Ütemterv

Párhuzamos algoritmusok c. tárgyhoz **(GEMAK 243-B)**
BSc programtervező informatikus alapszak számára
Óraszám: heti 2+2, (aláírás+kollokvium, 5 kredit)
2019/20-es tanév II. félév.

Előfeltétel: legalább elégséges jegy Szoftvertechnológia (GEIAL 314-B) tárgyból

1. hét Alapfogalmak, hatékonysági mértékek, pszeudokód.

2. hét Számítási modellek. Rekurzió, véletlenített algoritmusok.

3. hét Párhuzamos gépek: alapvető módszerek.

4. hét Prefixszámítás.

5. hét Tömb elemeinek rangsorolása.

6. hét Kiválasztás. Összefésülés.

7. hét *1. zárthelyi dolgozat megírása.*

8. hét Rendezés. Rácsok: számítási modellek, csomagirányítás.

9. hét Alapfeladatok.

10. hét Kiválasztás, összefésülés, rendezés.

11. hét Hiperkocka: alapvető algoritmusok.

12. hét Szinkronizált hálózat: számítási modellek, vezetőválasztás.

13. hét *2. zárthelyi dolgozat megírása.*

14. hét Pótzárthelyi dolgozat megírása.

A tárgy lezárásának módja: aláírás, kollokvium

Az aláírás feltétele:

- Az előadások felkészült, rendszeres látogatása. Legfeljebb az előadások 40%-áról (azaz 6 alkalommal) lehet hiányozni. Ellenkező esetben a hallgató nem szerezhet aláírást!
- A gyakorlatokon való részvétel, legfeljebb a gyakorlatok 30%-áról (azaz 4 alkalommal) lehet hiányozni. Ellenkező esetben a hallgató nem szerezhet aláírást!
- A félév során a két zárthelyi dolgozat (7. és 13. héten) legalább elégséges szintű (6 pontból legalább 3 pont) megírása. A zárthelyi dolgozatok megírása mindenki számára kötelező. A zárthelyi dolgozat elméleti kérdéseket (tételeket, definíciókat, algoritmusokat) illetve gyakorlati feladatokat tartalmaz.
- Ha valamelyik zárthelyi dolgozat nem sikeres, akkor azt az utolsó héten lehet pótolni a megfelelő tananyagrészekből. Ha ez sem sikeres, akkor a későbbiekben az egész féléves anyagból kell pótolni.

A vizsga írásbeli. A vizsga 90 perces és 8 pontot lehet maximálisan megszerezni (azaz összesen 8 feladat beugró nélkül és minden feladat tökéletes megoldása 1 pontot ér). A vizsga során számonkérésre kerülnek pl. az alapalgoritmusok, melyek beugrónak számítanak a vizsgán, azaz ezek teljesítése kötelező a legalább elégséges jegy megszerzéséhez. Az elégséges jegy 4 ponttól van meg. Minden újabb pont megszerzése lényegében egy jeggyel javítja a vizsgajegyet, azaz a pontok alapján a jegyek kiosztása a következő: 0-3p elégtelen(1); 4p elégséges(2); 5p (közepes); 6p (jó); 7-8p jeles(5) az eredmény.

Meg nem engedett eszközök használata esetén a vizsga elégtelen és további vizsga abban a vizsgaidőszakban csak szóban, bizottság előtt, a tanszék által megadott időpontban lehetséges.

Miskolc, 2019. szeptember 2. Dr. Olajos Péter (a tárgy jegyzője)

Párhuzamos algoritmusok, 1. zh., MINTA

1. Ismertesse a Multi-Pascal környezetben a *ranks* párhuzamos programot, amely 10 egész számot nagyság szerinti növekedő sorrendbe rendez, ahol a bemenő adatok között lehetnek azonosak is! (1 pont)

```
PROGRAM Ranksort;
CONST n=10;
VAR values, final: ARRAY [1..n] OF INTEGER;
i: integer;
PROCEDURE PutinPlace( src: INTEGER);
VAR testval, j, rank, count: INTEGER;
BEGIN
        testval :=values[src];
        rank :=1;
        count :=0;
        FOR j :=1 TO n DO
        BEGIN
                IF testval > values[j] THEN rank :=rank+1;
                IF (testval = values[j]) AND (j < src) THEN count :=count+1;
        END;
        rank := rank+count;
        final[rank] :=testval; (*put value into its sorted position*)
END;
BEGIN
FOR i :=1 TO n DO
        Readln( values[i]); (*initialize values to be sorted*)
FORALL i :=1 TO n DO
        PutinPlace(i); (*find rank of values[i] and put in position*)
FOR i :=1 TO n DO
        Writeln( final[i]);
END.
```

2. Fogalmazza meg Brent tételét! (1 pont)

Tétel. (Brent tétele) *Ha egy feladatot a $\mathcal{P}$ párhuzamos algoritmus p processzoron t lépésben old meg, eközben az i-edik ($i = 1, 2, \ldots, t$) lépésben x_i műveletet végez, akkor ez a feladat $q < p$ processzoron megoldható $t + \lceil x/q \rceil$ lépésben, ahol $x = \sum_{i=1}^{t} x_i$.*

3. Ismertesse a prefixszámítás EREW-PREFIX algoritmusát EREW PRAM modellen! Milyen lépésszám jellemzi ezt az algoritmust? Munkaoptimális-e ez az algoritmus, válaszát indokolja! (1+1 pont)

EREW-PREFIX(p, X, Y) **párhuzamos eljárás**
Számítási modell: EREW PRAM
Input: p (a bemenet hossza) és $X[1:p] = \langle x_1, x_2, \ldots, x_p \rangle$ (a bemenő sorozat)
Output: $Y[1:p] = \langle y_1, y_2, \ldots, y_p \rangle$ (a prefixek)

1. $Y[1] \leftarrow X[1]$
2. P_i IN PARALLEL FOR $i \leftarrow 2$ TO p DO
3. $\quad Y[i] \leftarrow X[i-1] \oplus X[i]$
4. $k \leftarrow 2$
5. WHILE $k < p$
6. $\quad P_i$ IN PARALLEL FOR $i \leftarrow k+1$ TO p DO
7. $\quad\quad Y[i] \leftarrow Y[i-k] \oplus Y[i]$
8. $\quad k \leftarrow k+k$
9. RETURN Y

Tétel. *Az EREW-PREFIX algoritmus p EREW PRAM processzoron $\Theta(\log p)$ lépésben számítja ki p elem prefixeit.*

Ez az algoritmus nem munkaoptimális, mivel $\Theta(p \log p)$ munkát végez, és ismert olyan soros algoritmus, amely $O(p)$ lépést tesz. Ezek hányadosa nem egyenlő a $O(1)$-gyel.

4. Írja le a PÁROS-PÁRATLAN-ÖSSZEFÉSÜL algoritmust, amely két rendezett sorozatot fésül össze! Milyen lépésszám jellemzi ezt az algoritmust? (1+1 pont)

PÁROS-PÁRATLAN-ÖSSZEFÉSÜL(X_1, X_2, Y) **párh. rek. eljárás**
Számítási modell: EREW PRAM
Input: X_1 és X_2 (a bemenő sorozatok)
Output: L (az összefésült sorozat)

1. IF $m = 1$ THEN
2. $\quad$ Fésüljük össze a sorozatokat egyetlen összehasonlítással.
3. IF $m > 1$ THEN
4. $\quad$ Készítsük el a következő sorozatokat:
$\quad\quad X_1^{pn} = \langle k_1, k_3, \ldots, k_{m-1} \rangle$,
$\quad\quad X_1^{ps} = \langle k_2, k_4, \ldots, k_m \rangle$,
$\quad\quad X_2^{pn} = \langle k_{m+1}, k_{m+3}, \ldots, k_{2m-1} \rangle$,
$\quad\quad X_2^{ps} = \langle k_{m+2}, k_{m+4}, \ldots, k_{2m} \rangle$.
$\triangle$ Legyen $L_1 = \langle l_1, l_2, \ldots, l_m \rangle$ és $L_2 = \langle l_{m+1}, l_{m+2}, \ldots, l_{2m} \rangle$.
5. $\quad$ CALL PÁROS-PÁRATLAN-ÖSSZEFÉSÜL(X_1^{pn}, X_2^{pn}, L_1)
6. $\quad$ CALL PÁROS-PÁRATLAN-ÖSSZEFÉSÜL(X_1^{ps}, X_2^{ps}, L_2)
7. Fésüljük össze az L_1, L_2 sorozatokat, azaz legyen
$\quad\quad L = \langle l_1, l_{m+1}, l_2, l_{m+2}, \ldots, l_m, l_{2m} \rangle$.
8. Hasonlítsuk össze az (l_{m+i}, l_{i+1}) $(i = 1, 2, \ldots, m-1)$ párokat
$\quad$ és szükség esetén cseréljük fel őket.
9. RETURN L

Tétel. *A PÁROS-PÁRATLAN-ÖSSZEFÉSÜL algoritmus két m hosszúságú kulcssorozatot $O(\log m)$ lépéssel összefésül $2m$ EREW PRAM processzoron.*

Összes pont: 6 pont

Párhuzamos algoritmusok, 2. zh., MINTA

1. Írja le a `forkjoin` programot PVM környezetben! (2 pont)

```
/* defines and prototypes for the PVM library */
#include <pvm3.h>
#include <stdio.h>
#include <stdlib.h>
/* Maximum number of children this program will spawn */
#define MAXNCHILD 20
/* Tag to use for the joing message */
#define JOINTAG 11
int
main(int argc, char* argv[])
{
      /* number of tasks to spawn, use 3 as the default */
      int ntask = 3;
      /* return code from pvm calls */
      int info;
      /* my task id */
      int mytid;
      /* my parents task id */
      int myparent;
      /* children task id array */
      int child[MAXNCHILD];
      int i, mydata, buf, len, tag, tid;
      /* find out my task id number */
      mytid = pvm_mytid();
      /* check for error */
      if (mytid < 0) {
        /* print out the error */
        pvm_perror(argv[0]);
        /* exit the program */
        return -1;
        }
      /* find my parent's task id number */
      myparent = pvm_parent();
      /* exit if there is some error other than PvmNoParent */
      if ((myparent < 0) && (myparent != PvmNoParent)) {
        pvm_perror(argv[0]);
        pvm_exit();
        return -1;
        }
      /* if i don't have a parent then i am the parent */
      if (myparent == PvmNoParent) {
        /* find out how many tasks to spawn */
        if (argc == 2) ntask = atoi(argv[1]);
```

```
    /* make sure ntask is legal */
    if ((ntask < 1) || (ntask > MAXNCHILD)) { pvm_exit(); return 0; }
    /* spawn the child tasks */
    info= pvm_spawn(argv[0], (char**)0, PvmTaskDefault, (char*)0,
        ntask, child);
    /* print out the task ids */
    for (i = 0; i < ntask; i++)
        if(child[i] < 0) /* print the error code in decimal*/
          printf(" %d", child[i]);
        else /* print the task id in hex */
          printf("t%x ", child[i]);
    putchar('\n');
    /* make sure spawn succeeded */
    if (info == 0) { pvm_exit(); return -1; }
    /* only expect responses from those spawned correctly */
    ntask = info;
    for (i = 0; i < ntask; i++) {
        /* recv a message from any child process */
        buf = pvm_recv(-1, JOINTAG);
        if (buf < 0) pvm_perror("calling recv");
        info = pvm_bufinfo(buf, &len, &tag, &tid);
        if (info < 0) pvm_perror("calling pvm_bufinfo");
        info = pvm_upkint(&mydata, 1, 1);
        if (info < 0) pvm_perror("calling pvm_upkint");
        if (mydata != tid) printf("This should not happen!\n");
        printf("Length %d, Tag %d, Tid t%x\n", len, tag, tid);
        }
    pvm_exit();
    return 0;
    }
  /* i'm a child */
  info = pvm_initsend(PvmDataDefault);
  if (info < 0) {
    pvm_perror("calling pvm_initsend"); pvm_exit(); return -1;
  }
  info = pvm_pkint(&mytid, 1, 1);
  if (info < 0) {
    pvm_perror("calling pvm_pkint"); pvm_exit(); return -1;
  }
  info = pvm_send(myparent, JOINTAG);
  if (info < 0) {
    pvm_perror("calling pvm_send"); pvm_exit(); return -1;
  }
  pvm_exit();
  return 0;
}
```

2. Ismertesse a mohó algoritmust a PPR-re rácson és mutassa be a legfontosabb jellemzőit! (1 pont)

   A PPR probléma megoldásához egy $\sqrt{p} \times \sqrt{p} = a \times a$ méretű rácson egy üzenetnek az (1,1) csúcsból az (a, a) csúcsba szállításához legalább $N(n, a^2, P) \geq 2(a-1)$ lépésre van szükség. Ezért $2(a-1)$ egy alsó korlát bármely csomagirányító algoritmus lépésszámának legrosszabb esetét vizsgálva: $W(n, a^2, A) \geq 2(a-1)$.

   Egy egyszerű PPR algoritmus, amely felhasználja a láncoknál ismertetett csomagirányítási algoritmusokat, a következő. Legyen q egy tetszőleges csomag, amelyet a $P_{i,j}$ processzortól a $P_{k,l}$ indexűnek kell elküldeni. Az csomag az első fázisban a j-edik oszlop mentén a k-adik sorig halad a legrövidebb úton. A második fázisban a k-adik sor mentén a legrövidebb úton az l-edik oszlophoz elérve a csomag megérkezett a rendeltetési helyére.

   Egy csomag azonnal megkezdheti a második fázist az első befejezése után, nem kell semmi másra várnia.

   Az első fázis legfeljebb $a-1$ lépésben elvégezhető, mivel az egy kezdőcsomagos feladatra vonatkozó lemma alkalmazható. A második fázis az egy célcsomagos feladatra vonatkozó lemmából következően nem tart $a-1$ lépésnél tovább. Így az algoritmus lépésszáma legfeljebb $2(a-1)$, ami az elméleti alsó korlát, azaz az algoritmus abszolút optimális.

   De van egy komoly hátránya ennek az algoritmusnak, mégpedig az, hogy a várakozási sor hossza $a/2$. Nézzünk erre egy példát. Legyen a csomagirányítási feladat olyan, hogy az első oszlop minden csomagját a $a/2$-edik sorba kelljen szállítani. Egy ilyen PPR esetén a $(a/2, 1)$ indexű processzor minden lépésben két csomagot kap.

3. Ismertesse a ritka rendezés algoritmust négyzeten (RITKA-RENDEZ)! (1 pont)

   RITKA-RENDEZ(X, Y) **párhuzamos eljárás**
   Számítási modell: négyzet
   Bemenet: X (a rendezendő kulcsok)
   Kimenet: Y (a rendezett kulcsok)

   1. $P_{1,j}$ IN PARALLEL FOR $j \leftarrow 1$ TO a DO
   2. Szórja a k_j kulcsot a j-edik oszlopban.
   3. $P_{i,i}$ IN PARALLEL FOR $i \leftarrow 1$ TO a DO
   4. Szórja a k_i kulcsot az i-edik sorban.
   5. $P_{i,i}$ IN PARALLEL FOR $i \leftarrow 1$ TO a DO
   6. Kiszámítja k_i rangját az i-edik sorban összehasonlításokat végezve.
   7. $P_{j,j}$ IN PARALLEL FOR $j \leftarrow 1$ TO a DO
   8. Elküldi a k_j kulcs rangját $P_{1,j}$-nek.
   9. $P_{1,r}$ IN PARALLEL FOR $r \leftarrow 1$ TO a DO
   10. Az r rangú kulcs elküldése a $P_{1,r}$ processzorhoz.
   $\triangle$ Y az első sorban levő rendezett kulcsok sorozata.
   11. RETURN Y

4. Ismertesse a bináris fa beágyazását a hiperkockába megoldást! (1 pont)

   Egy d szintes (teljes) bináris fában $p = 2^d - 1$ processzor van: $P_1, P_2, \ldots, P_p$. Az adatszerkezetekkel kapcsolatos terminológia szerint a P_1 processzort *gyökér*nek, a

$P_{(p+1)/2}, P_{(p+1)/2+1}, \ldots, P_p$ processzorokat *levél*nek, a többi processzort *belső processzor*nak nevezzük. Ha P_i nem levél, akkor össze van kötve a *gyerekei*nek nevezett P_{2i} és P_{2i+1} processzorokkal. Ha P_j nem a gyökér, akkor össze van kötve a *szülő*jének nevezett $P_{\lfloor j/2 \rfloor}$ processzorral.

A következőkben megmutatjuk, hogy egy F p levelű (teljes, bináris) fa (ahol $p = 2^d$) beágyazható $\mathcal{H}_d$-be. Mivel a p levelű fának $2p-1$ processzora van, így a leképezés nem lehet kölcsönösen egyértelmű. Ha a leveleket $0, 1, \ldots, p-1$ jelöli, akkor képezzük az i-edik levelet $\mathcal{H}_d$ i-edik processzorára. F belső processzorait pedig képezzük arra a processzorra, amelyre az adott processzor legbaloldalibb leszármazottját képeztük.

A megadott beágyazás segítségével fa algoritmusokat hatékonyan szimulálhatunk soros hiperkockákkal.

5. Írja le a LeLann algoritmust a vezetőválasztás gyűrűben problémára! (1 pont)

LELANN(K) **párhuzamos eljárás**
Számítási modell: egyirányú gyűrű
Bemenet: K (a kezdő processzorok indexeinek halmaza)
Kimenet: i (a vezető processzor indexe)

```
1. P_i IN PARALLEL FOR i ← 1 TO n DO
2.     IF i ∈ K THEN
3.         áll[i] ← jelölt
4.         J_i ← {i}
5.     ELSE
6.         áll[i] ← nem_jelölt
7. P_i IN PARALLEL FOR i ← 1 TO n DO
△ a egy lokális változó az indexek összehasonlításához.
8.     IF áll[i] = jelölt THEN
9.         KÜLD_i(i)
10.        FOGAD_i(a)
11.        WHILE a ≠ i DO
12.                J_i ← J_i ∪ {a}
13.                KÜLD_i(a)
14.                FOGAD_i(a)
15.        IF i = min(J_i) THEN
16.                áll[i] ← vez
17.                l ← i
18.        ELSE
19.                áll[i] ← nem_vez
20.    ELSE
21.        FOGAD_i(a)
22.        WHILE a ≠ i DO
23.                KÜLD_i(a)
24.                FOGAD_i(a)
25. RETURN(l);
```

Összes pont: 6 pont

Párhuzamos algoritmusok, vizsga, MINTA

Beugró feladat: Írja le a `hello` programot JCluster környezetben! (Sikeres — Nem sikeres)

```
import jcluster.*;
public class hello extends JTasklet {
	public hello() {
	}
	public void child() {
		System.out.println(env.pvm_me() + " : child begin");
		try {
			env.pvm_joinGroup("test");
			System.out.println(env.pvm_me() + " : child join group");
			env.pvm_barrier("test", 3);
			System.out.println(env.pvm_me() + " : child barrier over");
			JMessage m = env.pvm_recvTag(9999);
			System.out.println(env.pvm_me() + " : child recv over");
			System.out.println(env.pvm_me() + " : " + m.unpack());
			env.pvm_lvGroup("test");
		} catch (JException e) {
			System.out.println(e.toString());
		}
	}
	public void parent() {
		try {
			System.out.println(env.pvm_me() + " : parent begin spawn");
			env.pvm_spawn("hello", 2);
			env.pvm_joinGroup("test");
			System.out.println(env.pvm_me() + " : parent join group");
			env.pvm_barrier("test", 3);
			System.out.println(env.pvm_me() + " : parent barrier over");
			JMessage m = new JMessage(env);
			m.pack("Hello World! -by jcluster");
			System.out.println(env.pvm_me() + " : parent before bcast");
			env.pvm_bcast("test", m);
			System.out.println(env.pvm_me() + " : parent bcast over");
			env.pvm_lvGroup("test");
		} catch (JException e) {
			System.out.println(e.toString());
		}
	}
	public void work() {
		if (env.pvm_parent() == -1) {
			parent();
		} else {
			child();
		}
	}
}
```

1. Ismertesse a prefixszámítás munkaoptimális algoritmusát és az ehhez szükséges CREW-prefix (CREW PRAM modellen) algoritmust! Miért lesz ez az algoritmus munkaoptimális, a CREW-prefix pedig miért nem az? Válaszát indokolja! (1 + 1 + 1 pont)

CREW-PREFIX(p, X, Y) **párhuzamos rekurzív eljárás**
Számítási modell: CREW PRAM
Input: p (a bemenet hossza) és $X[1:p] = \langle x_1, x_2, \ldots, x_p \rangle$ (a bemenő adatok)
Output: $Y[1:p] = \langle y_1, y_2, \ldots, y_p \rangle$ (a kimenő sorozat, azaz a prefix)

1. IF $p = 1$ THEN
$\triangle$ Az i az elem indexe az X inputból.
2. $y_i \leftarrow x_i$
3. IF $p > 1$ THEN
4. CALL CREW-PREFIX($p/2$, $X[1..p/2]$, $Y[1..p/2]$)
5. CALL CREW-PREFIX($p/2$, $X[p/2+1..p]$, $Y[p/2+1..p]$)
6. P_i IN PARALLEL FOR $i \leftarrow (p/2+1)$ TO p DO
7. $y_i \leftarrow y_{p/2} \oplus y_i$
8. RETURN Y

OPTIMÁLIS-PREFIX(p, X, Y) **párhuzamos eljárás**
Számítási modell: CREW PRAM
Input: p (a bemenet hossza) és $X[1:p] = \langle x_1, x_2, \ldots, x_p \rangle$ (a bemenő sorozat)
Output: $Y[1:p] = \langle y_1, y_2, \ldots, y_p \rangle$ (a prefixek)

1. P_i IN PARALLEL $i \leftarrow 1$ TO $\lceil p/\log p \rceil$ DO
2. $z_{(i-1)\log p+1} \leftarrow x_{(i-1)\log p+1}$
3. FOR $j \leftarrow 2$ TO $\log p$ DO
4. $z_{(i-1)\log p+j} \leftarrow z_{(i-1)\log p+j-1} \oplus x_{(i-1)\log p+j}$
$\triangle$ Legyen $\hat{Z} = \{z_{\log p}, z_{2\log p}, \ldots, z_p\}$, $\hat{W} = \{w_{\log p}, w_{2\log p}, \ldots, w_p\}$.
5. P_i IN PARALLEL $i \leftarrow 1$ TO $\lceil p/\log p \rceil$ DO
6. CALL CREW-PREFIX($\lceil p/\log p \rceil$, $\hat{Z}$, $\hat{W}$)
7. P_i IN PARALLEL $i \leftarrow 1$ TO $\lceil p/\log p \rceil$ DO
8. IF $i = 1$ THEN
9. FOR $j \leftarrow 1$ TO $\log p$ DO
10. $y_j \leftarrow z_j$
12. ELSE
13. FOR $j \leftarrow 1$ TO $\log p$ DO
14. $y_{(i-1)\log p+j} \leftarrow w_{(i-1)\log p} \oplus z_{(i-1)\log p+j}$
15. RETURN Y

Az OPTIMÁLIS-PREFIX nevű CREW PRAM modellen megadott algoritmus $\lceil p/\log p \rceil$ processzort használ és $\Theta(\log p)$ lépést tesz. Ennek elvégzett munkája csak $\Theta(p)$, ezért a hatékonysága $\Theta(1)$ és az algoritmus munkaoptimális.

CREW-PREFIX algoritmus nem munkaoptimális, mivel $\Theta(p \log p)$ munkát végez, és ismert olyan soros algoritmus, amely $O(p)$ lépést tesz. Ezek hányadosa nem egyenlő a $O(1)$-gyel.

2. Ismertesse a determinisztikus tömbrangsorolás algoritmust! Munkaoptimális-e ez az algoritmus? Válaszát indokolja!
(1 + 1 pont)

DET-RANGSOROL($szomsz[1:p]$**,** $rang[1:p]$**)** **párh. elj.**
Számítási modell: EREW PRAM
Input: $szomsz[1:p]$ (a mutatók)
Output: $rang[1:p]$ (a rangok)

1. P_i IN PARALLEL FOR $i \leftarrow 1$ TO p DO
2. IF $szomsz[i] = 0$ THEN
3. $rang[i] \leftarrow 0$
4. ELSE
5. $rang[i] \leftarrow 1$
6. FOR $j \leftarrow 1$ TO $\lceil \log p \rceil$ DO
7. P_i IN PARALLEL FOR $i \leftarrow 1$ TO p DO
8. IF $szomsz[i] \neq 0$ THEN
9. $rang[i] \leftarrow rang[i] + rang[szomsz[i]]$
10. $szomsz[i] \leftarrow szomsz[szomsz[i]]$
11. RETURN *rang*

Mivel a DET-RANGSOROL algoritmus $\Theta(p \log p)$ munkát végez, ezért nem munkaoptimális, viszont logaritmikusan munkahatékony.

3. Írja le, hogyan lehet beágyazni a tóruszt a $\mathcal{H}_d$-be úgy, hogy a felfúvódás, a késleltetés és a torlódás mindegyike 1! (2 pont)

Egy $m \times n$ tórusz származtatható egy $m \times n$ méretű rácsból úgy, hogy a rács minden sorának első és utolsó processzorát, valamint minden oszlopának első és utolsó processzorát is összekötjük: tehát a rácsot kiegészítjük a $P_{i,1}P_{i,m}$ $(1 \leq i \leq m)$ és $P_{1,j}P_{1,n}$ $(1 \leq j \leq n)$ élekkel.

Legyen M egy $2^r \times 2^c$ méretű tórusz. M beágyazható úgy egy hiperkockába, hogy a felfúvódás, a késleltetés és a torlódás mindegyike 1 legyen. Van 2^r sor és 2^c oszlop M-ben. Ha egy d-dimenziós hiperkockában a d bitből valamely q-t rögzítjük $(1 \leq q \leq d-1)$, akkor a feltételnek megfelelő processzorok egy $\mathcal{H}_{d-q}$ alkockát alkotnak $\mathcal{H}_d$-ben. Ha egy $(r+c)$-bites bináris számnak rögzítjük az *r legnagyobb helyi értékű bit*jét (Most Significant Bits = MSB) és a maradék c bitet tetszőlegesen variáljuk, a kapott 2^c szám egy alkockát határoz meg $\mathcal{H}_{r+c}$-ben. Ebbe az alkockába beágyazható egy 2^c processzorból álló gyűrű. Az r MSB minden lehetséges megválasztásához megvan a megfelelő $\mathcal{H}_c$, így M minden sorát leképezhetjük egyre.

Egészen pontosan az i-edik sort azon $\mathcal{H}_c$-re képezzük, amelyet úgy kapunk, hogy az r MSB értékét pontosan $g(i,r)$-re állítjuk. Ebből következik, hogy általában a tórusz $P_{i,j}$ processzora a $\mathcal{H}_{r+c}$ $P_{g(i,r),g(j,c)}$ processzorára képződik. Így minden sor szomszédos processzorai a hiperkocka szomszédos processzoraira képződnek. A fentihez hasonló gondolatmenettel belátható, hogy a megadott leképezés az oszlopok szomszédos elemeit is szomszédos processzorokra képezi. Ebből következik, hogy mind a felfúvódás a késleltetés és a torlódás 1.

4. Írja le a Chang-Roberts algoritmust a vezetőválasztás gyűrűben problémára! (1 pont)

CHANG-ROBERTS(K) **párhuzamos eljárás**

Számítási modell: egyirányú gyűrű
Bemenet: K (a kezdő processzorok indexeinek halmaza)
Kimenet: i (a vezető processzor indexe)

1. P_i IN PARALLEL FOR $i \leftarrow 1$ TO n DO
2. IF $i \in K$ THEN
3. áll[i] $\leftarrow$ jelölt
4. $J_i \leftarrow \{i\}$
5. ELSE
6. áll[i] $\leftarrow$ nem_jelölt
7. P_i IN PARALLEL FOR $i \leftarrow 1$ TO n DO

$\triangle$ a egy lokális változó az indexek összehasonlításához.

8. IF áll[i] = jelölt THEN
9. KÜLD$_i(i)$
10. FOGAD$_i(a)$
11. WHILE $a < i$ DO
12. $J_i \leftarrow J_i \cup \{a\}$
13. KÜLD$_i(a)$
14. FOGAD$_i(a)$
15. IF $i = \min(J_i)$ THEN
16. áll[i] $\leftarrow$ vez
17. $l \leftarrow i$
18. ELSE
19. áll[i] $\leftarrow$ nem_vez
20. ELSE
21. FOGAD$_i(a)$
22. WHILE $a \neq i$ DO
23. KÜLD$_i(a)$
24. FOGAD$_i(a)$
25. RETURN(l);

Összes pont: 8 pont [0-3p elégtelen(1); 4p elégséges(2); 5p (közepes); 6p (jó); 7-8p jeles(5)]