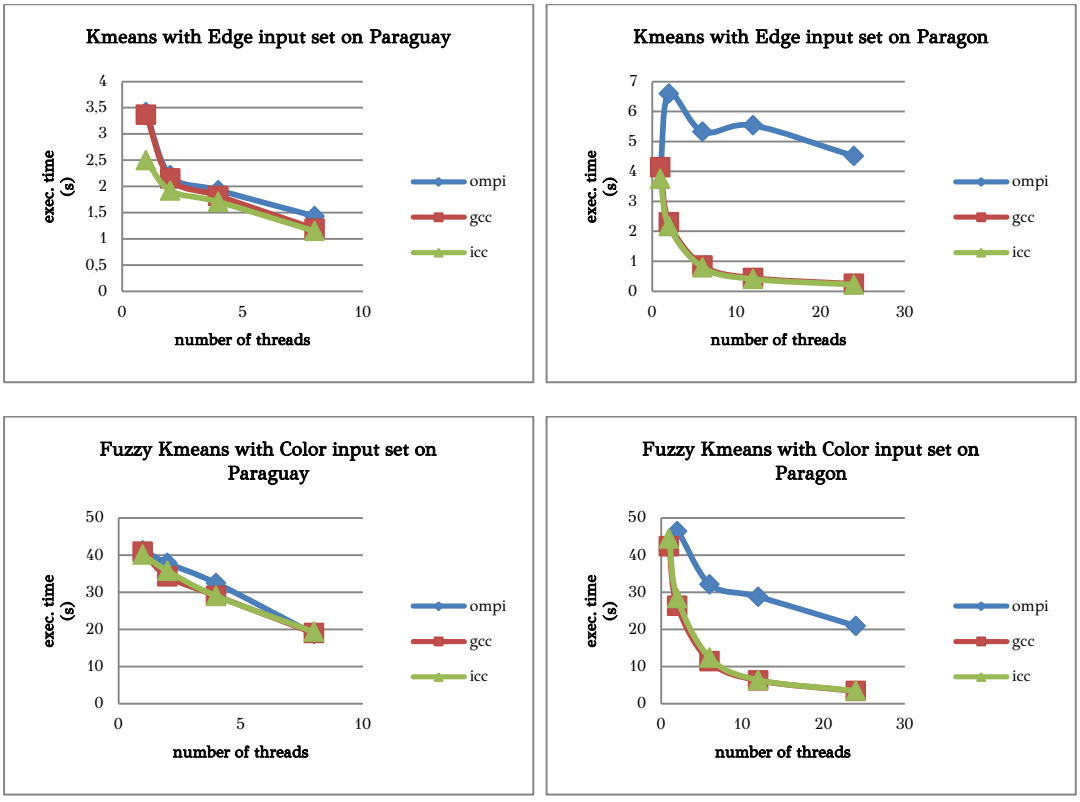
PLSA: Χρησιμοποιεί μια προσέγγιση δυναμικού προγραμματισμού για να λύσει το πρόβλημα της ευθυγράμμισης ακολουθιών. Βασίζεται στον αλγόριθμο που προτάθηκε από τους Smith και Waterman, ο οποίος χρησιμοποιεί τη τοπική ευθυγράμμιση για να βρει τη μακρύτερη κοινή υποακολουθία. Το πρόγραμμα χρησιμοποιεί ειδικές δομές δεδομένων για να τεμαχίσει το όλο πρόβλημα σε ανεξάρτητα υποπροβλήματα, κάτι το οποίο μειώνει δραματικά τους απαραίτητους υπολογισμούς και, επομένως, παρέχει περισσότερο παραλληλισμό από τις υπόλοιπες μεθόδους ευθυγράμμισης ακολουθιών.

Όσον αφορά τα δεδομένα εισόδου, αυτά παίζουν πολύ σημαντικό ρόλο στις εφαρμογές εξόρυξης δεδομένων. Στα benchmarks χρησιμοποιούνται είτε δεδομένα του πραγματικού κόσμου, είτε συνθετικά δεδομένα που δημιουργήθηκαν από διάφορα εργαλεία αυτού του σκοπού που χρησιμοποιούνται σε επιστημονικές και στατιστικές προσομοιώσεις.

Στα αποτελέσματα συμπεριλαμβάνονται μετρήσεις πάνω στον αλγόριθμο Kmeans, ο οποίος είναι γραμμένος σε C, και ήταν η μοναδική εφαρμογή γραμμένη σε C που δεν παρουσίασε προβλήματα κατά τη μετάφραση, με εισόδους color και edge. Τα γραφήματα δείχνουν τη συμπεριφορά του

χρόνου εκτέλεσης σε seconds (άξονας y) σε σχέση με το πλήθος νημάτων (άξονας χ) και οι κουκίδες αφορούν τα αντίστοιχα πειράματα. Στο Σχήμα 8.9 φαίνονται οι δύο διαφορετικές εκδοχές του, ενώ άξια σχολιασμού είναι η συμπεριφορά του Ompi στον Paragon, η οποία είναι πολύ διαφορετική από ότι στην Paraguay. Γενικά, η κλιμάκωση του χρόνου εκτέλεσης δεν είναι πολύ καλή όσο αυξάνουν τα νήματα, κάτι που πιθανώς έγκειται στο γεγονός ότι απαιτούνται πολλές επικοινωνίες για την εύρεση του νέου κέντρου μετά από κάθε επανάληψη, με τον OMPi να μην ανταποκρίνεται θετικά σε αυτές, σε αρχιτεκτονικές NUMA.

Σχήμα 8.8 : Αποτελέσματα πειραμάτων - Kmeans



ΚΕΦΑΛΑΙΟ 9
ΣΥΜΠΕΡΑΣΜΑΤΑ

Στη διπλωματική εργασία αυτή έγινε μία πλήρης έρευνα και συλλογή όλων των μετροπρογραμμάτων που σχετίζονται με το μοντέλο OpenMP. Τα μετροπρογράμματα που συλλέχθηκαν αρχικά χαρακτηρίστηκαν και κατηγοριοποιήθηκαν σε τρείς κατηγορίες:

- Προγράμματα ελέγχου ορθότητας.
- Microbenchmarks.
- Σουίτες εφαρμογών μέτρησης επιδόσεων, με μετροπρογράμματα γενικού και ειδικού σκοπού.

Το επόμενο στάδιο της εργασίας περιελάμβανε την εκτέλεση των μετροπρόγραμμάτων αυτών σε δύο αντιπροσωπευτικές κατηγορίες αρχιτεκτονικών παράλληλων συστημάτων κοινόχρηστης μνήμης - UMA και NUMA - και τη συγκριτική μελέτη τυπικών μεταφραστών του OpenMP. Οι μεταφραστές που επιλέχθηκαν ήταν οι GCC και Intel, καθώς υλοποιούν πλήρως τις προδιαγραφές του OpenMP, είναι ελεύθερα διαθέσιμοι και ανοιχτού κώδικα και υποστηρίζουν τα συνήθη συστήματα και αρχιτεκτονικές. Επιπλέον μελετάται συγκριτικά και ο δικός μας μεταφραστής OMPi.

Τα μετροπρογράμματα που δοκιμάστηκαν, μπορούν να προκαλέσουν συζητήσεις σχετικά με την επιλογή της κατάλληλης αρχιτεκτονικής για την επίλυση προβλημάτων στα παράλληλα συστήματα, ανάλογα με τη φύση της εφαρμογής. Μπορούν να αναδείξουν επίσης θετικές και αρνητικές ιδιαιτερότητες των μεταφραστών και να υποδείξουν στον προγραμματιστή ποιοι είναι κατάλληλοι να ανταποκριθούν στις απαιτήσεις της εφαρμογής τους. Γενικά οι επιδόσεις του OMPi είναι πολύ κοντά σε αυτές των GCC και Intel και ορισμένες φορές πολύ καλύτερες, κάτι που αναδεικνύει την ωριμότητα της υλοποίησης του, ασχέτως του γεγονότος ότι είναι ένας ερευνητικός μεταφραστής. Οι επιδόσεις του GCC σε μετροπρογράμματα που αφορούν tasks είναι ένα θέμα. Όσον αφορά τα μετροπρογράμματα τα ίδια, το εύρος των συνθηκών που προσπαθούν να αναλύσουν είναι αρκετά αντιπροσωπευτικό του συνόλου των προδιαγραφών του OpenMP και των τεχνικών παραλληλοποίησης που υπάρχουν στα παράλληλα συστήματα κοινόχρηστης μνήμης. Οι μόνες προδιαγραφές του OpenMP για τις οποίες δεν υπάρχουν μετροπρογράμματα μέτρησης επιδόσεων είναι αυτές που συγκαταλέγονται στις πρόσφατες εκδόσεις και αφορούν τα devices.

Επομένως, καλό θα ήταν να είχαν αναπτυχθεί τέτοια μετροπρογράμματα ή να τροποποιούνταν ήδη υπάρχοντα, όπως τα Rodinia, για να συμπεριλάβουν τις πρόσφατες προδιαγραφές του μοντέλου και να μελετηθούν οι υλοποιήσεις μεταφραστών που στοχεύουν σε επιταχυντές.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] V. V. Dimakopoulos, E. Leontiadis, and G. Tzoumas, "A portable c compiler for openmp v.2.0," *In Proc. of the 5th European Workshop on OpenMP (EWOMP '03)*, 2003.

[2] Wang, Cheng & Chandrasekaran, Sunita & Chapman, Barbara, "An OpenMP 3.1 validation testsuite", 2012.

[3] Diaz J.M., Pophale S., Hernandez O., Bernholdt D.E., Chandrasekaran S. "OpenMP 4.5 Validation and Verification Suite for Device Offload." In: de Supinski B., Valero-Lara P., Martorell X., Mateo Bellido S., Labarta J. (eds) *Evolving OpenMP for Evolving Architectures. IWOMP 2018. Lecture Notes in Computer Science, vol 11128. Springer, Cham* , 2018

[4] OmniMP compiler Project, http://www.hpcs.cs.tsukuba.ac.jp/omni-compiler/

[5]Nas Parallel Benchmarks 3.0 Unofficial OpenMP C version, https://github.com/benchmark-subsetting/NPB3.0-omp-C

[6] J. M. Bull and D. O'Neill, "A microbenchmark suite for OpenMP 2.0", *SIGARCH Comput. Archit. News, vol. 29, no. 5, pp. 41–48*, 2001.

[7] Bailey D.H. (2011) "NAS Parallel Benchmarks." In: Padua D. (eds) *Encyclopedia of Parallel Computing. Springer, Boston, MA*, 2011

[8] Jin, H. & MA, Frumkin. , "The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance", 2000.

[9] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. "The PARSEC benchmark suite: characterization and architectural implications." *In Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT '08). ACM, New York, NY, USA, 72-81.*, 2008.

[10] Christian Bienia. "Benchmarking Modern Multiprocessors. Ph.D. Dissertation. Princeton University, Princeton, NJ, USA. Advisor(s) Kai Li. ,2011.

[11] Müller, Matthias & Baron, John & Brantley, William & Feng, Huiyu & Hackenberg, Daniel & Henschel, Robert & Jost, Gabriele & Molka, Daniel & Parrott, Chris & Robichaux, Joe & Shelepugin, Pavel & van Waveren, Matthijs & Whitney, Brian & Kumaran, Kalyan. "SPEC OMP2012 — An Application Benchmark Suite for Parallel Systems Using OpenMP", 2012.

[12] A. J. Dorta, C. Rodriguez and F. de Sande, "The OpenMP source code repository," *13th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Lugano, Switzerland*, 2005.

[13] A. Duran, X. Teruel, R. Ferrer, X. Martorell and E. Ayguade, "Barcelona OpenMP Tasks Suite: A Set of Benchmarks Targeting the Exploitation of Task Parallelism in OpenMP," *International Conference on Parallel Processing, Vienna*, 2009.

[14] Olivier S. et al. "UTS: An Unbalanced Tree Search Benchmark." In: Almási G., Caşcaval C., Wu P. (eds) *Languages and Compilers for Parallel Computing. LCPC 2006. Lecture Notes in Computer Science, vol 4382. Springer, Berlin, Heidelberg*, 2007.

[15] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," *IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, 2009, pp. 44-54*, 2009

[16] Narayanan, Ramanathan & Ozisikyilmaz, Berkin & Zambreno, Joseph & Memik, Gokhan & Choudhary, Alok. "MineBench: A Benchmark Suite for Data Mining Workloads". *IEEE Workload Characterization Symposium. 182-188*, 2006