Ιόνιο Πανεπιστήμιο – Τμήμα Πληροφορικής
Παράλληλος Προγραμματισμός
2025-26

OpenMP και εντολές SIMD

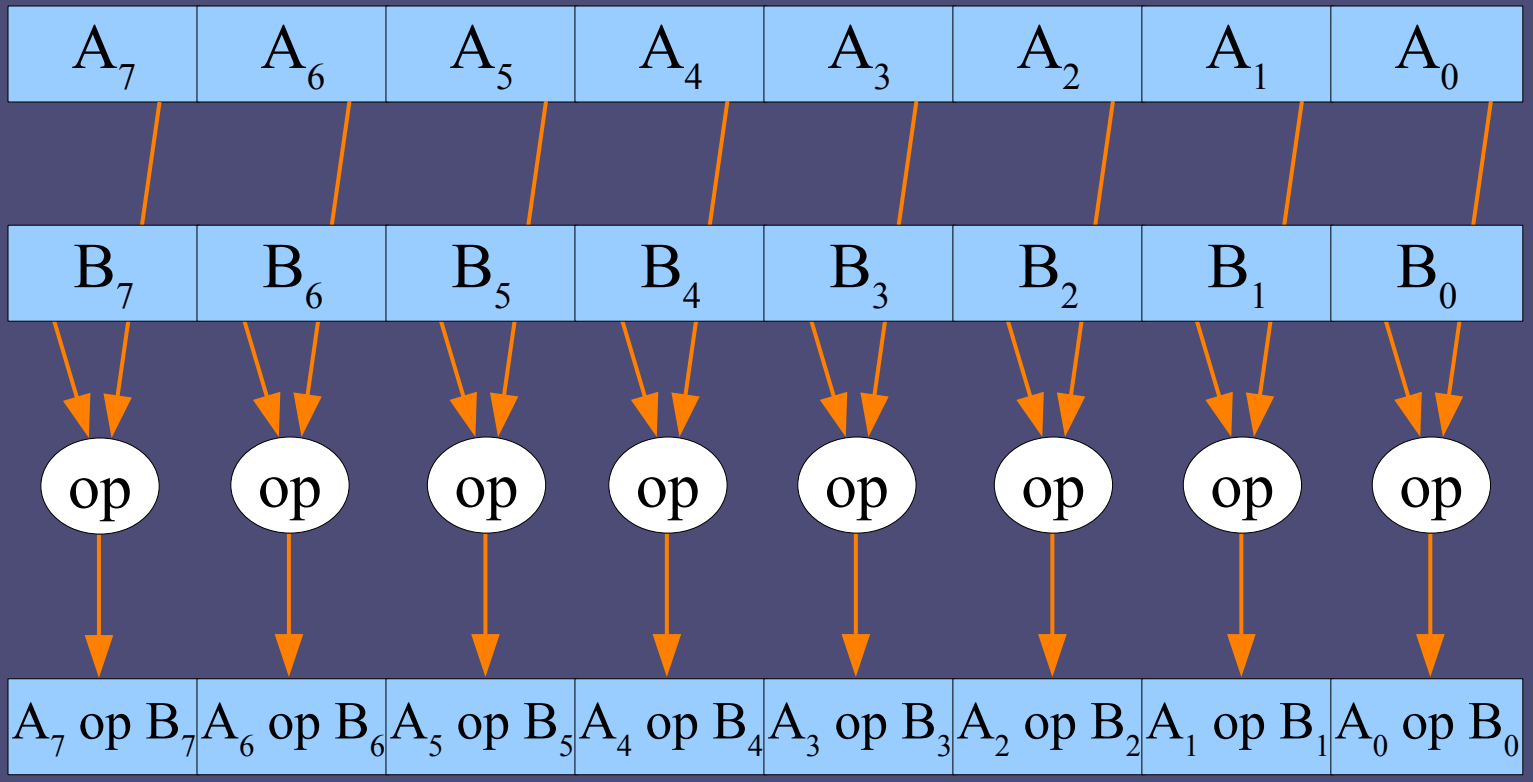
(Παραλληλισμός τύπου vector)

https://mixstef.github.io/courses/parprog/

Μ.Στεφανιδάκης

Επανάληψη: εντολές SSE/AVX

- dst[i] = a[i] op b[i]



Streaming SIMD Instructions

- Single Instruction Multiple Data (SIMD)
  - Λειτουργίες τύπου “vector”
  - Εντολές που εκτελούν την ίδια πράξη σε μια ομάδα «πακεταρισμένων» (packed) δεδομένων
  - ALU και καταχωρητές μεγάλου εύρους (128 – 512 bits)
    - Σε αντίθεση με τις πράξεις σε απλές (scalar) ποσότητες ενός int, ενός float, ενός double κλπ
  - Οι εντολές τύπου SIMD είναι κατάλληλες για παράλληλες λειτουργίες map
    - «Κάθετες» πράξεις
    - Έχουμε ήδη ασχοληθεί με SSE/AVX intrinsics

Παραδείγματα intrinsics SSE/AVX

- Αριθμητικές πράξεις
  - add, sub, addsub, mul, div, max, min
    - π.χ. __m256 _mm256_add_ps (__m256 a, __m256 b)
    - π.χ. __m256i _mm256_max_epi32 (__m256i a, __m256i b) αλλά και __m256i _mm256_max_epu32 (__m256i a, __m256i b)
  - sqrt, rsqrt, rcp, ceil, floor, round
    - π.χ. __m256d _mm256_sqrt_pd (__m256d a)
  - fmadd, fmsub, fmaddsub, fnmadd, fnmsub (`-mfma` flag)
    - π.χ. __m256 _mm256_fmadd_ps (__m256 a, __m256 b, __m256 c)
- Λογικές πράξεις
  - and, or, xor, andnot
    - π.χ. __m256 _mm256_and_ps (__m256 a, __m256 b)
    - π.χ. __m256i _mm256_and_si256 (__m256i a, __m256i b)

Διαφορετικοί τρόποι προγραμματισμού με εντολές SIMD

- Κατευθείαν σε assembly
  - C inline assembly → ο μεταγλωττιστής βοηθά σε μικρό βαθμό στην ανάθεση μεταβλητών σε καταχωρητές
- Intrinsics
  - Ευκολότερη λύση από την assembly, ο προγραμματισμός παραμένει δύσκολος
- OpenMP pragmas
  - Ρητή δήλωση στον μεταγλωττιστή των ευκαιριών χρήσης λειτουργιών vector
- Αυτόματη μετατροπή σειριακού κώδικα
  - Οι μεταγλωττιστές έχουν προοδεύσει πολύ στον τομέα αυτόν αλλά δεν μπορούν να ανακαλύψουν πάντα όλες τις ευκαιρίες παραλληλισμού SIMD

Εντολές SIMD και OpenMP

- #pragma omp simd

```
#pragma omp simd
for (int i=0;i<N;i++) {
  a[i] = b[i]+c[i];
}
```

- Κύριος τρόπος χρήσης εντολών SIMD
  - Εφαρμόζεται σε ένα for loop
  - Ισχύουν οι συνήθεις περιορισμοί για τη μορφή του loop (γνωστή και σταθερή αρχή/τέλος, συγκεκριμένοι τελεστές...)
- Δηλώνει ρητά ότι το περιεχόμενο του loop μπορεί να εκτελεστεί με εντολές SIMD
  - Στην περίπτωσή μας (αρχιτετονική x86) με εντολές SSE/AVX
- Δεν απαιτείται η ύπαρξη παράλληλης περιοχής

Ο κώδικας assemby που δημιουργείται

- Εκτέλεση με εντολές AVX σε καταχωρητές ymm (256 bits, 8 floats)

```
        xorl    %eax, %eax
.L13:
        vmovaps (%r14,%rax), %ymm4
        vaddps  0(%r13,%rax), %ymm4, %ymm0
        vmovaps %ymm0, (%r12,%rax)
        addq    $32, %rax
        cmpq    $40000, %rax
        jne     .L13
```

- Στο παράδειγμα που βλέπουμε εκτελούνται οι επαναλήψεις του for που μπορούν να εκτελεστούν κατά δάδες
- Το OpenMP θα φροντίσει να εκτελέσει τις υπόλοιπες επαναλήψεις που περισσεύουν χωρίς κώδικα SIMD:

```
.L14:   vmovss  (%r14,%rax), %xmm0
        vaddss  0(%r13,%rax), %xmm0, %xmm0
        vmovss  %xmm0, (%r12,%rax)
        addq    $4, %rax
        cmpq    $40016, %rax
        jne .L14
```

Ρυθμίσεις και υποδείξεις (1)

- Πρόσθετα clauses στο #pragma omp
  - Υποδεικνύουν στο OpenMP (= στον μεταγλωττιστή) τα χαρακτηριστικά και τον τρόπο χρήσης των μεταβλητών μέσα στο SIMD loop
    - Πολλά από αυτά ένας σύγχρονος μεταγλωττιστής μπορεί να τα συνάγει μόνος του
- simdlen(*length*)
  - Θετική ακέραια σταθερά που υποδηλώνει το επιθυμητό εύρος των vectors
    - Πόσες επαναλήψεις του loop θα γίνουν με μία εντολή SIMD
  - Προσοχή: το εύρος επιλέγεται τελικά από το OpenMP, ανάλογα με την αρχιτεκτονική – στόχο της μεταγλώττισης
    - Default τιμή ανάλογα με την αρχιτεκτονική – στόχο
    - Μην ξεχνάτε το -mavx αν θέλετε να επιλεγούν εντολές AVX

Ρυθμίσεις και υποδείξεις (2)

- safelen(*length*)
  - Οι επαναλήψεις που θα εκτελεστούν στην SIMD εντολή δεν μπορούν να απέχουν μεταξύ τους περισσότερο από length
    - Για την τήρηση αλληλοεξαρτήσεων
    - Π.χ. a[i] = a[i-4]+c[i] (safelen = 4 αντί 8)
- aligned(*list[: alignment]*)
  - Υποδεικνύει ότι οι μεταβλητές στο list έχουν ένα συγκεκριμένο alignment (bytes)

Ρυθμίσεις στη χρήση των μεταβλητών

- private(*list*) και lastprivate(*list*)
  - Κάθε επανάληψη έχει τη δική της μεταβλητή
    - «Πακέτο» μεταβλητών σε κάθε λειτουργία SIMD
    - Το lastprivate αντιγράφει προς τα έξω την τιμή της τελευταίας επανάληψης
- reduction(*reduction-identifier : list*)
  - Όπως το private, με οριζόντια άθροιση στο τέλος
- linear(*list[: linear-step]*)
  - Όπως το private, δηλώνει μια γραμμική σχέση με το i (index) του simd loop
    - μεταβλητές int ή δείκτες
    - Αν a στο list, a_i = a + i*linear-step
    - linear-step σταθερό σε όλο το simd loop, default = 1

Δήλωση συναρτήσεων SIMD

- Συναρτήσεις που καλούνται μέσα από το simd construct
  - Δημιουργία διαφορετικών εκδοχών για scalar και vector χρήση
  - #pragma omp declare simd
    - Πριν τη δήλωση της συνάρτησης
- Προαιρετικά clauses
  - Τα γνωστά simdlen, aligned, linear
    - Το simdlen υποδηλώνει το εύρος ορισμάτων και επιστροφής
- uniform(*argument-list*)
  - Τα ορίσματα στο argument-list έχουν σταθερή τιμή σε κάθε επανάληψη
- inbranch/notinbranch
  - Δηλώνει την κλήση υπό συνθήκη ή όχι της συνάρτησης
    - Αν, ναι παράγεται μια παραλλαγή της συνάρτησης με μάσκα

Συνδυασμός threading και εντολών SIMD

- #pragma omp for simd

```
#pragma omp parallel
{
  for (int j=0;j<R;j++) {

    #pragma omp for simd
    for (int i=0;i<N;i++) {
      a[i] = b[i]+c[i];
    }

  }
}
```

- Το έργο μοιράζεται σε όλα τα threads και κάθε thread εκτελεί εντολές SIMD
  - Πρέπει να υπάρχει παράλληλη περιοχή για να δημιουργήσει τα threads
  - Το simd construct μπορεί να υπάρχει και ανεξάρτητα μέσα σε ένα for construct