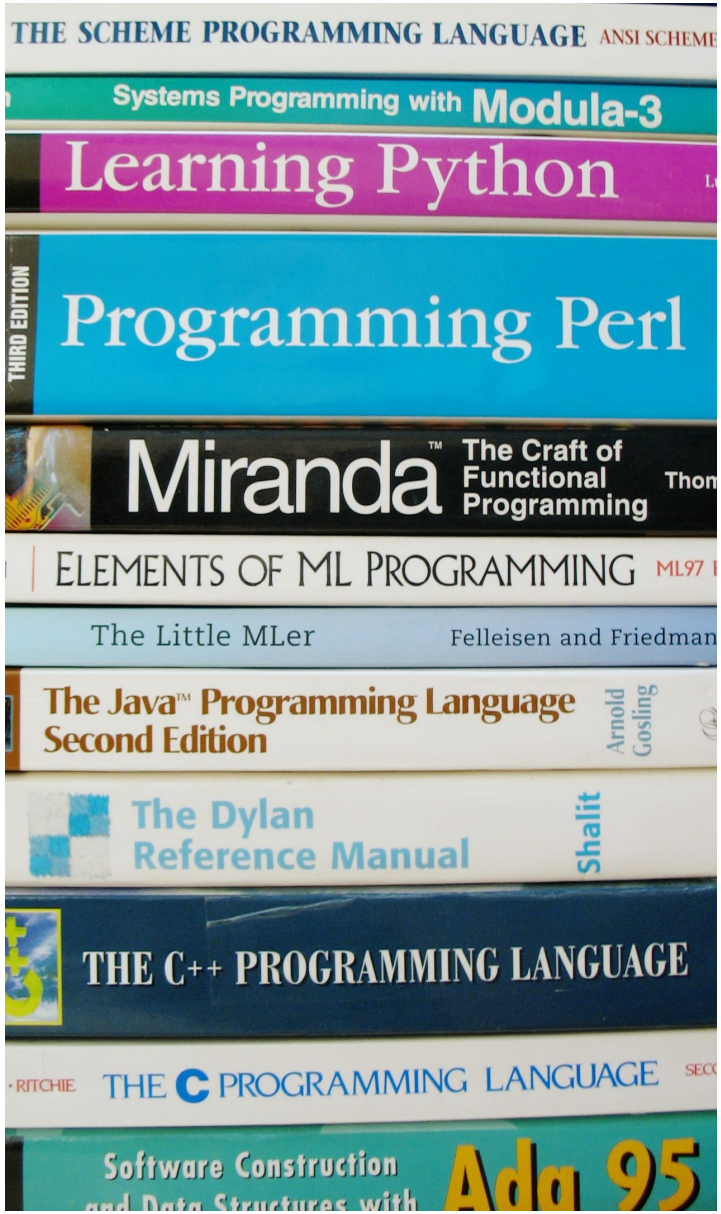
IV100 Distribuované výpočty

Rasťo Královič

Katedra informatiky, FMFI UK Bratislava
kralovic@dcs.fmph.uniba.sk

- **Gerard Tel:** *Introduction to Distributed Algorithms*, Cambridge University Press, 2000, ISBN 0521794838
- **Nancy Lynch:** *Distributed Algorithms*, Morgan Kaufmann Publishers, 1996, ISBN 1558603484
- **Frank Thomson Leighton:** *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, 1991, ISBN 1558601171

sekvenčné programovacie jazyky



- rôzne paradigmy, syntax, ...
- rôzne použitie
- interpreter (simulácia)
- z pohľadu počítania sú všetky "rovnako silné"
- jednoduchý abstraktný model (návrh algoritmov)

RAM – jednoduchý "abstraktný počítač"

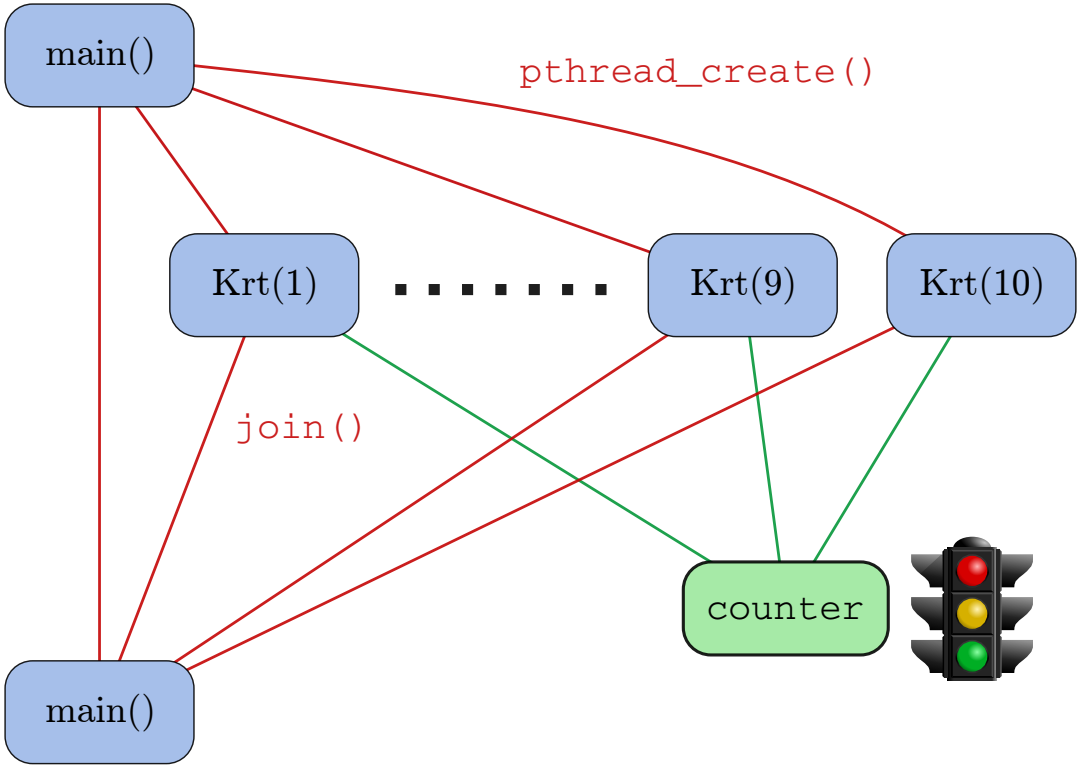
- registre $r_0, r_1, r_2, r_3, \ldots$
- vstup $a_1, a_2, a_3, \ldots$
- program = postupnosť inštrukcií
- pozícia v programe

inštrukcie:

priradenie	$r_X := r_Y$	X, Y je konštanta alebo register
	$r_X := a_Y$	
	$r_X := c$	c je konštanta
výpočet	$r_X := r_Y \square r_Z$	$\square$ je $+, -, *, /$
skok	**goto** i	
podmienka	**if** $r_X \begin{matrix} < \\ \leq \\ = \\ \geq \\ > \end{matrix} r_Y$ **then goto** i	
	if $r_X \begin{matrix} < \\ \leq \\ = \\ \geq \\ > \end{matrix} c$ **then goto** i	

zdieľaná pamäť – thready

- dynamické vytváranie
- asynchrónne (`join`)
- riadenie prístupu (mutex/semafór/...)



thready (POSIX)

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

pthread_mutex_t mutex1 =
    PTHREAD_MUTEX_INITIALIZER;
int  counter = 0;


void *Krt(void *p) {
  pthread_mutex_lock( &mutex1 );
  counter += *(int *)p;
  pthread_mutex_unlock( &mutex1 );
}
```

```
main()  {
  pthread_t thread_id[NTHREADS];
  int i, j, param[NTHREADS];

  for(i=0; i < NTHREADS; i++) {
    param[i]=i;
    pthread_create( &thread_id[i],
      NULL, Krt, param + i );
  }

  for(j=0; j < NTHREADS; j++)
    pthread_join( thread_id[j], NULL);
}
```

thready (POSIX)

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

pthread_mutex_t mutex1 =
    PTHREAD_MUTEX_INITIALIZER;
int  counter = 0;


void *Krt(void *p) {
  pthread_mutex_lock( &mutex1 );
  counter += *(int *)p;
  pthread_mutex_unlock( &mutex1 );
}
```

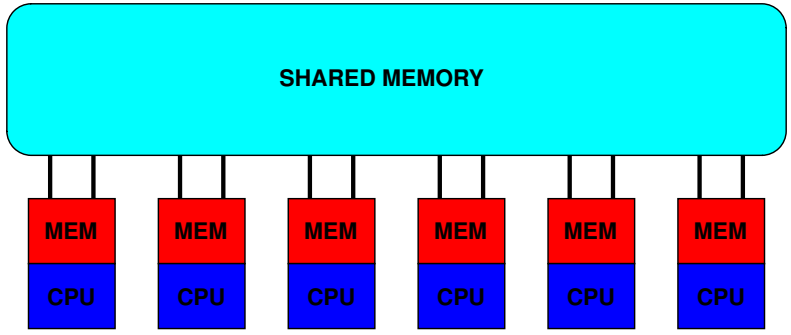
```
main()  {
  pthread_t thread_id[NTHREADS];
  int i, j, param[NTHREADS];

  for(i=0; i < NTHREADS; i++) {
    param[i]=i;
    pthread_create( &thread_id[i],
      NULL, Krt, param + i );
  }

  for(j=0; j < NTHREADS; j++)
    pthread_join( thread_id[j], NULL);
}
```

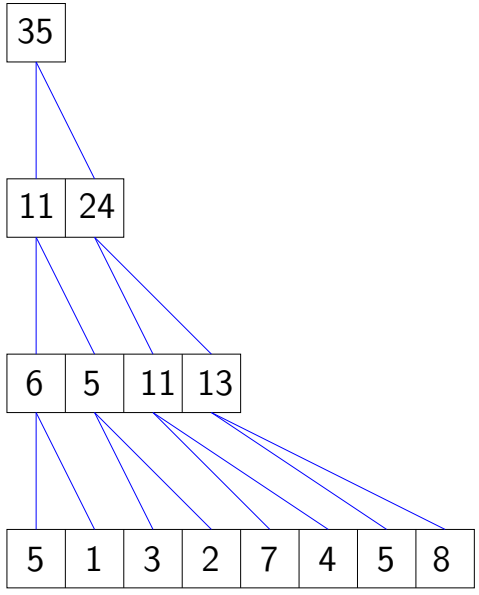
- deadlock
- synchronizácia
- debugovanie

PRAM



- synchrónne procesory
- kedy je problém paralelizovateľný?

```
global read( A(i), a )
global write( a, B(i) )
for h = 1 to log n do
    if (i ≤ n/2^h)
       global read( B(2i − 1),x )
       global read( B(2i),y )
       z := x + y
       global write(z, B(i))
if i = 1 global write( z,S )
```



WTF – Work-Time Framework

- písať algoritmy pre ľubovoľný počet procesorov
- **work** – počet použitých operácií

for $l \leq i \leq u$ **pardo** statement

```
for 1 ≤ i ≤ n pardo
    B(i) := A(i)
for h = 1 to log n do
    for 1 ≤ i ≤ n/2^h pardo
        B(i) := B(2i − 1) + B(2i)
S := B(1)
```

```
global read( A(i), a )
global write( a, B(i) )
for h = 1 to log n do
    if (i ≤ n/2^h)
        global read( B(2i − 1),x )
        global read( B(2i),y )
        z := x + y
        global write(z, B(i))
if i = 1 global write( z,S )
```

WT algoritmus s $T(n)$, $W(n)$ sa dá simulovať na PRAMe s p procesormi v čase $\left\lfloor \frac{W(n)}{p} \right\rfloor + T(n)$

PRAM

- EREW / CREW / CRCW
- príklad: pozícia prvej jednotky na common CRCW PRAM

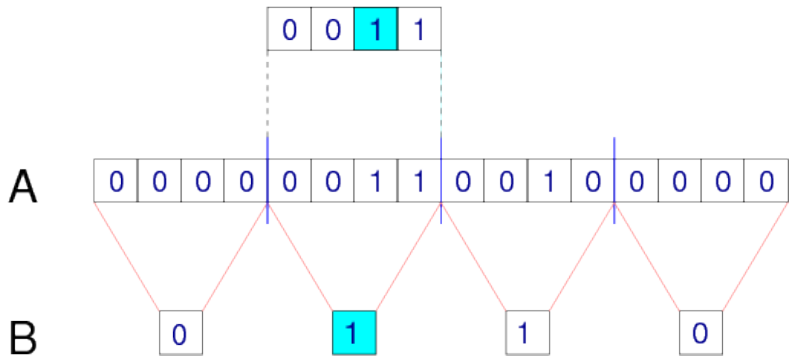
```
for i = 1 to n pardo B(i) := 0
for i = 1 to n^2 pardo
    if A(i) = 1 then B(⌈i/n⌉) := 1
```

```
for i = 1 to n pardo
    for j = 1 to i − 1 pardo
        if B(j) = 1 then B(i) := 0
```

```
for i = 1 to n pardo
    if B(i) = 1 then D := i − 1
```

```
for i = 1 to n pardo
    for j = 1 to i − 1 pardo
        if A(D + j) = 1 then
            A(D + i) := 0
```

```
for i = 1 to n pardo
    if A(D + i) = 1 then X := D + i
```



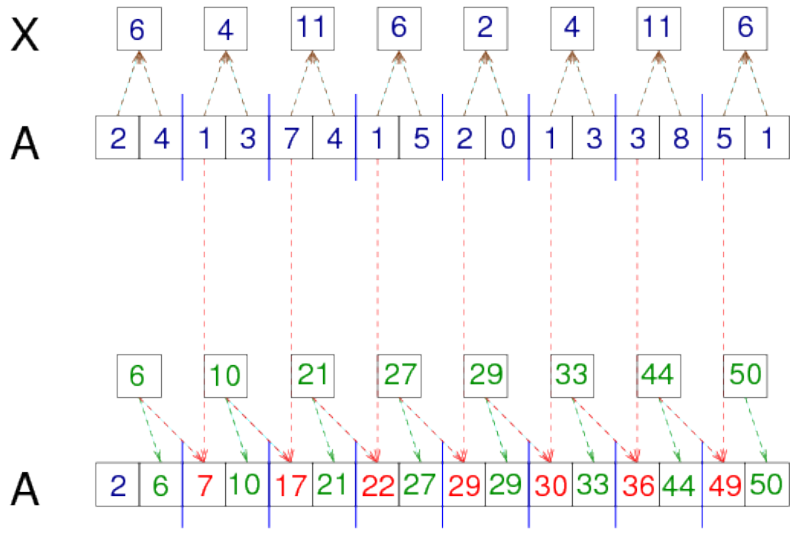
Prefix Sum Problem

Dané pole A dĺžky n. Pre každý index i nájsť $\sum_{j=0}^{i} a_i$

```
for 1 ≤ i ≤ n/2 pardo
    x_i := a_{2i-1} + a_{2i}

z :=prefix_sums( X, n/2 )

for 1 ≤ i ≤ n/2 pardo
    a_i := { z_i/2          ak i = 2k
           { 1              ak i = 1
           { z_{(i-1)/2} + a_i   ak i = 2k+3
```



práca $= O(n)$
čas $= O(\log n)$

bez zdieľanej pamäte – procesy

```
void do_child(int data_pipe[]) {
  int c,rc;

  close(data_pipe[1]);
  while ((rc = read(data_pipe[0], &c, 1)) > 0)
    putchar(c);
  exit(0);
}
```

```
void do_parent(int data_pipe[]) {
  int c,rc;

  close(data_pipe[0]);
  while ((c = getchar()) > 0)
    rc = write(data_pipe[1], &c, 1);
  close(data_pipe[1]);
  exit(0);
}
```

```
int main() {
  int data_pipe[2];
  int pid,rc;

  rc = pipe(data_pipe);
  pid = fork();
  switch (pid) {
    case 0:
      do_child(data_pipe);
    default:
      do_parent(data_pipe);
  }
}
```

v sieti: posielanie správ (sokety)

server

```
int main(int argc, char *argv[]) {
    int sockfd, newsockfd, portno;
    socklen_t clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);

    bind(sockfd, (struct sockaddr *) &serv_addr,
        sizeof(serv_addr));
    listen(sockfd,5);
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd,
        (struct sockaddr *) &cli_addr, &clilen);
    bzero(buffer,256);

    n = read(newsockfd,buffer,255);
    printf("Here is the message: %s\n",buffer);

    n = write(newsockfd,"I got your message",18);
    close(newsockfd);
    close(sockfd);
 }
```

klient

```
int main(int argc, char *argv[]) {
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[256];

    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server = gethostbyname(argv[1]);

    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
    (char *)&serv_addr.sin_addr.s_addr,
    server->h_length);
    serv_addr.sin_port = htons(portno);

    connect(sockfd,(struct sockaddr *) &serv_addr,
        sizeof(serv_addr));

    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    printf("%s\n",buffer);
    close(sockfd);
    return 0;
}
```

v sieti: posielanie správ (MPI)

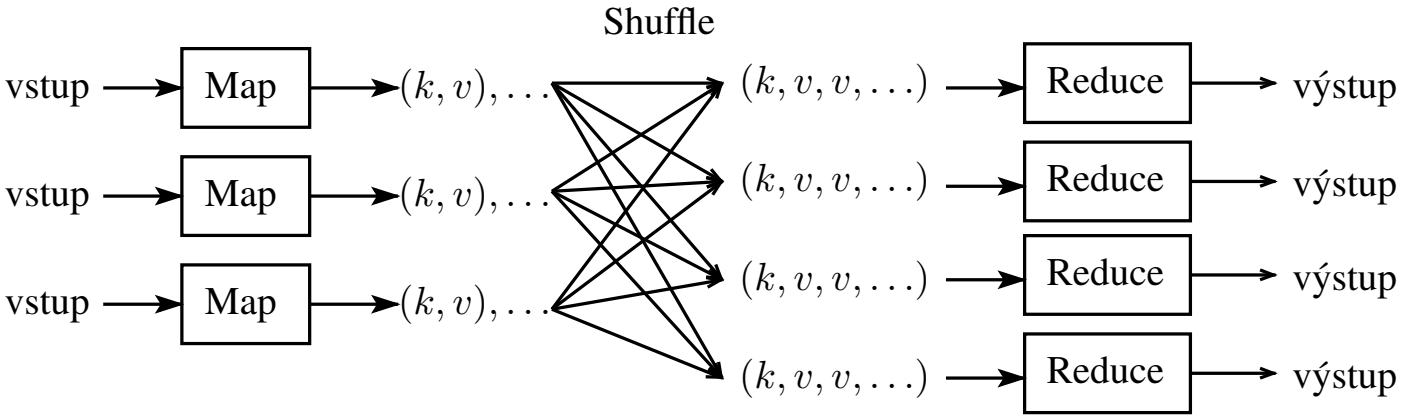
```
int main(int argc, char *argv[]) {
    const int tag = 47;
    ...
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ntasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);

    if (id == 0) {
      MPI_Get_processor_name(msg, &length);
      printf("Hello World from process %d running on %s\n", id, msg);
      for (i=1; i<ntasks; i++) {
        MPI_Recv(msg, 80, MPI_CHAR, MPI_ANY_SOURCE,
                      tag, MPI_COMM_WORLD, &status);
        source_id = status.MPI_SOURCE;
        printf("Hello World from process %d running on %s\n", source_id, msg);
      }
    }
    else {
      MPI_Get_processor_name(msg, &length);
      MPI_Send(msg, length, MPI_CHAR, 0, tag, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    if (id==0) printf("Ready\n");
}
```

Google: MapReduce

- odolnosť voči chybám, synchronizácia, scheduling
- menej flexibility: komunikácia Map $\rightarrow$ Shuffle $\rightarrow$ Reduce

- Map: vstup $\rightarrow (k_1, v_1), (k_2, v_2), \ldots$
- Shuffle: $\rightarrow (k_1, v, v, v, \ldots), (k_2, v, v, v, \ldots)$
- Reduce: $(k, v_1, v_2, \ldots) \rightarrow$ výstup



Príklad

	Map		Shuffle		Reduce	
škrtia sa krty	→	(škrtia, 1), (sa, 1), (krty, 1)	...	(škrtia, 1)	→	škrtia: 1
			...	(sa, 1)	→	sa: 1
krt krta škrtí	→	(krt, 1), (krta, 1), (škrtí, 1)	...	(krty, 1)	→	krty: 1
			...	(krt, 1, 1)	→	krt: 2
keď krt krta zaškrtí	→	(keď, 1), (krt, 1), (krta, 1), (zaškrtí, 1)	...	(krta, 1, 1)	→	krta: 2
			...	(škrtí, 1)	→	škrtí: 1
do rána ho zmaškrtí	→	(do, 1), (rána, 1), (ho, 1), (zmaškrtí, 1)	...	(keď, 1)	→	keď: 1
			...	(zaškrtí, 1)	→	zaškrtí: 1
			...	(do, 1)	→	do: 1
			...	(rána, 1)	→	rána: 1
			...	(ho, 1)	→	ho: 1
			...	(zmaškrtí, 1)	→	zmaškrtí: 1

```
Map(String input_line):
  for each word w in input_line: Emit(w, 1);

Reduce(String key, Iterator values):
  int result = 0;
  for each v in values: result += v;
  Emit(key + ": " + result);
```

abstraktný sieťový model

- nezávislé zariadenia (procesy, procesory, uzly)
- posielanie správ
- lokálny pohľad (číslovanie portov)
- asynchrónne, spoľahlivé správy
- topológia siete
- wakeup/terminácia

zložitosť: v závislosti od počtu procesorov

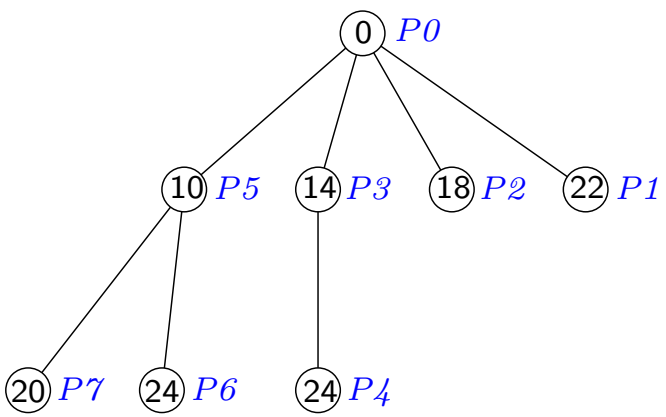
- počet správ / bitov
- čas (beh normovaný na max. dĺžku 1)

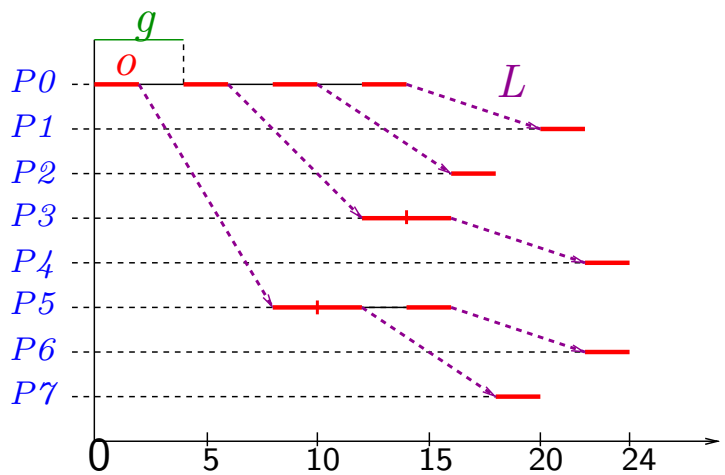
LogP model

- asynchrónny model
- komunikácia pomocou správ
- abstrahuje od štruktúry siete
- parametre:
  - L: horné ohraničenie latencie – zdržanie v sieti
  - o: overhead – zdržanie v procesore pri každej komunikácii
  - g: gap – minimálny čas medzi dvoma komunikáciami v procesore

poslať správu dĺžky m trvá $L \cdot m + 2o$

Príklad: broadcast pre $P = 8$, $L = 6$, $g = 4$, $o = 2$





BSP (Bulk Synchronous Parallel)

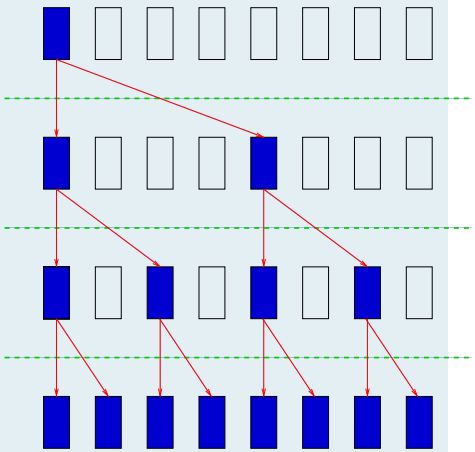
- sada procesorov spojených sieťou
- **superstep** pozostáva z
  - lokálny výpočet v každom procesore
  - komunikácia pomocou správ
  - barierová synchronizácia
- čas superstepu je $\max\{comp_i\} + \max\{comm_i\} \cdot g + l$
  - $comp_i$ je čas počítania i-teho procesora
  - $comm_i$ je objem dát prijatých a poslaných procesorom i
  - g je *lokálna priepustnosť* siete: výpočtový výkon / priepustnosť routera
  - l je čas potrebný na synchronizáciu

BSP broadcast – n hodnôt, p procesorov

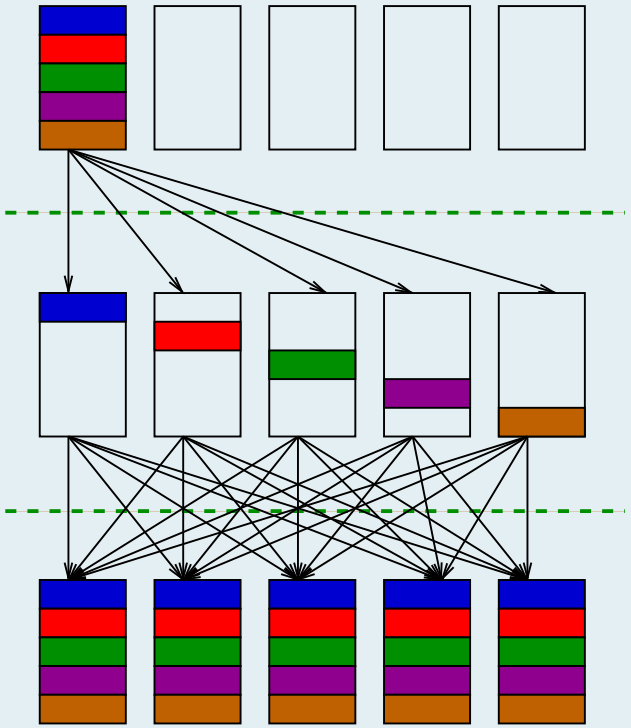
$$\max\{comp_i\} + \max\{comm_i\} \cdot g + l$$

jeden všetkým: cena $pn \cdot g + l$

strom: cena $\leq$
$\log p\,(3n \cdot g + l)$



dvojfázový: cena
$n \cdot g + l + \frac{n}{p}(p+1) \cdot g + l = O(n) \cdot g + O(1) \cdot l$



$\alpha\beta$ kalkulus

- použité v CM1, CM5 (CM LISP)
- distribuovaná dátová štruktúra: **xektor** – zoznam dvojíc $\{$index$\mapsto$hodnota$\}$
- α: konverzia hodnota$\mapsto$ konšt. xektor (t.j. zavedie hodnotu do všetkých procesorov)
  `(`α`+ '{a↦1 b↦2 c↦3} '{a↦3 b↦3} )` $\Rightarrow$ `{a↦4 b↦5}`
- •: ruší α
  `(CONS A X)` $\Rightarrow$ `([a b c ]. [x y z])`
  α`(CONS •A •X)` $\Rightarrow$ `[(a.x) (b.y) (c.z)]`
  α`(+ (* •x 2) 1)` $\equiv$ `(`α`+ (`α`* x `α`2) `α`1)`
- β redukcia: paralelne v log. čase
  `(`β`+ '{A↦1 B↦2 C↦3})` $\Rightarrow$ 6 (ignorujú sa indexy)
- veľkosť xektora x: `(SQRT (`β`+ (`α`* x x)))`
  výpočet $\sum x_i^3$: `(`β`+ `α`(POW •x 3))`

CSP (Communicating Sequential Processes)

- sémantika
- pozorovanie procesu: postupnosť udalostí
- proces: množina pozorovaní
- vytváranie procesov:
  - **prefix**: $(x \mapsto P)$
  - **rekurzia**: `CLOCK = (tick↦CLOCK)`
    $F \equiv \mu X.F(X)$
  - **deterministický výber**: `(x↦P | y↦Q)`
  - **paralelizmus**: `(P || Q)` všetky "premiešania" pozorovaní
- **komunikácia**: zdieľaním udalostí

`K` = μ `X.minca↦(káva↦X | čaj↦X)`
`Z` = μ `X.( čaj↦X | káva↦X | minca↦čaj↦X)`
`(Z || K)` = μ `X.(minca↦čaj↦X)`

abstraktný sieťový model

- nezávislé zariadenia (procesy, procesory, uzly)
- posielanie správ
- lokálny pohľad (číslovanie portov)
- asynchrónne, spoľahlivé správy
- topológia siete
- wakeup/terminácia

zložitosť: v závislosti od počtu procesorov

- počet správ / bitov
- čas (beh normovaný na max. dĺžku 1)

abstraktný sieťový model

horný odhad pre problém

Existuje algoritmus, ktorý **pre všetky** topológie, vstupy, časovania, ...
... pracuje správne a vymení najviac $f(n)$ správ

dolný odhad pre problém

Pre každý algoritmus, **existuje kombinácia** topológie, vstupu, časovania, ...
... že buď nepracuje správne alebo vymení aspoň $f(n)$ správ

Sieťový model: broadcast a convergecast (maximum)

const: *deg* : integer
ID : integer
Neigh : [1...*deg*] link
parent : link
var: *msg* : text

Init:

if $parent = NULL$
send $\langle$**dispatch**, *msg*$\rangle$ **to self**

Code:

loop forever

On receipt $\langle$**dispatch**, *new_msg*$\rangle$ from *parent* or self :

msg := *new_msg*
for all $l \in Neigh - \{parent\}$ **do**
send $\langle$**dispatch**, *msg*$\rangle$ **to** *l*
skonči algoritmus

Sieťový model: broadcast a convergecast (maximum)

const: *deg* : integer
ID : integer
Neigh : [1...*deg*] link
parent : link
msg : integer
var: *count* : integer
max : integer

Init:

$count = 0$
$max = msg$
if $deg = 1$
  send $\langle$**my_max**, $max\rangle$ **to** *parent*

Code:

loop forever

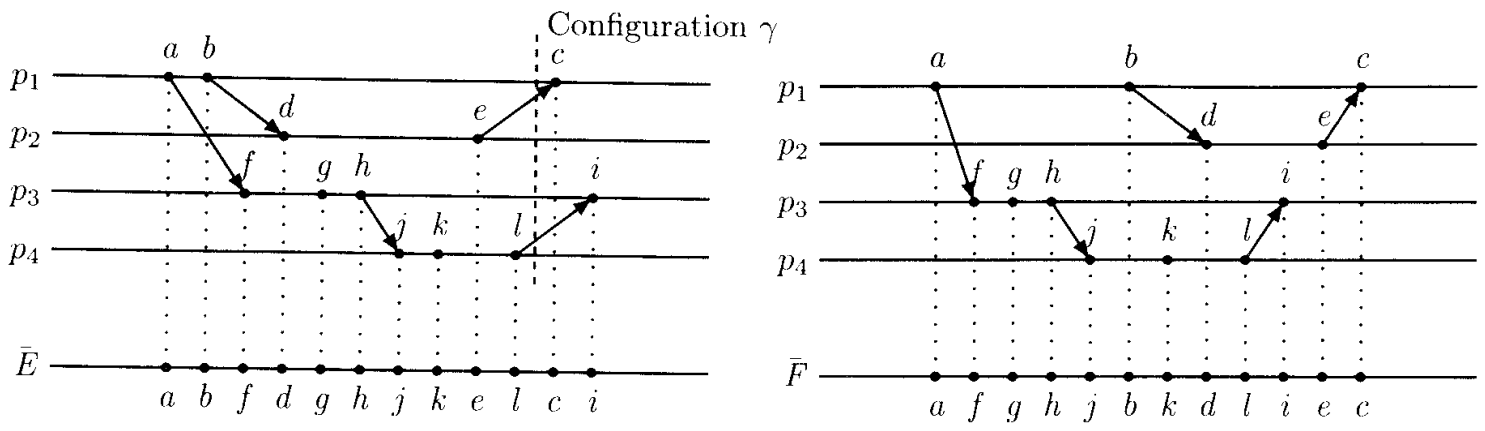
On receipt $\langle$**my_max**, $x\rangle$ from *Neigh*[*i*]:

  $max = \mathrm{maximum}\{max, x\}$
  $count++$
  if $count = deg - 1$
    send $\langle \mathrm{my\_max}, max\rangle$ **to** *parent*
    skonči algoritmus

Sieťový model

formálny model: prechodové systémy

- systém je trojica $(\mathcal{C}, \mapsto, \mathcal{I})$
- beh (execution) je postupnosť $\gamma_0 \mapsto \gamma_1 \mapsto \gamma_2 \mapsto \cdots$, kde $\gamma_0 \in \mathcal{I}$
- $\mathcal{C} = \mathcal{C}_L^n \times \mathcal{M}$: stavy systému = stavy procesorov + správy na linkách
- $\mapsto$: krok jedného procesora (výpočet, poslanie správy, prijatie správy)
- fair scheduler?



Sieťový model

formálny model: prechodové systémy

- (ne)závislosť udalostí:
  1) poslanie pred prijatím
  2) časovanie v rámci procesora
  3) tranzitivita
- výpočet (computation): trieda ekvivalencie behov

logické hodiny (Lamport)

var θ_p : integer **init** 0 ;

(* An internal event *)
$\theta_p := \theta_p + 1$;
Change state

(* A send event *)
$\theta_p := \theta_p + 1$:
send $\langle$ **messg**, θ_p $\rangle$; Change state

(* A receive event *)
receive $\langle$ **messg**, θ $\rangle$; $\theta_p := \max(\theta_p, \theta) + 1$;
Change state

cvičenia

- voľba šéfa na (anonymných) stromoch
- voľba šéfa na (anonymnej) mriežke