ΣΤΡΑΤΙΩΤΙΚΗ ΣΧΟΛΗ ΕΥΕΛΠΙΔΩΝ
Τμήμα Στρατιωτικών Επιστημών

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΔΙΙΔΡΥΜΑΤΙΚΟ ΔΙΑΤΜΗΜΑΤΙΚΟ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΑΚΑΔΗΜΑΪΚΟΥ ΕΤΟΥΣ 2022-23

ΕΦΑΡΜΟΣΜΕΝΗ
ΕΠΙΧΕΙΡΗΣΙΑΚΗ ΕΡΕΥΝΑ & ΑΝΑΛΥΣΗ

(ΠΔ 97 /2015/ΦΕΚ 163Α'/20.08.2014)

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
Σχολή Μηχανικών Παραγωγής & Διοίκησης

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

ΣΥΓΚΡΙΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΤΗΣ ΦΟΤΗΤΟΤΗΤΑΣ ΤΗΣ ΑΠΟΔΟΣΗΣ ΣΕ ΕΤΕΡΟΓΕΝΕΙΣ ΥΠΟΛΟΓΙΣΤΙΚΕΣ ΠΛΑΤΦΟΡΜΕΣ

Διατριβή που υπεβλήθη για την μερική ικανοποίηση των απαιτήσεων για την απόκτηση Μεταπτυχιακού Διπλώματος Ειδίκευσης

Υπό:

ΣΤΑΥΡΟΥΛΑΣ ΚΑΜΠΥΛΗ

Α.Μ.: 2022018105

ΙΑΝΟΥΑΡΙΟΣ 2024

Η Μεταπτυχιακή Διατριβή της κ. Σταυρούλας Καμπύλη εγκρίνεται:

ΤΡΙΜΕΛΗΣ ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Καθηγητής ΝΙΚΟΛΑΟΣ ΔΑΡΑΣ (Επιβλέπων)

Καθηγητής ΝΙΚΟΛΑΟΣ ΜΑΤΣΑΤΣΙΝΗΣ

Nikolaos Matsatsinis
Digitally signed by Nikolaos Matsatsinis
Date: 2024.02.14 17:38:47 +02'00'

Καθηγητής ΝΙΚΟΛΑΟΣ ΠΑΠΑΔΑΚΗΣ

Νικόλαος Παπαδάκης

ΣΕΛΙΔΑ ΣΚΟΠΙΜΑ ΚΕΝΗ

Έτος 2024

ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστώ θερμά τους καθηγητές μου και ειδικότερα τον επιβλέπων της παρούσας διατριβής, καθηγητή κ. Νικόλαο Δάρα, ο οποίος με συμβούλευσε και συνέβαλε τα μέγιστα για την ομαλή ολοκλήρωση των σπουδών μου. Επίσης, ευχαριστώ τους συμφοιτητές μου, με τους οποίους συνεργάστηκα άριστα κατά την διάρκεια της φοίτησης μου.

ΣΕΛΙΔΑ ΣΚΟΠΙΜΑ ΚΕΝΗ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Κατάλογος Πινάκων

ΠΕΡΙΛΗΨΗ

Οι ετερογενείς υπολογιστικές πλατφόρμες είναι διαδεδομένες στα σύγχρονα ηλεκτρονικά συστήματα και καλύπτουν ένα μεγάλο πεδίο εφαρμογών από τις κινητές συσκευές έως τους υπερυπολογιστές. Η υποστήριξη κάθε συγκριτικής αξιολόγησης (benchmarking) της απόδοσης τέτοιων ετερογενών πλατφορμών απαιτεί τη δημιουργία ενός φορητού ετερογενούς κώδικα απόδοσης για την επίτευξη της σχετικής αξιολόγησης. Ένας φορητός κώδικας πρέπει να είναι ευέλικτος και να μπορεί να λειτουργήσει σε διάφορες αρχιτεκτονικές, όπως σε κεντρικές μονάδες επεξεργασίας (CPU), μονάδες επεξεργασίας γραφικών (GPU), πολλαπλά ολοκληρωμένα κυρίαρχης επεξεργασίας (MIC), προγραμματιζόμενα πυλώνες σε επίπεδο υλικού (FPGA) και προηγμένες μικροεπεξεργαστικές μονάδες μειωμένου συνόλου εντολών (ARMs).

Αν και οι διεπαφές προγραμματισμού εφαρμογών (μεταξύ συγκεκριμένων προμηθευτών και ιδιοκτητών) δεν πληρούν πάντα την απαίτηση της φορητότητας σε μεγάλο εύρος αρχιτεκτονικών, υπάρχει το βιομηχανικό πρότυπο OpenCL το οποίο έχει δημιουργηθεί για διεξαγωγή ετερογενών παράλληλων προγραμματισμών και το οποίο προϋποθέτει την ιδιότητα της φορητότητας υπεράνω διαφόρων πλατφορμών. Ωστόσο, η φορητότητα απόδοσης δεν είναι εγγυημένη στο πλαίσιο OpenCL και, για το λόγο αυτόν, η εισαγωγή και μελέτη μιας αυτοματοποιημένης λειτουργικής προσέγγισης στο πρόβλημα της επίτευξης φορητότητας απόδοσης πρέπει να θεωρείται ως απαραίτητη προϋπόθεση προκειμένου να εξασφαλιστεί μια αποτελεσματική μέθοδος συγκριτικής αξιολόγησης σε ετερογενή υπολογιστικά συστήματα.

Έχοντας υπ' όψη τα παραπάνω, βασικός σκοπός της παρούσας Μεταπτυχιακής Διατριβής είναι η εξέταση του προβλήματος της φορητότητας συγκριτικής αξιολόγησης (κυρίως της απόδοσης) του εκάστοτε παρουσιαζόμενου σημείου αναφοράς σε ετερογενείς πλατφόρμες υπολογιστών στις οποίες μπορούμε να αναπαραστούμε με ακρίβεια υπολογιστικές δυνατότητες αναδυόμενων υβριδικών συστημάτων.

Προς αυτήν την κατεύθυνση, θα θεωρήσουμε μια προσέγγιση η οποία θα παράσχει μια ουσιαστική και στέρεα μέθοδο συγκριτικής αξιολόγησης για ετερογενείς πλατφόρμες που διαθέτουν

· φορητότητα απόδοσης,
· αξιόπιστα σχήματα χρονισμού και
· σύγκλιση χρόνων εκτέλεσης.

Για το σκοπό αυτό, θα αναζητηθεί μια τεχνική παραμετροποίησης πυρήνων και αυτόματης βελτιστοποίησης της υψηλής απόδοσης με σκοπό την υποστήριξη μεγάλης ποικιλίας επεξεργαστών. Επιπλέον, θα συζητηθεί το πρόβλημα της δυναμικής δειγματοληψίας και της ανάλυσης κλιμάκωσης αφενός για τη μέτρηση των σημείων αναφοράς της συγκριτικής αξιολόγησης επιδιώκοντας μεγαλύτερη συνέπεια ως προς τα

δεδομένα των εφαρμογών και αφετέρου για την υποβοήθηση του προσανατολισμού και, γενικότερα, της κατεύθυνσης της δειγματοληψίας επί της πλατφόρμας.

ΕΙΣΑΓΩΓΗ

Οι διάφορες υπολογιστικές πλατφόρμες είναι ευρέως χρησιμοποιούμενες σε σύγχρονα συστήματα, καλύπτοντας ένα ευρύ φάσμα από κινητές συσκευές έως υπερυπολογιστές. Κάνοντας αναφορά στο γεγονός ότι το Εθνικό Εργαστήριο του Λος Άλαμος και το Roadrunner της IBM έχουν επιτύχει τα 1 petaflop, τα υβριδικά μοντέλα με τη χρήση επιταχυντών συνεχίζουν να αποτελούν την κορυφαία κατηγορία υπερυπολογιστών παγκοσμίως. Ο κύριος λόγος πίσω από αυτό το έργο βασίζεται στον παρατηρητή ότι οι τρέχουσες τάσεις και οι μελλοντικές εξελίξεις επικεντρώνονται σε υπολογιστικές πλατφόρμες με διαφορετικές αρχιτεκτονικές. Καθώς η ποικιλία των συστημάτων αυξάνεται, είναι αναγκαία η ύπαρξη μιας μεθόδου που να επιτρέπει τη σύγκριση των επιδόσεων μεταξύ διαφορετικών τύπων υπολογιστικών συστημάτων. Αυτή η μέθοδος μπορεί να θεωρηθεί ως ένας τρόπος για να διασφαλιστεί ότι η αξιολόγηση της απόδοσης είναι δίκαιη σε μια ποικιλία εφαρμογών και αρχιτεκτονικών. Η χρήση των μεθόδων αξιολόγησης παρέχει ένα αξιόπιστο μέσο για την αξιολόγηση της απόδοσης, καθώς και ένα κοινό πλαίσιο μέτρησης για τη σύγκριση των επιδόσεων σε διάφορες αρχιτεκτονικές.

Για να υποστηριχθεί η σύγκριση της απόδοσης σε ετερογενείς πλατφόρμες, απαιτείται η ανάπτυξη ενός πολύπλευρου κώδικα που να είναι σε θέση να αλληλεπιδρά με διάφορες αρχιτεκτονικές. Ένας φορητός κώδικας πρέπει να είναι ευέλικτος και να μπορεί να προσαρμόζεται αυτόματα στις υπολογιστικές συσκευές χωρίς την ανάγκη επέμβασης των χρηστών για προσαρμογές σε κάθε συγκεκριμένη αρχιτεκτονική. Παράλληλα, πρέπει να υποστηρίζει διάφορες αρχιτεκτονικές όπως CPUs, GPUs, MICs, FPGAs και ARMs.

Τα προγραμματιστικά διεπαφές εφαρμογών (APIs) που παρέχονται από συγκεκριμένους προμηθευτές, όπως το CUDA, δεν είναι κατάλληλα για την επίτευξη φορητότητας σε πολλές αρχιτεκτονικές. Αντίθετα, το πρότυπο OpenCL αντιπροσωπεύει μια βιομηχανική προσέγγιση για τον ετερογενή παράλληλο προγραμματισμό με στόχο τη φορητότητα πλατφόρμας. Ωστόσο, η διασφάλιση ότι η απόδοση είναι συνεπής σε διάφορες πλατφόρμες δεν είναι πάντα εφικτή με το OpenCL. Ως αποτέλεσμα, είναι αναγκαία μια αυτοματοποιημένη προσέγγιση προκειμένου να επιτευχθεί η φορητότητα της απόδοσης σε ετερογενείς αρχιτεκτονικές, προκειμένου να υπάρξει μια ουσιαστική μέθοδος συγκριτικής αξιολόγησης.

Μια σουίτα αναφοράς συνήθως περιλαμβάνει ένα σύνολο κοινών προγραμμάτων εφαρμογής. Αυτά τα προγράμματα χρησιμοποιούνται είτε για να μετρήσουν την

απόδοση σε διάφορες αρχιτεκτονικές επεξεργαστών είτε για να προβλέψουν την απόδοση σε ένα συγκεκριμένο είδος αρχιτεκτονικής. Μια σουίτα αναφοράς δεν περιορίζεται μόνο στα προγράμματα εφαρμογής, γνωστά και ως πυρήνες. Αυτή η μεθοδολογία αναφέρεται στον γενικό τρόπο που χρησιμοποιείται για να διασφαλιστεί μια αξιόπιστη σύγκριση μεταξύ διαφορετικών αρχιτεκτονικών και για να γίνει μια ακριβής πρόβλεψη της απόδοσης σε όλες τις αρχιτεκτονικές. Μερικά παραδείγματα τεχνικών που χρησιμοποιούνται περιλαμβάνουν την αυτόματη προσαρμογή και τη δυναμική δειγματοληψία. Η πολυπλοκότητα των σύγχρονων υπολογιστικών πλατφορμών αποτελεί πρόκληση για την ανάπτυξη ενός συστήματος που μπορεί να συγκρίνει αξιόπιστα διαφορετικές αρχιτεκτονικές. Αυτές οι σύγχρονες πλατφόρμες αποτελούνται από μια ποικιλία πολυπύρηνων και πολυπύρηνων επεξεργαστών, που εισάγουν ετερογενή υπολογιστική με μία ποικιλία τύπων επεξεργαστών και μοναδικούς τρόπους παραλληλισμού για κάθε αρχιτεκτονική. Συνεπώς, η ακριβής προσέγγιση για τη σύγκριση των αποδόσεων των πυρήνων εφαρμογής σε ετερογενή συστήματα αποτελεί πρόκληση και παραμένει ανοικτό ερευνητικό θέμα. Κατά τη διαδικασία σύγκρισης των αποδόσεων συγκριτικής αξιολόγησης μεταξύ διαφορετικών αρχιτεκτονικών σε ένα ετερογενές σύστημα, είναι ουσιώδους σημασίας να διασφαλιστεί ότι γίνεται σύγκριση μεταξύ της βέλτιστης απόδοσης σε μια αρχιτεκτονική και της βέλτιστης απόδοσης σε μια άλλη αρχιτεκτονική, πράγμα που αποτελεί μια προκλητική εργασία λόγω των ριζικών διαφορών μεταξύ των αρχιτεκτονικών. Πρέπει να επιβεβαιωθεί ότι η ίδια υλοποίηση του ίδιου πυρήνα εφαρμογής λειτουργεί με τις βέλτιστες παραμέτρους για τη συγκεκριμένη αρχιτεκτονική. Αυτό επιτρέπει να πραγματοποιούνται δίκαιες συγκρίσεις μεταξύ διαφορετικών αρχιτεκτονικών.

Επιπλέον, μια σημαντική πρόκληση και στόχος για μια ποικιλομορφική σουίτα δεικτών αναφοράς είναι να θεμελιώσει μια σχέση μεταξύ των αποτελεσμάτων απόδοσης και των πραγματικών εφαρμογών που χρησιμοποιούνται σε σύγχρονες υπολογιστικές πλατφόρμες. Επιπλέον, οι δυνατότητες των μεθόδων συγκριτικής αξιολόγησης επεκτείνονται στον τομέα της ανάπτυξης ενός μηχανισμού που μπορεί να προβλέπει τον χρόνο εκτέλεσης διαφορετικών κατηγοριών εφαρμογών. Οι ακριβείς και εγκαίρως διεξαγόμενες αξιολογήσεις εφαρμογών σε νέες αρχιτεκτονικές είναι πολύτιμο εργαλείο για την κοινότητα των υπολογιστών υψηλής απόδοσης (HPC).

Ο βασικός στόχος της προσέγγισής μας είναι η δυνατότητα μεταφοράς του παρουσιαζόμενου δείκτη αναφοράς σε διάφορες ετερογενείς υπολογιστικές πλατφόρμες. Αυτό μας επιτρέπει να εξετάσουμε πιο λεπτομερώς τις υπολογιστικές δυνατότητες των νέων υβριδικών συστημάτων. Παρουσιάζουμε μια καινοτόμα προσέγγιση που παρέχει μια σταθερή και αξιόπιστη μέθοδο για τη σύγκριση ετερογενών πλατφορμών. Η προσέγγισή μας περιλαμβάνει φορητότητα απόδοσης, αξιόπιστα χρονοδιαγράμματα και συγχρονισμένες εκτελέσεις. Έχουμε αναπτύξει μια τεχνική παραμετροποίησης και αυτόματης βελτιστοποίησης του πυρήνα για την επίτευξη υψηλής απόδοσης σε μια ευρεία γκάμα επεξεργαστών. Επιπλέον, αυτή η μεθοδολογία περιλαμβάνει τη χρήση

δυναμικής δειγματοληψίας και ανάλυσης κλιμάκωσης για την πιο συνεπή μέτρηση των σημείων αναφοράς, ενισχύοντας την εφαρμογή στο πλαίσιο της πλατφόρμας.

ΚΕΦΑΛΑΙΟ 1

1.1 Ορισμοί

"Φορητοτητα Συγκριτικης Αξιολογησης Αποδοσης Σε Ετερογενεις Υπολογιστικες Πλατφορμες" αναφέρεται σε έναν τομέα της επιστήμης των υπολογιστών και της πληροφορικής. Ας αναλύσουμε τους όρους αυτής της φράσης:

Φορητότητα: Στον τομέα της πληροφορικής, η φορητότητα αναφέρεται στην ικανότητα εφαρμογής ενός συστήματος ή λογισμικού σε διάφορες υπολογιστικές πλατφόρμες χωρίς ή με ελάχιστες τροποποιήσεις.

Συγκριτική Αξιολόγηση: Η συγκριτική αξιολόγηση αναφέρεται στη διαδικασία σύγκρισης διαφόρων στοιχείων ή συστημάτων με σκοπό τον προσδιορισμό των διαφορών και των πλεονεκτημάτων τους.

Απόδοση: Η απόδοση είναι μια μέτρηση που αναφέρεται στην αποτελεσματικότητα ενός υπολογιστικού συστήματος ή εφαρμογής. Συνήθως, αφορά την ταχύτητα, την αποκρισιμότητα και άλλες παραμέτρους που μετρούν την απόδοση ενός συστήματος.

Ορισμός της φορητότητας: Υπάρχουν συγκεκριμένα κριτήρια που πρέπει να πληροί ένας επίσημος ορισμός της φορητότητας. Αυτός ο ορισμός πρέπει:

1.Να αντικατροπτίζει τις ξεχωριστές σημασίες των όρων "επίδοση" και "φορητότητα".

2.Να είναι αντικειμενικός.

3.Να είναι μετρήσιμος (και οι συγκρίσεις πρέπει να έχουν νόημα όταν πραγματοποιούνται με τη μετρημένη τιμή.)

Η ικανοποίηση αυτών των κριτηρίων διασφαλίζει: ότι η χρήση του όρου είναι σύμφωνη με τις προσδοκίες του αναγνώστη (δηλαδή, ότι οι εφαρμογές με φορητή απόδοση είναι αποτελεσματικές σε πολλές πλατφόρμες);
Η διαπίστωση ότι οι δηλώσεις σχετικά με τη φορητότητα της απόδοσης βασίζονται σε γεγονότα και δεν εξαρτώνται από απόψεις των γλωσσών προγραμματισμού και πλατφορμών, επισημαίνει ότι υπάρχει μια αντικειμενική βάση για αυτές τις δηλώσεις. Επιπλέον, υπογραμμίζει την ύπαρξη ενός τυποποιημένου τρόπου αξιολόγησης της επιτυχίας των κοινοτικών προσπαθειών για την προώθηση της φορητότητας της απόδοσης και ότι ένας τελικός χρήστης που έχει τη δυνατότητα επιλογής μεταξύ δύο εφαρμογών είναι σε θέση να σκεφτεί ποια είναι η πιο φορητή σε απόδοση σε διάφορες πλατφόρμες υλικού που είναι διαθέσιμες για αυτόν.

Παρόλο που η παραγωγικότητα είναι ένας από τους κινητήριους παράγοντες πίσω από το ενδιαφέρον της κοινότητας για λύσεις στην επίδοση φορητότητας (συγκρίνεται με τη συνάντηση του Υπουργείου Ενέργειας το 2016 για το θέμα [4]), εξαιρούμε

επίτηδες την παραγωγικότητα από τα κριτήρια ορισμού μας, επειδή είναι σε αντίφαση με την αντικειμενικότητα και τη μετρησιμότητα. Η απόδοση ενός προγραμματιστή εξαρτάται κυρίως από τις ικανότητές του, ενώ συνήθεις μετρικές παραγωγικότητας (όπως ο αριθμός γραμμών ή λέξεων κώδικα) δεν αντανακλούν πάντα το γεγονός ότι ένα εργαλείο ή μια βιβλιοθήκη μπορεί να βοηθήσει περισσότερο στην παραγωγικότητα κατά τη χρήση παρά στην ανάπτυξη ή συντήρηση. Ενώ η ύπαρξη ενός συγκεκριμένου ορισμού για την παραγωγική φορητότητα είναι πέραν του πλαισίου αυτής της συζήτησης, η αξιολόγηση της παραγωγικότητας ανεξάρτητα από την εφαρμογή της φορητότητας είναι συμβατή με τις τρέχουσες προσπάθειες μέτρησης των επιπτώσεων διαφόρων εργαλείων φορητότητας στην παραγωγικότητα των προγραμματιστών - υπήρξαν πολλές σοβαρές προσπάθειες για αυτό τον σκοπό.

Ετερογενείς Υπολογιστικές Πλατφόρμες: Αναφέρεται σε πλατφόρμες ή υπολογιστικά περιβάλλοντα που διαφέρουν μεταξύ τους σε θεμελιώδεις τρόπους, όπως αρχιτεκτονική, λειτουργικό σύστημα, επεξεργαστές, γλώσσες προγραμματισμού κλπ.

Ένας ορισμός για τη "Φορητότητα Συγκριτικής Αξιολόγησης Απόδοσης σε Ετερογενείς Υπολογιστικές Πλατφόρμες" θα μπορούσε να είναι ο εξής:

Η φορητότητα συγκριτικής αξιολόγησης απόδοσης σε ετερογενείς υπολογιστικές πλατφόρμες αναφέρεται στη διαδικασία σχεδιασμού, ανάπτυξης και αξιολόγησης λογισμικού ή συστημάτων που μπορούν να λειτουργήσουν αποδοτικά και αξιόπιστα σε ποικίλες υπολογιστικές πλατφόρμες, παρέχοντας τα ίδια επίπεδα απόδοσης ανεξαρτήτως της αρχιτεκτονικής του υπολογιστή, του λειτουργικού συστήματος και άλλων παραμέτρων.

Η ιστορική αναδρομή του θέματος θα μπορούσε να περιλαμβάνει την εξέλιξη των τεχνολογιών φορητότητας, την ανάπτυξη εργαλείων που υποστηρίζουν την ετερογενή αξιολόγηση απόδοσης και την επέκταση της έννοιας σε νέες πλατφόρμες, όπως τα κινητά τηλέφωνα, τα αισθητήρια IoT και άλλες σύγχρονες υπολογιστικές συσκευές.

1.2 Αναφορά

Η φορητότητα συγκριτικής αξιολόγησης απόδοσης σε ετερογενείς υπολογιστικές πλατφόρμες είναι ένας σημαντικός τομέας στην επιστήμη των υπολογιστών και την πληροφορική. Αυτή η διακυβέρνηση αφορά τη δυνατότητα ανάπτυξης λογισμικού και συστημάτων που μπορούν να λειτουργήσουν αποδοτικά σε πολλαπλές υπολογιστικές πλατφόρμες.

Η ιστορία αυτού του πεδίου μπορεί να ανιχνευθεί από την εξέλιξη των τεχνολογιών φορητότητας, από τον αναπτυξιακό κύκλο των προγραμμάτων που πρέπει να λειτουργούν σε διάφορες αρχιτεκτονικές, και από την ανάγκη για εργαλεία και μεθόδους που υποστηρίζουν αυτήν την φορητότητα.

Κατά την εξέλιξη της τεχνολογίας, η φορητότητα συγκριτικής αξιολόγησης απόδοσης έχει επεκταθεί σε νέες πλατφόρμες, συμπεριλαμβανομένων των κινητών τηλεφώνων, των αισθητήρων IoT, των ενσωματωμένων συστημάτων και πολλών άλλων σύγχρονων υπολογιστικών συσκευών.

Η έρευνα και η ανάπτυξη σε αυτόν τον τομέα συνεχίζουν να είναι σημαντικές, καθώς η ανάγκη για λογισμικό και συστήματα που μπορούν να προσαρμοστούν σε διάφορες πλατφόρμες συνεχίζεται να αυξάνεται με την εξέλιξη της τεχνολογίας.

Η φορητότητα συγκριτικής αξιολόγησης απόδοσης σε ετερογενείς υπολογιστικές πλατφόρμες αναφέρεται στη δυνατότητα μέτρησης και σύγκρισης της απόδοσης φαρμογών ή αλγορίθμων σε διάφορες υπολογιστικές περιβάλλοντα ή πλατφόρμες που είναι ετερόκλητες ως προς την υλικοσυνθετική τους διάρθρωση. Η φορητότητα αυτή είναι σημαντική για διάφορους λόγους, όπως:

- Ευκολία συγκριτικής αξιολόγησης: Οι ερευνητές και οι αναπτυσσόμενοι επιθυμούν να μπορούν να συγκρίνουν την απόδοση των εφαρμογών ή των αλγορίθμων τους σε διάφορα υπολογιστικά περιβάλλοντα.
- Εκμετάλλευση των πόρων: Εκμεταλλευόμενοι τη φορητότητα, οι εφαρμογές μπορούν να εκτελούνται σε διάφορες πλατφόρμες, αξιοποιώντας τα διάφορα χαρακτηριστικά και τους πόρους που προσφέρει καθεμία.

Για την επίτευξη φορητότητας συγκριτικής αξιολόγησης απόδοσης, μπορούν να ληφθούν υπόψη τα παρακάτω:

- Προδιαγραφές ανοιχτού κώδικα: Χρησιμοποιήστε ανοιχτού κώδικα βιβλιοθήκες και πλατφόρμες που είναι διαθέσιμες σε διάφορα υπολογιστικά περιβάλλοντα. Αυτό επιτρέπει στους χρήστες να αναπτύσσουν τον κώδικά τους μία φορά και να τον εκτελούν σε διάφορες πλατφόρμες.
- Χρήση εικονικοποίησης: Η εικονικοποίηση επιτρέπει τη δημιουργία εικονικών περιβαλλόντων που εκτελούνται σε διάφορες πλατφόρμες. Αυτό μπορεί να επιτρέψει στον κώδικα να εκτελείται ανεξάρτητα από το υποστηριζόμενο λειτουργικό σύστημα και την υλική υποδομή.
- Εξομοίωση περιβάλλοντος: Η εξομοίωση περιβάλλοντος επιτρέπει την αναπαράσταση ενός συγκεκριμένου υπολογιστικού περιβάλλοντος σε άλλη πλατφόρμα. Αυτό είναι χρήσιμο για αξιολόγηση της απόδοσης σε διάφορες περιβαλλοντικές συνθήκες.
- Χρήση αφαιρετικών διεπαφών: Η δημιουργία αφαιρετικών διεπαφών (API) επιτρέπει την ανταλλαγή δεδομένων και την επικοινωνία μεταξύ διαφορετικών μερών της εφαρμογής, ανεξάρτητα από την υποκείμενη υλική υποδομή.

Με αυτούς τους τρόπους, είναι δυνατό να δημιουργηθούν εφαρμογές και αλγόριθμοι που είναι φορητοί και μπορούν να εκτελούνται αποτελεσματικά σε ετερογενείς υπολογιστικές πλατφόρμες.

ΚΕΦΑΛΑΙΟ 2

Συνεισφορές

Οι τάσεις τόσο στους καταναλωτικούς όσο και στους υπερ-υπολογιστές φέρνουν όχι μόνο περισσότερους πυρήνες, αλλά επίσης αυξημένη ετερογένεια μεταξύ των υπολογιστικών πόρων εντός ενός μόνο μηχανήματος. Σε πολλούς υπολογιστές, οι επεξεργαστές γραφικών (GPUs) έχουν γίνει ένας από τους κύριους πόρους υπολογισμού, όχι μόνο οι κεντρικοί επεξεργαστές (CPUs). Ωστόσο, ο προγραμματισμός και οι τρόποι λειτουργίας της μνήμης στους GPUs είναι διαφορετικοί σε σχέση με τους συμβατικούς CPUs, και η απόδοσή τους διαφέρει σημαντικά μεταξύ των διαφορετικών μηχανημάτων. Συχνά, διάφοροι επεξεργαστές μέσα σε ένα σύστημα χρησιμοποιούν διαφορετικούς αλγορίθμους και πρότυπα χρήσης μνήμης για βέλτιστη απόδοση, και η επίτευξη της καλύτερης συνολικής απόδοσης μπορεί να απαιτεί την αποτελεσματική χρήση όλων των τύπων πόρων στο σύστημα.

Για να λύσουμε το πρόβλημα του αποδοτικού προγραμματισμού σε μηχανήματα που διαθέτουν διάφορους τύπους υπολογιστικών πόρων, προτείνουμε ένα μοντέλο προγραμματισμού όπου η βέλτιστη αντιστοίχιση προγραμμάτων σε επεξεργαστές και μνήμες καθορίζεται με βάση πρακτικές εμπειρίες και παρατηρήσεις. Τα προγράμματα ορίζουν επιλογές στο πώς μπορεί να λειτουργήσουν οι αλγόριθμοί τους, και ο μεταγλωττιστής δημιουργεί περαιτέρω επιλογές στο πύργο των επεξεργαστών CPU και GPU και στα συστήματα μνήμης. Αυτές οι επιλογές δίνονται σε ένα πλαίσιο εμπειρικής αυτόματης ρύθμισης που επιτρέπει την αναζήτηση του χώρου δυνατών υλοποιήσεων κατά την εγκατάσταση. Η πληθώρα επιλογών που προσφέρεται επιτρέπει στο σύστημα αυτόματης ρύθμισης να δημιουργήσει πολύπλοκους αλγορίθμους που συνδυάζουν διάφορες τεχνικές. Αυτοί οι αλγόριθμοι χρησιμοποιούν τόσο τον κεντρικό επεξεργαστή (CPU) όσο και το γραφικό επεξεργαστή (GPU) για να επιτύχουν καλύτερη απόδοση σε σχέση με τη χρήση μιας μόνο τεχνικής. Τα πειραματικά αποτελέσματα έδειξαν ότι οι αλλαγές στους αλγορίθμους και η διαφορετική χρήση τόσο των CPUs όσο και των GPUs αποτελούν αναγκαίες προϋποθέσεις για την επίτευξη επιτάχυνσης έως 16.5 φορές σε σύγκριση με τη χρήση μιας μοναδικής ρύθμισης προγράμματος για όλες τις αρχιτεκτονικές.

2.1. Αρχιτεκτονική Υπολογιστών: Από τους Πολυπύρηνους στα Ετερογενή Συστήματα"

Την τελευταία δεκαετία παρατηρείται μια έκρηξη στην παράλληλη επεξεργασία των επεξεργαστών με τη μετάβαση στους πολυπύρηνους επεξεργαστές. Μια σημαντική τάση στις επόμενες αρχιτεκτονικές είναι η μετάβαση από τους παραδοσιακούς, συμμετρικούς πολυεπεξεργαστές σε πιο ασύμμετρα συστήματα, που περιλαμβάνουν πολλούς διαφορετικούς επεξεργαστές και μνήμες. Ένας σημαντικός παράγοντας αυτής της ετερογένειας είναι η ευρεία διαθεσιμότητα των γραφικών επεξεργαστών (GPUs), οι οποίοι τώρα χρησιμοποιούνται ευρέως για γενικού σκοπού υπολογισμού. Οι GPUs προσφέρουν σημαντικές επιταχύνσεις λόγω της υψηλής πυκνότητας των επεξεργαστών και του υψηλού εύρους ζώνης μνήμης. Συνήθως, θεωρείται ότι αν ένα πρόγραμμα μπορεί να εκτελεστεί στο GPU, θα πρέπει να γίνεται, καθώς μπορεί να επωφεληθεί από την υψηλή υπολογιστική ισχύ που προσφέρει.. Ωστόσο, τα αποτελέσματά μας δείχνουν ότι οι επιλογές για την αντιστοίχιση σε ετερογενή μηχανήματα είναι παραπολύπλοκες. Ένας σημαντικός προκλητικός παράγοντας στη χρήση ετερογενών πόρων είναι η ποικιλία των συσκευών σε διάφορα μηχανήματα, οι οποίες παρέχουν διαφορετικά χαρακτηριστικά απόδοσης. Ένα πρόγραμμα που έχει βελτιστοποιηθεί για να λειτουργεί σε έναν συγκεκριμένο τύπο επεξεργαστή μπορεί να μην λειτουργεί με την ίδια αποτελεσματικότητα σε έναν διαφορετικό τύπο επεξεργαστή ή σε μια συσκευή που κατασκευάζεται από διαφορετικό κατασκευαστή. Επίσης, ένα πρόγραμμα που έχει βελτιστοποιηθεί για να λειτουργεί σε μια μονάδα επεξεργασίας γραφικών (GPU) μπορεί να είναι πολύ διαφορετικό από ένα πρόγραμμα που έχει βελτιστοποιηθεί για να λειτουργεί σε μια μονάδα επεξεργασίας κεντρικής μονάδας (CPU). Επιπλέον, η σχετική απόδοση των CPUs και GPUs μπορεί να διαφέρει ανάλογα με το μηχάνημα. Σε ορισμένες περιπτώσεις, ένας συγκεκριμένος τύπος υπολογισμού μπορεί να εκτελείται καλύτερα σε ένα CPU, ενώ σε άλλες περιπτώσεις μπορεί να εκτελείται καλύτερα σε ένα GPU. Υπάρχει επίσης ένα πρόβλημα προγραμματισμού: ακόμη κι αν ένα συγκεκριμένο κομμάτι κώδικα μπορεί να αποδώσει καλύτερα στο GPU, αν το GPU είναι υπερφορτωμένο και η CPU είναι αδρανής, τότε μπορεί να είναι καλύτερο να ισορροπηθεί το φορτίο μεταξύ των δύο, ή μπορεί να είναι καλύτερο να τοποθετηθούν οι υπολογισμοί κάπου που εκτελούνται πιο αργά αλλά κοντά στο σημείο όπου θα χρησιμοποιηθεί η έξοδος τους. Σε πολλές περιπτώσεις, η απόδοση ενός προγράμματος εξαρτάται από την ικανότητά του να χρησιμοποιεί διαφορετικούς πόρους ενός υπολογιστικού συστήματος. Για παράδειγμα, σε ορισμένες περιπτώσεις, η βέλτιστη απόδοση επιτυγχάνεται όταν ένα πρόγραμμα χρησιμοποιεί ταυτόχρονα και τον επεξεργαστή (CPU) και τη μονάδα επεξεργασίας γραφικών (GPU), ενώ σε άλλες περιπτώσεις μπορεί να είναι πιο αποδοτικό να χρησιμοποιεί μόνο τον έναν ή τον άλλον. Η διαφορετική απόδοση ανάλογα με τους διαθέσιμους πόρους επηρεάζει την ανάπτυξη αρχιτεκτονικών, καθώς και την αξιοποίηση των πόρων ενός συστήματος. Οι ερευνητές προτείνουν νέες αρχιτεκτονικές με διαφορετικούς τύπους πυρήνων, ενώ ακόμη και οι πολυπύρηνες αρχιτεκτονικές δεν λειτουργούν πάντα όπως αναμένεται, καθώς η απόδοσή τους εξαρτάται από τον τρόπο λειτουργίας τους και τον τρόπο διαχείρισης των πόρων

τους. Συνεπώς, η βέλτιστη αξιοποίηση των πόρων ενός συστήματος απαιτεί προσαρμοστικές και εξελισσόμενες στρατηγικές.

Ο βασικός στόχος ενός μεταγλωττιστή είναι να προσαρμόσει ένα πρόγραμμα ώστε να λειτουργεί σωστά σε ένα συγκεκριμένο σύστημα υλικού. Ο μεταγλωττιστής πραγματοποιεί βελτιστοποιήσεις, λαμβάνοντας υπόψη ένα απλοποιημένο μοντέλο του συγκεκριμένου επεξεργαστή. Αυτό το μοντέλο αναφέρεται στην απόδοση του επεξεργαστή και καθοδηγεί τις βελτιστοποιήσεις που πραγματοποιούνται. Ωστόσο, σε ετερογενή συστήματα υπολογιστών, όπου χρησιμοποιούνται διαφορετικοί τύποι επεξεργαστών και υποσυστήματα μνήμης, είναι δύσκολο να χρησιμοποιηθεί ένα μοντέλο που να αντιπροσωπεύει τη συμπεριφορά όλων των πιθανών περιβαλλόντων εκτέλεσης. Καθώς οι επεξεργαστές και τα συστήματα μνήμης μπορεί να διαφέρουν σημαντικά, είναι προβληματικό να χρησιμοποιηθεί ένα μοντέλο που να είναι ικανό να προβλέψει τη συμπεριφορά του προγράμματος σε όλες αυτές τις πιθανές περιπτώσεις.. Ο μεταγλωττισμός ενός προγράμματος για ένα ετερογενές μηχάνημα απαιτεί πολύπλοκες, αλληλοεξαρτώμενες επιλογές αλγορίθμου: ποιον αλγόριθμο να χρησιμοποιήσεις, πού να τοποθετήσεις υπολογισμούς και δεδομένα και πώς να αντιστοιχίσεις το παράλληλισμο και τη χρήση εξειδικευμένων μνημών.

Ο κύριος σκοπός ενός μεταγλωττιστή είναι να προσαρμόσει ένα πρόγραμμα ώστε να λειτουργεί σωστά σε ένα συγκεκριμένο σύστημα υλικού. Κατά τη μεταγλώττιση, ο μεταγλωττιστής προσπαθεί να βελτιστοποιήσει το πρόγραμμα, λαμβάνοντας υπόψη ένα απλοποιημένο μοντέλο του επεξεργαστή του συστήματος. Αυτό το μοντέλο αναφέρεται στην απόδοση του επεξεργαστή και καθοδηγεί τις βελτιστοποιήσεις που γίνονται στο πρόγραμμα. Ωστόσο, σε συστήματα με διαφορετικούς τύπους επεξεργαστών και υποσυστήματα μνήμης, όπως τα ετερογενή συστήματα υπολογιστών, είναι δύσκολο να χρησιμοποιηθεί ένα μοντέλο που να αντιπροσωπεύει τη συμπεριφορά σε όλα τα πιθανά περιβάλλοντα εκτέλεσης. Εξαιτίας των σημαντικών διαφορών μεταξύ των επεξεργαστών και των συστημάτων μνήμης, είναι δύσκολο να δημιουργηθεί ένα μοντέλο που θα μπορεί να προβλέπει πώς θα εκτελεστεί το πρόγραμμα σε όλες αυτές τις πιθανές περιπτώσεις.

Ο προγραμματιστής , επομένως, είναι σε θέση να καθορίσει αλγοριθμικές επιλογές στο επίπεδο της γλώσσας, όπου επεκτείνει τη γλώσσα και τον μεταγλωττιστή PetaBricks [3]. Χρησιμοποιώντας αυτήν την προσέγγιση, τα προγράμματα PetaBricks δεν περιορίζονται σε έναν μόνο αλγόριθμο, αλλά αντίθετα, δίνουν τη δυνατότητα για ένα ευρύτερο φάσμα πιθανών αλγορίθμων. Οι επεκτάσεις στο PetaBricks που παρουσιάζονται σε αυτήν την εργασία περιλαμβάνουν:

Με τη χρήση περασμάτων μεταγλώττισης και στατικών αναλύσεων, γίνεται η αυτόματη μετατροπή υποσυνόλων των υφιστάμενων προγραμμάτων PetaBricks σε πυρήνες OpenCL. Αυτοί οι πυρήνες μπορούν να λειτουργήσουν σε διάφορες αρχιτεκτονικές πίσω από το σύστημα και μπορούν να ενσωματωθούν ως επιλογές που

είναι διαθέσιμες στον αυτορυθμιστή. Αυτή η διαδικασία περιλαμβάνει στατικές αναλύσεις που βοηθούν στην αποτελεσματική διαχείριση της μνήμης αυτών των πολυπύρηνων συσκευών και στη μείωση της μεταφοράς δεδομένων μεταξύ των εκτελέσεων των πυρήνων.

Ένα χρονοδιαγραφία διαχείρισης GPU που επιτρέπει την αποτελεσματική χρήση των συσκευών συνεπεξεργασίας στο χρόνο εκτέλεσης PetaBricks χωρίς να απαιτείται η αποκοπή του νήματος CPU που καλείται να εκτελέσει τις λειτουργίες GPU, παρέχοντας αυτόματη διαχείριση μνήμης και επιτρέποντας την αποτελεσματική επικάλυψη της υπολογιστικής εργασίας και της επικοινωνίας.

Βελτιώσεις στον αυτορυθμιστή για την αναζήτηση διαφορετικών διαιρέσεων της εργασίας μεταξύ GPU και CPU και για τη δυνατότητα εξερεύνησης επιλογών στο GPU, όπως επιλογές τοποθέτησης διαφόρων χώρων μνήμης για διάφορα δεδομένα και επιλογές πόσα δεδομένα να εκτελεστούν στο GPU.

Μέσω ποικίλων πειραμάτων, αποδεικνύεται ότι διάφορα ετερογενή συστήματα απαιτούν τη χρήση διαφορετικών αλγοριθμικών επιλογών για να επιτευχθεί η βέλτιστη απόδοση. Είναι γεγονός ότι, ακόμη και όταν υπάρχει μετακίνηση μεταξύ δύο αρχιτεκτονικών με σύγχρονους GPUs, μπορεί να επιτευχθεί επιτάχυνση έως και 16,5 φορές με εκπαίδευση για κάθε αρχιτεκτονική. Αποδεικνύεται πώς διάφορες αρχιτεκτονικές επιτυγχάνουν την καλύτερη απόδοση με διαφορετικούς τρόπους κατανομής της εργασίας μεταξύ του GPU και του CPU σε κάθε μηχάνημα. Οι διάφορες ρυθμίσεις που προέκυψαν είναι συχνά μη-προφανείς, υποδεικνύοντας ότι απαιτούνται εμπειρικές τεχνικές προσαρμογής για την επίτευξη βέλτιστης απόδοσης σε σύγχρονα συστήματα.

2.2 Προσέγγιση

Μια θεωρία που προτείνεται είναι ότι αυτό το σύστημα είναι ο πρώτος που αυτοματοποιεί τον βέλτιστο χαρτογραφισμό προγραμμάτων σε γλώσσα υψηλού επιπέδου σε ένα σύνολο αποκεντρωμένων παράλληλων μονάδων επεξεργασίας. Αυτό περιλαμβάνει την τοποθέτηση των υπολογισμών, την επιλογή του κατάλληλου αλγορίθμου και την προσαρμογή για εξειδικευμένες ιεραρχίες μνήμης. Με αυτόν τον τρόπο, μια γλώσσα υψηλού επιπέδου, ανεξάρτητη από την αρχιτεκτονική, μπορεί να επιτύχει συγκρίσιμη απόδοση με προγράμματα που έχουν γραφτεί χειροκίνητα για συγκεκριμένες αρχιτεκτονικές.

Για να υλοποιηθεί αυτό το σύστημα,

- ενσωματώνουμε τις επιλογές συσκευών, μνημών και παραμέτρων εκτέλεσης στο μεταγλωττιστή αυτορύθμισης PetaBricks.
- αυτοματοποιούμε τη δημιουργία κώδικα OpenCL και τη διαδικασία επιλογής για μια παραδοσιακά χειρόγραφη βελτιστοποίηση μνήμης αποθηκευτικού

χώρου που απαιτεί σημαντική αναδιατύπωση της πρόσβασης στη μνήμη και τη δημιουργία πολυφασικών συνεργατικών φορτώσεων και αποθηκεύσεων.
- εισάγουμε ένα σύστημα χρόνου εκτέλεσης που ενσωματώνει τη χρήση ενός συνεπεζών GPU στο σύστημα αφαίρεσης εργασίας χωρίς την ανάγκη να αποκλείεται ο καλών CPU νήματος σε λειτουργίες GPU.
- εισάγουμε μια ανάλυση μεταγλωττιστή για τον προσδιορισμό πότε να χρησιμοποιήσουμε τεχνικές διαχείρισης μνήμης χωρίς αντιγραφή, αντιγραφή κατά λήθαρή και αντιγραφή μνήμης προσφυγής.

Μια προσέγγιση του θέματος θα μπορούσε να είναι η εστίαση όχι στο πώς κάθε μεμονωμένη βελτιστοποίηση επηρεάζει το σύστημα, αλλά στο πώς μπορούμε να συνδυάσουμε πολλές τεχνικές μαζί, προκειμένου να αυτοματοποιήσουμε τη δημιουργία κώδικα και να επιτύχουμε αποδοτική λειτουργία σε οποιοδήποτε ετερογενές σύστημα.

Μελετώντας το σύστημα σε διάφορες εφαρμογές και αρχιτεκτονικές, φαίνεται ότι οι βέλτιστες επιδόσεις απαιτούν διαφορετικές επιλογές αλγορίθμων, ειδικά σε μηχανήματα με διάφορους συνδυασμούς επεξεργαστών. Αυτό που επίσης προκύπτει είναι η ανάγκη για αλγοριθμικές επιλογές και μάθησης, προσθέτοντας έναν επιπλέον παράγοντα στην αναζήτηση βέλτιστων λύσεων. Η διαδικασία γίνεται ακόμα πιο περίπλοκη λόγω του ότι η καλύτερη συσκευή που θα χρησιμοποιηθεί εξαρτάται από τον αλγόριθμο, το πλαίσιο και την υλική υποστήριξη που χρησιμοποιούνται. Τέλος, η ανάθεση εργασίας για ταυτόχρονη εκτέλεση σε GPU και CPU φαίνεται να έχει σημαντικά καλύτερη απόδοση από τη χρήση ενός μόνο επεξεργαστή.

2.3 Γλώσσας PetaBricks

Η γλώσσα PetaBricks δίνει στον προγραμματιστή τη δυνατότητα να περιγράψει πολλές διαφορετικές μεθόδους για την επίλυση ενός προβλήματος. Στη συνέχεια, ο αυτορυθμιστής μπορεί να αναλύσει την κατάσταση και να αποφασίσει ποια από αυτές τις μεθόδους είναι η καλύτερη για το συγκεκριμένο περιβάλλον του χρήστη. [3] Παρέχει τόσο αλγοριθμική ευελιξία (πολλαπλές αλγοριθμικές επιλογές) όσο και ευελιξία γεννήτριας κώδικα μεγάλης κόκκινης κλίμακας (συνθετική εξωτερική ροή ελέγχου).

Σε απλά λόγια, στη γλώσσα PetaBricks, ο προγραμματιστής μπορεί να ορίσει έναν μετασχηματισμό που παίρνει μια σειρά εισόδων (τη λέξη-κλειδί "from") και παράγει μια σειρά εξόδων (τη λέξη-κλειδί "to"). Αυτό μπορεί να θεωρηθεί σαν την κλήση μιας συνάρτησης σε άλλες γλώσσες προγραμματισμού. Η διαφορά εδώ είναι ότι ο προγραμματιστής μπορεί να ορίσει πολλούς διαφορετικούς τρόπους για το πώς να μετασχηματίσει τις εισόδους σε εξόδους για αυτόν τον μετασχηματισμό. Οι τρόποι καθορίζονται με τρόπο διαφορετικό από την τυπική ροή δεδομένων χρησιμοποιώντας

πολλά μικρότερα κομμάτια κατασκευής που ονομάζονται "κανόνες" (rules), οι οποίοι κωδικοποιούν τόσο τις εξαρτήσεις δεδομένων του κανόνα όσο και κώδικα παρόμοιο με τη γλώσσα C++ που μετατρέπει τις εισόδους του κανόνα σε εξόδους.

Οι εξαρτήσεις καθορίζονται με τον καθορισμό των εισόδων και εξόδων κάθε κανόνα, αλλά, αντίθετα με τον τυπικό μοντέλο προγραμματισμού ροής δεδομένων, μπορούν να καθοριστούν περισσότεροι από έναν κανόνες για την παραγωγή των ίδιων δεδομένων. Στην PetaBricks, οι εισαγωγικές εξαρτήσεις μπορούν να επιλυθούν μέσω της εξόδου ενός ή περισσότερων κανόνων. Ο μεταγλωττιστής και ο αυτορυθμιστής της PetaBricks καθορίζουν ποιους κανόνες θα χρησιμοποιήσουν για να ικανοποιήσουν αυτές τις εξαρτήσεις, λαμβάνοντας υπόψη την αρχιτεκτονική και την είσοδο δεδομένων. Για παράδειγμα, σε συστήματα με πολλούς επεξεργαστές, ο αυτορυθμιστής μπορεί να επιλέξει κανόνες που μειώνουν τον κρίσιμο δρόμο του μετασχηματισμού, ενώ σε συστήματα με σειριακή αρχιτεκτονική, προτιμώνται κανόνες με χαμηλότερη υπολογιστική πολυπλοκότητα.

ΚΕΦΑΛΑΙΟ 3

Ετερογενής σχεδιασμός συγκριτικής αξιολόγησης

Ο σχεδιασμός συγκριτικής αξιολόγησης ετερογενούς φύσης αναφέρεται στη διαδικασία αξιολόγησης και σύγκρισης των επιδόσεων μεταξύ διαφορετικών συστημάτων, συσκευών, ή αρχιτεκτονικών που έχουν διαφορετική φύση ή χαρακτηριστικά. Αυτή η διαδικασία συγκρίνει την απόδοση των εν λόγω στοιχείων, εξετάζοντας πώς λειτουργούν και επιδρούν σε διάφορες συνθήκες.

Οι τεχνικές συγκριτικής αξιολόγησης είναι σημαντικές για την κατανόηση της απόδοσης των διαφορετικών συστημάτων και την επιλογή του κατάλληλου εξοπλισμού για συγκεκριμένες εφαρμογές. Αυτό μπορεί να συμπεριλαμβάνει τη σύγκριση των επιδόσεων μεταξύ διαφορετικών υπολογιστικών αρχιτεκτονικών, επεξεργαστών, γραφικών επιταχυντών, ή ακόμη και λογισμικού.

Η ετερογενής φύση αναφέρεται στο γεγονός ότι τα συστήματα που συγκρίνονται έχουν διαφορετικές χαρακτηριστικές φυσικές ιδιότητες ή είναι σχεδιασμένα για διαφορετικές εργασίες. Η συγκριτική αξιολόγηση ετερογενούς φύσης μπορεί να συμπεριλάβει την αξιολόγηση της απόδοσης σε όρους επεξεργαστικής ισχύος, κατανάλωσης ενέργειας, ευελιξίας, κόστους ή άλλων παραμέτρων.

Στόχος της ετερογενούς σχεδίασης συγκριτικής αξιολόγησης είναι να προσφέρει έναν αντικειμενικό και συγκριτικό τρόπο αξιολόγησης των διαφορετικών στοιχείων, ώστε οι χρήστες να μπορούν να λαμβάνουν ενημερωμένες αποφάσεις σχετικά με τον κατάλληλο εξοπλισμό για τις ανάγκες τους.

Μερικά από τα σύνολα εφαρμογών που εξετάστηκαν περιλαμβάνουν τα OpenDwarfs, Scalable Heterogeneous Computing (SHOC), Rodinia, Parboil, LINPACK και Phoronix. OpenDwarfs: Το OpenDwarfs είναι ένα πρόγραμμα που περιέχει διάφορα μικρά πυρήνες υπολογιστικής εργασίας (kernels) σχεδιασμένα για τη δοκιμή και τη σύγκριση της απόδοσης σε διάφορες πλατφόρμες. Χρησιμοποιείται συχνά για μελέτες απόδοσης στον τομέα της ετερογενούς υπολογιστικής. Η συλλογή των OpenDwarfs αποτελείται από 13 προγράμματα που αναπτύχθηκαν από το Πολυτεχνείο της Βιρτζίνια και το Πολιτειακό Πανεπιστήμιο. Αυτά τα προγράμματα εφαρμογών καλύπτουν διάφορους τομείς επιστημών και μηχανικής.

Scalable Heterogeneous Computing (SHOC): Το SHOC είναι ένα εργαλείο ανοικτού κώδικα που χρησιμοποιείται για την αξιολόγηση της απόδοσης ετερογενών συστημάτων υπολογισμού. Παρέχει διάφορες εφαρμογές και πυρήνες υπολογιστικής εργασίας που μπορούν να χρησιμοποιηθούν για μελέτες απόδοσης σε διάφορες πλατφόρμες. Οι συγγραφείς προσπάθησαν να μην προσαρμόσουν τα προγράμματα ώστε να λειτουργούν βέλτιστα μόνο σε μια συγκεκριμένη πλατφόρμα. Η σουίτα αξιολόγησης SHOC, που αναπτύχθηκε από το Oak Ridge National Laboratory και το Πανεπιστήμιο του Τενεσί, αποτελείται από τρία επίπεδα προγραμμάτων εφαρμογής. Ο στόχος της είναι η σύγκριση όχι μόνο της απόδοσης της μηχανής, αλλά και των χαρακτηριστικών υλικού χαμηλού επιπέδου. Η SHOC παρέχει υλοποιήσεις σε OpenCL, CUDA και MPI.

Μια άλλη σουίτα που χρησιμοποιείται για την σύγκριση απόδοσης σε ετερογενείς πλατφόρμες είναι η Rodinia. Το Rodinia είναι ένα σύνολο εφαρμογών και πυρήνων υπολογιστικής εργασίας που έχουν σχεδιαστεί για τη δοκιμή της απόδοσης σε ετερογενείς πλατφόρμες. Συχνά χρησιμοποιείται για συγκριτικές αναλύσεις απόδοσης και για την αξιολόγηση της επιδόσεως υπολογιστικών συστημάτων Αποτελείται από 19 προγράμματα εφαρμογής που αναπτύχθηκαν με τη χρήση τεχνολογιών όπως CUDA και OpenCL για αρχιτεκτονικές που βασίζονται σε GPU και MIC.

Το Parboil είναι ένα σύνολο πυρήνων υπολογιστικής εργασίας που χρησιμοποιείται για την αξιολόγηση της απόδοσης σε ετερογενείς υπολογιστικές πλατφόρμες. Παρέχει πυρήνες για διάφορες εφαρμογές, συμπεριλαμβανομένων των εφαρμογών επεξεργασίας εικόνας και υπολογισμού πίνακα. Η σουίτα Parboil αναπτύχθηκε για την ανάλυση της απόδοσης διάφορων υπολογιστικών αρχιτεκτονικών και περιλαμβάνει μια συλλογή από 11 προγράμματα εφαρμογών. Το κύριο αντικείμενο του Parboil είναι να βοηθήσει τους προγραμματιστές να αξιοποιήσουν πλήρως τις δυνατότητες της διαθέσιμης αρχιτεκτονικής υπολογιστών. Η συλλογή αυτή περιλαμβάνει βελτιστοποιημένες εκδόσεις των προγραμμάτων που είναι συμβατές με CUDA, OpenCL και C. Αντίστοιχα, το LINPACK αποτελεί τη σουίτα που χρησιμοποιείται για τη δημιουργία της κατάταξης υπερυπολογιστών TOP500, η οποία ενημερώνεται διετώς. Το LINPACK είναι μια σουίτα προγραμμάτων που αναπτύχθηκε

από τους Dongarra και Luszczek με στόχο τον ακριβή υπολογισμό της μέγιστης απόδοσης ενός υπολογιστικού συστήματος. Χρησιμοποιεί τις βιβλιοθήκες BLAS για την εκτέλεση των βασικών υπολογισμών..

LINPACK: Το LINPACK είναι ένα γνωστό πρόγραμμα για τον υπολογισμό της απόδοσης υπολογιστικών συστημάτων, κυρίως στον τομέα των υπολογισμών γραμμικών αλγεβρικών εξισώσεων. Χρησιμοποιείται συχνά ως μέτρο αξιολόγησης για υπολογιστικές συσκευές. Το Phoronix είναι μια συλλογή δοκιμαστικών προγραμμάτων που περιλαμβάνει περισσότερα από 60 διαφορετικά προγράμματα. Αυτά τα προγράμματα είναι σχεδιασμένα ειδικά για να εκτελούνται σε κάρτες γραφικών. Το Phoronix περιλαμβάνει μια ευρεία γκάμα διαφορετικών εφαρμογών που μπορούν να προσαρμοστούν σε συγκεκριμένες απαιτήσεις και σενάρια χρήσης.

Σε αυτήν την εργασία, γίνεται αναφορά σε πυρήνες από το SHOC ως αφετηρία λόγω της ικανότητας να λειτουργούν σε διάφορες πλατφόρμες, προσφέροντας φορητότητα. Για να μπορέσουμε να προβούμε σε συγκριτική αξιολόγηση μεταξύ ετερογενών πλατφορμών, ο ίδιος ο κώδικας πρέπει να είναι σε θέση να λειτουργεί σε κάθε διαθέσιμη πλατφόρμα που υποστηρίζει το OpenCL. Το SHOC παρέχει τη δυνατότητα εκτέλεσης διάφορων τύπων δοκιμών, ονομαζόμενων επίπεδα εντός της σουίτας, προσφέροντας περισσότερες επιλογές για τη σύγκριση. Επιπρόσθετα, η ανάλυση των χαρακτηριστικών υλικού χαμηλού επιπέδου για μια συγκεκριμένη μηχανή έχει ως αποτέλεσμα μια πιο λεπτομερή κατανόηση των διαφορετικών αρχιτεκτονικών.

Στο παρελθόν, είχε αξιολογηθεί ξεχωριστά αλγόριθμους υπολογισμού έντασης σε ετερογενείς αρχιτεκτονικές που χρησιμοποιούν επιταχυντές ([10,11]). Στην περίπτωσή μας γίνεται προσπάθεια να αναπτυχθεί μια γενική μέθοδο συγκριτικής αξιολόγησης για ετερογενή συστήματα αντί να περιορίζεται σε μια συγκεκριμένη εφαρμογή σε μια συγκεκριμένη πλατφόρμα. Ως αποτέλεσμα, διερευνήθηκαν τεχνικές για τη συγκριτική αξιολόγηση των τεχνικών χαρακτηριστικών των επιταχυντών, συμπεριλαμβανομένων πτυχών όπως η σύγκριση μεταξύ αρχιτεκτονικών, η ευαισθησία κατά την εφαρμογή του πυρήνα, τα προβλήματα καθυστέρησης και απόδοσης, ο ασύγχρονος χειρισμός του κόστους προγραμματισμού και τα καθεστώτα κορεσμού εύρους ζώνης και φόρτου εργασίας. Σε ετερογενείς αρχιτεκτονικές, οι σημεία αναφοράς πρέπει να καταγράφουν περισσότερα από ένα είδη αποτυχιών.

Συνεπώς, γίνεται αναφορά σε μεθόδους και τεχνικές σύγκρισης προκειμένου να δημιουργηθεί ένας πιο ακριβής και ισχυρός αναφορικός κώδικας για την αξιολόγηση των τρεχουσών και μελλοντικών συστημάτων. Αυτές οι βασικές μεθόδοι και τεχνικές περιλαμβάνουν την χρήση παραμετροποίησης πυρήνα για την αυτόματη επιλογή και διαμόρφωση του πυρήνα καθώς και την αυτόματη ανάλυση της κλιμάκωσης προκειμένου να εντοπίσουμε καταστάσεις λειτουργίας που εξασφαλίζουν επαρκή και ακριβή δειγματοληψία.

Επιπλέον, γίνεται έρευνα στις πρακτικές μεθόδους της διαχρονισμού και προσδιορίζεται η υποεκτίμηση της απόδοσης με τη χρήση κοινών χρονομέτρων γεγονότων υλικού όταν πραγματοποιούνται συγκριτικές αξιολογήσεις σε διάφορες ετερογενείς πλατφόρμες. Ο διαχρονισμός (profiling) είναι η διαδικασία μέτρησης της απόδοσης ενός προγράμματος ή ενός υπολογιστικού συστήματος, ενώ ο όρος "κοινά χρονόμετρα γεγονότα υλικού" αναφέρεται στη χρήση κοινών σημείων αναφοράς για τη μέτρηση της απόδοσης σε διάφορες ετερογενείς πλατφόρμες. Παρακάτω παρουσιάζονται πρακτικές μεθόδους για τη διαχρονισμό και την αποτροπή της υποεκτίμησης της απόδοσης με τη χρήση κοινών χρονομέτρων γεγονότων υλικού κατά τη διάρκεια συγκριτικών αξιολογήσεων:

Χρήση Ενσωματωμένων Χρονομέτρων: Οι περισσότερες σύγχρονες υπολογιστικές πλατφόρμες διαθέτουν ενσωματωμένους αισθητήρες και χρονομετρητές που μπορούν να χρησιμοποιηθούν για τη μέτρηση της απόδοσης. Οι περισσότεροι επεξεργαστές και γραφικές κάρτες προσφέρουν δυνατότητες που επιτρέπουν τη μέτρηση του χρόνου που απαιτείται για την εκτέλεση διαφορετικών εργασιών.

Χρήση Βιβλιοθηκών Απόδοσης: Ορισμένες πλατφόρμες παρέχουν βιβλιοθήκες που επιτρέπουν τη μέτρηση της απόδοσης. Για παράδειγμα, η NVIDIA παρέχει τη βιβλιοθήκη CUDA Profiling Tools Interface (CUPTI) για τη μέτρηση της απόδοσης σε GPU.

Χρήση Προηγμένων Εργαλείων Προφίλινγκ: Υπάρχουν πολλά εργαλεία προφίλινγκ (profiling tools) που μπορούν να χρησιμοποιηθούν για τη μέτρηση της απόδοσης σε ετερογενείς πλατφόρμες. Παραδείγματα περιλαμβάνουν το Intel VTune, το NVIDIA Nsight, και το AMD ROCm Profiler.

Καλή Σχεδίαση Εξόδου: Η εξόδος από τα χρονομετρημένα γεγονότα πρέπει να είναι αξιόπιστη και εκτελεστική. Αυτό μπορεί να επιτευχθεί με την εγγραφή των αποτελεσμάτων σε αρχεία και τη χρήση κατάλληλων μηχανισμών εξαγωγής δεδομένων.

Επαναλαμβανόμενες Μετρήσεις: Είναι σημαντικό να πραγματοποιούνται επαναλαμβανόμενες μετρήσεις για να εξασφαλιστεί η αξιοπιστία των αποτελεσμάτων. Οι μέσοι όροι μπορούν να χρησιμοποιηθούν για να μειωθεί η επίδραση τυχαίων παραγόντων.

Η σωστή μέτρηση της απόδοσης με τη χρήση κοινών χρονομέτρων γεγονότων υλικού είναι κρίσιμη για την αξιολόγηση της απόδοσης σε ετερογενείς πλατφόρμες και την αποφυγή της υποεκτίμησης της απόδοσης.

Καθιστούμε σαφή τη χρησιμοποίηση του αναπτυγμένου κώδικα αναφοράς με τον τρόπο που τον εφαρμόζουμε σε πυρήνες που είναι συγκρίσιμοι με τους πυρήνες που χρησιμοποιούνται στα πρότυπα αναφοράς του SHOC Level-1 και του 2-D stencil.

Αυτή η πρακτική μας επιτρέπει να προβούμε σε άμεση σύγκριση χρησιμοποιώντας τους δείκτες αναφοράς του SHOC ως βάση σύγκρισης.

Το σημείο αναφοράς μείωσης εκτελεί μια απλή λειτουργία: παίρνει ένα μονοδιάστατο διάνυσμα δεδομένων και υπολογίζει το σύνολο των τιμών αυτού του διανύσματος. Παρόλο που αυτός ο αλγόριθμος φαίνεται απλός, η επίτευξη υψηλής απόδοσης σε ποικίλες αρχιτεκτονικές είναι σημαντική. Επιπλέον, εξαιτίας του γεγονότος ότι το διάνυσμα διαβάζεται μόνο μια φορά, η απόδοση του αλγορίθμου συχνά περιορίζεται από την ταχύτητα της διασύνδεσης και το εύρος ζώνης σε πολλές αρχιτεκτονικές.. Συνήθως, οι λειτουργίες μείωσης επιτυγχάνουν τη μέγιστη απόδοση στις μονάδες επεξεργασίας γραφικών (GPU) όταν τα δεδομένα έχουν ήδη μεταφερθεί και είναι διαθέσιμα στη συσκευή χωρίς ανάγκη για αντίγραφα από την κύρια μνήμη.

Οι διαφορετικές σειρές εκτέλεσης των λειτουργιών μπορεί να προκαλέσουν σφάλματα στο αποτέλεσμα. Για παράδειγμα, με τη χρήση λειτουργιών χαμηλής κινητής υποδιαστολής, όπου προσθέτουμε πολύ μικρούς αριθμούς σε πολύ μεγάλους αριθμούς, μπορεί να φτάσουμε στα όρια της ακρίβειας που υποστηρίζει το υλικό.

Ο πυρήνας στένσιλ 2-D χρησιμοποιεί ένα είδος λειτουργίας στένσιλ με 9 σημεία, το οποίο είναι συμμετρικό, για τον υπολογισμό ενός ενδιάμεσου 2-D φιλτραρισμένου αποτελέσματος. Σε αντίθεση με τον πυρήνα μείωσης, κάθε τιμή στον εισαγόμενο 2-D πίνακα μπορεί να χρησιμοποιηθεί πολλές φορές, μέχρι και εννέα φορές, λόγω της συμμετρίας του πυρήνα. Κάθε ενημέρωση περιλαμβάνει 11 λειτουργίες χαμηλής κινητής υποδιαστολής για κάθε σημείο του πυρήνα στένσιλ. Οι λειτουργίες επαναχρησιμοποιούν τα δεδομένα και προσθέτουν επιπλέον υπολογισμούς, βελτιώντας έτσι τη σχετική απόδοση σε σύγκριση με τον πυρήνα μείωσης. Ο πυρήνας στένσιλ 2-D έχει τις δικές του βελτιστοποιήσεις, οι οποίες είναι ειδικά σχεδιασμένες για να αξιοποιούν αποτελεσματικά τα χαρακτηριστικά των αλγορίθμων που χρησιμοποιούνται σε αυτόν.

3.1 Βελτίωση του συνόλου δοκιμών OpenDwarfs

Το σύνολο δοκιμών OpenDwarfs περιλαμβάνει μια ποικιλία κώδικα OpenCL, ταξινομημένους ανά μοτίβα υπολογισμού και επικοινωνίας που είναι γνωστά ως τα 13 Berkeley Dwarfs.[1] Αρχικά, επικεντρωθήκαμε στη συλλογή δοκιμών που αντιπροσωπεύουν επιστημονικές εφαρμογές. Κάναμε μια λεπτομερή ανάλυση της ποικιλίας αυτών των δοκιμών προκειμένου να δικαιολογήσουμε την προσθήκη κάθε μιας στη συλλογή. Σκοπός μας είναι να επεκτείνουμε αυτές τις προσπάθειες, έτσι ώστε να καταφέρουμε να έχουμε μια ολοκληρωμένη αντιπροσώπευση της κάθε "dwarf". Αυτό μπορεί να επιτευχθεί είτε με την ενσωμάτωση άλλων συνόλων δοκιμών είτε με την προσθήκη προσαρμοσμένων πυρήνων.

ΚΕΦΑΛΑΙΟ 4

Συναφείς Εργασίες

Υπάρχουν πολλές διαθέσιμες σουίτες για συγκριτική αξιολόγηση, συμπεριλαμβανομένων των OpenDwarfs, SHOC, Rodinia, Graph500, LINPACK και πολλές άλλες. Αυτές οι σουίτες διαμορφώνονται συνεχώς, προσαρμόζοντας τον εαυτό τους στις σύγχρονες ανάγκες των υπολογιστικών εφαρμογών. Ωστόσο, το πρόσφατο ενδιαφέρον έχει στραφεί στη δημιουργία μιας γενικής, υψηλής ποιότητας, φορητής σουίτας συγκριτικής αξιολόγησης απόδοσης που μπορεί να προσαρμοστεί σε ποικίλες σύγχρονες και αναδυόμενες πλατφόρμες υπολογισμού. Αυτή η σουίτα αναφοράς έχει σχεδιαστεί για να αντικατοπτρίζει καλύτερα τις απαιτήσεις των σύγχρονων εφαρμογών που εκτελούνται σε διάφορες πλατφόρμες.

Για αυτή την εργασία, επιδιώξαμε να χρησιμοποιήσουμε μια σουίτα αναφοράς ως βάση για τη δημιουργία μιας ετερογενούς σουίτας που θα μπορεί να βελτιστοποιηθεί για οποιαδήποτε διαθέσιμη πλατφόρμα. Προηγούμενες έρευνες επί του θέματος, όπως αυτή που πραγματοποίησαν οι Du και συνεργάτες12, εστίασαν στον αυτοτροφοδοτούμενο κώδικα OpenCL, με σκοπό να βελτιώσουν την απόδοση του σε διάφορες αρχιτεκτονικές.

Ένα παραδειγματικό παράδειγμα έρευνας παρέχεται από τη μελέτη των Fang και συνεργατών13, όπου συγκρίνεται η απόδοση των πυρήνων CUDA με αυτήν των πυρήνων OpenCL. Οι εν λόγω έρευνες επικεντρώθηκαν κυρίως στις μονάδες επεξεργασίας γραφικών (GPU) και στην αυτόματη προσαρμογή των πυρήνων σε διάφορες GPU, χρησιμοποιώντας την τεχνολογία CUDA. Οι ερευνητές εκμεταλλεύτηκαν την αυτόματη προσαρμογή για τη βελτίωση της απόδοσης, παρατηρώντας ότι η αυτόματη προσαρμογή συμβάλλει στην αύξηση της απόδοσης σε μια μονάδα επεξεργασίας γραφικών (GPU). Ωστόσο, λόγω των περιορισμών που επιβάλλονται από το CUDA, δεν ήταν δυνατή η σύγκριση απόδοσης μεταξύ διαφορετικών αρχιτεκτονικών, όπως προσπαθεί να πραγματοποιήσει η τρέχουσα έρευνα.

Ο στόχος είναι να αποκτήσουμε μια μέθοδο αυτόματης ρύθμισης και να την εφαρμόσουμε ομοιόμορφα σε όλες τις διαθέσιμες πλατφόρμες, συμπεριλαμβανομένων των Xeon Phi και CPU, χρησιμοποιώντας την ίδια βάση κώδικα. Αυτή η προσέγγιση μειώνει τον χρόνο και την προσπάθεια που απαιτείται για τη διατήρηση του λογισμικού και επιτρέπει τη βέλτιστη επιλογή του πυρήνα.

Προηγούμενες εργασίες έχουν πραγματοποιήσει συγκρίσεις αποτελεσμάτων σε διάφορες αρχιτεκτονικές, παρόμοιες με αυτές που περιγράφονται εδώ, χρησιμοποιώντας τη γλώσσα προγραμματισμού OpenCL. Η διαφορά σε αυτή την έρευνα είναι ότι δεν αξιοποιήθηκε η μέθοδος αυτόματης ρύθμισης. Συγκεκριμένα, ο

ίδιος πυρήνας εκτελέστηκε με τις ίδιες παραμέτρους σε κάθε διαφορετική αρχιτεκτονική, και τα αποτελέσματα αξιολογήθηκαν σε σχέση με αυτόν τον σταθερό πυρήνα.

Έχουν διενεργηθεί ερευνητικές εργασίες που ασχολούνται με τον αυτόματο συντονισμό της απόδοσης, όπου χρησιμοποιούνται τεχνικές αυτόματου συντονισμού για τη βελτιστοποίηση εφαρμογών σε μια συγκεκριμένη κατηγορία ([15,16]). Η έρευνα που παρουσίασαν οι Abu-Sufah και Karim16 ανακοίνωσε μια εναλλακτική προσέγγιση στον αυτόματο συντονισμό, χρησιμοποιώντας δεδομένα εκπαίδευσης για την αναγνώριση του τύπου μητρών που προσδιορίζονται ως είσοδοι. Έπειτα, ορίσανε παραμέτρους χρόνου εκτέλεσης και προσδιόρισαν την επεξεργασία των δεδομένων με βάση αυτή την ταξινόμηση των μητρών. Οι ερευνητές χρησιμοποίησαν τα δεδομένα εκπαίδευσης για να αναπτύξουν μια σειρά βημάτων που εξασφαλίζουν την βέλτιστη απόδοση ανάλογα με τον τύπο της μήτρας που δίνεται ως είσοδος.

Σε σχέση με τις εργασίες των Lin et al. ([17]), εξέτασαν τον συντονισμό ενός πυρήνα γενικού πολλαπλασιασμού πινάκων (GEMM) που χρησιμοποιεί την πλατφόρμα CUDA σε μια GPU. Σε αντίθεση με αυτήν την προσέγγιση, η εργασία μας εφάρμοσε τεχνικές αυτόματης ρύθμισης σε διάφορες πλατφόρμες χρησιμοποιώντας το OpenCL, με διαφορετικές ανησυχίες που προκύπτουν από τη χρήση αυτής της τεχνολογίας. Επιπλέον, εστιάζουμε σε διαφορετικούς πυρήνες, και έχουμε αναπτύξει μια μεθοδολογία για την αυτόματη παραμετροποίηση αυτών των πυρήνων.

Σε σχέση με την έρευνα που παρουσίασαν οι Phothilimthana et al. ([18]), επικεντρωθήκαμε στον συντονισμό διαφορετικού τύπου αλγορίθμων, χρησιμοποιώντας το OpenCL και ασχολούμαστε με τη φορητότητα της απόδοσης σε πολλαπλές πλατφόρμες. Αντίθετα με την εργασία των Pennycook et al. ([19]), όπου εξέτασαν την επίδραση του μεγέθους ομάδας εργασίας στα αποτελέσματα, εμείς δημιουργούμε μια μεθοδολογία που επιτρέπει στο λογισμικό να αυτόματα προσδιορίζει τις βέλτιστες παραμέτρους, συμπεριλαμβανομένου του μεγέθους της ομάδας εργασίας, με βάση τον χρήσιμο πυρήνα που χρησιμοποιείται.

4.1 Παραδείγματα εξερευνητικού αυτοματισμού

Έχουν αναπτυχθεί αρκετά πλαίσια εξερευνητικού αυτοματισμού εκτός σύνδεσης για τη δημιουργία αποδοτικών, φορητών βιβλιοθηκών σε συγκεκριμένους τομείς. Το ATLAS [27] χρησιμοποιεί τον εξερευνητικό αυτοματισμό για τη δημιουργία πολυκατοικίας πίνακα πολλαπλασιασμού σε cache που χρησιμοποιείται στις μεγαλύτερες πράξεις μεταξύ πινάκων στα BLAS και LAPACK. Το FFTW χρησιμοποιεί τον εξερευνητικό αυτοματισμό για να συνδυάσει λύτες για τις μετασχηματισμούς Fourier (FFT). Άλλα συστήματα αυτοματισμού περιλαμβάνουν το SPARSITY για υπολογισμούς πυκνότητας πίνακα, το SPIRAL για την επεξεργασία

ψηφιακών σημάτων και το OSKI για πυκνούς πυρήνες πινάκων. Ο τομέας του επαναληπτικού συλλογισμού περιλαμβάνει πολλά έργα που χρησιμοποιούν διάφορες τεχνικές μηχανικής μάθησης για τη βελτιστοποίηση των χαμηλότερου επιπέδου βελτιστοποιήσεων των μεταγλωττιστών. Αυτά τα έργα αλλάζουν τη σειρά εφαρμογής των περάσεων των μεταγλωττιστών και τους τύπους των περατών που εφαρμόζονται. Ωστόσο, δεν εξερευνούν τον τύπο των αλγοριθμικών επιλογών που παρουσιάζει η γλώσσα PetaBricks, και αυτά τα συστήματα δημιουργούν μόνο στιγμαίες, όχι χρονικές επιλογές. Οι τεχνικές αυτοματισμού χρησιμοποιούνται ακόμα περισσότερο κατά τη βελτιστοποίηση προγραμμάτων GPGPU. Ο αυτοματισμός συνήθως εφαρμόζεται με τρόπο προσαρμοσμένο στο πρόβλημα ή στον τομέα του προβλήματος. Τέτοια συστήματα χρησιμοποιούν αυτοματιστές για τη δημιουργία πολυ-αλγορίθμων για την επίλυση μεγάλων συστημάτων τριδιαγωνικών πινάκων [9], για τη βελτιστοποίηση αλγορίθμων ταξινόμησης GPU, για τη βελτιστοποίηση 3D FFT με εστίαση στο padding και τη βελτιστοποίηση της εύρους ζώνης [21], για τη βελτιστοποίηση του πολλαπλασιασμού πυκνού πίνακα-διανυσμάτων με τη δημιουργία μοντέλων απόδοσης [8], και για τη βελτιστοποίηση πυκνών γραμμικών αλγεβρικών μείγματα μοντέλων [25]. Συχνά, αυτές οι τεχνικές είναι προσαρμοσμένες για ένα συγκεκριμένο πρόβλημα ή μια ομάδα προβλημάτων.

Εκτός από τεχνικές που είναι συγκεκριμένες για συγκεκριμένα προβλήματα, υπάρχει και υψηλού επιπέδου προγραμματισμός με κατευθυντήριες οδηγίες για την εκμετάλλευση της επεξεργασίας σε μονάδες GPU που χρησιμοποιεί το HMPP Workbench για την αυτόματη δημιουργία κώδικα CUDA/OpenCL και την αυτόματη προσαρμογή στον χώρο βελτιστοποίησης για τους δημιουργημένους πυρήνες GPU [13]. Ωστόσο, αυτή η τεχνική και οι προηγούμενες αναφέρουν απλώς την ανάγκη για αυτόνομη βελτιστοποίηση για την επίτευξη της καλύτερης απόδοσης σε σύγχρονες μονάδες GPU. Δεν επιδιώκουν την βέλτιστη αξιοποίηση όλων των διαθέσιμων πόρων ταυτόχρονα για την επίτευξη της μέγιστης απόδοσης σε ένα ομοιόμορφο σύστημα.

Έχουν μελετηθεί αρκετές μέθοδοι για την αποτελεσματική κατανομή του φόρτου εργασίας μεταξύ διάφορων συσκευών. Το StarPU χρησιμοποιεί ένα σύστημα εργασίας work-stealing για να εξισορροπήσει το φορτίο εργασίας μεταξύ διαφορετικών τμημάτων του συστήματος. Ωστόσο, για να χρησιμοποιήσει το StarPU, ο προγραμματιστής πρέπει να γράψει ξεχωριστό κώδικα για την CPU και την GPU και να βασιστεί σε δυναμικές διαδικασίες εξαργύρωσης που οδηγούνται από αυτόματες υποδείξεις για τη διανομή των εργασιών. Αντίθετα, το CnC-HC δημιουργεί αυτόματα κώδικα για CPU, GPU και FPGA και χρησιμοποιεί ένα σύστημα εξαργύρωσης εργασίας για τη διανομή των εργασιών μεταξύ διαφορετικών συσκευών, με επικεφαλής τις χειροκατευθύνσεις. Το Qilin δημιουργεί αυτόματα κώδικα και χρησιμοποιεί προσαρμογή της διανομής του φόρτου εργασίας για τη βελτιστοποίηση της απόδοσης [20]. Κατά τη διάρκεια της φάσης εκπαίδευσης, το Qilin εκτελεί το πρόγραμμα με

διάφορα μεγέθη εισόδου σε διαφορετικές CPU και GPU συσκευές και χρησιμοποιεί τα αποτελέσματα για να προσδιορίσει πώς θα κατανέμει το φορτίο εργασίας μεταξύ των CPU και GPU. Ωστόσο, η αναπαράσταση αυτών των συστημάτων μπορεί να μην είναι ιδανική. Το σύστημά μας αυτοματοποιεί ολόκληρη τη διαδικασία, μεταφράζοντας τους πυρήνες σε διάφορες στόχους και αναγνωρίζοντας αντικειμενικά πού πρέπει να εκτελείται ο κώδικας στο υβριδικό μας σύστημα εργασίας.. Για πραγματικές εφαρμογές, η ιδανική απεικόνιση είναι παγκοσμίως μη γραμμικά αλληλοεξαρτώμενη από όλες τις άλλες επιλογές, και η παγκόσμια βελτιστοποίηση μας αποτυπώνει αυτό κατά την εξαργύρωση. Τα αποτελέσματά μας υπογραμμίζουν τη σημαντικότητα της διασυνοριακής μάθησης.

Η CGCM [15] χρησιμοποιεί μια τεχνική για την αυτόματη διαχείριση της επικοινωνίας μνήμης μεταξύ GPU/CPU. Η τεχνική τους χρησιμοποιεί μια συντηρητική ανάλυση προσβασιμότητας για να καθορίσει πού θα εισαχθούν κλήσεις σε ένα περιβάλλον εκτέλεσης που παρακολουθεί και απεικονίζει δυναμικά δεδομένα σε διάφορες μνήμες. Ακολουθούν μια σειρά βελτιστοποιήσεων που αποσκοπούν στη μείωση του χρόνου εκτέλεσης. Η CGCM αναλαμβάνει τη μετακίνηση δεδομένων αυτόματα, αλλά απαιτεί συνεργασία από τον προγραμματιστή για τον παραλληλισμό του κώδικα.

Πολλές άλλες γλώσσες προγραμματισμού προσπαθούν να ευκολύνουν τον προγραμματισμό για συσκευές GPGPU. Το CUDA-lite αυτοματοποιεί ορισμένες εργασίες παραλληλοποίησης και διαχείρισης μνήμης στο CUDA. Το JCUDA απλοποιεί τη χρήση της GPU από κώδικα Java. Υπάρχουν επίσης προσπάθειες να αντιστοιχιστούν υποσύνολα της γλώσσας C στη GPU. Αυτές οι τεχνικές επιβάλλουν περιορισμούς στην πρόσβαση δεδομένων στο πρόγραμμα. Υπήρξαν επίσης προσπάθειες για την αντιστοίχιση του OpenMP στη GPU. [17]. Ενώ αυτές οι προσπάθειες διευκολύνουν την εκτέλεση κώδικα στη GPU, παράγουν τον ίδιο κώδικα για κάθε ετερογενές σύστημα και δεν επιτρέπουν την επιλογή αλγορίθμων ή την εμπορική αξιολόγηση της καλύτερης αντιστοίχισης για κάθε μηχάνημα.

4.2 Αξιολόγηση Συστημάτων Υψηλής Απόδοσης

Τα δοκιμαστικά προτυπα NAS parallel [2] ακολουθούν μια "μολύβι και χαρτί" προσέγγιση, καθορίζοντας τον υπολογιστικό προβληματισμό, αλλά αφήνοντας τις επιλογές υλοποίησης, όπως η γλώσσα, οι δομές δεδομένων και οι αλγόριθμοι, στον χρήστη. Τα δοκιμαστικά πρότυπα περιλαμβάνουν ποικίλους πυρήνες και εφαρμογές που επιτρέπουν μια λεπτομερή αξιολόγηση του συστήματος HPC. Παρόλα αυτά, η μη περιορισμένη προσέγγιση δεν επιτρέπει απευθείας τη σύγκριση της απόδοσης μεταξύ διαφορετικών υλικών επιταχυντών χρησιμοποιώντας ένα μοναδικό σύνολο κώδικα.

Οι Martineau και οι συνεργάτες τους συγκέντρωσαν ένα σύνολο δοκιμών και τρεις μικρές εφαρμογές για να αξιολογήσουν πώς το Clang OpenMP 4.5 υποστηρίζει τις Nvidia GPUs. Η έρευνα τους επικεντρώθηκε στη σύγκριση με το CUDA, αγνοώντας το OpenCL. Αντίθετα, το σύνολο δοκιμών Scalable Heterogeneous Computing (SHOC) υποστηρίζει πολλαπλούς κόμβους χρησιμοποιώντας MPI για κατανεμημένο παραλληλισμό. Το SHOC υποστηρίζει πολλαπλά μοντέλα προγραμματισμού, συμπεριλαμβανομένων του OpenCL, CUDA και OpenACC, και δοκιμές που καλύπτουν εύρος από χαμηλό επίπεδο χαρακτηριστικών υλικού έως πυρήνες εφαρμογών. Ο Sun και οι συνεργάτες του παρουσιάζουν το Hetero-Mark, ένα σύνολο δοκιμών για τον συνεργατικό υπολογισμό CPU-GPU, το οποίο περιλαμβάνει πέντε εφαρμογές δοκιμής, υλοποιημένες σε HCC, HIP και CUDA. Τέλος, το Chai από τον Gómez-Luna και τους συνεργάτες του προσφέρει 15 εφαρμογές σε 7 διαφορετικές υλοποιήσεις, με έμφαση στην υποστήριξη ολοκληρωμένων αρχιτεκτονικών.

Αυτά τα σύνολα δοκιμών επικεντρώνονται στη σύγκριση μεταξύ γλωσσών και περιβαλλόντων, ενώ η εργασία μας επικεντρώνεται στις δοκιμές απόδοσης που είναι ειδικές για τη συσκευή, εξετάζοντας, για παράδειγμα, τα μεγέθη προβλημάτων όπου συναντώνται αυτοί οι περιορισμοί. Η προσέγγιση αυτή παραβλέπεται σε μεγάλο βαθμό από τα σύνολα δοκιμών που χρησιμοποιούν σταθερά μεγέθη προβλημάτων. Επιπλέον, το ενισχυμένο σύνολο δοκιμών OpenDwarfs στοχεύει στο να καλύψει ένα ευρύ φάσμα μοτίβων εφαρμογής, επικεντρώνοντας αποκλειστικά στο OpenCL. Ο Barnes και η ομάδα του συγκέντρωσαν ένα εκτεταμένο σύνολο εφαρμογών από το τρέχον φορτίο εργασίας του NERSC, προκειμένου να καθοδηγήσουν τη βελτιστοποίηση για την πλατφόρμα Knights Landing στον υπερυπολογιστή Cori. Δεδομένου ότι μια λεπτομερής μελέτη απόδοσης για διάφορες υπολογιστικές συσκευές δεν είναι πάντα εφικτή, τα αποτελέσματα που παρουσιάζονται σε αυτό το άρθρο μπορούν να δώσουν μια γενική κατανόηση της απόδοσης και των περιορισμών των διαφόρων συσκευών.

ΚΕΦΑΛΑΙΟ 5

Μέθοδος

5.1 Παραμετροποίηση πυρήνα και αυτόματη βελτιστοποίηση

Η σημασία των βελτιστοποιήσεων στη συγκριτική αξιολόγηση είναι ευρέως αποδεκτή και θεωρείται κρίσιμη, καθώς συχνά επηρεάζει τις συμπεράσματα σχετικά με την απόδοση διαφορετικών αρχιτεκτονικών. Αυτή η σημασία της βελτιστοποίησης είναι ιδιαίτερα έντονη όταν αναφερόμαστε σε υπολογιστικές πλατφόρμες με επιταχυντές, διότι εκεί η τεχνολογία είναι λιγότερο ανεπτυγμένη και είναι περισσότερο ευαίσθητη στις βελτιστοποιήσεις στον κώδικα πηγής και στις προσαρμοσμένες παραμέτρους χρόνου εκτέλεσης.

Η ευαισθησία στη βελτιστοποίηση στον κώδικα πηγής μπορεί επίσης να παραπλανήσει ως προς τη σχετική απόδοση διαφόρων API που χρησιμοποιούνται για την αντιστοίχιση στους επιταχυντές. Ένα μη φορητό API, όπως το CUDA ή τα Intel offload pragma, είναι πιθανό να χρησιμοποιηθεί προς το καλύτερο στόχο, λαμβάνοντας υπόψη τις λεπτομέρειες της αρχιτεκτονικής του συγκεκριμένου προμηθευτή. Αυτή η βελτίωση, ωστόσο, συνήθως θυσιάζει την φορητότητα, καθώς ο κώδικας πηγής πρέπει να προσαρμοστεί ειδικά για κάθε αρχιτεκτονική.

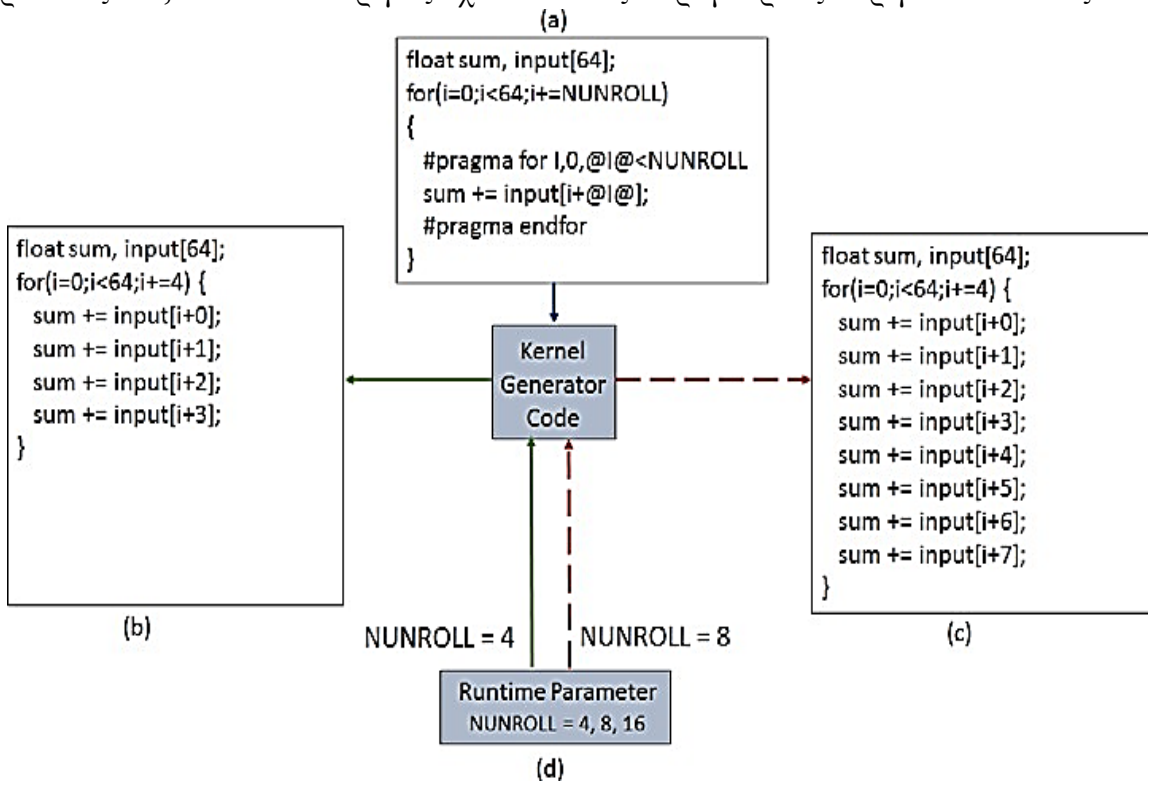
Αντίθετα, το OpenCL λειτουργεί ως ένα φορητό API που συχνά καθοδηγεί τους προγραμματιστές να αναπτύσσουν εφαρμογές με γενικότερη φύση, αποφεύγοντας προσαρμοσμένες βελτιστοποιήσεις για συγκεκριμένους στόχους. Το OpenCL είναι φορητό, ενώ η απόδοση εξαρτάται από την κάθε αρχιτεκτονική. Για παράδειγμα, ο δείκτης αναφοράς SHOC παρέχει διάφορες υλοποιήσεις διαφόρων δοκιμαστικών πυρήνων που εκτελούνται με CUDA, OpenCL και Intel offload pragma. Αναμενόμενα, οι φορητές υλοποιήσεις OpenCL εμφανίζουν βελτιωμένη απόδοση σε σχέση με τις εκδόσεις που χρησιμοποιούν ειδικά API προς όφελος συγκεκριμένης αρχιτεκτονικής. Αυτό προκύπτει από το γεγονός ότι τα API για συγκεκριμένους προμηθευτές έχουν πλεονεκτήματα στην προσαρμογή σε συγκεκριμένες αρχιτεκτονικές, ενώ το OpenCL είναι φορητό αλλά προσφέρει λιγότερο απόδοση. Αυτή η διαφορά σε απόδοση μπορεί να οφείλεται στην προσαρμογή των ειδικών API σε συγκεκριμένες αρχιτεκτονικές, στην υψηλή ποιότητα των υλοποιήσεών τους ή στη σύγκριση μεταξύ γενικών και βελτιστοποιημένων υλοποιήσεων πυρήνων κατά την αξιολόγηση σε διαφορετικές αρχιτεκτονικές.

Η υπάρχουσα κατάσταση θέτει ένα πιο γενικό πρόβλημα όσον αφορά την ανάπτυξη φορητών σημείων αναφοράς, τα οποία προσφέρουν ουσιαστικά αποτελέσματα που μπορούν να χρησιμοποιηθούν για δίκαιες συγκρίσεις μεταξύ διαφορετικών αρχιτεκτονικών. Για να χρησιμοποιήσει ένας κώδικας αναφοράς ένα φορητό API όπως το OpenCL, η βελτιστοποίηση πρέπει να ενσωματωθεί στη μεθοδολογία αξιολόγησης με τρόπο που δεν απαιτεί τον χειροκίνητο συντονισμό για συγκεκριμένες πλατφόρμες. Αυτό προστίθεται μεγάλη αβεβαιότητα και μπορεί να οδηγήσει σε μεγάλες αποκλίσεις στις αξιολογήσεις.

Σε αυτήν την έρευνα, η δημιουργία πηγαίου κώδικα γίνεται με την παραμετροποίηση των πυρήνων. Κατασκευάζουμε πολλούς ειδικούς πυρήνες, χρησιμοποιώντας έναν σύνθετο προ-επεξεργαστικό κώδικα στη γλώσσα C, ο οποίος δημιουργεί αυτούς τους πυρήνες με βάση συγκεκριμένες παραμέτρους πριν από τη μετάφραση σε γλώσσα που κατανοεί ο επεξεργαστής της πλατφόρμας. Οι παράμετροι που εισάγονται μπορεί να είναι λογικές, όπως η χρήση τοπικής μνήμης για την προσωρινή αποθήκευση δεδομένων, καθώς και αριθμητικές, όπως ο αριθμός των ενημερώσεων ανά κλήση του πυρήνα ή ο αριθμός των επαναλήψεων για την εκτέλεση ενός βρόχου.

Στο πλαίσιο της Εικόνας 1, απεικονίζεται μια απλοποιημένη διαδικασία δημιουργίας κώδικα. Ο προ-επεξεργαστικός κώδικας C (Τμήμα a) παράγει τους ειδικούς πυρήνες, λαμβάνοντας υπόψη τις διάφορες παραμέτρους χρόνου εκτέλεσης (Τμήμα d). Το αποτέλεσμα είναι οι βελτιστοποιημένοι πυρήνες που παράγουν αντιπροσωπευτικά αποτελέσματα αναφοράς, ιδανικά για συγκρίσεις, καθώς και πολλαπλές δοκιμές παραμέτρων για την ανάδειξη χαρακτηριστικών της πλατφόρμας.

Αυτή η περιγραφή αφορά την περίπτωση με μία είσοδο, αλλά στην έρευνα που παρουσιάζεται, οι ειδικοί πυρήνες έχουν πολλές παραμέτρους πυρήνα που αλλάζουν.



Εικ. 1 Διάγραμμα της γεννήτριας κώδικα

Αποφασίσαμε να προσαρμόσουμε τη μείωση και τους πυρήνες στένσιλ 2-Δ δειγμάτων που βρίσκονται στα σημεία αναφοράς SHOC Level-1. Κάθε πυρήνας αντιπροσωπεύει μια συγκεκριμένη αλγοριθμική διαδικασία που ανακαλύφθηκε στην αρχική συλλογή νανοδομών που παρουσίασαν οι ερευνητές από το Πανεπιστήμιο της Καλιφόρνια, Berkeley. Αυτή η συλλογή κατάρτισε τα θεμέλια για όλες τις διάφορες σουίτες αναφοράς. Το σύνολο των νανοδομών που παρουσιάζονται εκεί αντιπροσωπεύει μια ποικιλία υπολογισμών που συνήθως χρησιμοποιούνται από την επιστημονική κοινότητα.

Επιπλέον, προσαρμόζουμε τον πυρήνα κανονικής αναγωγής και τον πυρήνα στένσιλ 2-Δ για να συμπεριλάβουμε λογικές και αριθμητικές παραμέτρους που επηρεάζουν διάφορες βελτιστοποιήσεις στην πηγή του κώδικα και είναι γνωστό ότι επηρεάζουν την απόδοση. Οι παράμετροι που συχνά χρησιμοποιούνται και στους δύο

πυρήνες περιλαμβάνουν το μήκος του διανύσματος των τύπων δεδομένων, το μέγεθος των ομάδων εργασίας, τη χρήση της τοπικής μνήμης και τον βαθμό απεμπλοκής του βρόχου που χρησιμοποιείται. Σε ό,τι αφορά τον πυρήνα μείωσης, χρησιμοποιούμε παράμετρο για την επιλογή μίας από τις έξι διαθέσιμες μετατοπίσεις δείκτη που χρησιμοποιούνται για την πρόσβαση στα στοιχεία σε διάφορες σειρές. Επιπλέον, υπάρχει μια λογική παράμετρος που καθορίζει εάν πρέπει να χρησιμοποιηθεί αύξηση του δείκτη, αντί για ευρετηρίαση του πίνακα. Όσον αφορά τον πυρήνα στένσιλ 2-Δ, χρησιμοποιούμε μια παράμετρο για να ορίσουμε τον αριθμό των γραμμών που πρέπει να ενημερωθούν ανά νήμα. Επιπλέον, υπάρχει μια λογική παράμετρος που καθορίζει εάν ο τριπλός βρόχος πρέπει να συγχωνευθεί σε έναν ενιαίο βρόχο. Η προσαρμογή αυτών των παραμέτρων με τις τυπικές τιμές για αυτές μπορεί να οδηγήσει σε πολλούς μοναδικούς πυρήνες, με δείγματα από τον κώδικα αναφοράς.

5.1.1 Δημιουργία OpenCL Kernel

Επειδή η γλώσσα PetaBricks είναι πιο γενική και υποστηρίζει την ενσωμάτωση αυθαίρετου κώδικα C/C++, μόνο ένα υποσύνολο ενός προγράμματος PetaBricks μπορεί να μετατραπεί σε πυρήνες OpenCL. Ο διαδικασία γεννήσεως επιπλέον επιλογών για εκτέλεση OpenCL πραγματοποιείται στα πρώτα στάδια της διαδικασίας μεταγλώττισης, πριν από τον προγραμματισμό της εκτέλεσης. Η διαδικασία μετατροπής περιλαμβάνει τρία στάδια που εφαρμόζονται σε κάθε αρχικό κανόνα που καθορίζεται από τον χρήστη, με στόχο την εισαγωγή ισοδύναμων συνθετικών κανόνων OpenCL όταν είναι εφικτό.

Στο πρώτο στάδιο, πραγματοποιείται ανάλυση των εξαρτήσεων για να καθορίσει αν το πρότυπο εκτέλεσης του κανόνα ταιριάζει με το πρότυπο εκτέλεσης του OpenCL. Και οι σειριακές και οι παράλληλες ακολουθίες εξαρτήσεων μπορούν να αναπαρασταθούν σε πυρήνες OpenCL. Ωστόσο, πιο πολύπλοκα παράλληλα πρότυπα, όπως το παράλληλο πρότυπο wavefront, δεν είναι εφικτό να υλοποιηθούν στην παρούσα έκδοση της υλοποίησής μας. Είναι δυνατόν να υπάρχει προτίμηση για τη χρήση κάποιων συνόλων αλγοριθμικών επιλογών ενώ άλλα όχι. Το γράφημα εξαρτήσεων αναλύεται για να διαπιστωθεί εάν ένας κανόνας μπορεί να μεταφραστεί σε κώδικα OpenCL. Κατά τη διαδικασία αυτή, εξετάζεται η κατεύθυνση των εξαρτήσεων που προκύπτουν από τον κανόνα. Εάν δεν υπάρχει εξάρτηση ή εάν η εξάρτηση εξαλείφεται με την επιλογή του συγκεκριμένου κανόνα, τότε οι εξωτερικές εξαρτήσεις του μπορούν να μεταφραστούν σε κώδικα OpenCL.

Κατά το δεύτερο στάδιο της διαδικασίας μετατροπής, αν ένας κανόνας περάσει το πρώτο στάδιο, το σώμα του μετασχηματίζεται σε κώδικα OpenCL. Αυτό το στάδιο περιλαμβάνει πολλές συντακτικές μετατροπές και την αναδιάταξη της πρόσβασης στα δεδομένα ώστε να χρησιμοποιείται η γενική μνήμη της GPU. Επίσης, ανιχνεύονται ορισμένες γλωσσικές κατασκευές, όπως κλήσεις σε εξωτερικές βιβλιοθήκες, που δεν

μπορούν να μεταφερθούν στο OpenCL. Επιπλέον, αποκλείονται ορισμένοι κανόνες από τη μετατροπή σε OpenCL εάν περιέχουν γλωσσικές κατασκευές που δεν υποστηρίζονται από το OpenCL, όπως ενσωματωμένος κώδικας native. Οι περισσότερες από αυτές τις κατασκευές ανιχνεύονται στατικά, αλλά υπάρχουν και ορισμένες λεπτές απαιτήσεις που ανακαλύπτονται κατά τη μεταγλώττιση του τελικού μετασχηματισμού, και απορρίπτονται συνθετικοί κανόνες OpenCL που δεν μπορούν να μεταγλωττιστούν.

Το τρίτο στάδιο της διαδικασίας επιδιώκει να βελτιστοποιήσει την κύρια έκδοση των κωδικών OpenCL που παράγονται από το προηγούμενο στάδιο. Χρησιμοποιεί την τοπική μνήμη της GPU, γνωστή και ως τοπική μνήμη OpenCL ή κοινόχρηστη μνήμη CUDA, για βελτιστοποιήσεις. Για ορισμένους κανόνες OpenCL, δημιουργείται μια έκδοση της τοπικής μνήμης, εστιάζοντας στο μοτίβο πρόσβασης των δεδομένων εισόδου. Ένα "πλαίσιο δέσμευσης" είναι μια ορθογώνια περιοχή ενός πίνακα εισόδου που χρησιμοποιείται για τον υπολογισμό μιας καταχώρησης του πίνακα εξόδου. Αν το μέγεθος του πλαισίου δέσμευσης είναι μεγαλύτερο από ένα, δημιουργείται μια έκδοση της τοπικής μνήμης. Αν είναι ένα, δεν απαιτείται η αντιγραφή δεδομένων στην τοπική μνήμη, καθώς τα νήματα που μοιράζονται την ίδια τοπική μνήμη δεν έχουν πρόσβαση στα ίδια δεδομένα. Η έκδοση της τοπικής μνήμης περιλαμβάνει δύο μέρη: η φόρτωση δεδομένων στην τοπική μνήμη από όλα τα νήματα εργασίας και η υποκατάσταση της πρόσβασης στη γενική μνήμη με πρόσβαση στην τοπική. Αυτή η βελτιστοποίηση παρουσιάζεται ως επιλογή από τον αυτορυθμιστή.

5.1.2 Ανάλυση Μετακίνησης Δεδομένων

Ένα δεύτερο σύνολο αναλύσεων OpenCL πραγματοποιείται κατά τον προγραμματισμό για τον προσδιορισμό του τρόπου αντιγραφής των δεδομένων μέσα και έξω από τη μνήμη της GPU. Κάθε φορά που γίνεται μια επιλογή στο PetaBricks για τον τρόπο μετασχηματισμού, ο μεταγλωττιστής δημιουργεί ένα μοναδικό πρόγραμμα προγραμματισμού. Αυτά τα προγράμματα αναπαριστούν την παραλληλία ως δέντρα εξαρτημάτων που δίνονται στο χρόνο εκτέλεσης του συστήματος εκμετάλλευσης εργασίας (workstealing runtime). Μετά τη δημιουργία κάθε προγράμματος προγραμματισμού, γίνεται ανάλυση για τον τρόπο μετακίνησης δεδομένων προς τη GPU. Αυτή η ανάλυση εξετάζει τις προηγούμενες εφαρμογές κανόνων στο πρόγραμμα προγραμματισμού και κατατάσσει τις περιοχές εξόδου που παράγονται στη GPU σε τρεις κατηγορίες:

- Οι "περιοχές αντιγραφής" αναφέρονται σε τμήματα του προγράμματος προγραμματισμού που ακολουθούν αμέσως έναν κανόνα εκτέλεσης στην CPU. Σε αυτές τις περιοχές, τα δεδομένα αντιγράφονται απευθείας.

- Οι "επαναχρησιμοποιούμενες περιοχές" αφορούν τμήματα του προγράμματος που ακολουθούν αμέσως έναν άλλο κανόνα εκτέλεσης στη GPU. Σε αυτές τις περιοχές, τα δεδομένα διατηρούνται στη μνήμη της GPU μεταξύ των εκτελέσεων κανόνων.
- Οι "περιοχές που μπορεί να απαιτηθεί αντιγραφή" αφορούν τμήματα που ακολουθούνται από δυναμική ροή ελέγχου, η οποία δεν μπορεί να αναλυθεί στατικά. Σε αυτές τις περιοχές, χρησιμοποιείται μια προσεκτική προσέγγιση για τη διαχείριση των δεδομένων, με ειδικό έλεγχο για τη διαθεσιμότητά τους στη μνήμη της CPU πριν από την ανάγκη χρήσης τους. Αν χρειαστεί, τότε γίνεται η αντιγραφή τους.

Με βάση την ανάλυση αυτή, ο μεταγλωττιστής ενσωματώνει διάφορες διαδικασίες στο πρόγραμμα προγραμματισμού, όπως προετοιμασία δεδομένων, διαχείριση αντιγραφής εισόδου, εκτέλεση, και παρακολούθηση της κατάστασης αντιγραφής.

5.1.3 Βελτίωση του συνόλου δοκιμών OpenDwarfs

Το σύνολο δοκιμών OpenDwarfs περιλαμβάνει μια ποικιλία κώδικα OpenCL, ταξινομημένους ανά μοτίβα υπολογισμού και επικοινωνίας που είναι γνωστά ως τα 13 Berkeley Dwarfs.[1] Αρχικά, επικεντρωθήκαμε στη συγκέντρωση ενδεικτικών δοκιμών για επιστημονικές εφαρμογές, με λεπτομερή ανάλυση της ποικιλίας για να δικαιολογήσουμε την περιλαμβανόμενη κάθε δοκιμή στο συγκεκριμένο σύνολο. Στόχος μας είναι να επεκτείνουμε αυτές τις προσπάθειες για να πετύχουμε μια ολοκληρωμένη κάλυψη κάθε "dwarf", είτε με την ένταξη άλλων συνόλων δοκιμών είτε με την προσθήκη εξειδικευμένων πυρήνων.

Ο Marjanović και συν. [18] υποστηρίζουν ότι η επιλογή του μεγέθους του προβλήματος για τον ελέγχο της απόδοσης υπολογιστικών υψηλών επιδόσεων επηρεάζει κρίσιμα ποιες ιδιότητες του υλικού είναι σχετικές. Παρατηρήσαμε ότι αυτό ισχύει για μια ευρεία γκάμα επιταχυντών, επομένως επανασχεδιάσαμε το σύνολο δοκιμών OpenDwarfs έτσι ώστε να υποστηρίζει την εκτέλεση διαφορετικών μεγεθών προβλημάτων για κάθε δοκιμή. Προκειμένου να εξασφαλίσουμε την αναπαραγωγιμότητα των αποτελεσμάτων, τροποποιήσαμε επίσης κάθε δοκιμή έτσι ώστε να εκτελείται εντός ενός βρόχου για τουλάχιστον δύο δευτερόλεπτα. Αυτό γίνεται προκειμένου να διασφαλίσουμε ότι η μέτρηση του χρόνου εκτέλεσης και οι δείκτες απόδοσης δεν επηρεάζονται σημαντικά από τον θόρυβο που προκαλείται από το λειτουργικό σύστημα.

Για τον "Spectral Methods dwarf", το αρχικό σύνολο δοκιμών FFT των OpenDwarfs ήταν πολύπλοκο και προκαλούσε προβλήματα όταν εκτελούνταν για το προεπιλεγμένο μέγεθος προβλήματος. Αυτό οδήγησε σε εσφαλμένα αποτελέσματα ή αποτυχίες σε ορισμένες περιπτώσεις. Αποφασίσαμε να αντικαταστήσουμε την

πολύπλοκη έκδοση με μια πιο απλή και υψηλής απόδοσης δοκιμή FFT που δημιουργήθηκε από τον Eric Bainville. Αυτή η νέα δοκιμή λειτούργησε σωστά σε όλες τις περιπτώσεις που δοκιμάσαμε. Επιπλέον, προσθέσαμε έναν 2-διάστατο μετασχηματισμό κυματοσειράς από το σύνολο δοκιμών Rodinia, με τροποποιήσεις για βελτίωση της φορητότητας. Επί του παρόντος σχεδιάζουμε να προσθέσουμε κώδικα για συνεχή μετασχηματισμό κυματοσειράς.

Για να αξιολογήσουμε την απόδοση των δοκιμών, είναι χρήσιμο να μπορούμε να μετρήσουμε διάφορες παραμέτρους απόδοσης του υλικού που σχετίζονται με κάθε τμήμα του χρονομετρημένου κώδικα. Το LibSciBench είναι ένα εργαλείο μέτρησης απόδοσης που επιτρέπει τη συλλογή υψηλής ακρίβειας γεγονότων χρονομέτρησης για στατιστική ανάλυση [12]. Προσφέρει έναν υψηλής ανάλυσης χρονόμετρο για τη μέτρηση κωδικών πυρήνων που εκτελούνται γρήγορα, αναφερόμενο με ανάλυση ενός κύκλου και περίπου 6ns χρόνο επικαλύψεων. Χρησιμοποιήσαμε το LibSciBench για να καταγράψουμε τους χρόνους εκτέλεσης, σε συνδυασμό με τα γεγονότα υλικού. Τα γεγονότα υλικού συλλέχθηκαν χρησιμοποιώντας τους μετρητές PAPI [24]. Τροποποιήθηκαν οι εφαρμογές στο σύνολο δοκιμών OpenDwarfs για να εισάγουν κλήσεις βιβλιοθήκης στο LibSciBench για να καταγράφουν τους χρόνους και τα γεγονότα PAPI για τα τρία κύρια συστατικά του χρόνου εφαρμογής: εκτέλεση πυρήνα, ρύθμιση κεντρικού υπολογιστή και λειτουργίες μεταφοράς μνήμης. Μέσω των ενοτήτων PAPI, όπως το Running Average Power Limit (RAPL) της Intel και η Nvidia Management Library (NVML), το LibSciBench επιτρέπει επίσης την καταγραφή μετρήσεων ενέργειας.

5.2 Δυναμική Δειγματοληψία και Ανάλυση Κλιμάκωσης

Οι πυρήνες εργασίας που λειτουργούν σε χαμηλούς φόρτους μπορούν να είναι ευάλωτοι στις γενικές επιβαρύνσεις της εκτέλεσης, οι οποίες μπορούν να προκαλέσουν μεγαλύτερη διακύμανση στα χρονικά αποτελέσματα. Αντίθετα, οι πυρήνες που εκτελούν δύσκολες εργασίες μπορεί να προσφέρουν σταθερότερες μετρήσεις χρόνου, καθώς ο επιπλέον φόρτος συνήθως έχει σταθερό κόστος σε σύγκριση με τον συνολικό χρόνο εκτέλεσης. Αυτό συχνά οδηγεί τους προγραμματιστές σε χρήση πυρήνων μεγάλου φόρτου για την αξιολόγηση της μέγιστης απόδοσης μιας πλατφόρμας.

Παρόλα αυτά, η σύγκριση πλατφορμών πρέπει να βασίζεται στην αξία τους για πραγματικές εφαρμογές. Το πραγματικό μέγεθος των φορτίων εργασίας εξαρτάται από το πρόβλημα που επιλύεται, και όχι από την απόλυτη δυνατότητα της πλατφόρμας. Έτσι, η σύγκριση πρέπει να καλύπτει ένα ευρύ φάσμα φορτίων εργασίας, συμπεριλαμβανομένων των περιπτώσεων όπου το υλικό δεν λειτουργεί με τη μέγιστη δυνατή ικανότητα. Για παράδειγμα, μια σύγχρονη GPU μπορεί να απαιτεί πολλά

ταυτόχρονα νήματα για να αξιοποιηθεί πλήρως, αλλά ένα πραγματικό πρόβλημα μπορεί να μην επιτρέπει τόσο υψηλό βαθμό παραλληλισμού.

Κατά τη σύγκριση των πλατφορμών σε διάφορα σενάρια, ενδέχεται να απαιτείται διαφορετικός αριθμός δειγμάτων για να ληφθούν υπόψη οι διαφορετικά επίπεδα μεταβλητότητας. Για μεγάλους φορτίους εργασίας ή πυρήνες που είναι υψηλά υπολογιστικά απαιτητικοί, ο συγχρονισμός με 10 επαναλήψεις μπορεί να αποδώσει σταθερές μετρήσεις, και ένας απλός μέσος όρος είναι αρκετός. Σε περιπτώσεις με μεγαλύτερη μεταβλητότητα, μπορεί να απαιτούνται περισσότερα δείγματα για να επιτευχθεί σύγκλιση σε μια σημαντική μέση τιμή. Για να αξιολογηθούν αντικειμενικά, εφαρμόζουμε ένα δυναμικό πρόγραμμα χρονοποίησης που συγκεντρώνει αποτελέσματα ενώ υπολογίζει τρέχουσες μέσες τιμές και τυπικές αποκλίσεις. Ο αριθμός των αποτελεσμάτων που συλλέγονται εξαρτάται από την επίτευξη ενός συγκεκριμένου επιπέδου σύγκλισης.

Θεωρητικά, αναμένουμε ότι μια αυξημένη εργασία, είτε πρόκειται για μεταφορά δεδομένων είτε για υπολογισμούς, θα ανταποκριθεί με αναλογική αύξηση του χρόνου εκτέλεσης. Στην πράξη, υπάρχουν πολλές παράμετροι και περιπτώσεις όπου αυτό δεν ισχύει. Πολυπλοκότητα της πλατφόρμας, αντίσταση σε σημεία καμπής και άλλοι παράγοντες μπορεί να οδηγήσουν σε μη γραμμική συμπεριφορά στην απόδοση. Επομένως, είναι σημαντικό να πραγματοποιηθούν δυναμικές αναλύσεις κλιμάκωσης στον ίδιο τον κώδικα αναφοράς, έτσι ώστε να χρησιμοποιείται αποτελεσματικότερα η δειγματοληψία. Για παράδειγμα, πραγματοποιείται χρονισμό μέσα σε σύντομο εσωτερικό βρόχο για να ελεγχθούν τα αποτελέσματα εκτέλεσης πυρήνων σε συνεχόμενη σειρά. Ο αριθμός των επαναλήψεων αυξάνεται μέχρις ότου παρατηρηθεί γραμμική συμπεριφορά. Συνολικά, αυτή η προσέγγιση μπορεί να απαιτεί τουλάχιστον τρεις επαναλήψεις και σε ορισμένες περιπτώσεις, ακόμα περισσότερες.

5.3 Μέθοδοι Χρονομέτρησης

Η κατάλληλη τεχνική για την σύγκριση παράλληλων υπολογιστικών συστημάτων αποτελεί ένα πολυσύνθετο θέμα που συζητείται επί σειρά ετών. Όταν πρόκειται για ετερογενείς πλατφόρμες που περιλαμβάνουν επιταχυντές, η πολυπλοκότητα αυξάνεται ακόμα περισσότερο.

Μια κρίσιμη μεθοδολογία για την εξασφάλιση αξιόπιστων αποτελεσμάτων σε αυτό το πλαίσιο είναι η χρονομέτρηση. Πολλοί επιταχυντές διαθέτουν εσωτερικά χρονόμετρα συμβάντων στο υλικό τους, τα οποία μπορούν να προσπελαστούν μέσω ενός κοινού προγραμματιστικού περιβάλλοντος όπως το OpenCL. Αυτά τα χρονόμετρα μπορούν να παρέχουν σημαντικές πληροφορίες όταν πρέπει να συγκρίνουμε δύο αλγορίθμους σε πλατφόρμες με παρόμοια υλική διάταξη και λογισμικό.

Ωστόσο, πρέπει να σημειώσουμε ότι η χρήση αυτών των χρονομέτρων για τη δημιουργία συγκριτικών δεικτών μεταξύ διαφορετικών αρχιτεκτονικών θα πρέπει να αποφεύγεται. Αυτό οφείλεται στο γεγονός ότι αυτή η πρακτική οδηγεί σε ασυνεπή αποτελέσματα και υπερεκτιμά γενικά την απόδοση. Στην πραγματικότητα, ο πιο αξιόπιστος τρόπος για την αξιολόγηση της απόδοσης σε ετερογενείς αρχιτεκτονικές είναι ο χρόνος τοίχου ρολογιού.

Εντούτοις, χρησιμοποιούμε ακόμα και τους χρονοδιακόπτες συμβάντων και τους χρονοδιακόπτες τοίχου ρολογιού για να συγκρίνουμε τα αποτελέσματα με τη σουίτα δεικτών αναφοράς SHOC. Ωστόσο, αναγνωρίζουμε ότι η χρήση των χρονομετρητών συμβάντων υπερεκτιμά την απόδοση και αναδεικνύει την ανάγκη για προσεκτική αξιολόγηση σε αυτού του είδους τις συγκρίσεις.

ΚΕΦΑΛΑΙΟ 6

Η Εξέλιξη προς Ετερογενείς Υπολογιστικές Αρχιτεκτονικές: Προκλήσεις και Προσεγγίσεις Προγραμματισμού με Έμφαση στο OpenCL

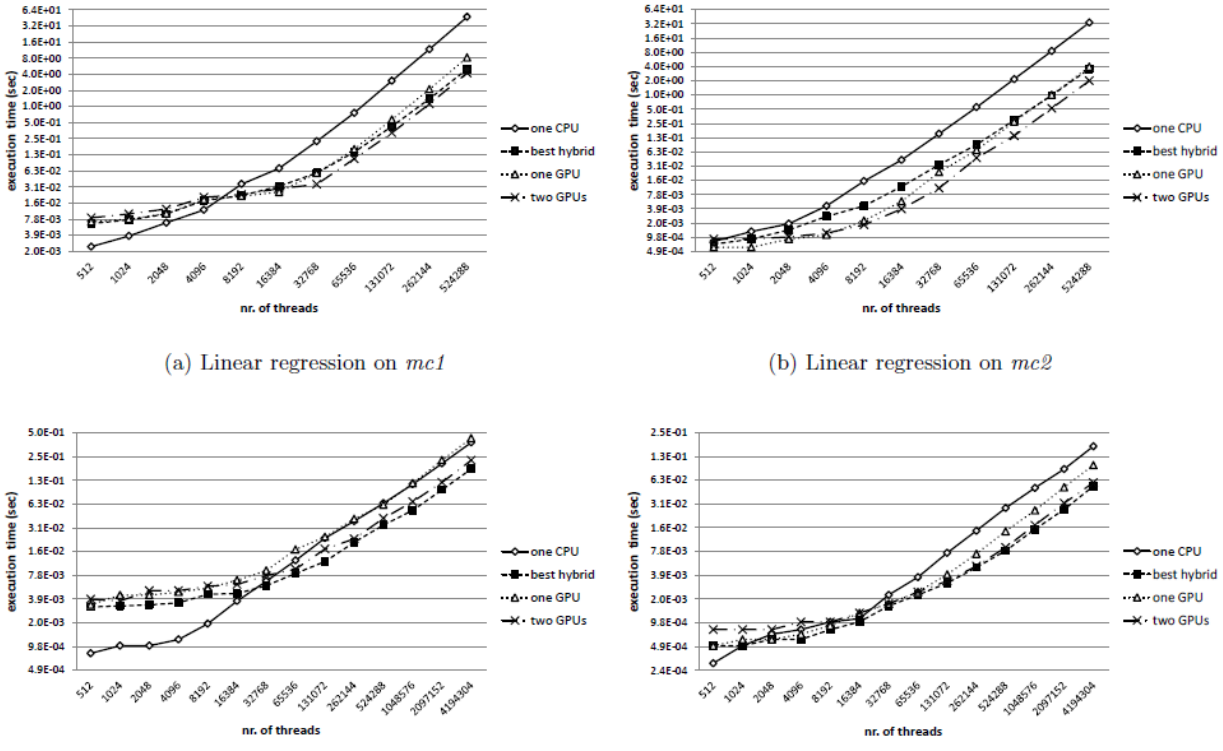
Τα τελευταία χρόνια, τα ετερογενή υπολογιστικά συστήματα έχουν εμφανιστεί ως κύρια και οικονομική λύση για την ανάπτυξη σε μεγάλη κλίμακα. Σε σύγκριση με τα συμβατικά ομοιογενή συστήματα, προσφέρουν ανώτερη απόδοση και αποτελεσματικότητα στη χρήση ενέργειας. Δεν είναι έκπληξη το γεγονός ότι τρία από τα δέκα γρηγορότερα υπερυπολογιστικά συστήματα στον κόσμο είναι ετερογενή συστήματα [5], που αποτελούνται από κόμβους με πολυπύρηνους επεξεργαστές (CPUs) και μονάδες επεξεργασίας γραφικών (GPUs). Η μετάβαση από ομοιογενή σε ετερογενή αρχιτεκτονική αποτελεί πρόκληση όσον αφορά την αποτελεσματική χρήση των υλικών πόρων και την επαναχρησιμοποίηση του σωφράγματος λογισμικού. Αυτό το πρόβλημα έχει προκαλέσει μεγάλο ενδιαφέρον τόσο από τους ερευνητές όσο και από τη βιομηχανία, οδηγώντας στην ανάπτυξη πολλών μοντέλων προγραμματισμού, όπως το HMPP, το OpenACC, το CUDA και το OpenCL. Το OpenCL (Open Computing Language) είναι ένα ανοιχτό πρότυπο για παράλληλο υπολογισμό πολλαπλών πλατφορμών, υποστηριζόμενο από πολλούς κατασκευαστές υλικού, όπως η AMD, η ARM, η IBM, η Intel και η NVIDIA. Το OpenCL παρέχει υποστήριξη για μια ευρεία γκάμα υλικού μέσω ενός υψηλής απόδοσης στρώματος αφαίρεσης χαμηλού επιπέδου, επιτρέποντας στους προγραμματιστές να αναπτύσσουν εφαρμογές χωρίς την ανάγκη για γνώση της αρχιτεκτονικής του συστήματος. Ωστόσο, η ανάπτυξη προγραμμάτων για ετερογενή συστήματα εξακολουθεί να είναι δύσκολη λόγω των διαφορών στις δυνατότητες επεξεργασίας, τη διαθεσιμότητα μνήμης και τις

καθυστερήσεις επικοινωνίας μεταξύ των υπολογιστικών πόρων (γνωστών ως συσκευές στο OpenCL).

Κίνητρο

Η πρόκληση που παρουσιάζει ο ετερογενής υπολογισμός ανοίγει νέες προοπτικές για την ανάπτυξη παράλληλων αλγορίθμων, αλλά ταυτόχρονα εισάγει επιπλέον πολυπλοκότητα. Ένα από τα κύρια προβλήματα είναι η διανομή των εργασιών μεταξύ των διαθέσιμων συσκευών OpenCL για την επίτευξη της μέγιστης δυνατής απόδοσης του συστήματος. Η διανομή των εργασιών καθορίζει τον τρόπο με τον οποίο ο συνολικός φορτίος εργασίας κατανέμεται μεταξύ πολλών υπολογιστικών πόρων.

Είναι σημαντικό να σημειωθεί ότι η βέλτιστη διανομή εργασιών μπορεί να ποικίλλει ανάλογα με την εφαρμογή, το μέγεθος του προβλήματος και τις ρυθμίσεις του υλικού. Αυτό αποδεικνύεται μέσω μιας μελέτης περίπτωσης με δύο προγράμματα που ανήκουν στις πειραματικές μας περιπτώσεις: τη γραμμική παλινδρόμηση και τη μείωση. Αυτά τα προγράμματα εκτελέστηκαν με διαφορετικά μεγέθη προβλημάτων και με διαφορετικές διανομές εργασιών. Έγιναν μετρήσεις στους χρόνους εκτέλεσης σε δύο ετερογενείς αρχιτεκτονικές στόχους, οι οποίες αποτελούνται από έναν επεξεργαστή (CPU) και δύο μονάδες επεξεργασίας γραφικών (GPUs). (οι ρυθμίσεις αρχιτεκτονικής προορίζονται στον Πίνακα 3). Τα αποτελέσματα αυτών των πειραμάτων παρουσιάζονται στο Σχήμα 1.



(a) Linear regression on *mc1*

(b) Linear regression on *mc2*

(c) Reduction on *mc1*

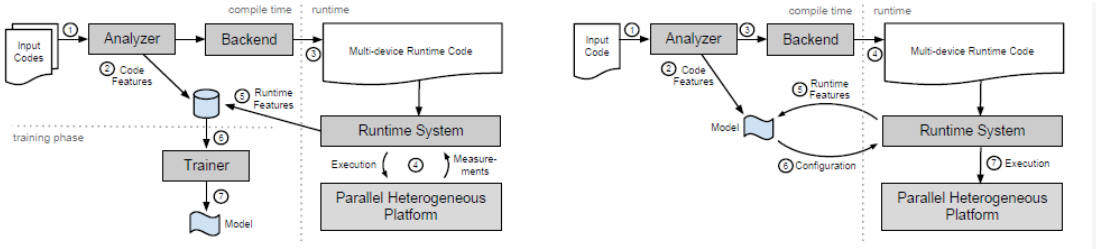
(d) Reduction on *mc2*

Σχήμα 1: Συμπεριφορά απόδοσης δύο προγραμμάτων σε διάφορες αρχιτεκτονικές με μεταβαλλόμενο μέγεθος προβλήματος (δηλαδή, ο αριθμός των

νημάτων). Κάθε γράφημα παρουσιάζει τους χρόνους εκτέλεσης (σε δευτερόλεπτα) σε λογαριθμική κλίμακα (άξονας y) με διάφορο αριθμό νημάτων (εργαστηριακά στοιχεία στο OpenCL) (άξονας x).

Σε μια από τις αρχιτεκτονικές (mc1, όπως φαίνεται στον Πίνακα 3), για μικρά προβλήματα, η GPU δεν είναι τόσο αποτελεσματική, και η καλύτερη απόδοση προέρχεται από τη χρήση μόνο του CPU για και τα δύο προγράμματα. Ωστόσο, για συγκεκριμένα μεγέθη προβλημάτων, είναι προτιμητέα μια υβριδική προσέγγιση που χρησιμοποιεί και τον CPU και μία ή δύο GPUs, ή ακόμη και μια προσέγγιση που βασίζεται μόνο στη GPU. Στη δεύτερη αρχιτεκτονική (mc2, βλ. Πίνακα 3), η γραμμική παλινδρόμηση επιτυγχάνει την καλύτερη απόδοση με μία GPU για μικρά προβλήματα, ενώ η μείωση επιτυγχάνει την καλύτερη απόδοση με έναν CPU. Καθώς τα μεγέθη των προβλημάτων αυξάνονται, οι GPUs γίνονται πιο αποτελεσματικές και η γραμμική παλινδρόμηση απαιτεί τη χρήση δύο GPUs τόσο στο mc1 όσο και στο mc2. Το πρόγραμμα μείωσης εμφανίζει διαφορετική συμπεριφορά για μεγαλύτερα προβλήματα, προτιμώντας υβριδικές λύσεις που υπερβαίνουν ομοιογενείς ρυθμίσεις κατά έως και 44% και 19% στα mc1 και mc2 αντίστοιχα.

Τα πειράματα αυτά αποδεικνύουν ότι, ακόμα και για μια μόνο εφαρμογή, η ιδανική διαίρεση των εργασιών εξαρτάται σε μεγάλο βαθμό από το μέγεθος του προβλήματος και τις δυνατότητες του υλικού. Ένα άλλο σημαντικό εύρημα του ετερογενούς υπολογισμού είναι η πρόκληση που αντιμετωπίζουν οι προγραμματιστές στη συγγραφή προγραμμάτων που μπορούν να εκτελεστούν ταυτόχρονα σε πολλαπλές συσκευές. Δεδομένου ότι οι σύγχρονοι μεταγλωττιστές δεν μπορούν ακόμα να αυτοματοποιήσουν αυτήν την περίπλοκη διαδικασία, υπάρχει ανάγκη για νέα εργαλεία που θα διευκολύνουν τον μετασχηματισμό υπαρχόντων προγραμμάτων σε ετερογενείς συστήματα.



(a) Training phase (b) Deployment phase

Figure 2: Framework Overview

Απλούστευση του Προγραμματισμού σε Ετερογενείς Υπολογιστικούς Συστηματικούς"

6.1 Αρχιτεκτονική

Το Σχήμα 2 παρουσιάζει την αρχιτεκτονική του προτεινόμενου πλαισίου, το οποίο αποτελείται από δύο κύρια στάδια: την εκπαίδευση και την εφαρμογή. Στην φάση της εκπαίδευσης, το σύστημα αναπτύσσει ένα μοντέλο πρόβλεψης για τη διαίρεση των εργασιών. Αυτό το μοντέλο μπορεί να δημιουργηθεί αυτόματα, ακόμα και για συστοιχίες στόχους που δεν υποστηρίζονταν προηγουμένως, επιτρέποντας την επέκταση του συστήματος σε νέες αρχιτεκτονικές χωρίς παρέμβαση από τον χρήστη. Η διαδικασία κατασκευής του μοντέλου περιλαμβάνει την προσφορά ενός συνόλου προγραμμάτων OpenCL στο σύστημα, τα οποία μεταφράζονται σε μια ενδιάμεση παράσταση παράλληλης εκτέλεσης Insieme. Από τον αναλυτή κώδικα (1), παράγεται η παράσταση αυτή (intermediate representation), από την οποία εξάγονται και αποθηκεύονται σε μια βάση δεδομένων (2) τα χαρακτηριστικά του προγράμματος (στατικά χαρακτηριστικά προγράμματος). Στη συνέχεια, η ενδιάμεση παράσταση του προγράμματος περνά στο πίσω μέρος του συστήματος, το οποίο δημιουργεί πολυσυσκευαστικό κώδικα OpenCL (3). Αφού δημιουργηθεί το νέο πρόγραμμα, θα υποβληθεί σε εκτέλεση χρησιμοποιώντας διάφορα μεγέθη προβλημάτων και διαφορετικές διαιρέσεις εργασιών. Οι μετρήσεις απόδοσης που προκύπτουν (4), μαζί με τα χαρακτηριστικά του προγράμματος που εξαρτώνται από το μέγεθος του προβλήματος (δηλαδή, χαρακτηριστικά εκτέλεσης), συλλέγονται και προστίθενται στη βάση δεδομένων (5). Αφού αυτά τα βήματα ολοκληρωθούν για όλα τα προγράμματα, ο εκπαιδευτής χρησιμοποιεί τα χαρακτηριστικά και τις μετρήσεις απόδοσης που αποθηκεύονται στη βάση δεδομένων (6) για να δημιουργήσει ένα πρότυπο πρόβλεψης διαίρεσης εργασιών (7).

Κατά τη φάση εφαρμογής, ένα νέο πρόγραμμα OpenCL παρέχεται στον αναλυτή για βελτιστοποιήσεις. Αρχικά, εξάγονται τα στατικά χαρακτηριστικά, και η ενδιάμεση παράσταση περνά στο πίσω μέρος, το οποίο δημιουργεί ένα πολυσυσκευαστικό πρόγραμμα OpenCL. Κατά την εκτέλεση του προγράμματος, τα χαρακτηριστικά εκτέλεσης παρέχονται στο εκπαιδευμένο μοντέλο, το οποίο συνδυάζει αυτά τα χαρακτηριστικά με τα στατικά χαρακτηριστικά του προγράμματος. Έτσι, προβλέπει την καλύτερη διαίρεση εργασιών για το τρέχον πρόγραμμα με το επιλεγμένο μέγεθος προβλήματος. Τέλος, το σύστημα χρόνου εκτέλεσης εκτελεί το πρόγραμμα στο δοσμένο υλικό χρησιμοποιώντας την προβλεπόμενη διαίρεση εργασιών (7).

6.2 Υλοποίηση

Το Insieme παρέχει ένα ισχυρό πλαίσιο για την ανάλυση και την τροποποίηση κώδικα προγραμμάτων που λειτουργούν σε ετερογενείς παράλληλους υπολογιστές. Αποτελείται κυρίως από δύο συστατικά: έναν μεταγλωττιστή που μεταφράζει τον κώδικα OpenCL σε ενδιάμεση αναπαράσταση και πίσω σε OpenCL, και ένα σύστημα εκτέλεσης που αναλαμβάνει τη διαχείριση και τον προγραμματισμό των προγραμμάτων. Ο μεταγλωττιστής παρέχει μια ενδιάμεση αναπαράσταση που επιτρέπει την εύκολη ανάλυση και τροποποίηση του κώδικα, ενώ το σύστημα εκτέλεσης

διαχειρίζεται την εκτέλεση των προγραμμάτων σε ετερογενείς πόρους, χρησιμοποιώντας διαφορετικές πολιτικές προγραμματισμού για την αποτελεσματική διανομή των εργασιών.

Το Insieme abc δέχεται ως είσοδο ένα πρόγραμμα OpenCL που προορίζεται για μια μοναδική συσκευή. Ένα πρόγραμμα OpenCL αποτελείται από δύο κύρια μέρη: το μέρος οικοδεσπότη και το μέρος συσκευής. Το μέρος οικοδεσπότη εκτελείται στην CPU και αναλαμβάνει τη διαχείριση των συσκευών OpenCL, όπως μια GPU ή η CPU ίδια. Το μέρος συσκευής, γνωστό ως πυρήνας, αντιπροσωπεύει μια εργασία παράλληλης εκτέλεσης δεδομένων και περιλαμβάνει τις υπολογιστικές εργασίες ενός μόνο νήματος. Κατά τη διάρκεια της εκτέλεσης, πολλές εργασιακές μονάδες εκτελούν παράλληλα εργασίες, ο αριθμός των οποίων αυξάνεται με το μέγεθος του προβλήματος. Η ανταλλαγή δεδομένων μεταξύ του οικοδεσπότη και της συσκευής γίνεται μέσω buffer μνήμης που περνούν ως ορίσματα στον πυρήνα.

Η μετάφραση του εισαγωγικού κώδικα OpenCL σε κώδικα ενδιάμεσης αναπαράστασης (IR) συμβαίνει μέσω δύο σταδίων. Καταρχάς, ο Μεταγλωττιστής Insieme χρησιμοποιεί το Clang C frontend, το οποίο είναι ένα εργαλείο ανοικτού κώδικα, για να δημιουργήσει ένα Διακριτικό Δέντρο Σύνταξης (AST) από τον εισαγωγικό κώδικα. Στη συνέχεια, το AST μετατρέπεται σε Ενδιάμεση Αναπαράσταση Παράλληλης Εκτέλεσης (INSPIRE) του Insieme, στην οποία εφαρμόζονται αναλύσεις και μετασχηματισμοί.

Για να κατανείμει μια εργασία, ο Μεταγλωττιστής Insieme εξετάζει το ενδιάμεσο αναπαραστατικό (IR) που παράγεται από το εισαγωγικό πρόγραμμα. Συλλέγει τους υπογραφείς όλων των προσπελάσεων στις μνημονικές buffer για να παράγει το μοτίβο πρόσβασης της μνημονικής buffer. Η ανάλυση αυτή εξετάζει εάν μια μνημονική buffer πρέπει να αντιγραφεί σε όλες τις διαθέσιμες συσκευές ή να κατανεμηθεί ομοιόμορφα μεταξύ αυτών. Μετά τη συλλογή όλων των προτύπων πρόσβασης, ελέγχεται εάν η μορφή πρόσβασης είναι (α) σταθερή, (β) το αποτέλεσμα μιας κυρτής συνάρτησης που εξαρτάται από το αναγνωριστικό νήματος ή (γ) κάτι άλλο. Εάν υπάρχουν μόνο προσπελάσεις τύπου (β), η μνημονική buffer διαιρείται μεταξύ όλων των συσκευών (δηλαδή, η μνημονική buffer είναι διαιρούμενη). Εάν υπάρχουν προσπελάσεις τύπου (α) ή (γ), μέρος αυτής (α) ή ολόκληρη η μνημονική buffer (γ) πρέπει να αντιγραφεί σε κάθε συσκευή (δηλαδή, η μνημονική buffer είναι μη διαιρούμενη). Σε περιπτώσεις όπου η πρόσβαση στα δεδομένα είναι τύπου (α) ή (γ), η ποσότητα των δεδομένων που πρέπει να μεταφερθεί αυξάνεται γραμμικά με τον αριθμό των συσκευών που χρησιμοποιούνται. Η αντιγραφή των ίδιων δεδομένων σε κάθε συσκευή είναι εφικτή μόνο εάν οι προσπελάσεις είναι αναγνωστικές. Σε περιπτώσεις προσπελάσεων εγγραφής τύπου (α) ή (γ), ο πυρήνας δεν μπορεί να διανεμηθεί. Η δυνατότητα διανομής ενός πυρήνα σε πολλαπλές συσκευές εξαρτάται από το εάν οι προσπελάσεις εγγραφής είναι διαιρούμενες. Ωστόσο, στην πλειοψηφία των πυρήνων, οι προσπελάσεις εγγραφής ακολουθούν μοτίβο πρόσβασης (β), που είναι διαιρούμενο. Η ανάλυση του μοτίβου πρόσβασης βασίζεται στον κώδικα της συσκευής. Ωστόσο, ο κώδικας του οικοδεσπότη

πρέπει να προσαρμοστεί ανάλογα με τα αποτελέσματα της ανάλυσης του μοτίβου πρόσβασης, για να διασφαλίσει τη σωστή διανομή των δεδομένων. Για αυτό τον λόγο, ο Μεταγλωττιστής Insieme συνδέει τον κώδικα του οικοδεσπότη και της συσκευής κατά τη μετάφραση του Δέντρου Σύνταξης του Clang σε IR, δυνατοποιώντας την ανάλυση του συνολικού προγράμματος. Μετά την ανάλυση, το IR μεταφράζεται από τον πυρήνα σε ένα πολυσυσκευαστικό πρόγραμμα OpenCL. Ο παράγοντας κώδικας έχει την ίδια σημασιολογία με τον αρχικό κώδικα, αλλά οι πυρήνες του μπορούν να κατανεμηθούν σε πολλές διαφορετικές συσκευές από το σύστημα εκτέλεσης. Συνεπώς, ορισμένες μνημονικές buffer αντιγράφονται ενώ άλλες κατανέμονται σε επιλεγμένες συσκευές, ανάλογα με το πώς προσπελαύνται εντός του πυρήνα. Για να επιλέξει τη διαίρεση εργασίας από πριν, το σύστημα χρόνου εκτέλεσης χρησιμοποιεί ένα μοντέλο που παράγεται από μηχανική μάθηση. Αυτό το μοντέλο βασίζεται σε στατικά χαρακτηριστικά του προγράμματος που εξάγονται κατά τον χρόνο μεταγλώττισης και σε χαρακτηριστικά ευαίσθητα στο μέγεθος του προβλήματος που συλλέγονται κατά τη διάρκεια της εκτέλεσης.

6.3 Περιορισμοί

Αν και το πλαίσιο Insieme μπορεί να βελτιστοποιήσει την απόδοση πολλών προγραμμάτων σε ανετοκέντρωση συστήματα, υπάρχουν ακόμη περιορισμοί που αφήνουν χώρο για μελλοντικές βελτιώσεις. Αυτή τη στιγμή, η ανάλυση των buffer και η διαίρεση της εργασίας γίνονται ξεχωριστά για κάθε πυρήνα. Σε προγράμματα με πολλούς πυρήνες, αυτό μπορεί να οδηγήσει σε περιττές μεταφορές δεδομένων, καθώς τα αποτελέσματα κάθε πυρήνα πρέπει να επιστρέφονται στον οικοδεσπότη για να γίνει η διανομή εργασιών ξανά με νέα διαίρεση.

Οι βελτιστοποιήσεις που αφορούν τη συγκεκριμένη συσκευή δεν περιλαμβάνονται στο πεδίο αυτού του έργου, αν και η διαίρεση της εργασίας που καθοδηγείται από μηχανική μάθηση μας θα μπορούσε θεωρητικά να την υποστηρίξει. Είμαστε ενήμεροι για τις δυνατότητες που μπορεί να ανοίξει αυτή η προσέγγιση και θα την εξετάσουμε προσεκτικά σε μελλοντικές εργασίες.

Άλλοι περιορισμοί προκύπτουν από την ανάγκη για αποκεντρωμένη πρόσβαση στα δεδομένα και την εκτέλεση ατομικών λειτουργιών, οι οποίες απαιτούν πρόσβαση σε μνημονικές buffer στη γενική μνήμη. Για τις αποκεντρωμένες προσπελάσεις στις μνημονικές buffer, η ανάλυση διακρίνει δύο περιπτώσεις: μνημονικές buffer μόνο για ανάγνωση και μνημονικές buffer και για ανάγνωση και εγγραφή. Στην πρώτη περίπτωση, κάθε συσκευή αντιγράφει ολόκληρη τη μνημονική buffer, ακόμα και δεδομένα που δεν είναι απαραίτητα. Στη δεύτερη περίπτωση, ο πυρήνας δεν διανέμεται καθώς η συλλογή και η συγχώνευση δεδομένων από διάφορες συσκευές δεν είναι εφικτή ακόμα. Όσον αφορά τη χρήση ατομικών λειτουργιών στις μνημονικές buffer, το OpenCL δεν παρέχει κανένα μέσο για την υλοποίηση τέτοιων λειτουργιών σε πολλές συσκευές, γι' αυτό το Insieme δεν υποστηρίζει προς το παρόν πυρήνες με ατομικές λειτουργίες.

Η προσέγγιση που περιγράφετε δεν είναι ικανή να αντιμετωπίσει ανεπαρκείς φορτίους εργασίας λόγω της δυσκολίας να προβλεφθεί εκ των προτέρων με στατικό τρόπο μια βέλτιστη διαίρεση εργασίας για τέτοιες περιπτώσεις.

ΚΕΦΑΛΑΙΟ 7

Εφαρμογή

Έχει υλοποιηθεί ένας αναφορικός κώδικας, χρησιμοποιώντας τη γλώσσα προγραμματισμού C++. Ο αναφορικός κώδικας έχει σχεδιαστεί με συγκεκριμένους στόχους. Συγκεκριμένα, αφαιρεί εντελώς τα πραγματικά πυρήνες, τους χρόνους εκτέλεσης και τους φόρτους εργασίας χρησιμοποιώντας αντικειμενοστραφή σχεδιαστικά πρότυπα. Το λογισμικό έχει δημιουργηθεί με αυτήν την αφαίρεση, ώστε να διατηρεί τη συνοχή των μεθόδων που χρησιμοποιούνται στη σύγκριση, χωρίς να εξαρτώνται από τις συγκεκριμένες λεπτομέρειες.

Το σημείο αναφοράς είναι σχεδιασμένο έτσι ώστε να συλλέγει μια ποικιλία δειγμάτων, όπου κάθε δείγμα καθορίζεται από τα χαρακτηριστικά μιας συγκεκριμένης μεθόδου χρονισμού, τον πυρήνα, το φορτίο εργασίας και τις παραμέτρους χρόνου εκτέλεσης. Για κάθε δείγμα, είναι δυνατόν να πραγματοποιηθεί οποιοσδήποτε αριθμός μετρήσεων χρονισμού. Ο ακριβής αριθμός μετρήσεων μπορεί να ρυθμιστεί δυναμικά και αυτόματα, επιτρέποντας τη δημιουργία μιας σταθερής μέτρησης.

Για να μην χάνεται πολύς χρόνος στη λήψη ακριβών εκτιμήσεων απόδοσης για δείγματα με χαμηλή απόδοση, ο δείκτης αναφοράς λειτουργεί σε δύο στάδια. Καταρχάς, γίνεται σάρωση όλων των δειγμάτων χωρίς να χρησιμοποιείται δυναμική δειγματοληψία για την λήψη ακριβών χρονομετρήσεων. Στη συνέχεια, τα δείγματα που δείχνουν πολύ χαμηλή απόδοση απορρίπτονται, και ένα δεύτερο στάδιο χρησιμοποιείται για να μετρήσει την απόδοση των υπόλοιπων δειγμάτων με υψηλή ακρίβεια.

Όλα τα δεδομένα που συλλέγονται μέχρι την ανάλυση των ατομικών μετρήσεων χρονισμού αποθηκεύονται σε ένα αρχείο XML. Αυτό το αρχείο μπορεί να χρησιμοποιηθεί τόσο για μελλοντική ανάλυση όσο και για τον έλεγχο και την επανεκκίνηση. Η δυνατότητα επανεκκίνησης είναι σημαντική και προσαρμοζόμενη, καθώς ο κώδικας επιτρέπει τη διεξαγωγή λεπτομερούς έρευνας στην απόδοση της πλατφόρμας, συμπεριλαμβανομένων δυναμικών αποφάσεων για τα δείγματα που πρέπει να ελεγχθούν και τον αριθμό των μετρήσεων χρονισμού που συλλέγονται ανά δείγμα.

7.1 Παράγοντες

Οι μονάδες επεξεργασίας γραφικών (GPU) υπόσχονται σημαντικά οφέλη στην απόδοση σε σύγκριση με τις παραδοσιακές αρχιτεκτονικές επεξεργαστών. Αυτό συνήθως οδηγεί σε μείωση του χρόνου εκτέλεσης για εφαρμογές που απαιτούν εκτεταμένους υπολογισμούς ή αλγόριθμους που αναζητούν μεγάλο εύρος ζώνης. Παρόλα αυτά, ο χρόνος εκτέλεσης δεν είναι ο μόνος παράγοντας που πρέπει να ληφθεί υπόψη κατά τη σύγκριση αρχιτεκτονικών υπολογιστών. Στην πραγματικότητα, ο χρόνος ανάπτυξης ενός επιστημονικού κώδικα είναι σε πολλές περιπτώσεις ένα σημαντικό κομμάτι της χρήσιμης διάρκειας ζωής του. Οι GPU μπορεί τώρα να επηρεάσουν ακόμα περισσότερο αυτήν την ισορροπία μέσω διαφόρων παραγόντων που πρέπει να ληφθούν υπόψη.

Πρώτον, η περιοχή των μαζικά παράλληλων επεξεργαστών βρίσκεται σε συνεχή εξέλιξη και αλλαγή. Αυτή η δυναμική οφείλεται σε πολλούς παράγοντες, όπως οι εξελίξεις στις διαδικασίες παραγωγής επεξεργαστών, οι νέες τεχνολογικές ιδέες που εφαρμόζονται στο υλικό και το λογισμικό, καθώς και οι διαρκώς μεταβαλλόμενες συνθήκες της αγοράς. Τα προγράμματα που εκτελούνταν αποτελεσματικά σε προηγούμενες γενιές υπολογιστών ενδέχεται να μην είναι πλέον οι βέλτιστες επιλογές, καθώς ο τομέας συνεχίζει να εξελίσσεται με γρήγορους ρυθμούς. Ενώ το πρότυπο OpenCL [14] παρέχει μια ενοποιημένη αφαίρεση για την εννοιοποίηση των ευρέως αποκλινόμενων αρχιτεκτονικών υλικού, απόδοσης Η φορητότητα παραμένει ένα δύσκολο πρόβλημα, τόσο μεταξύ των σύγχρονων ανταγωνιστικών αρχιτεκτονικών που διατίθενται σήμερα, καθώς και μεταξύ των σημερινών αρχιτεκτονικών και εκείνων του μέλλοντος. Αν και υπάρχουν ορισμένα πρότυπα που ακολουθούνται, ο κόσμος των μαζικά παράλληλων υπολογιστικών συστημάτων δεν έχει ακόμη εγκατασταθεί σε ένα σταθερό μοντέλο προγραμματισμού, όπως αυτό που είχαμε συνηθίσει στους επεξεργαστές κεντρικής μονάδας (CPUs) τις τελευταίες δεκαετίες.

Δεύτερον, Ο κώδικας που εκτελείται σε μονάδες επεξεργασίας γραφικών (GPU) είναι εξαιρετικά ευαίσθητος σε μικρές αλλαγές που μπορεί να φανούν ασήμαντες. Οι λεπτομέρειες της υλοποίησης του υλικού έχουν πολύ μεγαλύτερο αντίκτυπο στην απόδοση των προγραμμάτων GPU σε σύγκριση με τα προγράμματα CPU σήμερα. Παράγοντες όπως οι συχνότητες λειτουργίας, το πλάτος του διαύλου επικοινωνίας, η διαθέσιμη μνήμη και τα μεγέθη buffer έχουν άμεση επίδραση στην απόδοση του κώδικα. Η ίδια η προϋπόθεση του υπολογιστού GPU είναι να προσπαθήσουμε να βρούμε μια καλύτερη Χρήση για το πυρίτιο δεμένο στην προσωρινή αποθήκευση, την κερδοσκοπία και την εκτέλεση εκτός τάξης που ελευθερώνει μια σύγχρονη CPU Προγραμματιστής από την ανάγκη να ανησυχεί για τις ιδιαιτερότητες του υλικού. Αναμένεται λοιπόν ότι οι προγραμματιστές της GPU θα το κάνουν να συνεχίσει να εκτεθείτε σε αυτές τις λεπτομέρειες.

Τρίτον, και ενδεχομένως ένα συμπέρασμα του τελευταίου σημείου, οι GPU προσφέρουν πολλές ακόμη επιλογές εφαρμογής και Συχνά ελάχιστη καθοδήγηση

σχετικά με την επιλογή που μπορεί να οδηγήσει σε αποτελεσματικό κώδικα. Είναι συνηθισμένο να παρατηρούμε σημαντικές διαφορές στον χρόνο εκτέλεσης μεταξύ κωδικών που επιτελούν την ίδια βασική εργασία σε μονάδες επεξεργασίας γραφικών (GPU). Αυτό δεν είναι τόσο κοινό στις σύγχρονες CPU, εκτός από λίγες εξαιρέσεις, καθώς οι κωδικοί που έχουν υλοποιηθεί λογικά και έχουν υποστεί εκτενείς βελτιστοποιήσεις συνήθως έχουν παρόμοιους χρόνους εκτέλεσης.

Ο τέταρτος παράγοντας, που προκαλεί μάλιστα και δυσκολίες, είναι η ανεπαρκής ώριμης ανάπτυξη των εργαλείων ανάπτυξης για τις μονάδες επεξεργασίας γραφικών (GPU). Σε αντίθεση με τα εργαλεία που έχουν αναπτυχθεί και εκδοθεί για πολλά χρόνια για τις CPU, τα αντίστοιχα εργαλεία για GPU είναι σε πολύ πιο αρχικό στάδιο ανάπτυξης. Αυτά τα εργαλεία για CPU περιλαμβάνουν γλώσσες προγραμματισμού υψηλού επιπέδου και βιβλιοθήκες που επιτρέπουν στους προγραμματιστές να αναπτύξουν εφαρμογές με υψηλή παραγωγικότητα. Αυτά τα εργαλεία διευκολύνουν τον προγραμματισμό και τη βελτιστοποίηση, παρέχοντας αφαιρετικότητα από τις λεπτομέρειες της αρχιτεκτονικής του υλικού. Ωστόσο, πολλά από αυτά τα εργαλεία δεν έχουν ακόμα αναπτυχθεί επαρκώς για να υποστηρίξουν τις σύγχρονες παράλληλες αρχιτεκτονικές των GPU.

Προτείνεται ότι η δημιουργία κώδικα εκτέλεσης GPU (RTCG) αποτελεί χρήσιμη πρακτική που βοηθά τους προγραμματιστές να ανακτήσουν μια σημαντική μερίδα της παραγωγικότητας που μπορεί να χαθεί λόγω διαφόρων παραγόντων. Με τον όρο "GPU RTCG", εννοούμε τη δυνατότητα αποτελεσματικής εκτέλεσης αυθαίρετου χαμηλού επιπέδου πηγαίου κώδικα, συνήθως σε γλώσσα προγραμματισμού C ή κάποια παρόμοια, για υπολογιστικές εργασίες υψηλής απόδοσης. Στη συγκεκριμένη περίπτωση, η γενιά και η εκτέλεση του χαμηλού επιπέδου κώδικα γίνεται μέσω μιας γλώσσας υψηλού επιπέδου δέσμης ενεργειών. Η έννοια της "γλώσσας δέσμης ενεργειών" ή "γλώσσας υψηλού επιπέδου" αναφέρεται σε μια γλώσσα προγραμματισμού που παρέχει πολλαπλά παραδείγματα προγραμματισμού, όπως λειτουργικό, διαδικαστικό, αντικειμενοστραφές κλπ. Είναι δυναμικά τύπου, παρέχει εγκαταστάσεις αναφοράς σφαλμάτων, διαχειρίζεται αυτόματα τους πόρους και προσφέρει ολοκληρωμένη ενσωματωμένη λειτουργικότητα. Δεν απαιτεί ορατή συλλογή από το χρήστη και είναι κατάλληλη για διαδραστική χρήση. Επιπλέον, λειτουργεί καλά ως "γλώσσα κόλλας" για δομικά στοιχεία χαμηλότερου επιπέδου.

Η οικογένεια των μεγάλων γλωσσών δέσμης ενεργειών αυτής της γραφής περιλαμβάνει την Python [49], Ruby [11], Lua [19], JavaScript [8] και πολλοί άλλοι.

Η παρούσα εργασία περιγράφει τα διδάγματα που συλλέχθηκαν από πολλές προηγούμενες προσεγγίσεις. Το GPU RTCG είναι μια μορφή "Metaprogramming": Αντί να κατευθύνει αμέσως τον κωδικό υπολογιστή σε ένα πρόβλημα, κάποιος κατευθύνει τον κώδικα στο τη δημιουργία και τη συλλογιστική για άλλους κωδικούς που στη συνέχεια επιλύουν το πρόβλημα στο χέρι. Δεν είναι αρχικά σαφές εάν αυτό το επιπλέον επίπεδο προσφέρει πραγματικά οφέλη, αλλά αναβάλλουμε αυτήν τη συζήτηση

για αργότερα. Προς το παρόν, πρέπει να σημειώσουμε ότι δεν είμαστε οι πρώτοι που θα εφαρμόσουν αυτήν τη βασική αρχή. Σήμερα, ο πιο συνηθισμένος μηχανισμός για την εφαρμογή ιδεών μετα-αγροτογράφων είναι ο μηχανισμός προτύπου της γλώσσας προγραμματισμού C++.

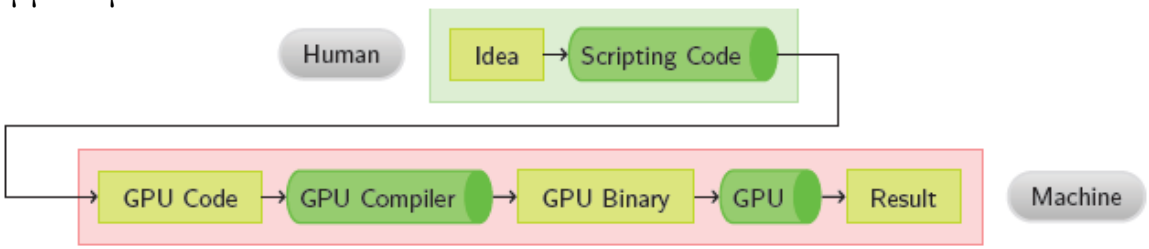


Figure 1. Operating principle of GPU code generation.

Αυτή η αποτελεσματική (αν και περιορισμένη σε ευελιξία) προσέγγιση συμπεριλαμβάνει τους αξιολογητές έκφρασης, τους γεννήτριες αναλυτές, και ακόμη και ολόκληρα πλαίσια διαλύτη PDE. Ωστόσο, η τεχνική αυτή περιορίζεται στην εφαρμογή της κατά την κατασκευή του λογισμικού, μειώνοντας έτσι την ευελιξία της. Υπάρχουν διάφοροι τρόποι που έχουν σχεδιαστεί για να παρακάμψουν αυτόν τον περιορισμό, συμπεριλαμβανομένης της συναρμολόγησης μικρών προκατασκευασμένων τμημάτων σε έναν πλήρη κώδικα, καθώς και την εκτέλεση αξιολόγησης χρόνου για διαφορετικές εκδόσεις κώδικα. Επιπλέον, πρέπει να σημειωθεί ότι ο προγραμματισμός σε LISP έφερε την θεμελιώδη έννοια της αρχιτεκτονικής Von Neumann, που υπογραμμίζει ότι "ο κώδικας είναι δεδομένα", μέσα από γλώσσες υψηλότερου επιπέδου στις αρχές της δεκαετίας του 1960, αν και ο κύριος στόχος δεν ήταν πάντα η υπολογιστική απόδοση.

Στον τομέα των Γραφικών Επεξεργαστών (GPUs), η προγραμματιζόμενη μονάδα έχει εφαρμοστεί κυρίως σε εφαρμογές γραφικών και επεξεργασίας εικόνας, με σκοπό να διευκολύνει τη χρήση ενός συνηθισμένου προτύπου απόδοσης για γενικές εφαρμογές. Άλλα έργα επικεντρώνονται στην ανάπτυξη κώδικα GPU με τη χρήση ενός πλαισίου που βασίζεται σε C++.

Επιπλέον, αυτή η προσπάθεια μπορεί να ενταχθεί στο πλαίσιο των πρόσφατων προσπαθειών για την προώθηση της παραγωγής προγραμμάτων ως κύρια ιδέα. Ωστόσο, σε αντίθεση με αυτό, επιλέγουμε μια πιο απλή προσέγγιση που βασίζεται στις αρχές της πρακτικής: Γιατί να επινοήσουμε νέα εργαλεία από το μηδέν όταν μπορούμε να επιτύχουμε καλά αποτελέσματα χρησιμοποιώντας μια γλώσσα δέσμης ενεργειών με GPU και Compiler; Παράδοξα, πολλοί προηγούμενοι συγγραφείς αναγνωρίζουν το ανεκτίμητο πλεονέκτημα ότι είναι δυνατό να δημιουργηθεί κώδικας κατά την εκτέλεση του προγράμματος με μεγάλη ευκολία.

7.2. Hardware GPU: Μια σύντομη εισαγωγή

Στις αρχικές ημέρες του προγραμματισμού GPU, οι προγραμματιστές αντιμετώπιζαν την πρόκληση να μετατρέψουν γραφικά με σταθερή λειτουργία σε γραφικά που μπορούν να προγραμματιστούν για υπολογιστικούς σκοπούς,

χρησιμοποιώντας διάφορες τεχνικές και μεθόδους. [36]. Με τη σημερινή γενιά των GPU, αυτό δεν ισχύει πια. Στην πραγματικότητα, οι GPU θα πρέπει να θεωρούνται ως εξαιρετικά παράλληλοι γενικοί επεξεργαστές που έχουν σχεδιαστεί για διαφορετικού τύπου φορτίο εργασίας σε σύγκριση με τις τρέχουσες CPU. Ο όρος "GPU" λειτουργεί απλά ως ένα βολικό μνημείο για αυτήν την τεχνολογία. Για τις CPU, το εύρος των εργασιών περιλαμβάνει συνήθως προγράμματα περιήγησης στον ιστό, επεξεργαστές κειμένου και μια ποικιλία άλλων εφαρμογών γραφείου, τα οποία είναι χαρακτηρισμένα από υψηλή πολυπλοκότητα και έχουν περιορισμένες δυνατότητες παραλληλισμού. Οι GPU, από την άλλη πλευρά, στοχεύουν στην εφαρμογή ομοιόμορφα, μέτρια πολύπλοκες λειτουργίες σε μεγάλους όγκους δεδομένων. Αυτό αποτελεί ένα ειδικό, αλλά σημαντικό, τμήμα του παράλληλου υπολογισμού που συχνά αναφέρεται ως δεδομένων-παράλληλος ή "ροής" υπολογιστών. [52].

Ένα από τα πιο σημαντικά προβλήματα που πρέπει να αντιμετωπίσει ο σύγχρονος σχεδιασμός του επεξεργαστή είναι η βραδύτητα του μνήμη. Παρά τις σημαντικές βελτιώσεις στην καθυστέρηση και την ταχύτητα πρόσβασης σε μεγάλης κλίμακας μνήμη RAM εκτός του επεξεργαστή, δεν έχουν συναρτηθεί με την ίδια ταχύτητα όπως η αύξηση της απόδοσης των επεξεργαστικών πυρήνων. Οι προβλέψεις που βασίζονται στις παραλλαγές του νόμου του Moore υποσχόντουσαν ότι αυτή η τελευταία πρόοδος θα ακολουθούσε εκθετική τάση, αλλά η πραγματικότητα διαφέρει από αυτήν την πρόβλεψη. Αποτέλεσμα είναι η αντιμετώπιση ενός αυξανόμενου χάσματος μεταξύ της υπολογιστικής ικανότητας των παράλληλων επεξεργαστών και της ταχύτητας πρόσβασης στα δεδομένα που απαιτούνται για την εκτέλεση των υπολογισμών. Ο χρόνος μεταξύ του αιτήματος μνήμης από έναν πυρήνα και της αντίστοιχης απόκρισης από τη μνήμη εκτός του επεξεργαστή μπορεί να αυξηθεί δραματικά, καθώς και το χάσμα μεταξύ τους.

Ενώ είναι δυνατή η αύξηση του εύρους ζώνης μέσω της βελτίωσης της διεπαφής μνήμης, η καθυστέρηση παραμένει σταθερή, αφού είναι βασική ιδιότητα του τύπου μνήμης. Οι CPU είναι ευάλωτες στις καθυστερήσεις μνήμης, και για να τις αντιμετωπίσουν, οι σχεδιαστές CPU υιοθετούν διάφορες στρατηγικές. Μια από αυτές είναι η χρήση μεγάλων ποσοστών γρήγορης προσωρινής μνήμης στο εσωτερικό του τσιπ για να μειώσουν την αναμονή για μνήμη εκτός του τσιπ. Επιπλέον, χρησιμοποιούν προβλέψεις και κερδοσκοπία για να εξασφαλίσουν ότι τα απαιτούμενα δεδομένα είναι διαθέσιμα εκ των προτέρων. Τέλος, αναδιατάσσουν τη ροή των εντολών για να μειώσουν την επίδραση των πάγκων που σχετίζονται με τη μνήμη.

Είναι προφανές ότι η υλοποίηση των στρατηγικών αυτών στην υλική δομή μπορεί να απαιτήσει μεγάλες ποσότητες πυρήνων. Στην αντίθετη περίπτωση, οι εργασίες-στόχοι για μια GPU είναι λιγότερο ευάλωτες στις καθυστερήσεις μνήμης. Επειδή οι GPU συνήθως επεξεργάζονται μεγάλα σύνολα δεδομένων, η συγκεκριμένη σειρά των δεδομένων είναι λιγότερο σημαντική. Αυτό επιτρέπει την απασχόληση ενός μεγάλου αριθμού πλαισίων εκτέλεσης, καθένα από τα οποία μπορεί να εκτελέσει λειτουργίες με

διαθέσιμα δεδομένα. Ενώ η διαχείριση αυτών των πλαισίων είναι πολύπλοκη, η σχετική λογική διαχείρισης είναι λιγότερο δαπανηρή από τις αντίστοιχες στρατηγικές στις CPU. Αυτό επιτρέπει στις GPU να επικεντρωθούν σε λειτουργικές μονάδες, αυξάνοντας τον παραλληλισμό.

Η υπερβολική πολυπλοκότητα που προκύπτει από τον μεγάλο αριθμό των λειτουργικών μονάδων αποτελεί μια πρόκληση για τους σχεδιαστές GPU. Η διαχείριση των πλαισίων εκτέλεσης απαιτεί περίπλοκη λογική που αυξάνεται σημαντικά με τον αριθμό των πλαισίων. Ένα κεντρικό σύστημα διαχείρισης για όλα τα πλαίσια σε όλες τις λειτουργικές μονάδες θα ήταν υπερβολικά μεγάλο και ανεφάρμοστο. Επιπλέον, οι φυσικοί περιορισμοί της ταχύτητας διάδοσης σήματος στο εσωτερικό του τσιπ υποδεικνύουν την ανάγκη να διαχωριστεί το διαθέσιμο χώρο σε μικρότερες μονάδες επεξεργασίας. Έτσι, οι σχεδιαστές εφαρμόζουν μια διαίρεση του τσιπ σε μικρότερους υπο-επεξεργαστές, καθένας από τους οποίους αναλαμβάνει τη διαχείριση ενός περιορισμένου συνόλου πλαισίων εκτέλεσης. Είναι το ίδιο σκέψη που οδηγεί CPU βαρέων βαρών προς την ενσωμάτωση πολλαπλών πυρήνες σε μία μόνο μήτρα. Σε σημερινές GPU, υπάρχουν πολλοί υποτομείς διαχείρισης, καθένας εκ των οποίων μπορεί να διαχειριστεί εκατοντάδες πλαίσια εκτέλεσης. Αυτοί οι υποτομείς, γνωστοί και ως "υπολογιστικές μονάδες" ή "πυρήνες", αποτελούν τον κύριο μηχανισμό εκτέλεσης εντολών. Για να αυξηθεί ακόμη περισσότερο η αναλογία μεταξύ λειτουργικών μονάδων και ελέγχου και να φτάσει στο επίπεδο εκατοντάδων πλαισίων ανά υποτομέα, οι περισσότερες GPU σχεδιάζονται ως ευρείες SIMD (Single Instruction, Multiple Data) μηχανές διανυσμάτων.

Η ιεραρχία του chip -> unit -> context έχει διπλάσια επίδραση στο λογισμικό GPU: Αρχικά, κάθε μονάδα εκτέλεσης σε μια GPU συνήθως λειτουργεί ανεξάρτητα από τις άλλες, περιορίζοντας την επικοινωνία μεταξύ των πλαισίων που εκτελούνται στην ίδια μονάδα. Στη συνέχεια, τα προγράμματα πρέπει να καθορίσουν ρητά πώς θα χρησιμοποιηθεί κάθε επίπεδο παραλληλισμού, συνήθως παρέχοντας κατάλληλο διαχωρισμό του χώρου εργασίας. Όλα αυτά δημιουργούν το πρόβλημα της τεμαχισμένης εκτέλεσης βρόχου. Ο όρος "τεμαχισμός βρόχου" αναφέρεται στη συνολική διαδικασία του:

- Καθορισμού των άξονων του βρόχου που μπορούν να χρησιμοποιηθούν ως δείκτες παραλληλισμού,
- Ανάθεση των άξονων βρόχου σε διαθέσιμους άξονες παραλληλισμού, όπως μονάδες υπολογισμού, αριθμούς εκτέλεσης μέσα σε μια μονάδα, και λωρίδες SIMD,
- τροποποίηση των εντολών για πιο ευνοϊκή σειρά προσβάσιμης μνήμης,
- Και, τέλος, επιβολή περιορισμών μεγέθους σε κάθε άξονα βρόχου και άξονες χωρισμού.

Κάθε ένα από αυτά τα βήματα εξαρτάται από το αποτέλεσμα όλων των άλλων, δημιουργώντας ένα πολύπλοκο πρόβλημα βελτιστοποίησης. Σκοπός του παρόντος άρθρου είναι να εξετάσει αυτές τις προκλήσεις λογισμικού και να προτείνει λύσεις για ορισμένες από αυτές.

7.3. Δημιουργία λογισμικού GPU

Όταν αναπτύσσουμε λογισμικό για GPU με χρήση χαμηλού επιπέδου περιβάλλοντα προγραμματισμού όπως το OpenCL και το CUDA, απαιτείται να κατανοήσουμε και να λαμβάνουμε υπόψη πολλούς παράγοντες της αρχιτεκτονικής της GPU. Η αποτελεσματική εκτέλεση εξαρτάται κρίσιμα από πολλές αρχιτεκτονικές παραμέτρους, όπως:

• Το πλάτος και ο αριθμός των υπολογιστικών μονάδων: Πόσες μονάδες εκτέλεσης υπάρχουν στη GPU και πόσα υπολογιστικά κομμάτια μπορούν να εκτελεστούν ταυτόχρονα.,

• η διαθέσιμη κατάσταση αρχείου μητρώου στο τσιπ: Το πόσο γρήγορα μπορούν οι μονάδες εκτέλεσης να αποκτήσουν πρόσβαση σε πληροφορίες που αποθηκεύονται στο τσιπ.

• η διαθέσιμη μνήμη buffer on-chip: Η ποσότητα της γρήγορης μνήμης που υπάρχει απευθείας στο τσιπ.

• Η ταχύτητα διαφόρων προτύπων πρόσβασης σε μνήμη: Πόσο γρήγορα μπορούν οι μονάδες εκτέλεσης να διαβάσουν και να γράψουν στη μνήμη, είτε είναι on-chip είτε off-chip.

• Ο λόγος του διαθέσιμου εύρους ζώνης μνήμης: Πόσα δεδομένα μπορούν να μεταφερθούν μεταξύ της μνήμης του υπολογιστή (CPU) και της GPU.

• Η λανθάνουσα κατάσταση και το εύρος ζώνης μεταξύ **CPU και GPU:** Πόσο αποτελεσματικά μπορούν οι δύο μονάδες επεξεργασίας να επικοινωνούν και να μοιράζονται δεδομένα.

• Οι λεπτομέρειες προγραμματισμού οδηγιών του επεξεργαστή: Πώς ο προγραμματιστής χρησιμοποιεί τις δυνατότητες του OpenCL ή του CUDA για να εκμεταλλευτεί τις δυνατότητες της GPU.

Ο καθορισμός αυτών των παραμέτρων είναι κρίσιμος για την αποτελεσματική εκτέλεση του λογισμικού στην αρχιτεκτονική της GPU.

Η διαδικασία της αντιστοίχισης ενός υπολογισμού σε διαφορετικές δομές υλικού GPU είναι πολύπλοκη και εμπλέκει πολλούς παράγοντες απόδοσης. Οι προγραμματιστές συχνά πρέπει να κάνουν συμβιβασμούς μεταξύ της αποτελεσματικότητας σε διάφορες πτυχές, όπως η ταχύτητα και η χωρητικότητα, καθώς και άλλοι παράγοντες απόδοσης. Σε πολλές περιπτώσεις, οι προγραμματιστές λαμβάνουν αποφάσεις χωρίς πλήρεις πληροφορίες σχετικά με τον εξοπλισμό ή τις λεπτομέρειες του υπολογισμού. Αυτό μπορεί να οδηγήσει σε συμβιβασμούς που δεν είναι πάντα βέλτιστοι για την εκάστοτε κατάσταση. Αλλά ακόμα κι αν είναι, εκτέλεση προγράμματος μαζικά Οι παράλληλοι επεξεργαστές είναι μια περίπλοκη και μη τοπική διαδικασία που μπορεί να αψηφήσει την εύκολη κατανόηση ακόμη και από το Οι σχεδιαστές του επεξεργαστή.

Ο προγραμματισμός GPU απαιτεί πολύ πειραματισμό και μικροεπεξεργασία για να αντιμετωπίσει την έλλειψη γνώσης σχετικά με τις αιτίες των προβλημάτων. Αυτό δεν είναι ικανοποιητικό για τους προγραμματιστές, καθώς οι αλλαγές στο υλικό ή σε άλλες παραμέτρους μπορεί να μην οδηγούν πάντα σε ισχυρά αποτελέσματα. Η βελτιστοποίηση γίνεται ακόμα πιο δύσκολη εξαιτίας της απλότητας των GPU πυρήνων, που συχνά λείπουν από χαρακτηριστικά που μπορούν να δώσουν κατευθυντήρια σημάδια για τις αποφάσεις. Επιπλέον, η διαδικασία αυτή είναι κουραστική και επαναλαμβανόμενη, οπότε οι προγραμματιστές επιθυμούν αυτοματοποίηση. Αυτό μπορεί να οδηγήσει στον μεταπρογραμματισμό, τον αυτόματο προγραμματισμό και τη χρήση Τεχνητής Νοημοσύνης για την επίλυση των προβλημάτων.

7.4. Προβλήματα που επιλύθηκαν με γενιά κωδικών χρόνου εκτέλεσης GPU

Αυτή η ενότητα είναι αφιερωμένη στην περιγραφή ορισμένων προβλημάτων που αντιμετωπίζουν συνήθως κατά τον προγραμματισμό GPU. Γενικά, προτείνουμε μια στρατηγική GPU RTCG ως απλό και αποτελεσματικό τρόπο αντιμετώπισης αυτών των προβλημάτων.

7.4.1. Αυτοματοποιημένος συντονισμός

Κατά τη δημιουργία ενός προγράμματος GPU, είναι φυσικό να δημιουργηθούν πολλές παραλλαγές ενός κώδικα, καθένα με διαφορετικές ιδιότητες σχετικά με τη διάταξη των δεδομένων και την απόδοση του υπολογισμού. Η συνήθης πρακτική είναι να διαλέξει ο προγραμματιστής την ταχύτερη παραλλαγή και να απορρίψει τις υπόλοιπες. Αυτή η προσέγγιση μπορεί να αποτελέσει απώλεια πληροφοριών. Ένα εναλλακτικό προσεγγίση είναι να διατηρηθούν όσο το δυνατόν περισσότερες παραλλαγές, υποθέτοντας ότι κάθε μία μπορεί να έχει κάποια υποσχόμενα χαρακτηριστικά. Επιπλέον, κάθε παραλλαγή μπορεί να περιλαμβάνει πολλαπλές παραμέτρους που μπορούν να προσαρμοστούν, όπως το μέγεθος των βρόχων, των μπλοκ, κ.λπ. Η διατήρηση των παραλλαγών επιτρέπει την επιλογή του καλύτερου από ένα λογικό μέγεθος υποψηφίων με αυτοματοποιημένο τρόπο, καθοδηγούμενη από κάποια μέτρηση όπως η ταχύτητα εκτέλεσης. Η βασική προϋπόθεση του αυτοματοποιημένου συντονισμού είναι να ενεργοποιείται συνεχώς από τη GPU RTCG. Επιπλέον, ο αυτοματοποιημένος συντονισμός δρα όχι μόνο υπό την επήρεια της RTCG, αλλά και στον σωστό χρόνο, δηλαδή κατά την εκτέλεση, όταν οι πλήρεις πληροφορίες είναι διαθέσιμες. Παρουσιάζουμε τρία παραδείγματα που απεικονίζουν τον τύπο των επιλογών που επιλύονται βέλτιστα με αυτόματο συντονισμό.

Πρώτον, ένας κρίσιμος παράγοντας στον σχεδιασμό αλγορίθμων GPU είναι η κατάλληλη κοπή των βρόχων. Αν και οι βρόχοι μπορεί να είναι απλοί στην CPU, στην GPU απαιτούν πολλαπλά επίπεδα διαίρεσης για αποτελεσματική εκτέλεση, περιλαμβανομένων λωρίδων SIMD και μονάδων εκτέλεσης. Σε ορισμένους

αλγορίθμους, όπως ο πολλαπλασιασμός μητρώων, ο τεμαχισμός του βρόχου είναι σημαντικός ακόμη και στη CPU, για την αποτελεσματική διαχείριση της πρόσβασης στη μνήμη και τη βελτιστοποίηση των μνημών cache. Με τις περιορισμένες προσωρινές μνήμες και τα πολλά επίπεδα τεμαχισμού των GPU, η ορθή κοπή του βρόχου είναι ουσιώδης για την επίτευξη αποδοτικής εκτέλεσης.

Δεύτερον, Οι σύγχρονες αρχιτεκτονικές GPU περιλαμβάνουν μνήμες που μπορούν να διαχειριστούν οι χρήστες. Όταν αναπτύσσεται κώδικας, δεν είναι πάντα εμφανές ποια δεδομένα θα επωφεληθούν περισσότερο από την τοπική αποθήκευση χαμηλής λανθάνουσας μνήμης. Η μνήμη στο τσιπ είναι περιορισμένη και πολύτιμη, και για να επιτευχθεί υψηλή απόδοση, απαιτούνται συμβιβασμοί που προσαρμόζονται στην τρέχουσα κατάσταση του υλικού.

Τρίτον, Οι αρχιτεκτονικές GPU σχεδιάζονται για να εκμεταλλεύονται μεγάλα ποσά εύρους ζώνης στη μνήμη DRAM. Η διαμάχη για την πρόσβαση στη μνήμη DRAM επομένως αποτελεί κρίσιμο παράγοντα περιορισμού της απόδοσης. Λόγω των χαρακτηριστικών του DRAM και Αρχιτεκτονικές ελεγκτών μνήμης, η αποτελεσματική χρήση του εύρους ζώνης DRAM απαιτεί να δίνει ιδιαίτερη προσοχή Πώς είναι πρόσβαση στη μνήμη. Επομένως, ο συντονισμός απόδοσης στην GPU συχνά επικεντρώνεται στη βελτιστοποίηση των προτύπων πρόσβασης στη μνήμη, τα οποία περιλαμβάνουν την αλλαγή της δομής των δεδομένων και τον τρόπο με τον οποίο γίνεται η χαρτογράφηση των υπολογισμών στη GPU. Αυτό το πρόβλημα βελτιστοποίησης περιλαμβάνει την αντιμετώπιση συμβιβασμών μεταξύ του τρόπου εκτέλεσης των υπολογισμών και του τρόπου αποθήκευσης των δεδομένων. Χωρίς εργαλεία για την ανάλυση αυτών των συμβιβασμών, ο προγραμματιστής πρέπει να αναδιαρθρώσει χειροκίνητα τον κώδικα και τα δεδομένα του για να εξερευνήσει τον χώρο του συντονισμού, μια διαδικασία που είναι κουραστική και επιρρεπή σε σφάλματα.

7.4.2. Το κόστος ευελιξίας

Η ευελιξία αποτελεί συνήθως επιθυμητό χαρακτηριστικό ενός προγράμματος - όπου το "πρόγραμμα" συνήθως αναφέρεται σε ένα εκτελέσιμο που χρησιμοποιείται από τους χρήστες. Η ικανότητα ενός εκτελέσιμου να εκτελεί πολλές λειτουργίες χωρίς τροποποίηση θεωρείται ιδιαίτερα επιθυμητή. Ωστόσο, αυτή η ευελιξία μπορεί να συγκρουστεί με την απόδοση. Για παράδειγμα, ένας κώδικας που είναι βελτιστοποιημένος για να πολλαπλασιάζει μόνο μητρώα συγκεκριμένου μεγέθους μπορεί να είναι άχρηστος αν η εφαρμογή απαιτεί πολλαπλασιασμούς μητρών διαφορετικών μεγεθών. Επομένως, σχεδόν όλοι οι κώδικες υπολογιστών σχεδιάζονται με τουλάχιστον κάποιο βαθμό ευελιξίας.

Είναι σημαντικό να συνειδητοποιήσουμε ότι η ευελιξία έχει το κόστος της: σταθερές τιμές αντικαθίστανται από μεταβλητές, σταθεροί μετρητές βρόχων γίνονται μεταβλητοί, και ο μεταγλωττιστής έχει λιγότερη πληροφορία κατά τον χρόνο

μεταγλώττισης, που καθιστά τον βελτιστοποιητή λιγότερο αποτελεσματικό. Η διαδικασία αφαίρεσης αυτής της ευελιξίας, που είναι ευρέως γνωστή ως "hardcoding", μπορεί να μειώσει την πολυπλοκότητα του κώδικα, αλλά ταυτόχρονα μειώνει και την ευελιξία του συστήματος.Αισθανόμαστε, ωστόσο, ότι έχει αυτή η άποψη Δεν υπάρχει διαθέσιμη γενιά κώδικα εκτέλεσης, καθώς κάποιος είναι ελεύθερος να δημιουργήσει κώδικα για ένα ακριβός σκοπό - οποιαδήποτε επιπλέον ευελιξία είναι πιθανό απλώς άγνωστο έρμα.

Σε πλαίσια compile-time metaprogramming, το hardcoding αντικαθίσταται μερικές φορές από τη δημιουργία ενός μεγάλου Αριθμός δυνητικά απαραίτητων παραλλαγών κώδικα μπροστά από το χρόνο, εξετάζοντας τις αναμενόμενες ανάγκες για διαφορετικές μεγέθη προβλημάτων, τύποι δεδομένων, κλπ. Όταν ο αριθμός των παραλλαγών ξεπεράσει ένα συγκεκριμένο όριο, το κόστος αυτής της προσέγγισης αυξάνεται γρήγορα τόσο στον χρόνο σύνταξης όσο και στο αποτύπωμα μνήμης του εκτελέσιμου προγράμματος. Σε αντίθεση, η GPU RTCG δεν υποφέρει από τέτοιες ποινές κλιμάκωσης: μπορεί να χρησιμοποιήσει πληροφορίες που είναι διαθέσιμες μόνο κατά την εκτέλεση για να μειώσει τον αριθμό των παραλλαγών που πρέπει να δημιουργηθούν. Μπορεί να εκμεταλλευτεί την προσωρινή αποθήκευση για να απορροφήσει το κόστος της διαχείρισης του κώδικα. Επιπλέον, μπορεί να απορρίψει αμέσως τυχόν αχρησιμοποίητες παραλλαγές κώδικα, βοηθώντας στην εύρεση του βέλτιστου κώδικα.

7.4.3. Αφαίρεση υψηλής απόδοσης

Στην πλειονότητα των προγραμμάτων υπολογιστών, η δομή οργάνωσης είναι ιεραρχική, όπου κάθε επίπεδο (στρώμα) αναλαμβάνει την επίλυση ενός συγκεκριμένου υποπροβλήματος. Κάθε στρώμα παρέχει μια αφηρημένη διεπαφή προς τα άνω στρώματα, επιτρέποντας τη διάσπαση ενός μεγάλου προβλήματος σε πιο διαχειρίσιμα μικρότερα. Αυτή η πρακτική είναι αποδεκτή στη μηχανική, καθώς διευκολύνει τη διάσπαση και την επαναχρησιμοποίηση της προσπάθειας. Ωστόσο, υπάρχουν περιπτώσεις όπου η υπερβολική αφαίρεση είναι αντιοικονομική και μπορεί να προκαλέσει απώλεια πληροφοριών για τα άνω στρώματα. Σε αυτό το πλαίσιο, η GPU RTCG αντιμετωπίζει αυτό το ζήτημα, επιδιορθώνοντας την κατάσταση με προχωρημένες τεχνικές που βελτιώνουν την ισορροπία μεταξύ της αφαίρεσης και της διατήρησης των πληροφοριών. Ένα κοινό παράδειγμα αντιοικονομικών αφαίρεσης συμβαίνει όταν ο καταναλωτής μιας διεπαφής πρέπει να καθορίσει Λεπτομέρειες σχετικά με μια λειτουργία που πρόκειται να εκτελεστεί σε μεγάλους όγκους δεδομένων, ως μέρος ενός εσωτερικού βρόχου στο αφαίρεση. Ως ασήμαντο παράδειγμα, εξετάστε μια αφηρημένη μορφή προσθήκης διανυσμάτων που επιτρέπει μια ποικιλία κλιμακωτών τύποι.

Μια απλή τεχνική για τη βελτίωση του χρόνου εκτέλεσης είναι η χρήση δεικτών λειτουργιών, γνωστή και ως εικονικές μεθόδους. Στο παράδειγμά μας, το κόστος μιας

κλιμακωτής προσθήκης είναι πολύ μικρότερο από αυτό της κλήσης μέσω ενός δείκτη λειτουργίας, συχνά με αρκετές τάξεις μεγέθους. Αυτό συμβαίνει επειδή μια τέτοια κλήση μπορεί να ξεπεράσει τη λογική πρόβλεψης και να προκαλέσει τη διακοπή της ροής εκτέλεσης. Έτσι, η χρήση μιας υπολογιζόμενης πρόσκλησης για μία μόνο πράξη προσθήκης είναι ανέφικτη και απαιτεί την απόσβεση του κόστους σε πολλές επιχειρήσεις (κλιμακωτές προσθήκες στο παράδειγμα), αν αυτό είναι εφικτό. Εκτός από την πρόσθετη πολυπλοκότητα, αυτή η προσέγγιση μπορεί να μην είναι βιώσιμη σε ορισμένες αρχιτεκτονικές GPU

Τα μειονεκτήματα της προσέγγισης του δείκτη λειτουργίας οδήγησαν στην ανάπτυξη μηχανισμών για τη σύνταξη Πολυμορφισμός στη CPU και τη GPU. Στο C ++, αυτό επιτυγχάνεται με τη χρήση της κλάσης και της λειτουργίας πρότυπα. Αν ο μεταγλωττιστής γνωρίζει τις προτιμήσεις του χρήστη κατά τον χρόνο μεταγλώττισης, μπορεί να εκμεταλλευτεί αυτές τις πληροφορίες για να δημιουργήσει αποτελεσματικότερο κώδικα. Για παράδειγμα, η πρόσθεση φορέα μπορεί να υλοποιηθεί με σεβασμό σε έναν αόριστο τύπο, βασιζόμενη στην υπόθεση ότι οι υποκείμενες κλιμακωτές προσθήκες υπάρχουν. Αυτός ο τύπος πρέπει να είναι γνωστός κατά τον χρόνο μεταγλώττισης, επιτρέποντας στο μεταγλωττιστή να βρει δυναμικά τη ρουτίνα πρόσθεσης και να την ενσωματώσει ("inline") στον κώδικα, εξαλείφοντας τα γενικά έξοδα. Αυτή η προσέγγιση είναι δημοφιλής, αλλά έχει δύο ελλείψεις: Πρώτον, απαιτεί πρόωρη δέσμευση. Όλες οι επιθυμητές χρήσεις του κώδικα πρόσθεσης φορέα πρέπει να είναι γνωστές πριν την εκτέλεση του προγράμματος. Δεύτερον, το C ++ Ο μηχανισμός προτύπου ιδιαίτερα ανταποκρίνεται δυσμενώς στην ανάπτυξη της πολυπλοκότητας. Κάνει απλά πράγματα όπως Πληκτρολογήστε την αντικατάσταση αρκετά εύκολη. Τα πρότυπα, ακόμη και χωρίς το υπόλοιπο του C++, αποτελούν μια πλήρως ικανή αν αρκετά αμφίκοπη γλώσσα προγραμματισμού, και ορισμένοι υλοποιητές το βλέπουν ως πρόκληση για να κάνουν πιο προχωρημένα πράγματα μαζί τους. Αυτή η προσέγγιση επιβεβαιώνει την ανάγκη για ένα ενδιάμεσο επίπεδο όπου ο κώδικας μπορεί να αλληλεπιδράσει με άλλον κώδικα, αλλά το τελικό αποτέλεσμα σε αυτή την περίπτωση είναι τόσο εύθραυστο όσο και περίπλοκο.

Η ιδανική λύση θα ήταν ένας συμβιβασμός αυτών των δύο προσεγγίσεων. Οι δείκτες λειτουργιών είναι απλοί, ευέλικτοι και δεν απαιτούν πρόωρη σκυροδέματα, ενώ τα πρότυπα προσφέρουν ελάχιστη προσθήκη στον κώδικα. Με την αφαίρεση της διάκρισης μεταξύ "χρόνου μεταγλώττισης" και "χρόνου εκτέλεσης", το RTCG γεμίζει αυτό το κενό. Μόλις είναι διαθέσιμο το RTCG, ο κατάλληλος κώδικας μπορεί να δημιουργηθεί κάθε φορά που προκύπτει μια διαφορετική απαίτηση, παρέχοντας έτσι ευελιξία. Ο κωδικός RTCG είναι επίσης γρήγορος - Μπορεί να απομακρύνει κάθε είδους ευελιξία, επειδή μπορεί να θεωρηθεί με ασφάλεια "μονής χρήσης". Επιπλέον, κωδικός μπορεί να θεωρηθεί ως εργασία επεξεργασίας κειμένου. Δεδομένου ότι το ένα δεν περιορίζεται στην επιλογή των εργαλείων, οι κωδικοί που βασίζονται σε RTCG

μπορούν να είναι όσο το δυνατόν πιο απλοί και να ανταποκριθούν ευνοϊκά στην πολυπλοκότητα ανάπτυξη.

7.4.4. GPU και η ανάγκη για ευελιξία

Τελικώς, πρέπει να σημειωθεί ότι στο παρελθόν, λόγω της σχετικά υψηλής πολυπλοκότητας της ανάπτυξης, ειδικά όταν πρόκειται για τεχνικές βασισμένες σε C++, ο μεταπρογραμματισμός ήταν περιορισμένος σε εφαρμογές με υψηλές απαιτήσεις. Το κόστος της μεταπρογραμματισμού αντισταθμίζει τα μειονεκτήματα της "hardcoding" μόνο σε μεγάλη κλίμακα έργων.

Οι GPU, ωστόσο, εκδημοκρατίζουν αυτήν την ανάγκη, καθώς έβαλαν μεγαλύτερη ποινή σε άκαμπτο, μη άκαμπτο κώδικα. Με την απόφαση να εφαρμοστεί ένας αλγόριθμος στη GPU, κάποιος εκφράζει σιωπηρά την προθυμία του να ανταλλάξει μερική προσπάθεια υλοποίησης για ένα σημαντικό κέρδος στην απόδοση. Όπως έχει εξηγηθεί προηγουμένως, η εύρεση μιας καλής εφαρμογής συχνά δεν είναι τμηματική, επομένως το πιθανό κέρδος από τον RTCG είναι σημαντικό. Με άλλα λόγια, η αύξηση της χρήσης των GPU αυξάνει το σχετικό κόστος της μη χρήσης τεχνικών μεταπρογραμματισμού, και επομένως είναι πιθανό να διαπιστωθεί ευρύτερη υιοθέτηση της δημιουργίας κώδικα και τεχνικών όπως αυτές που προσφέρει ο RTCG. Ωστόσο, είναι απαραίτητη η ύπαρξη καλών εργαλείων που θα επιτρέπουν την ευρύτερη αποδοχή του RTCG από προγραμματιστές.

ΚΕΦΑΛΑΙΟ 8

Αποτελέσματα

8.1. Υλικό και Πλατφόρμα Λογισμικού

Υλικό και Πλατφόρμα Λογισμικού: Αυτό αναφέρεται στην ανάγκη να ληφθούν υπόψη τόσο το υλικό (hardware) όσο και η πλατφόρμα λογισμικού (software platform) όταν πραγματοποιείται συγκριτική αξιολόγηση απόδοσης. Αυτά τα δύο στοιχεία επηρεάζουν την εκτέλεση των εφαρμογών και πρέπει να συμπεριληφθούν στην αξιολόγηση.

Οι αναφορές αναφέρονται σε ένα ετερογενές σύστημα με 64 κόμβους. Από αυτούς, 48 κόμβοι είναι IBM dx360M4 και εκτοπίζονται με επεξεργαστές Intel Phi 5110P, ενώ οι υπόλοιποι 16 κόμβοι είναι επίσης IBM dx360M4, με κάθε ένα να εξοπλίζεται με κάρτες επιτάχυνσης NVIDIA K20M GPU. Κάθε κόμβος διαθέτει διπλούς επεξεργαστές Intel Xeon E5-2670 (Sandy Bridge), 64 GB μνήμης, και δικτυακό εξοπλισμό Mellanox FDR-10 InfiniBand. Όλες οι αναφορές που αναφέρονται εδώ χρησιμοποίησαν μόνο μία συσκευή επιτάχυνσης.

Ο στόχος της εργασίας αυτής είναι να προσφέρει μια συγκριτική αξιολόγηση διαφορετικών αρχιτεκτονικών επεξεργαστών. Αυτό επιτυγχάνεται καλύτερα όταν χρησιμοποιείται ένας μόνο επεξεργαστής, καθώς τα πολλαπλά σημεία αναφοράς δεν επηρεάζουν σημαντικά την ανάλυση μιας επεξεργαστικής αρχιτεκτονικής. Αντίθετα, αυτά τα σημεία αναφοράς μετρούν την απόδοση της πλατφόρμας που είναι πιο εκτεταμένη (που δεν είναι ο βασικός στόχος αυτής της έρευνας). Όλα τα σημεία αναφοράς εκτελέστηκαν με αποκλειστική πρόσβαση στον κόμβο υποδοχής και την κάρτα επιτάχυνσης χρησιμοποιώντας μια διαδραστική συνεδρία προγραμματισμού.

Όλα τα σημεία αναφοράς εκτελέστηκαν με την ενεργοποιημένη μνήμη κώδικα διόρθωσης σφαλμάτων (ECC). Αυτό διασφαλίζει τη συνέπεια στην ανίχνευση και διόρθωση διπλών ψηφίων σφαλμάτων για τη σύγκριση μεταξύ των αποτελεσμάτων των επεξεργαστών Phi και K20. Η ενεργοποίηση του ECC στο K20s έχει ως αποτέλεσμα τη μείωση της διαθέσιμης χωρητικότητας μνήμης κατά περίπου 12,5% και αντίστοιχη μείωση στο εύρος ζώνης.

Το λογισμικό στο σύστημα περιελάμβανε τον Intel Compiler XE, την εργαλειοθήκη NVIDIA CUDA-5, το κιτ ανάπτυξης λογισμικού COPRTHR (SDK) έκδοση 1.6, καθώς και τον κώδικα αναφοράς που αναπτύχθηκε σε αυτή την εργασία. Ο κώδικας αναφοράς SHOC χρησιμοποιήθηκε για βασικές συγκρίσεις και περιλάμβανε τις εκδόσεις OpenCL και CUDA, καθώς και την έκδοση βελτιστοποίησης MIC με χρήση του ιδιόκτητου API Intel με προσαρμογές εκφόρτωσης.

8.2. Βελτιωμένη απόδοση δεικτών αναφοράς

Βελτιωμένη απόδοση δεικτών αναφοράς: Αναφέρεται στην προσπάθεια βελτιστοποίησης της απόδοσης των δεικτών αναφοράς, που χρησιμοποιούνται για τη μέτρηση της απόδοσης μιας εφαρμογής. Η βελτιωμένη απόδοση είναι συχνά το αποτέλεσμα προσεκτικής παραμετροποίησης και βελτιστοποίησης των εφαρμογών ή των αλγορίθμων.

Συγκεντρώθηκαν αποτελέσματα αναφοράς με σκοπό την βελτιστοποίηση των πυρήνων στένσιλ 2-D για την GPU K20, τον επιταχυντή Phi και την CPU της Intel. Αυτή η αξιολόγηση βασίστηκε στον κώδικα αναφοράς που αναπτύχθηκε και χρησιμοποιεί μια διαδικασία αυτόματης ρύθμισης προκειμένου να εντοπιστεί ο βέλτιστος πυρήνας για κάθε αρχιτεκτονική-στόχο.

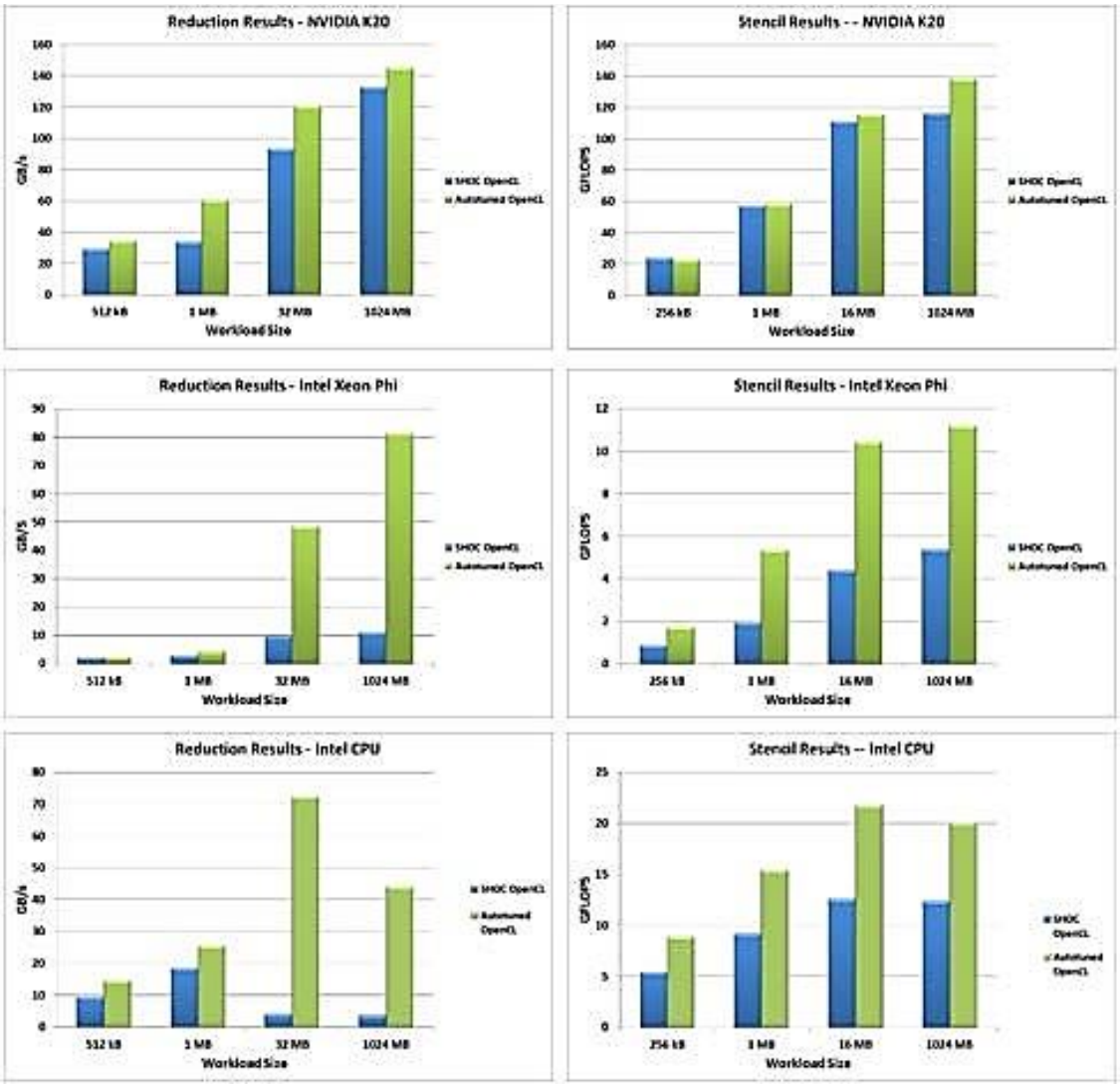
Για τη σύγκριση, αξιοποιήθηκαν ισοδύναμοι δείκτες αναφοράς, χρησιμοποιώντας διάφορες εκδόσεις του δείκτη αναφοράς SHOC. Τα βασικά αποτελέσματα αναφοράς προτείνονται για πληρότητα και δεν παρατίθενται εδώ για να αποφευχθεί η πρόκλιση της σύγχυσης.

Απεικονίστηκαν φορτία εργασίας που κάλυπταν ένα ευρύ φάσμα μεγεθών προβλημάτων. Όσον αφορά τον πυρήνα μείωσης, χρησιμοποιήθηκαν φορτία εργασίας με μεγέθη 131.072 (μικρό), 8.388.608 (μεσαίο) και 268.435.456 (μεγάλο) στοιχείων, αντιστοιχώντας σε 256 kB, 32 MB και 1024 MB μνήμης αντίστοιχα. Αυτά τα μεγέθη αντιπροσωπεύουν εκτενείς υπολογιστικές εργασίες που καλύπτουν πολλές τάξεις μεγέθους.

Αναφορικά με τον πυρήνα στένσιλ 2-D, οι φορτία εργασίας είχαν πλευρές μήκους 256 (μικρό), 2048 (μεσαίο) και 16384 (μεγάλο), αντιστοιχώντας σε 256 kB, 16 MB και 1024 MB μνήμης αντίστοιχα. Και πάλι, αυτά τα μεγέθη καλύπτουν εκτενείς υπολογιστικές εργασίες που διατρέχουν πολλές τάξεις μεγέθους.

Αξίζει να σημειωθεί ότι οι φορτία εργασίας δεν προσαρμόζονται στην αρχιτεκτονική-στόχο, αφού το μέγεθος του προβλήματος καθορίζεται από τον χαρακτήρα του προβλήματος που πρέπει να λυθεί, ανεξάρτητα από το υλικό που χρησιμοποιείται.

Τα αποτελέσματα που παρουσιάζονται στο Διάγραμμα 2 αφορούν τη σύγκριση μεταξύ του πυρήνα μείωσης και του πυρήνα στένσιλ 2-D. Σε όλα τα σενάρια, η GPU ξεπερνά τον επεξεργαστή Phi και τη CPU της Intel, ακόμα και μετά την αυτόματη ρύθμιση. Είναι σημαντικό να συγκρίνουμε τα αποτελέσματα πριν και μετά την αυτόματη ρύθμιση για να αποτιμήσουμε την πραγματική αξία της ρύθμισης. Αυτό μας επιτρέπει να κάνουμε μια δίκαιη αξιολόγηση των πυρήνων με βέλτιστη απόδοση για κάθε αρχιτεκτονική. Αυτά τα αποτελέσματα δείχνουν ότι οι εν λόγω πυρήνες είχαν την υψηλότερη απόδοση στη GPU, όπως αναμενόταν για τη συγκεκριμένη αρχιτεκτονική. Αυτό το συμπέρασμα μπορεί να βγει με βεβαιότητα, καθώς γνωρίζουμε ότι οι βέλτιστοι πυρήνες εκτελέστηκαν σε κάθε αρχιτεκτονική.



Εικ.2 Αποτελέσματα για πυρήνες αναγωγής (αριστερή στήλη) και πυρήνες στένσιλ (δεξιά στήλη) με χρήση διαφορετικών υλοποιήσεων δεικτών αναφοράς σε μια σειρά φορτίων εργασίας σε πλατφόρμες Intel CPU, Intel MIC και NVIDIA K20. Οι κλίμακες άξονα ψ διαφέρουν μεταξύ των αποτελεσμάτων μείωσης και των αποτελεσμάτων στένσιλ

Τα αποτελέσματα υπογραμμίζουν τη σημασία της βελτιστοποίησης των πυρήνων σε σημεία αναφοράς που έχουν σχεδιαστεί για φορητότητα και χρησιμοποιούνται σε συγκριτικές αξιολογήσεις.

Η χρήση σταθερών πυρήνων, που μπορεί να έχουν βελτιστοποιηθεί για μια συγκεκριμένη αρχιτεκτονική όπως οι αρχικοί πυρήνες του δείκτη αναφοράς SHOC, δεν αξιολογεί επαρκώς την απόδοση σε ετερογενείς αρχιτεκτονικές.

Τα αποτελέσματα για τη GPU και τους αρχικούς πυρήνες SHOC είναι σημαντικά πιο κοντά στον βέλτιστο πυρήνα που επιτεύχθηκε μέσω της μεθόδου αυτόματης ρύθμισης. Αυτό επιβεβαιώνει την άποψη ότι οι αρχικοί πυρήνες αρχικά αναπτύχθηκαν με σκοπό να χρησιμοποιούνται σε γραφικούς επεξεργαστές (GPU). Αντίθετα, η απόδοση στο Xeon Phi είναι σημαντικά χειρότερη με αυτούς τους πυρήνες, ιδιαίτερα όταν συγκρίνεται με τα αποτελέσματα από τη GPU. Αυτή η σύγκριση είναι αδικαιολόγητη όταν προσπαθούμε να αναδείξουμε τις διαφορές στην απόδοση σε διαφορετικές αρχιτεκτονικές. Η ανάλυση των αποτελεσμάτων αυτόματης ρύθμισης αποτελεί μια δικαιότερη σύγκριση που επιτρέπει να κατανοήσουμε πώς οι μεμονωμένοι πυρήνες θα αποδίδουν σε κάθε αρχιτεκτονική όταν χρησιμοποιούν το βέλτιστο σύνολο παραμέτρων. Τα αποτελέσματα καθιστούν σαφές ότι οι πυρήνες που εκτελούνται και στις δύο αρχιτεκτονικές αποτελούν τους βέλτιστους πυρήνες για κάθε μια από αυτές.

8.3. Σύγκριση Μεθόδων Χρονομέτρησης

Σύγκριση Μεθόδων Χρονομέτρησης: Πρόκειται για τη σύγκριση διάφορων τεχνικών και μεθόδων που χρησιμοποιούνται για την μέτρηση του χρόνου εκτέλεσης. Οι μέθοδοι αυτές μπορεί να είναι διάφορες, και η επιλογή της κατάλληλης μεθόδου προσφέρει σημαντικές επιδόσεις στη συγκριτική αξιολόγηση.

Όπως αναφέρθηκε προηγουμένως, για την αντικειμενική αξιολόγηση διαφορετικών πλατφορμών, χρησιμοποιούνται δύο διαφορετικές μεθόδους μέτρησης χρόνου. Για παράδειγμα, στην περίπτωση του δείκτη αναφοράς SHOC, χρησιμοποιούνται τόσο χρονόμετρα συμβάντων όσο και ρολόγια τοίχου για να μετρήσουν τον χρόνο εκτέλεσης των πυρήνων στο πλαίσιο των δεικτών αναφοράς Επιπέδου-1. Οι χρόνοι εκτέλεσης που προέκυψαν από αυτές τις δύο μεθόδους χρησιμοποιήθηκαν για να υπολογιστεί ο συνολικός χρόνος εκτέλεσης ενός δείκτη αναφοράς που πρέπει να εκτελεστεί σε 1 ώρα. Οι αποκλίσεις από την πραγματική απόδοση φαίνονται στον Πίνακα 1. Τα χρονόμετρα συμβάντων όχι μόνο υπερεκτιμούν την απόδοση, αλλά είναι επίσης ανεπαρκή και ανακριβή. Αντίθετα, η χρήση ρολογιού τοίχου παρέχει μια πιο ακριβή και ρεαλιστική εκτίμηση της απόδοσης των συστημάτων επιτάχυνσης σε συγκριτικές αξιολογήσεις.

Πίνακας 1

Σφάλμα πρόβλεψης μεγάλου χρόνου εκτέλεσης με τη βοήθεια δεικτών αναφοράς απόδοσης που βασίζονται σε χρονόμετρα συμβάντων έναντι χρονισμού ρολογιού τοίχου

Αρχιτεκτονική/Πυρήνας	Χρονοδιακόπτης γεγονότος	Χρονοδιακόπτης ρολογιού τοίχου
K20/Μείωση	4,6%	< 0,1%
K20/Στένσιλ	3,7%	< 0,1%
Phi/Μείωση	11,8%	1,0%
Phi/Στένσιλ	7,0%	1,8%

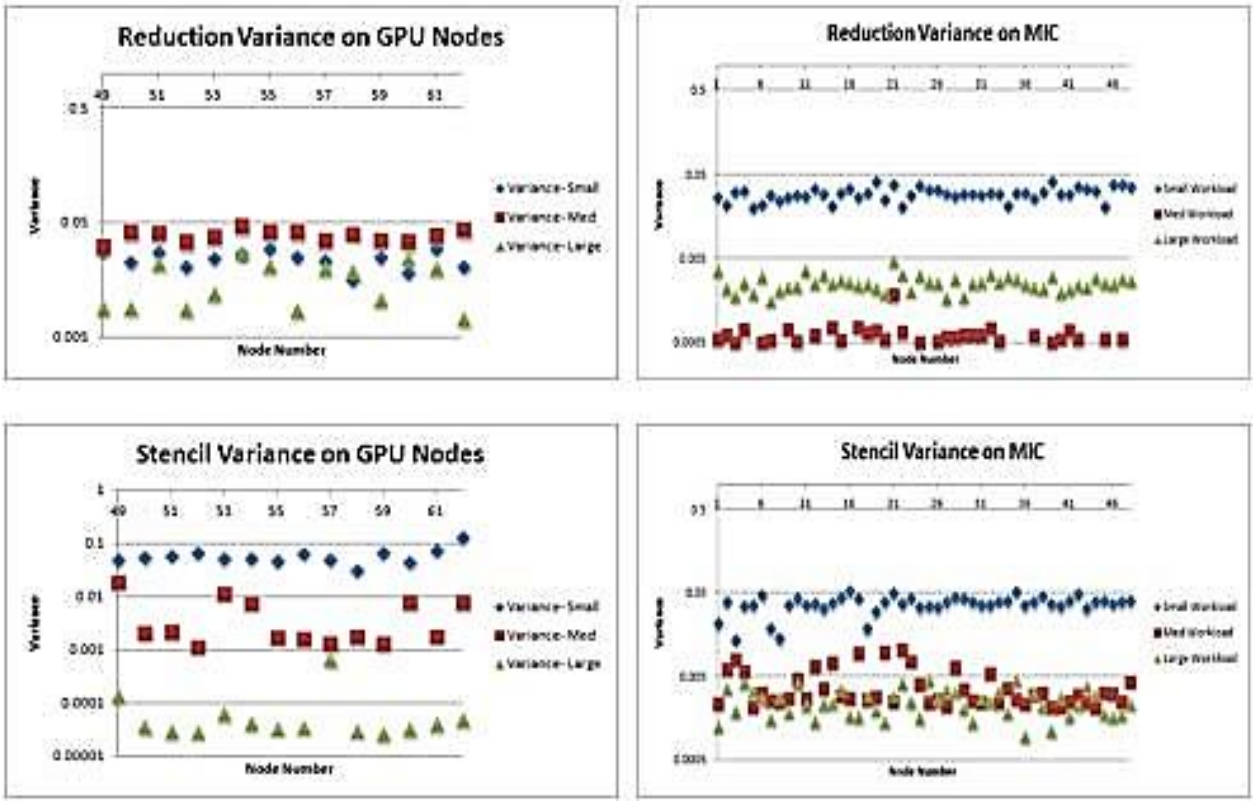
8.4. Μεταβλητότητα σε όλη την έκταση του συστήματος στην απόδοση

Μεταβλητότητα σε όλη την έκταση του συστήματος στην απόδοση: Αυτό αναφέρεται στο γεγονός ότι η απόδοση μιας εφαρμογής εξαρτάται από πολλούς παράγοντες και μπορεί να διακυμαίνει σε όλη τη διάρκεια εκτέλεσής της. Αυτή η μεταβλητότητα πρέπει να ληφθεί υπόψη κατά την αξιολόγηση.

Πραγματοποιήθηκε μέτρηση της αποκλίσεως στο χρονισμό του ρολογιού τοίχου μεταξύ διαφόρων κόμβων, καθώς αυτή η παράμετρος αποτελεί σημαντικό παράγοντα για την επεκτασιμότητα του κώδικα σε επίπεδο συμπλέγματος, ιδίως σε σχέση με την αστάθεια απόδοσης της εφαρμογής.

Δεν μπορεί να γίνει αναμφίβολα σαφές εάν μικρότεροι φορείς εργασίας με μικρότερους χρόνους εκτέλεσης θα εμφάνιζαν μεγαλύτερη διακύμανση σε σύγκριση με μεγαλύτερους φορείς εργασίας. Αν και παρατηρούνται τάσεις στην απόδοση σε πολλούς κόμβους, τα αποτελέσματα σχετικά με τον χρονισμό εκτέλεσης σε σχέση με το μέγεθος του φορέα εργασίας δεν παρέχουν σαφή κατευθύνσεις για τον πυρήνα μείωσης, είτε στη GPU K20 είτε στον επιταχυντή Phi.

Γενικά, από αυτά τα δεδομένα δεν είναι δυνατό να εξαχθούν καθολικά συμπεράσματα σχετικά με το μέγεθος του προβλήματος και την αναμενόμενη διακύμανση των χρονισμών για αυθαίρετους αλγορίθμους.



Εικ.3

Διακύμανση των αποτελεσμάτων απόδοσης για τους κωδικούς μείωσης και στένσιλ σε πολλαπλά μεγέθη φόρτου εργασίας και για το K20 και για το PHI.

Συνοψίζοντας, η φορητότητα συγκριτικής αξιολόγησης απόδοσης σε ετερογενείς υπολογιστικές πλατφόρμες απαιτεί διερεύνηση, βελτιστοποίηση, και σύγκριση πολλών παραγόντων που επηρεάζουν την απόδοση των εφαρμογών.

ΚΕΦΑΛΑΙΟ 9

Αναπτύχθηκε μια φορητή, πολυμορφική μεθοδολογία σχετικά με την απόδοση που στοχεύει στην αξιολόγηση αναδυόμενων αρχιτεκτονικών επιταχυντών. Για να επιτευχθεί αποτελεσματική φορητότητα των επιδόσεων, εφαρμόστηκε μια τεχνική που εκμεταλλεύεται παραμετροποιημένους πυρήνες σάρωσης και τέθηκε σε εφαρμογή σε GPU NVIDIA και επιταχυντές Intel Xeon Phi.

Τα αποτελέσματα δείχνουν πως οι επιδόσεις των επιταχυντών είναι ιδιαίτερα ευαίσθητες στις παραμέτρους εισόδου και τον χρόνο εκτέλεσης, καθιστώντας αναγκαία τη χρήση μεθόδων αυτόματης ρύθμισης για την επίτευξη φορητότητας της απόδοσης. Επιπρόσθετα, αυτά τα αποτελέσματα αποδεικνύουν ότι η αυτόματη ρύθμιση είναι απαραίτητη για ακριβείς συγκρίσεις της απόδοσης μεταξύ διαφορετικών αρχιτεκτονικών, διασφαλίζοντας την βέλτιστη απόδοση για κάθε αρχιτεκτονική.

Επιπλέον, αυτή η μελέτη αναλύει τις μεθόδους χρονισμού και αποδεικνύει ότι η χρήση πραγματικού χρονισμού ρολογιού τοίχου είναι πιο ακριβής και συνεκτική σε σύγκριση με τη χρήση χρονομέτρων συμβάντων για την αξιολόγηση της απόδοσης των δεικτών αναφοράς. Αυτή η προσέγγιση είναι εξαιρετική όχι μόνο για τους σχεδιαστές λογισμικού και αλγορίθμων σε υπολογιστική απόδοση, αλλά και για την πρόβλεψη της απόδοσης και των χρόνων εκτέλεσης εφαρμογών που χρησιμοποιούν λειτουργίες στένσιλ ή απαιτούν μείωση στον πυρήνα τους.

Στο μέλλον, θα ερευνηθεί η ενσωμάτωση δυνατοτήτων αυτόματου συντονισμού σε περισσότερους πυρήνες SHOC και η ενσωμάτωσή τους στη σουίτα προκειμένου να ανακαλύψουμε το βέλτιστο σύνολο παραμέτρων σε όλες τις αρχιτεκτονικές. Αυτά θα ενσωματωθούν στη σουίτα μας, επιτρέποντας τη χρήση δυναμικών μεθόδων δειγματοληψίας που χρησιμοποιούνται επίσης σε αυτή την εργασία.

Μελλοντικές έρευνες θα μπορούσαν να εξετάσουν την εφαρμογή πιο προηγμένων αλγορίθμων εξερεύνησης του χώρου παραμέτρων προκειμένου να βελτιωθεί η ισορροπία μεταξύ του χρόνου εκτέλεσης και της βέλτιστης διαμόρφωσης παραμέτρων.

9.1 Συμπεράσματα

Οι προγραμματιστές και οι μεταγλωττιστές συνεργάζονται για να αντιστοιχίσουν προβλήματα σε ακριβείς εντολές μηχανής. Η επιλογή της κατάλληλης μηχανής για ένα συγκεκριμένο πρόβλημα συνήθως είναι αποτέλεσμα των αποφάσεων που λαμβάνουν ο προγραμματιστής και ο μεταγλωττιστής. Ο προγραμματιστής επιλέγει τις υψηλού επιπέδου παραμέτρους, όπως τους αλγορίθμους, ενώ ο μεταγλωττιστής αναλαμβάνει τις χαμηλότερου επιπέδου επιλογές, όπως την αντιστοίχιση κώδικα στη μηχανή εκτέλεσης. Οι αποφάσεις και των δύο συνήθως βασίζονται σε ευρετικά μοντέλα της απόδοσης της μηχανής. Απλά μοντέλα απόδοσης χρησιμοποιούνται κατά την ανάλυση της μνήμης ή την απόφαση σχετικά με τις εντολές που εκτελούνται, καθώς και κατά την επιλογή των μετασχηματισμών που μπορεί να εφαρμοστούν στον κώδικα.

Δυσκολεύει όλο και περισσότερο για ένα μόνο μοντέλο να αντιπροσωπεύσει με ακρίβεια τις πολύπλοκες ετερογενείς μηχανές που χρησιμοποιούνται ευρέως σήμερα. Επιπλέον, σε αυτές τις μηχανές, οι αποφάσεις σε υψηλού επιπέδου σχετικά με τους αλγόριθμους και τον προγραμματισμό, καθώς και οι αποφάσεις σε χαμηλότερο επίπεδο

σχετικά με την αντιστοίχιση στη μηχανή, αλληλεπιδρούν με πολύπλοκο τρόπο για να επηρεάσουν την πραγματική απόδοση της μηχανής.

Έχουμε βρει, και τα αποτελέσματά μας δείχνουν, ότι η ταυτόχρονη εμπειρική εξερεύνηση των επιλογών αλγορίθμου και αντιστοίχισης στη μηχανή μπορεί να συντάσσει αποτελεσματικά μια σειρά προγραμμάτων για αποτελεσματική εκτέλεση σε σύγχρονες ετερογενείς παράλληλες μηχανές, συμπεριλαμβανομένων τόσο των πολυπύρηνων CPU όσο και των GPU. Οι βέλτιστες επιλογές για κάθε ετερογενή μηχανή συνήθως είναι πολύπλοκες, περιλαμβάνοντας τμήματα διαφόρων αλγορίθμων και αντιστοιχώντας τα σε διάφορους επεξεργαστές και μνήμες της μηχανής. Τα αποτελέσματα μας υποδεικνύουν επίσης ότι ολόκληρος αυτός ο χώρος επιλογών είναι σημαντικός, καθώς οι βέλτιστες τεχνικές για μια ετερογενή μηχανή μπορεί να διαφέρουν σημαντικά από αυτές για μια άλλη.

Σε ορισμένες καταστάσεις τα μοντέλα είναι ακόμα χρήσιμα. Ο χώρος αναζήτησης όλων των πιθανών επιλογών σε αλγόριθμο και αντιστοίχιση στη μηχανή είναι τεράστιος, και πολλές μεμονωμένες επιλογές έχουν βέλτιστη υποδομή, έτσι ώστε η μείωση του εμπειρικού χώρου αναζήτησης με επιλογές που βασίζονται σε μοντέλα μπορεί να είναι ταυτόχρονα αναγκαία και αποτελεσματική, ακόμη και κατά την αυτόματη προσαρμογή. Κυρίως, τα μοντέλα βοηθούν στην ανθρώπινη κατανόηση, που μπορεί να είναι αναγκαία για τη δημιουργία νέων αλγορίθμων και βελτιστοποιήσεων και νέων αρχιτεκτονικών στις οποίες θα τρέχουν. Παρ' όλ' αυτά, οι μεταγλωττιστές δεν πρέπει να είναι δεμένοι αποκλειστικά σε μοντέλα όταν αντιμετωπίζουν προγράμματα και μηχανές με όλο και περισσότερη πολυπλοκότητα.

Επίσης, ο όρος "ετερογενής" χρησιμοποιείται για να περιγράψει τη διαφορετικότητα ή ποικιλομορφία που υπάρχει σε μια υπολογιστική πλατφόρμα λόγω της χρήσης διαφορετικών τύπων επεξεργαστών και συσκευών. Στον τομέα των ετερογενών υπολογιστικών πλατφορμών, αυτό σημαίνει ότι το σύστημα χρησιμοποιεί διάφορες μορφές επεξεργαστών με διαφορετικές αρχιτεκτονικές, χαρακτηριστικά και λειτουργίες.

Η χρήση ετερογενών υπολογιστικών πλατφορμών μπορεί να προσφέρει οφέλη σε διάφορους τομείς. . Οι βασικοί παίκτες σε αυτήν την περίπτωση είναι οι επεξεργαστές κεντρικής επεξεργασίας (CPU) και οι επεξεργαστικές μονάδες γραφικών (GPU). Οι γραφικοί επεξεργαστές (GPUs) μπορεί να είναι αποτελεσματικοί στην εκτέλεση παράλληλων υπολογισμών, ενώ οι κεντρικοί επεξεργαστές (CPUs) μπορεί να έχουν καλύτερη απόδοση σε συγκεκριμένους τύπους εργασιών. Με τη χρήση ετερογενών πλατφορμών, οι εφαρμογές μπορούν να εκτελούνται στον κατάλληλο τύπο επεξεργαστή ανάλογα με τις απαιτήσεις τους, προσφέροντας έτσι βελτιωμένη απόδοση και αποτελεσματικότητα.

Οι εννοιές των επεξεργαστών CPU (Central Processing Unit) και GPU (Graphics Processing Unit) διαφέρουν ως προς τη σχεδίαση, τη λειτουργία και τον τρόπο λειτουργίας τους

Οι CPUs είναι σχεδιασμένες για γενικού σκοπού υπολογισμό, με λίγους, ισχυρούς πυρήνες και υψηλή σχετικά ταχύτητα ρολογιού. Είναι ιδανικές για σειριακούς υπολογισμούς και εργασίες που απαιτούν υψηλό επίπεδο ενδεχομένων. Η μνήμη τους περιλαμβάνει περιορισμένη αλλά γρήγορη μνήμη cache και πρόσβαση σε κύρια μνήμη RAM.

Αντίθετα, οι GPUs είναι σχεδιασμένες για παράλληλους υπολογισμούς και είναι κατάλληλες για εφαρμογές που απαιτούν υψηλή διαφοροποίηση υπολογιστικών εργασιών, όπως γραφικά και επιστημονικοί υπολογισμοί. Διαθέτουν πολλούς μικρούς πυρήνες που λειτουργούν παράλληλα, αν και με χαμηλότερη σχετικά ταχύτητα ρολογιού. Ωστόσο, η μνήμη τους είναι μεγάλη αλλά σχετικά αργή.

Ας δούμε τις βασικές διαφορές:

Σκοπός Χρήσης:

CPU: Ο CPU είναι σχεδιασμένος για γενικού σκοπού υπολογισμού. Είναι υπεύθυνος για την εκτέλεση ευρέων κατηγοριών εργασιών, όπως οι λειτουργίες του λειτουργικού συστήματος, οι εφαρμογές γραφείου, οι διαδικτυακοί περιηγητές και άλλα.

GPU: Ο GPU, αρχικά σχεδιασμένος για γραφικά, είναι εξειδικευμένος στην επεξεργασία γραφικών και παράλληλων υπολογισμών. Χρησιμοποιείται κυρίως για εφαρμογές όπως τα παιχνίδια, η γραφική επεξεργασία, οι υπολογιστικές εργασίες βαθειάς μάθησης (deep learning) και άλλες εφαρμογές παράλληλου υπολογισμού.

Αρχιτεκτονική:

CPU: Η αρχιτεκτονική των CPU εστιάζει σε λίγους αλλά πολύ ισχυρούς πυρήνες. Οι πυρήνες αυτοί είναι καλοί στον έλεγχο ροής εργασιών και την εκτέλεση εντολών σε σειριακή μορφή.

GPU: Οι GPU έχουν πολλούς μικρούς πυρήνες που λειτουργούν παράλληλα. Αυτό είναι χρήσιμο για εργασίες που μπορούν να εκτελεστούν ταυτόχρονα, όπως οι γραφικές διεργασίες και οι υπολογιστικές εργασίες παράλληλου υπολογισμού.

Επιδόσεις:

CPU: Οι CPU είναι καλές στην εκτέλεση εργασιών που απαιτούν υψηλό επίπεδο ενδεχομένων (branch prediction) και ελέγχου.

GPU: Οι GPU είναι καλές στην παράλληλη επεξεργασία μεγάλου όγκου δεδομένων, όπως στα γραφικά και τον παράλληλο υπολογισμό.

Παραλληλισμός:

CPU: Οι CPUs είναι καλές σε σειριακούς υπολογισμούς και εργασίες που απαιτούν υψηλό επίπεδο ενδεχομένων.

GPU: Οι GPUs είναι σχεδιασμένες για παράλληλους υπολογισμούς και είναι ιδανικές για εργασίες που μπορούν να εκτελεστούν ταυτόχρονα.

Μνήμη:

CPU: Οι CPUs έχουν περιορισμένη αλλά γρήγορη μνήμη cache, καθώς και πρόσβαση σε κύρια μνήμη RAM.

GPU: Οι GPUs έχουν μεγάλη, αλλά σχετικά αργή μνήμη, καθώς η προσπέλαση της RAM μπορεί να είναι πιο αργή.

Προγραμματισμός:

CPU: Ο προγραμματισμός των CPUs είναι πιο συνήθως σειριακός και γίνεται συχνά με χρήση γλωσσών όπως η C, η C++, και η Java.

GPU: Ο προγραμματισμός των GPUs γίνεται συχνά με χρήση παράλληλων προγραμματιστικών μοντέλων όπως το CUDA (για NVIDIA GPUs) ή το OpenCL.

Αυτός είναι ένας συνοπτικός πίνακας που δείχνει κάποιες από τις βασικές διαφορές μεταξύ CPU και GPU σε διάφορα χαρακτηριστικά.

Χαρακτηριστικό	CPU	GPU
Σκοπός Χρήσης	Γενικός υπολογισμός, έλεγχος ροής	Παράλληλος υπολογισμός, γραφικά, επιστημονικοί υπολογισμοί
Αρχιτεκτονική	Λίγοι ισχυροί πυρήνες	Πολλοί μικροί πυρήνες, παράλληλη εκτέλεση εργασιών
Παραλληλισμός	Κατάλληλος για σειριακούς υπολογισμούς	Κατάλληλος για παράλληλους υπολογισμούς
Ταχύτητα Ρολογιού	Υψηλή	Σχετικά χαμηλή
Μνήμη Cache	Λίγη, αλλά γρήγορη	Λίγη, αλλά γρήγορη (περιορισμένη)
Κύρια Μνήμη (RAM)	Υψηλή χωρητικότητα	Χαμηλή χωρητικότητα, αλλά γρήγορη
Εργασίες	Γενική χρήση, εφαρμογές γραφείου	Γραφικά, επιστημονικοί υπολογισμοί
Προγραμματισμός	Συνήθως σειριακός, C, C++, Java	Παράλληλος, CUDA (για NVIDIA GPUs), OpenCL

Για να επιτύχουν την τεχνολογική ετερογένεια, χρησιμοποιούνται πλατφόρμες και τεχνολογίες όπως η Heterogeneous System Architecture (HSA), το OpenCL, το CUDA, και οι επεξεργαστές AMD Accelerated Processing Unit (APU). Αυτές οι πλατφόρμες ενισχύουν τη συνεργασία μεταξύ διαφορετικών τύπων επεξεργαστών, βελτιώνοντας έτσι την απόδοση και την αποτελεσματικότητα σε ποικίλες εφαρμογές. Επιπλέον, βιβλιοθήκες όπως το TensorFlow επιτρέπουν τον προγραμματισμό πάνω σε αυτές τις πλατφόρμες για αριθμητικούς υπολογισμούς, είτε αυτοί γίνονται σε CPUs, GPUs, είτε σε άλλες μονάδες επεξεργασίας.

Τεχνολογία	Σκοπός Χρήσης	Παράδειγμα
Heterogeneous System Architecture	Ενοποιεί διάφορους τύπους επεξεργαστών σε ένα ενιαίο σύστημα.	AMD Fusion, Qualcomm Snapdragon
OpenCL	Πρότυπο για παράλληλο υπολογισμό σε διάφορους τύπους επεξεργαστών.	Χρησιμοποιείται σε CPU, GPU, FPGA
CUDA	Πλατφόρμα προγραμματισμού για παράλληλους υπολογισμούς σε GPUs της NVIDIA.	Χρησιμοποιείται σε NVIDIA GPUs
AMD Accelerated Processing Unit	Συνδυάζει CPU και GPU σε ένα ενιαίο επεξεργαστή.	AMD Ryzen APU
TensorFlow	Βιβλιοθήκη ανοιχτού κώδικα για αριθμητικούς υπολογισμούς, ευέλικτη για χρήση με διάφορες μονάδες επεξεργασίας.	Χρησιμοποιείται σε CPU, GPU, TPU

9.2 ΜΕΛΛΟΝΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

Σχεδιάζεται να ολοκληρωθεί η ανάλυση των υπόλοιπων δοκιμών στο σύνολο για πολλά μεγέθη προβλημάτων. Εκτός από τη σύγκριση της απόδοσης μεταξύ των συσκευών, μελλοντικός στόχος ειναι επίσης να αναπτυχθεί κάποια έννοια της "ιδανικής" απόδοσης για κάθε συνδυασμό δοκιμής και συσκευής, η οποία θα καθοδηγήσει τις προσπάθειες βελτίωσης της φορητότητας της απόδοσης. Επιπλέον, γίνεται λόγος να εξεταστούν επιπλέον αρχιτεκτονικές, όπως τα FPGA, DSP και APU βασισμένα στο

Radeon Open Compute, τα οποία θα συνεισφέρουν στην κατάρριψη των φραγμάτων μεταξύ της CPU και της GPU.

Κάθε πυρήνας OpenCL που παρουσιάζεται σε αυτή την εργασία, έχει ελεγχθεί χρησιμοποιώντας το Ανεξάρτητο από την Αρχιτεκτονική Χαρακτηριστικά Φόρτου Εργασίας (AIWC). Η ανάλυση χρησιμοποιώντας το AIWC βοηθά να κατανοήσουμε πώς η δομή των πυρήνων συνεισφέρει στα διάφορα χαρακτηριστικά χρόνου εκτέλεσης που παρουσιάζονται στην παρούσα εργασία και θα δημοσιευθεί στο μέλλον.

Ορισμένοι παράμετροι ρύθμισης για τις δοκιμές, όπως το τοπικό μέγεθος της ομάδας εργασίας, είναι υπόληψης για αυτόματη ρύθμιση. Έχει σχεδιαστεί να ενσωματωθεί η αυτόματη ρύθμιση στο πλαίσιο δοκιμών με σκοπό να διασφαλιστεί ότι χρησιμοποιούνται οι βέλτιστες παράμετροι για κάθε συνδυασμό κώδικα και επιταχυντή.

Μέχρι στιγμής, έχει διαπιστωθεί ότι τα διαθέσιμα σύνολα δοκιμών OpenCL δεν ήταν αρκετά πλούσια για να χαρακτηρίσουν επαρκώς την απόδοση σε μια ποικιλία εφαρμογών και υπολογιστικών συσκευών που μας ενδιαφέρουν. Έχοντας ως βάση το γεγονός ότι ένα ευέλικτο σύνολο δοκιμών είναι σε λειτουργία και μπορούν να παράγονται γρήγορα και αξιόπιστα σε μια ποικιλία επιταχυντών, σχεδιάζεται να χρησιμοποιηθούν αυτές τις δοκιμές για να αξιολογούνται οι προσεγγίσεις προγραμματισμού.

ΕΠΙΛΟΓΟΣ

Η συγκριτική αξιολόγηση της απόδοσης ετερογενών πλατφορμών (όπως διάφορων ειδών υπολογιστικές συσκευές, επεξεργαστές, γραφικές κάρτες, κλπ.) απαιτεί τη δημιουργία ενός φορητού ετερογενούς κώδικα απόδοσης για την επίτευξη της σχετικής αξιολόγησης. Αυτό συμβαίνει για διάφορους λόγους:

Ετερογενότητα των πλατφορμών: Οι διάφορες πλατφόρμες έχουν διαφορετικές αρχιτεκτονικές, χαρακτηριστικά και περιορισμούς. Ο κώδικας που εκτελείται αποτελεσματικά σε μία πλατφόρμα ενδέχεται να μην είναι αποδοτικός σε μία άλλη.

Βελτιστοποίηση για την απόδοση: Για να αξιολογήσετε αντικειμενικά την απόδοση διαφόρων πλατφορμών, πρέπει να δημιουργήσετε κώδικα που είναι βελτιστοποιημένος για κάθε πλατφόρμα. Αυτό σημαίνει ότι ο κώδικας πρέπει να αξιοποιεί τα χαρακτηριστικά και τους πόρους κάθε πλατφόρμας για την επίτευξη της υψηλότερης απόδοσης.

Διαφορετικοί γλωσσικοί περιορισμοί: Οι διάφορες πλατφόρμες μπορεί να υποστηρίζουν διαφορετικές γλώσσες προγραμματισμού, συνεπώς ο κώδικας απόδοσης πρέπει να προσαρμόζεται σε αυτές τις γλωσσικές περιορισμούς.

Στόχος αξιολόγησης: Η αξιολόγηση μπορεί να έχει διάφορους στόχους, όπως τη μέτρηση της απόδοσης σε συγκεκριμένες εργασίες (όπως γραφικά, υπολογισμοί, μηχανική μάθηση κλπ.). Κατά συνέπεια, ο κώδικας πρέπει να προσαρμόζεται ανάλογα για να εκτελεί αυτές τις εργασίες.

Ο φορητός ετερογενής κώδικας απόδοσης πρέπει να είναι προσεκτικά σχεδιασμένος και βελτιστοποιημένος για να εκμεταλλεύεται τις δυνατότητες κάθε πλατφόρμας. Η δημιουργία τέτοιου κώδικα είναι συχνά πολύπλοκη και απαιτεί εξειδικευμένες γνώσεις στον προγραμματισμό και τις αρχιτεκτονικές των πλατφορμών.

Κατάλογος Συμβόλων, Συντομογραφιών και Ακρωνύμιων

1-D	Μονοδιάστατο
2-D	Δισδιάστατο
API	Διεπαφή Προγραμματισμού Εφαρμογών
CPU	Κεντρική Μονάδα Επεξεργασίας.
ECC	Κώδικας Διόρθωσης Σφάλματος
GEMM	Γενικός Πολλαπλασιασμός Πίνακα-Πίνακα
GPU	Μονάδα Επεξεργασίας Γραφικών
HPC	Πληροφορική Υψηλών Επιδόσεων
MIC	Πολλές Ολοκληρωμένες Βασικές Αρχιτεκτονικές
MPI	Διεπαφή Διαβίβασης Μηνυμάτων
SDK	Κιτ Ανάπτυξης Λογισμικού
SHOC	Κλιμακούμενη Ετερογενής Υπολογιστική
XML	Επεκτάσιμη Γλώσσα Σήμανσης

Βιβλιογραφία

Andreas Kl¨ocknera, Nicolas Pintob,c, Yunsup Leed, Bryan Catanzarod, Paul Ivanove,f, Ahmed Fasihg, PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation

Klaus Kofler, Ivan Grasso, Biagio Cosenza and Thomas Fahringer. An Automatic Input-Sensitive Approach for Heterogeneous Task Partitioning

Phitchaya Mangpo Phothilimthana Jason Ansel Jonathan Ragan-Kelley Saman Amarasinghe, Portable Performance on Heterogeneous Architectures

Jamie K Infantolino, James A Ross, Song J Park, Dale R Shires, Thomas M Kendall, and David A Richie, Performance-Portable Benchmarking Methods for Investigating Heterogeneous Computing Platforms

Beau Johnston, Josh Milthorpe, Dwarfs on Accelerators: Enhancing OpenCL Benchmarking for Heterogeneous Computing Architectures

Towards Scalable Adaptive Mesh Refinement on Future Parallel Architectures by David Alexander Beckingsale

An Automatic Input-Sensitive Approach for Heterogeneous Task Partitioning by Klaus Kofler, Ivan Grasso, Biagio Cosenza and Thomas Fahringer

[1]. Barker KJ, Davis K, Hoisie A, Kerbyson DJ, Lang M, Pakin S, Sancho JC. Entering the petaflop era: the architecture and performance of roadrunner. In Proceedings of the 2008 ACM/IEEE Conference on Super-Computing; IEEE Press; 2008; Austin, TX. p. 1.

[2]. Advanced Micro Devices, Inc. The industry-changing impact of accelerated computing. Austin (TX): Advanced Micro Devices, Inc.; 2018 [accessed 2014]. http://sites.amd.com/jp/Documents/AMD_fusion_Whitepaper.

[3]. Chung ES, Milder PA, Hoe JC, Mai K. Single-chip heterogeneous computing: does the future include custom logic, FPGAs, and GPGPUs? In: Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture; IEEE Computer Society; 2010; Atlanta, GA. p. 225–236.

[4]. Hameed R, Qadeer W, Wachs M, Azizi O, Solomatnikov A, Lee BC, Richardson S, Kozyrakis C, Horowitz M. Understanding sources of

inefficiency in general-purpose chips. In: ACM SIGARCH Computer Architecture News. Vol. 38, ACM; 2010. p. 37–47.

[5]. Feng W-c, Lin H, Scogland T, Zhang J. OpenCL and the 13 dwarfs: a work in progress. In: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering; ACM; 2012; Boston, MA. p. 291–294.

[6]. Danalis A, Marin G, McCurdy C, Meredith JS, Roth PC, Spafford K, Tipparaju V, Vetter JS. The scalable heterogeneous computing (SHOC) benchmark suite. In: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units; ACM; 2010; Pittsburgh, PA. p. 63–74.

[6] W. J. Dally, P. Hanrahan, M. Erez, T. J. Knight, F. Labont´e, J. H. Ahn, N. Jayasena, U. J. Kapasi, A. Das, and J. Gummaraju. Merrimac: Supercomputing with streams. In Proc. of the ACM/IEEE SC2003 Conference (SC'03), volume 1, 2003.

[7]. Che S, Boyer M, Meng J, Tarjan D, Sheaffer JW, Lee S-H, Skadron K. Rodinia: A benchmark suite for heterogeneous computing. In: Workload Characterization, IISWC 2009. IEEE International Symposium on IEEE; 2009; Austin, TX. p. 44–54.

[7] J. de Guzman. The Boost Spirit Parser Generator Framework, 2008. URL http://spirit.sourceforge.net/.

[8]. Stratton JA, Rodrigues C, Sung IJ, Obeid N, Chang LW, Anssari N, Liu GD, Hwu WW. Parboil: a revised benchmark suite for scientific and commercial throughput computing. Urbana–Champaign (IL): University of Illinois Center for Reliable and High-Performance Computing; 2012. Report No.: IMPACT12-01.

[8] B. Eich. JavaScript at ten years. In Proceedings of the tenth ACM SIGPLAN international conference on Functional programming, page 129. ACM, 2005. ISBN 1595930647. URL http://ecmascript.org.

[9]. Dongarra J, Luszczek P. Linpack benchmark. In: Padua D, editor. Encyclopedia of parallel computing. New York (NY): Springer; 2011. p. 1033– 1036.

[10]. Richie DA, Ross JA, Park SJ, Shires DR. Ray-tracing-based geospatial optimization for heterogeneous architectures enhancing situational awareness. In: Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on IEEE; 2013; Sydney, NSW, Australia. p. 81–86.

[11]. Richie D, Ross J, Ruloff J, Park S, Pollock L, Shires D. Investigation of parallel programmability and performance of a Calxeda arm server using OpenCL. In: Euro-Par 2013: Parallel Processing Workshops; 2014; Berlin–Heidelberg (Germany): Springer. p. 865–874.

[11] D. Flanagan and Y. Matsumoto. The Ruby programming language. O'Reilly, 2008. URL http://www.ruby-lang.org.

[12] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. Proc. IEEE, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[12]. Du P, Weber R, Luszczek P, Tomov S, Peterson G, Dongarra J. From CUDA to OpenCL: towards a performance-portable solution for multi-platform GPU programming. Para Comp. 2012;38(8):391–407.

[13]. Fang J, Varbanescu AL, Sips H. A comprehensive performance comparison of CUDA and OpenCL. In: Parallel Processing (ICPP), 2011 International Conference on IEEE; 2011; Taipei City, Taiwan. p. 216–225.

[14]. McIntosh-Smith S, Boulton M, Curran D, Price J. On the performance portability of structured grid codes on many-core computer architectures. In: Kunkel JM, Ludwig T, Meuer HW, editors. Supercomputing. Proceedings of ISC 2014, 29th International Conference; 2014 June 22–26; Leipzig, Germany. Switzerland: Springer International Publishing; 2014. p. 53–75.

K. O. W. Group. The OpenCL 1.0 Specification. Khronos Group, Beaverton, OR, Dec. 2008.

[15]. Whaley RC, Petitet A, Dongarra JJ. Automated empirical optimizations of software and the atlas project. Para Comp. 2001;27(1):3–35.

[16]. Abu-Sufah W, Karim AA. Auto-tuning of sparse matrix-vector multiplication on graphics processors. In: Kunkel JM, Ludwig T, Meuer HW. Supercomputing. Proceedings of ISC 2013, 28th International Supercomputing Conference; 2013 June 16–20; Leipzig, Germany. Switzerland: Springer International Publishing; 2013. p. 151–164.

[17]. Li Y, Dongarra J, Tomov S. A note on auto-tuning GEMM for GPUS. In: Allen G, editor. Computational science. Proceedings of ICCS 2009. Berlin, Germany: Springer-Verlag; 2009. p. 884–892.

[18]. Phothilimthana PM, Ansel J, Ragan-Kelley J, Amarasinghe S. Portable performance on heterogeneous architectures. ACM SIGPLAN Notices. 2013; 48(4):431–444.

[19]. Pennycook SJ, Hammond SD, Wright SA, Herdman J, Miller I, Jarvis SA. An investigation of the performance portability of OpenCL. J Para Distr Comp. 2013;73(11):1439–1450.

[19] R. Ierusalimschy. Programming in Lua. Roberto Ierusalimschy, 2006. ISBN 8590379825. URL http://www.lua.org.

[20]. Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Plishker WL, Shalf J, Williams SW, et al. The landscape of parallel computing research: a view from Berkeley. Berkeley (CA): University of California EECS Department; 2006. Report No.: UCB/EECS-2006-183.

[26] C. Lejdfors and L. Ohlsson. Implementing an embedded GPU language by combining translation and

generation. In Proceedings of the 2006 ACM symposium on Applied computing, pages 1610–1614, 2006.

[27] C. Lengauer, D. Batory, C. Consel, and M. Odersky, editors. Domain-Specific Program Generation.

Number 3016 in Lecture Notes in Computer Science. Springer-Verlag, 2004.

[29] J. McCarthy. LISP 1.5 Programmer's Manual. MIT Press, Aug. 1962.

[30] M. McCool and R. Inc. Data-parallel programming on the Cell BE and the GPU using the RapidMind development platform. In Proc. GSPx Multicore Applications Conference, 2006.

M. McCool and S. D. Toit. Metaprogramming GPUs with Sh. A K Peters, Wellesley MA, 2004.

[36] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kr¨uger, A. E. Lefohn, and T. J. Purcell. A survey of General-Purpose computation on graphics hardware. Computer Graphics Forum, 26(1):80–113, 2007.

[44] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many-core x86 architecture for visual computing. ACM Trans. Graph., 27(3):1–15, 2008. ISSN 0730-0301.

C. Prud'homme. A domain specific embedded language in C++ for automatic differentiation, projection, integration and variational formulations. Sci. Prog., 14(2):81–110, 2006.

J. Reynders, P. Hinker, J. Cummings, S. Atlas, S. Banerjee, W. Humphrey, K. Keahey, M. Srikant, and M. Tholburn. POOMA: A Framework for Scientific Simulation on Parallel Architectures. In G. Wilson and P. Lu, editors, Parallel Programming using C++. MIT Press, 1996.

D. Tarditi, S. Puri, and J. Oglesby. Accelerator: using data parallelism to program GPUs for generalpurpose uses. In Proceedings of the 2006 ASPLOS Conference, volume 40, page 325–335, 2006.

[49] G. van Rossum et al. The Python programming language, 1994. URL http://python.org.

T. L. Veldhuizen. C++ templates are turing complete. Technical report, Indiana University Computer Science, 2003.

T. L. Veldhuizen and M. E. Jernigan. Will C++ be faster than Fortran? In Proceedings of the 1st International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'97), Lecture Notes in Computer Science. Springer-Verlag, 1997.

[52] S. Venkatasubramanian. The graphics card as a stream computer. In SIGMODDIMACS Workshop on Management and Processing of Data Streams, 2003.

J. Wedekind, B. Amavasai, K. Dutton, and M. Boissenin. A machine vision extension for the Ruby programming language. In Int. Conf. on Information and Automation, pages 991–996, 2008.

R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimizations of software and the ATLAS project. Par. Comp., 27:3–35, 2001