Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχ. και Μηχανικών Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Παρουσίαση 3ης Άσκησης:

Θέματα Συγχρονισμού σε Σύγχρονα Πολυπύρηνα Συστήματα

Ακ. Έτος 2019-2020

Συστήματα Παράλληλης Επεξεργασίας

9ο Εξάμηνο

National Technical University of Athens
CSLab

Posix Threads (Pthreads)

- Βιβλιοθήκη + σύστημα χρόνου εκτέλεσης (runtime)
  - για την διαχείριση (δημιουργία, εκτέλεση, τερματισμό) νημάτων
  - παρέχονται και δομές/ρουτίνες για τον συγχρονισμό των νημάτων
- `pthread_t`: αναγνωριστικό ενός νήματος
- Κυριότερες ρουτίνες διαχείρισης νημάτων
  - `pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)`
    - Δημιουργία ενός νήματος το οποίο θα εκτελέσει την συνάρτηση `void *start_routine(void *)` με όρισμα το `arg`
  - `pthread_join(pthread_t thread, void **retval)`
    - Αναμονή εώς τον τερματισμό του νήματος με το αναγνωριστικό `thread`. Η τιμή που επιστρέφεται από το νήμα αποθηκεύεται στο `*retval`.
- Κάποιες δομές/ρουτίνες συγχρονισμού
  - `pthread_mutex_t, pthread_mutex_init(…), pthread_mutex_lock(…), pthread_mutex_unlock(…), pthread_mutex_destroy(…)`
  - `pthread_spinlock_t, pthread_spin_init(…), pthread_spin_lock(…), pthread_spin_unlock(…), pthread_spin_destroy(…)`
  - `pthread_barrier_t, pthread_barrier_init(…), pthread_barrier_wait(…), pthread_barrier_destroy(…)`

Παράδειγμα Pthreads

```c
#include <stdio.h>
#include <pthread.h>

/* Global data. */
pthread_spinlock_t output_spinlock;
pthread_barrier_t  global_barrier;

/* The function executed by each thread. */
void *hello_tid(void *targ) {
    int myid = *((int *)targ);

    /* Wait until all threads are here. */
    pthread_barrier_wait(&global_barrier);

    /* Grab the lock and print thread's message. */
    pthread_spin_lock(&output_spinlock);
    printf("Hello, I am thread %d\n", myid);
    pthread_spin_unlock(&output_spinlock);

    return NULL;
}
```

Όλα τα νήματα μπαίνουν στο «κρίσιμο τμήμα» ταυτόχρονα

```c
#define NTHREADS 16
int main() {
    pthread_t threads[NTHREADS];
    int thread_ids[NTHREADS];

    /* Global data initializations. */
    pthread_spin_init(&output_spinlock, PTHREAD_PROCESS_SHARED);
    pthread_barrier_init(&global_barrier, NULL, NTHREADS);

    /* Create and spawn all threads. */
    for (i=0; i < nthreads; i++) {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL,
                       hello_tid, &thread_ids[i]);
    }

    /* Wait all threads to finish. */
    for (i=0; i < nthreads; i++)
        pthread_join(threads[i], NULL);

    pthread_spin_destroy(&output_spinlock);
    pthread_barrier_destroy(&global_barrier);

    return 0;
}
```

Δημιουργία των νημάτων

Εκτέλεση της συνάρτησης hello_tid()

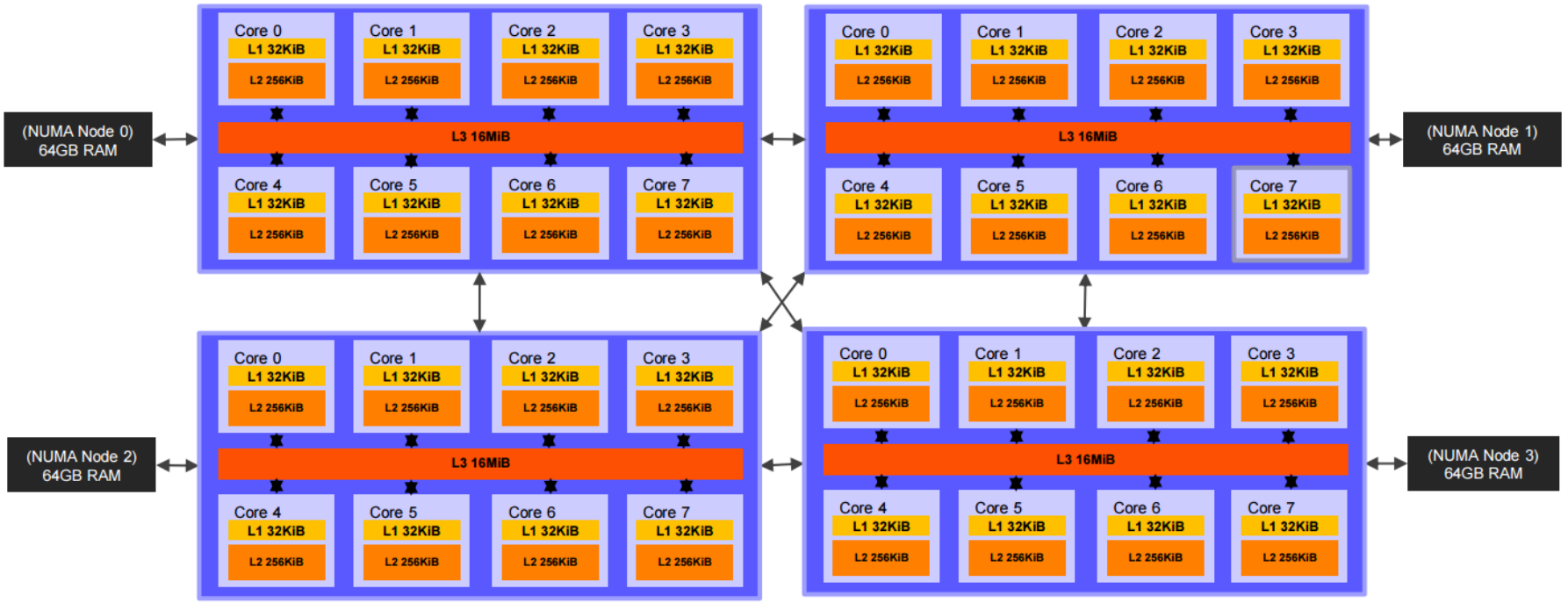
Αναμονή μέχρι να τερματίσουν τα νήματα

Μεταγλώττιση με:

```
$ gcc –Wall –Wextra -pthread –o threads threads.c
```

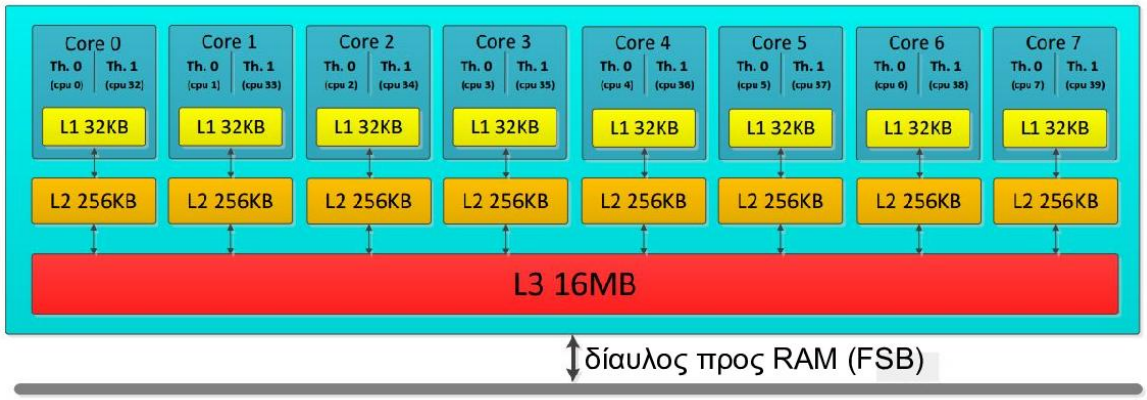
Περιβάλλον εκτέλεσης

- Sandman: 4 x Intel Xeon E5-4620 (Sandy Bridge 8-core/16-threads)
  - Συνολικά 32 πυρήνες και 64 threads (Hyperthreading)
  - NUMA (Non-Uniform Memory Access)



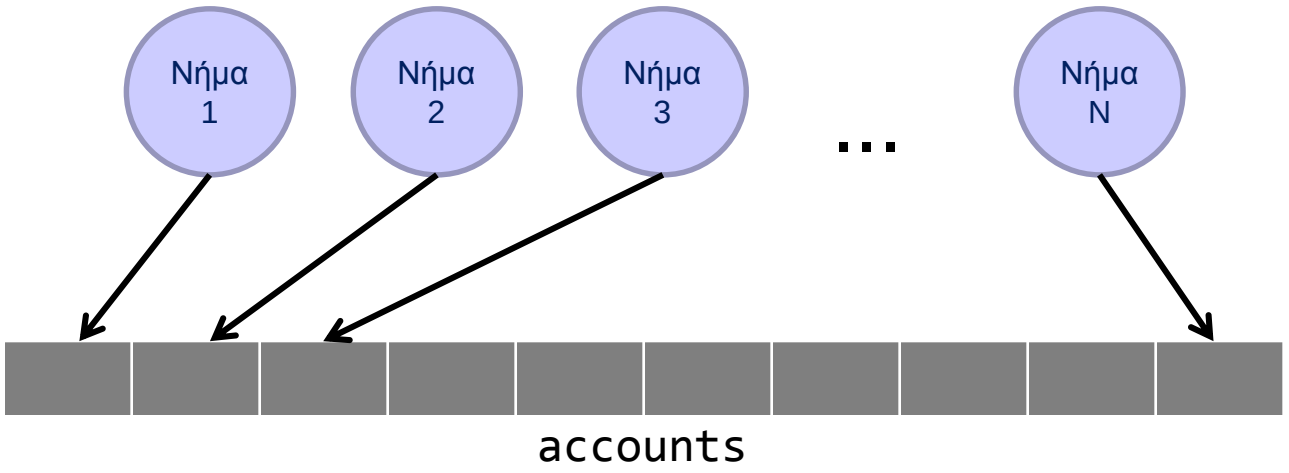
Thread affinity

- Ένα νήμα μπορεί να τοποθετηθεί σε οποιονδήποτε πυρήνα
  - είναι επιλογή του χρονοδρομολογητή του ΛΣ
- Μπορούμε να ορίσουμε συγκεκριμένο πυρήνα για κάθε νήμα με χρήση της `sched_setaffinity()`.
  - για τους σκοπούς της άσκησης παρέχεται η `setaffinity_oncpu(unsigned int cpu)` στο /home/parallel/pps/2019-2020/a3/common/aff.c
- Αρίθμηση των πυρήνων του sandman
  - socket0: 0-7, 32-39
  - socket1: 8-15, 40-47
  - socket2: 16-23, 48-55
  - socket3: 24-31, 56-63



Ζήτημα 1: Λογαριασμοί Τράπεζας

- Ένας πίνακας αντιπροσωπεύει τους λογαριασμούς των πελατών μιας τράπεζας
- Κάθε νήμα εκτελεί ένα σύνολο λειτουργιών σε ξεχωριστό λογαριασμό
- Υπάρχει ανάγκη συγχρονισμού;
- Ποια αναμένετε να είναι η συμπεριφορά του προγράμματος καθώς προστίθενται νήματα;
- Πώς επηρεάζει η τοποθέτηση των νημάτων στους πυρήνες του μηχανήματος την επίδοση της εφαρμογής;



Ζήτημα 2: Αμοιβαίος Αποκλεισμός - Κλειδώματα

- Προστασία κρίσιμου τμήματος με διαφορετικές υλοποιήσεις κλειδωμάτων
- **Κρίσιμο τμήμα:** αναζητήσεις τυχαίων κλειδιών σε ταξινομημένη συνδεδεμένη λίστα.
  - **read-only CS:** μας ενδιαφέρει απλά να εξετάσουμε το overhead που εισάγεται από κάθε υλοποίηση κλειδώματος
- Δίνονται:
  - tas lock: test and set
  - clh lock: queue lock
- Ζητούνται:
  - pthread lock: χρησιμοποιεί το pthread_spinlock_t των pthreads
  - ttas lock: test-and-test-and-set lock
  - array lock: lock βασισμένο σε πίνακα

Ζήτημα 3: Τακτικές συγχρονισμού για δομές δεδομένων

- Υλοποίηση ταξινομημένης συνδεδεμένης λίστας με χρήση διαφορετικών τεχνικών συγχρονισμού.
- Καλείστε να υλοποιήσετε:
  - fine-grain locking: hand-over-hand locking
  - optimistic
  - lazy
  - non-blocking
- **Λεπτό ζήτημα:** σε δομές όπως οι optimistic, lazy και non-blocking η διαχείριση μνήμης είναι δύσκολη. Για τους σκοπούς της άσκησης δεν χρειάζεται να ελευθερώνετε τους κόμβους της λίστας.
- Πειράματα
  - για διαφορετικά μεγέθη λίστας
  - για διαφορετικά workloads (αναλογία αναζητήσεων, εισαγωγών, διαγραφών)