Πολυνηματικές Αρχιτεκτονικές
&
«Μη-Παραδοσιακός» Παραλληλισμός

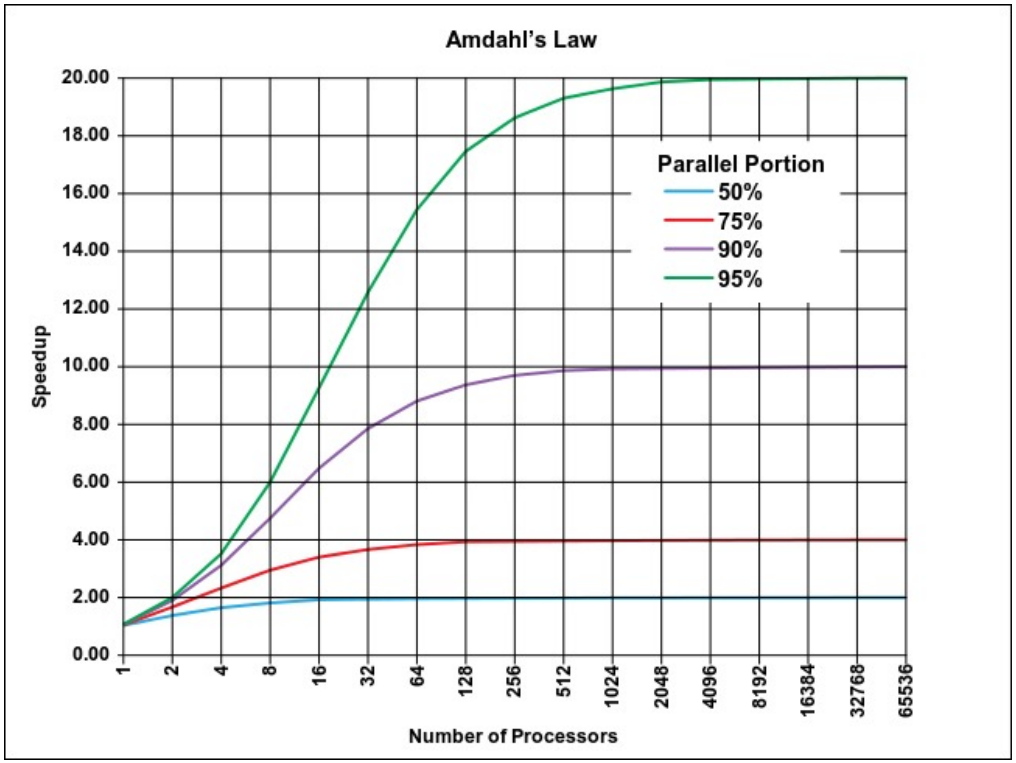
Πολυνηματικοί επεξεργαστές

- *Στοχεύουν:*
  - throughput πολυπρογραμματιζόμενων φορτίων
  - latency πολυνηματικών εφαρμογών

Πολυνηματικοί επεξεργαστές

- *Στοχεύουν:*
  - throughput πολυπρογραμματιζόμενων φορτίων
  - latency πολυνηματικών εφαρμογών

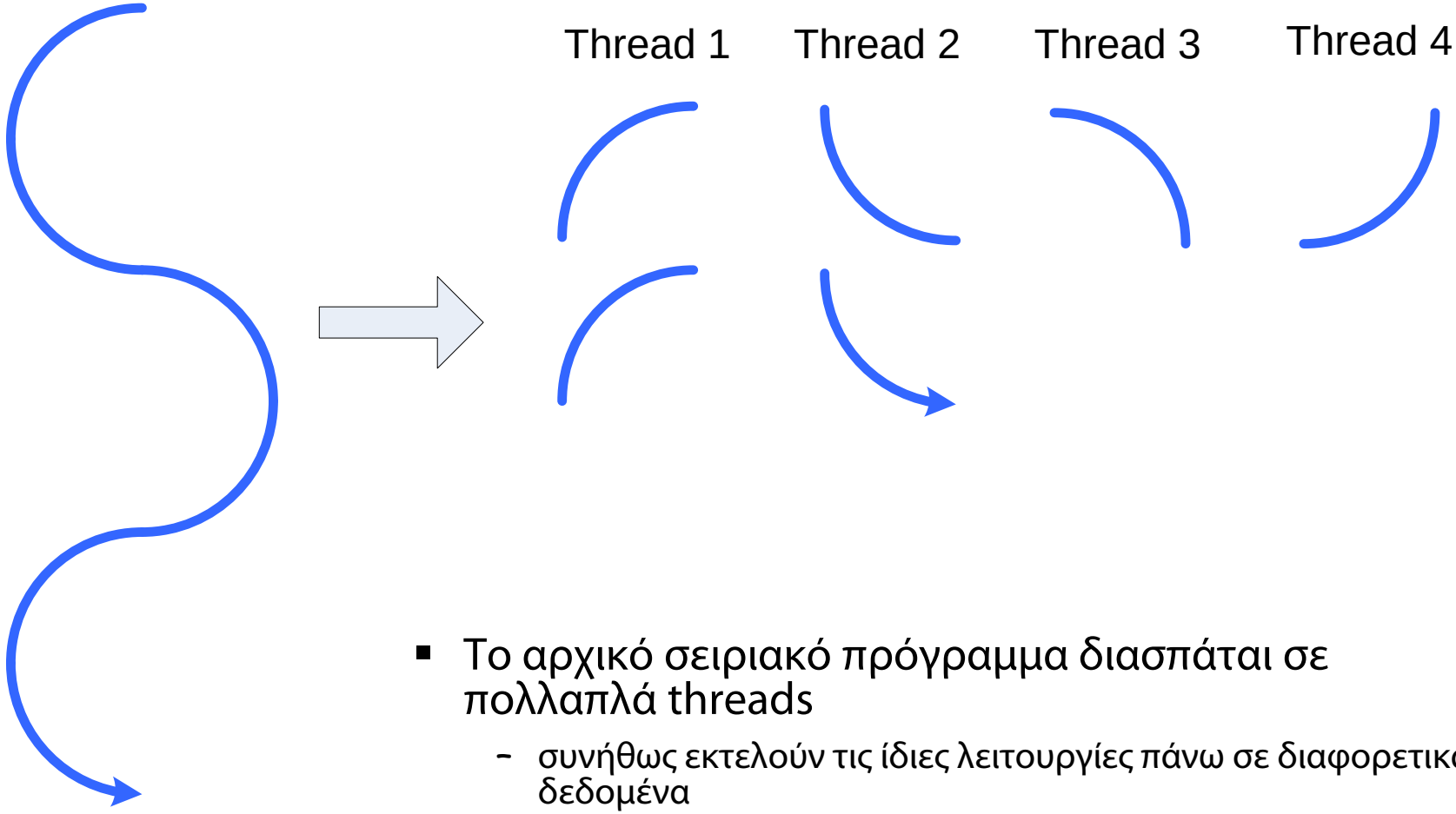
$$Speedup(n) = \frac{1}{(1-P) + \frac{P}{n}}$$



Πολυνηματικοί επεξεργαστές

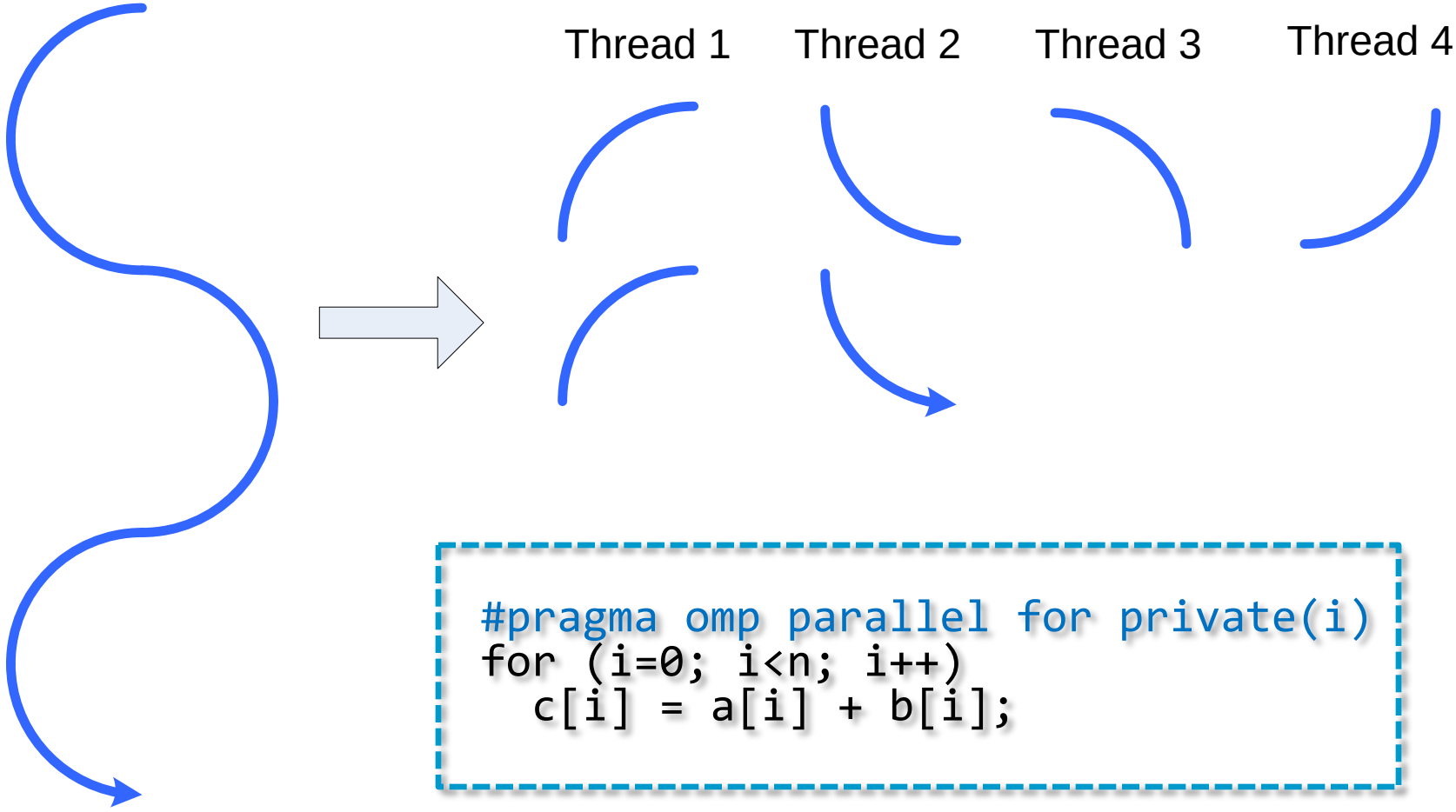
- *Στοχεύουν:*
  - throughput πολυπρογραμματιζόμενων φορτίων
  - latency πολυνηματικών εφαρμογών
- *Πρόκληση:*
  - latency μονο-νηματικών, μη (εύκολα) παραλληλοποιήσιμων εφαρμογών
- *Πώς;*
  - Μη-συμβατικές μέθοδοι παραλληλισμού

«Παραδοσιακός» παραλληλισμός



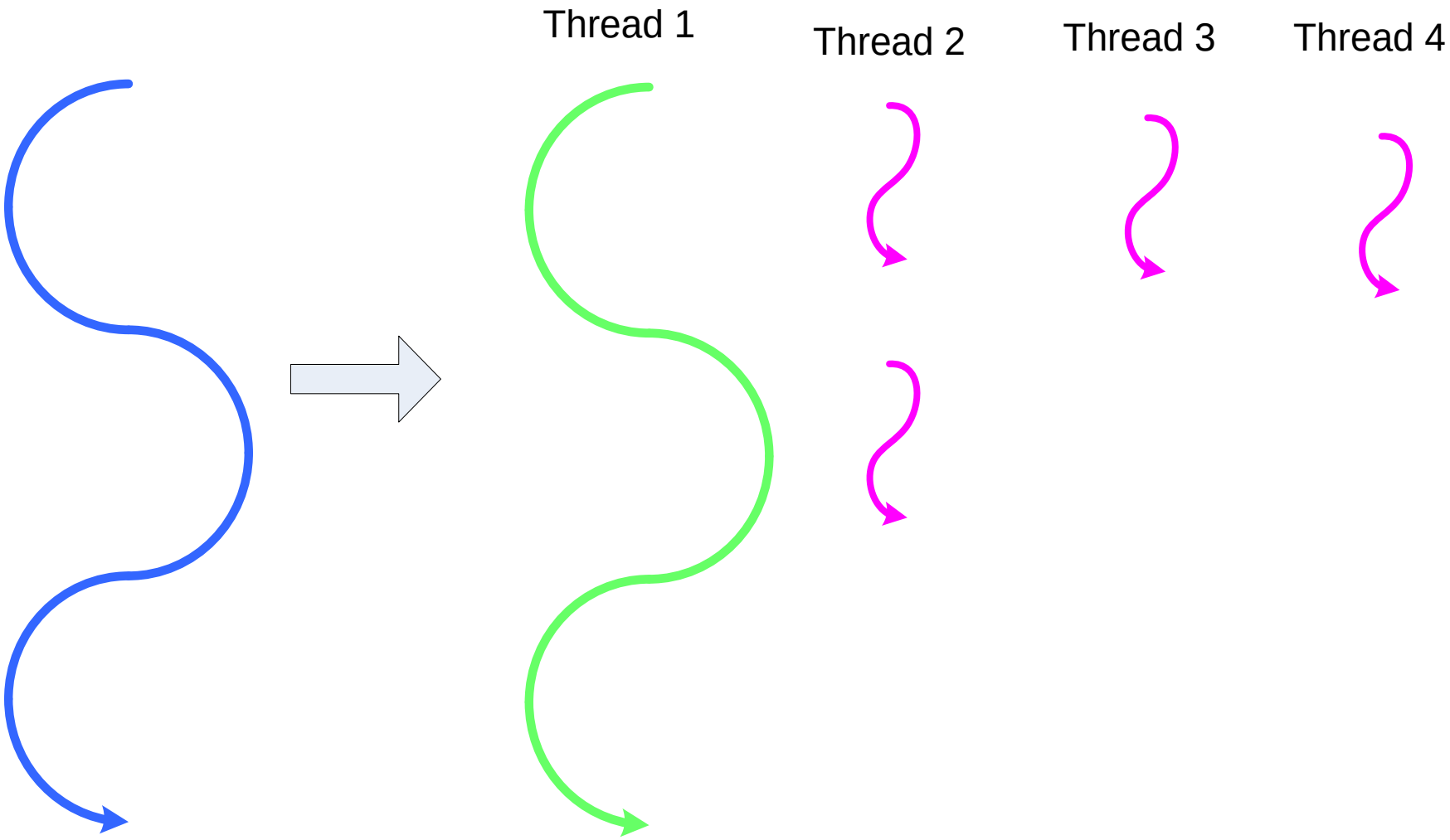
- Το αρχικό σειριακό πρόγραμμα διασπάται σε πολλαπλά threads
  - συνήθως εκτελούν τις ίδιες λειτουργίες πάνω σε διαφορετικά δεδομένα
  - ίδιος φόρτος, ίδιο προφίλ
- Βελτίωση απόδοσης από την παράλληλη εκτέλεση πολλαπλών threads σε πολλαπλά contexts

«Παραδοσιακός» παραλληλισμός




```
#pragma omp parallel for private(i)
for (i=0; i<n; i++)
  c[i] = a[i] + b[i];
```

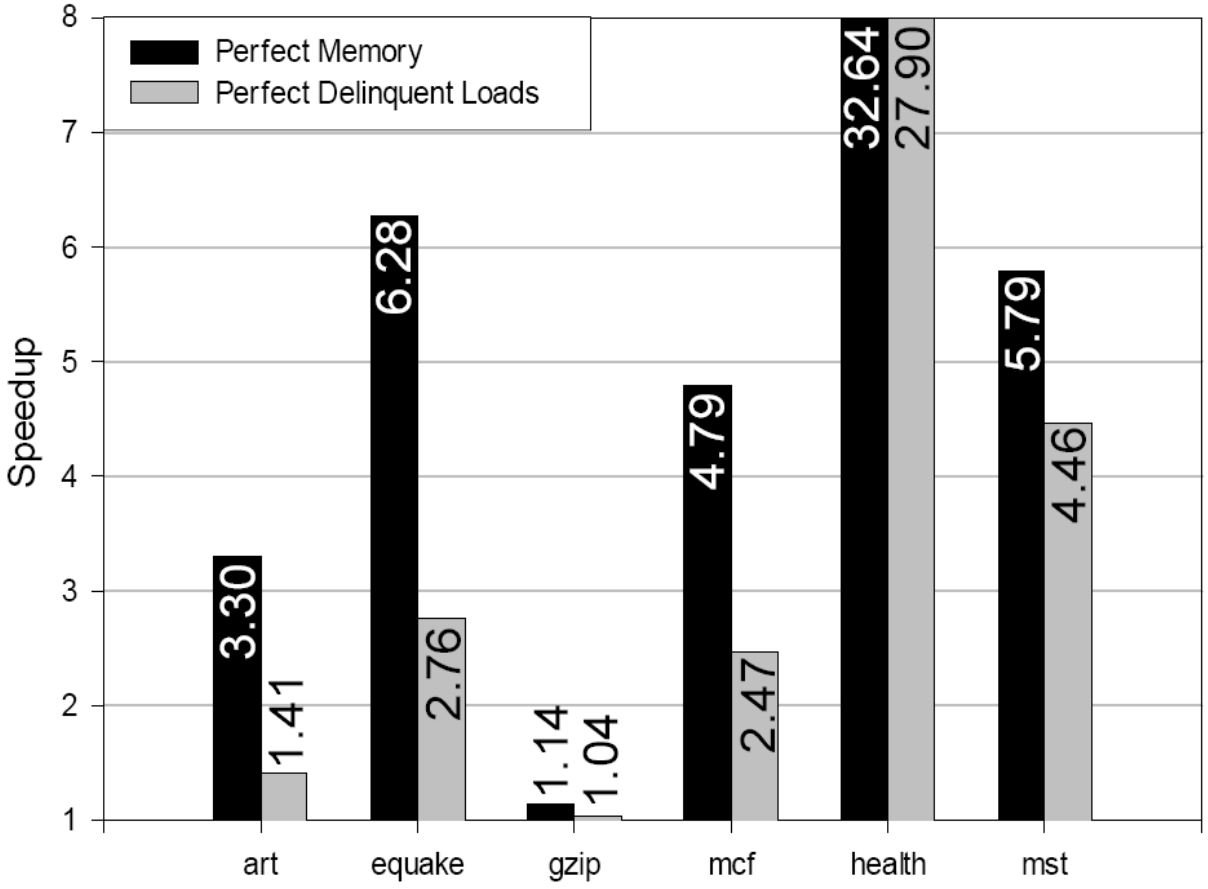
«Μη-παραδοσιακός» παραλληλισμός



«Μη-παραδοσιακός» παραλληλισμός

- Προσθήκη ενός ή περισσότερων νημάτων για να επιταχύνουν *έμμεσα* την εκτέλεση του αρχικού προγράμματος, χωρίς αναγκαστικά να αναλαμβάνουν *άμεσα* κάποιο μέρος των υπολογισμών του
- Πώς;
  - μείωση memory latency μέσω prefetching
  - προ-υπολογισμός branches
  - event-driven dynamic code optimizations
- Πού;
  - στο software (προγραμματιστής / compiler)
  - στο hardware
- Πλεονεκτήματα:
  - εφαρμόσιμο σε όλους τους κώδικες (παραλληλοποιήσιμους ή μη)
  - διακριτική λειτουργία βοηθητικών νημάτων
  - αποδέσμευση βοηθητικών νημάτων από την πορεία του κύριου νήματος
  - αποτελεσματικό για «δύσκολα» μοτίβα

Speculative Precomputation (SP) – Κίνητρο

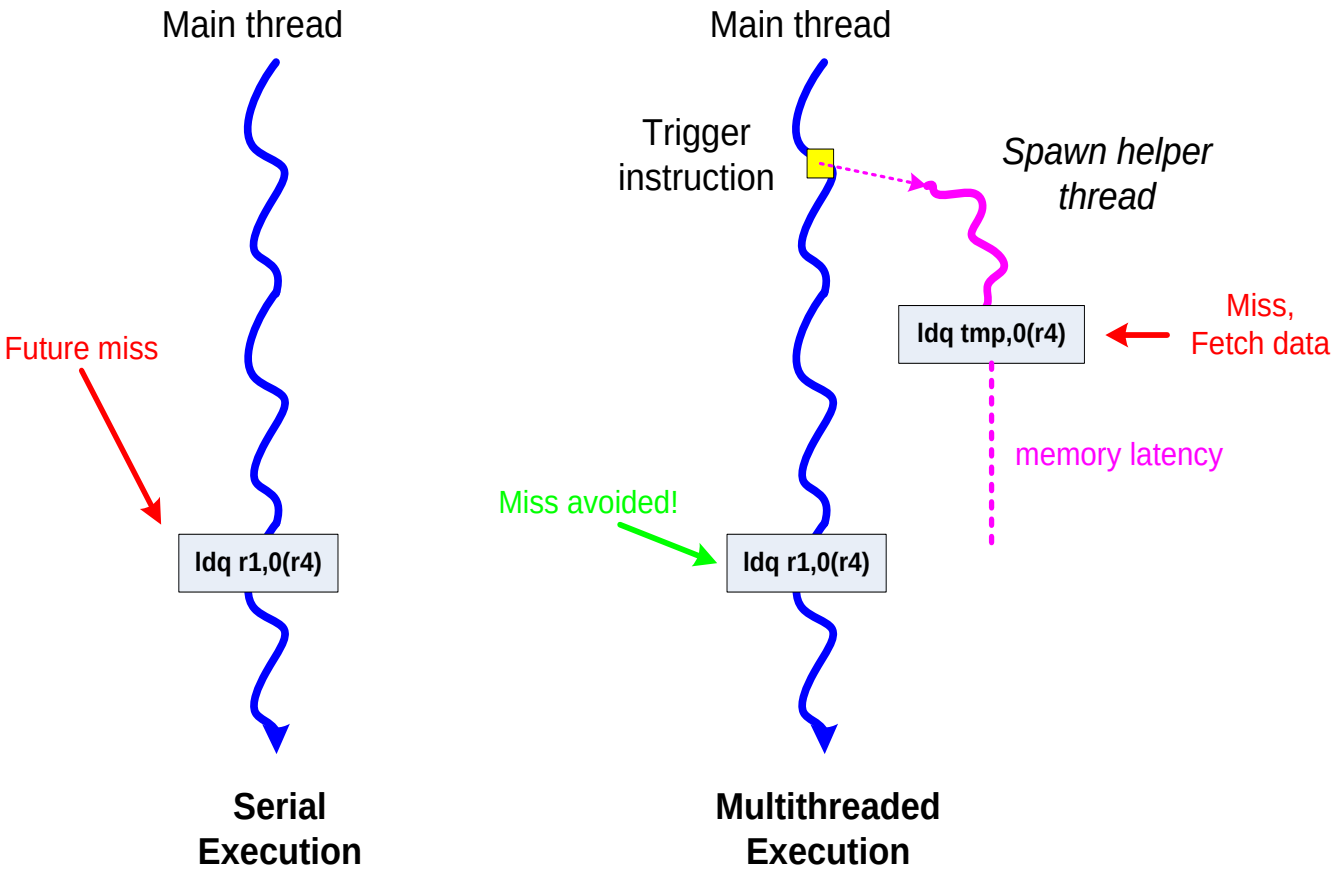


- Delinquent loads: τα πιο «cache-missing» loads

Speculative Precomputation (SP)

- Precomputation slice (*P-slice*): είναι ένα νήμα που περιλαμβάνει το δυναμικό ίχνος εκτέλεσης (execution trace) ανάμεσα σε μια triggering εντολή στο main thread και ένα delinquent load
- Όλες οι εντολές που δεν οδηγούν στον υπολογισμό της διεύθυνσης του DL μπορούν να αφαιρεθούν με ασφάλεια (~90-95%)
- Τιμές «live-in» καταχωρητών (τυπικά, 2-6) για το p-slice αντιγράφονται από το main thread στο helper thread

Speculative Precomputation (SP)



Ενδεικτικά βήματα

1. Εντοπισμός delinquent loads
   - memory access profiling: simulators (π.χ. Valgrind, Simics), profilers (Oprofile)
   - εδώ: L1 cache misses (shared L1 cache)
2. Εισαγωγή triggering points
   - 1st pass:
     - εκτέλεση προγράμματος μέχρι κάποιο DL
     - τοποθέτηση *πιθανού triggering point* αρκετές (π.χ.128) θέσεις πριν στο δυναμικό instruction stream
   - 2nd pass:
     - εκτέλεση προγράμματος μέχρι κάποιο πιθανό triggering point
     - παρατήρηση επόμενων (π.χ. 256) εντολών
     - αν το αντίστοιχο DL εκτελείται με μεγάλη πιθανότητα, το triggering point κατοχυρώνεται και καταγράφεται το δυναμικό ίχνος εντολών ανάμεσα σε αυτό και το DL

Ενδεικτικά βήματα

3. Κατασκευή P-slices
   - από το αρχικό P-slice αφαιρούνται όσες εντολές δε χρειάζονται για τον υπολογισμό του DL (→ P-slices από 5-15 εντολές, συνήθως)
   - από το τελικό P-slice καταγράφονται οι «live-in» registers
4. Σύνδεση των P-slices στο binary του προγράμματος
   - π.χ. προσθήκη σε ειδικό μέρος του text segment
5. Εκτέλεση προγράμματος
   - όποτε συναντάται triggering point, δημιουργείται ένα speculative thread:
     - δέσμευση hardware context
     - αντιγραφή live-in registers από το main στο speculative thread
     - εκκίνηση speculative thread

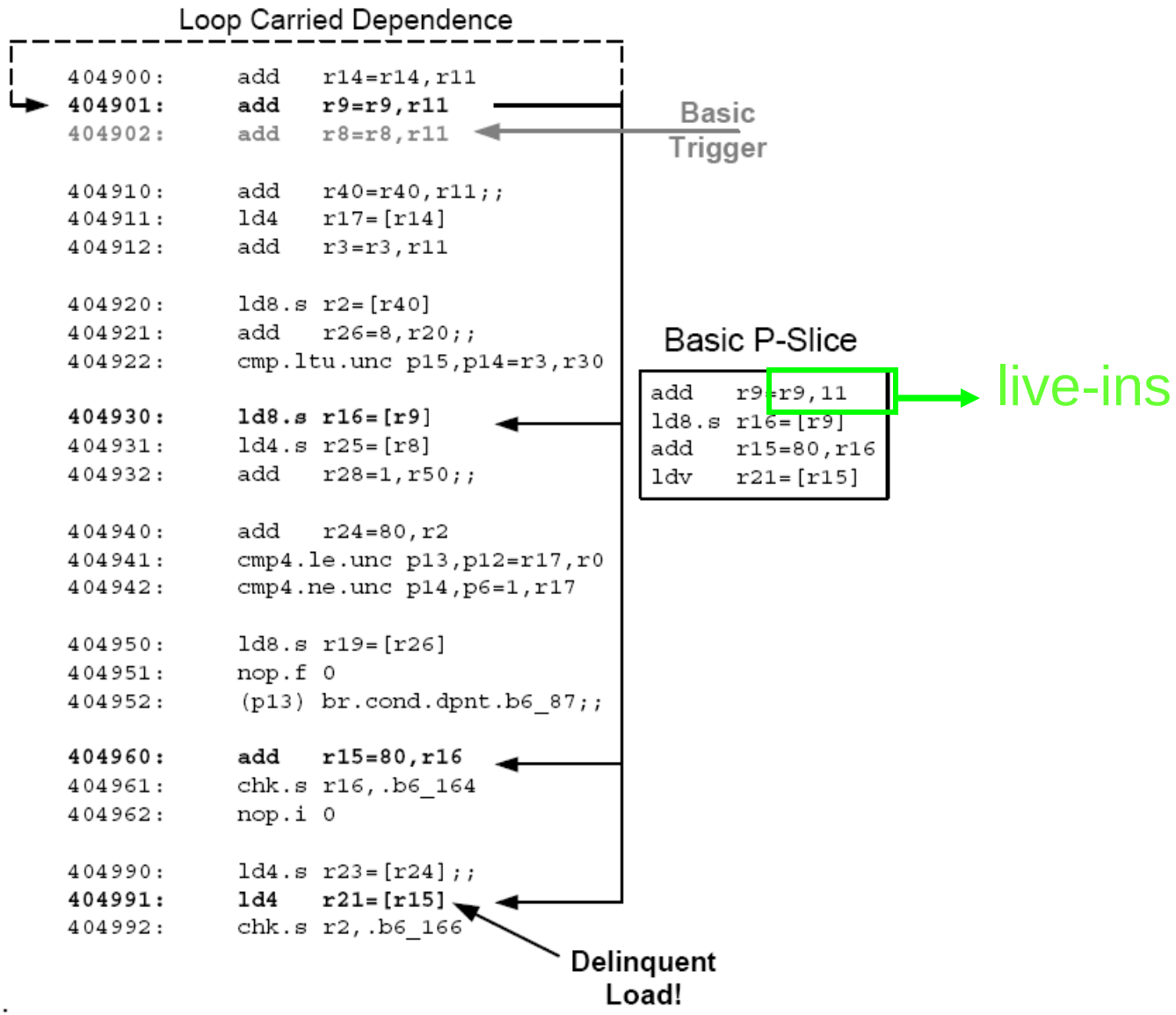
Παράδειγμα: mcf

```
arc=arcs+group_pos;
for(;arc<stop_arcs;arc+=nr_group){
   if(arc->ident>BASIC){
      red_cost=arc->cost-arc->tail->potential+
                        arc->head->potential;
      if((red_cost<0&&arc->ident==AT_LOWER)||
         (red_cost>0&&arc->ident==AT_UPPER)){
         basket_size++;
         perm[basket_size]->a=arc;
         perm[basket_size]->cost=red_cost;
         perm[basket_size]->abs_cost=ABS(red_cost);
      }
   }
}
```

Delinquent Load#1
Delinquent Load#2
Delinquent Load#3

	L1 Miss Rate / % Capacity Miss	L2 Miss Rate / % Capacity Miss	L3 Miss Rate / % Capacity Miss
Delinquent Load# 1	99.95% / 99.98%	48.06% / 82.78%	67.64% / 97.38%
Delinquent Load# 2	80.92% / 97.60%	63.55% / 86.51%	20.04% / 47.88%
Delinquent Load# 3	93.10% / 99.1%	45.33% / 74.65%	20.70% / 44.74%

Παράδειγμα: mcf



Πλεονεκτήματα σε σχέση με το «παραδοσιακό» prefetching

- Software prefetching
  - prefetch εντολές στο σώμα του κύριου thread που στοχεύουν μελλοντικές αναφορές μνήμης
  - από τον προγραμματιστή ή τον compiler
  - αποδοτικό για λίγα + εύκολα υπολογίσιμα DLs (π.χ. με βάση κάποιο offset)
- Hardware prefetching
  - παρατηρεί το stream αναφορών, αναγνωρίζει μοτίβα αναφορών ή/και misses και τα προφορτώνει από τη μνήμη
  - αποδοτικό για προβλέψιμα και «κανονικά» access patterns

Πλεονεκτήματα σε σχέση με το «παραδοσιακό» prefetching

- Thread-based prefetching (SP)
  - επειδή χρησιμοποιεί μέρος του αρχικού κώδικα, προ-υπολογίζει αποτελεσματικά «δύσκολες» αναφορές στη μνήμη
    - » pointer chasing codes, indirect accesses, random/complex/recursive control flows, etc.
  - επειδή το prefetching γίνεται σε ξεχωριστό(ά) thread(s), ο «θόρυβος» που εισάγεται στο βασικό thread (π.χ. στα instruction queues) είναι μικρότερος σε σχέση με το software prefetching
  - επειδή η εκτέλεση των speculative threads είναι αποδεσμευμένη από αυτή του βασικού thread, το prefetching δεν περιορίζεται από την πορεία εκτέλεσης του βασικού thread

Ζητήματα απόδοσης

- Γρήγορη αντιγραφή των live-ins από το main στο speculative thread
- Γρήγορη δέσμευση διαθέσιμου hardware context + εκκίνηση speculative thread
- Contention για shared resources
- Ποιος ο ρόλος του OS?

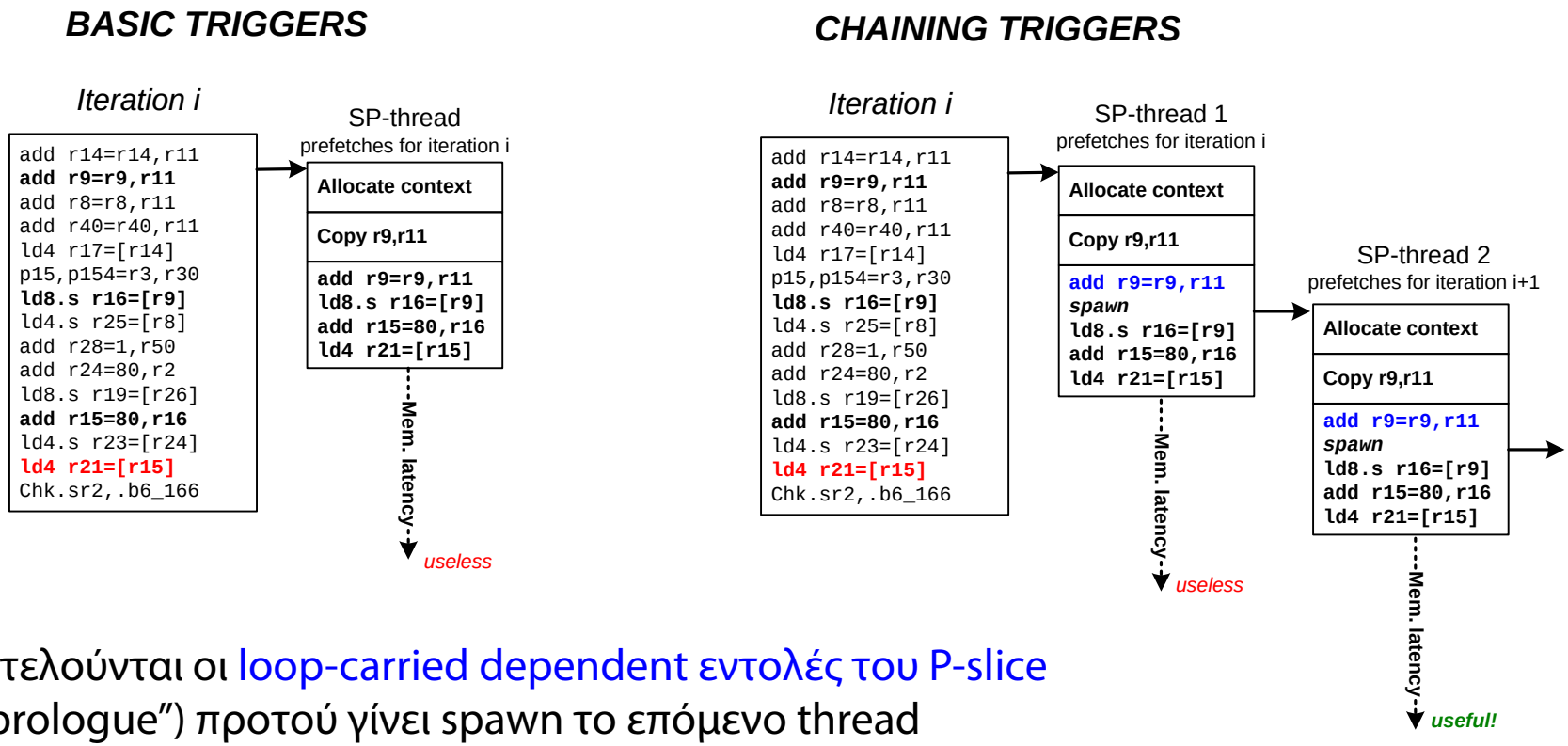
- Τι γίνεται όταν η απόσταση ανάμεσα στο triggering point και το DL είναι μικρή (σε σχέση με το memory latency)

  ή αλλιώς…

  Τι γίνεται αν θέλουμε να στοχεύσουμε σε DLs που βρίσκονται αρκετά iterations μπροστά από το main thread?

Chaining triggers

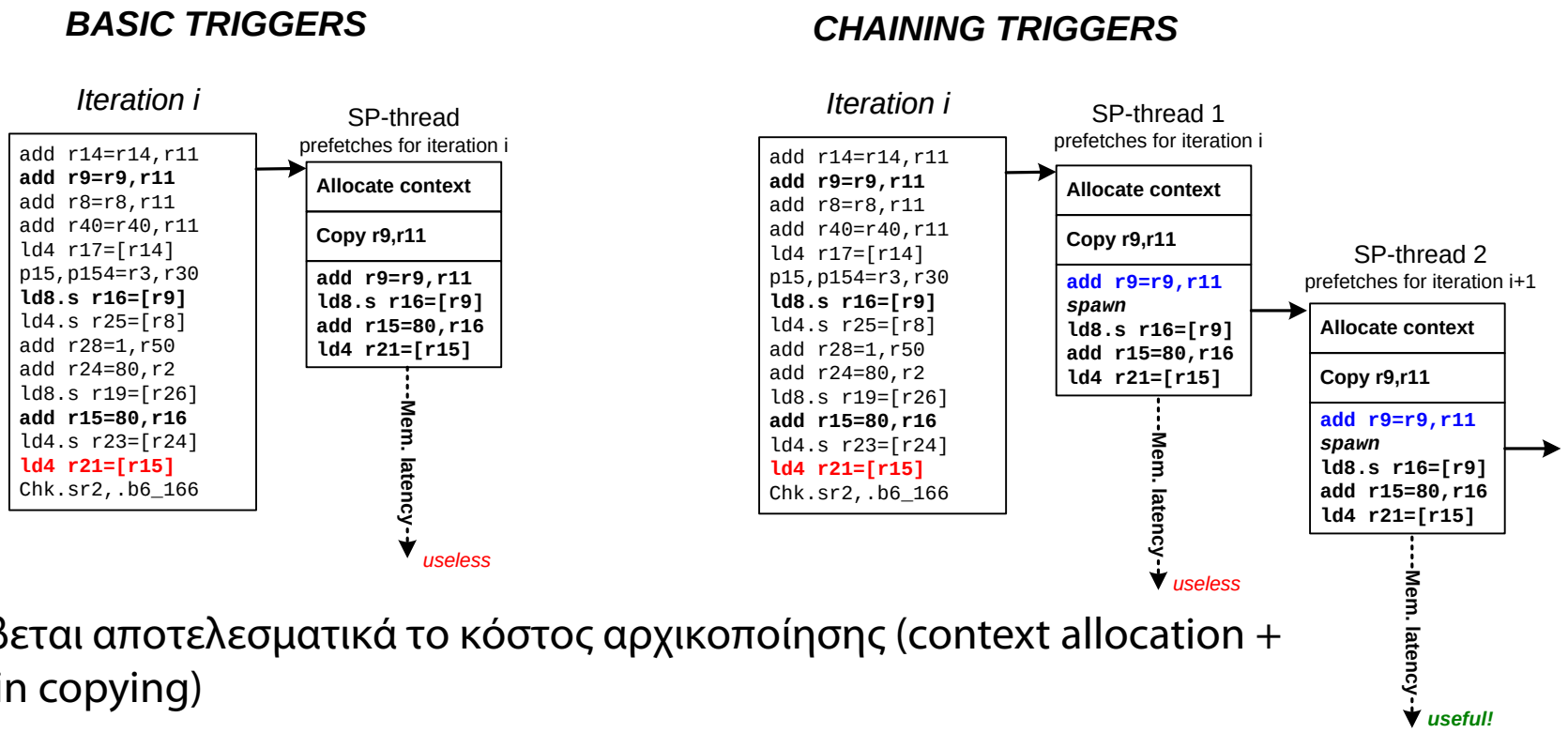
- Επιτρέπουν σε ένα speculative thread να δημιουργήσει άλλα speculative threads τα οποία μπορούν να τρέξουν «αυθαίρετα μπροστά» από το βασικό thread



- εκτελούνται οι loop-carried dependent εντολές του P-slice ("prologue") προτού γίνει spawn το επόμενο thread
  - γιατί να μην εκτελέσουμε N φορές τον πρόλογο για να φτάσουμε στις επόμενες N επαναλήψεις («*induction unrolling*»)?

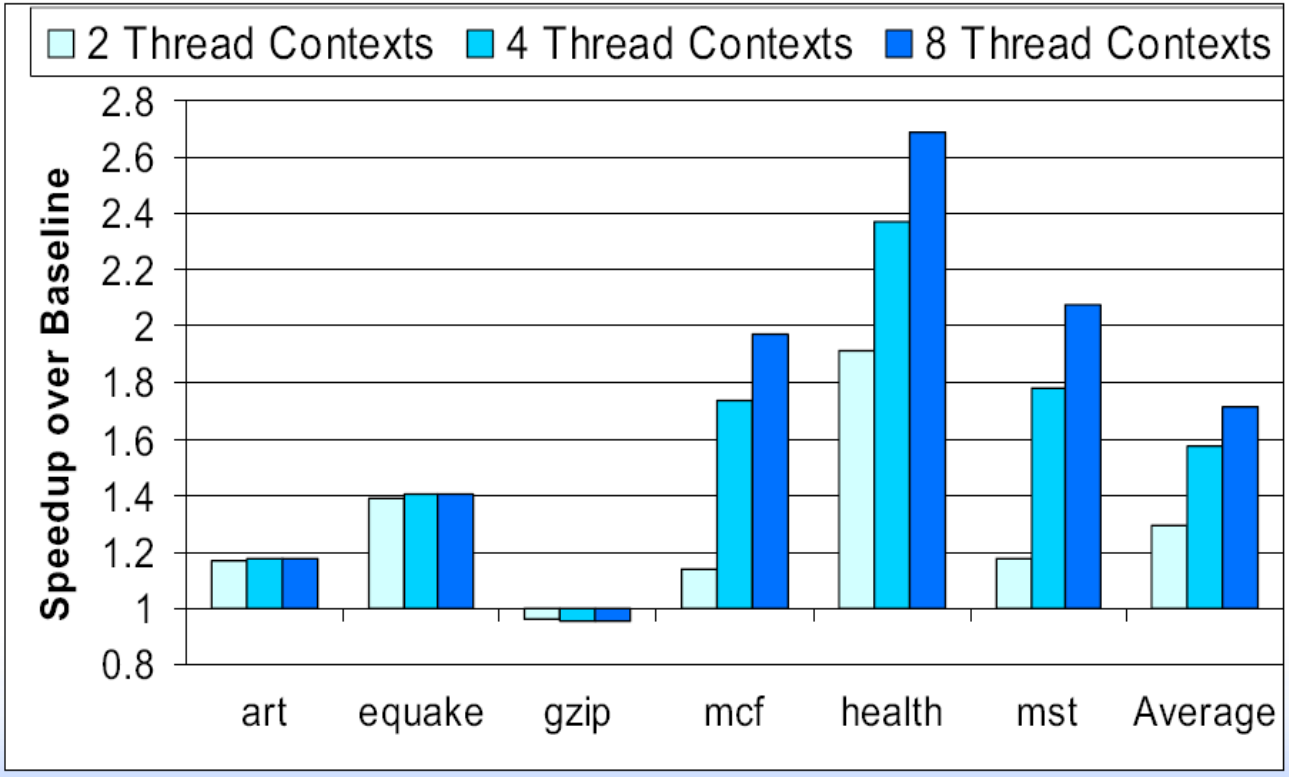
Chaining triggers

- Επιτρέπουν σε ένα speculative thread να δημιουργήσει άλλα speculative threads τα οποία μπορούν να τρέξουν «αυθαίρετα μπροστά» από το βασικό thread



- κρύβεται αποτελεσματικά το κόστος αρχικοποίησης (context allocation + live-in copying)
- απαιτείται μηχανισμός ανάσχεσης δημιουργίας νέων threads
  - αποτρέπεται η εκτόπιση prefetched cache lines που δεν έχουν χρησιμοποιηθεί ακόμα από το main thread

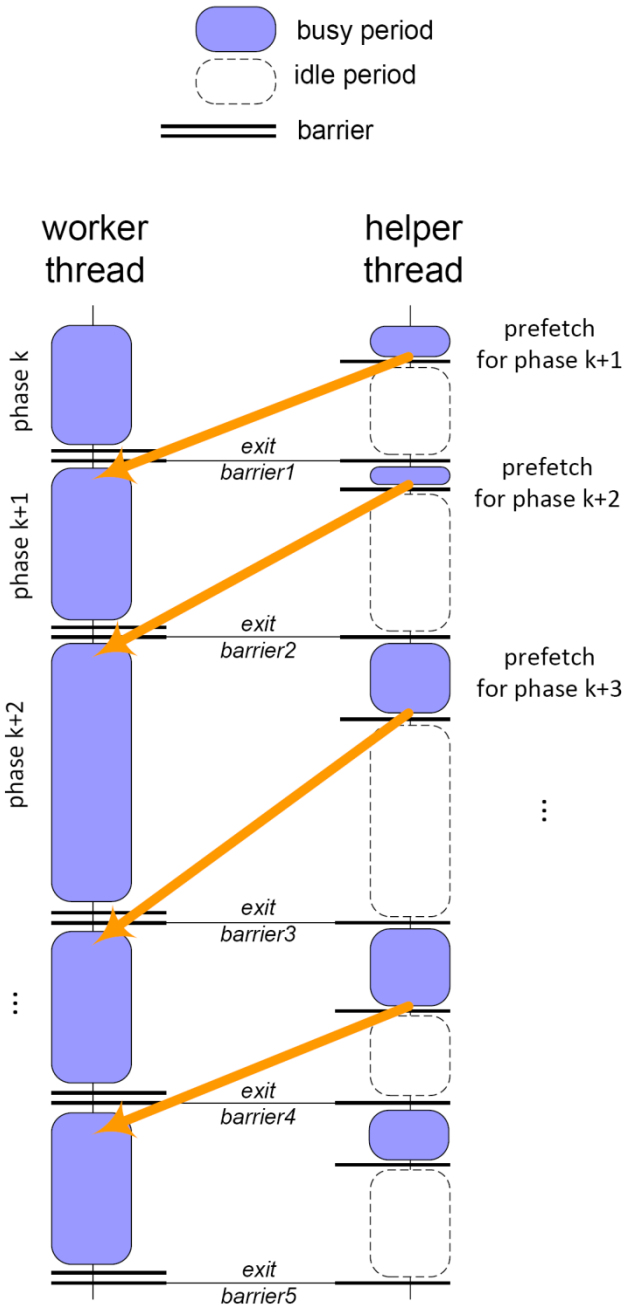
SP Performance



- περισσότερα contexts → περισσότερα chaining triggers

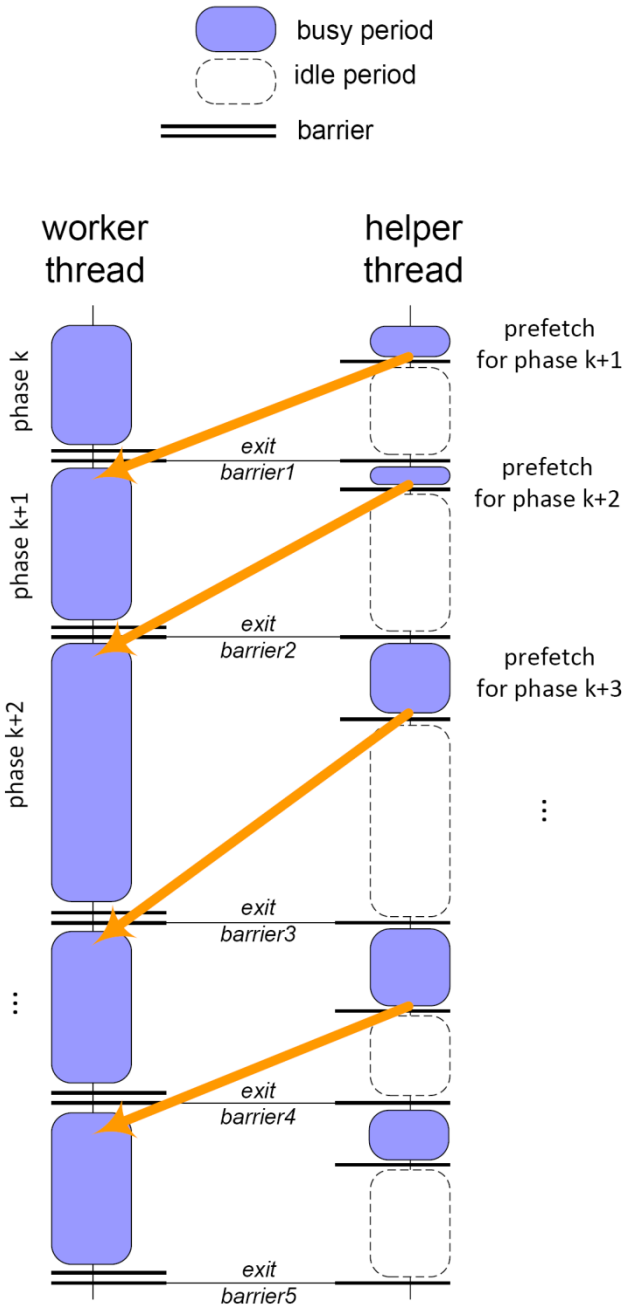
Παράδειγμα: υλοποίηση SPR σε πραγματικό σύστημα (HT-based)

- Εντοπισμός κρίσιμων εντολών ανάγνωσης
  - προσομοίωση κρυφής μνήμης L2
- Κατασκευή prefetching threads
  - απομόνωση προς-τα-πίσω εξαρτώμενων εντολών
- Εισαγωγή σημείων συγχρονισμού
  - μοντέλο παραγωγού-καταναλωτή
  - επιλογή περιοχών προφόρτωσης
  - φράγματα συγχρονισμού στα σημεία εισόδου και εξόδου κάθε περιοχής



Παράδειγμα: υλοποίηση SPR σε πραγματικό σύστημα (HT-based)

```
for (i=0; i<N; i++) {
  for (j=0; j<M; j++) {
    for (k=0; k<L; k++) {
      do_work(A[i], B[j], C[B[j]]);
    }

  }
}
```



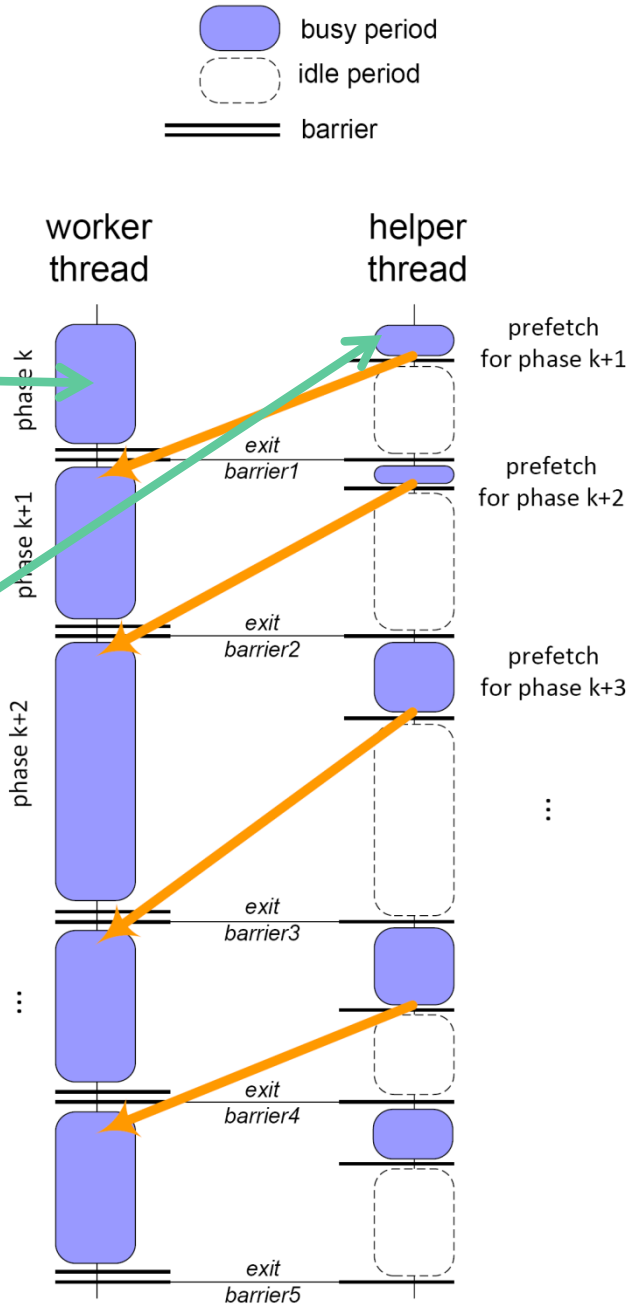
Παράδειγμα: υλοποίηση SPR σε πραγματικό σύστημα (HT-based)

Main thread

```
for (i=0; i<N; i++) {
  for (j=0; j<M; j++) {

    barrier;
    for (k=0; k<L; k++) {
      do_work(A[i], B[j], C[B[j]]);
    }

  }
}
```

Helper thread

```
for (i=0; i<N; i++) {
  for (j=0; j<M; j++) {

    for (k=0; k<L; k++) {
      prefetch_data(C[B[j]]);
    }
    barrier;

  }
}
```



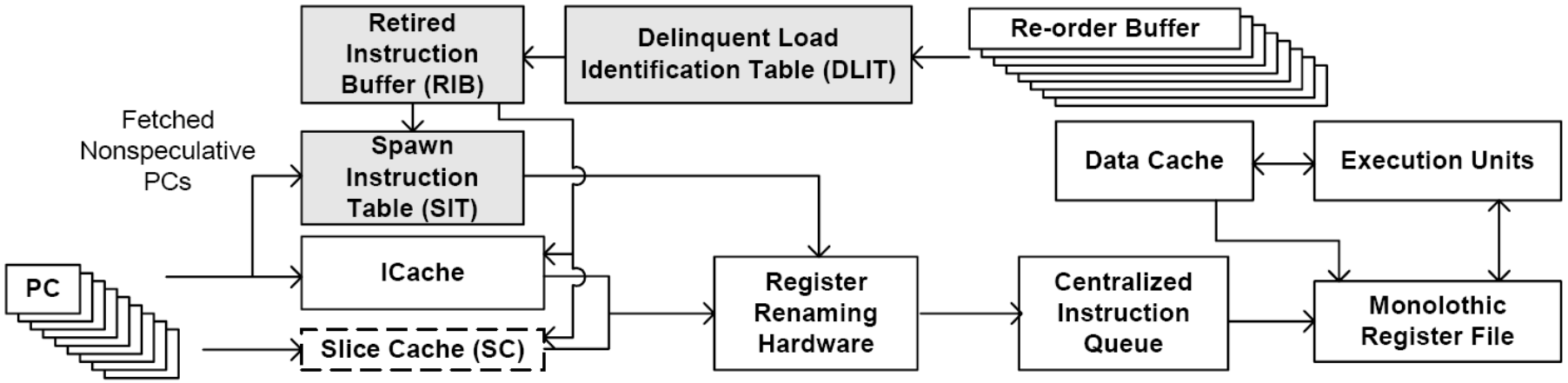
Dynamic Speculative Precomputation

- Όπως το SP, με τη διαφορά ότι υπάρχει hardware support για:
  - εντοπισμό DLs
  - κατασκευή p-slices και διαχείριση speculative threads
  - βελτιστοποίηση των sp. threads όταν δε δουλεύουν καλά
  - κατάργηση των sp. threads όταν δε δουλεύουν καθόλου
  - καταστροφή των sp. threads όταν δε χρειάζονται άλλο

Dynamic Speculative Precomputation

- Όπως το hardware prefetching...
  - δεν απαιτεί software support
  - δεν απαιτεί recompilation
  - είναι ανεξάρτητο της υποκείμενης αρχιτεκτονικής
- Όπως το SP...
  - εισάγει ελάχιστη παρεμβολή στο βασικό thread
  - εκτελείται ανεξάρτητα από αυτό
  - αποδίδει σε irregular access patterns

Τροποποιημένο SMT pipeline για DSP



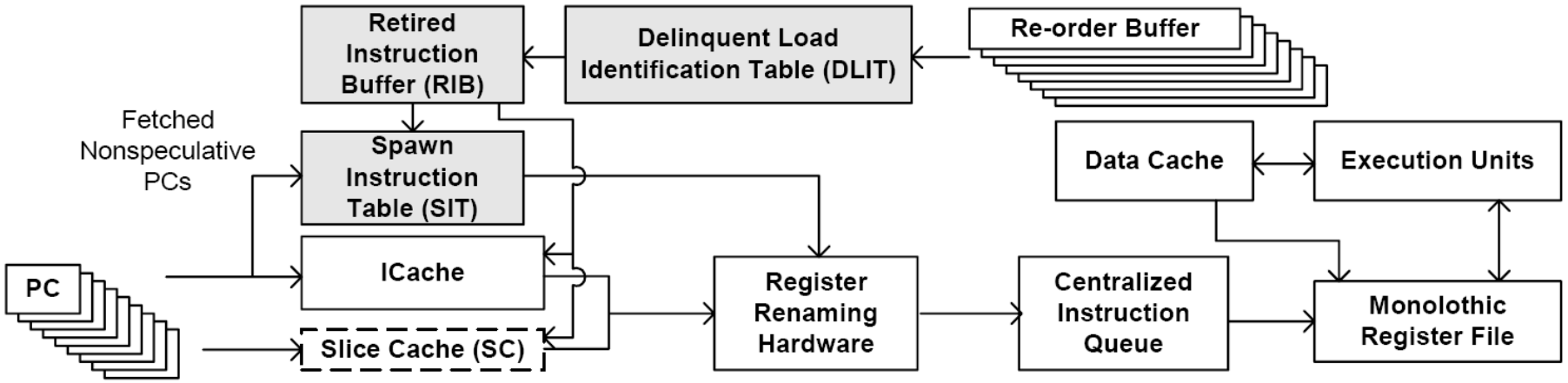
- Εντοπισμός DLs
  - loads που απέτυχαν στην L2 εισέρχονται στο DLIT (FCFS)
  - heuristic για κατοχύρωση ως DL και κλείδωμα στο DLIT:

    αν μετά από 128K εντολές το load εξακολουθεί να κατέχει το entry του στο DLIT

    & έχει εκτελεστεί 100+ φορές από τότε

    & κάθε φορά χρειάστηκε >4 κύκλους για να πάρει τα δεδομένα

Τροποποιημένο SMT pipeline για DSP

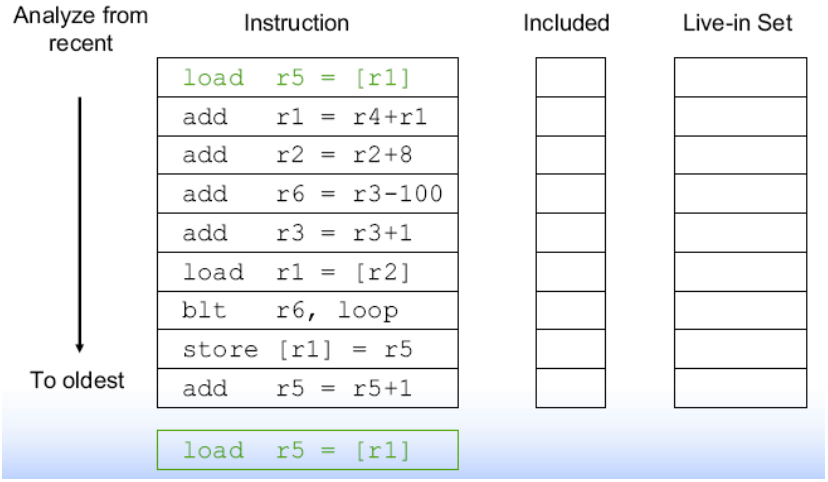


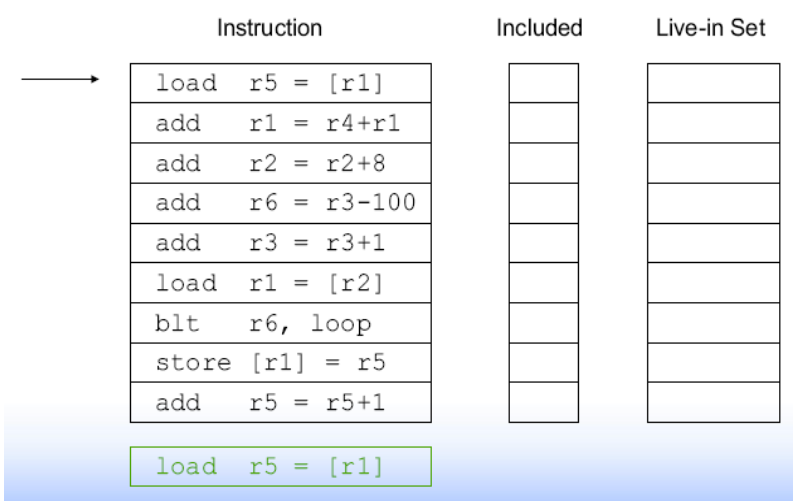
- Κατασκευή P-slices
  - όταν ένα DL (για το οποίο δεν υπάρχει P-slice) γίνεται commit, τότε οι εντολές του thread γίνονται buffer στον RIB μέχρι το επόμενο instance του DL
  - ο RIB πραγματοποιεί backward dependence analysis στο instruction trace και κρατά μόνο τις εντολές από τις οποίες εξαρτάται ο υπολογισμός του DL, ανανεώνοντας κατάλληλα το live-in set του P-slice
  - μόλις φτάσει στην πρώτη εντολή στο trace (το 1ο instance του DL, στην ουσία) την μαρκάρει ως triggering

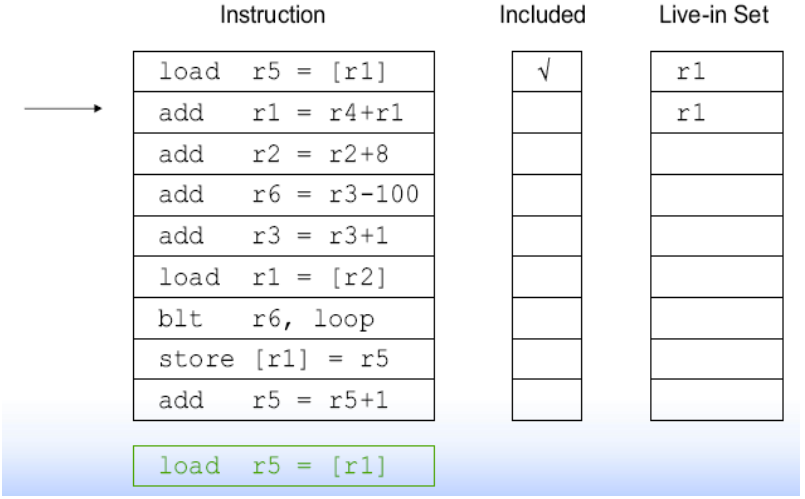
Παράδειγμα κατασκευής P-slice

```
struct DATATYPE {
  int val[10];
};

DATATYPE * data [100];

for(j = 0; j < 10; j++) {
  for(i = 0; i < 100; i++) {
    data[i]->val[j]++;
  }
}
```

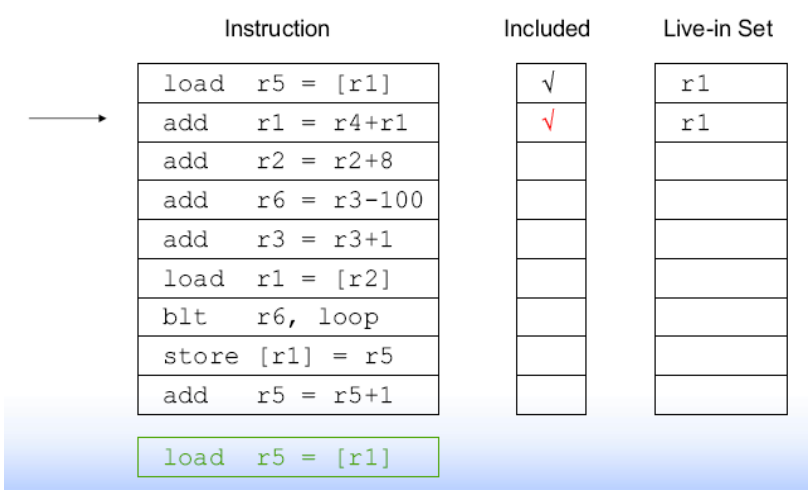
```
        loop:
I1      load    r1=[r2]
I2      add     r3=r3+1
I3      add     r6=r3-100
I4      add     r2=r2+8
I5      add     r1=r4+r1
I6      load    r5=[r1]
I7      add     r5=r5+1
I8      store   [r1]=r5
I9      blt     r6, loop
```

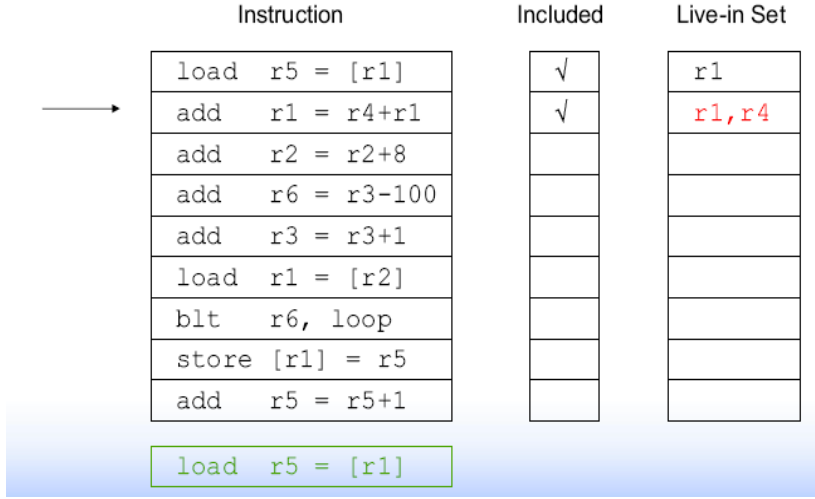


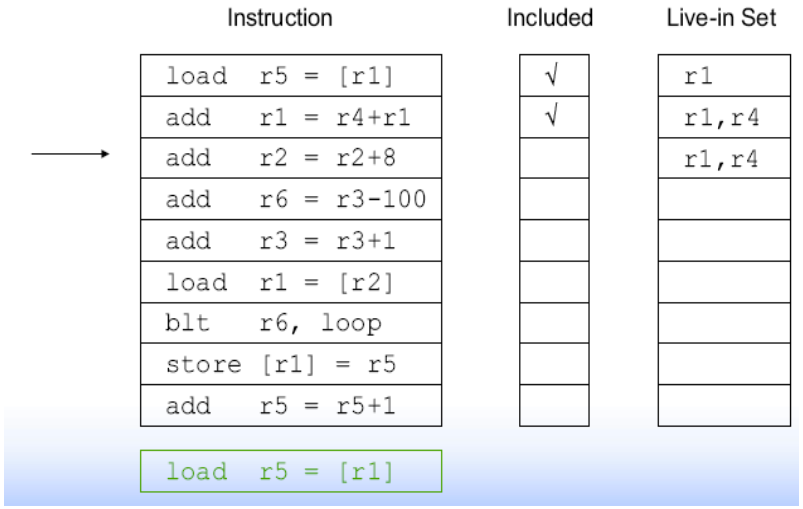




Παράδειγμα κατασκευής P-slice

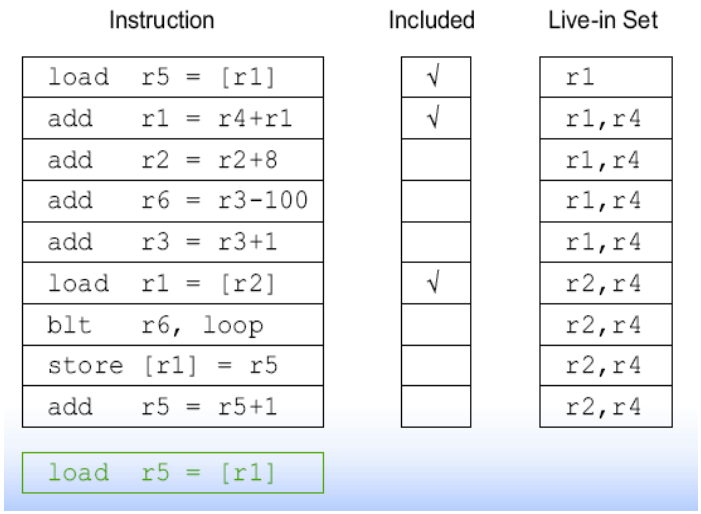


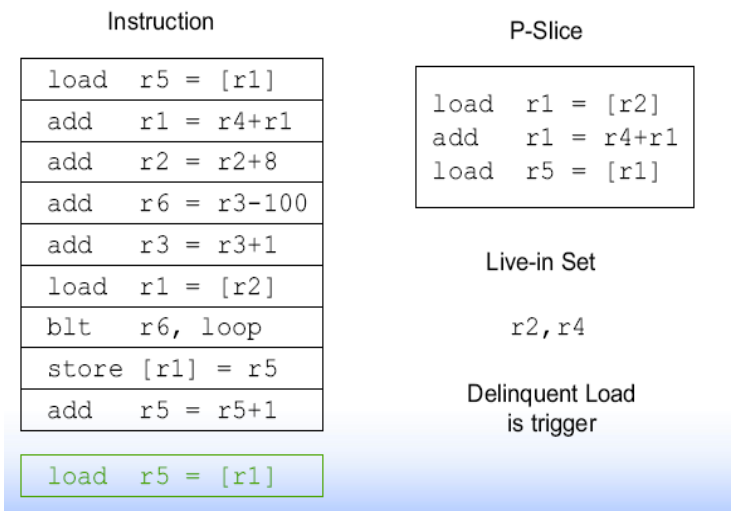




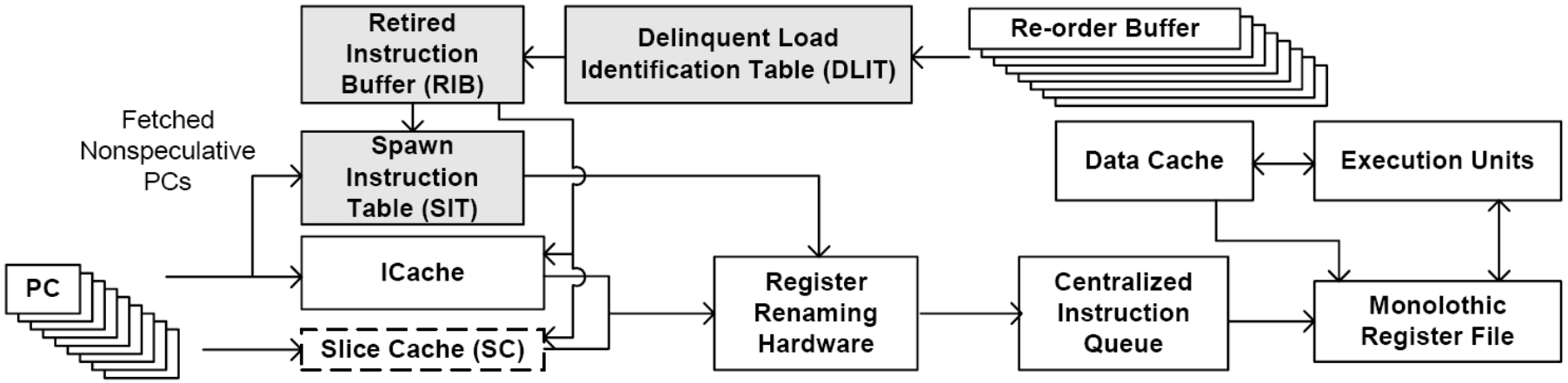
.....

Παράδειγμα κατασκευής P-slice



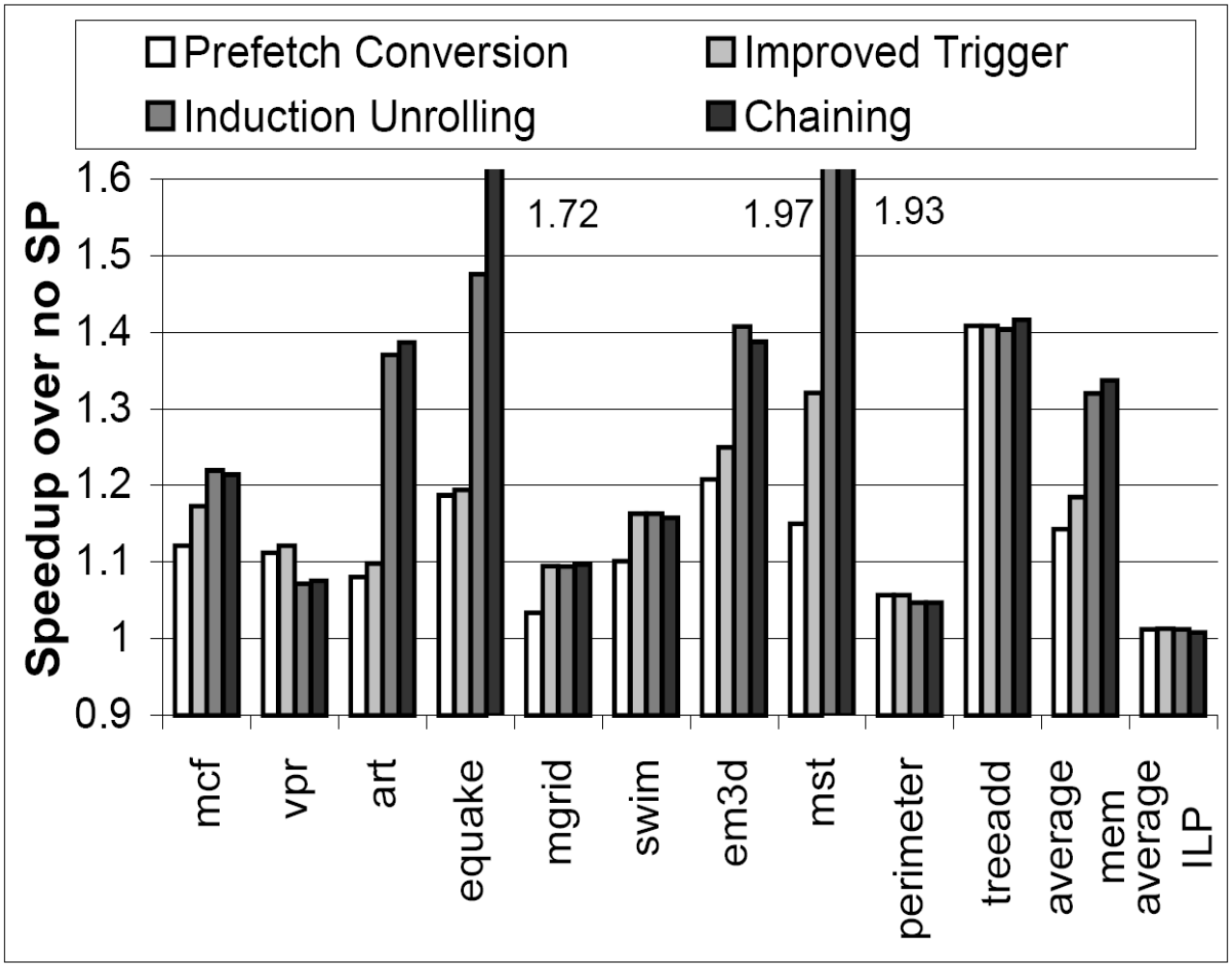


Τροποποιημένο SMT pipeline για DSP



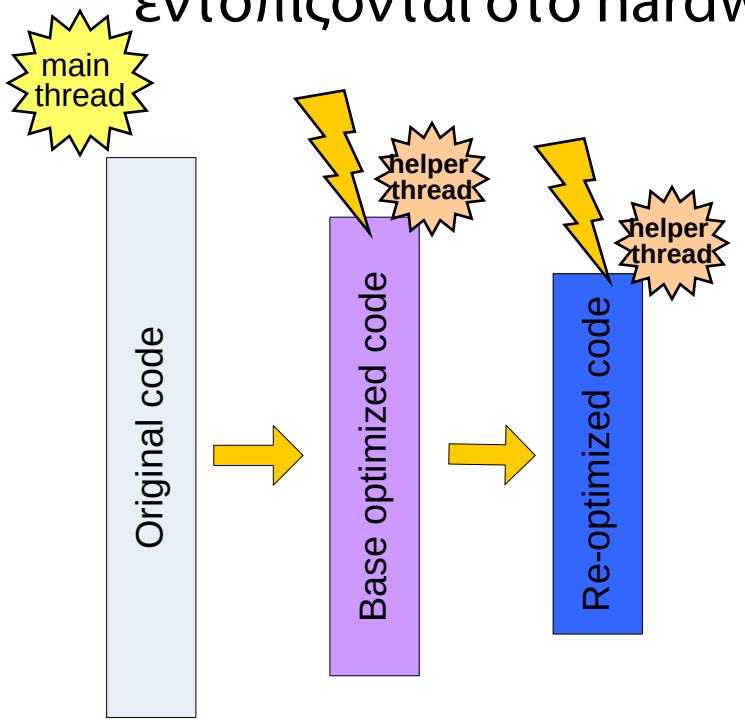
- Εκκίνηση και διαχείριση speculative threads
  - SIT: αποθηκεύει τα P-slices
  - όταν κατά την αποκωδικοποίηση εντολών του main thread διαπιστώνεται μια triggering εντολή, τότε, αν υπάρχει διαθέσιμο context:
    - » τα live-ins του αντίστοιχου P-slice αντιγράφονται από το main στο speculative thread (register renaming hardware),
    - » και το P-slice γίνεται spawn τον επόμενο κύκλο

DSP Performance



Event-driven dynamic optimization

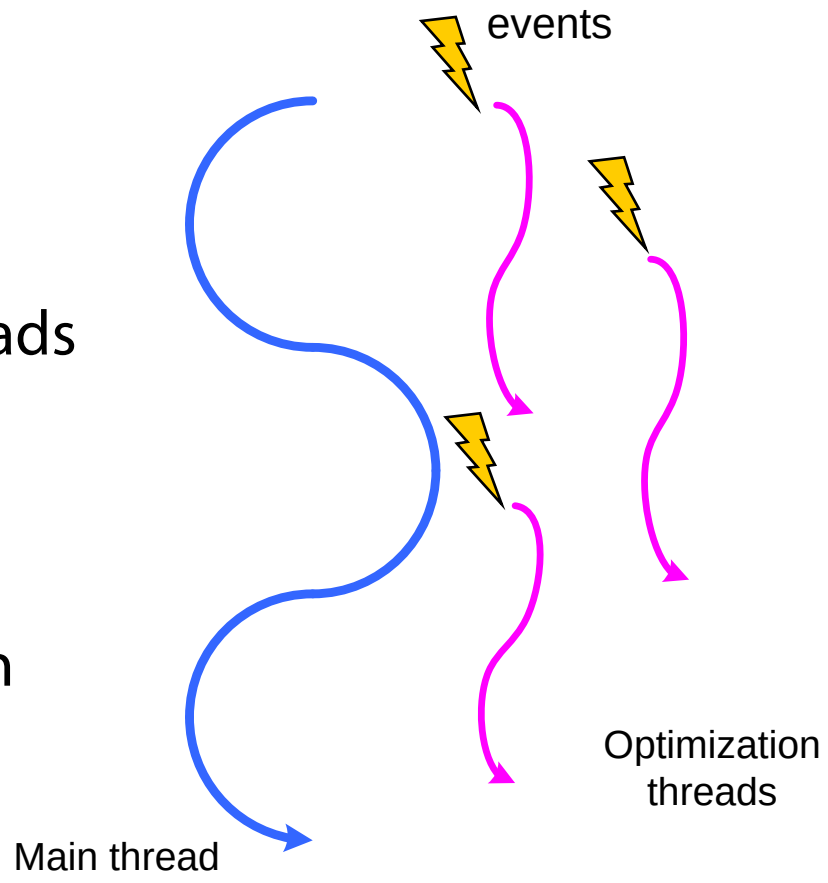
- Χρησιμοποίηση helper threads για επαναμεταγλώττιση / βελτιστοποίηση του main thread
  - η εκτέλεση των υπολογισμών του main thread και η βελτιστοποίηση από τα helper threads πραγματοποιούνται ***παράλληλα***
- Η βελτιστοποίηση πυροδοτείται από γεγονότα που εντοπίζονται στο hardware (event-driven)



• μία έκδοση του κώδικα, η οποία σταδιακά βελτιστοποιείται όταν παρατηρείται αλλαγή στη συμπεριφορά του προγράμματος κατά το χρόνο εκτέλεσης

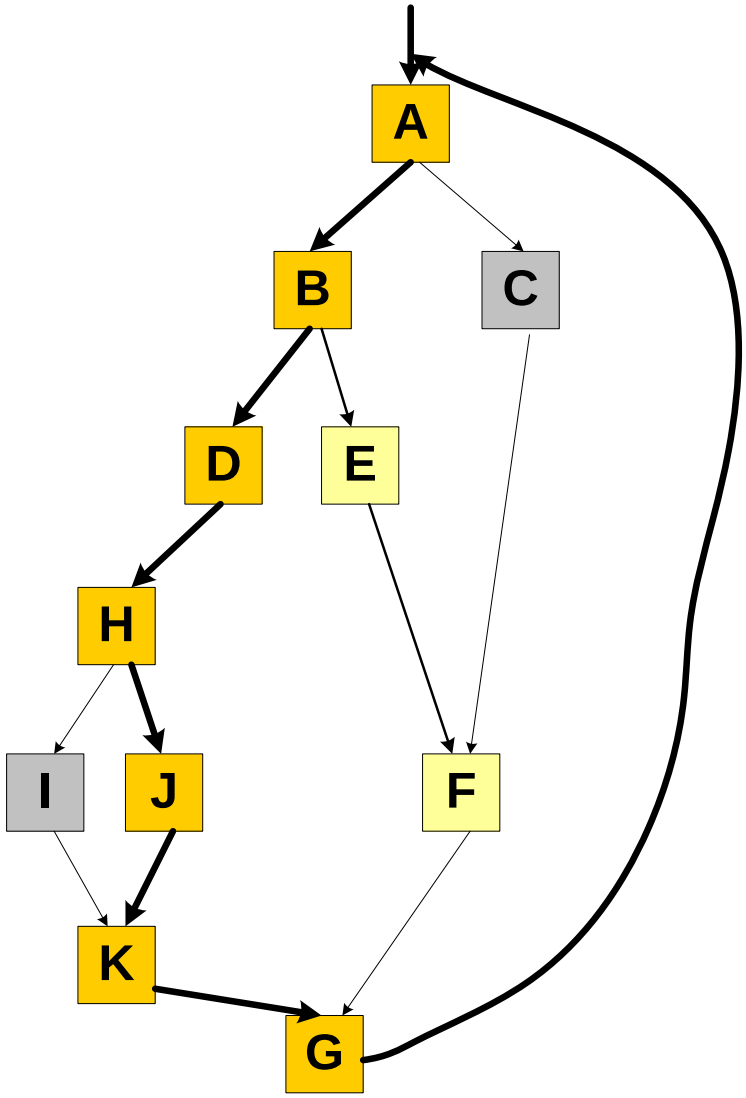
Event-driven dynamic optimization

- Το hardware παρατηρεί τη συμπεριφορά του προγράμματος χωρίς software overhead
- Πυροδοτούνται optimization threads σε «ανταπόκριση» συγκεκριμένων γεγονότων
- Βελτιστοποιούν όσο το δυνατόν συντομότερα τον κώδικα του main thread

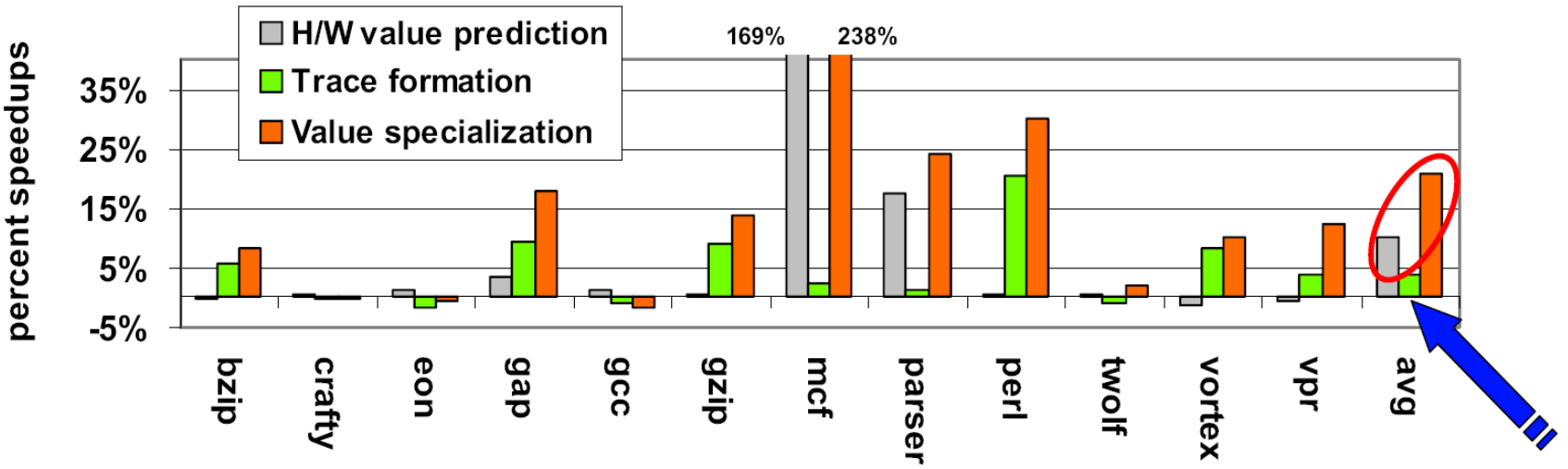


Παράδειγμα: hot paths

- Hot path: μια ακολουθία από basic blocks που εκτελούνται συχνά μαζί
- Path formation: αναδιοργάνωση των basic blocks και διάταξή τους σύμφωνα με τη σειρά που εκτελούνται στο hot path, για καλύτερη τοπικότητα στην I-cache
- Αποθήκευση του νέου κώδικα στην code-cache
- Events monitored: frequently executed branches



Performance of event-driven optimization



credits for slides:

- D. Tullsen (University of California, San Diego)
  - ACACES summer school 2008

Βιβλιογραφία

Software-based Helper Threading

- [Collins 01] J. Collins, H. Wang, D. Tullsen, C. Hughes, Y. Lee, D. Lavery & J. Shen. *Speculative Precomputation: Long-Range Prefetching of Delinquent Loads*. 28th Annual International Symposium on Computer Architecture (ISCA '01).
- [Kim 02] D. Kim & D. Yeung. *Design and Evaluation of Compiler Algorithms for Pre-Execution*. 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '02).
- [Kim 04] D. Kim, S. Liao, P. Wang, J. del Cuvillo, X. Tian, X. Zou, H. Wang, D. Yeung, M. Girkar & J. Shen. *Physical Experimentation with Prefetching Helper Threads on Intel's Hyper-Threaded Processors*. 2nd IEEE / ACM International Symposium on Code Generation and Optimization (CGO '04).
- [Luk 01] C. Luk. *Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors*. 28th Annual International Symposium on Computer Architecture (ISCA '01).
- [Song 05] Y. Song, S. Kalogeropoulos, P. Tirumalai. *"Design and Implementation of a Compiler Framework for Helper Threading on Multi-core Processors"* (PACT '05)
- [Wang 04] T. Wang, F. Blagojevic & D. Nikolopoulos. *Runtime Support for Integrating Precomputation and Thread-Level Parallelism on Simultaneous Multithreaded Processors*. 7th ACM SIGPLAN Workshop on Languages, Compilers, and Runtime Support for Scalable Systems (LCR '04).
- [Zhang05] W. Zhang, B. Calder and D. Tullsen. *An Event-Driven Multithreaded Dynamic Optimization Framework*. 14th International Conference on Parallel Architectures and Compilation Techniques (PACT '05)
- [Zhang06] W. Zhang, B. Calder and D. Tullsen. *A Self-Repairing Prefetcher in an Event-Driven Dynamic Optimization Framework*. (CGO '06)

Hardware-based Helper Threading

- [Chappell 99] R. Chappell, J. Stark, S. Kim, S. Reinhardt & Y. Patt. *Simultaneous Subordinate Microthreading (SSMT)*. 26th Annual International Symposium on Computer architecture (ISCA '99).
- [Collins 01] J. Collins, D. Tullsen, H. Wang, J. Shen. *Dynamic Speculative Precomputation*. 34th annual ACM/IEEE international symposium on Microarchitecture (MICRO '01)
- [Ibrahim 03] K. Ibrahim, G. Byrd, E. Rotenberg. *Slipstream Execution Mode for CMP-Based Multiprocessors*. (HPCA '03)
- [Roth 01] A. Roth & G. Sohi. *Speculative Data-Driven Multithreading*. 7th International Symposium on High Performance Computer Architecture (HPCA '01).
- [Sundaramoorthy 00] K. Sundaramoorthy, Z. Purser & E. Rotenberg. *Slipstream Processors: Improving both Performance and Fault Tolerance*. 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '00).
- [Zilles 01] C. Zilles & G. Sohi. *Execution-Based Prediction Using Speculative Slices*. 28th Annual International Symposium on Computer Architecture (ISCA '01).