Ιωάννης Ε. Βενέτης

ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

18/02/2017

POSIX Threads (I)

Τι είναι τα POSIX Threads

- Κατασκευαστές παράλληλων συστημάτων παρείχαν τις δικές τους υλοποιήσεις νημάτων
  - Διέφεραν σημαντικά μεταξύ τους
  - Δύσκολη η συγγραφή προγραμμάτων που να μπορούν να εκτελεστούν σε διάφορα συστήματα (portability)
- Απαραίτητη η προτυποποίηση των λειτουργιών νημάτων
  - Για συστήματα UNIX η διεπαφή ορίστηκε μέσω του προτύπου IEEE POSIX 1003.1c standard (1995).
  - Υλοποιήσεις του προτύπου είναι γνωστές ως POSIX Threads ή Pthreads.
  - Οι περισσότεροι κατασκευαστές παράλληλων συστημάτων παρέχουν υλοποίηση των POSIX Threads
    - Πιθανόν μαζί με δικές τους υλοποιήσεις νημάτων

Πλεονεκτήματα χρήσης νημάτων (1/2)

- Ταχύτητα δημιουργίας νημάτων
  - Δημιουργία 50000 διεργασιών ή νημάτων (σε sec)

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
Intel 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
Intel 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

Πλεονεκτήματα χρήσης νημάτων (2/2)

- Ταχύτητα ανταλλαγής δεδομένων

Platform	MPI Shared Memory Bandwidth (GB/sec)	Pthreads Memory-to-CPU Bandwidth (Worst Case) (GB/sec)
Intel 2.6 GHz Xeon E5-2670	4.5	51.2
Intel 2.8 GHz Xeon 5660	5.6	32
AMD 2.3 GHz Opteron	1.8	5.3
AMD 2.4 GHz Opteron	1.2	5.3
IBM 1.9 GHz POWER5 p5-575	4.1	16
IBM 1.5 GHz POWER4	2.1	4
Intel 2.4 GHz Xeon	0.3	4.3
Intel 1.4 GHz Itanium 2	1.8	6.4

Προγραμματισμός με POSIX Threads

- Συμπεριλαμβάνουμε το αρχείο `pthread.h`

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

- Κατά την μεταγλώττιση του προγράμματος προσθέτουμε την παράμετρο `–pthread`

  gcc –O3 –Wall –pthread –o my_prog my_prog.c

Τύποι δεδομένων POSIX Threads

- pthread_t
  - Περιγραφέας (handle) νήματος
- pthread_attr_t
  - Γνωρίσματα (attributes) νήματος
    - Joinable ή Detached
    - Μέγεθος στοίβας νήματος
    - Ορισμός αρχικής διεύθυνσης στοίβας
    - Μέγεθος προστασίας στοίβας (stack guard)
    - Παράμετροι χρονοπρογραμματισμού
    - …
  - Δίνοντας την τιμή NULL επιλέγονται οι εξ ορισμού τιμές για τα γνωρίσματα ενός νήματος

Δημιουργία νημάτων

- `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);`
  - `pthread_t *thread`
    - Περιγραφέας νήματος που θα δημιουργηθεί
  - `const pthread_attr_t *attr`
    - Γνωρίσματα νήματος που θα δημιουργηθεί
      - NULL για προκαθορισμένες τιμές γνωρισμάτων
  - `void *(*start_routine) (void *)`
    - Συνάρτηση που θα εκτελέσει το νήμα
      - Δέχεται μια παράμετρο τύπου void *
      - Επιστρέφει μια τιμή τύπου void *
  - `void *arg`
    - Παράμετρος της συνάρτησης start_routine
- Μην ξεχνάτε τους ελέγχους για πιθανά λάθη!

Παράδειγμα

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

void *printMessage(void *arg) {
   printf(“%s\n”, (char *)arg);

   return(NULL);
}

int main(int argc, char *argv[]) {
   pthread_t tid1, tid2;
   char *msg1 = “This is thread 1”;
   char *msg2 = “This is thread 2”;

   pthread_create(&tid1, NULL, printMessage, (void *)msg1);
   pthread_create(&tid2, NULL, printMessage, (void *)msg2);

   return(0);
}  /* main() ends here */
```

Παράδειγμα (συνέχεια)

- Υπάρχει πιθανότητα η συνάρτηση `main()` να τερματίσει πριν ξεκινήσει η εκτέλεση των νημάτων
  - Τερματίζει όλη η διεργασία
  - Τα δύο νήματα δεν θα εκτελεστούν ποτέ

Ένωση νημάτων

- `int pthread_join(pthread_t thread, void **retval);`
  - `pthread_t thread`
    - Περιγραφέας νήματος του οποίου αναμένεται ο τερματισμός
  - `void **retval`
    - Τιμή που επιστρέφεται από το νήμα

Παράδειγμα

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

void *printMessage(void *arg) {
    printf(“%s\n”, (char *)arg);

    return(NULL);
}

int main(int argc, char *argv[]) {
    pthread_t tid1, tid2;
    char *msg1 = “This is thread 1”;
    char *msg2 = “This is thread 2”;

    pthread_create(&tid1, NULL, printMessage, (void *)msg1);
    pthread_create(&tid2, NULL, printMessage, (void *)msg2);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    return(0);
}   /* main() ends here */
```

Τερματισμός ενός μόνο νήματος

- void pthread_exit(void *retval);
  - `void *retval`
    - Τιμή που επιστρέφει το νήμα
- Δεν τερματίζει όλη τη διεργασία αλλά μόνο το νήμα που την κάλεσε
- Μπορεί να κληθεί από οποιοδήποτε νήμα
  - Όχι μόνο τη συνάρτηση `main()`

Παράδειγμα

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

void *printMessage(void *arg) {
   printf("%s\n", (char *)arg);

   return(NULL);
}

int main(int argc, char *argv[]) {
   pthread_t tid1, tid2;
   char *msg1 = "This is thread 1";
   char *msg2 = "This is thread 2";

   pthread_create(&tid1, NULL, printMessage, (void *)msg1);
   pthread_create(&tid2, NULL, printMessage, (void *)msg2);

   pthread_exit(0);
}  /* main() ends here */
```

Δημιουργία περισσότερων νημάτων

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

void *printMessage(void *arg) {
    printf("Hello from a thread!\n");

    return(NULL);
}

int main(int argc, char *argv[]) {
    pthread_t tid[100];
    int i;

    for (i = 0; i < 100; i++) {
          pthread_create(&tid[i], NULL, printMessage, NULL);
    }

    for (i = 0; i < 100; i++) {
          pthread_join(tid[i], NULL);
    }

    return(0);
}  /* main() ends here */
```

Τι λάθος έχει αυτό το παράδειγμα;

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

void *printMessage(void *arg) {
    printf("Hello from a thread!\n");

    return(NULL);
}

int main(int argc, char *argv[]) {
    pthread_t tid[100];
    int i;

    for (i = 0; i < 100; i++) {
          pthread_create(&tid[i], NULL, printMessage, NULL);
          pthread_join(tid[i], NULL);
    }

    return(0);
}  /* main() ends here */
```

Παραμετροποίηση πλήθους νημάτων

- Συνήθως θέλουμε να ελέγξουμε την απόδοση μιας εφαρμογής με POSIX Threads για διαφορετικά πλήθη νημάτων
  - Περνάμε το πλήθος των νημάτων ως παράμετρο κατά την εκτέλεση της εφαρμογής

Παράδειγμα

```
int main(int argc, char *argv[]) {
    pthread_t *tid;
    int i, numOfThreads;

    if (argc != 2) {
          printf("Provide the number of threads to create\n");
          exit(0);
    }

    numOfThreads = atoi(argv[1]); /* Use of strtol() is more robust. */

    tid = (pthread_t *)malloc(numOfThreads * sizeof(pthread_t));
    if (tid == NULL) { /* Handle error. */ }

    for (i = 0; i < numOfThreads; i++) {
          pthread_create(&tid[i], NULL, printMessage, NULL);
    }

    for (i = 0; i < numOfThreads; i++) {
          pthread_join(tid[i], NULL);
    }

    return(0);
}  /* main() ends here */
```

Παράδειγμα (Μεταγλώττιση/Εκτέλεση)

```
$ gcc –O3 –Wall –pthread –o my_prog my_prog.c
$ ./my_prog
  Provide the number of threads to create.
$ ./my_prog 2
  Hello from a thread!
  Hello from a thread!
$ ./my_prog 4
  Hello from a thread!
  Hello from a thread!
  Hello from a thread!
  Hello from a thread!
$
```

Πέρασμα παραμέτρων στη συνάρτηση νήματος

- Η συνάρτηση που θα εκτελέσει το νήμα παίρνει μια παράμετρο τύπου `void *`
  - Μπορούμε να μετατρέψουμε (cast) οποιονδήποτε τύπο δεδομένων σε `void *`
  - Συνήθως περνάμε έναν ακέραιο αριθμό που δηλώνει το ID του νήματος
- Αν χρειαζόμαστε περισσότερες παραμέτρους;
  - Ορίζουμε μια νέα δομή (struct) με πεδία τις παραμέτρους που χρειαζόμαστε
  - Περνάμε ως παράμετρο δείκτη προς την δομή

Παράδειγμα

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

void *printMessage(void *arg) {
    int id = (int)arg;
    printf("Hello from thread %d!\n", id);

    return(NULL);
}

int main(int argc, char *argv[]) {
    pthread_t tid[100];
    int i;

    for (i = 0; i < 100; i++) {
          pthread_create(&tid[i], NULL, printMessage, (void *)i);
    }

    for (i = 0; i < 100; i++) {
          pthread_join(tid[i], NULL);
    }

    return(0);
}  /* main() ends here */
```

Τι λάθος έχει αυτό το παράδειγμα;

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

void *printMessage(void *arg) {
    int *p_to_id = (int *)arg;
    int id = *p_to_id;
    printf("Hello from thread %d!\n", id);

    return(NULL);
}

int main(int argc, char *argv[]) {
    pthread_t tid[100];
    int i;

    for (i = 0; i < 100; i++) {
          pthread_create(&tid[i], NULL, printMessage, (void *)(&i));
    }

    for (i = 0; i < 100; i++) {
          pthread_join(tid[i], NULL);
    }

    return(0);
}   /* main() ends here */
```

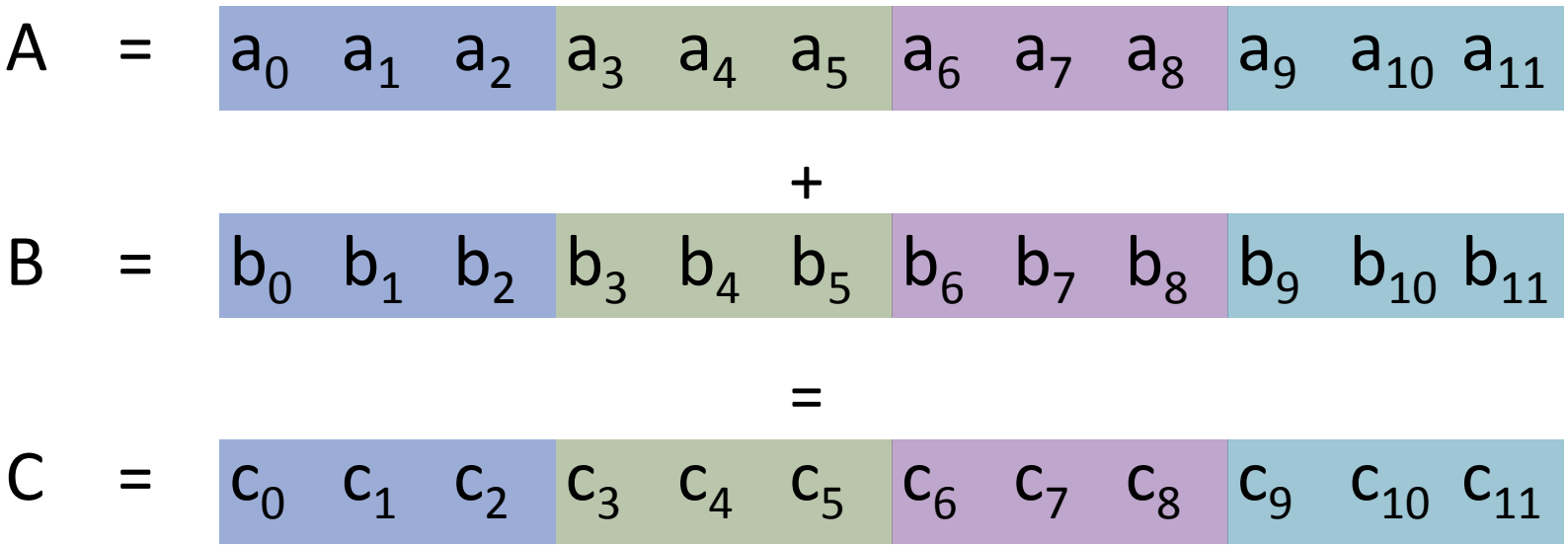
Πιθανή εκτέλεση

Χρόνος	main()	Νήμα 0	Νήμα 1
T_0	i = 0	---	---
T_1	create(&i)	---	---
T_2	i++ (i == 1)	Έναρξη εκτέλεσης	---
T_3	create(&i)	p_to_id = (int *)arg id = *p_to_id	---
T_4	i++ (i == 2)	printf(id) (== 1)	Έναρξη εκτέλεσης
T_5	join	exit	p_to_id = (int *)arg id = *p_to_id
T_6	join		printf(id) (== 2)

- Τυπώνεται 1 και 2, αντί για 0 και 1
  - Δεν περνάμε ως παράμετρο ποτέ δείκτη σε μεταβλητή που χρησιμοποιείται από περισσότερα νήματα!

Διαμοιρασμός φόρτου εργασίας (1/2)

- Πρόσθεση διανυσμάτων
  - Μέγεθος διανυσμάτων N = 12
  - numOfThreads = 4
  - Πλήθος στοιχείων ανά νήμα
    - numOfElements = N / numOfThreads (== 3)

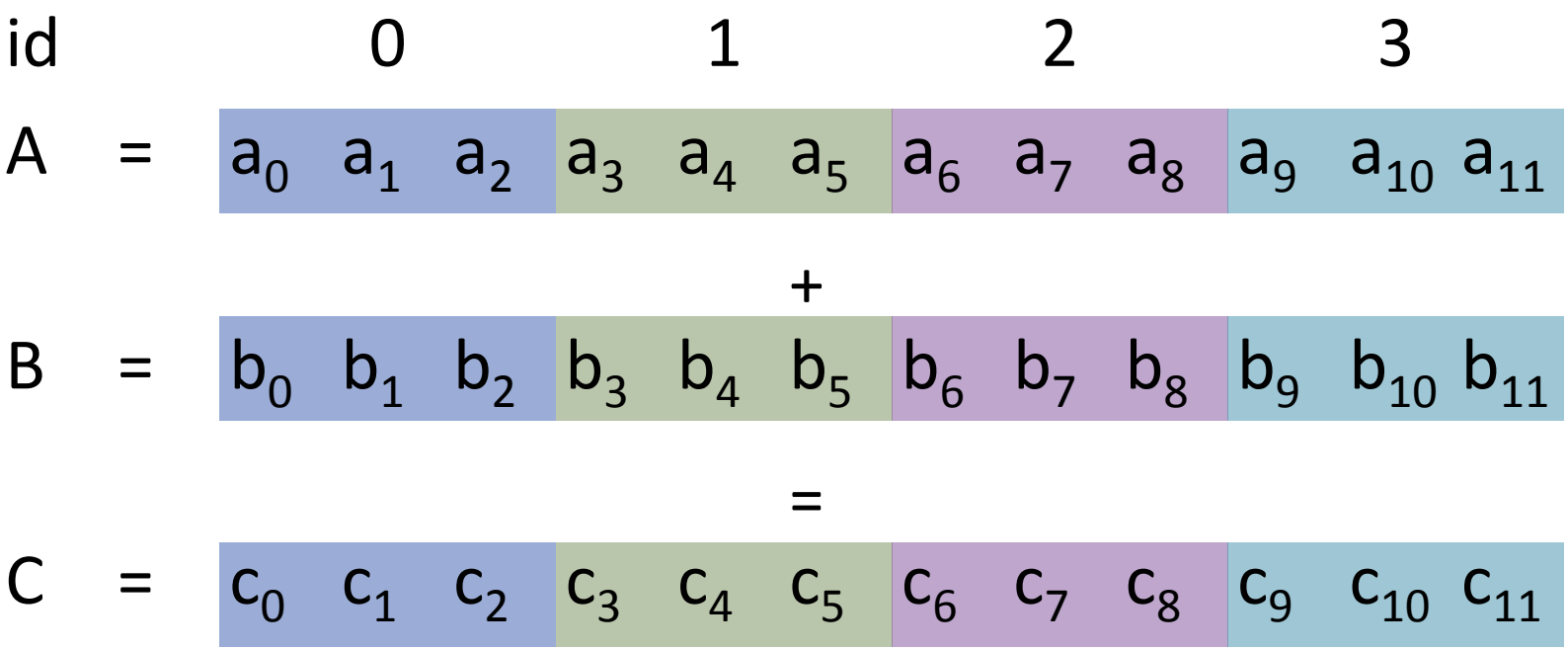


Διαμοιρασμός φόρτου εργασίας (2/2)

- Πως υπολογίζω ποια στοιχεία θα πάρει κάθε νήμα;
  - Λαμβάνω υπόψη το id κάθε νήματος

    start = numOfElements * id;

    end = start + numOfElements;



Πρόσθεση διανυσμάτων (1/3)

```
int *A, *B, *C;      /* Why must these be global? */
int numOfThreads, N; /* Why must these be global? */

int main(int argc, char *argv[]) {
    pthread_t *tid;
    int i;

    if (argc != 3) {
          printf("Provide the number of threads to create and the problem size.\n");
          exit(0);
    }

    numOfThreads = atoi(argv[1]); /* Use of strtol() is more robust. */
    N = atoi(argv[2]);

    tid = (pthread_t *)malloc(numOfThreads * sizeof(pthread_t));
    if (tid == NULL) {
          printf("Could not allocate memory.\n");
          exit(0);
    }
```

Πρόσθεση διανυσμάτων (2/3)

```
    A = (int *)malloc(N * sizeof(int));
    B = (int *)malloc(N * sizeof(int));
    C = (int *)malloc(N * sizeof(int));
    if ((A == NULL) || (B == NULL) || (C == NULL)) {
          printf("Could not allocate memory.\n");
          exit(0);
    }

    for (i = 0; i < numOfThreads; i++) {
          pthread_create(&tid[i], NULL, vector_add, (void *)i);
    }

    for (i = 0; i < numOfThreads; i++) {
          pthread_join(tid[i], NULL);
    }

    printf("C = ");
    for (i = 0; i < N; i++) {
          printf("%d ", C[i]);
    }

    return(0);
}   /* main() ends here */
```

Πρόσθεση διανυσμάτων (3/3)

```
void *vector_add(void *arg)
{
   int i, start, end, numOfElements, id = (int)arg;

   numOfElements = N / numOfThreads;
   start = numOfElements * id;
   end = start + numOfElements;

   for (i = start; i < end; i++) {
         C[i] = A[i] + B[i];
   }

   return(NULL);
}
```

Παράδειγμα (Μεταγλώττιση/Εκτέλεση)

$ gcc –O3 –Wall –pthread –o my_prog my_prog.c

$ ./my_prog 4 12

C = …

$

Πλήθος νημάτων

Μέγεθος μητρώου

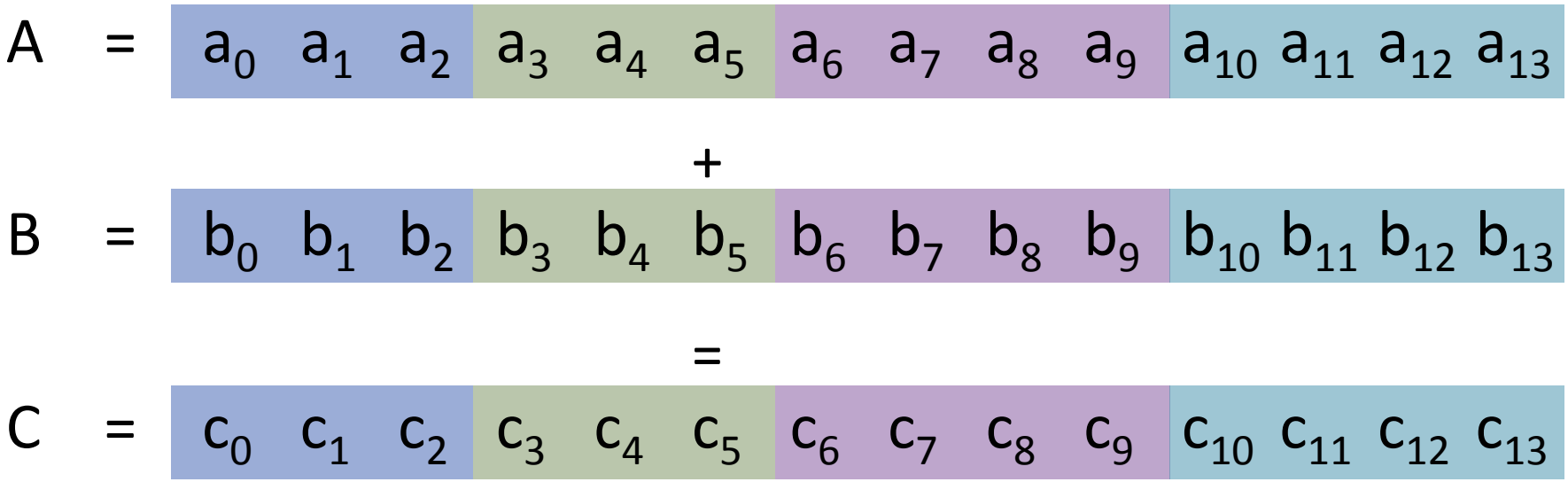
Πρόβλημα

- Και αν N / numOfThreads δεν διαιρείται ακριβώς;
  - Μέγεθος διανυσμάτων N = 14
  - numOfThreads = 4

$$
\begin{array}{lcccccccccccccc}
A & = & a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} & a_{11} & a_{12} & a_{13} \\
 & & & & & & & + & & & & & & & & \\
B & = & b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & b_{10} & b_{11} & b_{12} & b_{13} \\
 & & & & & & & = & & & & & & & & \\
C & = & c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} & c_{13}
\end{array}
$$

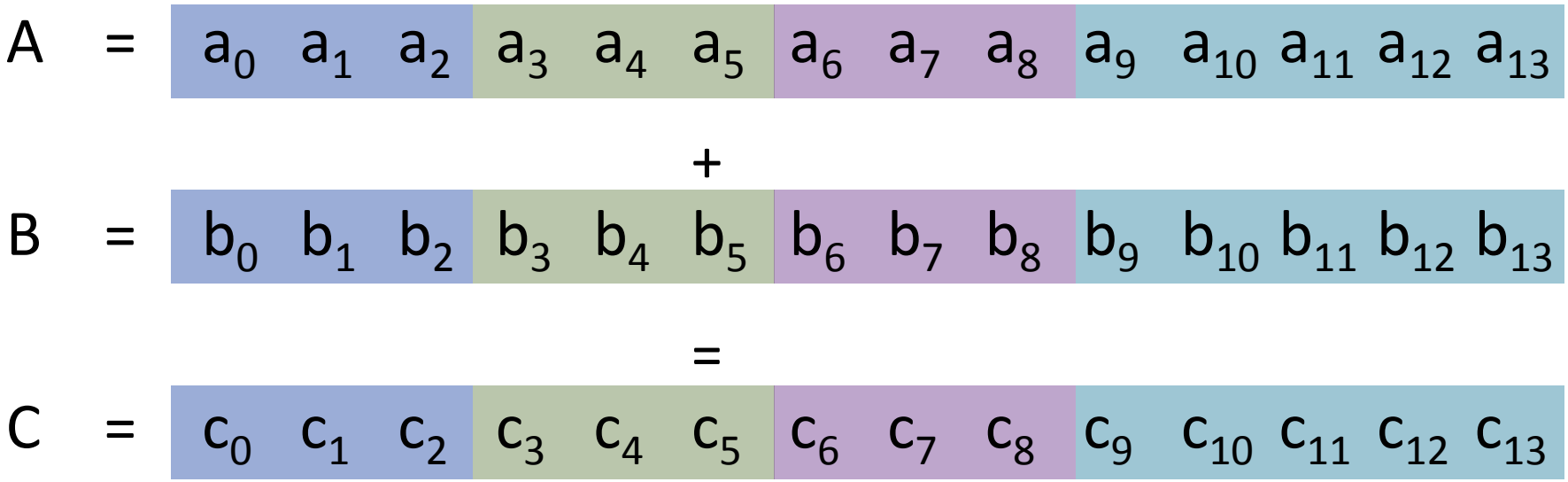
Η καλή (αλλά δύσκολη) λύση

- Υπολόγισε: `remainder = N % numOfThreads;`
  - Πλήθος στοιχείων που περισσεύουν
  - Δώσε ένα στοιχείο επιπλέον σε `remainder` νήματα
- Δυσκολεύει αρκετά ο υπολογισμός των start, end
  - Δοκιμάστε το με numOfThreads = 8



Η κακή (αλλά εύκολη) λύση

- Δώσε τα στοιχεία που περισσεύουν στο τελευταίο νήμα
  - Μπορεί να δημιουργήσει σημαντική ανισοκατανομή στο φόρτο εργασίας
    - Σκεφτείτε N = 999999, numOfThreads = 1000
- Θα χρησιμοποιήσουμε αυτή τη λύση μόνο για λόγους απλότητας



Πρόσθεση διανυσμάτων (1/3)

```
int *A, *B, *C;      /* Why must these be global? */
int numOfThreads, N; /* Why must these be global? */

int main(int argc, char *argv[]) {
   pthread_t *tid;
   int i;

   if (argc != 2) {
         printf("Provide the number of threads to create and the problem size.\n");
         exit(0);
   }

   numOfThreads = atoi(argv[1]); /* Use of strtol() is more robust. */
   N = atoi(argv[2]);

   tid = (pthread_t *)malloc(numOfThreads * sizeof(pthread_t));
   if (tid == NULL) {
         printf("Could not allocate memory.\n");
         exit(0);
   }
```

Πρόσθεση διανυσμάτων (2/3)

```
    A = (int *)malloc(N * sizeof(int));
    B = (int *)malloc(N * sizeof(int));
    C = (int *)malloc(N * sizeof(int));
    if ((A == NULL) || (B == NULL) || (C == NULL)) {
          printf("Could not allocate memory.\n");
          exit(0);
    }

    for (i = 0; i < numOfThreads; i++) {
          pthread_create(&tid[i], NULL, vector_add, (void *)i);
    }

    for (i = 0; i < numOfThreads; i++) {
          pthread_join(tid[i], NULL);
    }

    printf("C = ");
    for (i = 0; i < N; i++) {
          printf("%d ", C[i]);
    }

    return(0);
} /* main() ends here */
```

Πρόσθεση διανυσμάτων (3/3)

```
void *vector_add(void *arg)
{
    int i, start, end, numOfElements, id = *((int *)arg);

    numOfElements = N / numOfThreads;
    start = numOfElements * id;

    if (id != numOfThreads – 1) {
          end = start + numOfElements;
    } else {
          end = N;
    }

    for (i = start; i < end; i++) {
          C[i] = A[i] + B[i];
    }

    return(NULL);
}
```

Αμοιβαίος αποκλεισμός

- pthread_mutex_t
  - Μεταβλητή αμοιβαίου αποκλεισμού (mutex)
- pthread_mutex_attr_t
  - Γνωρίσματα μεταβλητής αμοιβαίου αποκλεισμού
    - Πρωτόκολλο για την αποφυγή “αναστροφής προτεραιότητας” (priority inversions)
    - Πρωτόκολλο priority ceiling
    - Διαμοιρασμός εντός μιας διεργασίας
  - Δίνοντας την τιμή NULL επιλέγονται οι εξ ορισμού τιμές για τα γνωρίσματα μιας μεταβλητής αμοιβαίου αποκλεισμού

Αρχικοποίηση mutex

- Ανάθεση
  - `pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;`
    - Αναθέτει εξ ορισμού τιμές στα γνωρίσματα της μεταβλητής
- Κλήση σε συνάρτηση
  - int pthread_mutex_init(pthread_mutex_t *mutex,
    const pthread_mutexattr_t *attr);
    - `pthread_mutex_t *mutex`
      - Περιγραφέας μεταβλητής που αρχικοποιείται
    - pthread_mutexattr_t *attr
      - Γνωρίσματα μεταβλητής που αρχικοποιείται
        - NULL για προκαθορισμένες τιμές γνωρισμάτων

Κλείδωμα/Ξεκλείδωμα

- int pthread_mutex_lock(pthread_mutex_t *mutex);
  - Κλείδωμα μεταβλητής αμοιβαίου αποκλεισμού
- int pthread_mutex_unlock(pthread_mutex_t *mutex);
  - Ξεκλείδωμα μεταβλητής αμοιβαίου αποκλεισμού
- int pthread_mutex_trylock(pthread_mutex_t *mutex);
  - Προσπάθεια κλειδώματος
    - Συνήθως μπαίνει μέσα σε loop
    - Αν επιτύχει (μεταβλητή ξεκλείδωτη) τότε συμπεριφέρεται όπως η `pthread_mutex_lock()`
    - Αν αποτύχει (μεταβλητή ήδη κλειδωμένη) επιστρέφει αμέσως
      - Μπορούμε να εκτελέσουμε άλλα τμήματα κώδικα
      - Δοκιμάζουμε αργότερα πάλι

Καταστροφή μεταβλητής αμοιβαίου αποκλεισμού

- `int pthread_mutex_destroy(pthread_mutex_t *mutex);`
  - Δεν καταστρέφει πραγματικά την μεταβλητή αμοιβαίου αποκλεισμού
  - Θέτει μη αποδεκτή τιμή
    (φαίνεται ως μη αρχικοποιημένη)

Πολλαπλασιασμός διανυσμάτων

Πολλαπλασιασμός διανυσμάτων (1/3)

```
int *A, *B, res = 0;
int numOfThreads, N;
pthread_mutex_t mut; /* Why must this be global? */

int main(int argc, char *argv[]) {
    pthread_t *tid;
    int i;

    if (argc != 3) {
          printf("Provide the number of threads to create and the problem size.\n");
          exit(0);
    }

    numOfThreads = atoi(argv[1]); /* Use of strtol() is more robust. */
    N = atoi(argv[2]);

    tid = (pthread_t *)malloc(numOfThreads * sizeof(pthread_t));
    if (tid == NULL) {
          printf("Could not allocate memory.\n");
          exit(0);
    }
```

Πολλαπλασιασμός διανυσμάτων (2/3)

```
    A = (int *)malloc(N * sizeof(int));
    B = (int *)malloc(N * sizeof(int));
    if ((A == NULL) || (B == NULL)) {
          printf("Could not allocate memory.\n");
          exit(0);
    }

    pthread_mutex_init(&mut, NULL);

    for (i = 0; i < numOfThreads; i++) {
          pthread_create(&tid[i], NULL, dot_product, (void *)i);
    }

    for (i = 0; i < numOfThreads; i++) {
          pthread_join(tid[i], NULL);
    }

    printf("Result is %d\n", res);

    return(0);
} /* main() ends here */
```

Πολλαπλασιασμός διανυσμάτων (3/3)

```
void *dot_product(void *arg)
{
    int i, start, end, numOfElements, id = *((int *)arg);

    numOfElements = N / numOfThreads;
    start = numOfElements * id;

    if (id != numOfThreads – 1) {
          end = start + numOfElements;
    } else {
          end = N;
    }

    for (i = start; i < end; i++) {
          pthread_mutex_lock(&mut);
          res += (A[i] * B[i]);
          pthread_mutex_unlock(&mut);
    }

    return(NULL);
}
```

Παράδειγμα (Μεταγλώττιση/Εκτέλεση)

```
$ gcc –O3 –Wall –pthread –o my_prog my_prog.c
$ ./my_prog  4  12
  Result is …
$
```