ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

01/09/2013

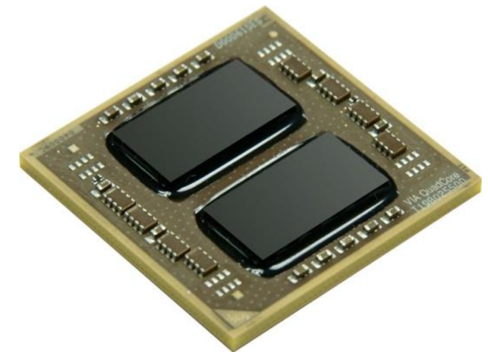
Επιβλέπων Καθηγητής
Δασυγένης Μηνάς

ΥΠΕΡΒΑΘΜΩΤΟΙ ΕΠΕΞΕΡΓΑΣΤΕΣ

http://arch.icte.uowm.gr

Συστήματα Παράλληλης & Κατανεμημένης Επεξεργασίας

Ντάγιου Άννα 432
Πούλιου Χρύσα 447



ΠΕΡΙΕΧΟΜΕΝΑ

- Εισαγωγή ……………………………………………………….6
- Από το ακολουθιακό στο παράλληλο μοντέλο εκτέλεσης ……….7
- Το παράλληλο μοντέλο εκτέλεσης ……………………………..8
- Αρχιτεκτονικές superscalar ……………………………….......9
- Superscalar pipeline ………………………………………………12
- Αύξηση ταχύτητας επεξεργαστή και μνήμης ………..……………13
- Ιεραρχία μνήμης………………………………………………………14
- Processor front end………………………………………..………15
- Processor back end…………………………………………………18
- Υψηλής απόδοσης superscalar επεξεργαστές……………………21
- Εξαρτήσεις ελέγχου στην παράλληλη εκτέλεση…………………..23

ΠΕΡΙΕΧΟΜΕΝΑ

- Διαδικασία ολοκλήρωσης μιας εντολης……………………………..26
- Αρχιτεκτονική τυπικού επεξεργαστή superscalar ………………...28
- Ανάκληση εντολών και πρόβλεψη διακλαδώσεων………………..30
- Πρόβλεψη διακλαδώσεων ………………………………………31
- Αποκωδικοποίηση, μετονομασία, προώθηση εντολών ………….35
- Εκτέλεση εντολών και παράλληλη εκτέλεση ……………………….39
- Τρόποι για την οργάνωση των αποθηκευτικών χώρων ………….40
- Λειτουργίες μνήμης και ανανέωση αρχιτεκτονικής κατάστασης ...43
- Μέθοδοι για ανάκτηση μιας ακριβής κατάστασης ………………...46
- Ο ρόλος των προγραμμάτων στην αύξηση της απόδοσης ……...47
- Βελτιστοποίηση υπερβαθμωτού προγραμματισμού ……………...48

ΠΕΡΙΕΧΟΜΕΝΑ

- Πολλαπλασιασμός πινάκων ……………………………………..49
- Παραδείγματα βελτιστοποίησης σε υπερβαθμωτούς επεξεργαστές Intel ……………………………………………50
- Περιγραφείς πινάκων ……………………………………………71
- Non-cache και cache τεχνικές………………………………………72
- Τεχνική μεταφοράς πινάκων …………………………………………73
- Υπέρ νημάτωση ……………………………………………………..91
- Ανάθεση Socket ……………………………………………………..97
- Τεχνική Tag Team ……………………………………………………105
- Συναρτήσεις για XMM INTRINSIC …………………………111
- Σύνοψη αποτελεσμάτων ………………………………………120

ΠΕΡΙΕΧΟΜΕΝΑ

- CILK++ σε INTEL επεξεργαστές 125
- Συντελεστής Κλιμάκωσης 127
- Γενικά συμπεράσματα 133
- Superscalar Programming - Εφαρμογές 141
- Σύνοψη-Συμπεράσματα 145
- Βιβλιογραφία 146

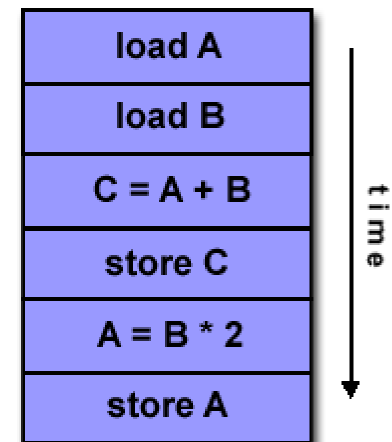
ΕΙΣΑΓΩΓΗ

- Ένας τυπικός superscalar επεξεργαστής πραγματοποιεί ανάκληση και αποκωδικοποίηση πολλών εντολών σε κάθε κύκλο ρολογιού. Μέρος της διαδικασίας ανάκλησης εντολών είναι και η ***πρόβλεψη*** του αποτελέσματος των υπό συνθήκη εντολών διακλάδωσης, προκειμένου να υπάρχει μια συνεχής, ***μη διακοπτόμενη ροή εντολών***. Η ροή εντολών που εισέρχεται στον επεξεργαστή υφίσταται ανάλυση για τον καθορισμό τυχόν εξαρτήσεων δεδομένων μεταξύ των εντολών και έπειτα οι εντολές κατανέμονται στις λειτουργικές μονάδες. Στη συνέχεια, οι εντολές αποστέλλονται για ***παράλληλη εκτέλεση***, με βάση συνήθως την διαθεσιμότητα των δεδομένων που αποτελούν τους τελεσταίους των εντολών και όχι την αρχική ακολουθία των εντολών στο πρόγραμμα.
- Όταν οι εντολές ολοκληρωθούν, τα αποτελέσματά τους τοποθετούνται ξανά στην ***αρχική σειρά*** που υποδηλώνει το πρόγραμμα έτσι ώστε η κατάσταση του επεξεργαστή να ανανεώνεται σύμφωνα με την κανονική ροή εντολών του προγράμματος. Με αυτόν τον τρόπο ο επεξεργαστής μπορεί να χειριστεί χωρίς λάθη τις περιπτώσεις όπου συμβαίνουν ***διακοπές*** (interrupts). Επειδή οι ξεχωριστές εντολές αποτελούν οντότητες που εκτελούνται παράλληλα, οι superscalar επεξεργαστές εκμεταλλεύονται την ιδιότητα των προγραμμάτων που ονομάζεται παραλληλισμός επιπέδου εντολών (instruction level parallelism, ILP).

ΤΟ ΑΚΟΛΟΥΘΙΑΚΟ ΜΟΝΤΕΛΟ ΕΚΤΕΛΕΣΗΣ

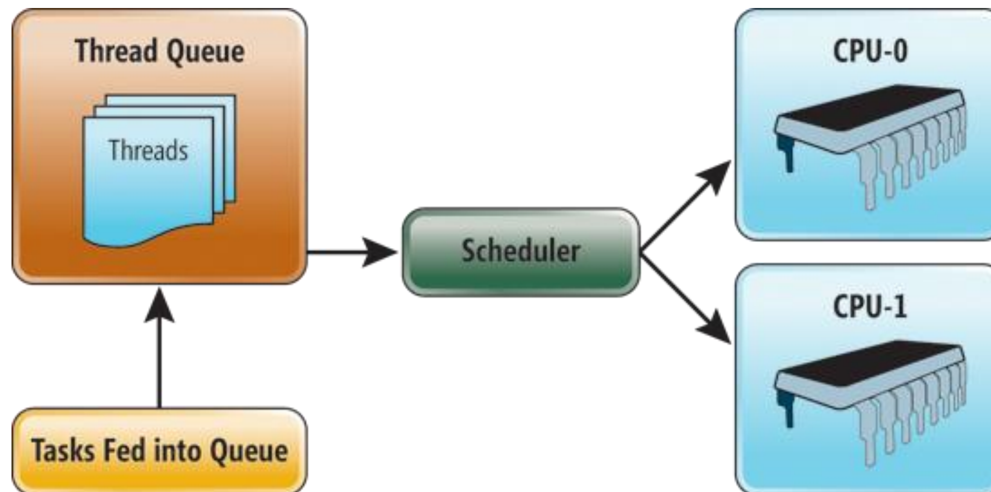
- Οι υπολογιστές αρχικά υλοποιήθηκαν με βάση το ***ακολουθιακό μοντέλο εκτέλεσης*** των προγραμμάτων. Με βάση το μοντέλο αυτό πραγματοποιείται ανάκληση μίας εντολής από τη μνήμη, η εντολή εκτελείται, και κατά τη διάρκεια της εκτέλεσής της ο επεξεργαστής μπορεί να φορτώσει ή να αποθηκεύσει δεδομένα στην κύρια μνήμη καθώς και να χρησιμοποιήσει τους καταχωρητές που διαθέτει. Μετά την εκτέλεση της εντολής, γίνεται ***ανάκληση*** της επόμενης εντολής κοκ.
  Η ακολουθιακή επεξεργασία των εντολών ***διακόπτεται*** όταν εκτελείται μία εντολή άλματος ή υπό συνθήκη διακλάδωσης, και η εκτέλεση να μεταφέρεται στην εντολή που ορίζει το πρόγραμμα.

- Σε περίπτωση που η εκτέλεση προγράμματος πρέπει να διακοπεί και να συνεχιστεί αργότερα, η ακριβής κατάσταση του επεξεργαστή τη στιγμή της διακοπής πρέπει να αποθηκευτεί.



ΤΟ ΠΑΡΑΛΛΗΛΟ ΜΟΝΤΕΛΟ ΕΚΤΕΛΕΣΗΣ

- Οι ***superscalar επεξεργαστές***, για να επιτύχουν υψηλότερη απόδοση, διαφοροποιούνται από την ακολουθιακή εκτέλεση, καθώς πολλές εργασίες πρέπει να πραγματοποιηθούν παράλληλα. Ένας σύγχρονος superscalar επεξεργαστής δημιουργεί μια παράλληλη έκδοση του προγράμματος η οποία θα χαρακτηρίζεται από ***υψηλότερη απόδοση***. Ωστόσο, ο επεξεργαστής διατηρεί την εξωτερική εικόνα της ακολουθιακής εκτέλεσης του προγράμματος

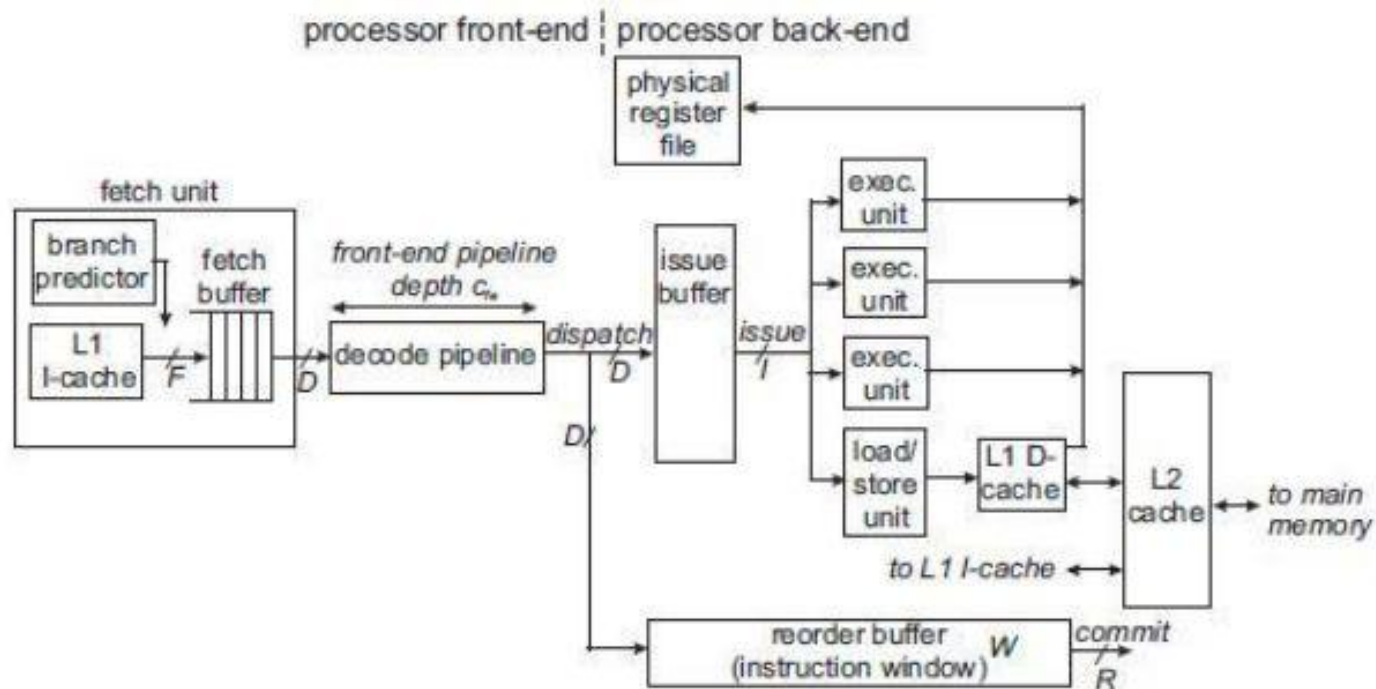


- Οι περισσότεροι επεξεργαστές γενικού σκοπού και υψηλής απόδοσης σήμερα είναι βασισμένοι στο σχεδιαστικό πρότυπο του υπερβαθμωτού επεξεργαστή με εκτέλεση εκτός σειράς. Η αρχιτεκτονική αυτή εκμεταλλεύεται σε όσο γίνεται μεγαλύτερο βαθμό το παραλληλισμό επιπέδου εντολών. Ως ***βασική ιδέα λειτουργίας*** έχει το να επιτρέπει σε κάθε στάδιο της ομοχειρίας(pipeline) του επεξεργαστή να χειρίζεται περισσότερες από μία εντολές και να μπορεί να εκτελεί εντολές με διαφορετική σειρά από αυτή του προγράμματος. Λειτουργώντας έτσι ενώ παράλληλα χωρίζοντας την εκτέλεση της εντολής σε ***πολλαπλά στάδια*** ώστε να μπορούν να αυξήσουν τη συχνότητα λειτουργίας, οι superscalar επεξεργαστές είναι μάλλον από τις καλύτερες σχεδιάσεις γενικού σκοπού σήμερα.

- Παραδοσιακά χρησιμοποιούνται πιο απλοί , με εκτέλεση εντός σειράς επεξεργαστές σε ενσωματωμένα συστήματα (embedded systems) με σκοπό το ***μικρότερο κόστος*** αλλά και τη ***κατανάλωση ενέργειας*** , αυτό τείνει να αλλάξει αφού οι σύγχρονες εφαρμογές έχουν όλο και περισσότερες απαιτήσεις.

ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ SUPERSCALAR(1)

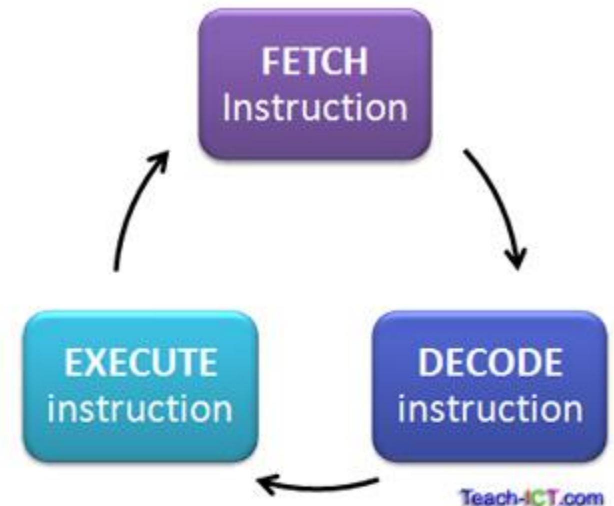
Σχέδιο ενός generic superscalar processor



ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ SUPERSCALAR(2)

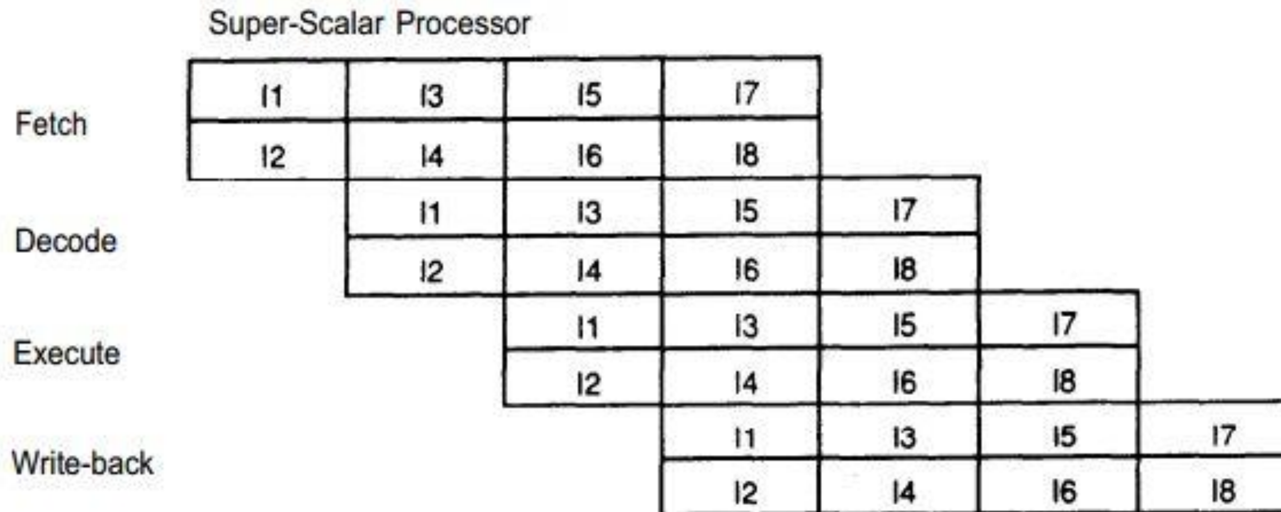
Στην προηγούμενη εικόνα απεικονίζεται ένας superscalar επεξεργαστής στη γενική του μορφή. Μπορούμε διαισθητικά να χωρίσουμε τη λειτουργία του υπολογιστή σε τρία βασικά μέρη:

- το front-end το οποίο προμηθεύει με νέες εντολές προς εκτέλεση τον επεξεργαστή που επιτελεί δύο βασικές λειτουργίες οι οποίες είναι:
  - Η ανάκληση εντολών από την μνήμη (fetch)
  - Η κωδικοποίηση των εντολών (decoding)
- το back-end, το οποίο είναι υπεύθυνο για την εκτέλεση των εντολών αυτών και την ενημέρωση της κατάστασης του συστήματος με τα αποτελέσματα των εκτελεσθέντων εντολών .
- την ιεραρχία μνήμης η οποία έχει σκοπό να βελτιστοποιήσει την αποδοτικότητα του συστήματος μνήμη



SUPERSCALAR PIPELINE

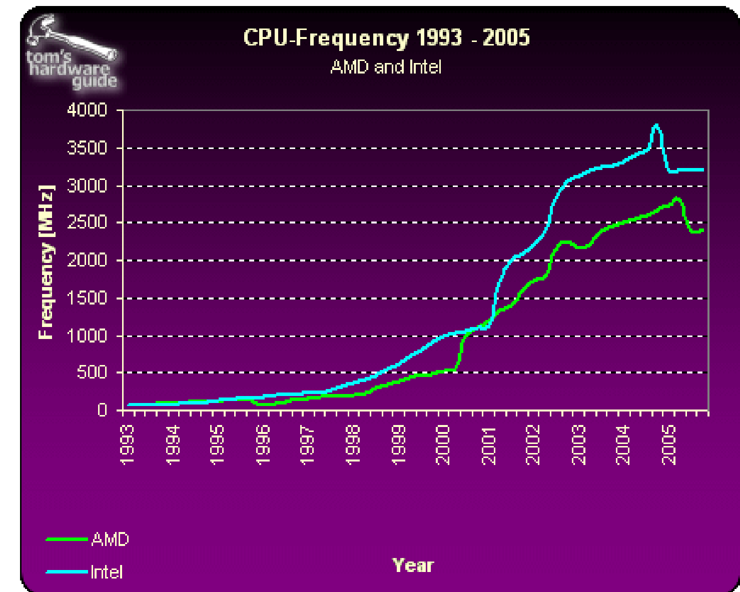
Η ***superscalar pipeline*** μπορεί να χειριστεί πάνω από μία εντολές σε διαφορετικά επίπεδα pipeline και μέσα στα ίδια επίπεδα pipeline.Η χωρητικότητα μίας pipeline ενός superscalar επεξεργαστή είναι δύο εντολές ανά κύκλο.



ΑΥΞΗΣΗ ΤΑΧΥΤΗΤΑΣ ΕΠΕΞΕΡΓΑΣΤΗ ΚΑΙ ΜΝΗΜΗΣ

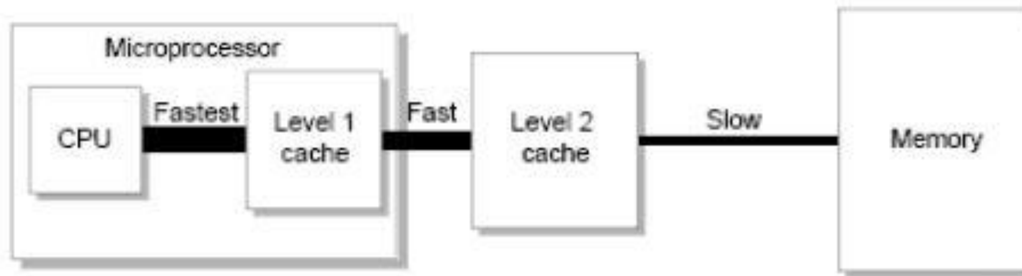
- Με διάφορες τεχνικές (χωρισμός του pipeline σε πολλαπλά στάδια, βελτίωση της τεχνολογίας πυριτίου), οι σχεδιαστές συστημάτων έχουν καταφέρει να αυξήσουν τη ***συχνότητα λειτουργίας*** και τη ***ταχύτητα*** του επεξεργαστή, όμως δε μπορούμε να πούμε το ίδιο και για τη μνήμη. Αυτό όμως σημαίνει ότι κάθε ***πρόσβαση*** στη κύρια μνήμη απαιτεί και ένα μεγάλο χρόνο αναμονής για τον επεξεργαστή. Το εύλογο θα ήταν να κατασκευάζονται πολύ ***γρηγορότερες*** μνήμες, όμως κατά κανόνα όσο γρηγορότερη η μνήμη, τόσο ακριβότερη και μικρότερη σε μέγεθος.

- Όμως συμβαίνει το εξής: είτε σε κώδικα assembly γραμμένο απευθείας από άνθρωπο , είτε παραγόμενο από κάποια μεταγλώττιση κάποιας higher-level γλώσσας παρατηρείται πως όταν χρησιμοποιείται μια θέση μνήμης που περιέχει instructions ή δεδομένα, αυτή είναι πολύ πιθανόν να χρησιμοποιηθεί ***ξανά*** πολύ σύντομα στο μέλλον. Επιπροσθέτως , είναι πολύ πιθανό ότι θα χρειασθεί μια θέση μνήμης κοντά σε αυτή. Αυτό το φαινόμενο ονομάζεται ***χρονική και χωρική τοπικότητα***.



ΙΕΡΑΡΧΙΑ ΜΝΗΜΗΣ

- Αυτό ακριβώς εκμεταλλεύεται η ιεραρχία μνήμης, προσπαθώντας να επιτύχει το 99% της ταχύτητας μιας ***πολύ γρήγορης*** και ***ακριβής*** μνήμης , στο 1% του κόστους της.



- Χρησιμοποιεί δηλαδή ***πολλαπλά επίπεδα*** μνήμης, το καθένα μικρότερο από το επόμενο σε χωρητικότητα , αλλά και πολύ ταχύτερα. Έτσι όποτε ο επεξεργαστής θέλει να προσπελάσει μια θέση μνήμης , ανατρέχει στο γρηγορότερο επίπεδο της ιεραρχίας. Εάν βρει τα δεδομένα που θέλει τα παίρνει σε πολύ ***μικρό χρόνο*** , αλλιώς ανατρέχει στο επόμενο επίπεδο συνεχίζοντας με τον ίδιο τρόπο μέχρι την κύρια μνήμη. Όποτε ***δε βρίσκει*** τα δεδομένα (cache miss) σε ένα επίπεδο μνήμης σημαίνει και επιπλέον χρόνο για να το βρει στο επόμενο, ενώ για το ποια ακριβώς δεδομένα μας συμφέρει να βρίσκονται κάθε στιγμή στην cache έχουν γίνει εκτενέστατες μελέτες που δεν είναι του παρόντος.

PROCESSOR FRONT END(1)

- Το ***front-end*** του επεξεργαστή επιτελεί δύο βασικές λειτουργίες. Η μία είναι η ***ανάκληση*** των εντολών από τη μνήμη και η άλλη είναι η ***μετατροπή*** των εντολών σε μια μορφή ευκολότερη για τον επεξεργαστή να διαχειριστεί. Την ανάκληση των εντολών αναλαμβάνει η μονάδα ανάκλησης η οποία ***διαβάζει εντολές*** από τη μνήμη. Για την ακρίβεια δεν διαβάζει εντολές από την κύρια μνήμη του επεξεργαστή, αφού αυτή είναι εξαιρετικά ***αργή*** εάν τη συγκρίνει κανείς με τη ταχύτητα του processor core.

- Έτσι, η μονάδα ανάκλησης προσπαθεί να διαβάσει από την μνήμη κορυφής (cache) εντολών, δηλαδή την I-cache του επιπέδου 1. Εάν δε βρει εκεί την εντολή που ψάχνει προχωράει στο ***επόμενο επίπεδο*** της ιεραρχίας, φτάνοντας στη χειρότερη περίπτωση και μέχρι την κύρια μνήμη του υπολογιστή.

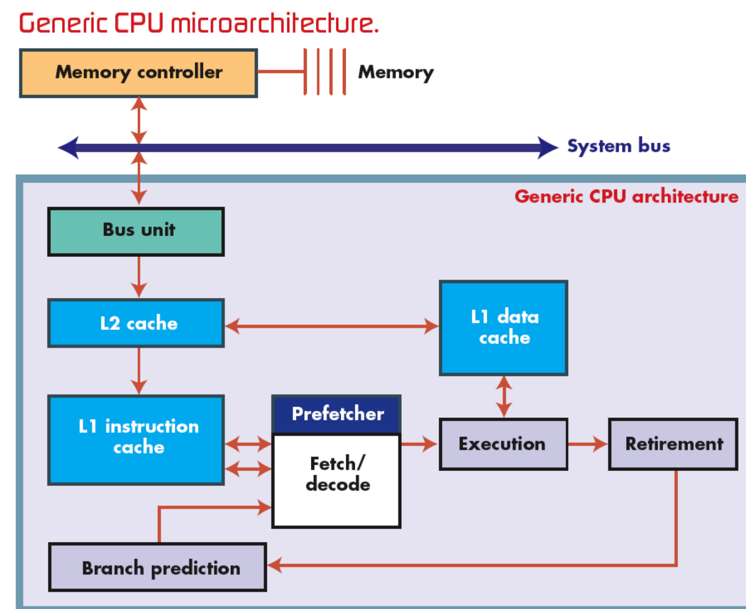


Figure 1

PROCESSOR FRONT END(2)

- Ιδεατά το front-end του επεξεργαστή πρέπει να μπορεί να προσφέρει τόσες εντολές στο execution core του επεξεργαστή όσες αυτό χρειάζεται και με ***σταθερό τρόπο***. Σε αυτό δρουν όμως ***ανασταλτικά*** οι εντολές του control flow (conditionals, returns, calls) οι οποίες αλλάζουν τη ροή του προγράμματος.

- Αυτό συμβαίνει γιατί από τη στιγμή που θα γίνει το ***fetch*** της εντολής αυτής μέχρις ότου να ικανοποιηθούν τα ***dependencies*** της εντολής, να εκτελεστεί και να φτάσει στο στάδιο του commit όπου μπορούμε να γνωρίζουμε με ακρίβεια τη θέση μνήμης στην οποία συνεχίζεται το ***execution flow***, μεσολαβεί ένα χρονικό διάστημα κατά το οποίο η fetch unit δε γνωρίζει ποιες εντολές να ανακαλέσει από τη μνήμη.

- Η λύση σε αυτό λέγεται **πρόβλεψη διακλαδώσεων**, μια οικογένεια τεχνικών με τις οποίες βάσει του παρελθόντος του dynamic execution stream γίνεται η ***πρόβλεψη*** του αποτελέσματος μιας εντολής flow control χωρίς να εκτελεστεί. Με αυτό τον τρόπο συνεχίζετε η ανάκληση και η εκτέλεση εντολών από τη μνήμη, και σε περίπτωση που αποδειχτεί με τη πραγματική εκτέλεση του control statement πως η πρόβλεψη είναι ***λάθος***, απλά δε λαμβάνονται τα αποτελέσματα των εντολών που εκτελέσθηκαν υποθετικά και συνεχίζετε η εκτέλεση στο ***σωστό*** path.

PROCESSOR FRONT END(3)

- Ο αριθμός των εντολών που μπορεί να φερθεί σε κάθε κύκλο από τη μνήμη η μονάδα ανάκλησης πρέπει να είναι τουλάχιστον ***ίσος ή και μεγαλύτερος*** από το πλάτος του επεξεργαστή . Το fetch width πρεπει να είναι διπλάσιο από το πλάτος του υπόλοιπου επεξεργαστή, ώστε παρά τα προβλήματα πρόσβασης στη cache και τα taken branches που διακόπτουν τη ροή των εντολών από τη cache να έχουν μικρότερο ***αντίκτυπο*** στην ***απόδοση*** του επεξεργαστή.

- Το δεύτερο βασικό σημείο του front-end είναι το στάδιο της ***αποκωδικοποίησης***. Στο στάδιο αυτό από τη δυαδική μορφή της κάθε εντολής αναγνωρίζονται οι καταχωρητές εισόδου / εξόδου καθώς και ο τύπος της εντολής(εντολές ακέραιας πρόσθεσης, φόρτωσης στη μνήμη). Ενώ εάν μιλάμε για επεξεργαστές τύπου ***Complex Instruction Set Computer*** οι οποίοι επιφανειακά διατηρούνε την Instruction Set Architecture λόγω συμβατότητας, σε αυτό το στάδιο μετατρέπουν τις εντολές τύπου CISC σε εντολές τύπου RISC.(οικογένεια x86 INTEL / AMD).

- Τέλος, στο στάδιο του decode γίνεται η άρση των ψευδών ***εξαρτήσεων***(war - waw) μέσω μιας τεχνικής που ονομάζεται ***μετονομασία*** καταχωρητών. Μετά η εντολή προχωράει (λέμε ότι γίνεται dispatch της εντολής) στο back-end του επεξεργαστή για να εκτελεστεί.

- Η καρδιά του ***back-end*** ενός superscalar επεξεργαστή είναι το instruction window (αλλιώς στους σημερινούς επεξεργαστές reorder buffer ή ROB). Ο reorder buffer περιέχει όλες τις εντολές προς εκτέλεση του επεξεργαστή, με λίγα λόγια όσες εντολές έχουν γίνει ***dispatch*** αλλά δεν έχουν ολοκληρωθεί ακόμα (δεν έχουν φτάσει στο στάδιο του commit) βρίσκονται μέσα στο reorder buffer, ο οποίος είναι σχεδιασμένος σαν μια ***FIFO*** ουρά για να διατηρεί τη σειρά προγράμματος, μαζί με επιπλέον πληροφορίες για το στάδιο που βρίσκονται, τις εισόδους και τις εξόδους τους.

- Οι εντολές μέσα στο reorder buffer μπορούν να χωριστούν σε δυο βασικές κατηγορίες:

  – Εντολές που έχουν εκτελεστεί αλλά δεν έχουν ακόμα ενημερώσει το σύστημα (registers, μνήμη) με τις αλλαγές που προκάλεσαν
  – Εντολές που περιμένουν να ικανοποιηθούν οι εξαρτήσεις τους για να αρχίσουν την εκτέλεση τους (issue stage)

PROCESSOR BACK END- ISSUE AND WRITEBACK

- Σε κάθε κύκλο επιλέγονται (ανάλογα με το scheduling) εντολές που είναι ***έτοιμες*** να αρχίσουν την εκτέλεση τους και ανατίθενται στις μονάδες εκτέλεσης. Με αυτό τον τρόπο, είναι δυνατόν να εκτελούνται και εντολές ***εκτός σειράς*** προγράμματος, αρκεί να μην έχουν ***εξαρτήσεις*** από εντολές που δεν έχουν εκτελεστεί ακόμα. Ενώ η τελική αναβάθμιση της κατάστασης του συστήματος όπως προαναφέρθηκε γίνεται συνήθως εντός σειράς προγράμματος.

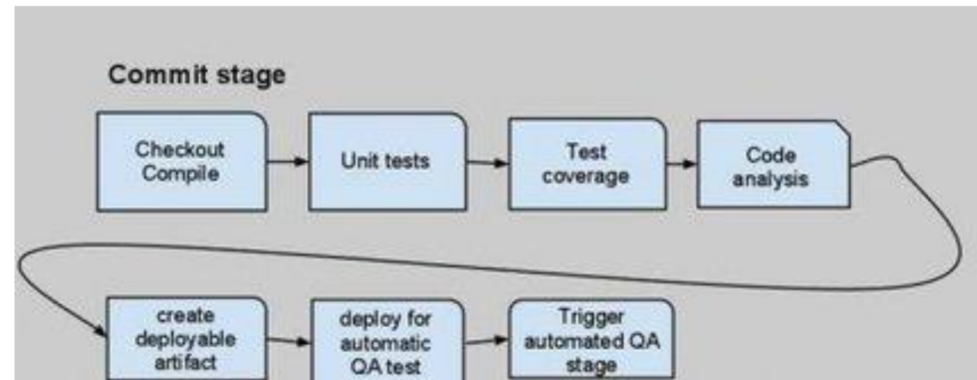
- Στο στάδιο ***issue*** (εκτέλεσης) οι εντολές ανατίθενται σε λειτουργικές μονάδες οι οποίες αναλαμβάνουν την πραγματική εκτέλεση του περιεχομένου της εντολής, τις προσβάσεις στην data cache και στη μνήμη. Όταν η εκτέλεση τελειώσει σε έναν ή περισσότερους κύκλους, το επόμενο στάδιο του pipeline είναι το ***writeback***. Οι εντολές που φτάνουν στο writeback αναβαθμίζουν την κατάσταση του reorder buffer ώστε να επιτρέπουν στις εντολές που αναμένουν την ικανοποίηση των ***dependencies*** τους να μπορούν να ξεκινήσουν με τη σειρά τους να εκτελούνται.

Pipeline Stage	Clock Cycle					
	1	2	3	4	5	6
Fetch	SUB	AND				
Decode		SUB	AND			
Execute			SUB	AND		
Access				SUB	AND	
Write-Back					SUB	AND

PROCESSOR BACK END- COMMIT STAGE

- Το τελικό στάδιο είναι το ***commit stage***, κατά στο οποίο με τη σειρά που βρίσκονται οι εντολές στο reorder buffer ενημερώνουν την ***κατάσταση του συστήματος*** (architectural state) και βγαίνουν από το instruction window. Ο περιορισμός του commit σε σειρά προγράμματος σημαίνει βέβαια ότι σε περίπτωση που η πρώτη εντολή του ROB δεν είναι έτοιμη, ***σταματάει*** όλο το commit μέχρι ότου να τελειώσει η εκτέλεσή της.

- Από την λειτουργία ενός τέτοιου επεξεργαστή είναι προφανές ότι κεντρικό ρόλο παίζει το ***instruction window***, ειδικά στο να αποκαλύψει μεγαλύτερο μέρος του παραλληλισμού του προγράμματος σε επίπεδο εντολών (ILP) αλλά όπως και παραλληλισμό σε επίπεδο μνήμης (MLP: Memory level parallelism).



ΥΨΗΛΗΣ ΑΠΟΔΟΣΗΣ SUPERSCALAR ΕΠΕΞΕΡΓΑΣΤΕΣ

Για να επιτευχθεί ***υψηλότερη απόδοση*** σημαίνει θα πρέπει ένα δεδομένο πρόγραμμα να εκτελεστεί σε λιγότερο χρόνο. Αυτό μπορεί να γίνει είτε μειώνοντας την ***καθυστέρηση*** κάθε εντολής ξεχωριστά είτε με την εκτέλεση περισσότερων εντολών παράλληλα. Οι superscalar επεξεργαστές επικεντρώνονται στον δεύτερο τρόπο. Για να επιτευχθεί παράλληλη εκτέλεση εντολών απαιτείται καθορισμός των σχέσεων ***εξάρτησης*** μεταξύ των εντολών, επαρκείς ***πόροι*** υλικού, καθορισμός της ***ακολουθίας*** των εντολών, καθώς και τεχνικές για τη ***διάδοση*** των αποτελεσμάτων από τη μία εντολή στην επόμενη.

Λειτουργίες ενός superscalar επεξεργαστή

- Στρατηγικές ανάκλησης εντολών οι οποίες φορτώνουν ***ταυτόχρονα*** πολλές εντολές από την μνήμη, προβλέποντας το αποτέλεσμα υπό συνθήκη εντολών διακλάδωσης και πραγματοποιώντας ανάκληση πέρα από αυτές.

Λειτουργίες ενός superscalar επεξεργαστή …

- Μεθόδους για την ***έναρξη*** της εκτέλεσης πολλαπλών εντολών παράλληλα.
- Μεθόδους για τον ***καθορισμό*** των πραγματικών σχέσεων εξάρτησης μεταξύ των τιμών των καταχωρητών και μηχανισμούς για την ***κοινοποίηση*** αυτών των τιμών όπου είναι απαραίτητο κατά τη διάρκεια της εκτέλεσης.
- Πόρους για παράλληλη εκτέλεση πολλών εντολών, συμπεριλαμβανομένων πολλών λειτουργικών μονάδων και ιεραρχιών μνήμης ικανών να εξυπηρετούν ***ταυτόχρονα πολλαπλές*** αναφορές στη μνήμη.
- Μεθόδους για την κοινοποίηση των ***τιμών*** δεδομένων μέσω εντολών φόρτωσης και αποθήκευσης, καθώς και διεπαφές μνήμης που επιτρέπουν τη δυναμική και συχνά απρόβλεπτη συμπεριφορά των ιεραρχιών μνήμης. Οι διεπαφές αυτές πρέπει να είναι σωστά ***ταιριασμένες*** με τις στρατηγικές εκτέλεσης εντολών.
- Μεθόδους για την ***ανανέωση*** της αρχιτεκτονικής κατάστασης του επεξεργαστή με τη σωστή σειρά. Οι μηχανισμοί διατηρούν την εξωτερική εικόνα της ακολουθιακής εκτέλεσης των εντολών.

ΕΞΑΡΤΗΣΕΙΣ ΕΛΕΓΧΟΥ ΣΤΗΝ ΠΑΡΑΛΛΗΛΗ ΕΚΤΕΛΕΣΗ(1)

- Κατά την εκτέλεση ενός προγράμματος αν οι εντολές που πρόκειται να εκτελεστούν είναι συνεχόμενες, η προσθήκη τους στην ακολουθία γίνεται αυξάνοντας τον ***απαριθμητή προγράμματος*** ώστε να δείχνει στην επόμενη εντολή. Η ύπαρξη μίας εντολής άλματος ή διακλάδωσης δημιουργεί **εξαρτήσεις ελέγχου** από τις προηγούμενες εντολές της ακολουθίας, διότι η ροή του προγράμματος καθορίζεται από τις εντολές που προηγούνται. Υπάρχουν δύο μέθοδοι για την απόδοση τιμής στον απαριθμητή προγράμματος, η ***αύξηση*** και η ***ανανέωση***.

- Οι εξαρτήσεις ελέγχου που προκύπτουν από αύξηση του απαριθμητή προγράμματος είναι οι πιο ***απλές*** και για να ξεπεραστούν μπορεί το πρόγραμμα να θεωρηθεί ως ένα σύνολο από ***βασικά τμήματα*** όπου ένα βασικό τμήμα είναι ένα συνεχόμενο τμήμα εντολών, με ένα σημείο εισόδου και ένα σημείο εξόδου. Όταν η ανάκληση εντολών αρχίσει να γίνεται σε ένα βασικό τμήμα, όλες οι εντολές του βασικού τμήματος τελικά θα εκτελεστούν παράλληλα με μόνο περιορισμό τις εξαρτήσεις.

- Για την επίτευξη του παραλληλισμού πρέπει να ξεπεραστεί η ***εξάρτηση ελέγχου***. Μία μέθοδος είναι ***η πρόβλεψη*** του αποτελέσματος μιας εντολής διακλάδωσης υπό συνθήκη και η ανάκληση εντολών από το μονοπάτι εντολών που προβλέφθηκε ότι θα ακολουθηθεί.
  - Αν η πρόβλεψη αργότερα ***επαληθευτεί***, τότε οι εντολές παύουν να θεωρούνται ότι ανήκουν σε ένα πιθανό μονοπάτι και το αποτέλεσμά τους όσον αφορά την κατάσταση του επεξεργαστή είναι το ίδιο με οποιαδήποτε άλλη εντολή.
  - Αν όμως η πρόβλεψη αποδειχθεί ***λανθασμένη***, τότε πρέπει να γίνουν ενέργειες ώστε η διαμόρφωση της κατάστασης του επεξεργαστή να μην επηρεαστεί από τις εντολές του λανθασμένου μονοπατιού.
- Οι **εξαρτήσεις δεδομένων** που αναφέρθηκαν μπορεί να υπάρχουν μεταξύ διαφορετικών εντολών οι οποίες προσπελαύνουν για ανάγνωση ή εγγραφή τις ίδιες ***περιοχές αποθήκευσης*** (σε καταχωρητές ή στη μνήμη). Όταν διαφορετικές εντολές αναφέρονται στην ίδια περιοχή αποθήκευσης θα πρέπει η περιοχή αποθήκευσης να προσπελαστεί με τη ***σωστή σειρά*** ώστε να μην υπάρξουν σφάλματα.

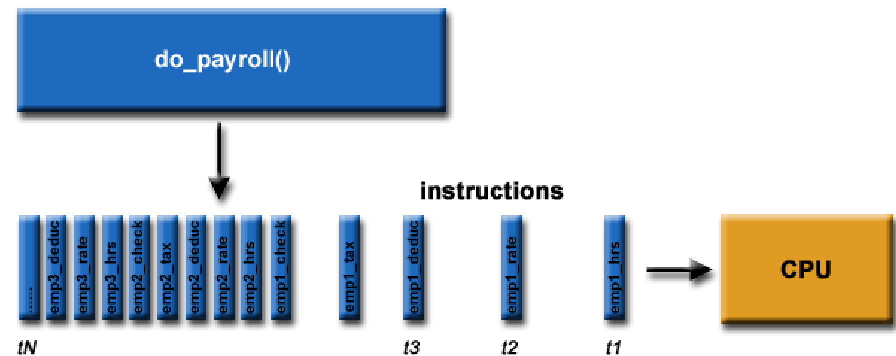
ΕΙΔΗ ΕΞΑΡΤΗΣΕΩΝ

- Υπάρχουν τρία διαφορετικά είδη εξαρτήσεων οι οποίες εμφανίζονται ανάμεσα στις εντολές και συναντώνται με τα ονόματα RAW, WAW, WAR.
  - **RAW** : Η πρώτη εντολή γραφεί σε μια θέση μνήμης την οποία και διαβάζει μια επόμενη εντολή.
  - **WAW** : Δυο εντολές θέλουν να γράψουν διαδοχικά στην ίδια θέση μνήμης.
  - **WAR** : Η πρώτη εντολή διαβάζει από μια θέση μνήμης στην οποία γραφεί η επόμενη εντολή.

- Μετά την επίλυση των εξαρτήσεων, οι εντολές είναι έτοιμες να εκτελεστούν παράλληλα. Το υλικό δημιουργεί ένα πρόγραμμα παράλληλης εκτέλεσης το οποίο λαμβάνει υπόψη τους ***απαραίτητους περιορισμούς***, όπως είναι οι πραγματικές εξαρτήσεις και οι περιορισμοί πόρων υλικού (λειτουργικές μονάδες και μονοπάτια δεδομένων).

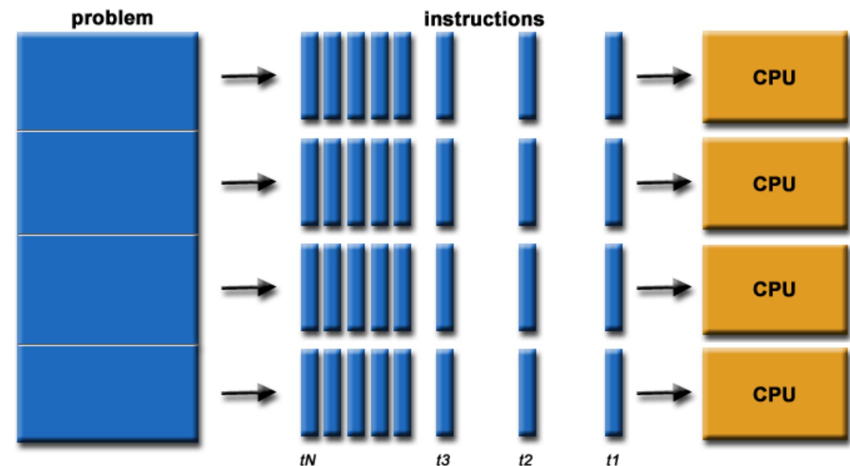
ΔΙΑΔΙΚΑΣΙΑ ΟΛΟΚΛΗΡΩΣΗΣ ΜΙΑΣ ΕΝΤΟΛΗΣ(1)

- Η ύπαρξη του προγράμματος παράλληλης εκτέλεσης σημαίνει ότι οι εντολές ολοκληρώνουν την εκτέλεσή τους με σειρά ***διαφορετική*** από αυτή του ακολουθιακού μοντέλου εκτέλεσης και ότι ορισμένες εντολές θα ολοκληρώσουν την εκτέλεσή τους ενώ δε θα έπρεπε να εκτελεστούν. Συνεπώς, η κατάσταση του επεξεργαστή δεν μπορεί να ανανεωθεί ***αμέσως*** όταν οι εντολές ολοκληρώνουν την εκτέλεσή τους. Τα αποτελέσματα των εντολών πρέπει να φυλάσσονται προσωρινά έως ότου η ***κατάσταση*** μπορεί να ανανεωθεί. Στο μεταξύ, για να διατηρηθεί υψηλή η ***απόδοση***, τα αποτελέσματα πρέπει να μπορούν να χρησιμοποιηθούν από εντολές που τα χρειάζονται.
- Τελικά, όταν καθοριστεί ότι μία εντολή θα έπρεπε να εκτελεστεί σύμφωνα με το ακολουθιακό μοντέλο, τα προσωρινά αποτελέσματά της γίνονται ***μόνιμα*** ανανεώνοντας την αρχιτεκτονική κατάσταση του επεξεργαστή. Αυτή η διαδικασία ονομάζεται ***ολοκλήρωση*** της εντολής.



ΔΙΑΔΙΚΑΣΙΑ ΟΛΟΚΛΗΡΩΣΗΣ ΜΙΑΣ ΕΝΤΟΛΗΣ(2)

- Συνοψίζοντας οι εντολές ***ορίζονται*** από ένα στατικό πρόγραμμα και η ανάκληση και η πρόβλεψη των εντολών διακλάδωσης υπό συνθήκη, χρησιμοποιούνται για τη δημιουργία μιας ροής δυναμικών εντολών. Έπειτα αντιμετωπίζονται οι διάφορες ***εξαρτήσεις*** ενώ δημιουργούνται τα βασικά τμήματα εκτέλεσης. Από το βασικό τμήμα εκτέλεσης επιλέγονται οι εντολές που θα εκτελεστούν με σειρά που καθορίζουν οι πραγματικές εξαρτήσεις δεδομένων και η διαθεσιμότητα των ***πόρων*** του υλικού. Τέλος, μετά την εκτέλεση, οι εντολές κατά μία έννοια ***επανέρχονται*** στην ***αρχική*** ακολουθιακή σειρά που ορίζει το πρόγραμμα καθώς ολοκληρώνονται.



ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΥΠΙΚΟΥ ΕΠΕΞΕΡΓΑΣΤΗ SUPERSCALAR(1)

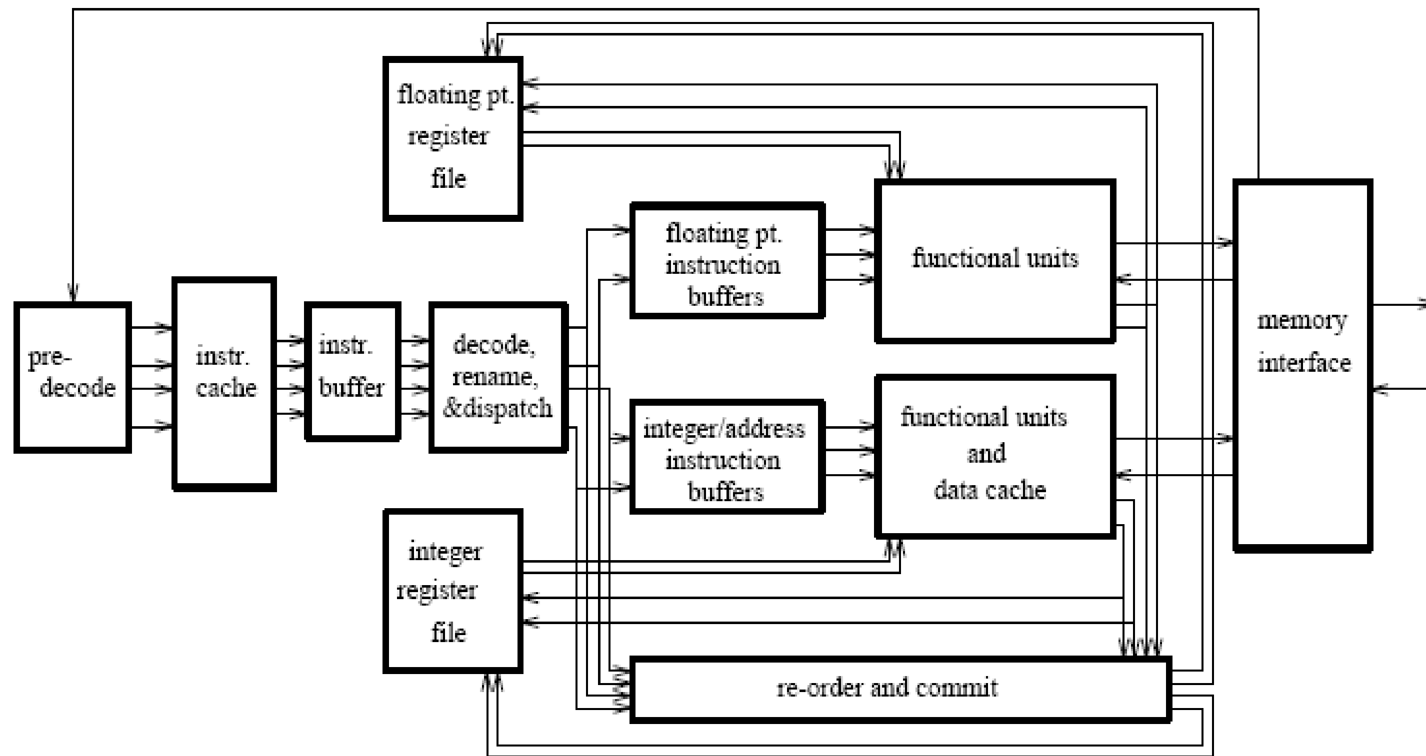
- Τα κύρια σημεία της αρχιτεκτονικής είναι:
  - Ανάκληση εντολών και πρόβλεψη διακλαδώσεων.
  - Αποκωδικοποίηση και ανάλυση εξαρτήσεων καταχωρητών.
  - Επιλογή από το παράθυρο εκτέλεσης και εκτέλεση.
  - Ανάλυση εργασιών μνήμης και εκτέλεση.
  - Επαναφορά των εντολών στην αρχική σειρά.
  - Ολοκλήρωση των εντολών.

- Δηλαδή οι superscalar επεξεργαστές διαθέτουν :

- Μονάδα απόκτησης εντολών η οποία έχει τη δυνατότητα απόκτησης πολλών εντολών ταυτόχρονα
- Μονάδα αποκωδικοποίησης εντολών, η οποία έχει τη δυνατότητα ανίχνευσης ανεξαρτήτων μεταξύ τους εντολών
- Πολλαπλές μονάδες εκτέλεσης οι οποίες έχουν τη δυνατότητα να εκτελούν παράλληλα εντολές όταν αυτό είναι δυνατόν
- Στις μονάδες εκτέλεσης εφαρμόζεται η τεχνική pipeline για περισσότερη ταχύτητα.

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΥΠΙΚΟΥ ΕΠΕΞΕΡΓΑΣΤΗ SUPERSCALAR(2)

Παρακάτω παρουσιάζεται η αρχιτεκτονική, ενός τυπικού superscalar επεξεργαστή.



ΑΝΑΚΛΗΣΗ ΕΝΤΟΛΩΝ ΚΑΙ ΠΡΟΒΛΕΨΗ ΔΙΑΚΛΑΔΩΣΕΩΝ

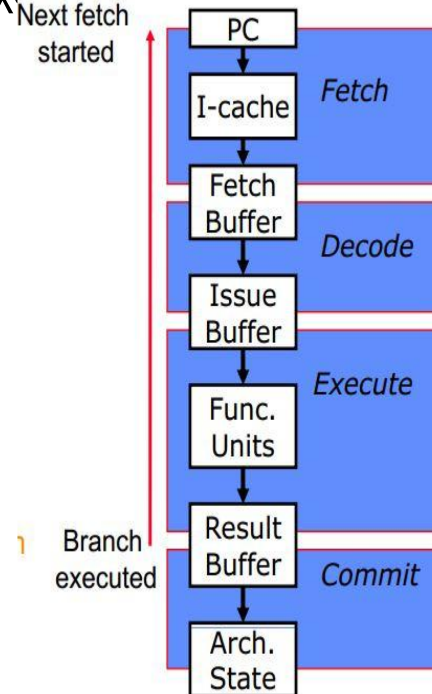
- Η φάση ***ανάκλησης*** εντολών είναι το τμήμα της superscalar επεξεργασίας που προμηθεύει με εντολές τις επόμενες ***φάσεις*** της επεξεργασίας. Μία κρυφή μνήμη εντολών, η οποία είναι μία μνήμη που περιέχει τις ***πρόσφατα*** χρησιμοποιηθείσες εντολές, για να μειωθεί η καθυστέρηση και να αυξηθεί το εύρος ζώνης της διαδικασίας ανάκλησης εντολών. Είναι οργανωμένη σε ***τμήματα ή γραμμές*** που περιέχουν αρκετές συνεχόμενες εντολές

- Η φάση ανάκλησης εντολών πρέπει να υποστηρίζει την ανάκληση πολλαπλών εντολών ***ανά κύκλο*** από την κρυφή μνήμη. Συνήθως έχουμε ξεχωριστές κρυφές μνήμες για εντολές και δεδομένα.

- Σε κάθε ανάκληση εντολών αυξάνεται ο απαριθμητής προγράμματος κατά τον αριθμό των εντολών που ανακλήθηκαν και συνεχίζει με τον ίδιο τρόπο για την ανάκληση του επόμενου τμήματος εντολών. Αν υπάρχουν εντολές διακλάδωσης οι οποίες ***ανακατευθύνουν*** τη ροή ελέγχου ο μηχανισμός ανάκλησης πρέπει επίσης να ανακατευθυνθεί ώστε η ανάκληση να συνεχιστεί από το σημείο στο οποίο οδηγεί η διακλάδωση. Η διαχείριση των εντολών διακλάδωσης είναι πολύ σημαντική για την επίτευξη υψηλής απόδοσης στους superscalar επεξεργαστές.

ΠΡΟΒΛΕΨΗ ΔΙΑΚΛΑΔΩΣΕΩΝ(1)

- Η επεξεργασία των εντολών διακλάδωσης υπό συνθήκη μπορεί να χωριστεί στα ακόλουθα μέρη:
  - Αναγνώριση ότι μία εντολή είναι εντολή διακλάδωσης υπό συνθήκη.
  - Υπολογισμός του σημείου στο οποίο οδηγεί η εντολή διακλάδωσης.
  - Μεταφορά του ελέγχου με ανακατεύθυνση της ανάκλησης εντολών (στην περίπτωση που η διακλάδωση πραγματοποιείται).
  - Καθορισμός του αποτελέσματος της διακλάδωσης (αν θα πραγματοποιηθεί ή όχι).



- Για τη διαχείριση καθενός από τα παραπάνω μέρη χρησιμοποιούνται συγκεκριμένες τεχνικές.

Παρακάτω θα προχωρήσουμε στην λεπτομερή ανάλυση αυτών των μερών.

ΠΡΟΒΛΕΨΗ ΔΙΑΚΛΑΔΩΣΕΩΝ(2)

- **Αναγνώριση των εντολών διακλάδωσης υπό συνθήκη**

  Πρόκειται για το ***πρώτο βήμα*** για την επίτευξη γρήγορης πρόβλεψης διακλαδώσεων. Η αναγνώριση όλων των ειδών εντολών και όχι μόνο των εντολών διακλάδωσης μπορεί να επιταχυνθεί αν ορισμένες από τις πληροφορίες αποκωδικοποίησης της εντολής φυλάσσονται στην κρυφή μνήμη εντολών μαζί με τις εντολές.

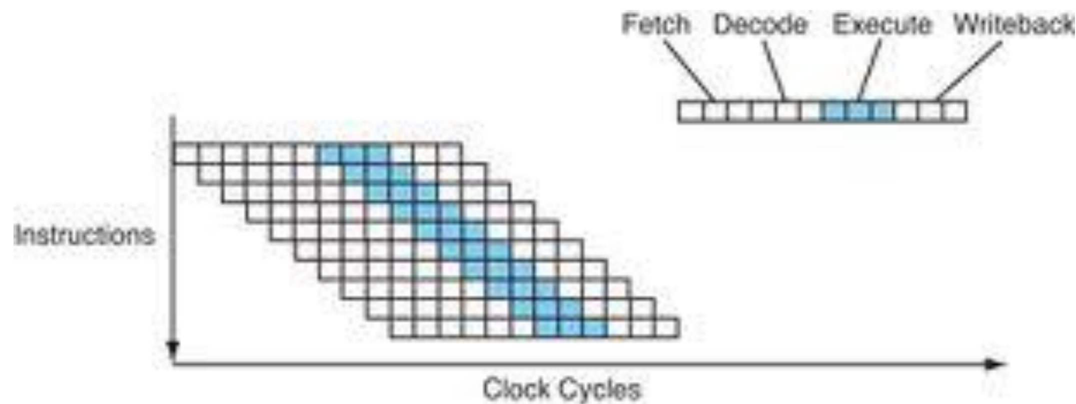
- **Υπολογισμός του σημείου στο οποίο οδηγεί η εντολή διακλάδωσης**

  Ο υπολογισμός του σημείου στο οποίο οδηγεί η εντολή διακλάδωσης συνήθως απαιτεί μία ***πρόσθεση ακεραίων***. Στις περισσότερες αρχιτεκτονικές το σημείο αυτό δίνεται σχετικά ως προς την απαριθμητή προγράμματος χρησιμοποιώντας μία τιμή διαφοράς (offset) η οποία είναι αποθηκευμένη στην εντολή. Επιπλέον, μπορεί να υπάρχουν κάποιες εντολές διακλάδωσης που χρειάζονται ***περιεχόμενα καταχωρητών*** για να υπολογιστεί το σημείο στο οποίο οδηγεί η διακλάδωση.

ΠΡΟΒΛΕΨΗ ΔΙΑΚΛΑΔΩΣΕΩΝ(3)

- **Μεταφορά του ελέγχου**

Όταν μία διακλάδωση πραγματοποιείται συνήθως υπάρχει μία ***καθυστέρηση*** λόγω της αναγνώρισης της , της τροποποίησης του απαριθμητή προγράμματος και της αρχής της ανάκληση εντολών από την νέα διεύθυνση. Αυτή η καθυστέρηση μπορεί να οδηγήσει σε ***κενά*** στο pipeline αν δεν ληφθούν μέτρα. Η πιο κοινή λύση είναι η χρήση του αποθηκευτικού χώρου εντολών .



ΠΡΟΒΛΕΨΗ ΔΙΑΚΛΑΔΩΣΕΩΝ(4)

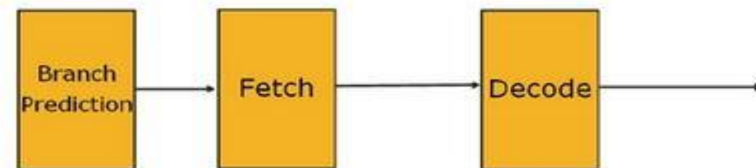
- **Καθορισμός του αποτελέσματος της διακλάδωσης**

Τα δεδομένα με βάση τα οποία θα παρθεί η ***απόφαση*** για τη διακλάδωση δεν είναι διαθέσιμα λόγω εξάρτησης ανάμεσα στην εντολή διακλάδωσης και σε εντολές του προγράμματος που δεν έχουν εκτελεστεί. Τότε το αποτέλεσμα μιας διακλάδωσης υπό συνθήκη μπορεί να ***προβλεφθεί*** με τον μεταγλωττιστή, ο οποίος μπορεί να υποδείξει την πιθανότερη κατάληξη της διακλάδωσης με βάση τη ***γνώση*** που διαθέτει για τη γλώσσα που χρησιμοποιείται αλλά και με τη χρήση στατιστικών πληροφοριών.

Άλλοι μηχανισμοί πρόβλεψης λειτουργούν ***δυναμικά***, χρησιμοποιώντας πληροφορία που γίνεται διαθέσιμη καθώς το πρόγραμμα εκτελείται. Την πληροφορία την παίρνουμε από το ***ιστορικό*** των διαφόρων εντολών διακλάδωσης το οποίο αποθηκεύεται σε έναν πίνακα ιστορικού. Στον ***πίνακα*** αυτό καταγράφονται τα αποτελέσματα προηγούμενων εκτελέσεων των εντολών διακλάδωσης σχηματίζοντας μικρούς ***μετρητές*** έτσι ώστε να συνοψίζουν τα αποτελέσματα αρκετών προηγούμενων εκτελέσεων κάθε εντολής διακλάδωσης.

ΑΠΟΚΩΔΙΚΟΠΟΙΗΣΗ ΕΝΤΟΛΩΝ

- Ο ρόλος της αποκωδικοποίησης είναι να δημιουργήσει για κάθε εντολή μία λίστα η οποία καθορίζει:
  - την εργασία που θα εκτελεστεί
  - την ταυτότητα των στοιχείων αποθήκευσης όπου βρίσκονται οι τελεσταίοι
  - οι τοποθεσίες στις οποίες θα τοποθετηθεί το αποτέλεσμα της εντολής.



- Στο ***στατικό*** πρόγραμμα, τα στοιχεία αποθήκευσης είναι τα αρχιτεκτονικά στοιχεία, προκειμένου όμως να ξεπεραστούν οι κίνδυνοι WAR και WAW και να αυξηθεί ο βαθμός παραλληλίας συχνά υπάρχουν ***φυσικά*** στοιχεία αποθήκευσης τα οποία μπορεί να διαφέρουν από τα ***λογικά*** στοιχεία αποθήκευσης. Τα λογικά στοιχεία αποθήκευσης μπορούν να θεωρηθούν σαν ένα απλό μέσο το οποίο βοηθά στην περιγραφή του τι πρέπει να κάνει το πρόγραμμα.

ΜΕΤΟΝΟΜΑΣΙΑ ΕΝΤΟΛΩΝ(1)

- Όταν μία εντολή δημιουργεί μία νέα τιμή για έναν λογικό καταχωρητή, η φυσική τοποθεσία που τοποθετείται η τιμή λαμβάνει ένα «***όνομα***» γνωστό από το υλικό. Σε οποιαδήποτε επόμενη εντολή η οποία χρησιμοποιεί την τιμή αυτή σαν ***είσοδο*** γνωστοποιείται το όνομα της φυσικής περιοχής αποθήκευσης.

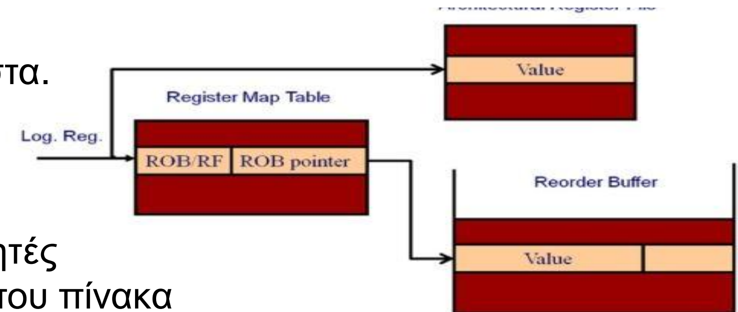
- Αυτό πραγματοποιείται στην φάση αποκωδικοποίησης/μετονομασίας/προώθησης αντικαθιστώντας τον λογικό καταχωρητή που ορίζει η εντολή με το νέο όνομα της φυσικής περιοχής αποθήκευσης. Ο καταχωρητής τότε λέμε ότι ***μετονομάζεται***.

ΜΕΤΟΝΟΜΑΣΙΑ ΕΝΤΟΛΩΝ(2)

- Δύο είναι οι δημοφιλέστερες ***μέθοδοι*** μετονομασίας που χρησιμοποιούνται συχνά. Στην πρώτη μέθοδο, υπάρχει ένα φυσικό αρχείο καταχωρητών μεγαλύτερο από το λογικό αρχείο καταχωρητών. Ένας ***πίνακας αντιστοίχησης*** χρησιμοποιείται για να αντιστοιχιστεί ένας φυσικός καταχωρητής με την τρέχουσα τιμή ενός λογικού καταχωρητή.

- Η αποκωδικοποίηση των εντολών και η μετονομασία των καταχωρητών πραγματοποιείται βάση της ***ακολουθιακής σειράς*** του προγράμματος. Κατά την αποκωδικοποίηση μιας εντολής, ο λογικός καταχωρητής όπου θα τοποθετηθεί το αποτέλεσμα αντιστοιχίζεται σε κάποιον φυσικό καταχωρητή μέσω μίας ***λίστας*** ελεύθερων καταχωρητών (free list), η οποία αποτελείται από τους φυσικούς καταχωρητές που δεν έχουν αντιστοιχιστεί σε κάποιον λογικό καταχωρητή και ο πίνακας αντιστοίχησης τροποποιείται ώστε να δείχνει την νέα αντιστοίχηση.

ΜΕΤΟΝΟΜΑΣΙΑ ΚΑΙ ΠΡΟΩΘΗΣΗ ΕΝΤΟΛΩΝ



- Επίσης, ο ***φυσικός*** καταχωρητής αφαιρείται από τη λίστα. Σαν μέρος της διαδικασίας μετονομασίας, οι λογικοί καταχωρητές στους οποίους ορίζεται από την εντολή ότι βρίσκονται τα δεδομένα εισόδου της εντολής χρησιμοποιούνται για να βρεθούν οι φυσικοί καταχωρητές στους οποίους βρίσκονται τα δεδομένα (με τη βοήθεια του πίνακα αντιστοίχησης). Από αυτούς τους καταχωρητές θα ***φορτωθούν*** οι τιμές των τελεσταίων της εντολής.

- Όταν ένας φυσικός καταχωρητής διαβαστεί για ***τελευταία*** φορά, μπορεί να τοποθετηθεί ξανά στη λίστα ***ελεύθερων*** καταχωρητών ώστε να χρησιμοποιηθεί από άλλες εντολές. Ανάλογα με την υλοποίηση, αυτό μπορεί να απαιτεί από το υλικό τη διατήρηση λιγότερων ή περισσότερων στοιχείων χρήσης του καταχωρητή.

- Μία απλή μέθοδος είναι η ***αναμονή*** έως ότου ο αντίστοιχος λογικός καταχωρητής όχι μόνο έχει μετονομαστεί από μία επόμενη εντολή, αλλά η εντολή αυτή έχει λάβει την τιμή της και έχει ολοκληρωθεί .

- Συνοψίζοντας , η μετονομασία καταχωρητών ***αφαιρεί*** τις τεχνητές εξαρτήσεις που οφείλονται σε WAW και WAR κινδύνους, αφήνοντας μόνο τις πραγματικές RAW εξαρτήσεις μεταξύ καταχωρητών.

- Το επόμενο βήμα είναι να αποφασιστεί ποιες εντολές μπορούν να αρχίσουν να εκτελούνται, εξετάζοντας τη ***λίστα*** κάθε εντολής. Η διαδικασία αυτή ονομάζεται ***issue*** *και ορίζεται ως ο έλεγχος* διαθεσιμότητας δεδομένων και πόρων κατά τη διάρκεια της εκτέλεσης του προγράμματος ώστε να καθοριστεί ποιες εντολές μπορούν να αρχίσουν να εκτελούνται.

- Μία εντολή είναι ***έτοιμη*** να εκτελεστεί τη στιγμή που οι τελεσταίοι εισόδου της γίνουν διαθέσιμοι. Παρ’ όλα αυτά, υπάρχουν περιορισμοί που εμποδίζουν την εκτέλεση μίας εντολής, όπως είναι η ***διαθεσιμότητα*** πόρων όπως είναι οι μονάδες εκτέλεσης, οι διαθέσιμες διασυνδέσεις και οι θύρες του αρχείου καταχωρητών***.*** Άλλοι περιορισμοί σχετίζονται με την οργάνωση των αποθηκευτικών χώρων όπου φυλάσσονται οι λίστες που δημιουργούνται για κάθε εντολή.

- Στη συνέχεια αναφέρονται συνοπτικά διάφοροι τρόποι για την οργάνωση των αποθηκευτικών χώρων που χρειάζονται για τη διαδικασία issue.

ΤΡΟΠΟΙ ΓΙΑ ΤΗΝ ΟΡΓΑΝΩΣΗ ΤΩΝ ΑΠΟΘΗΚΕΥΤΙΚΩΝ ΧΩΡΩΝ(1)

- **Μέθοδος απλής ουράς**

Αν υπάρχει μία απλή ουρά, και δεν πραγματοποιείται εκτέλεση εκτός σειράς, τότε η μετονομασία καταχωρητών δεν είναι απαραίτητη και η διαθεσιμότητα των τελεσταίων μπορεί να ελεγχθεί με ορισμένα ***bits κράτησης*** σε κάθε καταχωρητή.

Ένας καταχωρητής είναι ***κατειλημμένος*** όταν μία εντολή που τροποποιεί τον καταχωρητή αρχίζει να εκτελείται και παύει να είναι κατειλημμένος όταν η εντολή εκτελεστεί. Μία εντολή μπορεί να αρχίσει να εκτελείται αν οι καταχωρητές όπου βρίσκονται οι τελεσταίοι της δεν είναι κατειλημμένοι.

ΤΡΟΠΟΙ ΓΙΑ ΤΗΝ ΟΡΓΑΝΩΣΗ ΤΩΝ ΑΠΟΘΗΚΕΥΤΙΚΩΝ ΧΩΡΩΝ(2)

- **Μέθοδος πολλαπλών ουρών**

Όταν υπάρχουν πολλαπλές ουρές, οι εντολές από κάθε ουρά αρχίζουν να εκτελούνται με τη ***σειρά***, αλλά η εκτέλεση μπορεί να είναι εκτός σειράς όταν αναφερόμαστε σε διαφορετικές ουρές. Η κάθε ουρά ***οργανώνεται*** ανάλογα με τον τύπο των εντολών.

Σε αυτή τη μέθοδο, η μετονομασία χρησιμοποιείται σε ***περιορισμένη*** έκταση. Για παράδειγμα, μπορούν να μετονομαστούν μόνο οι καταχωρητές που φορτώνονται από τη μνήμη. Αυτό ***επιτρέπει*** την ουρά εντολών φόρτωσης/αποθήκευσης να προηγούνται των υπολοίπων ουρών εντολών, πραγματοποιώντας ανάκληση δεδομένων πριν αυτά αναζητηθούν από κάποιες εντολές.

ΤΡΟΠΟΙ ΓΙΑ ΤΗΝ ΟΡΓΑΝΩΣΗ ΤΩΝ ΑΠΟΘΗΚΕΥΤΙΚΩΝ ΧΩΡΩΝ(3)

- **Σταθμοί κράτησης**

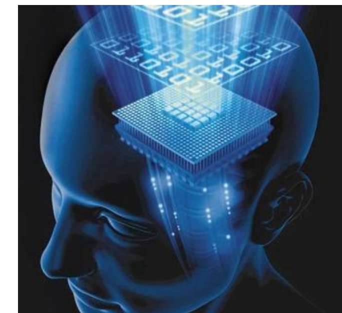
Με τους σταθμούς κράτησης οι εντολές μπορούν να εκτελούνται εκτός σειράς. Δεν υπάρχει ***αυστηρή*** FIFO διάταξη. Συνεπώς, όλοι οι σταθμοί κράτησης ελέγχουν ταυτόχρονα τους τελεσταίους πηγής για διαθεσιμότητα των δεδομένων.

Ο πιο συνηθισμένος τρόπος για να γίνει αυτό είναι η ***κράτηση*** των τελεσταίων στο σταθμό κράτησης. Όταν η εντολή προωθείται στο σταθμό κράτησης, οποιεσδήποτε ήδη διαθέσιμες τιμές τελεσταίων διαβάζονται από το αρχείο καταχωρητών και τοποθετούνται στο σταθμό κράτησης. Η ***λογική*** του σταθμού κράτησης συγκρίνει το ***όνομα*** των καταχωρητών στους οποίους δεν έχουν τοποθετηθεί ακόμα τα δεδομένα με το όνομα των καταχωρητών στους οποίους τοποθετούνται τα δεδομένα των εντολών που τελειώνουν την εκτέλεσή τους. Όταν υπάρχει ***ταίριασμα*** το αποτέλεσμα τοποθετείται στην αντίστοιχη θέση στο σταθμό κράτησης. Όταν όλοι οι τελεσταίοι είναι διαθέσιμοι στο σταθμό κράτησης, η εντολή μπορεί να αρχίσει να εκτελείται. Οι σταθμοί κράτησης μπορούν να χωριστούν ή να αποτελούν ένα ενιαίο σύνολο.

ΛΕΙΤΟΥΡΓΙΕΣ ΜΝΗΜΗΣ ΚΑΙ ΑΝΑΝΕΩΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΚΑΤΑΣΤΑΣΗΣ(1)

- Οι λειτουργίες μνήμης απαιτούν ειδική μεταχείριση στους superscalar επεξεργαστές. Για τη μείωση της καθυστέρησης των λειτουργιών μνήμης χρησιμοποιούνται ***ιεραρχίες*** μνήμης. Σχεδόν ***όλοι*** οι επεξεργαστές περιέχουν μία κρυφή μνήμη δεδομένων. Υπάρχουν πολλαπλά επίπεδα κρυφών μνημών δεδομένων συνήθως δυο.

- Οι περιοχές μνήμης που θα προσπελαστούν από τις εντολές φόρτωσης και αποθήκευσης ***αναγνωρίζονται*** μετά την έναρξη της εκτέλεσης (issue) της εντολής. Για να καθοριστεί η περιοχή μνήμης που θα προσπελαστεί είναι απαραίτητος ο υπολογισμός διεύθυνσης. Μετά τον υπολογισμό, είναι πιθανό να χρειαστεί μία ***μετάφραση*** διεύθυνσης προκειμένου να παραχθεί τελικά η φυσική διεύθυνση. Μία κρυφή μνήμη που διατηρεί τους περιγραφείς των σελίδων που προσπελάστηκαν πρόσφατα χρησιμοποιείται για την επιτάχυνση της διαδικασίας μετάφρασης. Όταν η ***έγκυρη*** διεύθυνση μνήμης έχει πλέον παραχθεί, μπορεί πλέον να αρχίσει η προσπέλαση της μνήμης.



ΛΕΙΤΟΥΡΓΙΕΣ ΜΝΗΜΗΣ ΚΑΙ ΑΝΑΝΕΩΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΚΑΤΑΣΤΑΣΗΣ(2)

- Είναι επιθυμητό οι λειτουργίες μνήμης να εκτελούνται σε όσο το ***δυνατόν λιγότερο*** χρόνο. Αυτό επιτυγχάνεται μειώνοντας την καθυστέρηση τους μέσω εκτέλεσης πολλαπλών λειτουργιών ***ταυτόχρονα*** επικαλύπτοντας την εκτέλεση των λειτουργιών μνήμης με άλλες λειτουργίες, και πιθανώς επιτρέποντας στις λειτουργίες μνήμης να εκτελούνται εκτός σειράς.

- Ορισμένοι superscalar επεξεργαστές επιτρέπουν ***μόνο μία*** λειτουργία μνήμης σε κάθε κύκλο. Έτσι όμως περιορίζεται σημαντικά η απόδοση του επεξεργαστή. Επιπλέον, οι ***μεταφορές*** από υψηλότερα επίπεδα στην κρυφή μνήμη πρώτου επιπέδου πραγματοποιείται συνήθως με τη μορφή γραμμών, οι οποίες περιέχουν πολλαπλές συνεχόμενες λέξεις δεδομένων. Για να μπορούν οι λειτουργίες μνήμης να ***επικαλύπτονται*** με άλλες λειτουργίες η ιεραρχία μνήμης πρέπει να είναι μη παρεμποδιστική. Αυτό σημαίνει ότι αν μία αίτηση μνήμης ***δε βρει*** τα δεδομένα στη κρυφή μνήμη δεδομένων, άλλες λειτουργίες που βρίσκονται σε εξέλιξη, συμπεριλαμβανομένων επιπλέον αιτήσεων μνήμης, θα πρέπει να επιτρέπεται να ***συνεχίσουν*** την εκτέλεσή τους.

ΛΕΙΤΟΥΡΓΙΕΣ ΜΝΗΜΗΣ ΚΑΙ ΑΝΑΝΕΩΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΚΑΤΑΣΤΑΣΗΣ(3)

- Από τη στιγμή που η λειτουργία έχει ***προωθηθεί*** προς την ιεραρχία μνήμης, μπορεί να υπάρχει είτε ευστοχία είτε ***αστοχία*** στην κρυφή μνήμη δεδομένων. Στην περίπτωση αστοχίας, η γραμμή που περιέχει την περιοχή που προσπελαύνεται πρέπει να τοποθετηθεί στην κρυφή μνήμη. Περαιτέρω προσπελάσεις στην ίδια γραμμή πρέπει να ***περιμένουν***, αλλά προσπελάσεις σε άλλες γραμμές μπορούν να πραγματοποιηθούν.

- Το ***τελευταίο στάδιο*** κάθε εντολής είναι η φάση διάπραξης ή απόσυρσης (commit ή retire), όπου τα αποτελέσματα της εντολής μπορούν πλέον να τροποποιήσουν την αρχιτεκτονική κατάσταση του επεξεργαστή ώστε να διατηρηθεί η προς τα έξω εμφάνιση της ***ακολουθιακής εκτέλεσης*** .

- Υπάρχουν ***δύο μέθοδοι*** που χρησιμοποιούνται για την ανάκτηση μίας ακριβής κατάστασης. Και στις δύο απαιτείται η διατήρηση δύο ειδών κατάστασης:
  - της κατάστασης που ανανεώνεται καθώς οι εντολές εκτελούνται
  - της κατάστασης που απαιτείται για την ανάκτηση.

ΜΕΘΟΔΟΙ ΓΙΑ ΑΝΑΚΤΗΣΗ ΜΙΑΣ ΑΚΡΙΒΗΣ ΚΑΤΑΣΤΑΣΗΣ

- Στην ***πρώτη μέθοδο***, η κατάσταση του επεξεργαστή σώζεται σε διάφορα σημεία και αποθηκεύεται σε έναν αποθηκευτικό χώρο ιστορικού. Καθώς οι εντολές εκτελούνται ανανεώνουν την κατάσταση του επεξεργαστή και όταν απαιτείται μία ακριβής κατάσταση του επεξεργαστή, ανακτάται από τον αποθηκευτικό χώρο ιστορικού. Σε αυτή την περίπτωση, η μόνη ενέργεια που πρέπει να πραγματοποιηθεί στη φάση διάπραξης (commit) είναι η ***διαγραφή*** του ιστορικού που δεν είναι πλέον απαραίτητο.

- Η ***δεύτερη μέθοδος*** είναι ο διαχωρισμός της κατάστασης του επεξεργαστή σε δύο διαφορετικές καταστάσεις: τη φυσική κατάσταση και τη λογική κατάσταση. Η φυσική κατάσταση ***ανανεώνεται αμέσως*** όταν οι εντολές ολοκληρώνονται. Η αρχιτεκτονική κατάσταση του επεξεργαστή ανανεώνεται βάση της ακολουθιακής σειράς του προγράμματος.

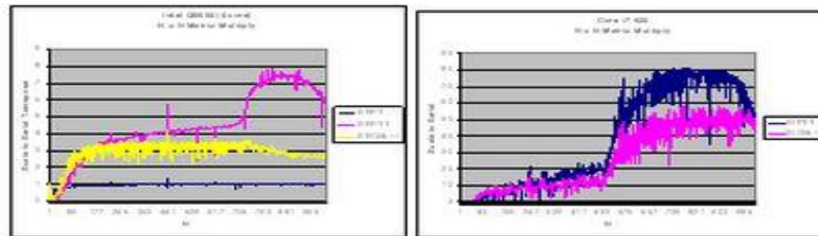
Ο ΡΟΛΟΣ ΤΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ ΣΤΗΝ ΑΥΞΗΣΗ ΤΗΣ ΑΠΟΔΟΣΗΣ

- Αν και οι superscalar επεξεργαστές στοχεύουν ***στην αύξηση της απόδοσης*** ήδη υπαρχόντων προγραμμάτων, η απόδοση μπορεί να αυξηθεί αν δημιουργηθούν νέα, ***βελτιστοποιημένα*** προγράμματα. Ο προγραμματιστής μπορεί να βοηθήσει τον επεξεργαστή, γράφοντας το πρόγραμμα με τέτοιο τρόπο ώστε η ανάκληση και η εκτέλεση των εντολών να πραγματοποιείται πιο αποδοτικά. Αυτό μπορεί να γίνει:
  - αυξάνοντας την πιθανότητα μία ομάδα εντολών να μπορεί να εκτελεστεί παράλληλα
  - μειώνοντας την πιθανότητα μία εντολή να πρέπει να περιμένει το αποτέλεσμα κάποιας προηγούμενης εντολής.
- Κατά τη συγγραφή του κώδικα, γράφοντας τις εντολές με τέτοιο ***τρόπο*** ώστε μία ομάδα εντολών να ταιριάζει με τους περιορισμούς παράλληλης εκτέλεσης του επεξεργαστή επιτυγχάνεται ο ***πρώτος στόχος***. Οι περιορισμοί αυτοί περιλαμβάνουν τις σχέσεις εξάρτησης μεταξύ των εντολών και τη διαθεσιμότητα πόρων του επεξεργαστή. Ο ***δεύτερος στόχος*** επιτυγχάνεται τοποθετώντας στον κώδικα του προγράμματος μία εντολή, της οποίας το αποτέλεσμα τροφοδοτεί μία άλλη εντολή, πολύ πριν την εντολή-καταναλωτή.

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΥΠΕΡΒΑΘΜΩΤΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

- Θα χρησιμοποιηθεί ο γνωστός αλγόριθμος ***πολλαπλασιασμού πινάκων***, μια κοινή προσέγγιση για τη παραλληλοποίηση αυτού του αλγόριθμου, και έπειτα θα παραχθεί μια ***πλήρη προσέγγιση*** σύμφωνα με τη cache για τη παραλληλοποίηση αυτού του αλγορίθμου. Ο σκοπός αυτής της ανάλυσης είναι να διδάξει μια μεθοδολογία για το πώς να ερμηνευτούν τα ***στατιστικά στοιχεία*** που συγκεντρώθηκαν κατά τη διάρκεια των δοκιμών και στη συνέχεια πώς να χρησιμοποιηθούν αυτές η ερμηνείες στη βελτίωση του παράλληλου κώδικα.

- Θα εξεταστεί πώς μπορεί να ανέλθει έως και ***80x*** η εκτέλεση του παράλληλου προγραμματισμού χρησιμοποιώντας ένα μόνο επεξεργαστή(4 πυρήνες με HT),αλλά και μέσω μιας απλής σειριακής μεθόδου στο ίδιο σύστημα.



- Ο αλγόριθμος προσεγγίζεται και σειριακά και παράλληλα για να γίνει αντιληπτό το πλεονέκτημα του υπερβαθμωτού προγραμματισμού έναντι του σειριακού.
- Το πρόγραμμα αυτό έχει εκτελεστεί σε έναν επεξεργαστή τεσσάρων πυρήνων και έχει συνταχθεί σε C++.
- Μετά την εκτέλεση του κώδικα παρατηρούμε ότι ο παραλληλοποιημένος αλγόριθμος είναι κατά προσέγγιση 1.97x φορές πιο γρήγορος.
- Παρακάτω δίνονται ο παράλληλος και ο σειριακός κώδικας, τα αποτελέσματα καθώς και οι συγκρίσεις τους.

- Μία τυπική μέθοδος σειριακού υπολογισμού πολλαπλασιασμού πινάκων για έναν τετραγωνικό πίνακα φαίνεται παρακάτω:

```
01 // Computes the product of two square matrices.
02 void matrix_multiply(
03 double** m1, double** m2, double** result, size_t size)
04 {
05     for (size_t i = 0; i < size; i++)
06     {
07         for (size_t j = 0; j < size; j++)
08         {
09             double temp = 0;
10             for (int k = 0; k < size; k++)
11             {
12                 temp += m1[i][k] * m2[k][j];
13             }
14             result[i][j] = temp;
15         }
16     }
17 }
```

ΠΑΡΑΛΛΗΛΗ ΜΕΘΟΔΟΣ

- Ο παραπάνω κώδικας συγκρίνεται με την παραλληλοποιημένη μορφή του, η οποία χρησιμοποιεί **collection parallel_for**.

```
01 // Compute the product of two square matrices in parallel.
02 void parallel_matrix_multiply(
03 double** m1, double** m2, double** result, size_t size)
04 {
05     parallel_for (size_t(0), size, [&](size_t i)
06     {
07         for (size_t j = 0; j < size; j++)
08         {
09             double temp = 0;
10             for (int k = 0; k < size; k++)
11             {
12                 temp += m1[i][k] * m2[k][j];
13             }
14             result[i][j] = temp;
15         }
16     });
17 }
```

ΕΦΑΡΜΟΓΗ ΣΤΟ MSDN

Το MSDN ανέφερε αύξηση της απόδοσης σε ένα σύστημα 4 επεξεργαστών για πολλαπλασιασμό 750 x 750 πινάκων ως εξής:

- Σειριακό: 3853 (ticks)
- Παράλληλο: 1311 (ticks)

Περίπου ***2.94x αύξηση ταχύτητας***.

Αυτό δεν είναι μια ιδανική κατάσταση κλιμάκωσης στην οποία δεν παράγεται μια 4x αύξηση ταχύτητας χρησιμοποιώντας 4 επεξεργαστές. Όμως λαμβάνοντας υπόψη ότι πρέπει να υπάρχει μια επιβάρυνση στην εγκατάσταση, η αύξηση ταχύτητας 2.94x δεν είναι και τόσο κακή σε αυτό τον μικρό πίνακα.

Ας γίνει γνωστό ότι υπάρχει μόνο 25% του χώρου διατεθειμένο για βελτίωση.

ΑΝΑΛΥΣΗ ΣΤΟΙΧΕΙΩΝ

- Η ανάλυση αυτή δεν σχεδίασε την ***κλιμάκωση*** ως συνάρτηση του N (η μία διάσταση του τετραγωνικού πίνακα), έτσι είναι δύσκολο να δώσει την μορφή της καμπύλη του κέρδους απόδοσης ως συνάρτηση του N.

- Παρά το γεγονός ότι η ανάλυση αυτή γράφτηκε για να τονίσει τη χρήση του parallel_for στο ταυτοχρονισμό MS, ένας προγραμματιστής παράλληλου προγραμματισμού μπορεί να ***έχανε προβάδισμα*** στην παραδοχή ότι αυτό το παράδειγμα απεικονίζει μια παράλληλη συνάρτηση πολλαπλασιασμού πινάκων. Μετά από όλα αυτά, η ανάλυση περιγράφει μια παράλληλη μέθοδο για τον πολλαπλασιασμό πίνακα (γράφτηκε σε ένα άρθρο MSDN - μια έγκυρη πηγή).

ΠΡΟΣΑΡΜΟΓΗ ΣΕ QUICKTHREAD (1)

- Δεν υπάρχει η δυνατότητα του Visual Studio 2010, οπότε ο κώδικας έχει προσαρμοστεί για να χρησιμοποιείται σε ***QuickThread***
- Η προσαρμογή είναι σχετικά εύκολη. Έγιναν αλλαγές των include αρχείων και υπήρχαν μικρές διαφορές στο συντακτικό.

```
01 // Computes the product of two square matrices in parallel.
02 void parallel_matrix_multiply(
03 double** m1, double** m2, double** result, size_t size)
04 {
05     parallel_for(
06         0, size,
07         [&](intptr_t iBegin, intptr_t iEnd)
08         {
09             for(intptr_t i = iBegin; i < iEnd; ++i)
10             {
11                 for (intptr_t j = 0; j < size; j++)
12                 {
13                     double temp = 0;
14                     for (intptr_t k = 0; k < size; k++)
15                     {
16                         temp += m1[i][k] * m2[k][j];
17                     }
18                     result[i][j] = temp;
19                 }
20             }
21         }
22     );
23 }
```

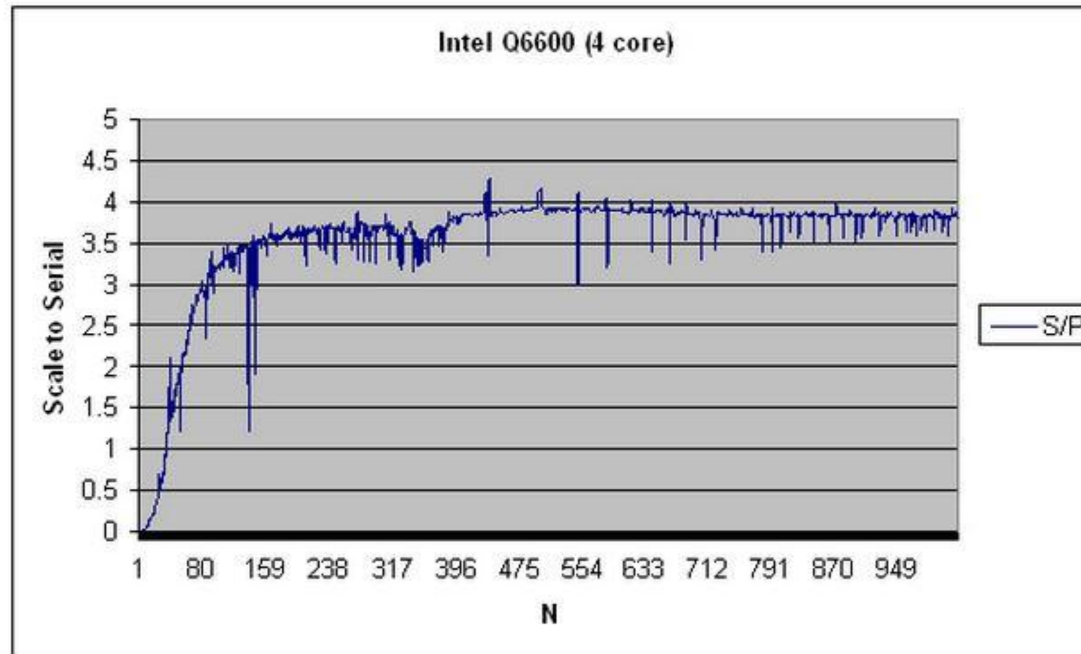
ΠΡΟΣΑΡΜΟΓΗ ΣΕ QUICKTHREAD (2)

- Στη γλώσσα QuickThread, η parallel_for περνάει το μισό ανοιχτό εύρος ως όρισμα στην συνάρτηση, ***σε αντίθεση*** με την parallel_for του VS 2010 τη συλλογή ταυτοχρονισμού που περνά τον ενιαίο δείκτη μέσα στο σώμα της συνάρτησης.

  Η QuickThread επέλεξε να περάσει το ήμισυ ανοικτό εύρος, σε αντίθεση με ένα ενιαίο δείκτη, διότι ***γνωρίζοντας το εύρος***, ο προγραμματιστής ή / και ο μεταγλωττιστής μπορεί να ***βελτιστοποιήσει*** καλύτερα τον κώδικα εντός του βρόχου. Αυτός ο βρόχος θα μπορούσε τόσο εύκολα έχει γραφτεί χρησιμοποιώντας OpenMP ή Cilk + +, ή TBB.

ΠΑΡΑΛΛΗΛΗ ΔΙΑΚΥΜΑΝΣΗ (1)

- Χρησιμοποιώντας τον τροποποιημένο κώδικα QuickThread ο οποίος τρέχει σε Windows XP x64 και χρησιμοποιώντας έναν επεξεργαστή Intel Q6600 4 Core χωρίς Hyper Threading, προκύπτει η εξής ***κλιμάκωση*** :



ΑΝΑΛΥΣΗ ΠΑΡΑΛΛΗΛΗΣ ΔΙΑΚΥΜΑΝΣΗΣ (1)

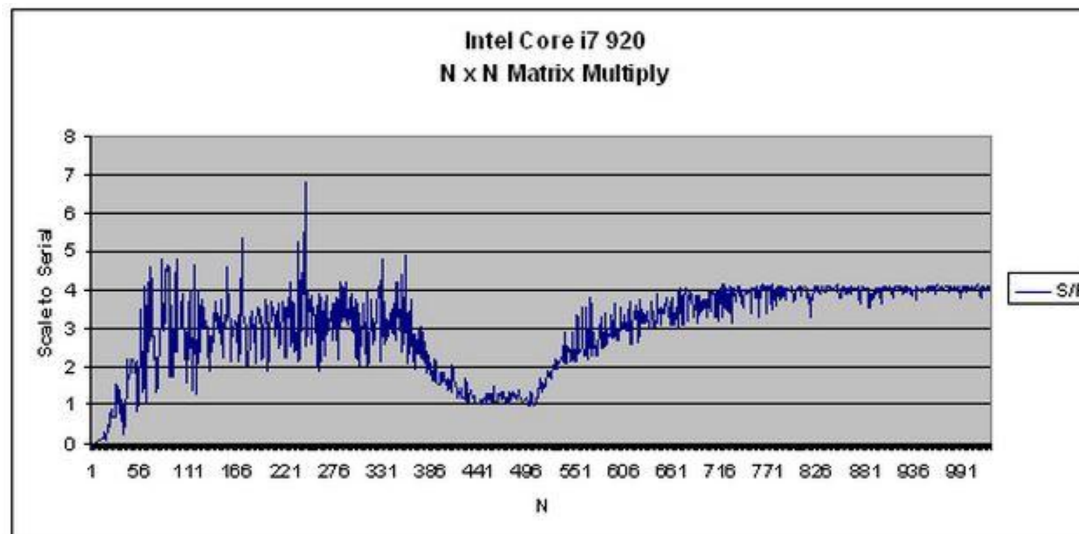
- Υπάρχει μια διακύμανση της τάξης των ***3.77x*** από Ν μεταξύ του 128 και του 1024. Και λαμβάνοντας υπόψη τα τέσσερα νήματα που εμπλέκονται, το πάνω διάγραμμα δείχνει τη παράλληλη έκδοση του σειριακού κώδικα αποδίδοντας ένα συντελεστή προσαύξησης 0.94(μέγεθος/αριθμός πυρήνων). Αυτός είναι ένα ***αρκετά καλός*** συντελεστής κλιμάκωσης.

- Το μοντέλο του επεξεργαστή δεν έχει αναφερθεί για το τεστ του Visual Studio γι' αυτό είναι δύσκολο να ειπωθεί ο λόγος γιατί το Quick Thread αποδίδει 3.77x βελτίωση σε 4 πυρήνες, ενώ το VS αποδίδει 2.94x βελτίωση σε 4 επεξεργαστές.

- Πρώτο βήμα που πρέπει να γίνει είναι η επιλογή ενός πρόσθετου(διαφορετικού) σύνολο δεδομένων του δείγματος.

ΠΑΡΑΛΛΗΛΗ ΔΙΑΚΥΜΑΝΣΗ (2)

- Η απόδοση σε έναν επεξεργαστή υποστήριξης L3 cache και Hyper Threading (HT):
- Με την εκτέλεση του προγράμματος σε έναν Intel Core i7 920, 4 πύρηνο με HT (8 υλικά νήματα- αλλά μόνο 4 κυμαινόμενα μονοπάτια εντολών) προκύπτει μια τελείως διαφορετική γραμμή τάσης:



ΑΝΑΛΥΣΗ ΠΑΡΑΛΛΗΛΗΣ ΔΙΑΚΥΜΑΝΣΗΣ (2)

- Προκύπτουν ορισμένα ερωτήματα.
  - Γιατί υπάρχει πτώση όταν το N κυμαίνεται από 350 σε 570?
  - Όταν η απόδοση ανακάμπτει, γιατί υπάρχει βελτίωση 4x αντί για 8x?
  - Γιατί υπάρχει οδοντωτή γραμμή(θόρυβος) προς χαμηλότερα N?

- Η ***πτώση*** οφείλεται σε ***εκδιώξεις της cache*** που προκαλείται από ένα νήμα που επιδράει αρνητικά με τα άλλα νήματα.
- Υπάρχει λίγο πάνω από ***4x επιδόσεις*** επειδή ο Core i7 920 είναι ένας τετραπύρηνος επεξεργαστής ικανός για ***4 ρεύματα εκτέλεσης κινητής υποδιαστολής***(αν και έχει 8 υλικά νήματα για ρεύματα ακέραιης εκτέλεσης). Στις υψηλές τιμές του N θα φαίνεται ότι μεγιστοποιούμε τις ικανότητες του επεξεργαστή, 4 πυρήνες == 4x αύξηση της απόδοσης.
- Η γραμμή τάσης είναι ακανόνιστη, λόγω ότι μόνο ένα τρέχον δείγμα έχει ληφθεί καθώς και λόγω ***των μικρών χρόνων λειτουργίας***, όταν το N είναι χαμηλό. Ακόμα, η μεταβλητή γενικά στην έναρξη της ομάδας από τα νήματα(τώρα 8 νήματα) είναι πιο αισθητή στην αριστερή πλευρά της καμπύλης. Κατά μέσο όρο ένας μεγαλύτερος αριθμός των γύρων θα εξομάλυνε αυτή τη γραμμή, αλλά αυτό καταναλώνει περιττό χρόνο από τον developer. Επιπρόσθετα, η μεγάλη ακίδα περίπου στο 250 δείχνει ότι η σειριακή εκδοχή πήρε ένα μεγάλο αριθμό υποδιαίρεσης σε κείνο το σημείο λόγω των εξωτερικών συνθηκών ελέγχου της εφαρμογής (επέκταση του αρχείου σελίδας ή άλλη δραστηριότητα για το σύστημα.

ΒΕΛΤΙΩΣΗ ΕΠΙΔΟΣΕΩΝ (1)

- Υπάρχει τρόπος για να βελτιωθούν αυτές οι επιδόσεις ή και να διορθωθεί η πτώση στην απόδοση για Ν μεταξύ 350 και 570?

- Το κύριο υπολογιστικό τμήμα του κώδικα τόσο για σειριακή και παράλληλη είναι το ίδιο:

```
1  for (intptr_t j = 0; j < size; j++)
2  {
3      double temp = 0;
4      for (intptr_t k = 0; k < size; k++)
5      {
6      temp += m1[i][k] * m2[k][j];
7      }
8      result[i][j] = temp;
9  }
```

ΒΕΛΤΙΩΣΗ ΕΠΙΔΟΣΕΩΝ (2)

- Αξίζει να σημειωθεί ότι η δήλωση του αποτελέσματος του πολλαπλασιασμού

```
1 temp += m1[i][k] * m2[k][j];
```

χρησιμοποιεί k για να δείξει τη στήλη του πίνακα m1 και τη γραμμή του πίνακα m2.

Αυτό το μοτίβο πρόσβασης έχει δύο προβλήματα:

Δεν είναι φιλικό προς τη cache.

Δεν μπορεί να αποτελέσει αντικείμενο διανυσματοποίησης από τον compiler .

Ας δούμε τον καθορισμό αυτών των προβλημάτων.

ΔΕΙΓΜΑΤΑ CLICK++

- Στα δείγματα προγραμμάτων Click++ , τα οποία είναι (ή θα είναι) διαθέσιμα στην Intel Parallel Studio , υπάρχει ένα δείγμα προγράμματος που δείχνει φιλική συμπεριφορά προς τη cache στην εφαρμογή του πολλαπλασιασμού των πινάκων:

```
01 // Multiply double precision square n x n matrices. A = B * C
02 // Matrices are stored in row major order.
03 void matrix_multiply(double* A, double* B, double* C, unsigned int n)
04 {
05     if (n < 1) {
06         return;
07     }
08
09     cilk_for(unsigned int i = 0; i < n; ++i) {
10 // This is the only Cilk++ keyword used in this program
11 // Note the order of the loops and the code motion of the i*n and k*n
12 // computation. This gives a 5-10 performance improvement over
13 // exchanging the j and k loops.
14     int itn = i * n;
15        for (unsigned int k = 0; k < n; ++k) {
16            for (unsigned int j = 0; j < n; ++j) {
17                int ktn = k * n;
18                // Compute A[i,j] in the inner loop.
19                A[itn + j] += B[itn + k] * C[ktn + j];
20            }
21        }
22     }
23     return;
24 }
```

ΑΝΑΛΥΣΗ ΔΕΙΓΜΑΤΩΝ CLICK++ - ΔΗΛΩΣΕΙΣ ΣΥΝΑΡΤΗΣΕΩΝ

- Οι μεταβλητές itn (i times n) και ktn(k times n) κάνουν μεγάλα βήματα μέσα από τους πίνακες. Οι δείκτες του πίνακα δείχνουν σε μονοδιάστατους πίνακες που είναι σε σειρά. Αυτό σημαίνει ότι κάθε γραμμή επισυνάπτεται στη προηγούμενη γραμμή σε έναν μεγάλο ενιαίο μονοδιάστατο πίνακα. Ο οποίος στη συνέχεια αναπροσαρμόζεται με το i*n και/ή το k*n.

- Cilk + + δηλώσεις συναρτήσεων :

```
1 void matrix_multiply(double* A, double* B, double* C, unsigned int n);
```

Σημειώστε ότι τα double * στο δισδιάστατο πίνακα είναι δείκτες.

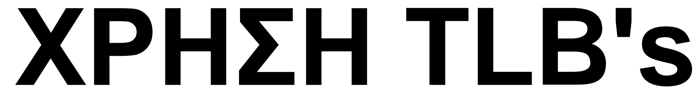
ΔΗΛΩΣΕΙΣ ΣΥΝΑΡΤΗΣΕΩΝ

- Σειριακές και παράλληλες δηλώσεις συναρτήσεων:

```c
void matrix_multiply(
    double** m1, double** m2, double** result, size_t size);
void parallel_matrix_multiply(
    double** m1, double** m2, double** result, size_t size);
```

Σημειώστε ότι τα double ** στο δυσδιάστατο πίνακα είναι δείκτες

- Η ***ένωση των γραμμών*** κάνει κατά κύριο λόγο ένα ευεργετικό πράγμα στο πρόγραμμά . Η ένωση των γραμμών εξαλείφει μια φέρουσα μνήμη ενός δείκτη στη γραμμή (όπως γίνεται με τους πίνακες των δεικτών των γραμμών). Αυτό χρησιμοποιεί λιγότερες Εικονικές Μνήμες Μετάφρασης Μέσω των Buffers. Και αυτό ***βελτιώνει τη χρήση της cache.***



ΧΡΗΣΗ TLB's

- Χρήσεις : 1 TLB για το κώδικα + 1 TLB για το σωρό(could be 0) + 1TLB για m1 πίνακα γραμμών + 1 TLB για m1 δεδομένων + 1 TLB για m2 πίνακα γραμμών + 1 TLB για m2 δεδομένων = 6 (or 5) TLB's.

```
1 temp += m1[i][k] * m2[k][j]; // serial/parallel
```

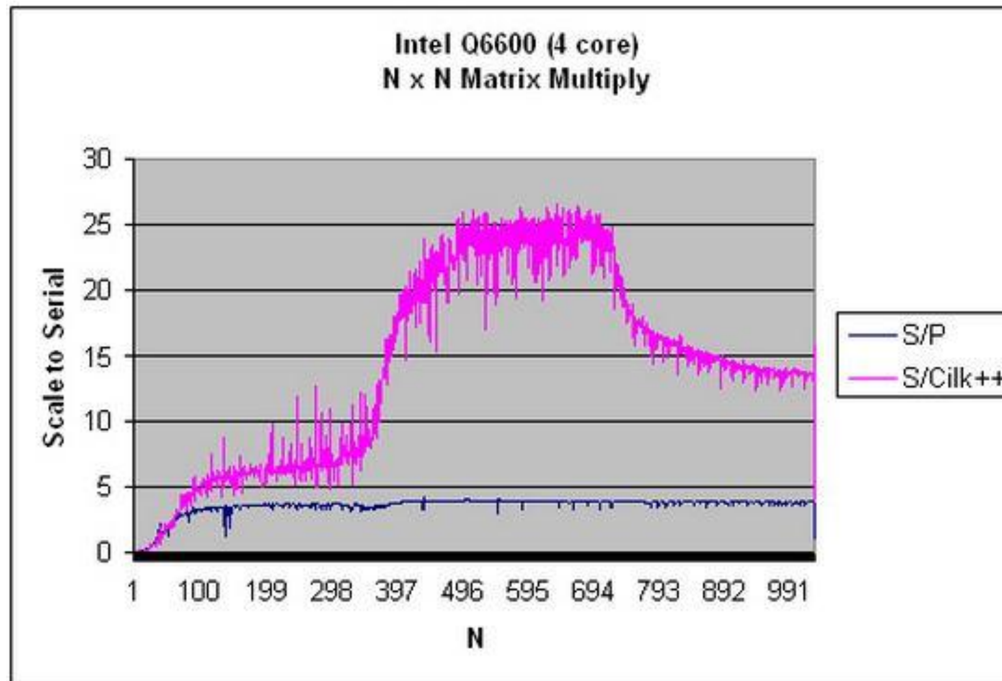
- Λαμβάνοντας υπόψη τη μέθοδο ένωσης των γραμμών που χρησιμοποιείται από το Click++ :
- Χρήσεις : 1 TLB για το κώδικα + 1 TLB για το σωρό(could be 0) + 1 TLB για m1 δεδομένων + 1 TLB για m2 δεδομένων = 4 (or 3) TLB's.

```
1 A[itn + j] += B[itn + k] * C[ktn + j];
```

- Η μείωση του αριθμού των απαιτούμενων TLB θα πρέπει να παράγει ένα πλεονέκτημα.

ΔΙΑΓΡΑΜΜΑ ΑΠΟΔΟΣΗΣ INTEL Q6600

- Το αντίστοιχο διάγραμμα:



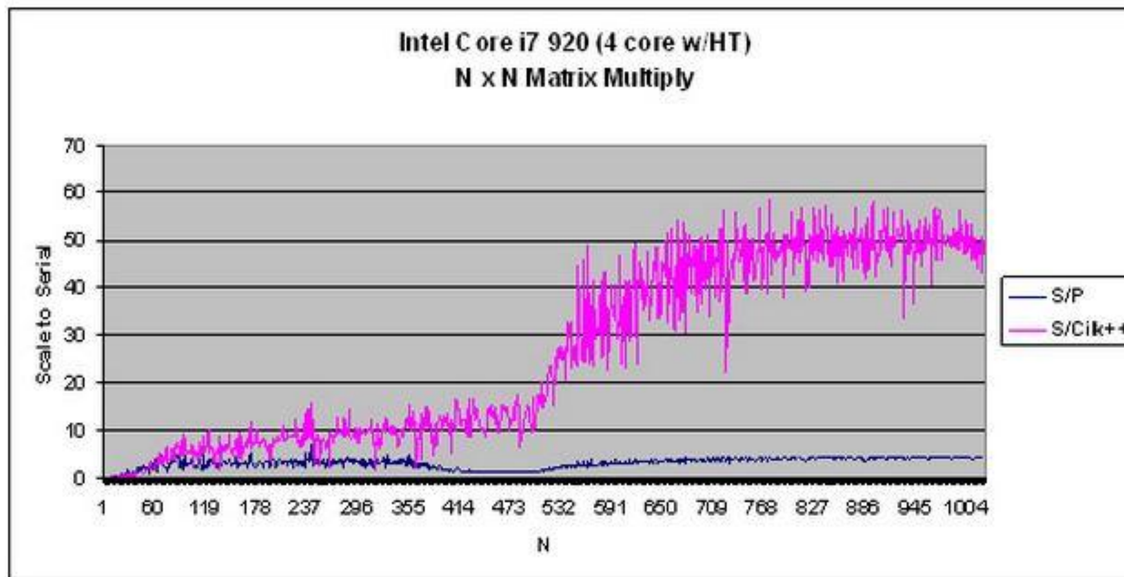
ΑΝΑΛΥΣΗ ΔΙΑΓΡΑΜΜΑΤΟΣ ΑΠΟΔΟΣΗΣ INTEL Q6600

- Η τεχνική του Click++ στο Q6600 έχει μια ωραία κορυφή μεταξύ του 400 και του 700 όταν φθάνει 25x απόδοση σε σχέση με το σειριακό αλλά πέφτει περίπου στο 14x στο πιο υψηλό εύρος. Αυτό αντιπροσωπεύει μια 3x με 6x βελτίωση έναντι της παράλληλης έκδοσης του προτύπου κώδικα πολλαπλασιασμού πινάκων. Η κατάργηση των πινάκων των δεικτών των γραμμών κάνουν μια σημαντική διαφορά.
- Η τεχνική που χρησιμοποιείται από το δοκιμαστικό πρόγραμμα Cilk + + αναφέρεται ως "Cilk + +" στα διαγράμματα και στο σώμα του κειμένου αυτού.

Η τεχνική που χρησιμοποιείται μπορεί να χρησιμοποιηθεί και από άλλες παράλληλες γλώσσες προγραμματισμού.

ΔΙΑΓΡΑΜΜΑ ΑΠΟΔΟΣΗΣ INTEL Core i7 920

- Τώρα, κοιτάζοντας το Core i7 920 επεξεργαστή προκύπτει:



- Η μέθοδος Click++ στο Core i7 920 επιτυγχάνει σχεδόν 50x κέρδος απόδοσης από μια σειριακή μέθοδο περίπου N=800(12x από μια παράλληλη), αλλά φαίνεται σαν να αφήνεται μετά το 1024(όπως έκανε νωρίτερα στο 700 για το Q66000). Τα επιπλέον σημεία πρέπει να συλλέγονται.

- *Αυτό διδάσκει:* κατασκευάζοντας δισδιάστατους πίνακες σαν ένα πίνακα από δείκτες σε έναν μονοδιάστατο πίνακα φαίνεται καλό, αλλά μπορεί να σας κοστίσει ακριβά.

  Είναι σαφές ότι η πιο ***αποτελεσματική οδός*** είναι να χρησιμοποιήσετε την ***ένωση γραμμών*** για να μειωθεί ο αριθμός των TLBs και οι αντίστοιχες καταχωρήσεις στη μνήμη cache.

ΑΛΛΑΓΕΣ ΣΕ ΚΛΑΣΕΙΣ

- Αλλά αυτό σημαίνει αλλαγή

```
1 double** A; // referenced as A[iRow][iCol]
```

σε μερικές κλάσεις

```
1 Array2D<double>  A;
```

- Και τότε αλλάζουν οι εκχωρήσεις και οι αναφορές (ή έντονες λειτουργίες του προτύπου).

ΠΕΡΙΓΡΑΦΕΙΣ ΠΙΝΑΚΩΝ

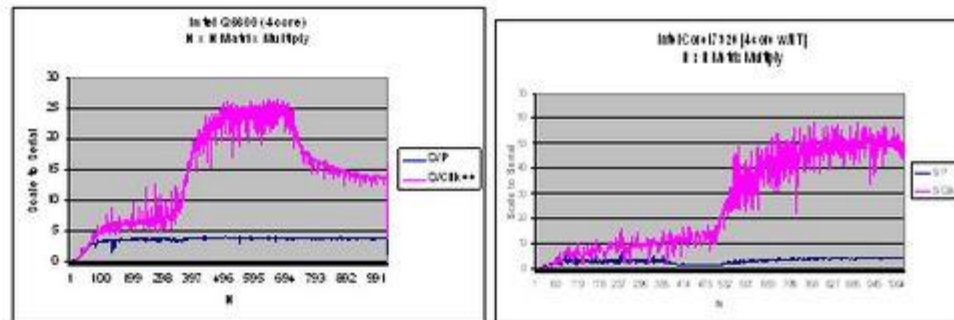
- Αν ελεγχθεί αυτό από μια διαφορετική οπτική γωνία τα αντικείμενα των κλάσεων των πινάκων είναι ουσιαστικά περιγραφείς πινάκων.

  Περιγραφείς πινάκων είναι μια πολύ καλή τεχνική που χρησιμοποιείται από τη FORTRAN.

- Με τη βελτίωση των 12x σε μια παράλληλη έκδοση σε μια σειριακή εφαρμογή δείχνει ότι η προσοχή σε θέματα cache πραγματικά πληρώνεται ακριβά.

NON-CACHE ΚΑΙ CACHE ΤΕΧΝΙΚΕΣ

- Το διάγραμμα συγκρίνει μια non-cache ευαίσθητη σειριακή τεχνική με μία cache ευαίσθητη παράλληλη τεχνική.



- ***Το πρώτο πράγμα*** που κάθε προγραμματιστής παράλληλου προγραμματισμού θα πρέπει να γνωρίζει είναι: πρώτα να ***βελτιωθεί ο σειριακός αλγόριθμος***, και στη συνέχεια να αντιμετωπιστεί το πρόβλημα μέσω παραλληλισμού (κρατώντας εγγενής μικρές συναρτήσεις διανύσματος για το τέλος).

ΑΝΑΛΥΣΗ ΕΣΩΤΕΡΙΚΟΥ ΒΡΟΧΟΥ

- Αν γίνει ανάλυση του πιο εσωτερικού βρόχου της σειριακής συνάρτησης παρατηρείται:

```
for (int k = 0; k < size; k++)
{
    temp += m1[i][k] * m2[k][j];
}
```

- Το θέμα της παραδοσιακής C / C + + πρακτικής προγραμματισμού αντιμετωπίστηκε χρησιμοποιώντας μια σειρά από δείκτες σε σειρές.
- Γίνεται φανερό ότι η διανομή με αυτή την πρακτική είχε μια 6x έως 12x βελτίωση στην απόδοση.

- Τι άλλο μπορεί να γίνει;

ΑΝΑΛΥΣΗ ΔΕΙΚΤΩΝ ΤΟΥ ΕΣΩΤΕΡΙΚΟΥ ΒΡΟΧΟΥ

- Ας μη ληφθεί υπόψη το θέμα των TLBs και τον πίνακα των γραμμών για μια στιγμή. Κατά την εξέταση της κύριας υπολογιστικής δήλωσης διαπιστώνουμε ότι, ενώ ο δείκτης k στο m1 [i] [k] διαδοχικά προσπελαύνει μνήμη, ο δείκτης k στο m2 [k] [j] δεν το κάνει. Αυτό σημαίνει ότι ο μεταγλωττιστής δεν είναι σε θέση να διανυσματοποίησει αυτό το βρόχο. Και πιθανώς χειρότερα, κατεβαίνοντας τις γραμμές σε διαστήματα ορισμένης απόστασης είναι γνωστό ότι προκαλούνται εξώσεις cache (η βουτιά στην καμπύλη παρατηρείται σε πυρήνα i7 920). Χρησιμοποιώντας διανύσματα δύο φορές μπορεί να επιτύχει μια 2x βελτίωση.

- Να σημειώσετε, ενώ θα μπορούσε να χρησιμοποιηθεί η προσέγγιση που γίνεται στην Cilk + + εφαρμογή (εξάλειψη πίνακα γραμμών από δείκτες), θα επιλεχθεί ένα εναλλακτικό μέσο που θα είναι χρήσιμο αργότερα.

  Πρώτα θα αντιμετωπιστεί η απλή σειριακή και η απλή παράλληλη προσέγγιση για την επισκόπηση των αποτελεσμάτων.

Για τη καλύτερη χρήση του compiler στη διανυσματοποίηση του πολλαπλασιασμού του πίνακα (και την ελαχιστοποίηση των εξώσεων cache), οφείλει να μεταφερθεί ο πίνακας m2 (στο m2t), και στη συνέχεια να μεταφερθούν και οι δείκτες στον εσωτερικό βρόχο.

Για τη μεταφορά του πίνακα m2 θα πρέπει να προστεθούν επιπλέον πόροι :

- Εκτέλεση μιας κατανομής (πραγματικά το μέγεθος +1 κατανέμεται με 1 για τους δείκτες της γραμμής)
- Εκτέλεση σε βρόχους μεταφοράς
- Ανάγνωση και την εκ νέου συγγραφή ενός πίνακα (πινάκας m2 στο m2t)

ΕΙΝΑΙ ΛΥΣΗ Η ΜΕΤΑΦΟΡΑ ΤΟΥ ΠΙΝΑΚΑ?

- Μπορεί να υποθέσει κανείς ότι αυτό πρέπει να είναι πιο αργό από τον απλό πολλαπλασιασμό πίνακα.

- Λοιπόν αυτό θα ήταν λάθος . Κάθε κελί στο m2 χρησιμοποιείται N φορές, και κάθε κελί στο M1 χρησιμοποιείται N φορές. Υπάρχει η δυνατότητα της ανταλλαγής , το cache χάνει σε 2N ** 2 αναγνώσεις και σε N * N εγγραφές εναντίον N αναγνώσεις + N / 2 εγγραφές + (2N ** 2) / 2 αναγνώσεις + N / 2 εγγραφές. Το / 2 οφείλεται στο εσωτερικό βρόχο που γίνεται τώρα μια DOT συνάρτηση που λειτουργεί σε δύο διαδοχικούς πίνακες και είναι ένα ιδανικός υποψήφιος για διανυσματοποιήσεις από τον compiler.

ΣΕΙΡΙΑΚΟΣ ΒΡΟΧΟΣ ΜΕ ΤΗΝ ΔΙΑΝΥΣΜΑΤΟΠΟΙΗΣΗ

- Ο σειριακός βρόχος που έχει ξαναγραφτεί (και εσωτερικός βρόχος που έχει μετατραπεί σε μια ενσωματωμένη συνάρτηση) μοιάζει με:

```
01 // compute DOT product of two vectors, return result as double
02 inline double DOT(double* v1, double* v2, size_t size)
03 {
04     double temp = 0.0;
05     for(size_t i = 0; i < size; i++)
06     {
07        temp += v1[i] * v2[i];
08     }
09     return temp;
10 }
```

ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ ΜΕ ΤΗ ΜΕΤΑΦΟΡΑ ΠΙΝΑΚΑ

- Η συνάρτηση του πολλαπλασιασμού πίνακα, με τη μεταφορά μοιάζει τώρα:

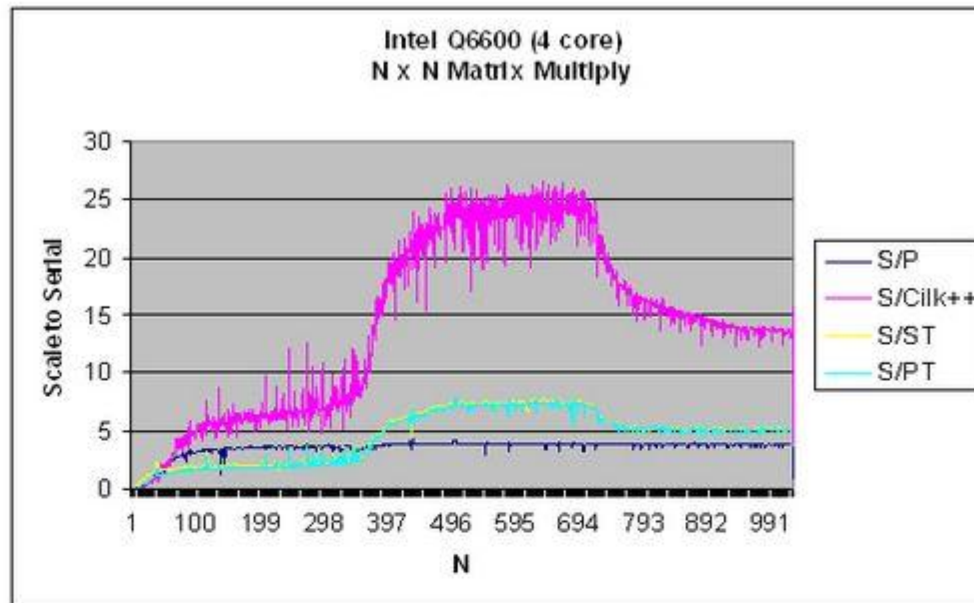
```
01 void matrix_multiplyTranspose(
02 double** m1, double** m2, double** result, size_t size)
03 {
04     // create a matrix to hold the transposition of m2
05     double** m2t = create_matrix(size);
06     // perform transposition m2 into m2t
07     for (size_t i = 0; i < size; i++)
08     {
09         for (size_t j = 0; j < size; j++)
10         {
11             m2t[j][i] = m2[i][j];
12       }
13     }
14     // Now matrix multiply with the transposed matrix m2t
15     for (size_t i = 0; i < size; i++)
16     {
17         for (size_t j = 0; j < size; j++)
18         {
19             result[i][j] = DOT(m1[i], m2t[j], size);
20         }
21     }
22     // remove the matrix that holds the transposition of m2
23     destroy_matrix(m2t, size);
24 }
```

- Να σημειωθεί, σε μια πραγματική εφαρμογή μπορεί να πρέπει να εξεταστεί η διατήρηση των buffers που διαθέτονται για την πρώτη κλήση. Στη συνέχεια, στις επόμενες κλήσεις να θα πρέπει να γίνει έλεγχος εάν η προηγούμενη διανομή ήταν επαρκής για την τρέχουσα κλήση.
  - Αν ναι, να παρακαμφθεί η διανομή.
  - Αν όχι να διαγραφούν οι buffers, και να διατεθούν οι νέοι buffers. Αυτή αποτελεί μια περαιτέρω λεπτομέρεια βελτιστοποίησης.

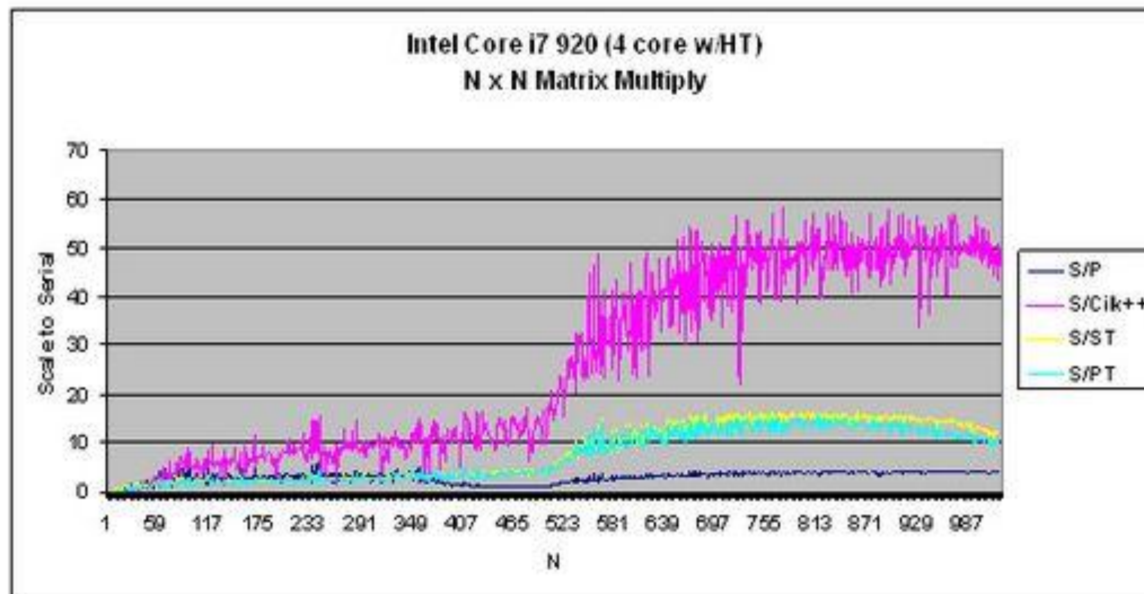
- Θα συμπεριλαμβάνονται οι «διανομή / αναδιανομή» γενικά στα διαγράμματα. Επίσης, να σημειωθεί , αν ο m2 πίνακας πρόκειται να χρησιμοποιηθεί πολλές φορές (π.χ. σε ένα σύστημα φίλτρου), τότε μπορεί να εκτελεστεί η μεταφορά μία φορά, και στη συνέχεια να ξαναχρησιμοποιηθεί ο μεταφερόμενος πίνακας πολλές φορές.

ΣΕΙΡΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕ ΤΗΝ ΜΕΤΑΦΟΡΑ ΣΕ INTEL Q6600

- Ας μελετηθεί πώς το σειριακό πρόγραμμα με την τεχνική μεταφοράς (ST) έρχεται αντιμέτωπο με τη σειριακή χωρίς μεταφορά (S) και του παράλληλου κώδικα (P) χρησιμοποιώντας τη μέθοδο η οποία βασίζεται στην μη-μεταφορά του πίνακα:



ΣΕΙΡΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕ ΤΗΝ ΜΕΤΑΦΟΡΑ ΣΕ INTEL core i7 920



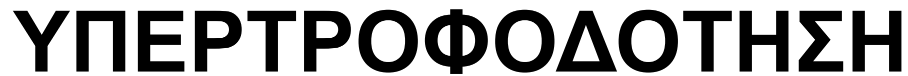
ΑΝΑΛΥΣΗ ΤΕΧΝΙΚΗΣ ΜΕΤΑΦΟΡΑΣ ΠΙΝΑΚΑ

- Η αναθεωρημένη τεχνική, η οποία εξακολουθεί να χρησιμοποιεί έναν ***πίνακα γραμμών*** είναι μια ***σημαντική βελτίωση*** σε σχέση με τις σειριακές και παράλληλες μεθόδους. Η αναθεωρημένη τεχνική δεν προσεγγίζει αυτή του πίνακα μεθόδου εξάλειψης γραμμών του Cilk + + προγράμματος.

- Ο αναθεωρημένος σειριακός κώδικας, χρησιμοποιώντας ένα νήμα, κατακλύζει τον παράλληλο κώδικα χρησιμοποιώντας 4 ή 8 threads για τους δύο διαφορετικούς επεξεργαστές σε υψηλότερες τιμές Ν.

- Αν και 4x είναι η καλύτερη παράλληλη απόδοση από ό, τι πριν, εξακολουθεί να είναι μικρότερη από την Cilk + + μέθοδο.

Αυτές οι πληροφορίες είναι χρήσιμες στην προσπάθεια να προκύψει ένας καλύτερος αλγόριθμος.



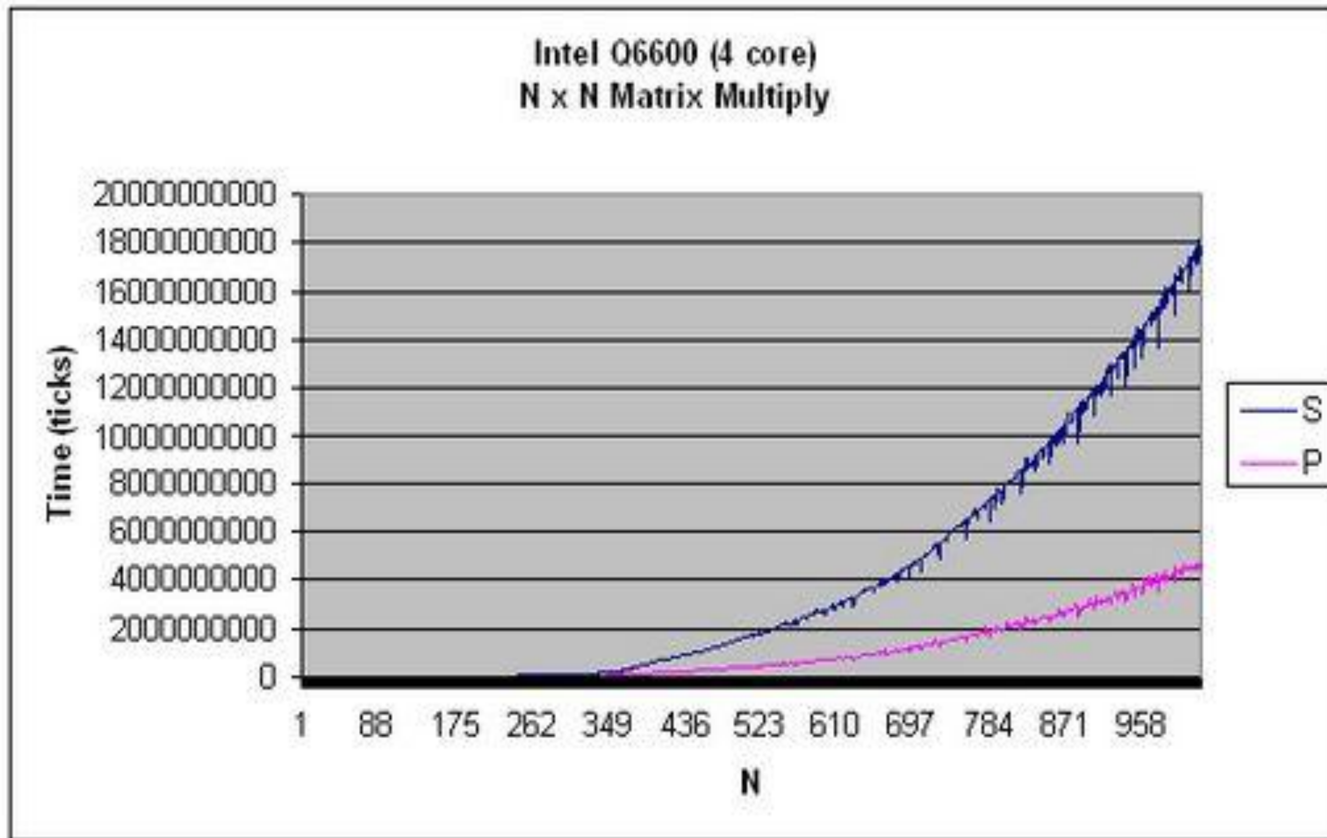
- Γιατί η "υπερτροφοδότηση" ενισχύεται στο 513 στην σειριακή μέθοδο που χρησιμοποιεί μεταφορά πίνακα;

- Αν και η παράλληλη μέθοδος που χρησιμοποιεί μεταφορά πίνακα (PT) που είναι ταχύτερη από 4x παράλληλη (P) είναι στην πραγματικότητα μικρότερη από την απόδοση της σειριακής μεθόδου που χρησιμοποιεί μεταφορά πίνακα. Γιατί;

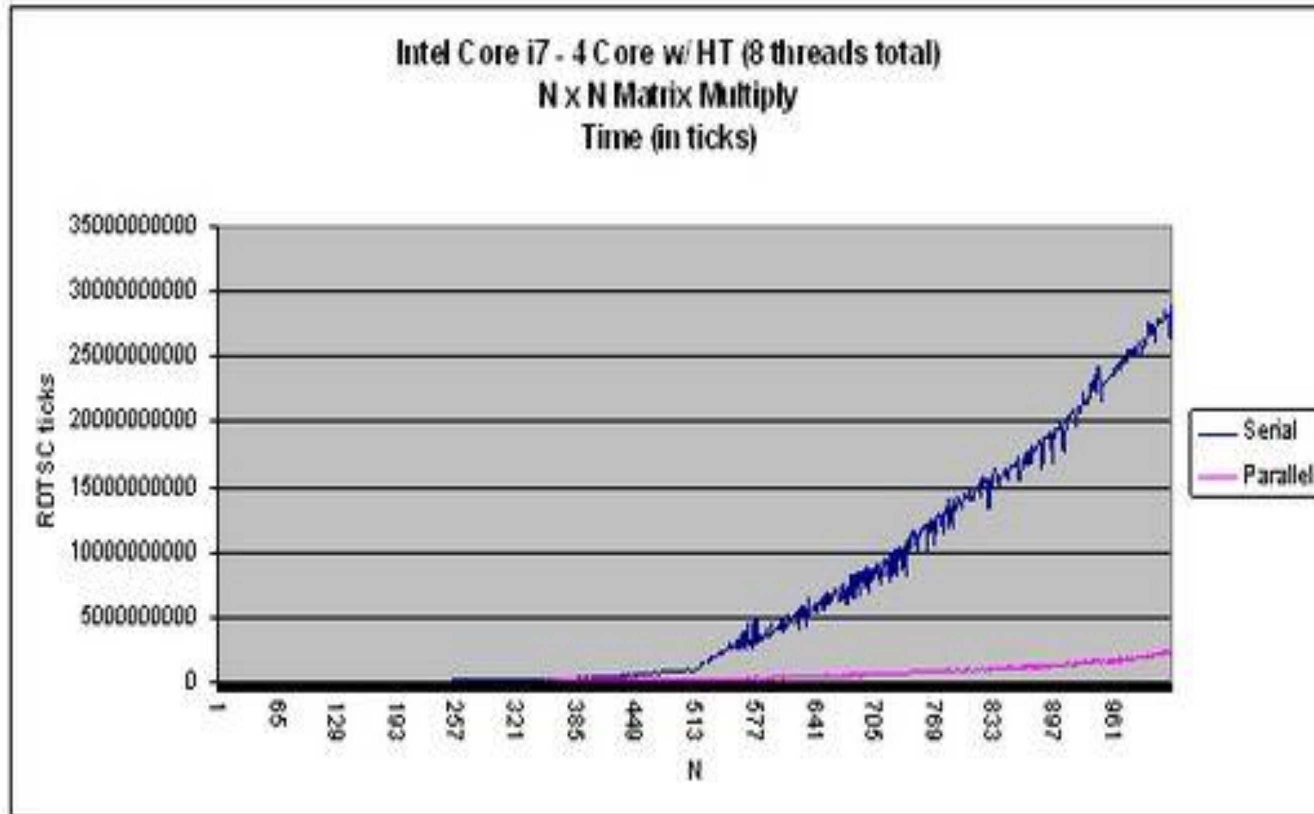
- Η απάντηση σε αυτό είναι ότι το γράφημα γραμμών σε σχέση με την αρχική (την μέθοδο με την μη μεταφορά πίνακα) σειριακή απόδοση.

- Θα εξεταστεί ο πραγματικός χρόνος εκτέλεσης για την (μη μεταφορά) σειριακή μέθοδο για να την επισκόπηση των αποτελεσμάτων.

ΠΡΑΓΜΑΤΙΚΟΣ ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ INTEL Q6600



ΠΡΑΓΜΑΤΙΚΟΣ ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ INTEL core i7 920



ΑΝΑΛΥΣΗ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ

Στο N = 513 παρατηρείται ο σειριακός χρόνος να βιώνει μια δραστική αλλαγή στη κλίση. (Q6600 δείχνει νωρίτερα αλλαγή κλίσης σε περίπου 350). Αυτό που είναι σημαντικό για το N = 513 για το Core i7 920 είναι ότι υπάρχει κυρίως πρόσβαση σε δύο πίνακες δύο διαστάσεων στα 513 x 513. Ο αριθμός των bytes είναι 2 x 513 x 513 x 8 = 4.210.704.

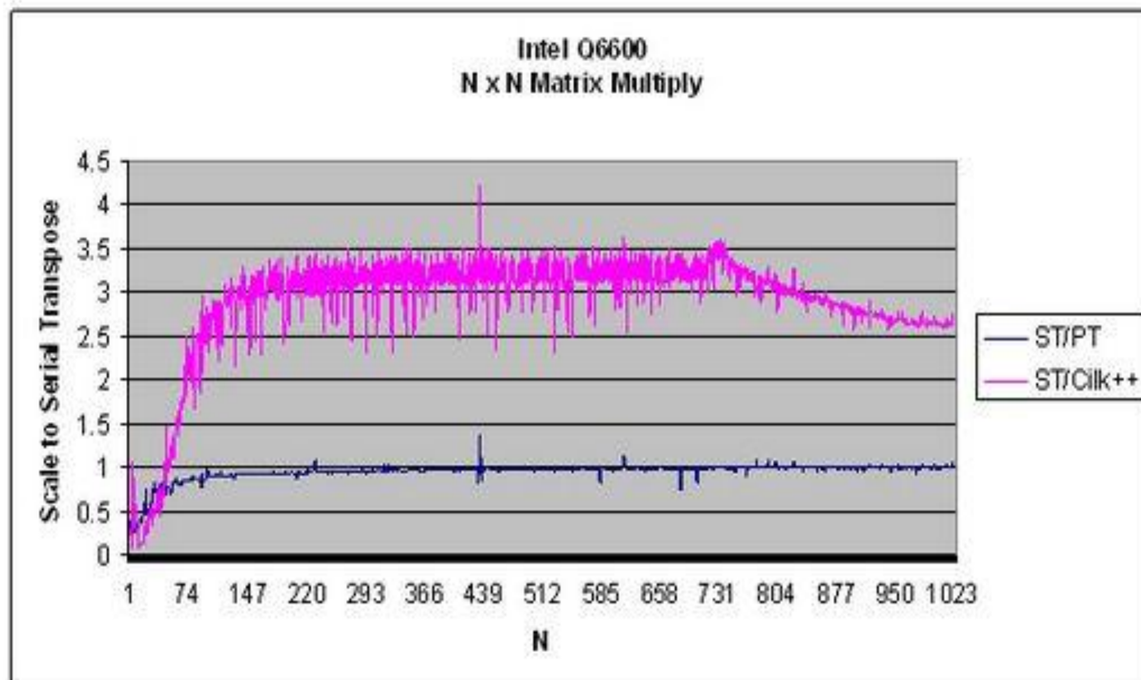
Ο Core i7 920 επεξεργαστής από τις προδιαγραφές που αποκτήθηκαν από το διαδίκτυο αναφέρει ότι έχει:

- L1 cache 32KB εντολών + 32KB δεδομένων (μία για κάθε πυρήνα)
- 256KB L2 cache (ένα για κάθε πυρήνα)
- 8MB L3 cache (shared)

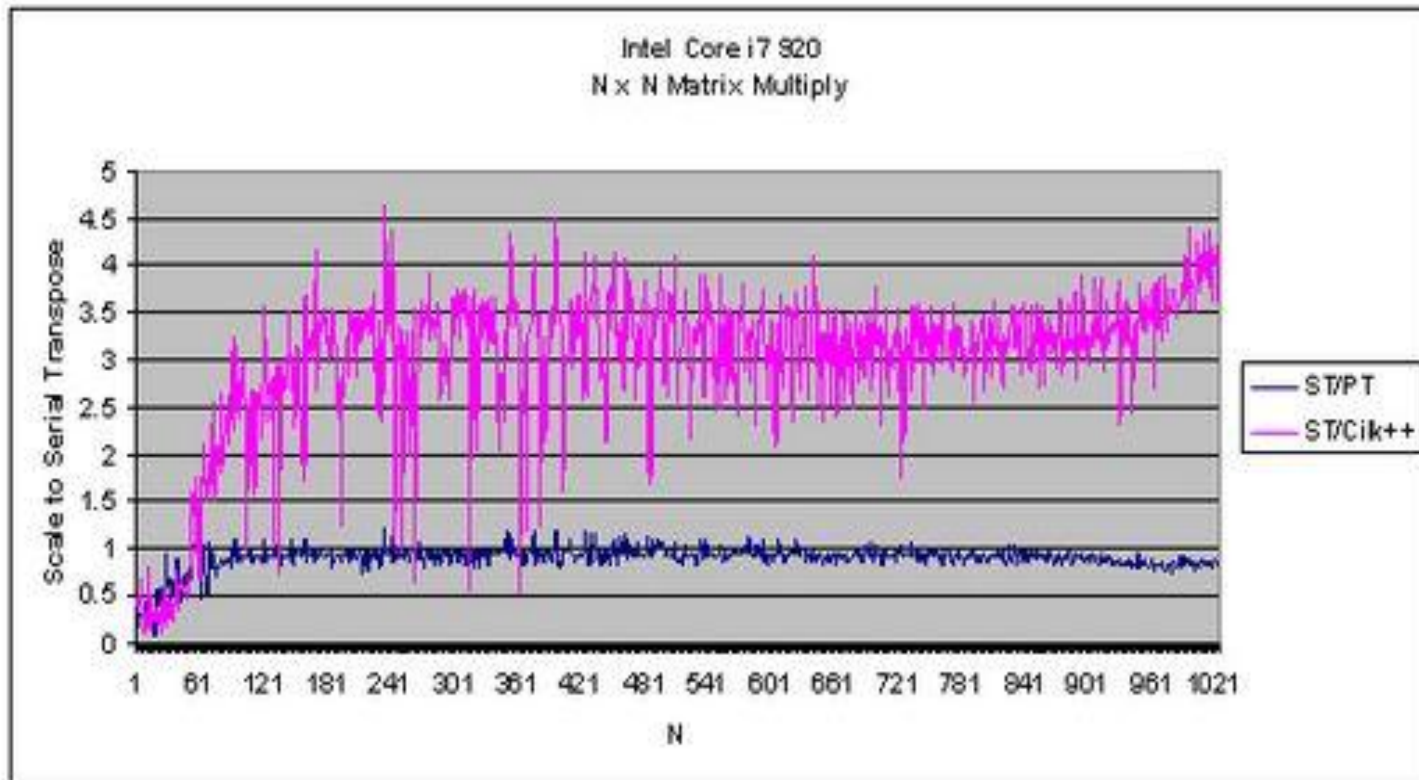
Το σημαντικό κομμάτι των πληροφοριών που αντλήθηκαν είναι η σειριακη μέθοδος που χρησιμοποιεί μεταφορά πίνακα και καταστρέφει την απόδοση της παράλληλης τεχνικής που χρησιμοποιει την τεχνική χωρίς την μεταφορά πίνακα. (Αν και η τεχνική Cilk εξακολουθεί να είναι ταχύτερη 3x από τότε).

ΤΕΧΝΙΚΗ ΜΕΤΑΦΟΡΑΣ ΠΙΝΑΚΑ INTEL Q6600

- Τώρα, όταν η μέθοδος απομάκρυνσης Cilk + + του πίνακα γραμμών συγκρίνεται με τη σειριακή μέθοδο που χρησιμοποιεί μεταφορά πίνακα έχουμε:



ΤΕΧΝΙΚΗ ΜΕΤΑΦΟΡΑΣ ΠΙΝΑΚΑ INTEL core i7 920



ΑΝΑΛΥΣΗ ΤΕΧΝΙΚΗΣ ΜΕΤΑΦΟΡΑΣ

- Η Cilk ++ τεχνική δείχνει περίπου 3.25x αύξηση επιδόσης σε σχέση με τη σειριακη μέθοδο που χρησιμοποιει μεταφορά πίνακα.

- Πιο αξιοσημείωτο, η παράλληλη μέθοδος που χρησιμοποιει μεταφορά πίνακα δεν είναι πιο γρήγορη από ό, τι η σειριακη μέθοδος που χρησιμοποιει μεταφορά πίνακα.
  Γιατί; Και το πιο σημαντικό, έχει σημασία για ποιο λόγο;

- Κατά γενικό κανόνα, όταν παρατηρείτε τον παράλληλο κωδικα σας να μην αποδίδει καλύτερα από σειριακό κωδικα σας, αυτό είναι μια καλή ένδειξη ότι έχετε φτάσει σε ένα πρόβλημα bandwidth μνήμης. Η απόδοση κωδικα της Cilk + + δηλώνει σαφώς το πρόβλημα του εύρος της μνήμης οφείλεται στην κακή χρήση cache από την παράλληλη τεχνική που χρησιμοποιει μεταφορά πίνακα.

- Είναι η Cilk + + μέθοδος, η πιο αποδοτική μέθοδος για την αξιοποίηση του κώδικα;
  Η απάντηση σε αυτό είναι: Όχι.

ΑΝΑΔΙΟΡΓΑΝΩΣΗ ΤΩΝ ΒΡΟΓΧΩΝ ΚΑΙ ΕΥΡΟΣ ΜΝΗΜΗΣ

- Είναι γνωστό ότι με την αναδιοργάνωση των βρόγχων και με τη χρήση προσωρινού πίνακα μπορούμε να παρατηρήσουμε μια αύξηση των επιδόσεων με SSE μικρές βελτιστοποιήσεις διανυσμάτων(ο compiler το κάνει αυτό) αλλά μια μεγαλύτερη αύξηση προήλθε από την καλύτερη αξιοποίηση της cache λόγω της αλλαγής στη διάταξη και στη σειρά της τάξης πρόσβασης. Οι βελτιστοποιήσεις μας ωθούν σε έναν περιορισμό του εύρους ζώνης της μνήμης σύμφωνα με την οποία η σειριακή μέθοδος υπερτερεί πλέον της παράλληλης μεθόδου.

- Ο ***περιορισμός του εύρους ζώνης της μνήμης*** είναι ένας σημαντικός παράγοντας για να εξετάσει και εγγυηθεί μια αλλαγή στην στρατηγική προγραμματισμού: Προκειμένου να επιτευχθούν επιπλέον κέρδη απόδοσης πρόκειται να αντιμετωπιστεί το πρόβλημα από τη σκοπιά που προσπαθεί να κρατήσει τα πρότυπα πρόσβασης δεδομένων στο πλησιέστερο επίπεδο της προσωρινής μνήμης.

- Αλλά πως θα το κάνουμε αυτό?

- Έστω ένα σύστημα με Υπέρ Νημάτωση, όπως στο Core i7 920

- Τα υπέρ-νήματα μέσα σε ένα μονοπύρηνο σύστημα μοιράζονται τα 256KB της L2 cache και, εξαρτώμενα από μια εσωτερική αρχιτεκτονική, μοιράζονται τα 32KB των δεδομένων της L1 cache . Ενώ όλοι οι πυρήνες εντός του socket μοιράζονται τα 8MB της L3 cache(6MB ή 12MB σε άλλα μοντέλα επεξεργαστών).

- Η μέθοδος Click++ χρησιμοποιεί τη κοινή πρακτική του "**διαίρει και βασίλευε**" για την κατάτμηση του πίνακα σε μικρότερες ομάδες εργασίας που ταιριάζει όμορφα στη cache που διαθέτει ένα νήμα. Αυτό είναι ένα καλό σημείο εκκίνησης.

  Το επόμενο στάδιο αφορά πώς να συντονίσουμε τη δραστηριότητα μεταξύ των μνημών cache μέσα στον επεξεργαστή (ες) του συστήματος.

ΥΠΕΡ ΝΗΜΑΤΩΣΗ (2)

- Όσο μειώνετε το μέγεθος του κομματιού, υπάρχει η δυνατότητα της αύξησης του αριθμού από τις αλληλεπιδράσεις του χρονοπρογραμματισμού των νημάτων(που εισέρχονται και εξέρχονται από το πιο εσωτερικό βρόγχο). Ο ***χρονοπρογραμματισμός των νημάτων*** δεν είναι κάτι φθηνό. Στο πρώτο γράφημα, η επιβάρυνση συμβαίνει σε μεγέθη κομματιού 50 x 50.

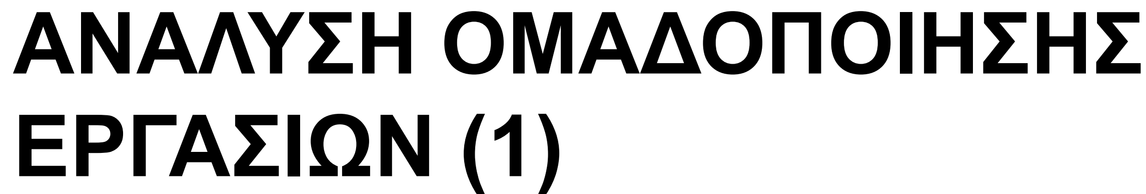
- Η στόχευση καθηκόντων για συγκεκριμένα θέματα, ή η ομαδοποίηση των νημάτων είναι δύσκολο να κάνει αποτελεσματική χρήση των περισσότερων σετ εργαλείων νημάτωσης (πχ MS Parallel Collections, OpenMP, or Cilk++). Η ομαδοποίηση εργασιών σε επίπεδο είναι ένα ενσωματωμένο χαρακτηριστικό του σχεδιασμού της QuickThread.

ΟΜΑΔΟΠΟΙΗΣΗ ΕΡΓΑΣΙΩΝ

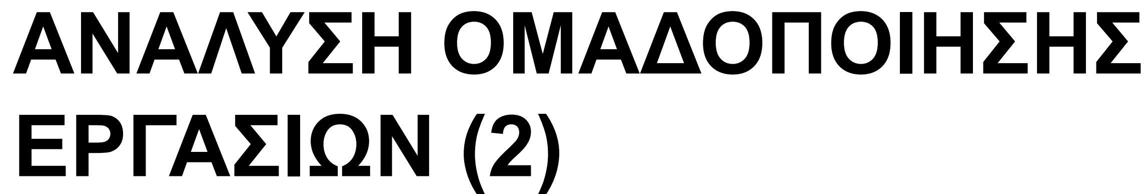
Παράδειγμα:

```
01 // divide the iteration space across each multi-core processor
02 // L3$ specifies by L3 cache
03 //       .OR.
04 // within processor (Socket) for processors without L3 cache
05 parallel_for( OneEach_L3$, intptr_t(0), size,
06      [&](intptr_t iBeginL3, intptr_t iEndL3)
07      {
08           // divide our L3 iteration space by L2 within this threads L3
09           parallel_for( OneEach_Within_L3$ + L2$, iBeginL3, iEndL3,
10                [&](intptr_t iBeginL2, intptr_t iEndL2)
11                {
12                     // Here we are running as the Master thread of a
13                     // 2 team member team (or 1 in the event of older
14                     // processor)
15                     //
16                     // Now bring in our other team member(s)
17                     // form team of threads sharing this threads L2 cache
18                     parallel_distribute( L2$,
19                          [&](intptr_t iTMinL2, intptr_t nTMinL2)
20                          {
21                               // ... (Do Work)
22                          } // [&](intptr_t iTMinL2, intptr_t nTMinL2)
23                     ); // parallel_distribute( L2$,
24                } // [&](intptr_t iBeginL2, intptr_t iEndL2)
25           ); // parallel_for( OneEach_Within_L3$ + L2$, iBeginL3, iEndL3,
26      } // [&](intptr_t iBeginL3, intptr_t iEndL3)
27 ); // parallel_for( OneEach_L3$, intptr_t(0), size,
```

ΑΝΑΛΥΣΗ ΟΜΑΔΟΠΟΙΗΣΗΣ ΕΡΓΑΣΙΩΝ (1)



- Η εξωτερική επανάληψη είναι μια διαίρεση ανά socket, η επόμενη πιο εμφωλευμένη επανάληψη είναι για κάθε L2 cache και το εσωτερικό επίπεδο είναι ένας n-τρόπος χωρισμού από τα νήματα που μοιράζονται την L2(2 νήματα στον Core i7 920 , 2 νήματα στον Q6600)
- Η τεχνική είναι να χρησιμοποιηθεί η parallel_distribute ως το πιο εσωτερικό τμήμα, έπειτα να χρησιμοποιηθεί η κατάτμηση των επαναλήψεων των εξωτερικών βρόγχων με μια μηχανή καταστάσεων που τοποθετείται στο εσωτερικό του επιπέδου parallel_distribute.
- Το parallel_for στο QuickThread εκτελεί τις εργασίες στη συλλογή των νημάτων και τη κατάτμηση του χώρου των επαναλήψεων, αλλά συγκεκριμένα δεν οδηγεί την επανάληψη. Εξαιτίας αυτού του χαρακτηριστικού των QuickThread η περιοχή επανάληψης μπορεί να περάσει βαθύτερα στα στρώματα επανάληψης. Το δεύτερο loop κάνει το ίδιο πράγμα.
- Αυτή η εναλλακτική τεχνική δημιουργεί το περιβάλλον για ένα έξυπνο πλήθος νημάτων που εκτελεί μια συντονισμένη επίθεση κατά του προβλήματος. Αυτά τα νήματα γνωρίζουν ποια νήματα μοιράζονται τη πιο κοντινή cache, ποια νήματα μοιράζονται της πιο μεγάλες cache, και ποια νήματα δε μοιράζονται μνήμες cache με το τρέχων νήμα. Αυτή η αναγνώριση είναι έμμεση από το υποφάσμα των εξωτερικών δύο βρόχων και από τον αριθμό μέλους της ομάδας (iTMinL2) εντός του parallel_distribute.



ΑΝΑΛΥΣΗ ΟΜΑΔΟΠΟΙΗΣΗΣ ΕΡΓΑΣΙΩΝ (2)

- Προσθέτοντας ένα επιπλέον εξωτερικό επίπεδο βρόγχου ανά κόμβο NUMA εύκολα θα μπορούσε να αντιμετωπιστεί χρησιμοποιώντας :

  parallel_for(OneEach_M0$, intptr_t(0), size)

- Η εσωτερική επανάληψη της μηχανής καταστάσεων θα διαιρέσει τον πίνακα εξόδου:

  Socket (L3 cache)
  Core Pair/HT Siblings
  Amongst Core Pair/HT Siblings

- Για να επιτευχθεί αυτό, διασπάται ο πίνακας σε 2x2 κομμάτια με κάθε 2x2 κομμάτι να εξυπηρετείτε από μια ομάδα με 2 νήματα. Η ομάδα με τα 2 νήματα, θα αποκαλείται «***Tag Team***» .
  - Στο πυρήνα i7 θα υπάρχουν 4 Tag Teams, ένα για κάθε πυρήνα, και τα δύο νήματα της κάθε ομάδας να γίνονται «αδέρφια» Υπερ-νήματα μέσα στον ίδιο πυρήνα.
  - Στο Q6600 η Tag Team γίνεται τα δύο νήματα που μοιράζονται την ίδια L2(Q6600 έχει 2 L2 κρυφής μνήμης, κάθε κοινόχρηστο από 2 πυρήνες).

ΚΑΤΕΥΘΥΝΟΜΕΝΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΕΝΑ ΝΗΜΑΤΑ

- Τα «αδέρφια» Υπερ-νημάτωσης μέσα σε ένα πυρήνα έχουν δύο ακέραιες διαδρομές εκτέλεσης, αλλά μόνο μία κυμαινόμενη διαδρομή. Γίνεται κωδικοποίηση για να υπάρξει όφελος από αυτό έχοντας ένα νήμα από τη tag team να εκτελεί τις μεταθέσεις ενώ το άλλο εκτελεί το εσωτερικό γινόμενο των χαμηλότερων τμημάτων 2x2, συν το πάνω δεξί εσωτερικό γινόμενο 1x1 καθώς συμβαίνει η μεταφορά. Το νήμα που θα εκτέλεσε τη μεταφορά έπειτα θα εκτελέσει και το εσωτερικό γινόμενο του αριστερού κελιού του 2x1 τμήμα του κομματιού. Αυτό καθιστά το έργο άνισο μεταξύ των δύο νημάτων της tag team των δύο νημάτων. Επειδή χρησιμοποιούμε το parallel_distribute, το νήμα που τελειώνει πρώτο (συνήθως αυτό που δεν εκτελεί τη μεταφορά), θα αναλάβει το ρόλο της μεταφοράς για το επόμενο κομμάτι.

- Ο παραγόμενος πίνακας διαιρείται σε 2x2 κομμάτια. Οι ομάδες των νημάτων που προκύπτουν από το κοινό ταμείο των νημάτων σε δύο ομάδες νημάτων, κάθε νήμα μοιράζετε το κοντινότερο επίπεδο της cache που είναι δυνατό.

  - Στον πυρήνα i7 αυτά είναι τα «αδέρφια» υπερ-νήματα μέσα σε κάθε πυρήνα
  - Στον Intel Q6600 αυτά είναι δύο πυρήνες που μοιράζονται την ίδια L2 cache
  - Στον AMD Opteron 270 αυτά είναι δύο πυρήνες μέσα στον ίδιο επεξεργαστή με το ίδιο NUMA

ΑΝΑΘΕΣΗ ΣΕ SOCKET

- Η εικόνα δείχνει την αρχική ανάθεση για κάθε κομμάτι socket (χρωματιστό φόντο , μια επαναληπτική ανά ομάδα νημάτων του επεξεργαστή(έντονο περίγραμμα) και 2x2 το ελαφρύτερο περίγραμμα του κομματιού).

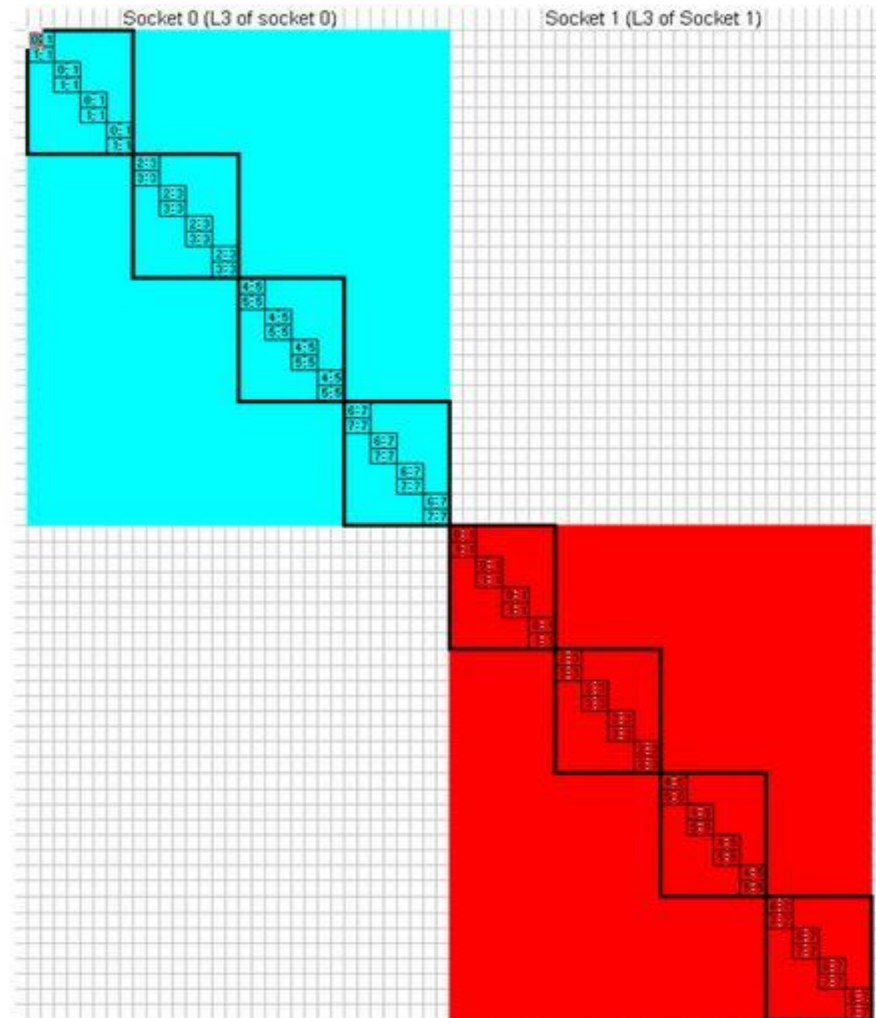
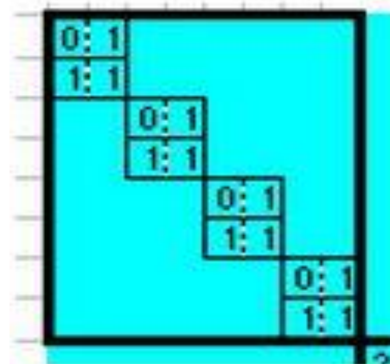


Fig 10

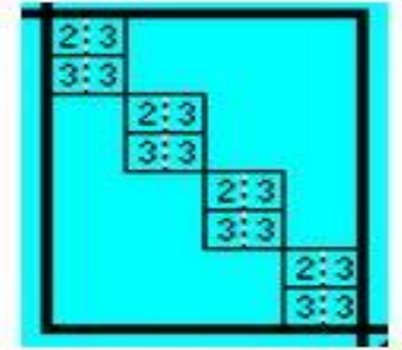
ΑΝΑΛΥΣΗ ΑΝΑΘΕΣΕΩΝ ΣΕ SOCKET (1)

Τα 2x2 κομμάτια που βρίσκονται στη διαγώνιο απαιτούν πρόσθετες εργασίες για τη μεταφορά του m2 πίνακα στον m2t πίνακα. Το ***μάστερ νήμα*** (ζυγός αριθμός από τα 2 νήματα της ομάδας) εκτελεί τη μεταφορά (εντατική προσπέλαση μνήμης) χρησιμοποιώντας την ακέραιη μονάδα εκτέλεσης του πυρήνα HT, ενώ το άλλο μέλος της ομάδας εκτελεί το εσωτερικό γινόμενο από τα 3 κελιά στο 2x2 κομμάτι χρησιμοποιώντας το κυμαινόμενο δείκτη της μονάδας εκτέλεσης του HT πυρήνα. Αυτά τα εσωτερικά γινόμενα πραγματοποιούνται ταυτόχρονα με τη μεταφορά μέσω της κατασκοπείας σχετικά με τη πρόοδο της μεταφοράς που γίνεται από το κύριο νήμα της tag team. Το κύριο νήμα της διμελής ομάδας στη συνέχεια εκτελεί το τελικό εσωτερικό γινόμενο. Κάποιοι πειραματισμοί χρειάζονται ακόμα να γίνουν για να γίνει αντιληπτό αν ο «σκλάβος» νήμα της διμελής ομάδας οφείλει να εκτελέσει όλα τα εσωτερικά γινόμενα .

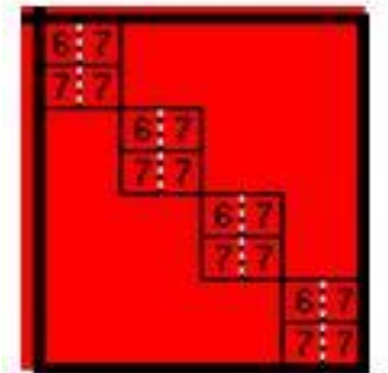


ΑΝΑΛΥΣΗ ΑΝΑΘΕΣΕΩΝ ΣΕ SOCKET (2)

- Τα δύο πρώτα μέλη της ομάδας των νημάτων εργάζονται στα επάνω αριστερά κομμάτια 2x2 (και μεταφοράς) της εν λόγω οριοθετημένης περιοχής. Παράλληλα , η δεύτερη ομάδα δουλεύει στο επάνω πιο αριστερό κομμάτι της εν λόγω οριοθετημένης περιοχής.



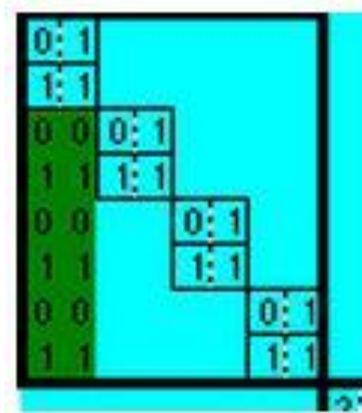
- Και τώρα στο τελευταίο socket, η τελευταία ομάδα εργάζεται στο ανώτερο αριστερό 2x2 κομμάτι (και μεταφορά) της εν λόγω οριοθετημένης περιοχής:



ΑΝΑΛΥΣΗ ΑΝΑΘΕΣΕΩΝ ΣΕ SOCKET (3)

- Η ολοκλήρωση του εν λόγω διαγώνιου κομματιού δε καθορίζεται από το τέλος ενός παράλληλου κατασκευάσματος. Αντ΄αυτού η ολοκλήρωση καθοριζόταν γράφοντας μια ολοκληρωμένη κατάσταση σε μια θυρίδα. Η ολοκλήρωση θα είναι ασύγχρονη με τις δραστηριότητες που συμβαίνουν από τα άλλα νήματα. Και έχει την «επιβάρυνση» που προκαλείται από μη συμπλεκόμενες εγγραφές σε μια κοινόχρηστη θυρίδα μνήμης.

- Μόλις το επάνω πιο αριστερό κομμάτι της διαγωνίου έχει ολοκληρωθεί, η διμελής ομάδα ***συμβουλεύεται μια σημαία*** που δείχνει αν υπάρχει έργο σε ένα στερημένο νήμα(η αρχή σε αυτή τη σημαία θα δείξει καμία εργασία στα στερημένα νήματα). Όταν δεν υπάρχουν καθόλου εργασίες για τα στερημένα νήματα, η διμελής ομάδα δουλεύει πάνω στο 2x2 κομμάτι στη στήλη στην ίδια στήλη του διαγωνίου κομματιού έχει μόλις ολοκληρωθεί:



ΑΝΑΛΥΣΗ ΑΝΑΘΕΣΕΩΝ ΣΕ SOCKET (4)

- Καθορίζετε από τη πράσινη στήλη επάνω. Ωστόσο, κάθε νήμα υπολογίζει ένα 1x2 διπλό εσωτερικό γινόμενο εντός του 2x2 κομματιού. Όταν το 2x2 Κομμάτι είναι πλήρες η διμελής ομάδα συμβουλεύεται μια σημαία που δείχνει αν υπάρχει έργο για στερημένο νήμα, αν δεν υπάρχει εργασία για στερημένο νήμα η ομάδα συνεχίζει προς τα κάτω της στήλης, αν υπάρχει στερημένη εργασία , εξελίσσεται κάτω από τη διαγώνιο.

- Ο στόχος της εργασίας κάτω από τη στήλη ,λίγο μετά τη μεταφορά είναι οι δύο στήλες μόλις μεταφερθούν είναι πιο πιθανό να κατοικούν στη cache (L1,L2 και L3).

- Όταν μια διμελής ομάδα νημάτων ολοκληρώσει τη διαγώνιο, και τα κελιά της στήλης πάνω/κάτω είναι διαγώνια (μαζί με τη ζώνη των δύο μελών της ομάδας). Η ομάδα των νημάτων μετά συμβουλεύεται την ολοκληρωμένη κατάσταση της μεταφοράς από τις άλλες ομάδες νημάτων μαζί με τη μνήμη cache L3(με τον τρόπο της θυρίδας)

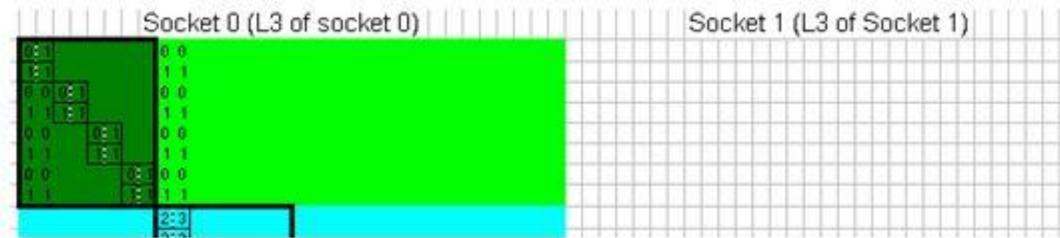
- Η έντονη πράσινη στήλη που επεξεργάζεται από τη πρώτη ομάδα (0/1 στο socket 0), μετά τη εργασία του που έχει ολοκληρωθεί, και μετά τη δεύτερη ομάδα(2/3) που έχει ολοκληρώσει την επάνω διαγώνιο. Τα οποία στατιστικά θα γίνουν από τη στιγμή που η θυρίδα έχει ενημερωθεί. (μετά από 4 μεταφορές με εσωτερικά γινόμενα, συν 12 εσωτερικά γινόμενα με χρονική καθυστέρηση)

- Κάθε ομάδα νήματος κάνει το ίδιο. Καθώς η ομάδα νήματος εξελίσσεται πάνω/κάτω από τις στήλες των διαγωνίων των ομάδων που μοιράζονται τη cache L3 , εάν διαπιστώσουν ανολοκλήρωτη μια καθορισμένη στήλη μέσα στην L3, αυτό σηματοδοτεί μια «στερημένη εργασία» έτσι ώστε η cache L3 οι ομάδες διαμοιρασμού να μπορούν να διακόψουν την επεξεργασία της στήλης και προκαταβολικά στην επόμενη διαγώνια διαδικασία πριν την ολοκλήρωση της τρέχουσας στήλης.

- Κάθε ομάδα , με εξαίρεση την αναφορά στις εργασίες των στερημένων νημάτων, λειτουργεί ανεξάρτητα από τις άλλες ομάδες νημάτων μέσα στο socket της.

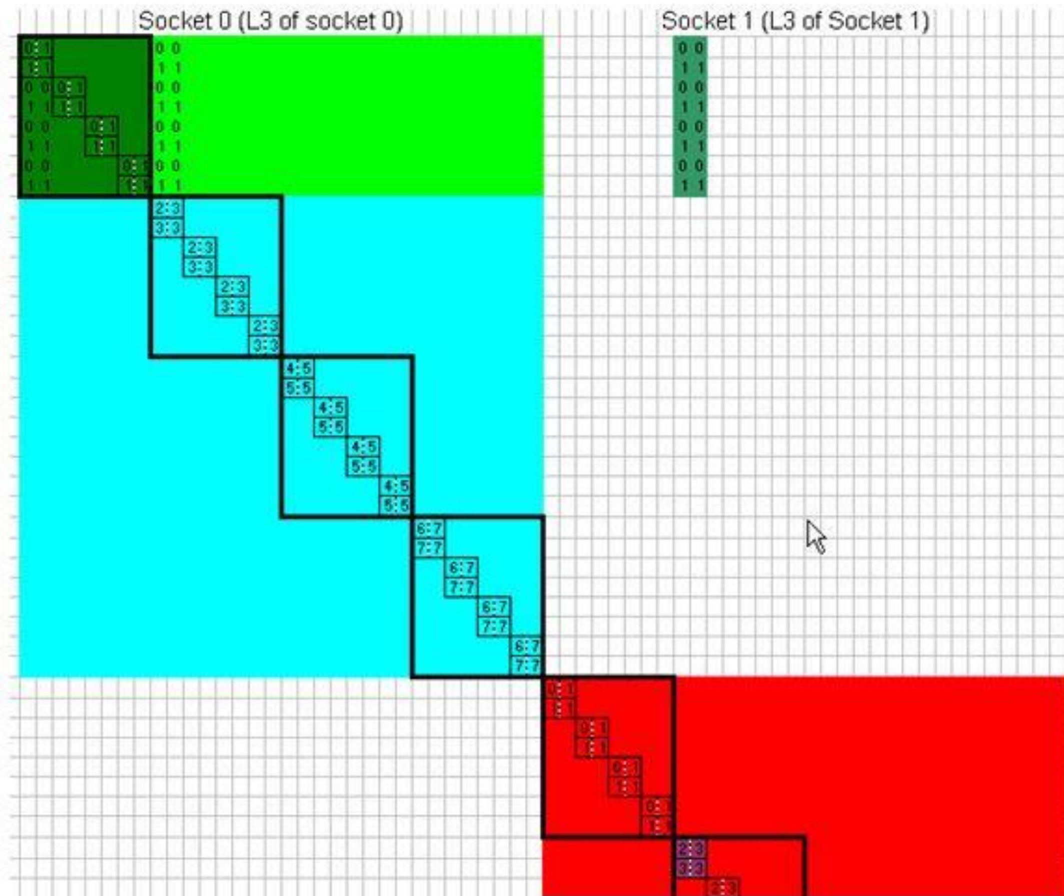
ΑΝΑΛΥΣΗ ΑΝΑΘΕΣΕΩΝ ΣΕ SOCKET (6)

- Αυτή η επαναληπτική διαδικασία συνεχίζεται έως ότου μια ομάδα νήματων ολοκληρώσει όλες τις εργασίες που ορίζονται στο κομμάτι socket:



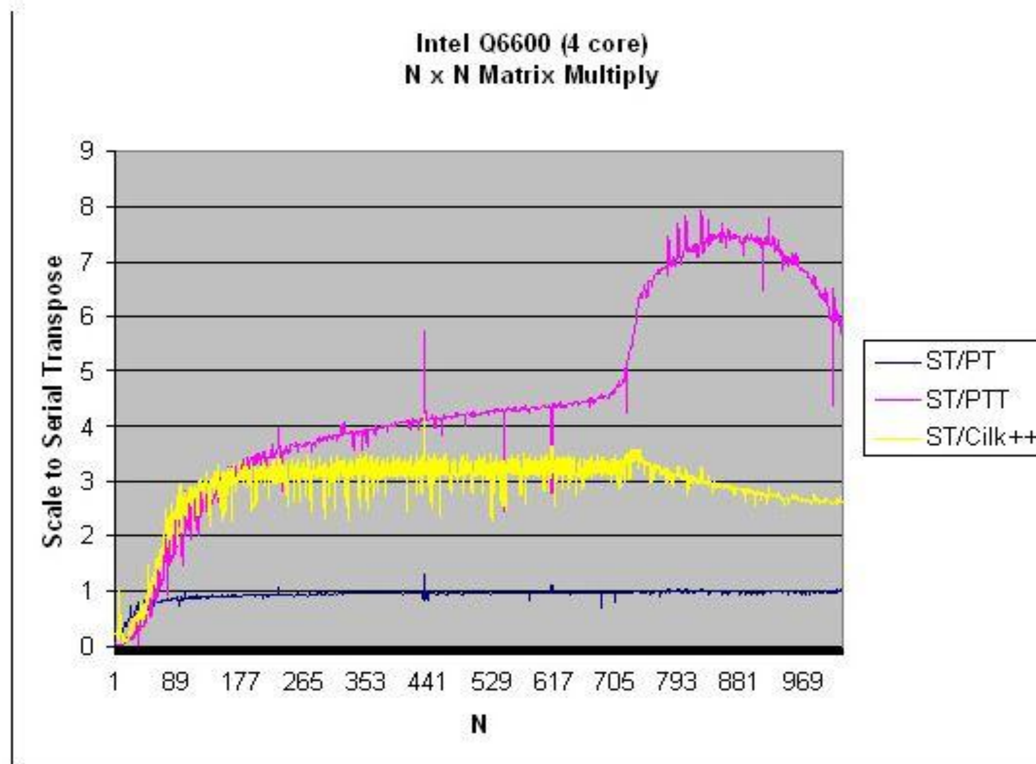
- Σε αυτό το σημείο , διαβουλεύεται τώρα η ολοκλήρωση της διαγωνίου των θυρίδων κατάστασης για τις διαγώνιες των άλλων sockets. Μιας και πρόκειται για μια μηχανή καταστάσεων αντί των ένθετων εσωτερικών επαναλήψεων, η ολοκλήρωση της διαγωνίου των άλλων socket μπορεί να προκύψει σε οποιαδήποτε σειρά, αλλά τείνει να προκύπτει σε διαγώνια 2x2 κομμάτια σε αύξουσα σειρά για κάθε διμελή ομάδα, σε κάθε πρόσθετο socket. Όπως προκύπτει από τη διαχωριστική πράσινη γραμμή , προς τα δεξιά του Socket 0 ομάδας 0/1 ζώνη στο socket 1 πάνω για την ολοκληρωμένη διαγώνιο για τη δεύτερη ομάδα στη κόκκινη ζώνη (socket 1) παρακάτω.

ΑΝΑΛΥΣΗ ΑΝΑΘΕΣΕΩΝ ΣΕ SOCKET (7)

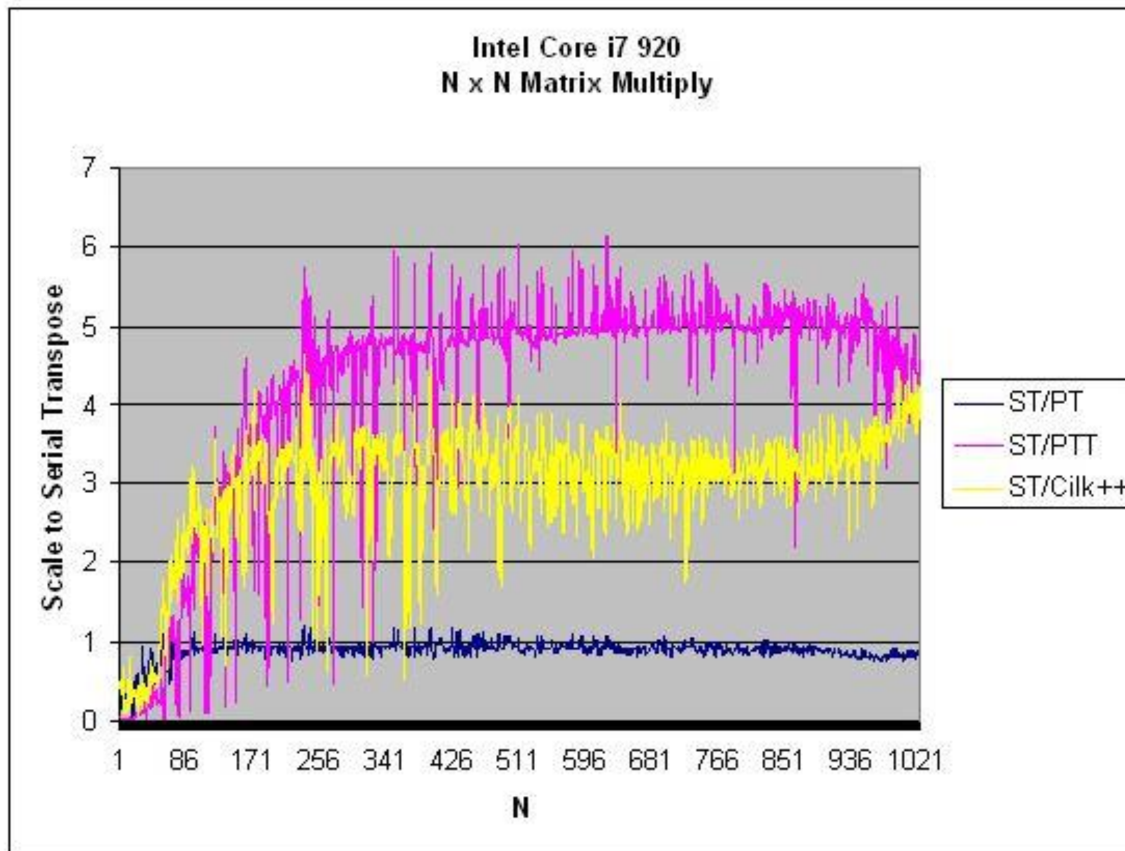


ΤΕΧΝΙΚΗ TAG TEAM ΜΕ ΜΕΤΑΦΟΡΑ INTEL Q6600

- Σε αυτό το σημείο θα γίνει η παραγωγή των διαγραμμάτων και ο έλεγχος των αποτελεσμάτων:



ΤΕΧΝΙΚΗ TAG TEAM ΜΕ ΜΕΤΑΦΟΡΑ INTEL core i7 920



ΑΝΑΛΥΣΗ ΤΕΧΝΙΚΗΣ TAG TEAM ΜΕ ΜΕΤΑΦΟΡΑ

Ο μέσος όρος των συντελεστών κλίμακας (σε σύγκριση με την σειριακη μέθοδο με μεταφορά πινάκων) για N = 128: 1024 είναι οι εξής:

- Q6600:

  4.96x για Παράλληλη A Core με μεταφορά πινάκων και 3.06x για Cilk ++

- Core i7 920:

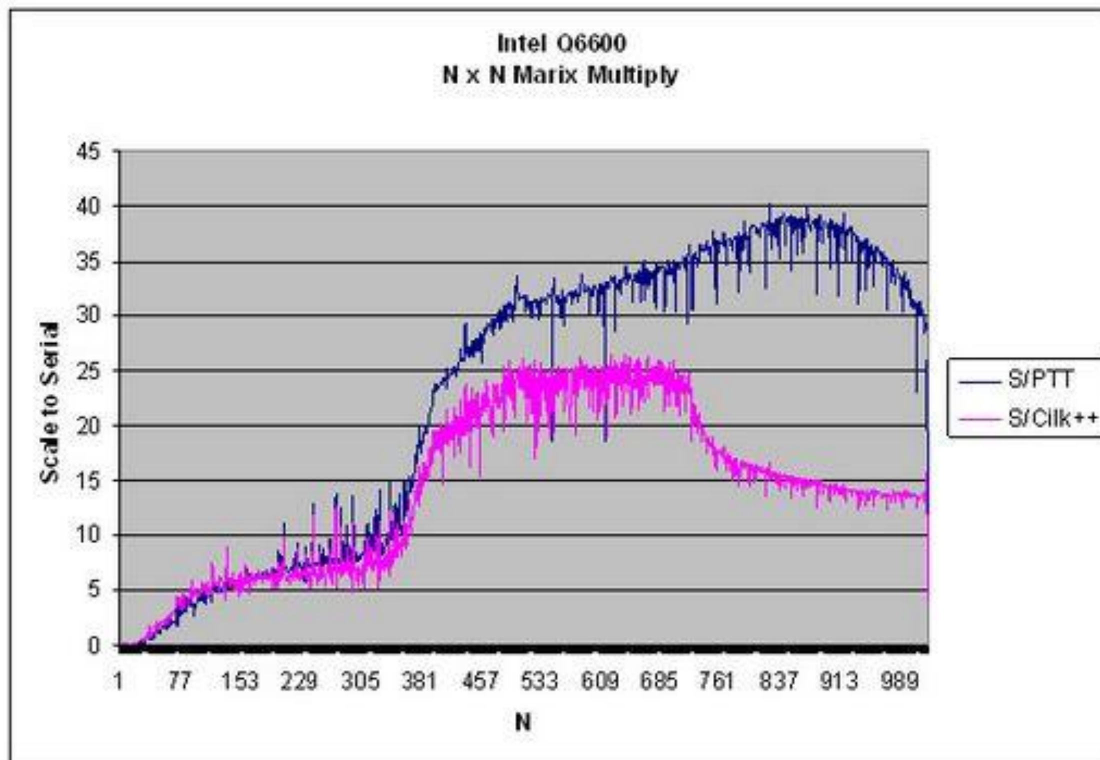
  4.69x για Παράλληλη Tag Team με μεταφορά πινάκων και 3.20X γιαCilk ++

- Οι μέγιστες τιμές για την επιλεγμένη ενότητα, βρίσκονται για N= 880 στο Q6600 και έχουμε αύξηση επίδοσης 7.5x για την παράλληλη μέθοδο Tag Team και 3x για την μέθοδος Cilk + + .

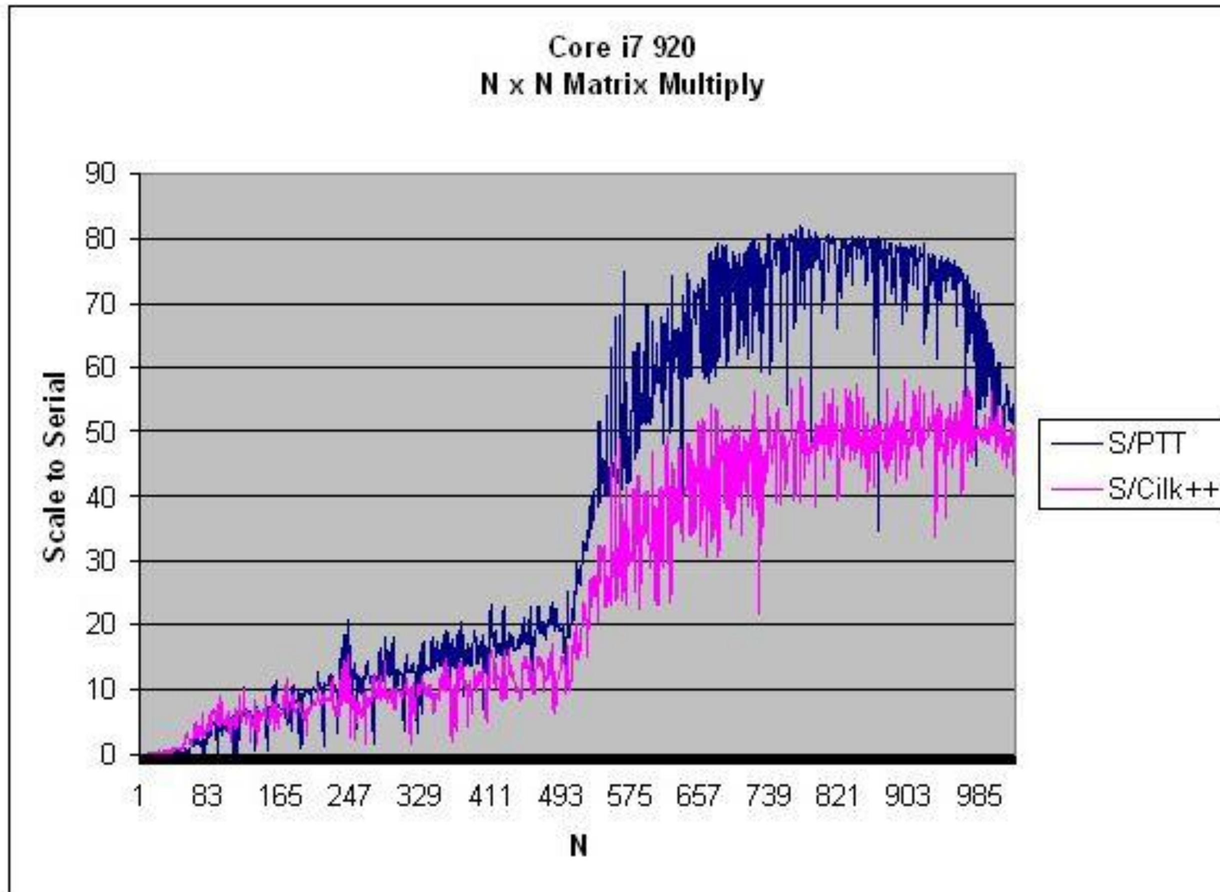
- ***Η παράλληλη Tag Team έχει σίγουρα το πλεονέκτημα έναντι της μεθόδου Cilk ++*** *(τουλάχιστον σε αυτό το εύρος N).*

ΤΕΧΝΙΚΗ TAG TEAM
INTEL Q6600

- Σύγκριση της τεχνικη tag team με την αρχική Serial μέθοδο, χωρίς μεταφορά πινάκων.



ΤΕΧΝΙΚΗ TAG TEAM
INTEL core i7 920



ΑΝΑΛΥΣΗ ΤΕΧΝΙΚΗΣ TAG TEAM

- Γίνεται αντιληπτό λοιπόν ότι σε επιλεγμένα σημεία στο διάγραμμα έχουμε φθάσει σε ένα 38x βελτίωση σε σχέση με Serial σε Q6600 και κοντά στην βελτίωση 80x σε κλιμάκωση πάνω από το Serial για Core i7 920.

- Στο εσωτερικό του MatrixMultiply, ο πιο εσωτερικός βρόχος εκτελεί ένα εσωτερικό γινόμενο. Όλες οι παραλλαγές του προγραμματισμού χρησιμοποιούν μια ενσωματωμένη λειτουργία για να εκτελέσουν το εσωτερικό γινόμενο. Η μέθοδος QuickThread Παράλληλης Tag team(PTT) χρησιμοποιεί μια πρόσθετη παραλλαγή σε αυτή τη λειτουργία εσωτερικού γινομένου που παράγει δύο εσωτερικά γινόμενα ταυτόχρονα. Κάθε νήμα της διμελούς ομάδας εργαζόταν σε ένα 2x2 κομμάτι , παράγοντας αποτελέσματα για το μισό από αυτό το κομμάτι (1x2). Εκτελώντας τα δυο εσωτερικά γινόμενα ταυτόχρονα με κάθε νήμα βελτιώνει την επιτυχία της cache σε μεγαλύτερα μεγέθη πινάκων.

- Παρακάτω είναι οι δυο συναρτήσεις εσωτερικού γινομένου , οι δυο συναρτήσεις τροποποιούνται για να χρησιμοποιούν xmm intrinsic και δυο επιλογείς συναρτήσεων που καλούν το κατάλληλο εσωτερικό γινόμενο στο πλαίσιο του δοκιμαστικού προγράμματος

ΣΥΝΑΡΤΗΣΕΙΣ ΕΣΩΤΕΡΙΚΟΥ ΓΙΝΟΜΕΝΟΥ

```
003 double DOT(double v1[], double v2[], intptr_t size)
004 {
005     double temp = 0.0;
006     for(intptr_t i = 0; i < size; i++)
007     {
008         temp += v1[i] * v2[i];
009     }
010     return temp;
011 }
012
```

```
042 void DOTDOT(double v1[], double v2[], double v3[], double r[2],  intptr_t size)
043 {
044     double  temp[2];
045     temp[0] = 0.0;
046     temp[1] = 0.0;
047     for(int i=0; i < size; ++i)
048     {
049         temp[0] += v1[i] * v2[i];
050         temp[1] += v1[i] * v3[i];
051     } // for(int i=0; i < size; ++i)
052     r[0] = temp[0];
053     r[1] = temp[1];
054 }
```

ΣΥΝΑΡΤΗΣΗ ΓΙΑ XMM INTRINSIC (1)

```
013 double xmmDOT(double v1[], double v2[], intptr_t size)
014 {
015     // __declspec(align(16)) not working reliably for me
016     double temp[4];
017     intptr_t alignedTemp = (((intptr_t)&temp[0]) & 8) >> 3;
018
019     __m128d _temp = _mm_set_pd(0.0, 0.0);
020     __m128d *_v1 = (__m128d *)v1;
021     __m128d *_v2 = (__m128d *)v2;
022
023     intptr_t    halfSize = size / 2;
024
025     for(intptr_t i = 0; i < halfSize; i++)
026     {
027         _temp = _mm_add_pd(_temp, _mm_mul_pd(_v1[i], _v2[i]));
028     }
029     // fix code to remove temp[4] array
030     _mm_store_pd( &temp[alignedTemp], _temp);
031     if(size & 1)
032         temp[alignedTemp] += v1[size-1] * v2[size-1];
033
034     return temp[alignedTemp] + temp[alignedTemp+1];
035 }
036
```

ΣΥΝΑΡΤΗΣΗ ΓΙΑ XMM INTRINSIC (2)

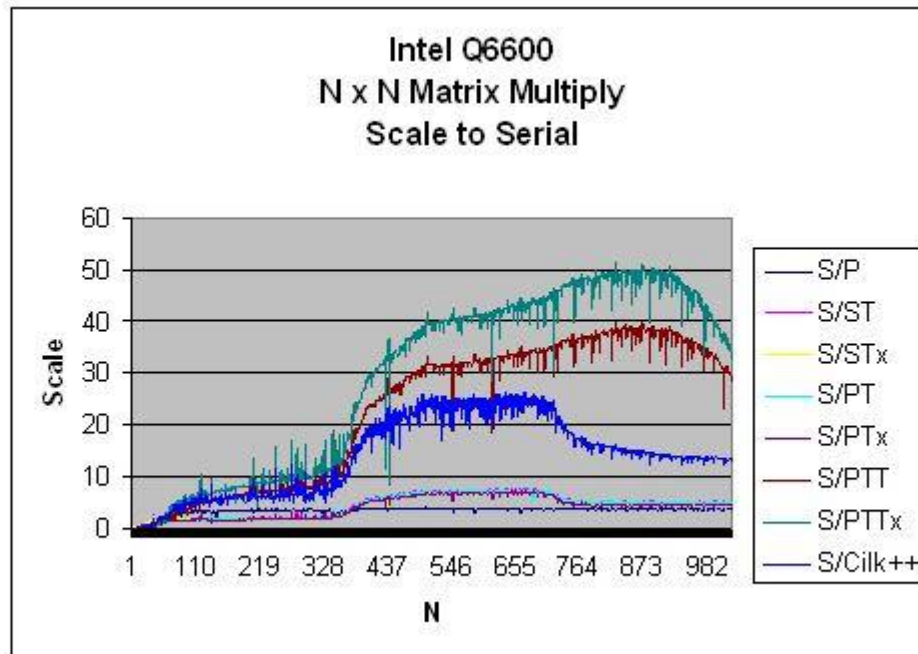
```
061 void xmmDOTDOT(double v1[], double v2[], double v3[], double r[2],  intptr_t size)
062 {
063     // __declspec(align(16)) not working reliably for me
064     double  temp[6];
065     intptr_t alignedTemp = (((intptr_t)&temp[0]) & 8) >> 3;
066     __m128d _temp0 = _mm_set_pd(0.0, 0.0);
067     __m128d _temp1 = _mm_set_pd(0.0, 0.0);
068     __m128d *_v1 = (__m128d *)v1;
069     __m128d *_v2 = (__m128d *)v2;
070     __m128d *_v3 = (__m128d *)v3;
071
072     intptr_t    halfSize = size / 2;
073
074     for(intptr_t i = 0; i < halfSize; i++)
075     {
076         _temp0 = _mm_add_pd(_temp0, _mm_mul_pd(_v1[i], _v2[i]));
077         _temp1 = _mm_add_pd(_temp1, _mm_mul_pd(_v1[i], _v3[i]));
078     }
079     _mm_store_pd( &temp[alignedTemp], _temp0);
080     _mm_store_pd( &temp[alignedTemp+2], _temp1);
081     if(size & 1)
082     {
083         temp[alignedTemp] += v1[size-1] * v2[size-1];
084         temp[alignedTemp+2] += v1[size-1] * v3[size-1];
085     }
086
087     r[0] =  temp[alignedTemp] + temp[alignedTemp+1];
088     r[1] =  temp[alignedTemp+2] + temp[alignedTemp+3];
089 }
```

ΣΥΝΑΡΤΗΣΕΙΣ ΓΙΑ ΚΛΗΣΗ ΕΣΩΤΕΡΙΚΟΥ ΓΙΝΟΜΕΝΟΥ

```
091 bool UseXMM = false;
092
093 double doDOT(double v1[], double v2[], intptr_t size)
094 {
095     if(UseXMM)
096         return xmmDOT(v1, v2, size);
097     return DOT(v1, v2, size);
098 }
099
100 void doDOTDOT(double v1[], double v2[], double v3[], double r[2],  intptr_t size)
101 {
102     if(UseXMM)
103         xmmDOTDOT(v1, v2, v3, r, size);
104     else
105         DOTDOT(v1, v2, v3, r, size);
106 }
```

XMM INTRINSIC INTEL Q6600

- Πρέπει να αντιμετωπιστεί ένα ζήτημα ***εναρμόνισης*** με τις οδηγίες του μεταγλωττιστή και πρέπει να το λυθεί με μια παρέμβαση στο κώδικα για να αποφευχθούν τα προβλήματα. Η ***ενσωμάτωση*** των συναρτήσεων xmm intrinsic σε αυτές τις δύο ρουτίνες είναι σχετικά εύκολη. Τα αποτελέσματα είναι τα εξής :



ΑΝΑΛΥΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ XMM INTRINSIC INTEL Q6600

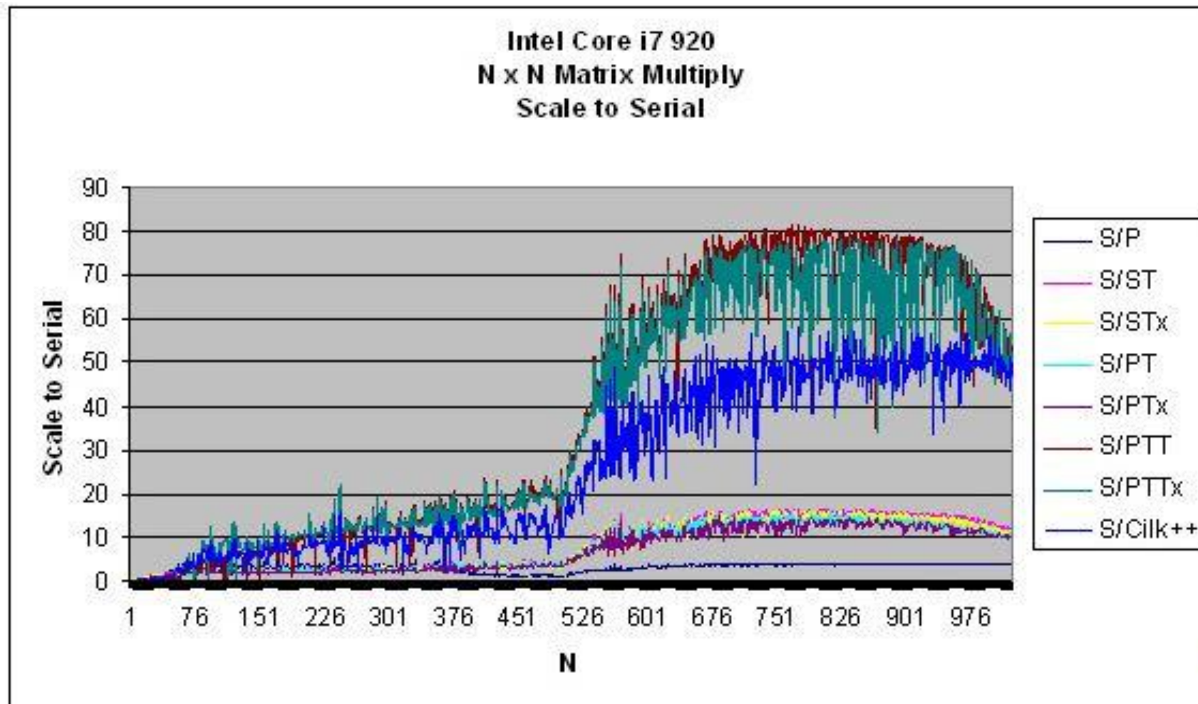
- Στο Q6600 (4 πυρήνες χωρίς HT) το S/PTTx οι συναρτήσεις xmm intrinsic ξεκάθαρα έχουν ένα ***πλεονέκτημα*** , περίπου 30 % βελτίωσης. Η μέθοδος QuickThread με τα tag team χρησιμοποιεί κατά μεγάλο ποσοστό τα xmmDOTDOT(διπλή συνάρτηση εσωτερικού γινομένου).

- Ωστόσο, η S/STX, χρησιμοποιώντας τη συνάρτηση xmmDOT (μονή συνάρτηση εσωτερικού γινομένου) εκτελεί ***ελαφρώς χειρότερα*** το κώδικα C++ χωρίς τις συναρτήσεις intrinsic.

- Αυτό δείχνει ότι οι βελτιστοποιήσεις του μεταγλωττιστή θα ήταν ***καλύτερες*** από τις χειρονακτικές βελτιστοποιήσεις ενός άπειρου προγραμματιστή intrinsic (καθόλου συνηθισμένο).

XMM INTRINSIC INTEL core i7 920

- Κοιτάζοντας το Core i7 920 θα εμφανιστεί μια διαφορετική εικόνα:



ΑΝΑΛΥΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ XMM INTRINSIC INTEL core i7 920

- Αυτός ο επεξεργαστής, με HT, εμφάνισε ***βάρος στην απόδοση*** (-12%) με τη χρήση του χειρόγραφων XMM βοηθητικών συναρτήσεων. Να σημειωθεί, το εκτελέσιμο ήταν το ίδιο και για τα δύο συστήματα.

- Για έναν πωλητή ενός προγράμματος για χρήση σε διάφορα συστήματα, αυτό είναι ένα σημαντικό κομμάτι πληροφοριών. Γνωρίζοντας τη συγκεκριμένη συμπεριφορά της πλατφόρμας σημαίνει ότι μπορεί να ελεγχθεί το σύστημα κατά την εκκίνηση του προγράμματος και στη συνέχεια να αλλαχτεί η επιλογή από το κώδικα που θα χρησιμοποιηθεί. Συνήθως θα περιλαμβάνει την επιλογή μιας συνάρτησης (***pointer λειτουργία***) στον κώδικα, σε αντίθεση με τη χρήση μιας συνάρτησης επιλογής.

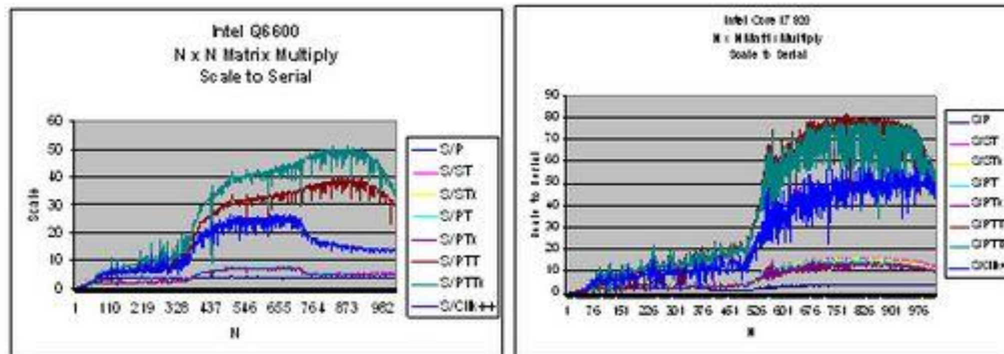
- Μια εναλλακτική προσέγγιση για την άγνωστη συμπεριφορά είναι να χρησιμοποιηθεί μια ***προσέγγιση heuristics***. Η εφαρμογή θα επιλεχτεί για κάθε παραλλαγή του κώδικα για τις πρώτες κλήσεις και τη ***μέτρηση*** του χρόνου εκτέλεσης για κάθε μέθοδο, στη συνέχεια, με τη ***λήψη*** αρκετών δειγμάτων, θα επιλεχτούν οι καλύτερες επιδόσεις στη μεταβολή των λειτουργιών και θα τοποθετηθεί η κατάλληλη συνάρτηση στο δείκτη αποστολής.

- Μπορεί αυτό να βελτιωθεί ? ***Ναι!!***

- Ας σημειωθεί ότι οι γραμμές του γραφήματος είναι μάλλον «***θορυβώδης***». Επιπλέον ρύθμιση μπορεί να βελτιώσει την αρμονία και, συνεπώς, να κινούνται πάνω στη γραμμή τάσης. Το ποσό αυτό ανέρχεται σε περίπου επιπλέον ***15% πάνω*** από την τρέχουσα παράλληλη μέθοδος Tag Team μεταφοράς. (xmmDOTDOT σε μη-HT συστήματα, DOTDOT για HT συστήματα) Με 15% συνήθως δεν αξίζει να πηγαίνει μετά, ωστόσο, να επισημανθεί ότι τόσο η παράλληλη μέθοδος Tag Team και η Cilk ++ μέθοδος φαίνεται να ***πέφτουν στα 1024***. Πρόσθετες δοκιμές θα πρέπει να εκτελούνται με μεγαλύτερους πίνακες, και το πιο σημαντικό για ***multi-socket*** συστήματα.

ΣΥΝΟΨΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ (1)

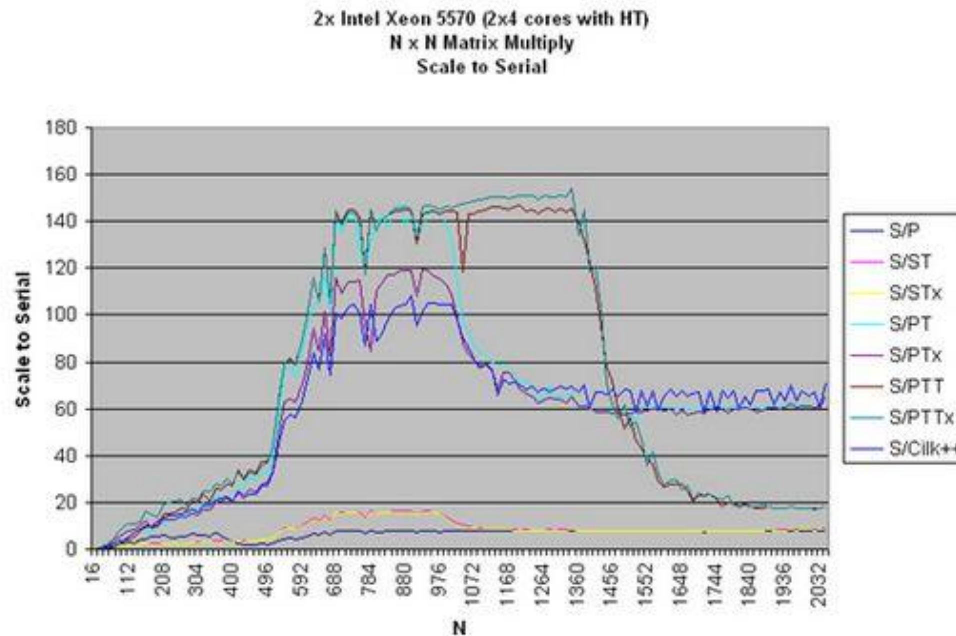
- Προηγουμένως εμφανίστηκαν τα αποτελέσματα από την QuickThread Parallel Tag Team μέθοδο στον πολλαπλασιασμό πινάκων που πραγματοποιήθηκε σε δύο συστήματα μονού επεξεργαστή:



- Όπως φαίνεται ο επεξεργαστής Intel Q6600 (4 πυρήνες - χωρίς HT) με δύο πυρήνες (δύο νήματα) που μοιράζονται τις L1 και L2 caches πετυχαίνει από 40x ως 50x βελτίωση σε σχέση με σειριακό τρόπο, και ο επεξεργαστής Intel Core i7 920 (4 πυρήνες - με HT) με τέσσερις πυρήνες (οκτώ νήματα) μοιράζεται μια L3 cache και ένα πυρήνα (δύο νήματα) που μοιράζεται τις L1 και L2 caches πετυχαίνει από 70x ως 80x βελτίωση. Ας εξεταστεί πώς αυτό εκτελείται χρησιμοποιώντας δύο επεξεργαστές, ο καθένας παρόμοιος με τον Core i7 920.

ΣΥΝΟΨΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ (2)

- Όταν τρέχει στο Dual Xeon 5570 με 2 υποδοχές και δύο L3 caches, οπου η κάθε μια μοιράζεται σε τέσσερις πυρήνες (8 threads) και κάθε επεξεργαστής με τέσσερις L2 και τέσσερις L1 caches οπου η κάθε μια μοιράζεται από έναν πυρήνα και 2 νήματα, εμφανίζετε το διάγραμμα παρακάτω.



ΣΥΝΟΨΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ (3)

- Για μια κλίμακα με το σειριακό από 140x έως 150x για φάσμα N = 700 έως 1344 η απόδοση είναι σχεδόν διπλάσια από αυτή των Core i7 920. Αυτό ήταν κάπως αναμενόμενο.

- Υπάρχουν κάποιες ενδιαφέρουσες παρατηρήσεις που πρέπει να γίνουν πάνω σε αυτό το προφίλ απόδοσης.

- Ενώ η αύξηση ταχύτητας 2x ήταν αναμενόμενη, η παράλληλη μέθοδος που χρησιμοποιει μεταφορά πίνακα καθώς και η παράλληλη μέθοδος Tag Team εκτελείται με N = 700 - 1024, και στη συνέχεια πέφτει απότομα. Αυτό είναι περίπου το μισό της μέγιστης απόδοσης της παράλληλης μεθόδου Tag Team (700 έως 1.344).

ΣΥΝΟΨΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ (4)

- Γιατί βρίσκονται στο ίδιο ύψος οι γραμμές του διαγράμματος που είναι σταθερές?
- Ποια είναι η ερμηνεία για την διαφορά της πτώσης?

- Οι σταθερές γραμμές του διαγράμματος βρίσκονται στο ίδιο ύψος για τον ίδιο λόγο των διαγραμμάτων των διαφανειών 75,76 οπου οι επιδόσεις της σειριακής και της παράλληλης μεθόδου που χρησιμοποιούσαν μεταφορά πίνακα ήταν ουσιαστικά ίδιες(κίτρινες και κόκκινες γραμμές στο προηγούμενο σχήμα). Ο λόγος είναι ο περιορισμός του εύρους ζώνης (bandwith) των πόρων.

- Στη διαφάνεια 75 και 76 ο περιορισμός των πόρων φαίνεται να είναι το εύρος ζώνης της μνήμης (λόγω του ότι η παράλληλη μέθοδος Tag Team έχει μεγάλο περιθώριο για την εκτέλεση της από την παράλληλη μέθοδο που χρησιμοποιεί μεταφορά πίνακα). Λογω του ότι οι σταθερές γραμμές του διαγράμματος βρίσκονται στο ίδιο ύψος (από N = 700 έως 1024) φαίνεται ότι και κάποιος άλλος πόρος λειτουργεί ως περιοριστικός παράγοντας εκτός του εύρους ζώνης την μνήμης. Αυτό επιφέρει επιβάρυνση στην πρόσβαση της cache ή SSE Floating σημείο συμφόρησης.

ΣΥΝΟΨΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ (5)

- Και τα δύο αυτά σημεία συμφόρησης θα τείνουν να μειώσουν το ύψος της καμπύλης απόδοσης, αλλά όχι το πλάτος. Παρατηρείτε στο διάγραμμα παραπάνω ότι οι δύο παράλληλες μεθόδους Tag Team κατάφεραν να διπλασιάσουν το εύρος της κορυφής της καμπύλης απόδοσης επιτρέποντας το πρόγραμμα να χειριστεί αποτελεσματικά μεγαλύτερους πίνακες .

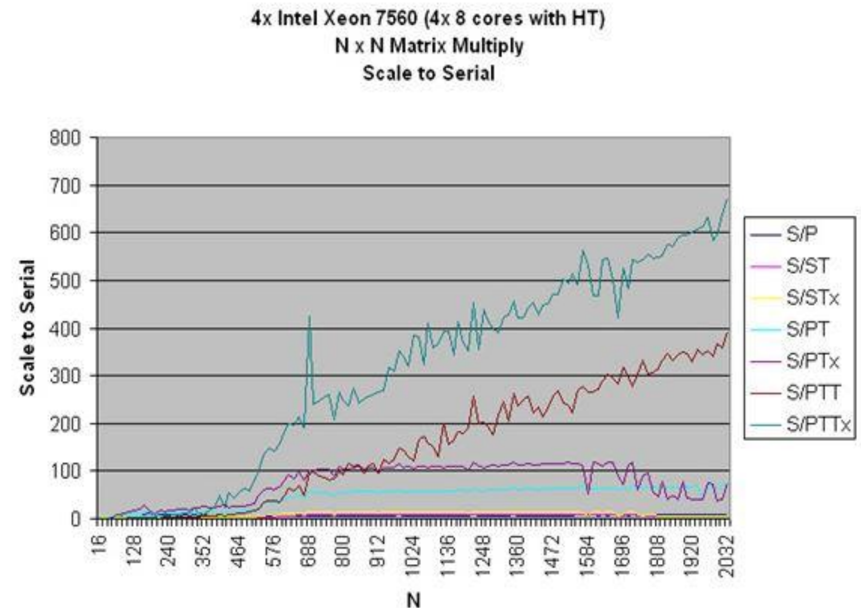
- Ο λόγος για την αύξηση του πλάτους (χειρισμός μεγαλύτερων πινάκων) οφείλεται κυρίως στην πιο αποτελεσματική επαναχρησιμοποίηση των προσωρινά αποθηκευμένων δεδομένων λόγω της διαδρομής της λύσης μέσω του προβλήματος (αλληλουχία με την οποία γίνονται υπολογισμοί).

- Οι γνώσεις που αντλήθηκαν από την διαφάνεια 116 είναι οι εξής: Όταν το πρόβλημά είναι ότι το σύνολο των δεδομένων εργασίας υπερβαίνουν εκείνα του συστήματος προσωρινής αποθήκευσης, μπορούν να βρεθούν μερικές διαδρομές για τη λύση που είναι πιο αποτελεσματικές από ό, τι ένας απλός ένθετος βρόχος.

Cilk++ ΣΕ 4x INTEL Xeon 7560

- Εκτελώντας την μέθοδο sans Cilk++ σε ένα 4 επεξεργαστή Intel Xeon 7560, κάθε επεξεργαστής με 8 πυρήνες συν Υπερ-νήματα (συνολικά 32 πυρήνες, 64 threads) εμφανίζετε το διαγραμμα:

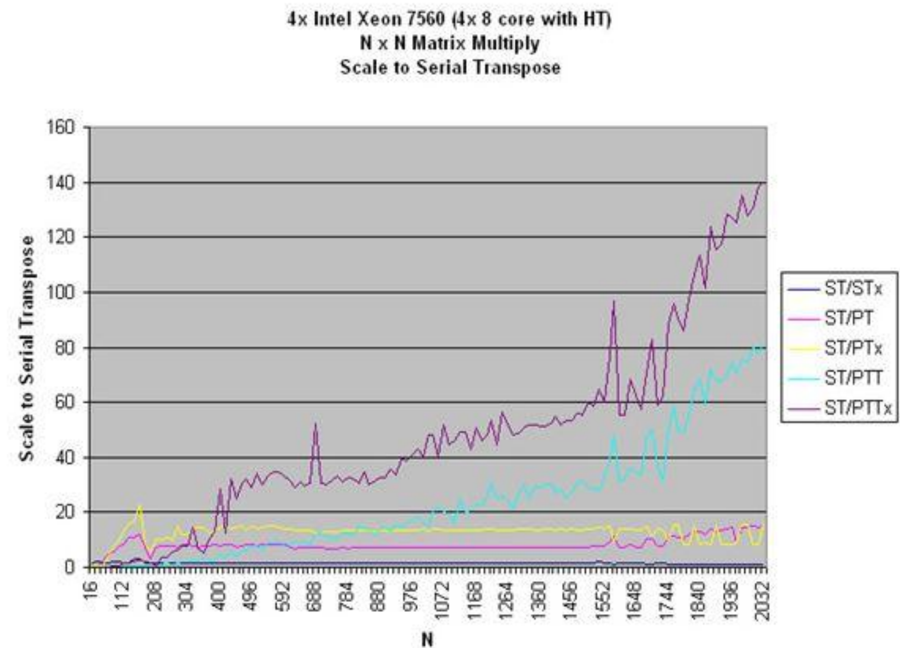
- Σε αυτό το διάγραμμα δεν βλέπουμε σταθερό επίπεδο στη κλιμάκωση. Αυτό οφείλεται στο μέγεθος του προβλήματος στο N = 2048 που περιέχεται πλήρως στην μνήμη cache του συστήματος. Προσοχή, να έχετε κατά νου ότι το παραπάνω διάγραμμα αντιπροσωπεύει την κλιμάκωση της cache στην ευαίσθητη σειριακή μέθοδο.



Cilk++ ΜΕ ΜΕΤΑΦΟΡΑ ΠΙΝΑΚΩΝ ΣΕ 4x INTEL Xeon 7560

- Στη σύγκριση αυτή με την ευαίσθητη σειριακή μέθοδο μεταφοράς εμφανίζετε ένα σύνολο διαφορετικών αποτελεσμάτων:

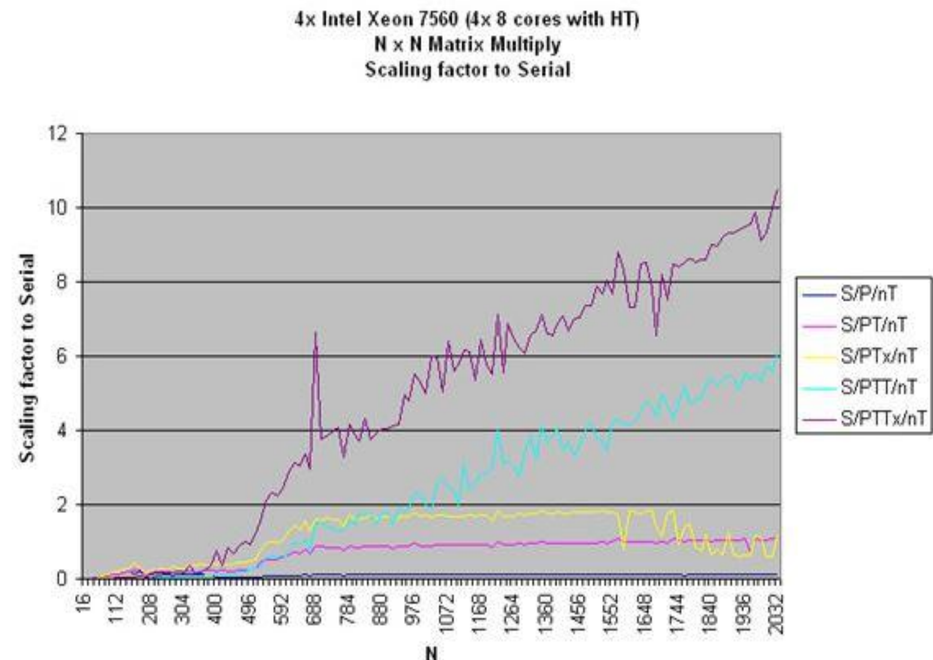
- Η απότομη αλλαγή στην κλίση σε N = 1500-1700 οφείλεται κυρίως στην πτώση της απόδοσης των στοιχείων αναφοράς του σειριακής μεθόδου μεταφοράς, και όχι λόγω της βελτίωσης στην PTTx.



ΣΥΝΤΕΛΕΣΤΗΣ ΚΛΙΜΑΚΩΣΗΣ ΣΕ 4x INTEL Xeon 7560

- Κοιτάζοντας τους συντελεστές κλιμάκωσης (παράλληλη απόδοση / αριθμός νημάτων hardware) που χρησιμοποιείται συχνά ως παράγοντας απόφαση για να κάνει μια αγορά. Παρακάτω εμφανίζονται τα διαγράμματα του συντελεστή κλιμάκωσης:

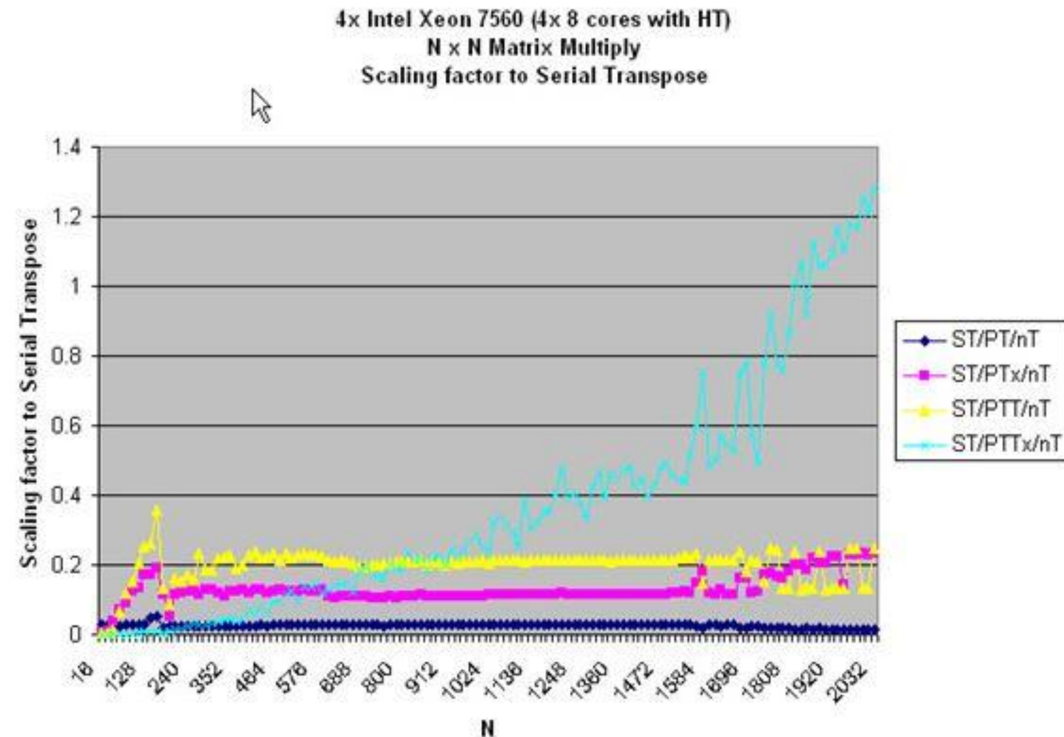
- Καθώς το μέγεθος του προβλήματος αυξάνεται παρατηρείτε μια ωραία θετική κλίση στον παράγοντα κλιμάκωσης. Αυτό φαίνεται εξαιρετικά καλό. Είναι σημαντικό ότι η σειριακή μέθοδος δεν είναι ευαίσθητη στη μνήμη cache και δεν είναι ένα έγκυρο σημείο αναφοράς για σύγκριση.



ΣΥΝΤΕΛΕΣΤΗΣ ΚΛΙΜΑΚΩΣΗΣ ΚΑΙ ΣΕΙΡΙΑΚΗ ΜΕΘΟΔΟΣ

- Έπειτα δημιουργώντας το παράγοντα κλιμάκωσης σε σύγκριση με τη σειριακή μέθοδο μεταφοράς φιλική προς τη cache παίρνουμε μια εντελώς διαφορετική εικόνα:

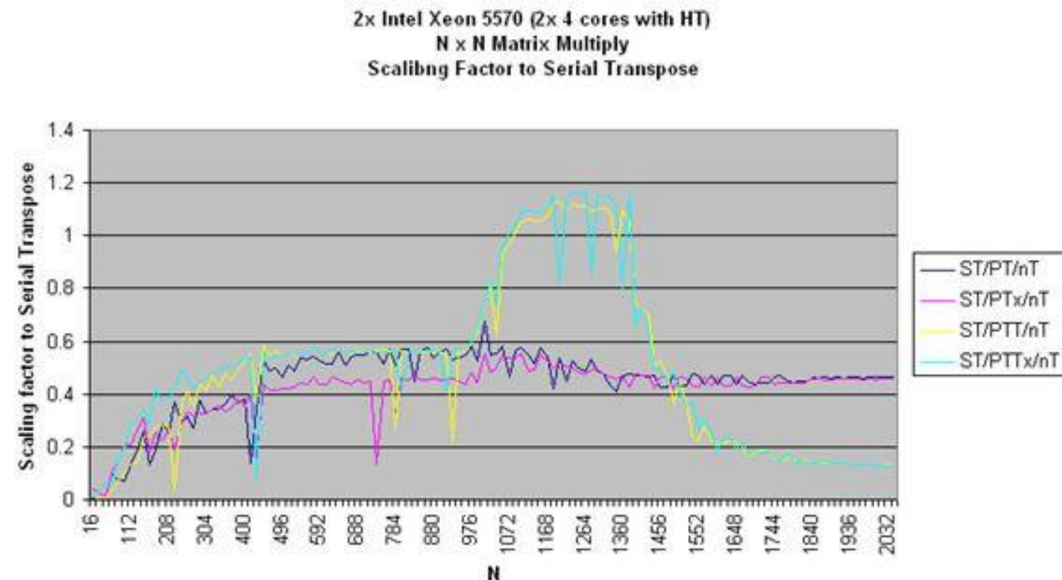
- Ο συντελεστής κλιμάκωσης σε μια σειριακή μέθοδο μεταφοράς ευαίσθητη στη μνήμη cache δεν αποδίδει (ο συντελεστής ξεπερνά το 1) μέχρι το N=1824.



ΣΥΝΤΕΛΕΣΤΗΣ ΚΛΙΜΑΚΩΣΗΣ ΣΕ 2x INTEL Xeon 5570

- Συγκρίνοντας τα στοιχεία του 2x Intel Xeon 5570, συνυπολογίζοντας με τη σειριακή Μεταφορά το διάγραμμα είναι το εξής:

- Όπως ήταν αναμενόμενο, τα δύο συστήματα μπορούν να επιτύχουν υπερβαθμωτά αποτελέσματα σε διαφορετικά μεγέθη ενός προβλήματος. Αυτό οφείλεται στο διαφορετικό ποσό της κρυφής μνήμης για κάθε σύστημα.



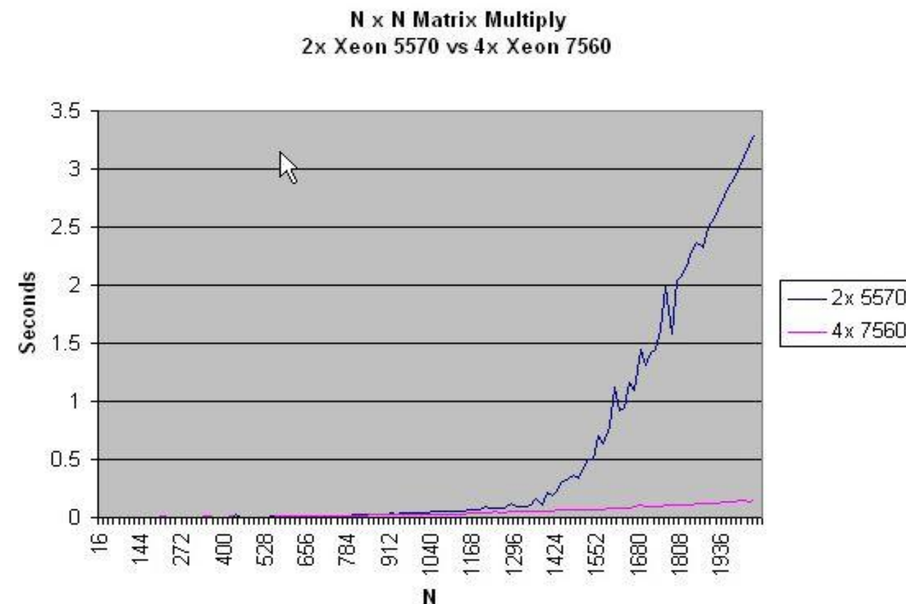
ΑΝΑΛΥΣΗ ΣΥΝΤΕΛΕΣΤΗ ΚΛΙΜΑΚΩΣΗ

- Αν και ο συντελεστής κλιμάκωσης αποτελεί μια καλή προοπτική σε σχέση με την απόδοση των επενδύσεων στην αγορά περισσότερων επεξεργαστών, δεν έχει νόημα όταν συγκρίνεται με μια διαφορετική αρχιτεκτονική του επεξεργαστή. Μια εταιρεία θα πρέπει να ενδιαφέρονται για συνολική απόδοση των επενδύσεων, και αυτό περιλαμβάνει και το στοιχείο του χρόνου.

ΧΡΟΝΟΣ ΣΤΟ 2x Xeon 5570 ΚΑΙ 4x Xeon 7560

- Κατά την εξέταση του στοιχείου του χρόνου, υπάρχει μια εντελώς διαφορετική εικόνα. Όταν γίνετε η σύγκριση με την ταχύτερη μέθοδο (QuickThread Parallel Tag Team Transpose with SSE intrinsic functions) το διάγραμμα είναι το παρακάτω.

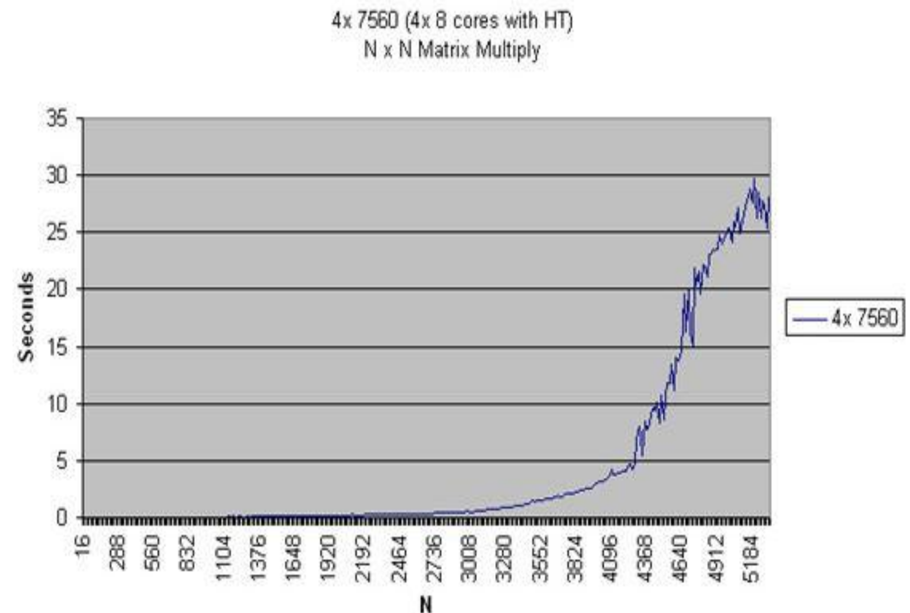
- Περιλαμβάνοντας το χρόνο, ως προσδιορισμό για κόστος-όφελος, διαπιστώνετε ότι υπάρχει μια μάλλον δραματική μετάβαση στην αναλογία κόστους-οφέλους μετά το όριο στο μέγεθος του προβλήματος (N = 1400). Το θέμα που γίνεται εδώ είναι η χρήση του κατάλληλου μεγέθους στις περιπτώσεις δοκιμών, όταν γίνονται αξιολογικές κρίσεις για τις αποφάσεις αγοράς. Οι καμπύλες κόστους / οφέλους και των επιδόσεων δεν θα είναι πάντα κατάλληλες για προέκταση.



ΧΡΟΝΟΣ ΣΤΟ 4x Xeon 7560 ΜΕ ΜΕΓΑΛΥΤΕΡΟΥΣ ΠΙΝΑΚΕΣ

- Όταν εκτελείτε η ταχύτερη μέθοδος (QuickThread Parallel Tag Team Transpose with SSE intrinsic functions) σε μεγαλύτερα μεγέθη πίνακα τα συμπεράσματα είναι τα εξής:

- **Πίνακες** έως N = 3.000 - 4.096 μπορούν να αντιμετωπιστούν με 4 επεξεργαστές (32 πυρήνες / 64 νήματα), μεγαλύτεροι πίνακες μπορεί να απαιτήσουν επιπλέον επεξεργαστές ή / και μια αναθεωρημένη μέθοδο.



ΓΕΝΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ (1)

- Η πιο γρήγορη μέθοδος: Parallel Tag Team Transpose SSE συναρτήσεις intrinsic, βασίζεται στην ικανότητα των QuickThread να προγραμματίζουν με συνάφεια τα επιλεγμένα νήματα από την εγγύτητα του επιπέδου cache. Η ικανότητα να συντονίζει τις εργασίες με τη χρήση της ευαισθησίας της cache μπορεί να κοστίσει πολύ στις στρατηγικές βελτιστοποίησης.

- Μεγαλύτερα μεγέθη πινάκων θα μπορούσαν να αντιμετωπιστούν με έναν βελτιωμένο τρόπο με τον ίδιο αριθμό των επεξεργαστών (4χ 8 πυρήνα με HT), όταν συνδυάζονται με μία πρόσθετη στρατηγική κομματιών η οποία θα περιλαμβάνει πρόσθετη επιβάρυνση. Αυτό συνήθως λέγεται η μέθοδος «του διαίρει και συμφώνησε» , που χρησιμοποιείται συχνά από τους προγραμματιστές παράλληλων.

- Λαμβάνοντας το πίνακα με N=5200 και χωρίζοντάς τον σε 2(2 άξονες) δόσεις κομματιού N=2600 και τέσσερα τέτοια πλακάκια . Αυτό απαιτεί 4x2=8 επαναλήψεις χρησιμοποιώντας τα μικρότερα κομμάτια. Ο πίνακας στο N=2600 παίρνει περίπου 0.33 δευτερόλεπτα για να υπολογιστεί , ως εκ τούτου εκτιμάται χρόνος υπολογισμού θα είναι στο 0,33 x 8 = 2.64. Εκτιμάται μια βελτίωση 10x σε σχέση με τη μέθοδο του μη-κατακερματισμού, αλλά μπορεί να μην είναι αρκετά γρήγορη για τους σκοπούς σας.

- Θα ήταν η μέθοδος « διαιρεί και βασίλευε» η καλύτερη στρατηγική για να χρησιμοποιηθεί;

Αυτό εξαρτάται από την ερμηνεία των βέλτιστων.

- Από την άποψη της σχετικής απόδοσης των επιδόσεων για την προσπάθεια στον προγραμματισμό, αυτό μπορεί να είναι έτσι. Ωστόσο, εξετάζοντας εικόνα 18 (17 από την part-4), και συγκρίνοντας την Cilk + + σε μέθοδο QuickThread Parallel Tag Team XMM, έχει αποδειχτεί ότι δίνοντας ιδιαίτερη προσοχή στη περιοχή της cache , συγκεκριμένα, τι είναι στη cache L1,L2 και L3, και όταν είναι σε αυτές τις μνήμες cache , θα μπορούσαν να επιταχύνουν ένα επιπλέον 1,4 x 2,5 x για να ωθήσει την απόδοση στις επιδόσεις.

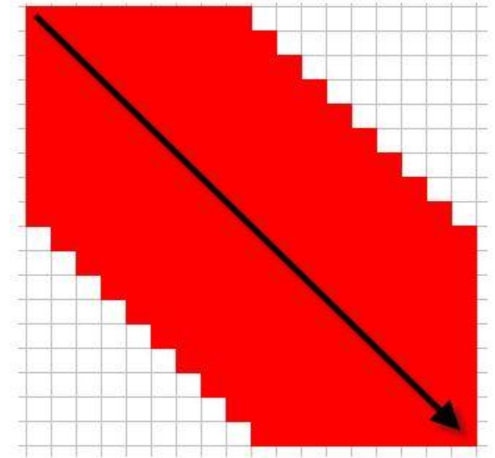
- Θα τεθεί η στρατηγική που θα κάνει αποτελεσματική χρήση των μνήμων cache του συστήματος. Ενώ το παρακάτω σχεδιάγραμμα δεν θα δείξει τη συγκεκριμένη μέθοδο, θα αποδείξει το γενικό σχέδιο της επίθεσης.

- Η τρέχουσα μέθοδος της Παράλληλης Tag Team (μεταφορά) διαιρεί το έργο από την L3, στη συνέχεια τις περιοχές της L2 και μετά παίρνει από την L1 ένα δρόμο φιλικό για την εξαγωγή των αποτελεσμάτων. Η στρατηγική αυτή λειτουργεί εξαιρετικά καλά μέχρι το μέγεθος του πίνακα να φθάσει σε ένα σημείο όπου η διαδρομή εκτέλεσης αρχίζει να διώχνει τα δεδομένα από τη μνήμη cache L3.

ΓΕΝΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ (4)

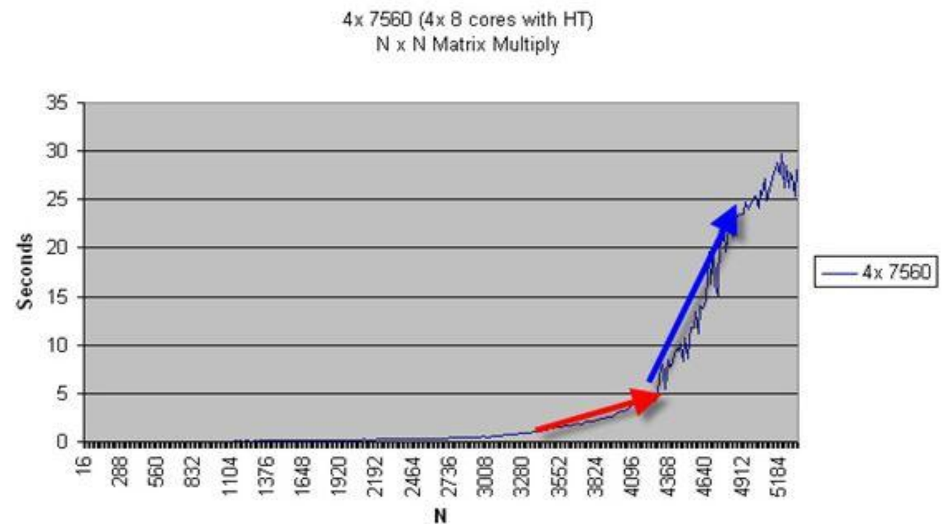
- Στο παραπάνω διάγραμμα, η γενική διαδρομή εκτέλεσης είναι αυτή που ακολουθεί το βέλος. Τα χρωματιστά (κόκκινα) κελιά δείχνουν τα κελιά εξόδου που τα αποτελέσματα τους έχουν υπολογιστεί. Τα λευκά κελιά υποδεικνύουν εκείνα τα κελιά εξόδου που πρέπει ακόμη να υπολογιστούν.

- Στη τρέχουσα μέθοδο του Παράλληλου Tag Team (μεταφορά) όλα τα παραπάνω κελιά θα ήταν χρωματισμένα, στη προτεινόμενη μέθοδο για μεγάλους πίνακες, μια τεχνική αποκοπής που περιορίζει την απόσταση από τη διαγώνιο των κελιών παραγωγής που πρέπει να υπολογιστούν κατά την επεξεργασία της διαγωνίου. Τα παραπάνω είναι μια απλοποίηση του συστήματος 1P.



ΓΕΝΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ (5)

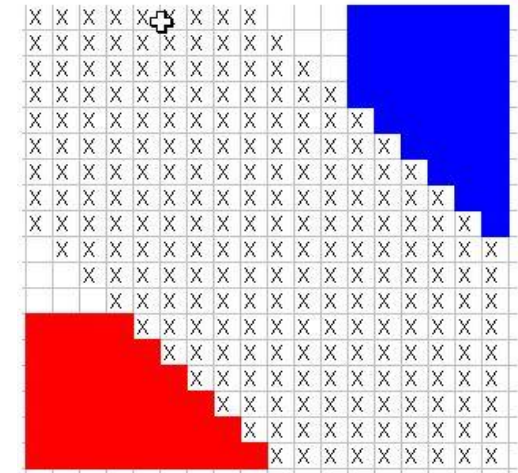
- Στην επεξεργασία ενός μεγάλου πίνακα, με τους υπολογισμούς να βρίσκονται στα κενά κελιά, ο υπολογισμός που θα υποστεί τη πρώτη έξωση από την L2 στην L3,σε κάποιο μέγεθος, θα αφαιρεθεί από την L3



- Στο παραπάνω σχήμα 27 (Σχ. 25 με πρόσθετα βέλη), το κόκκινο βέλος απεικονίζει τις L2 εξώσεις και το μπλε βέλος απεικονίζει τις L3 εξώσεις.

ΓΕΝΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ (6)

- Μετά την ολοκλήρωση της παραγωγής κελιών στο Σχ. της διαφάνειας 131 θα βρεθεί το παρακάτω σχήμα.

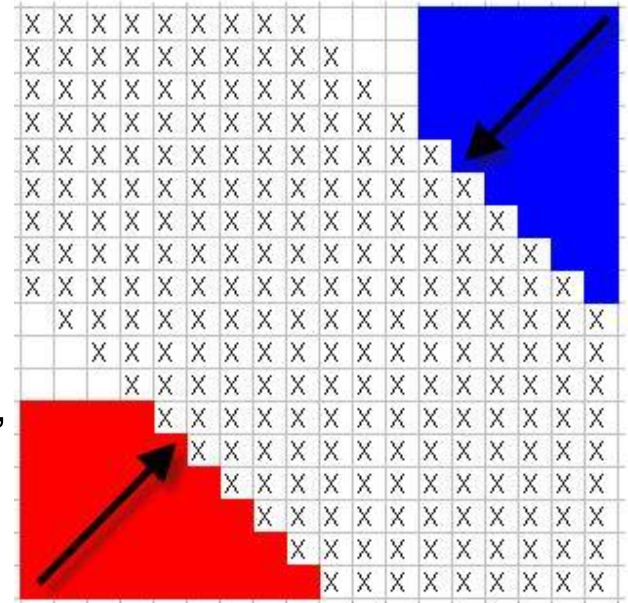


- Όπου είναι το σημάδι X στα κελιά είναι οι έξοδοι της L2, όπου τα αποτελέσματα έχουν ολοκληρωθεί. Τα μπλε κελιά αντιπροσωπεύουν στήλες (αποθηκεύονται ως γραμμές) στο m2t πίνακα που εξακολουθούν να διαμένουν στη μνήμη cache L2, και τα κόκκινα κελιά αντιπροσωπεύουν κελιά σειρά του πίνακα m1 που βρίσκονται στη μνήμη L2 cache. Τα μπλε κελιά αντιπροσωπεύουν στήλες (αποθηκεύονται ως γραμμές) στο m2t πίνακα που εξακολουθούν να διαμένουν στη μνήμη cache L2, και τα κόκκινα κελιά αντιπροσωπεύουν κελιά σειρών του πίνακα m1 που βρίσκονται στη μνήμη L2 cache. Επιπλέον, (δεν απεικονίζονται από χρωματισμό) κάποιο τμήμα των κάτω σειρών (ες) και των πιο δεξιά στηλών (ες) εξακολουθούν να κατοικούν στην κρυφή μνήμη L1. Τα υπόλοιπα λευκά που δεν είναι X μπορούν, ή δεν μπορούν, να βρίσκονται στην L3 cache.

ΓΕΝΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ (7)

- Η επόμενη ακολουθία υπολογισμού (που υπόκεινται σε επαλήθευση) θα έπρεπε να ακολουθήσει την ακολουθία όπως απεικονίζεται από τα βέλη στο παρακάτω σχήμα:

- Τα κόκκινα και μπλε άκρα του πίνακα εξόδου θα πρέπει να υποβάλλονται σε επεξεργασία σε εναλλασσόμενη σειρά καθώς προχωρείτε κατά μήκος των βελών προς την πρώτη διαγώνιο



- Στην προαναφερθείσα μέθοδο «διαίρει και βασίλευε» (κομμάτια), θα επεξεργάζεστε 4 μικρότερα κομμάτια δύο φορές ή 8x το χρόνο των μικρότερων κομματιών, προφανώς για ένα μέγεθος προς καλύτερη δυνατή χρήση για το μέγεθος μνήμης L2 cache.

- Ο κατακερματισμός θα μπορούσε να επωφεληθεί από τα υπολειμματικά δεδομένα της L2 που προκύπτουν σε ένα 6x ως 8x σε χρόνο εκτέλεσης του μικρότερου πίνακα σε αντίθεση με 8x του χρόνου εκτέλεσης.

- Στην προτεινόμενη μέθοδο (την αποκαλούν cross diagonal), καθώς και για το εύρος μεγέθους που απεικονίζεται παραπάνω στα σχήματα 28 και 29, εκτιμάται ότι μπορεί να είναι δυνατό να παραχθεί το αποτέλεσμα σε 1.5x έως 2x την ώρα του μικρότερου πίνακα. Δυνητικά επικράτησε η μέθοδος του διαίρει και βασίλευε με συντελεστή 4x. Θα πρέπει να τονιστεί ότι οι πραγματικές διαφορές μπορούν να διαφέρουν από αυτή την εκτίμηση. Η παρέκταση, όπως προαναφέρθηκε, συχνά δεν ακολουθεί την καμπύλη που ισχύει με τα σημερινά δεδομένα

SUPERSCALAR PROGRAMMING-ΕΦΑΡΜΟΓΗ 1

```
openuser@snf-17063:~/telikh/QuickThreadLinux2.0.0/MatrixMultiply/Debug$ ./MatrixMultiply 750 750 1
size: 750

 timeSerial: 28215222026

 timeParallel: 14942886093


 timeSerialTranspose: 3527808242
 timeSerialTransposeXMM: 2100344588
 timeParallelTranspose: 1750848780
 timeParallelTransposeXMM: 1206283206
 timeParallelTransposeTagTeam: 763463846
 timeParallelTransposeTagTeamXMM: 449226073
```

Πολλαπλασιασμός τετραγωνικού πίνακα 750 x 750 με βήμα 1

- Serial: 28215222026(ticks)
- Parallel: 14942886093(ticks)

Η παράλληλη εκτέλεση είναι 1.89x πιο γρήγορη.

SUPERSCALAR PROGRAMMING- ΕΦΑΡΜΟΓΗ 2

```
openuser@snf-17063:~/telikh/QuickThreadLinux2.0.0/MatrixMultiply/Debug$ ./MatrixMultiply 750 750
size: 750

 timeSerial: 27747020172

 timeParallel: 14191245940


 timeSerialTranspose: 2666465884
 timeSerialTransposeXMM: 1695812790
 timeParallelTranspose: 1809441306
 timeParallelTransposeXMM: 1135432332
 timeParallelTransposeTagTeam: 587579348
 timeParallelTransposeTagTeamXMM: 377137115
```

Πολλαπλασιασμός τετραγωνικού πίνακα 750 x 750 με default βήμα 2

- Serial: 27747020172(ticks)
- Parallel: 14191245940(ticks)

Η παράλληλη εκτέλεση είναι 1.96x πιο γρήγορη.

SUPERSCALAR PROGRAMMING-ΕΦΑΡΜΟΓΗ 3

```
openuser@snf-17063:~/telikh/QuickThreadLinux2.0.0/MatrixMultiply/Debug$ ./MatrixMultiply 1000 1000 1
size: 1000

 timeSerial: 85411290659

 timeParallel: 43257852742


 timeSerialTranspose: 8695494734
 timeSerialTransposeXMM: 6317281973
 timeParallelTranspose: 4455410886
 timeParallelTransposeXMM: 4009952465
 timeParallelTransposeTagTeam: 1921496952
 timeParallelTransposeTagTeamXMM: 1301719686
openuser@snf-17063:~/telikh/QuickThreadLinux2.0.0/MatrixMultiply/Debug$
```

Πολλαπλασιασμός τετραγωνικού πίνακα 1000 x 1000 με βήμα 1

- Serial: 85411290659(ticks)
- Parallel: 43257852742(ticks)

Η παράλληλη εκτέλεση είναι 1.97x πιο γρήγορη.

SUPERSCALAR PROGRAMMING- ΕΦΑΡΜΟΓΗ 4

```
openuser@snf-17063:~/telikh/QuickThreadLinux2.0.0/MatrixMultiply/Debug$ ./MatrixMultiply 1000 1000 2
size: 1000

 timeSerial: 84175686167

 timeParallel: 42748717350


 timeSerialTranspose: 8662519315
 timeSerialTransposeXMM: 6474831035
 timeParallelTranspose: 4376535421
 timeParallelTransposeXMM: 3613339320
 timeParallelTransposeTagTeam: 1584544584
 timeParallelTransposeTagTeamXMM: 1386327782
```

Πολλαπλασιασμός τετραγωνικού πίνακα 1000 x 1000 με default βήμα 2

- Serial: 84175686167(ticks)
- Parallel: 42748717350(ticks)

Η παράλληλη εκτέλεση είναι 1.97x πιο γρήγορη.

ΣΥΝΟΨΗ-ΣΥΜΠΕΡΑΣΜΑΤΑ

- Στην εργασία αυτή εκτιμήθηκαν με σχετική ακρίβεια οι επιδόσεις ενός superscalar επεξεργαστή αλλά υπήρχε και η δυνατότητα να γίνει κατανοητή η αρχιτεκτονική και η λειτουργία τέτοιου είδους επεξεργαστών.

- Το σημαντικότερο ίσως πλεονέκτημα αυτής της μεθόδου είναι ότι επιτρέπει σε πολύ λιγότερο χρόνο να εκτελεστεί μια αλληλουχία εντολών το οποίο είναι απαραίτητο για την επεξεργασία μεγάλου και πολύπλοκου όγκου προγράμματος.

- Εάν σε συνδυασμό με έναν superscalar επεξεργαστή υπάρχει και ένα πολύ ορθά δομημένο πρόγραμμα έτσι ώστε η ανάκληση και η εκτέλεση των εντολών να πραγματοποιείται αποδοτικά τότε οι επιδόσεις του συστήματος βελτιώνονται σημαντικά.

- Γι ’αυτό το λόγο στην σημερινή εποχή οι superscalar επεξεργαστές χρησιμοποιούνται ευρέως από μεγάλες εταιρίες όπως η Intel, η AMD κτλ.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- http://en.wikipedia.org/wiki/Superscalar
- http://whatis.techtarget.com/definition/superscalar
- http://www.techterms.com/definition/superscalar
- http://www.bsc.es/computer-sciences/programming-models/smp-superscalar
- ftp://reports.stanford.edu/pub/cstr/reports/csl/tr/89/383/CSL-TR-89-383.pdf
- http://www.ida.liu.se/~TDTS51/lectures/lectures7-8.pdf
- http://software.intel.com/en-us/articles/superscalar-programming-101-matrix-multiply-part-1