Paralelné programovanie Základné komunikačné operácie

doc. Ing. Michal Čerňanský, PhD.
FIIT STU Bratislava

Prehľad tém

- One-to-All Broadcast a All-to-One Reduction
  - Broadcast a redukcia, jeden posiela všetkým a všetci posielajú jednému
- All-to-All Broadcast a Reduction
  - Broadcast a redukcia, všetci posielajú všetkým
- All-Reduce a Prefix-Sum Operations
  - Redukcia, výsledok pre všetkých a čiastočná redukcia podľa prefixov
- Scatter a Gather
  - Rozptýlenie a zhromaždenie
- Komunikácia každý s každým
- Cyklické posunutie
- Zrýchlenie niektorých komunikačných operácií

Základné komunikačné operácie - úvod

- Mnohé interakcie paralelných programov nastávajú v dobre definovaných štruktúrach
- Efektívna implementácia
  - Lepšia výkonnosť, jednoduchší vývoj, cena, kvalita SW
  - Nutnosť využiť architektúru komunikačnej siete
- Vzorové architektúry
  - Kruhová topológia
  - 2D Mriežka
  - Hyperkocka

Základné komunikačné operácie - úvod

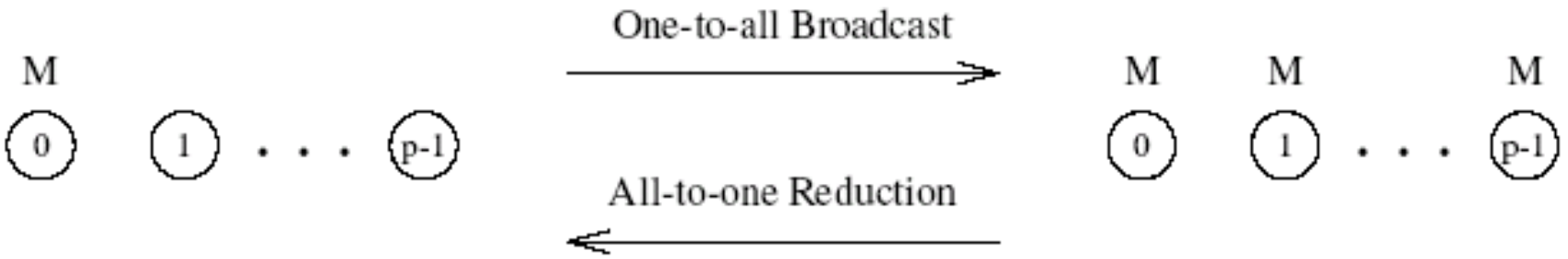
- Operácie skupinovej komunikácie – primitívi typu bod – bod
- Komunikácia správy veľkosti m v nezahltenej sieti trvá $t = t_s + t_w m$
- Zahltenie siete – člen t_w
- Obojsmerná komunikácia prostredníctvom jediného portu

One-to-All Broadcast a All-to-One Redukcia

- Jeden procesor vlastní údaje o veľkosti *m* a potrebuje ich zaslať ostatným
- Duálny (komplementárny) problém k broadcastu „jedného viacerým“ je redukcia „od viacerých jednému“
- Redukcia „jedného viacerým“
  - Každý procesor má údaje o veľkosti *m* (počte *m* položiek)
  - Dátové položky musia byť kombinované položka po položke a výsledok poskytnutý cieľovému procesoru

One-to-All Broadcast a All-to-One Redukcia

- One-to-all broadcast a all-to-one redukcia medzi procesormi

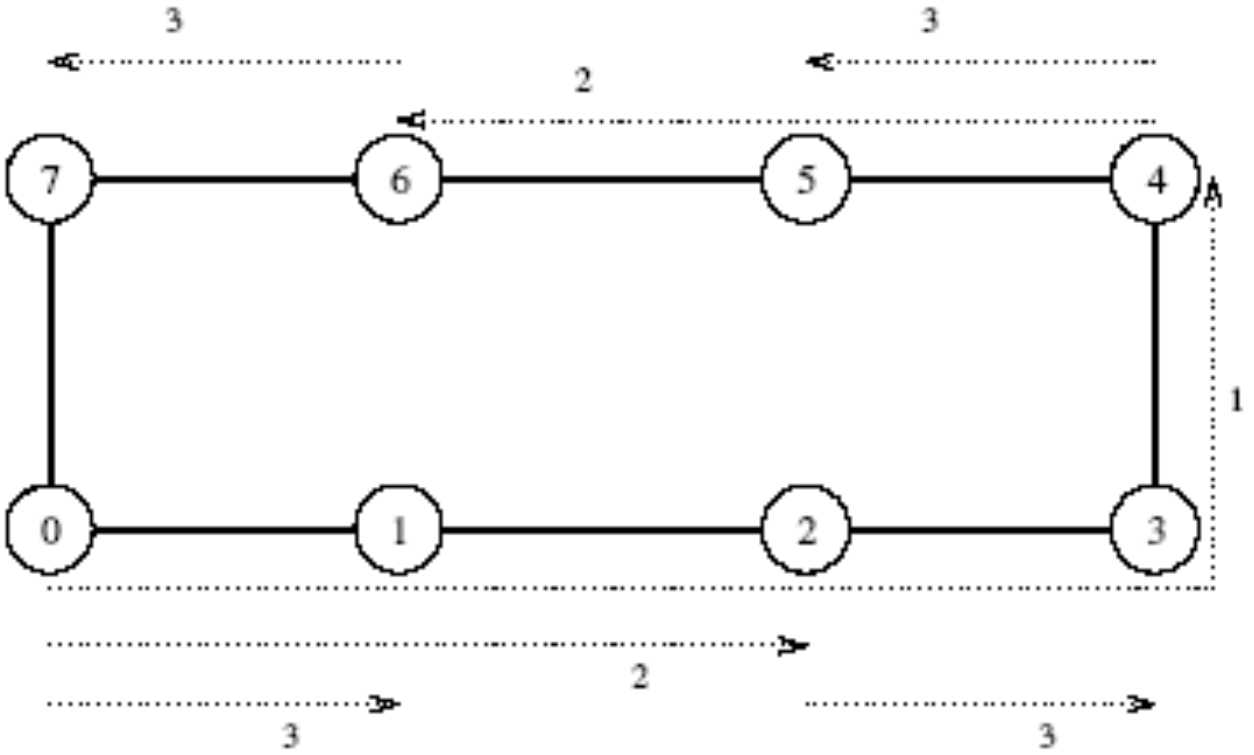


One-to-All Broadcast a All-to-One Redukcia v kruhovej topológií

- Jednoduché riešenie – poslať *p-1* správ zo zdroja ostatným *p-1* procesorom – neefektívne
- Použitie rekurzívneho zdvojovania – zdroj zašle správu vybranému procesoru a tým získame dva nezávislé problémy na dvoch poloviciach počítačového systému
- Redukcia môže byť vykonaná identickým spôsobom vykonaním krokov v opačnom poradí

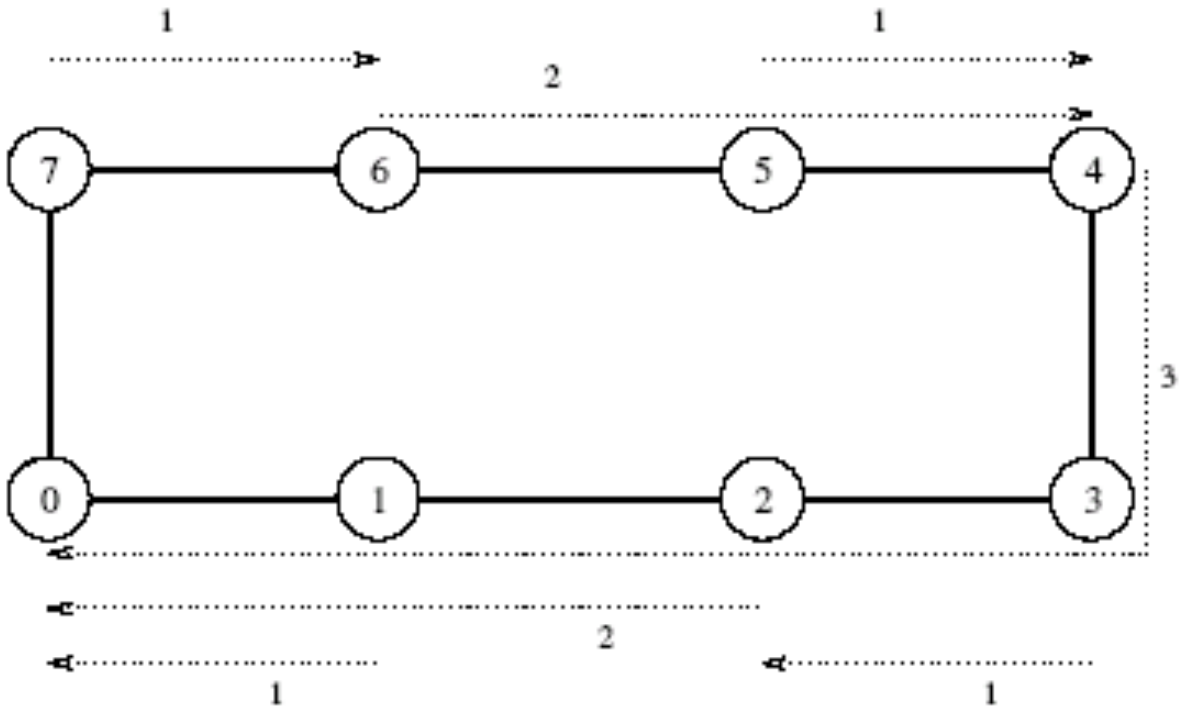
One-to-All Broadcast

- Broadcast jeden viacerým v kruhovej topológií s 8 uzlami



All-to-One Redukcia

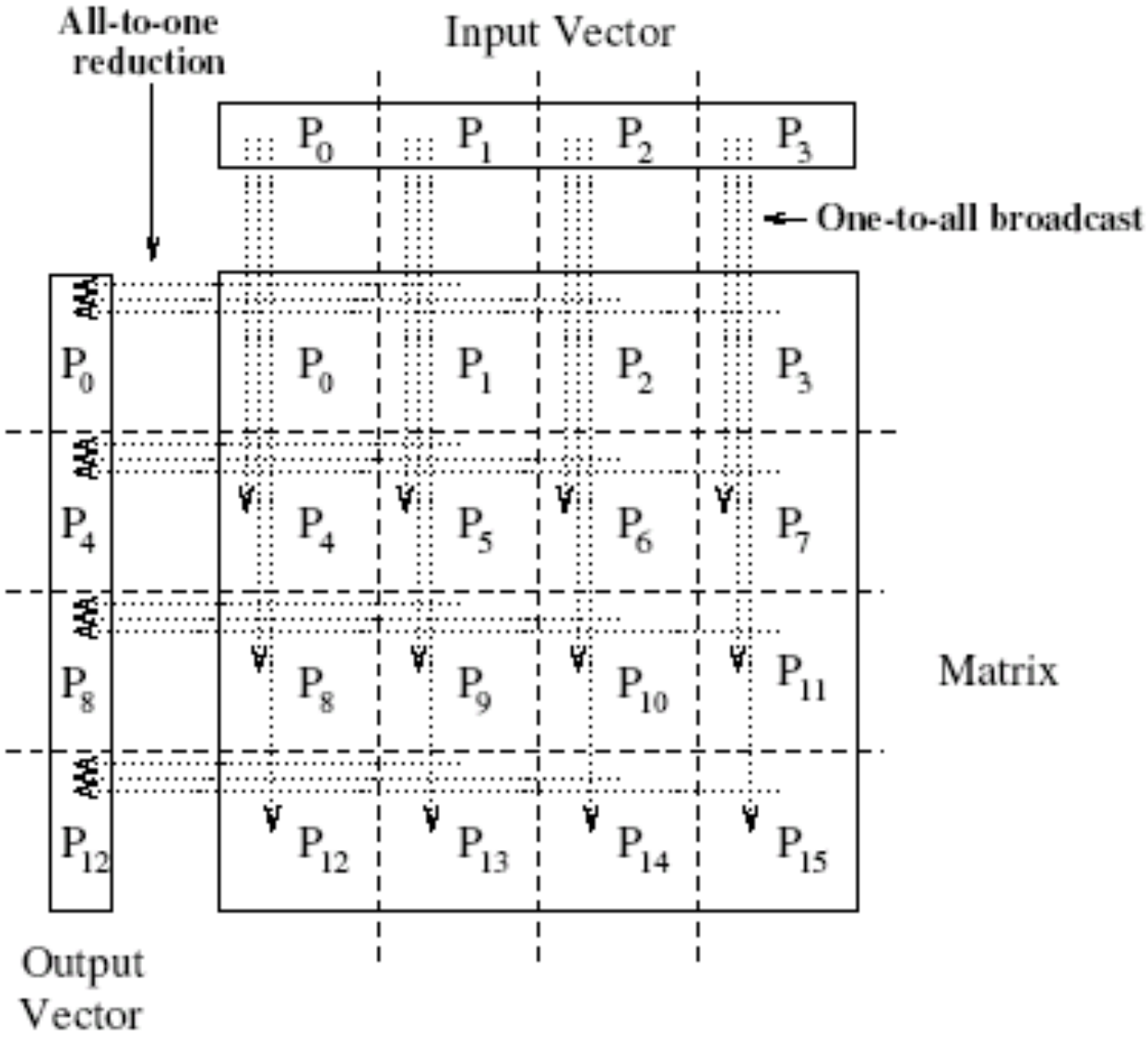
- All-to-One Redukcia v kruhovej topológií s 8 uzlami



Broadcast a redukcia - príklad

- Uvažujme problém násobenia matice s vektorom
- Matica n x n je namapovaná na procesorz organizované do (virtuálnej) mriežky n x n, vektor je namapovaný na prvý riadok mriežky
- Prvý krok – one-to-all broadcast prvkov vektora pozdĺž stĺpcov mriežky súbežne pre všetky stĺpce
- Procesory vypočítajú lokálne súčiny prvku vektora a lokálneho prvku matice
- Posledný krok – súbežné all-to-one redukcie akumulujúce čiastkové výsledky pomocou operácie „súčet“

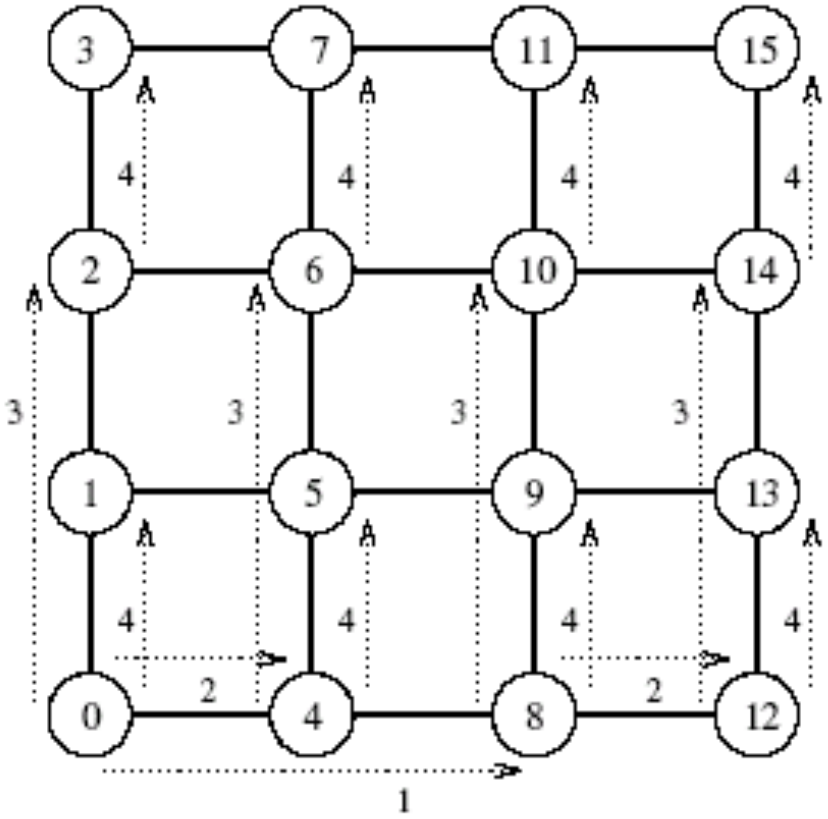
Broadcast a redukcia - príklad



Broadcast a redukcia na mriežke

- Každý riadok alebo stĺpec mriežky – lineárne pole $\sqrt{p}$ uzlov
- Broadcast a redukcia môžu byť vykonané v dvoch krokoch – prvý krok vykonanie operácie pozdĺž riadka a druhý krok pozdĺž každého stĺpca súbežne
- Tento spôsob je možné zovšeobecniť do vyšších dimenzií

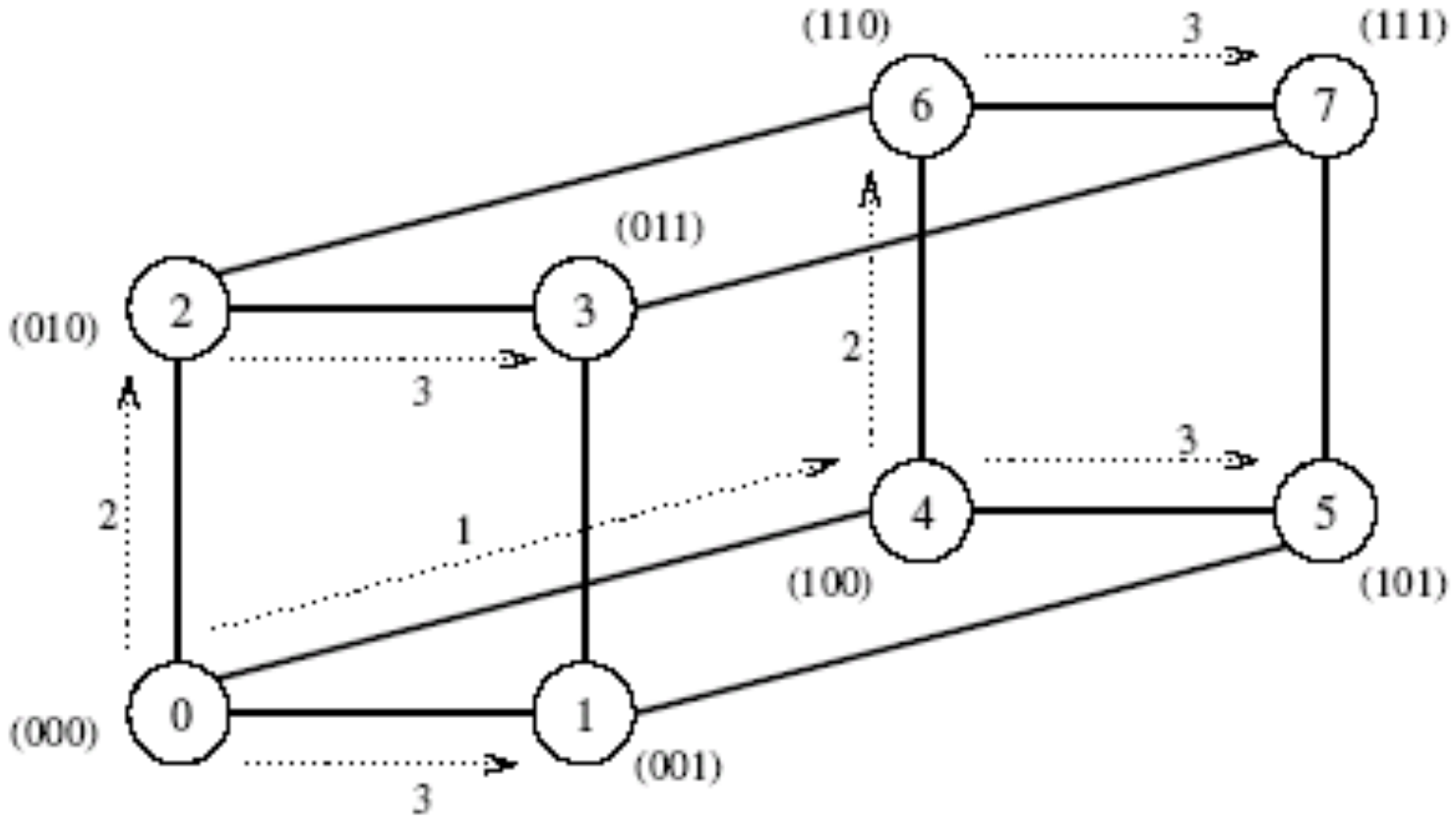
Broadcast a redukcia na mriežke - príklad



Broadcast a redukcia na hyperkocke

- Hyperkocka s 2^d uzlami d-dimenzionálna mriežka s dvoma uzlami v každej dimenzií
- Algoritmus na mriežke môže byť zovšeobecnený aj na hyperkocku – operácia je vykonaná v d (= $log\ p$) krokoch

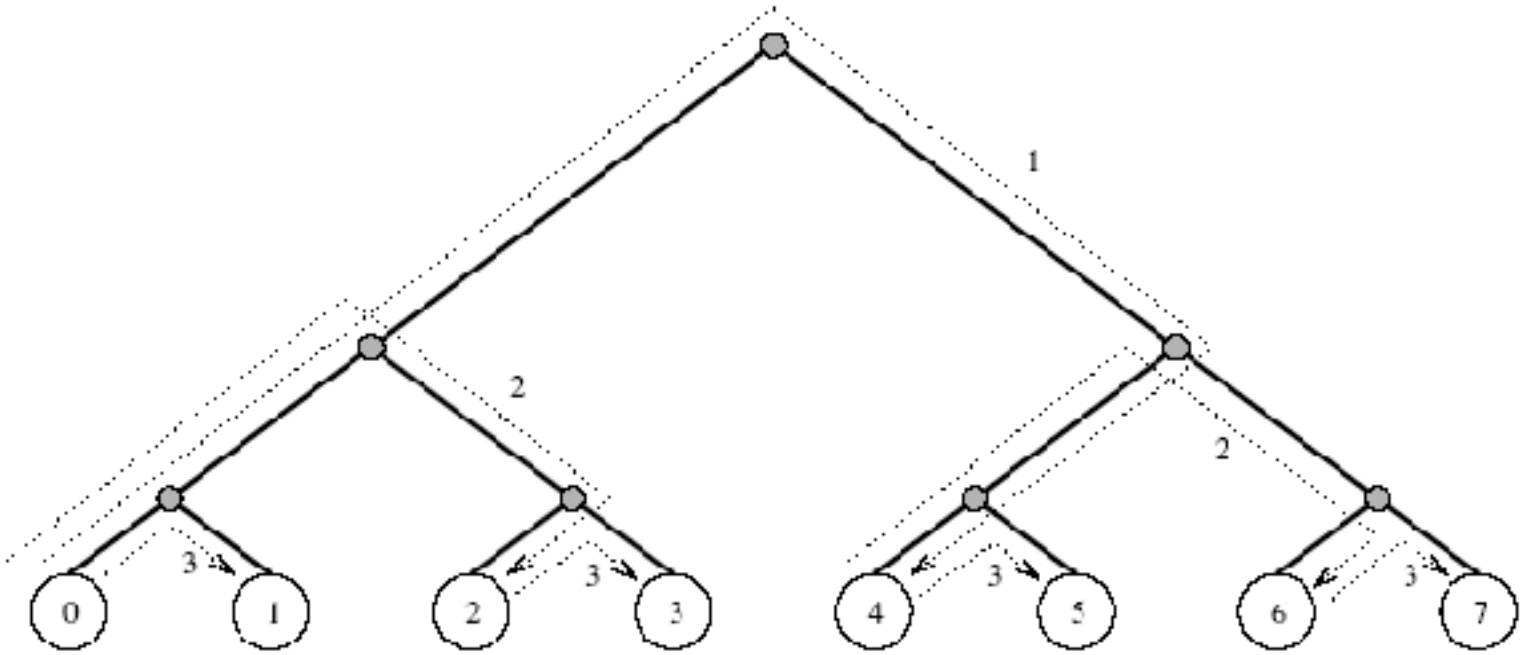
Broadcast a redukcia na hyperkocke - príklad



Broadcast a redukcia na vyváženom binárnom strome

- Binárny strom – procesory ako listy a prepínače ako vnútorné uzly
- Zdrojový procesor – koreň stromu.
- Prvý krok zdroj zašle údaje pravému synovi (predpokladajúc, že zdroj je súčasne aj ľavý syn)
- Problém je teraz dekomponovaný na dva problémy s polovičným počtom procesorov

Broadcast a redukcia na vyváženom binárnom strome



Algoritmy broadcast-u a redukcie

- Uvedené algoritmy zodpovedajú rovnakému algoritmickému vzoru
- Algoritmus bude ilustrovaný na topológií hyperkocky, ale (ako bolo uvedené) môže byť adaptovaný na iných architektúrach
- Hyperkocka má 2^d uzlov a *my_id* je označenie aktuálneho uzla
- *X* je správa na broadcast, iniciálne na uzle č. 0

Algoritmy broadcast-u a redukcie

```
1.    procedure GENERAL_ONE_TO_ALL_BC(d, my_id, source, X)
2.    begin
3.        my_virtual_id := my_id XOR source;
4.        mask := 2^d − 1;
5.        for i := d − 1 downto 0 do      /* Outer loop */
6.            mask := mask XOR 2^i;   /* Set bit i of mask to 0 */
7.            if (my_virtual_id AND mask) = 0 then
8.                if (my_virtual_id AND 2^i) = 0 then
9.                    virtual_dest := my_virtual_id XOR 2^i;
10.                   send X to (virtual_dest XOR source);
          /* Convert virtual_dest to the label of the physical destination */
11.               else
12.                   virtual_source := my_virtual_id XOR 2^i;
13.                   receive X from (virtual_source XOR source);
          /* Convert virtual_source to the label of the physical source */
14.               endelse;
15.       endfor;
16.   end GENERAL_ONE_TO_ALL_BC
```

Algoritmy broadcast-u a redukcie

```
1.    procedure ALL_TO_ONE_REDUCE(d, my_id, m, X, sum)
2.    begin
3.        for j := 0 to m − 1 do sum[j] := X[j];
4.        mask := 0;
5.        for i :   0 to d − 1 do
              /* Select nodes whose lower i bits are 0 */
6.            if (my_id AND mask) = 0 then
7.                if (my_id AND 2^i) ≠ 0 then
8.                    msg_destination := my_id XOR 2^i;
9.                    send sum to msg_destination;
10.               else
11.                   msg_source := my_id XOR 2^i;
12.                   receive X from msg_source;
13.                   for j :   0 to m − 1 do
14.                       sum[j] := sum[j] + X[j];
15.               endelse;
16.           mask := mask XOR 2^i;   /* Set bit i of mask to 1 */
17.       endfor;
18.   end ALL_TO_ONE_REDUCE
```

Analýza časovej náročnosti

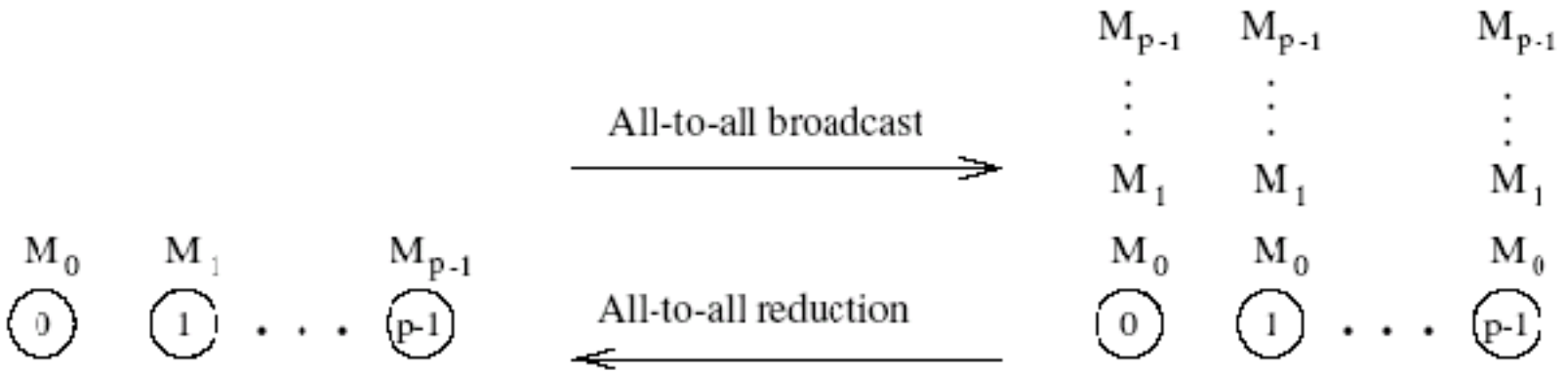
- Broadcast alebo redukcia vyžadujú *log p* správ typu bod-bod s časovou náročnosťou $t_s + t_w m$.
- Celkový čas je teda daný

$$T - (t_s + t_w m) \log p.$$

All-to-All Broadcast a Reduction

- Zovšeobecnenie operácie broadcast, v ktorej je každý proces zdrojom aj cieľom
- Proces posiela tú istú správu dĺžky m všetkým ostatným procesom avšak rôzne procesy môžu posielať rôzne správy

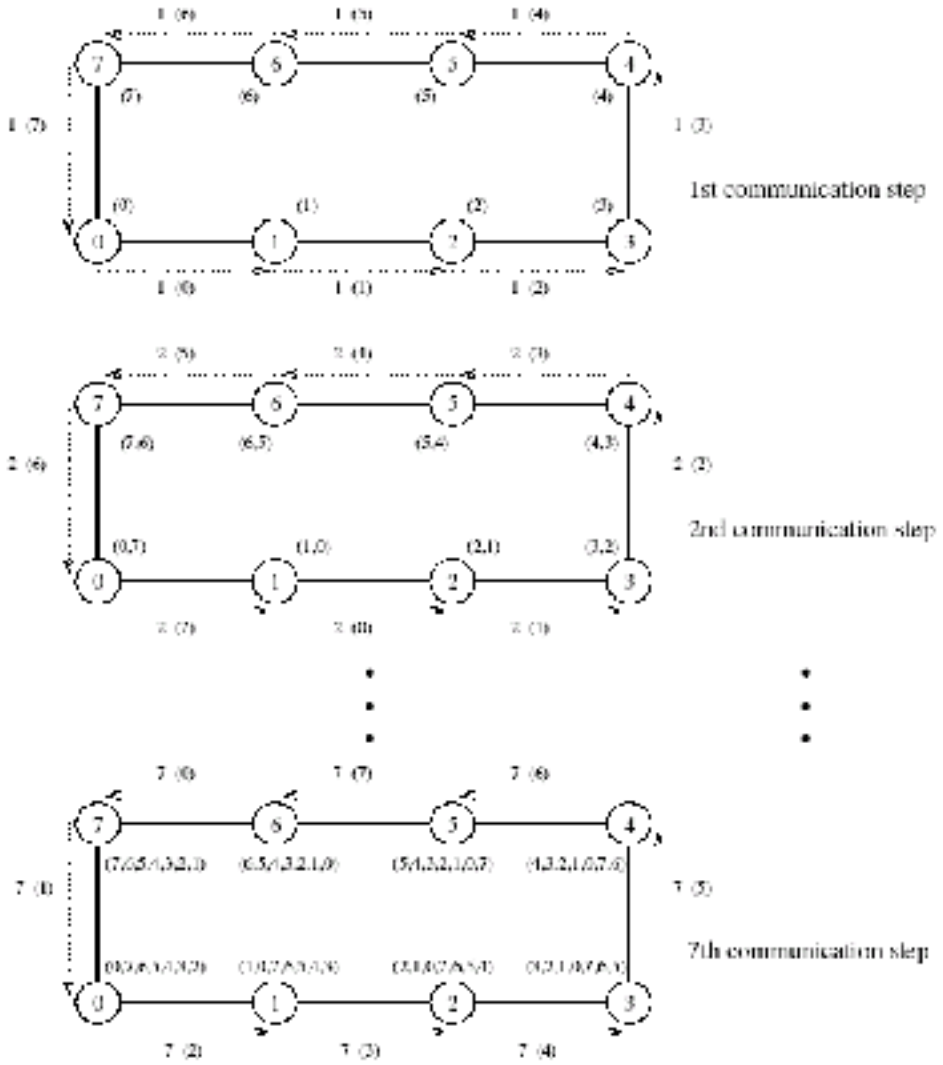
All-to-All Broadcast and Reduction



All-to-All Broadcast and Reduction na prstencovej topológií

- Jednoduchý prístup: vykonanie p broadcastov jeden viacerým - neefektívne
- Každý uzol najskôr zašle údaje jednému zo susedných uzlov
- V nasledujúcich krokoch odosiela už prijaté údaje od jedného zo susedných uzlov druhému
- Algoritmus končí za p-1 krokov

All-to-All Broadcast and Reduction na prstencovej topológií



All-to-All Broadcast and Reduction na prstencovej topológii

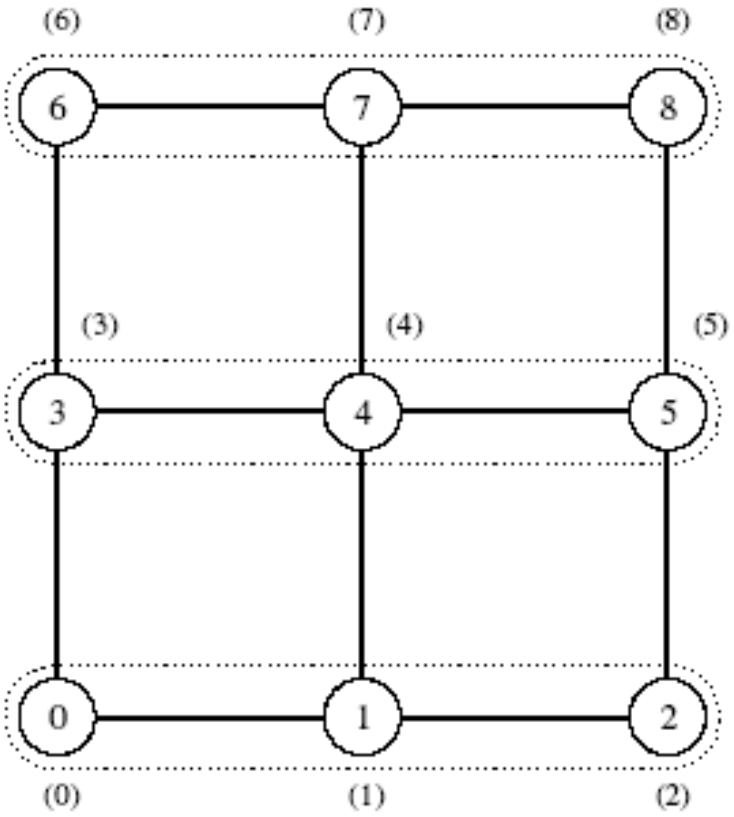
```
1.    procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
2.    begin
3.        left := (my_id − 1) mod p;
4.        right := (my_id + 1) mod p;
5.        result := my_msg;
6.        msg := result;
7.        for i := 1 to p − 1 do
8.            send msg to right;
9.            receive msg from left;
10.           result := result ∪ msg;
11.       endfor;
12.   end ALL_TO_ALL_BC_RING
```

All-to-All Broadcast na mriežke

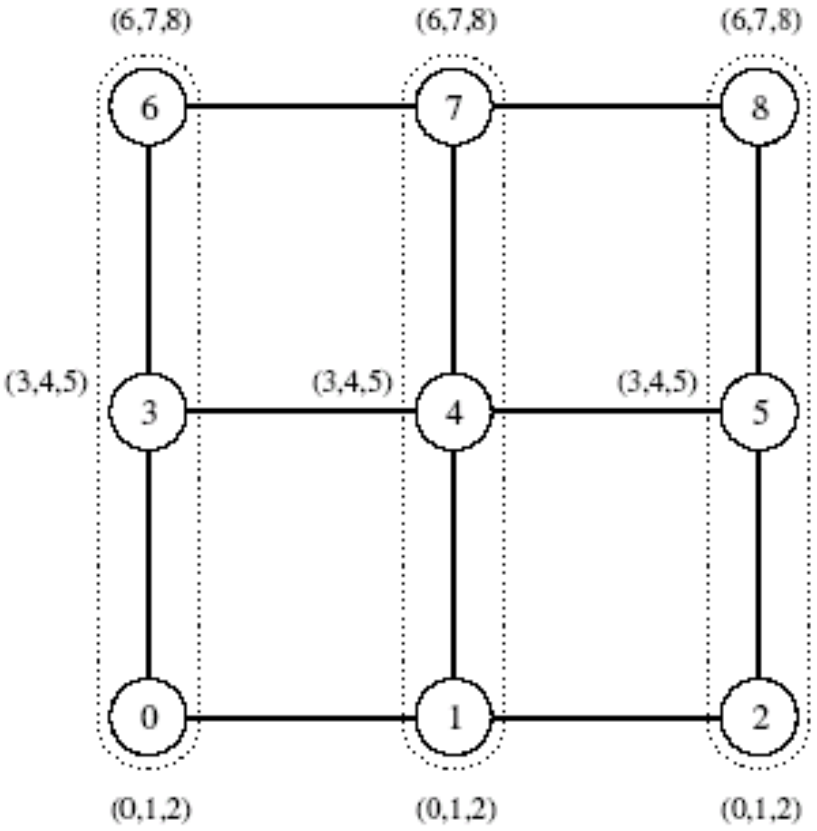
- Vykonané v dvoch fázach
- Prvá fáza – každý prvok v riadku vykoná „all-to-all broadcast“ využijúc procedúru uvedenú pre kruhovú topológiu
- Každý z uzlov príjme $\sqrt{p}$ správ od $\sqrt{p}$ uzlov v riadku
- Každý z uzlov tieto správy spracuje do jedinej správy veľkosti $m\sqrt{p}$
- Druhá fáza spočíva vo vykonaní „all-to-all broadcast“ po stĺpcoch

All-to-All Broadcast na mriežke

- All-to-All Broadcast na mriežke 3 x 3.



(a) Initial data distribution



(b) Data distribution after rowwise broadcast

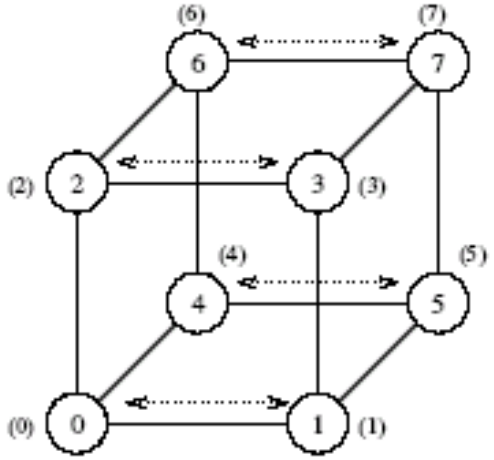
All-to-All Broadcast na mriežke

```
1.      procedure ALL_TO_ALL_BC_MESH(my_id, my_msg, p, result)
2.      begin
/* Communication along rows */
3.          left := my_id − (my_id mod √p)    (my_id − 1) mod √p;
4.          right := my_id − (my_id mod √p) − (my_id + 1) mod √p;
5.          result := my_msg;
6.          msg := result;
7.          for i := 1 to √p − 1 do
8.              send msg to right;
9.              receive msg from left;
10.             result := result ∪ msg;
11.         endfor;
/* Communication along columns */
12.         up := (my_id − √p) mod p;
13.         down := (my_id + √p) mod p;
14.         msg := result;
15.         for i := 1 to √p − 1 do
16.             send msg to down;
17.             receive msg from up;
18.             result := result ∪ msg;
19.         endfor;
20.     end ALL_TO_ALL_BC_MESH
```

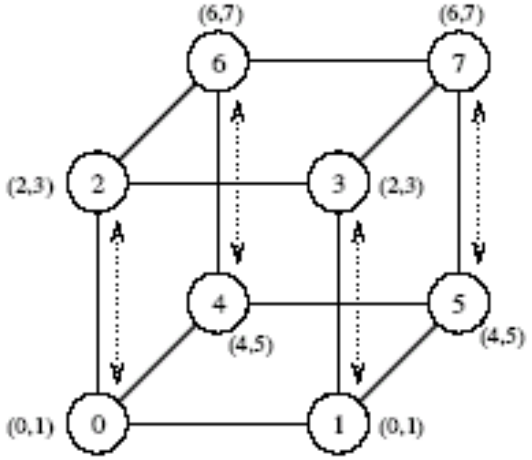
All-to-All Broadcast na hyperkocke

- Zovšeobecnenie algoritmu na mriežke na *log p* dimenzií
- Veľkosť správy sa zdvojnásobuje v každom s *log p* krokoch

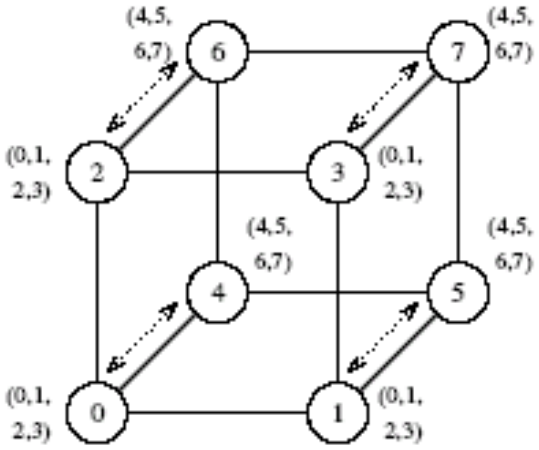
All-to-All Broadcast na hyperkocke



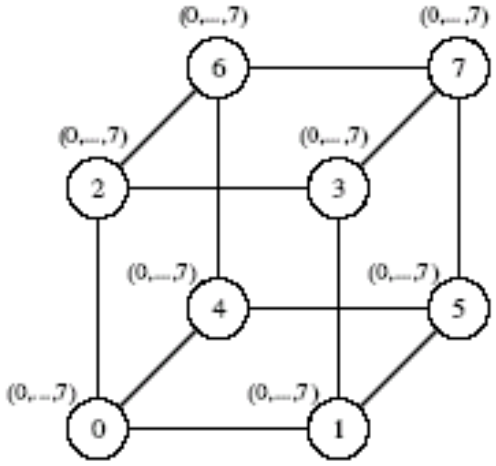
(a) Initial distribution of messages



(b) Distribution before the second step



(c) Distribution before the third step



(d) Final distribution of messages

All-to-All Broadcast na hyperkocke

```
1.    procedure ALL_TO_ALL_BC_HCUBE(my_id, my_msg, d, result)
2.    begin
3.        result := my_msg;
4.        for i := 0 to d − 1 do
5.            partner := my_id XOR 2^i;
6.            send result to partner;
7.            receive msg from partner;
8.            result := result ∪ msg;
9.        endfor;
10.   end ALL_TO_ALL_BC_HCUBE
```

All-to-All Redukction

- Podobná komunikačná štruktúra ako má „All-to-All Broadcast“, akurát operácie sú v opačnom poradí
- Po prijatí správy musí uzol kombinovať lokálnu správu s rovnakým adresátom ako má prijatá správa ešte pred jej preposlaním nasledujúcemu susednému uzlu

Analýza časovej náročnosti

- Na prstencovej topológií: $(t_s + t_w m)(p-1)$.
- Na mriežke: $2t_s(\sqrt{p} - 1) + t_w m(p-1)$.
- Na hyperkocke:

$$T = \sum_{i=1}^{\log p} (l_s + 2^{i-1} l_w m)$$

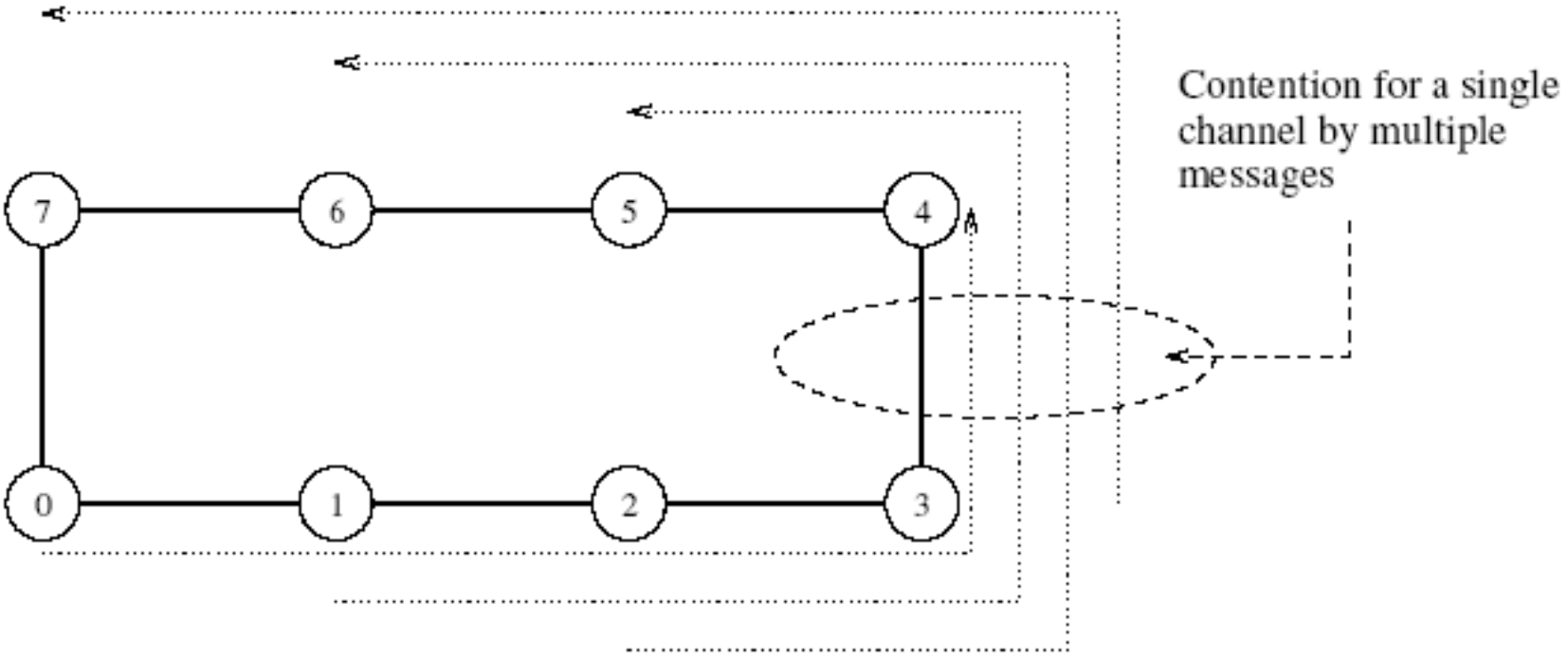
$$- l_s \log p + l_w m(p-1).$$

All-to-All Broadcast - poznámky

- Prezentované algoritmy sú asymptoticky opltimálne vzhľadom na veľkosť správy
- Nie je možné jednoducho adaptovať algoritmy pre vyššie dimenzie (ako napr. hyperkocka) do kruhovej topológie, lebo by to mohlo viesť k blokovaniu

All-to-All Broadcast - poznámky

- Blokovanie komunikačného kanála pri mapovaní hyperkocky na prstencovú topológiu



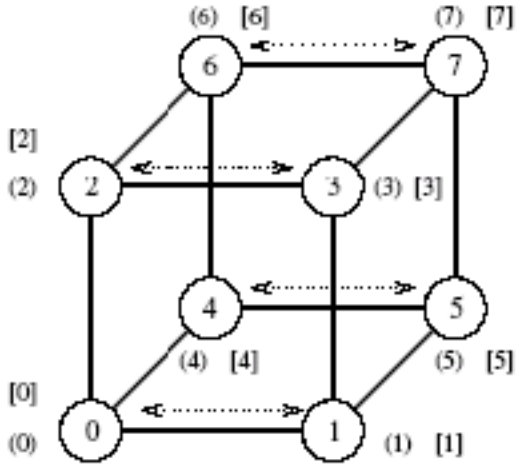
Operácie All-Reduce a Prefix-Sum

- All reduce operácia – každý uzol začína s vlastným bufrom o veľkosti *m* a výsledkom operácie sú identické bufre na každom uzle vytvorené z pôvodných *p* buffrov pomocou asociatívnej operácie
- Identické s „ All-to-One Redukciou“ nasledovanou One-to-All Broadcast-om – takto ale neefektívne, radšej využiť vzor „All-to-All Broadcast“, čas operácie je $(t_s + t_w m) \log p$
- Rozdielne oproti „All-to-All Reduction“, v ktorej sa vykonáva p simultánnych redukcíi, každá s iným cieľovým uzlom pre výsledok

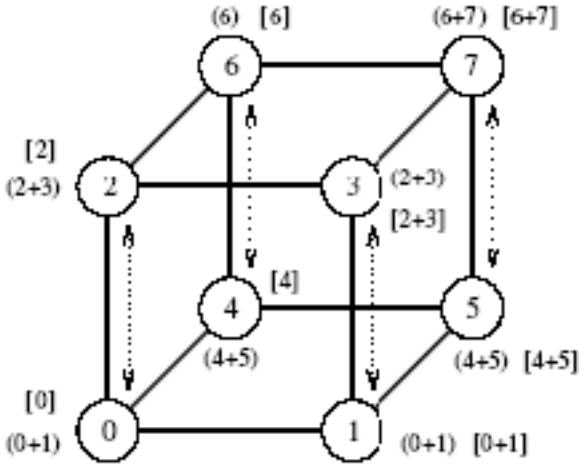
Operácia Prefix-Sum

- Majme p čísel $n_0, n_1, \ldots, n_{p-1}$ (každé číslo na inom uzle), cieľom je vypočítať súčty $s_k = \sum_{i=0}^{k} n_i$ pre všetky k medzi 0 and $p-1$.
- Číslo n_k je pôvodne situované na uzle k a po skončení operácie je na uzle umiestnený súčet S_k

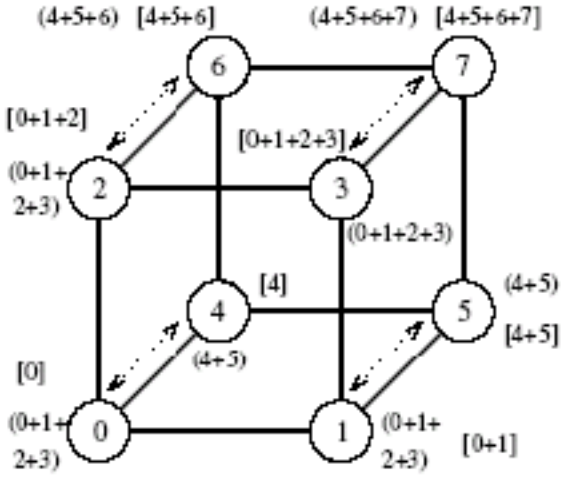
Operácia Prefix-Sum



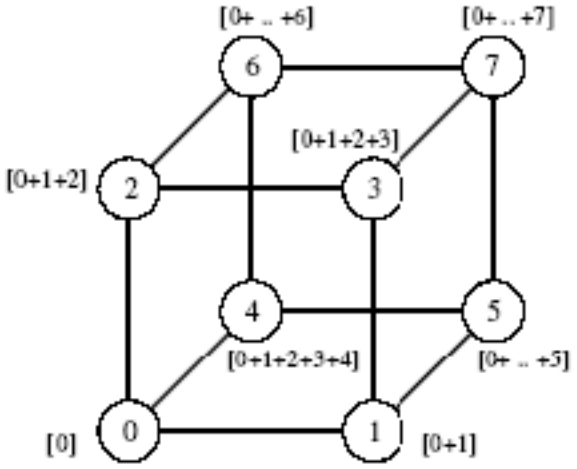
(a) Initial distribution of values



(b) Distribution of sums before second ste



(c) Distribution of sums before third step



(d) Final distribution of prefix sums

Operácia Prefix-Sum

- Operácia môže byť implementovaná pomocou rovnakého jadra operácie all-to-all broadcast
- Je potrebné zohľadniť že operácia „prefix-sum“ s číslom uzla *k* využíva iba informáciu z podmnožiny uzlov z id menším ako *k*
- Implementácia využíva dočasný bufer a obsah prichádzajúcej správy je skombinovaný s bufrom iba keď správa bola odoslaná z uzla z menším id ako má prijímajúci uzol
- Obsah odchádzajúcich správ je upravený s každou prichádzajúcou správou

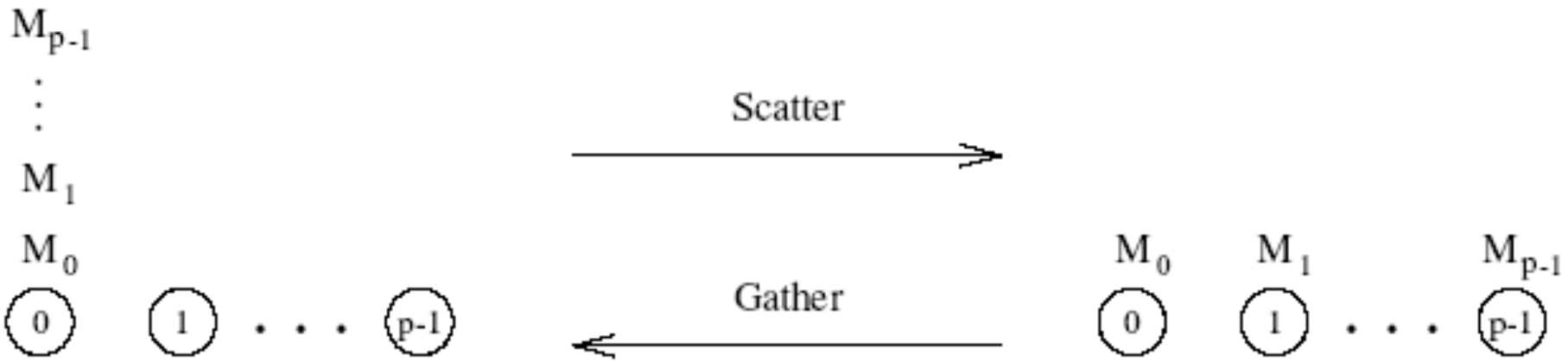
Operácia Prefix-Sum na hyperkocke

```
1.   procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
2.   begin
3.       result := my_number;
4.       msg := result;
5.       for i := 0 to d − 1 do
6.           partner := my_id XOR 2^i;
7.           send msg to partner;
8.           receive number from partner;
9.           msg := msg + number;
10.          if (partner < my_id) then result := result + number;
11.      endfor;
12.  end PREFIX_SUMS_HCUBE
```

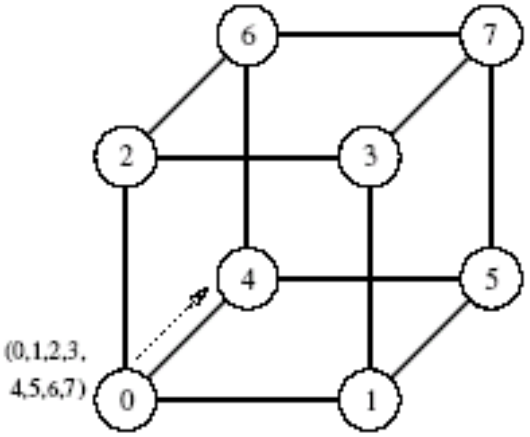
Scatter a Gather operácie (Rozptýlenie a zhromaždenie)

- Operácia scatter – jediný uzol pošle špecifickú správu veľkosti m každému uzlu
- Operácia typu „osobná komunikácia jedného ostatným“
- Operácia gather – jediný uzol zozbiera špecifické správy od každého uzla
- Scatter – odlišná operácia ako broadcast, ale podobná algoritmická štruktúra, ale rozdielne veľkosti správ (scatter – veľkosť správ sa zmenšuje)
- Gather operácia je inverznou operáciou ku scatter

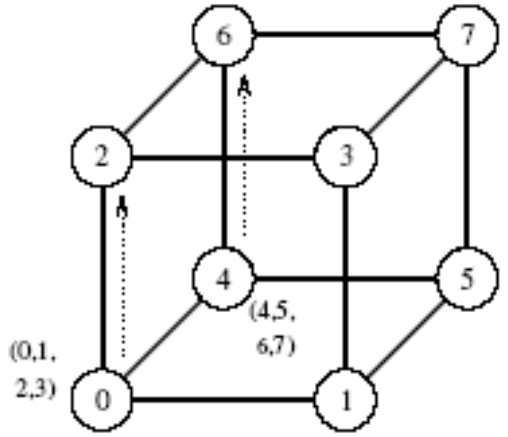
Scatter a Gather operácie



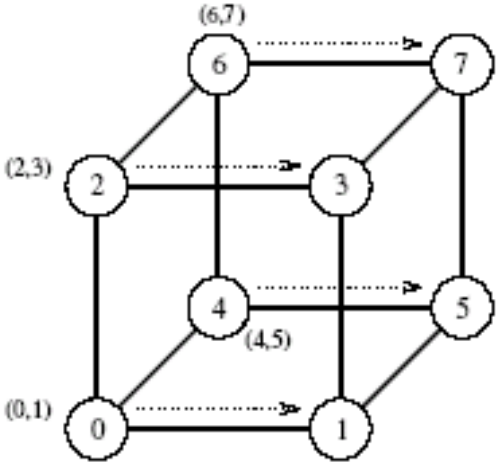
Scatter a Gather operácie



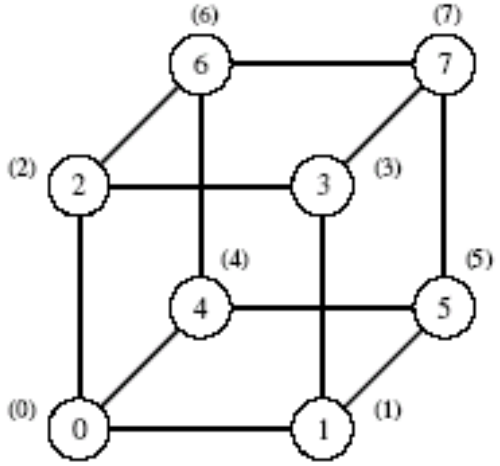
(a) Initial distribution of messages



(b) Distribution before the second step



(c) Distribution before the third step



(d) Final distribution of messages

Časová náročnosť scatter a gather operácií

- Počet krokov je *log p*, v každom kroku sa redukuje veľkosť systému aj veľkosť údajov
- Čas operácie:

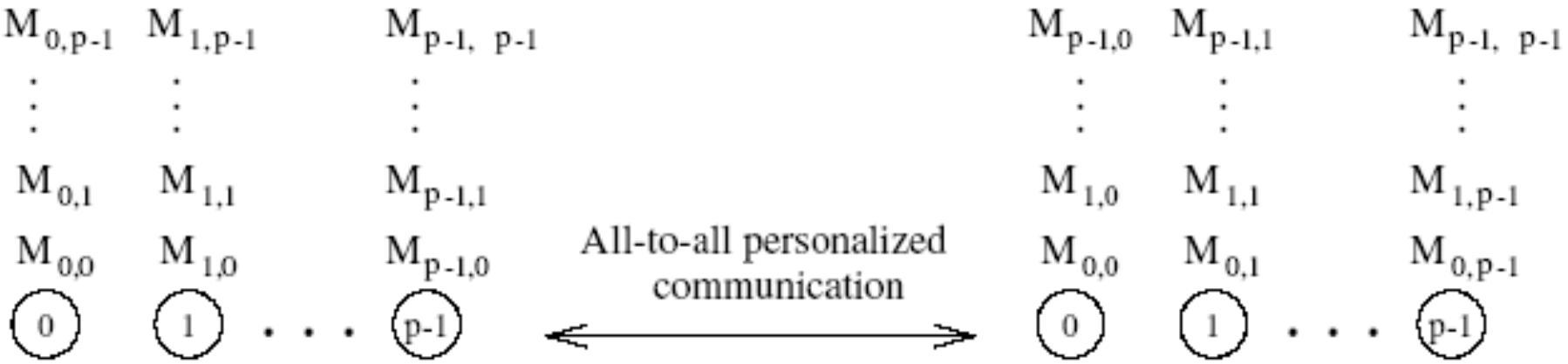
$$T = t_s \log p + t_w m(p-1).$$

- Čas platí pre kruhovú topológiu aj 2D mriežku.
- Tieto časy sú asymptoticky optimálne vzhľadom na veľkosť správy.

Osobná komunikácia každý s každým

- Každý uzol ma špecifickú správu veľkosti *m* pre každý ostatný uzol
- Iné ako all-to-all broadcast operácia, v ktorej každý uzol komunikuje tú istú správu ostatným uzlom
- Osobná komunikácia každý s každým sa tiež nazýva úplná výmena (Total Exchange)

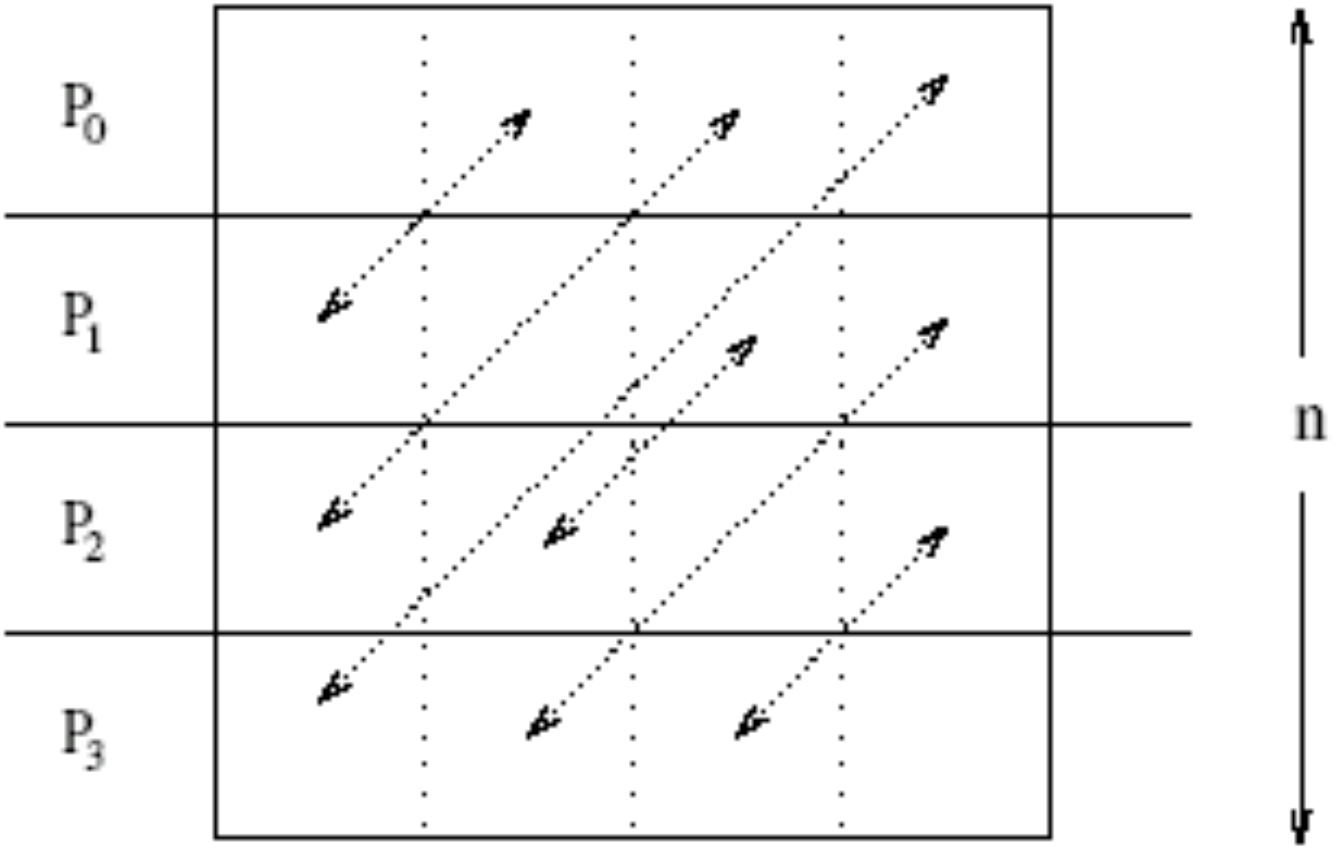
Osobná komunikácia každý s každým



Osobná komunikácia každý s každým - príklad

- Problém transponovania matice
- Každý procesor vlastní celý riadok matice
- Operácia transponovania je identická s osobnou komunikáciou každý s každým

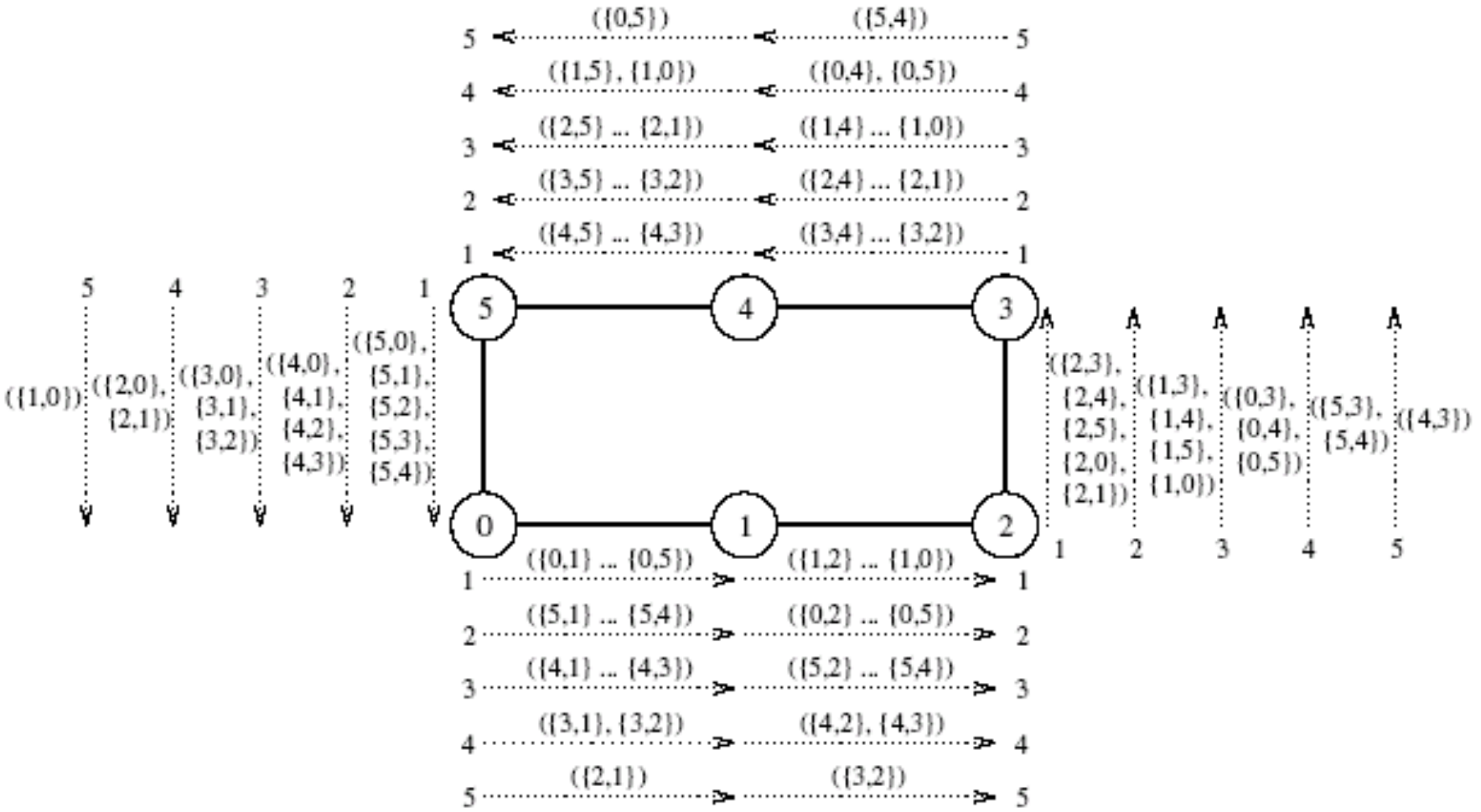
Osobná komunikácia každý s každým - príklad



Osobná komunikácia každý s každým – na prstencovej topológií

- Každý uzol zašle všetky prvky (časti správy) ako jednu konsolidovanú správ o veľkosti *m (p-1)* všetkým susedom
- Každý uzol si zo správy vyberie prvok určený pre neho a prepošle zvyšných *(p-2)* prvkov o veľkosti *m* každý ďalšiemu uzlu
- Algoritmus končí po p-1 krokoch
- Veľkosť správy sa zmenšuje o *m* v každom kroku

Osobná komunikácia každý s každým – na prstencovej topológií



Osobná komunikácia každý s každým na prstencovej topológií – čas. nároč.

- Celkovo *p – 1* krokov
- V každom kroku *i*, je veľkosť správy *m(p – i)*.
- Celkový čas:

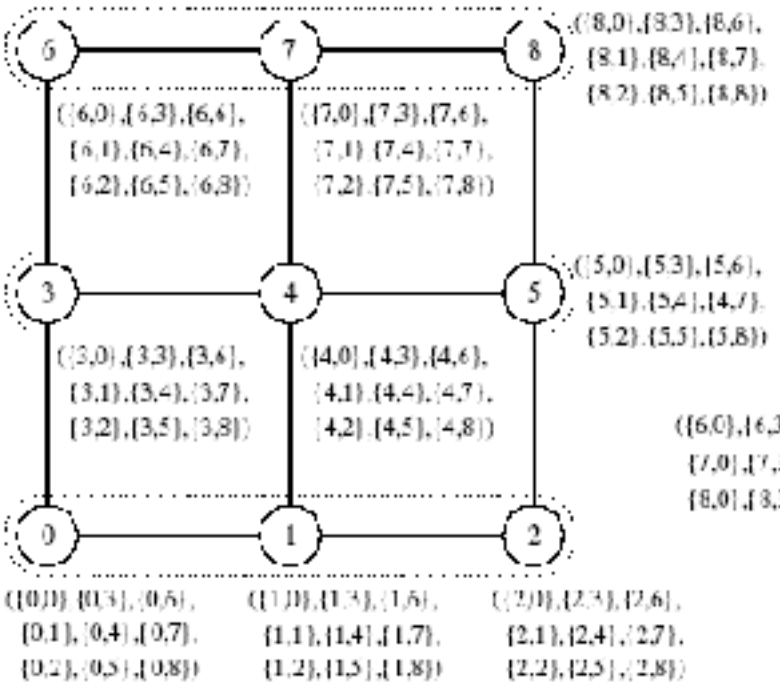
$$
\begin{aligned}
T &= \sum_{i=1}^{p-1} (t_s + t_w m(p-i)) \\
&= t_s(p-1) + \sum_{i=1}^{p-1} i t_w m \\
&= (t_s + t_w mp/2)(p-1).
\end{aligned}
$$

- Člen t_w môže byť znížený dvojnásobne komunikovaním správ v oboch smeroch

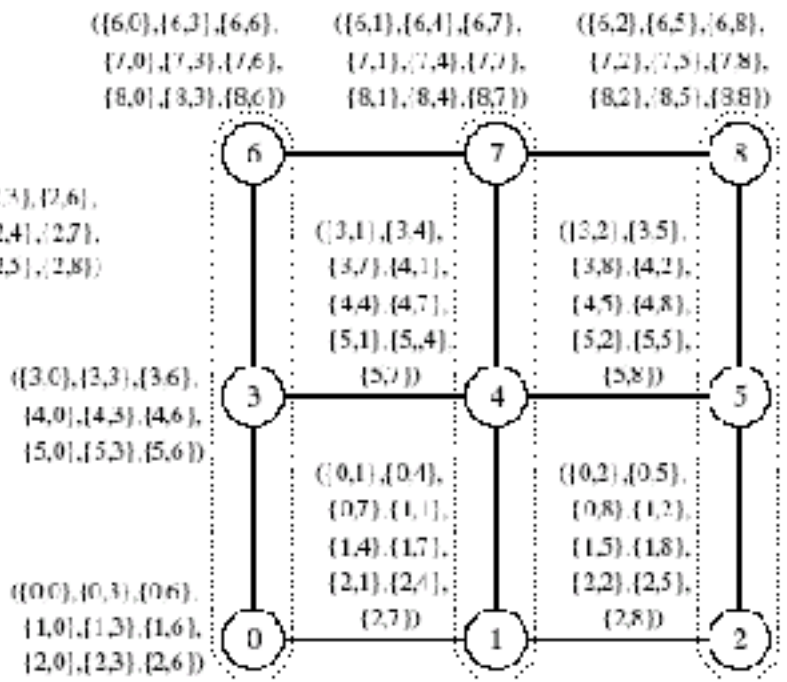
Osobná komunikácia každý s každým na mriežke

- Každý uzol najskôr zoskupí svojich p správ podľa stĺpcov prijímajúcich uzlov
- Osobná komunikácia je vykonaná nezávisle v každom riadku so zoskupenými správami o veľkosti $m\sqrt{p}$
- Správy v každom uzle sú potom opäť zoskupené teraz podľa riadkov prijímajúcich uzlov
- Osobná komunikácia je aj teraz vykonaná nezávisle v každom stĺpci so zoskupenými správami o veľkosti $m\sqrt{p}$

Osobná komunikácia každý s každým na mriežke



(a) Data distribution at the beginning of first phase



(b) Data distribution at the beginning of second phase

Osobná komunikácia každý s každým na mriežke – časová náročnosť

- Časová náročnosť prvej fázy je identická s čas. náročnosťou v prstencovej topológií s √p procesormi:
  $(t_s + t_w mp/2)(\sqrt{p} - 1)$

- Časová náročnosť druhej fázy je identická s prvou fázou
- Celkový čas je teda:

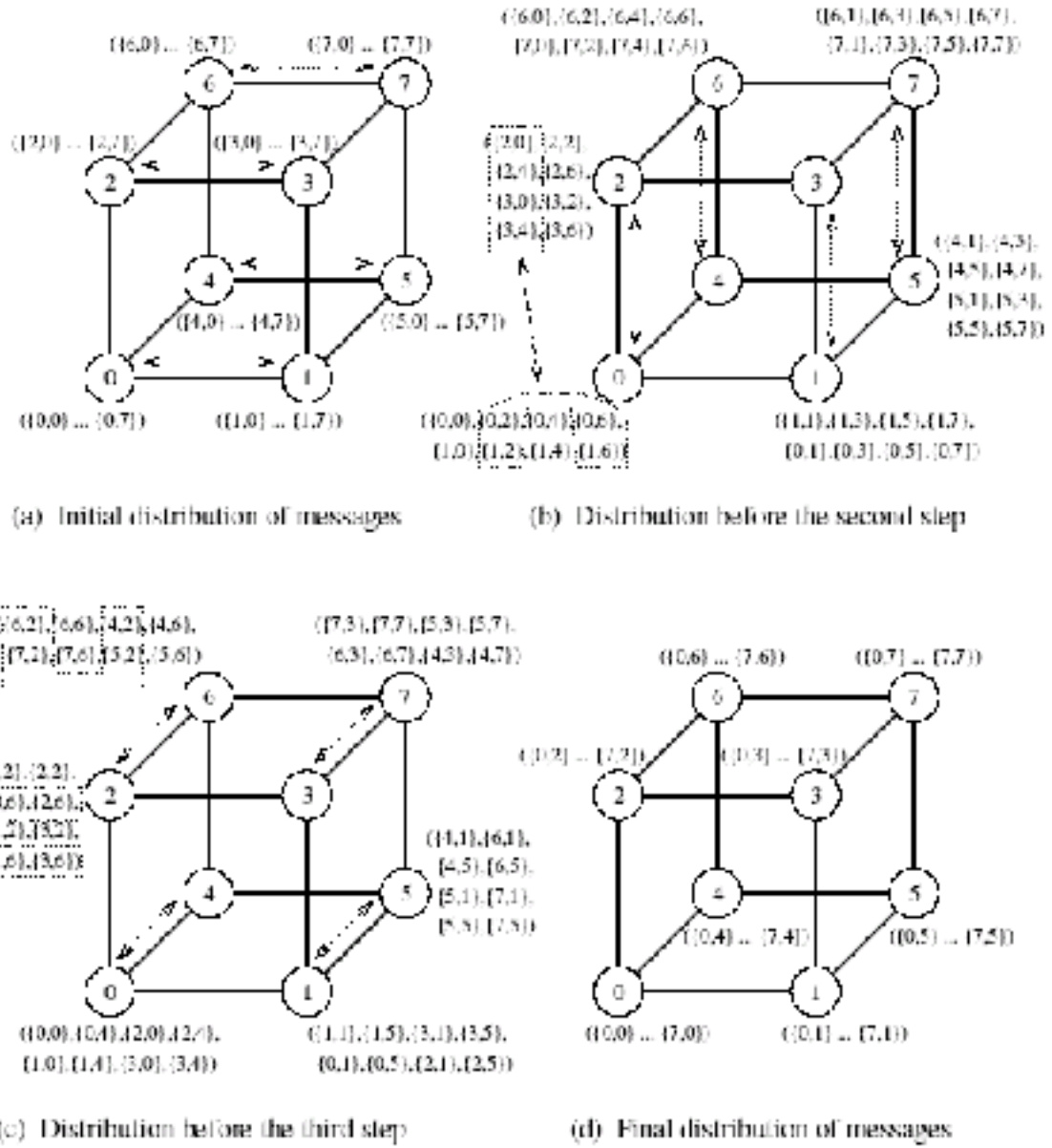
$$T = (2t_s + t_w mp)(\sqrt{p} - 1).$$

- Čas preusporiadania správ je výrazne menší ako čas komunikácie.

Osobná komunikácia každý s každým na hyperkocke

- Zovšeobecnenie algoritmu na mriežke na *log p* krokov
- V každej fáze algoritmu osobnej komunikácie každý uzol obsahuje *p* paketov veľkosti *m* každý
- Počas komunikácie v danom smere každý uzol pošle *p/2* týchto paketov (zoskupených ako jedna správa)
- Uzol musí lokálne preskupiť svoje správy ešte pred log p komunikačnými krokmi

Osobná komunikácia každý s každým na hyperkocke



Osobná komunikácia každý s každým na hyperkocke – časová náročnosť

- *log p* iterácií
- *mp/2* slov komunikovaných v každej iterácií
- Celková cena komunikácie:

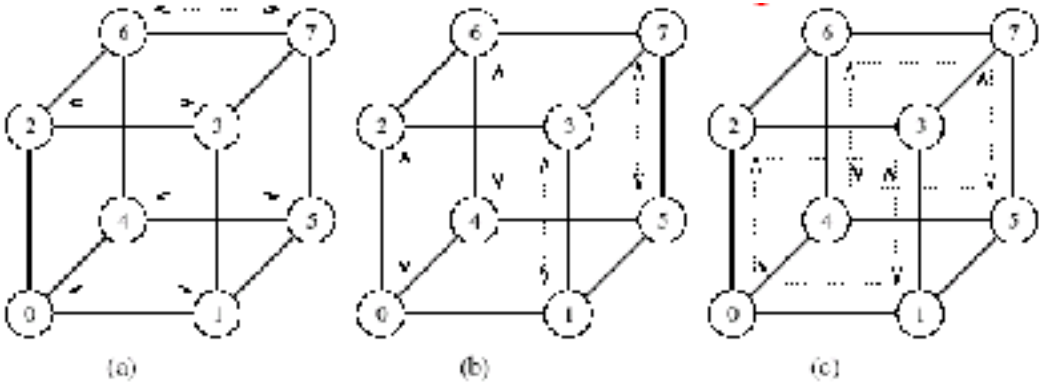
$$T = (t_s + t_w mp/2) \log p.$$

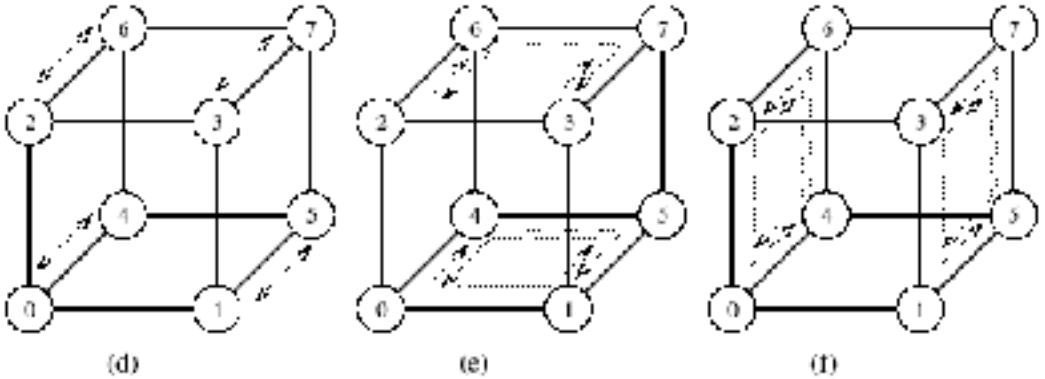
- Nie je to optimálne!

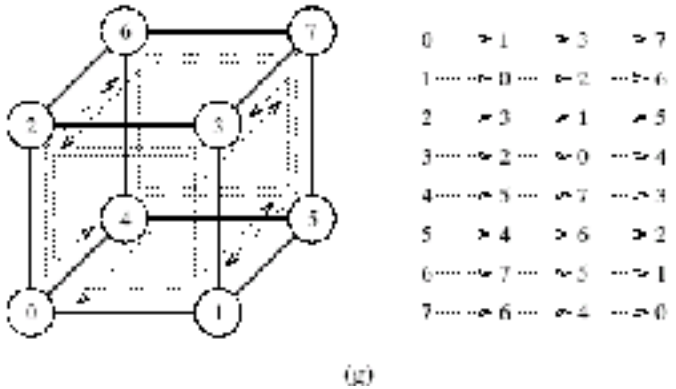
Osobná komunikácia každý s každým na hyperkocke – optimálny algoritmus

- Každý uzol vykoná *p – 1* krokov komunikácie, počas ktorých si v každom kroku vymení m slov s iným uzlom
- Uzol si musí vybrať takého komunikačného partnera v každom kroku, aby komunikačný kanál nebol zahltený
- V *j*-tom komunikačnom kroku si uzol *i* vymení údaje s uzlom *(i XOR j)*
- Pri takomto plánovaní sú všetky komunikačné kanály nezahltené a žiadny z obojsmerných kanálov neprenáša viac ako jednu správu daným smerom

Osobná komunikácia každý s každým na hyperkocke







Osobná komunikácia každý s každým na hyperkocke – optimálny algoritmus

```
1.    procedure ALL_TO_ALL_PERSONAL(d, my_id)
2.    begin
3.        for i := 1 to 2^d − 1 do
4.        begin
5.            partner := my_id XOR i;
6.            send M_{my_id,partner} to partner;
7.            receive M_{partner,my_id} from partner;
8.        endfor;
9.    end ALL_TO_ALL_PERSONAL
```

Osobná komunikácia každý s každým na hyperkocke – čas. nár. opt. alg.

- *p – 1* krokov
- *m* slov v každom kroku.
- Časová náročnosť:

$$T = (t_s + t_w m)(p - 1).$$

- Asymptoticky optimálne vzhľadom na veľkosť správy.

Cyklické posunutie

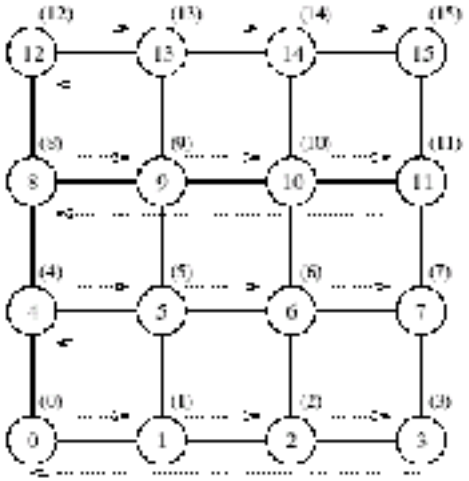
- Špeciálna permutácia, pri ktorej uzol *i* pošle paket uzlu *(i + q)* mod *p* v skupine s *p* uzlami *(0 ≤ q ≤ p)*.

Cyklické posunutie na mriežke

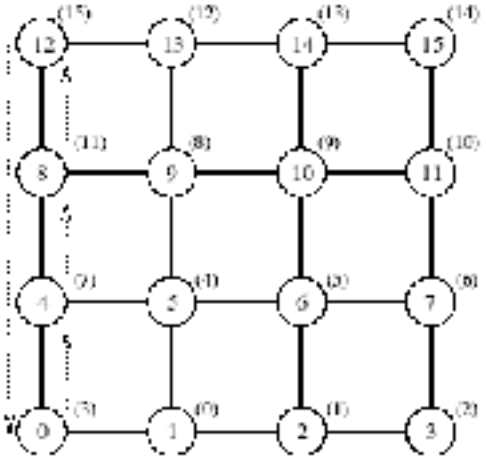
- Implementácia na kruhovej topológií je pomerne intuitívna a môže byť vykonaná v min*{q,p – q}* krokov komunikácie medzi susednými uzlami
- Algoritmus na mriežke vychádza z tejto myšlienky: vykonáme posun v jednom smere (všetky porcesy) a následne v druhom smere
- Horné ohraničenie času komunikácie:

$$T = (t_s + t_w m)(\sqrt{p} + 1).$$

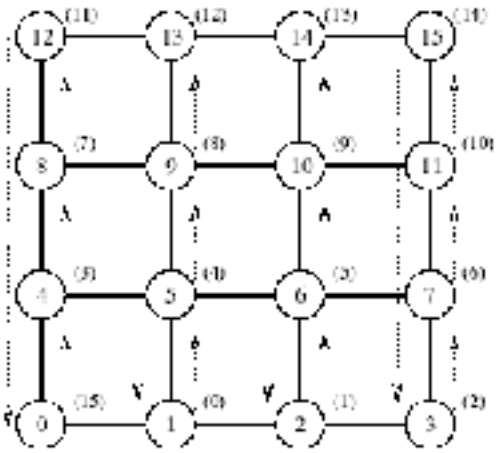
Cyklické posunutie na mriežke



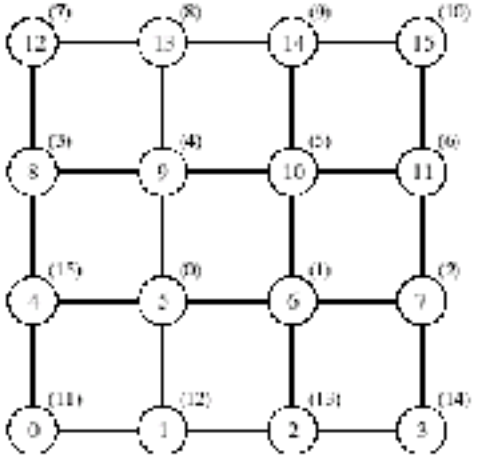
(a) Initial data distribution and the first communication step



(b) Step to compensate for backward row shift



(c) Column shifts in the third communication step



(d) Final distribution of the data

Cyklické posunutie na hyperkocke

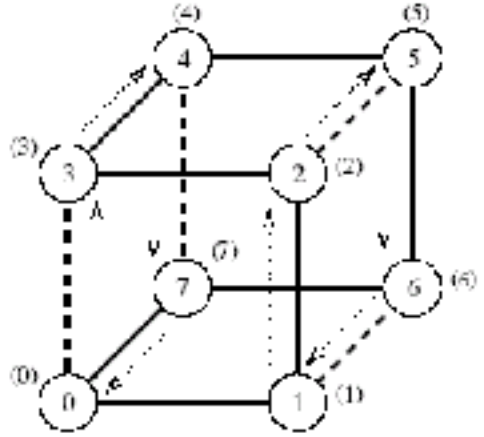
- Namapovanie lineárneho zoznamu s 2^d uzlami na d-dimenzionálnu hyperkocku
- Pre vykonanie posunutia s veľkosťou q rozložíme q na súčet rôznych mocnín 2.
- Ak q je súčet s rôznych mocnín 2, potom cyk. posunutie s veľkosťou q na hyperkocke je vykonané v s fázach.
- Celkový čas je zhora ohraničený:

$$T = (t_s + t_w m)(2 \log p - 1).$$

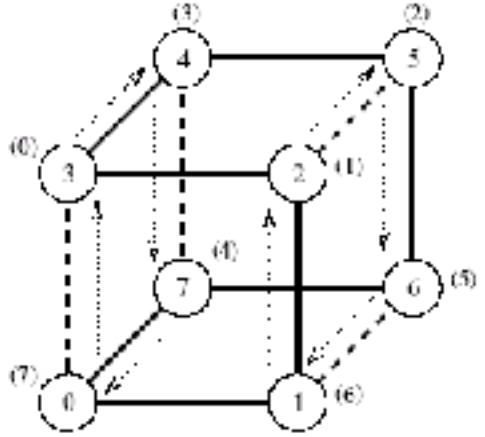
- Ak je použité E-kubické smerovanie, tento čas môže byť zredukovaný na:

$$T = t_s + t_w m.$$

Cyklické posunutie na hyperkocke

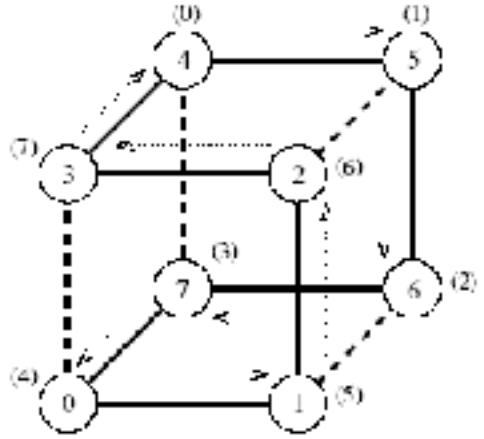


First communication step of the 4-shift

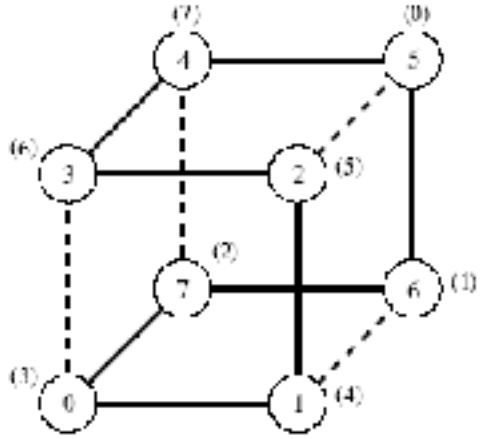


Second communication step of the 4-shift

(a) The first phase (a 4-shift)

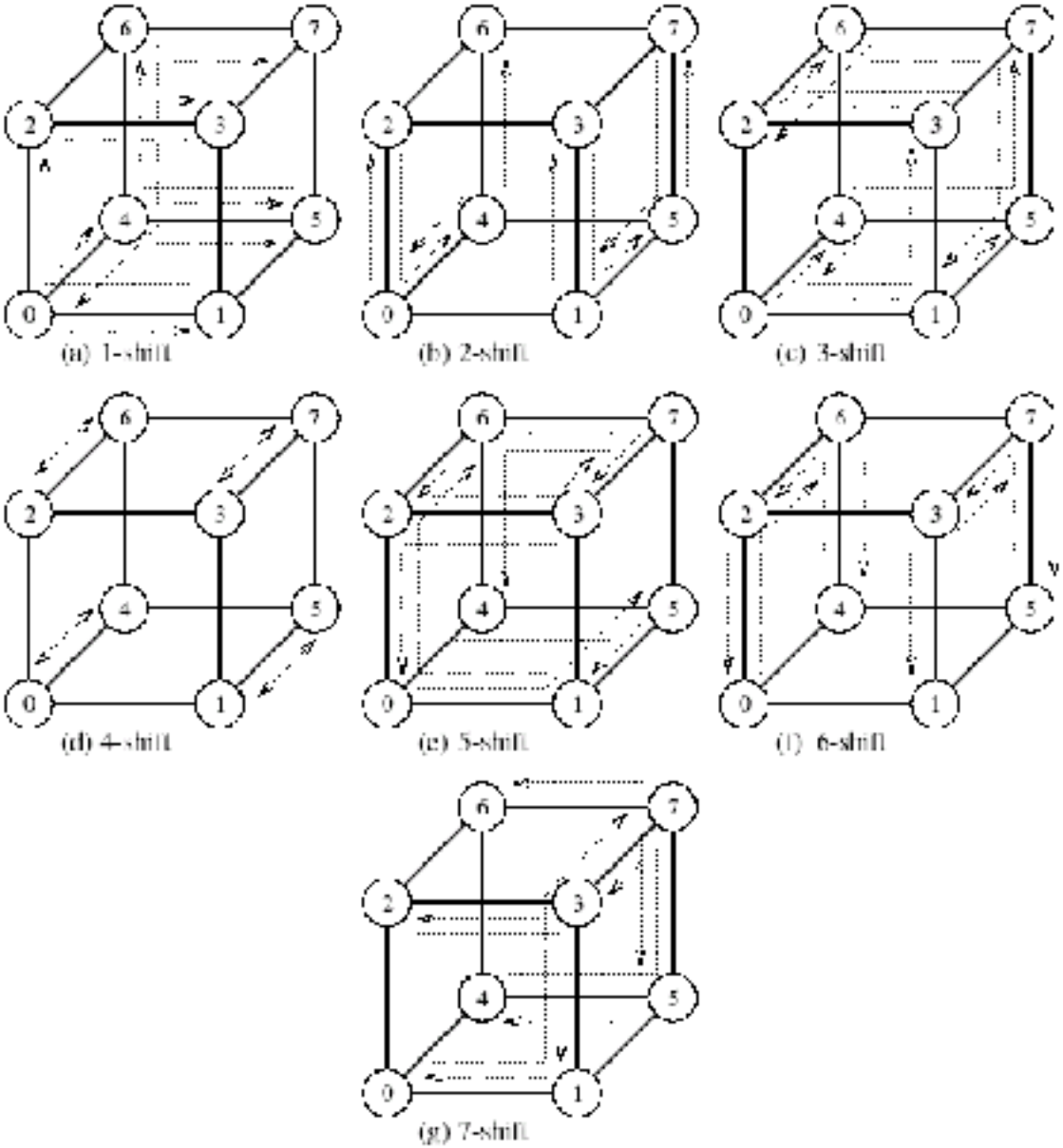


(b) The second phase (a 1-shift)



(c) Final data distribution after the 5-shift

Cyklické posunutie na hyperkocke



Urýchlenie niektorých komunikačných operácií

- Oddelenie a smerovanie správ na časti: ak môže byť správa rozdelená na *p* častí, one-to-all broadcast môže byť implementovaný ako operácia scatter nasledovaná all-to-all broadcast operáciou. Časová náročnosť je potom:

$$\begin{aligned} T &= 2 \times (t_s \log p + t_w (p-1)\frac{m}{p}) \\ &\approx 2 \times (t_s \log p + t_w m). \end{aligned}$$

- All-to-one redukcia môže byť vykonaná ako all-to-all redukcia (duálny problém k all-to-all broadcast) nasledovaná gather operáciou (duálny problém ku scatter operácií).

Zdroje

- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar. Introduction to ParallelComputing, 2nd Edition, Addison-Wesley 2003„Introduction to Parallel Computing“ http://www-users.cs.umn.edu/~karypis/parbook/

- Obrázky prevzaté z:
  - Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar. Introduction to ParallelComputing, 2nd Edition, Addison-Wesley 2003„Introduction to Parallel Computing“ http://www-users.cs.umn.edu/~karypis/parbook/