Paralelné algoritmy, časť č. 1

František Mráz

Kabinet software a výuky informatiky, MFF UK, Praha

Paralelné algoritmy, 2011/2012

Obsah

Úvod

Prečo paralelné počítanie

- zvyšovanie výkonu sekvenčných procesorov naráža na hranice možností

Môže paralelizácia pomôcť?

Úvod

Prečo paralelné počítanie

- zvyšovanie výkonu sekvenčných procesorov naráža na hranice možností
- vývoj hardware

Môže paralelizácia pomôcť?

Úvod

Prečo paralelné počítanie

- zvyšovanie výkonu sekvenčných procesorov naráža na hranice možností
- vývoj hardware
  - už v 80. rokoch 20. stor. sa používali vektorové počítače a boli stavané multiprocesorové systémy (nCube, Connection Machine (prvý s 2^{16} procesormi, Intel Paragon, . . .)

Môže paralelizácia pomôcť?

Úvod

Prečo paralelné počítanie

- zvyšovanie výkonu sekvenčných procesorov naráža na hranice možností
- vývoj hardware
  - už v 80. rokoch 20. stor. sa používali vektorové počítače a boli stavané multiprocesorové systémy (nCube, Connection Machine (prvý s 2^{16} procesormi, Intel Paragon, ...)
  - v 90. rokoch viacjadrové procesory

Môže paralelizácia pomôcť?

Úvod

Prečo paralelné počítanie

- zvyšovanie výkonu sekvenčných procesorov naráža na hranice možností
- vývoj hardware
  - už v 80. rokoch 20. stor. sa používali vektorové počítače a boli stavané multiprocesorové systémy (nCube, Connection Machine (prvý s 2^{16} procesormi, Intel Paragon, ...)
  - v 90. rokoch viacjadrové procesory

Môže paralelizácia pomôcť?

- Riešenie akých úloh môže paralelizácia zrýchliť?

Úvod

Prečo paralelné počítanie

- zvyšovanie výkonu sekvenčných procesorov naráža na hranice možností
- vývoj hardware
  - už v 80. rokoch 20. stor. sa používali vektorové počítače a boli stavané multiprocesorové systémy (nCube, Connection Machine (prvý s 2^{16} procesormi, Intel Paragon, . . .)
  - v 90. rokoch viacjadrové procesory

Môže paralelizácia pomôcť?

- Riešenie akých úloh môže paralelizácia zrýchliť?
- Aké zrýchlenie je teoreticky možné? Pre všetky úlohy, alebo iba pre niektoré?

Úvod

Prečo paralelné počítanie

- zvyšovanie výkonu sekvenčných procesorov naráža na hranice možností
- vývoj hardware
  - už v 80. rokoch 20. stor. sa používali vektorové počítače a boli stavané multiprocesorové systémy (nCube, Connection Machine (prvý s 2^{16} procesormi, Intel Paragon, ...)
  - v 90. rokoch viacjadrové procesory

Môže paralelizácia pomôcť?

- Riešenie akých úloh môže paralelizácia zrýchliť?
- Aké zrýchlenie je teoreticky možné? Pre všetky úlohy, alebo iba pre niektoré?
- $\Rightarrow$ potrebujeme dobrý model!

Obsah semestrálnej prednášky

1. Model PRAM. Vychádza zo sekvenčného modelu RAM, ale môže byť príliš silný!

Obsah semestrálnej prednášky

1. Model PRAM. Vychádza zo sekvenčného modelu RAM, ale môže byť príliš silný!
2. Základné techniky paralelných výpočtov.

Obsah semestrálnej prednášky

1. Model PRAM. Vychádza zo sekvenčného modelu RAM, ale môže byť príliš silný!
2. Základné techniky paralelných výpočtov.
3. Základné paralelné algoritmy – hlavne grafové.

Obsah semestrálnej prednášky

1. Model PRAM. Vychádza zo sekvenčného modelu RAM, ale môže byť príliš silný!
2. Základné techniky paralelných výpočtov.
3. Základné paralelné algoritmy – hlavne grafové.
4. Aké sú hranice paralelných výpočtov – ťažko paralelizovateľné úlohy.

Literatúra

1. Ian Parberry: Parallel Complexity Theory, Pitman Publishing, (John Wiley & Sons), 1987.

Literatúra

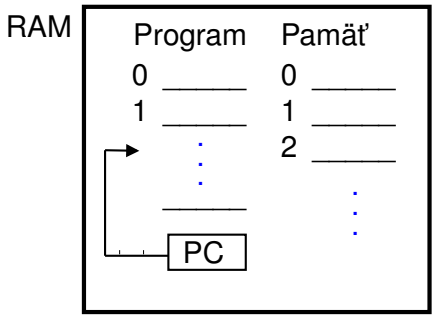
1. Ian Parberry: Parallel Complexity Theory, Pitman Publishing, (John Wiley & Sons), 1987.
2. Alan Gibbons, Wojciech Rytter: Efficient Parallel Algorithms, Cambridge University Press, 1988.

Literatúra

1. Ian Parberry: Parallel Complexity Theory, Pitman Publishing, (John Wiley & Sons), 1987.
2. Alan Gibbons, Wojciech Rytter: Efficient Parallel Algorithms, Cambridge University Press, 1988.
3. Joseph Jájá: An Introduction to Parallel Algorithms, Addison Wesley, Massachusetts, 1992.

Random Access Memory – RAM

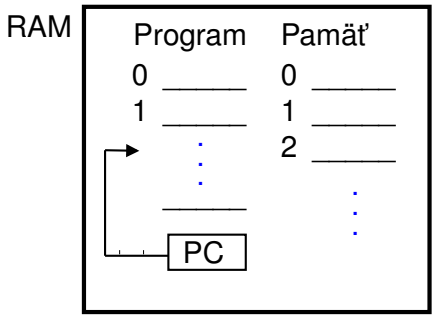
Počítač s náhodným prístupom (do pamäte).



- **Program** = pevná postupnosť inštrukcií

Random Access Memory – RAM

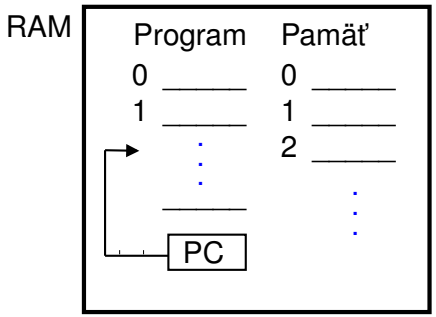
Počítač s náhodným prístupom (do pamäte).



- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).

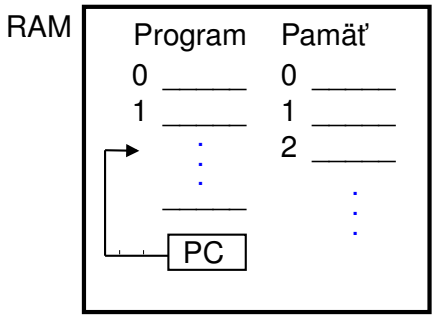


- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

- **Pamäť** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.

Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).

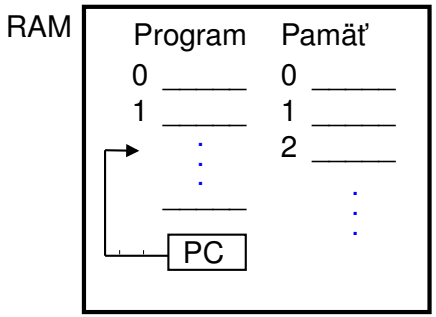


- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

- **Pamäť** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.
- **Inštrukcie** programu:

Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).



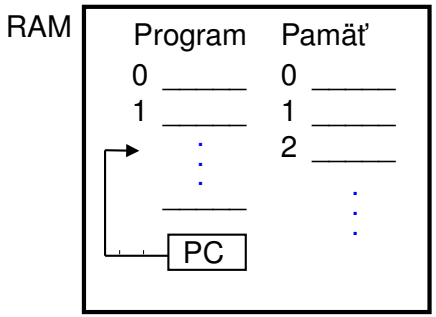
- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

- **Pamäť** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.
- **Inštrukcie** programu:

  $r_i :=$ konštanta;

Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).



- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

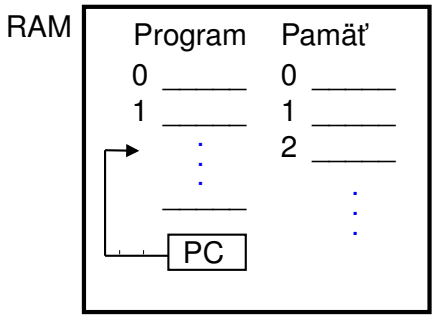
- **Pamäť** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.
- **Inštrukcie** programu:

  $r_i := \text{konštanta}$;

  $r_i := r_j \circ r_k$, kde $\circ$ je binárna operácia, napr. $+, -, *,$ mod , div;

Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).

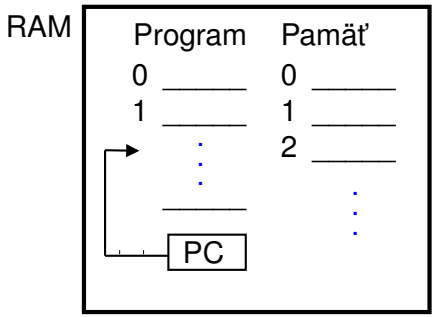


- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

- **Pamäť** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.
- **Inštrukcie** programu:
  - $r_i := \text{konštanta}$;
  - $r_i := r_j \circ r_k$, kde $\circ$ je binárna operácia, napr. $+, -, *,$ mod , div;
  - $r_i := r_{r_j}$, $r_{r_i} := r_j$ . . . nepriama adresácia;

Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).

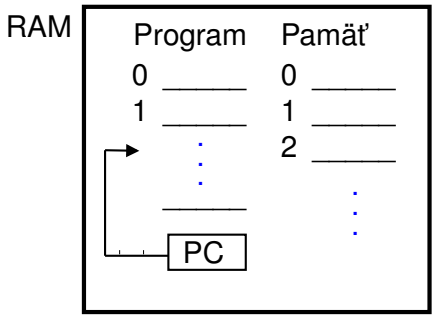


- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

- **Pamäť** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.
- **Inštrukcie** programu:
  - $r_i := \text{konštanta}$;
  - $r_i := r_j \circ r_k$, kde $\circ$ je binárna operácia, napr. $+, -, *,$ mod , div;
  - $r_i := r_{r_j}$, $r_{r_i} := r_j$ . . . nepriama adresácia;
  - `halt`;

Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).



- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

- **Pamäť** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.
- **Inštrukcie** programu:

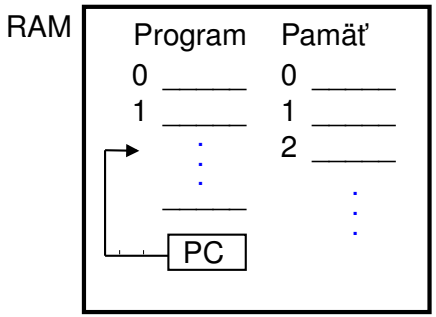
  $r_i := \text{konštanta}$;

  $r_i := r_j \circ r_k$, kde $\circ$ je binárna operácia, napr. $+, -, *,$ mod , div;

  - $r_i := r_{r_j}$, $r_{r_i} := r_j$ . . . nepriama adresácia;
  - `halt`;
  - `goto` m `if` $r_i > 0$.

Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).

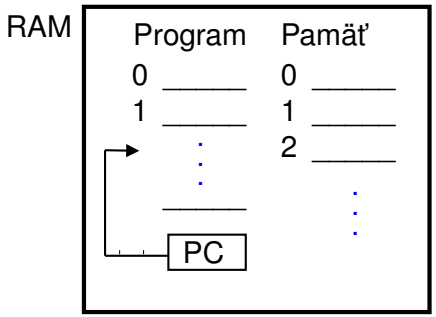


- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

- **Pamäť** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.
- **Inštrukcie** programu:
  - $r_i := \text{konštanta}$;
  - $r_i := r_j \circ r_k$, kde $\circ$ je binárna operácia, napr. $+, -, *,$ mod , div;
  - $r_i := r_{r_j}$, $r_{r_i} := r_j$ . . . nepriama adresácia;
  - `halt`;
  - `goto` m `if` $r_i > 0$.
- **Vstup**: r_0 obsahuje počet čísel, $r_1, \ldots, r_{r_0}$ obsahujú vstupné údaje;

Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).



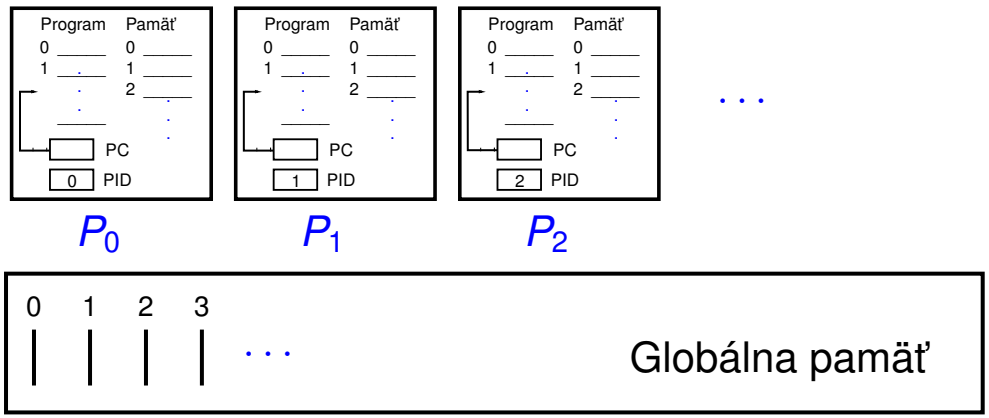
- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

- **Pamäť** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.
- **Inštrukcie** programu:
  - $r_i := \text{konštanta}$;
  - $r_i := r_j \circ r_k$, kde $\circ$ je binárna operácia, napr. $+, -, *,$ mod , div;
  - $r_i := r_{r_j}$, $r_{r_i} := r_j$ . . . nepriama adresácia;
  - `halt`;
  - `goto` m `if` $r_i > 0$.
- **Vstup**: r_0 obsahuje počet čísel, $r_1, \ldots, r_{r_0}$ obsahujú vstupné údaje;
- **Výstup**: dtto

Parallel RAM – PRAM

- postupnosť procesorov – RAM počítačov; každý z nich má naviac číslo PID (Processor IDentification number) uložené v špeciálnom registre; procesory pracujú synchrónne

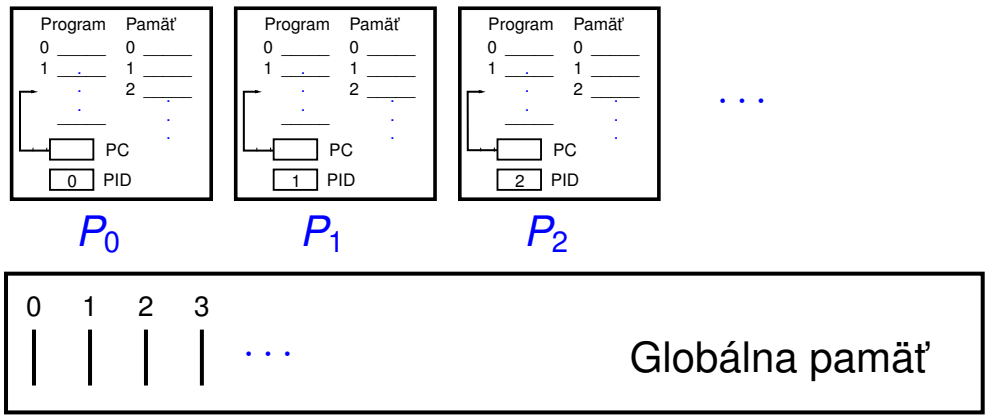
PRAM



Parallel RAM – PRAM

- postupnosť procesorov – RAM počítačov; každý z nich má naviac číslo PID (Processor IDentification number) uložené v špeciálnom registre; procesory pracujú synchrónne
- Spoločná globálna pamäť; jej registre budeme označovať s pruhom $\overline{r}_0, \overline{r}_1, \overline{r}_2 \ldots$

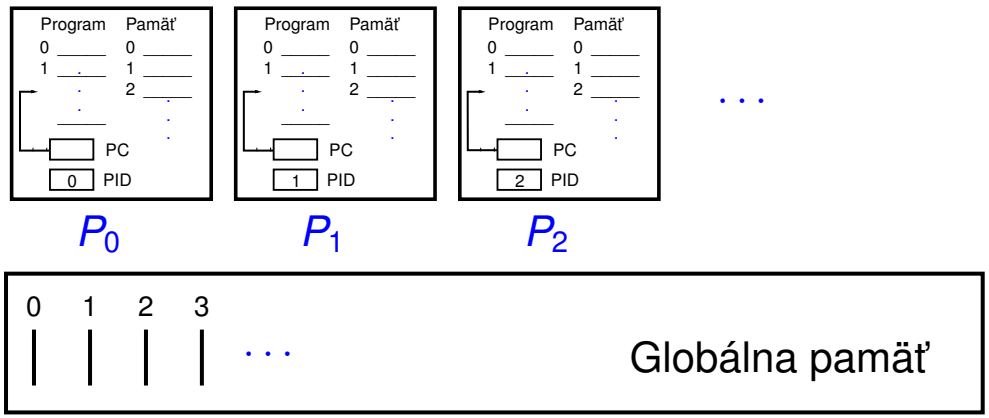
PRAM



Parallel RAM – PRAM

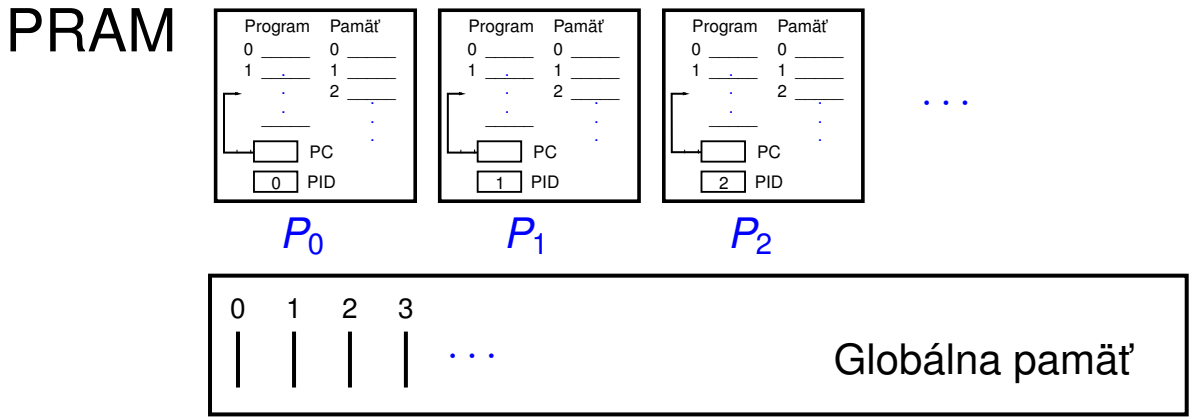
- postupnosť procesorov – RAM počítačov; každý z nich má naviac číslo PID (Processor IDentification number) uložené v špeciálnom registre; procesory pracujú synchrónne
- Spoločná globálna pamäť; jej registre budeme označovať s pruhom $\overline{r}_0, \overline{r}_1, \overline{r}_2 \ldots$
- Inštrukcie – tak ako RAM

PRAM



Parallel RAM – PRAM

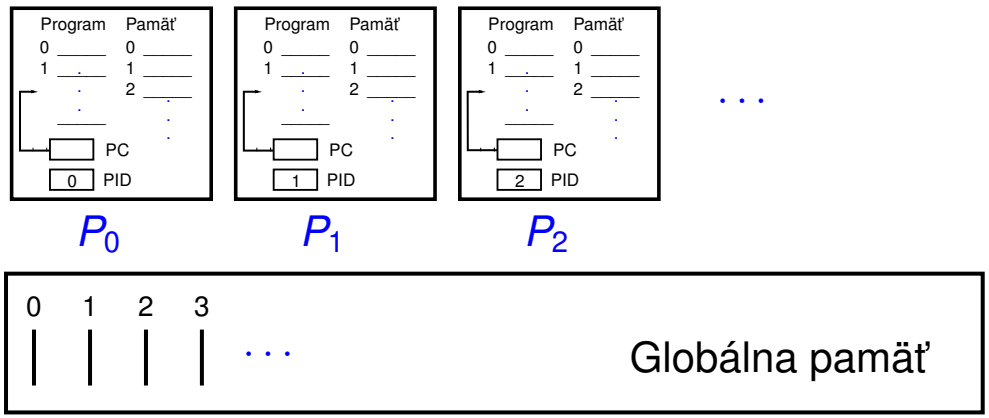
- postupnosť procesorov – RAM počítačov; každý z nich má naviac číslo PID (Processor IDentification number) uložené v špeciálnom registre; procesory pracujú synchrónne
- Spoločná globálna pamäť; jej registre budeme označovať s pruhom $\overline{r}_0, \overline{r}_1, \overline{r}_2 \ldots$
- Inštrukcie – tak ako RAM
  - \+ tie isté inštrukcie na prácu s globálnou pamäťou a



Parallel RAM – PRAM

- postupnosť procesorov – RAM počítačov; každý z nich má naviac číslo PID (Processor IDentification number) uložené v špeciálnom registre; procesory pracujú synchrónne
- Spoločná globálna pamäť; jej registre budeme označovať s pruhom $\overline{r}_0, \overline{r}_1, \overline{r}_2 \ldots$
- Inštrukcie – tak ako RAM
  - + tie isté inštrukcie na prácu s globálnou pamäťou a
  - + inštrukcia $r_i :=$ `PID`

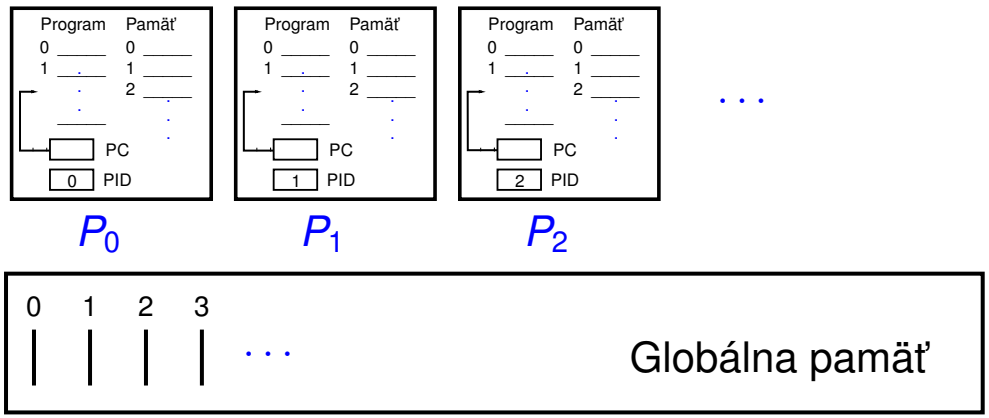
PRAM



Parallel RAM – PRAM

- postupnosť procesorov – RAM počítačov; každý z nich má naviac číslo PID (Processor IDentification number) uložené v špeciálnom registre; procesory pracujú synchrónne
- Spoločná globálna pamäť; jej registre budeme označovať s pruhom $\overline{r}_0, \overline{r}_1, \overline{r}_2 \ldots$
- Inštrukcie – tak ako RAM
  - + tie isté inštrukcie na prácu s globálnou pamäťou a
  - + inštrukcia $r_i :=$ `PID`
- Vstup a výstup v počiatočnom úseku globálnej pamäte.

PRAM



PRAM

Riešenie konfliktov

- Pri čítaní

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

  EW – Exclusive Write – konflikty pri zápise zakázané,

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

  EW – Exclusive Write – konflikty pri zápise zakázané,
  CW – Concurrent Write – konflikty povolené, ale čo s nimi?

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

  EW – Exclusive Write – konflikty pri zápise zakázané,
  CW – Concurrent Write – konflikty povolené, ale čo s nimi?
  PRIORITY zápis sa podarí procesoru s najnižším PID

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

  EW – Exclusive Write – konflikty pri zápise zakázané,
  CW – Concurrent Write – konflikty povolené, ale čo s nimi?

  PRIORITY zápis sa podarí procesoru s najnižším PID
  ARBITRARY zápis sa podarí ľubovoľnému procesoru

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

  EW – Exclusive Write – konflikty pri zápise zakázané,
  CW – Concurrent Write – konflikty povolené, ale čo s nimi?

  PRIORITY zápis sa podarí procesoru s najnižším PID
  ARBITRARY zápis sa podarí ľubovoľnému procesoru
  COLLISION podarí sa zapísať značku, že nastala kolízia

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

  EW – Exclusive Write – konflikty pri zápise zakázané,
  CW – Concurrent Write – konflikty povolené, ale čo s nimi?

  PRIORITY zápis sa podarí procesoru s najnižším PID
  ARBITRARY zápis sa podarí ľubovoľnému procesoru
  COLLISION podarí sa zapísať značku, že nastala kolízia
  COMMON zápis sa podarí, ale všetci musia zapisovať to isté číslo

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

  EW – Exclusive Write – konflikty pri zápise zakázané,
  CW – Concurrent Write – konflikty povolené, ale čo s nimi?

  PRIORITY zápis sa podarí procesoru s najnižším PID
  ARBITRARY zápis sa podarí ľubovoľnému procesoru
  COLLISION podarí sa zapísať značku, že nastala kolízia
  COMMON zápis sa podarí, ale všetci musia zapisovať to isté číslo
  W-PRAM zápis sa podarí, ale všetci musia zapisovať rovnaké číslo 1

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

  EW – Exclusive Write – konflikty pri zápise zakázané,
  CW – Concurrent Write – konflikty povolené, ale čo s nimi?

  PRIORITY zápis sa podarí procesoru s najnižším PID
  ARBITRARY zápis sa podarí ľubovoľnému procesoru
  COLLISION podarí sa zapísať značku, že nastala kolízia
  COMMON zápis sa podarí, ale všetci musia zapisovať to isté číslo
  W-PRAM zápis sa podarí, ale všetci musia zapisovať rovnaké číslo 1

  ...

PRAM

Riešenie konfliktov

- Pri čítaní

  ER Exclusive Read – konflikty pri čítaní zakázané,
  CR Concurrent Read – konflikty povolené.

- Pri zápise

  EW – Exclusive Write – konflikty pri zápise zakázané,
  CW – Concurrent Write – konflikty povolené, ale čo s nimi?

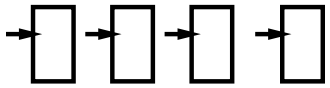
  PRIORITY zápis sa podarí procesoru s najnižším PID
  ARBITRARY zápis sa podarí ľubovoľnému procesoru
  COLLISION podarí sa zapísať značku, že nastala kolízia
  COMMON zápis sa podarí, ale všetci musia zapisovať to isté číslo
  W-PRAM zápis sa podarí, ale všetci musia zapisovať rovnaké číslo 1

  ...

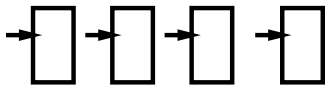
- Najčastejšie používané: CREW, COMMON, PRIORITY

Priradenie programov procesorom

SIMD – Single Instruction Multiple Data



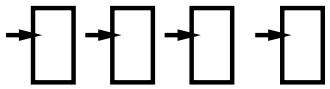
Priradenie programov procesorom



SIMD – Single Instruction Multiple Data

- zhodné programy a zhodné PC
- k riadeniu skokov iba globálne podmienky

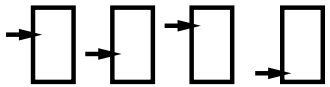
Priradenie programov procesorom



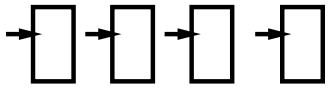
SIMD – Single Instruction Multiple Data

- zhodné programy a zhodné PC
- k riadeniu skokov iba globálne podmienky

Uniformné

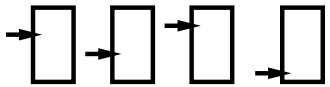


Priradenie programov procesorom



SIMD – Single Instruction Multiple Data

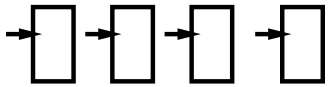
- zhodné programy a zhodné PC
- k riadeniu skokov iba globálne podmienky



Uniformné

- zhodné programy, rovnaký počet vykonaných inštrukcií
- rôzne PC

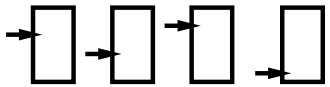
Priradenie programov procesorom



SIMD – Single Instruction Multiple Data

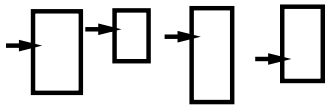
- zhodné programy a zhodné PC
- k riadeniu skokov iba globálne podmienky

Uniformné

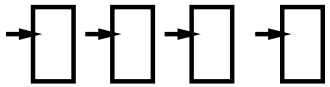


- zhodné programy, rovnaký počet vykonaných inštrukcií
- rôzne PC

MIMD – Multiple Instruction Multiple Data



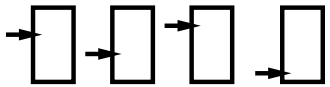
Priradenie programov procesorom



SIMD – Single Instruction Multiple Data

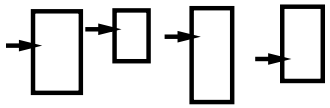
- zhodné programy a zhodné PC
- k riadeniu skokov iba globálne podmienky

Uniformné



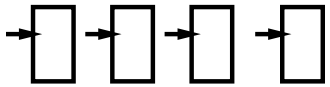
- zhodné programy, rovnaký počet vykonaných inštrukcií
- rôzne PC

MIMD – Multiple Instruction Multiple Data



- rôzne programy
- rôzne PC

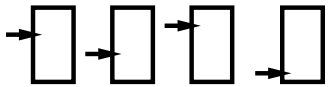
Priradenie programov procesorom



SIMD – Single Instruction Multiple Data

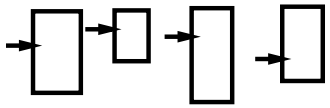
- zhodné programy a zhodné PC
- k riadeniu skokov iba globálne podmienky

Uniformné



- zhodné programy, rovnaký počet vykonaných inštrukcií
- rôzne PC

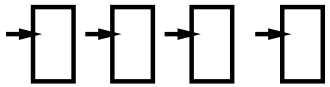
MIMD – Multiple Instruction Multiple Data



- rôzne programy
- rôzne PC

- simulácia SIMD $\dashrightarrow^{O(1)}$ uniformné

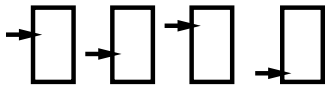
Priradenie programov procesorom



SIMD – Single Instruction Multiple Data

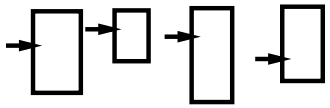
- zhodné programy a zhodné PC
- k riadeniu skokov iba globálne podmienky

Uniformné



- zhodné programy, rovnaký počet vykonaných inštrukcií
- rôzne PC

MIMD – Multiple Instruction Multiple Data



- rôzne programy
- rôzne PC

- simulácia SIMD $\dashrightarrow^{O(1)}$ uniformné
- simulácia SIMD $\dashleftarrow^{O(\text{dĺžka pgmu})}$ uniformné

Miery zložitosti paralelných výpočtov

Časová zložitosť jednej inštrukcie

Miery zložitosti paralelných výpočtov

Časová zložitosť jednej inštrukcie

- jednotková cena inštrukcie

Miery zložitosti paralelných výpočtov

Časová zložitosť jednej inštrukcie

- jednotková cena inštrukcie
- logaritmická cena inštrukcie = počet spracovaných bitov

Miery zložitosti paralelných výpočtov

Časová zložitosť jednej inštrukcie

- jednotková cena inštrukcie
- logaritmická cena inštrukcie = počet spracovaných bitov

uvažujeme jednotkovú (synchronizácia), ale je treba dať pozor na veľkosť spracovávaných čísel; predpokladáme model SIMD, resp. uniformný

Miery zložitosti paralelných výpočtov

Časová zložitosť jednej inštrukcie

- jednotková cena inštrukcie
- logaritmická cena inštrukcie = počet spracovaných bitov

uvažujeme jednotkovú (synchronizácia), ale je treba dať pozor na veľkosť spracovávaných čísel; predpokladáme model SIMD, resp. uniformný

$T(n)$ čas výpočtu na vstupe dĺžky n.

Miery zložitosti paralelných výpočtov

Časová zložitosť jednej inštrukcie

- jednotková cena inštrukcie
- logaritmická cena inštrukcie = počet spracovaných bitov

uvažujeme jednotkovú (synchronizácia), ale je treba dať pozor na veľkosť spracovávaných čísel; predpokladáme model SIMD, resp. uniformný

$T(n)$ čas výpočtu na vstupe dĺžky n.

$S(n)$ maximálny počet registrov pamäte potrebných pri výpočte nad vstupom veľkosti n,

Miery zložitosti paralelných výpočtov

Časová zložitosť jednej inštrukcie

- jednotková cena inštrukcie
- logaritmická cena inštrukcie = počet spracovaných bitov

uvažujeme jednotkovú (synchronizácia), ale je treba dať pozor na veľkosť spracovávaných čísel; predpokladáme model SIMD, resp. uniformný

$T(n)$ čas výpočtu na vstupe dĺžky n.

$S(n)$ maximálny počet registrov pamäte potrebných pri výpočte nad vstupom veľkosti n,

$P(n)$ počet aktivovaných procesorov,

Miery zložitosti paralelných výpočtov

Časová zložitosť jednej inštrukcie

- jednotková cena inštrukcie
- logaritmická cena inštrukcie = počet spracovaných bitov

uvažujeme jednotkovú (synchronizácia), ale je treba dať pozor na veľkosť spracovávaných čísel; predpokladáme model SIMD, resp. uniformný

$T(n)$ čas výpočtu na vstupe dĺžky n.

$S(n)$ maximálny počet registrov pamäte potrebných pri výpočte nad vstupom veľkosti n,

$P(n)$ počet aktivovaných procesorov,

$W(n)$ dĺžka slova = dĺžka najdlhšieho slova (počet bitov čísla) zapísaného v priebehu výpočtu do nejakého registra

Aktivácia procesorov

- iniciálna = “všetci naraz” – nerealistické
- lenivá (angl. “lazy”) =“prvý budí ostatných”

Inštrukčná sada

množina aritmetických inštrukcií

- minimálna: $+$, $-$, div 2

Inštrukčná sada

množina aritmetických inštrukcií

- minimálna: $+$, $-$, div 2
- obmedzená: $+$, $-$, div 2, $\lfloor x2^y \rfloor$ (aritmetické posuny, “shifts”)

Inštrukčná sada

množina aritmetických inštrukcií

- minimálna: $+$, $-$, div 2
- obmedzená: $+$, $-$, div 2, $\lfloor x2^y \rfloor$ (aritmetické posuny, “shifts”)
- úplná: $+$, $-$, div 2, $\lfloor x2^y \rfloor$, $\star$, div

Inštrukčná sada

množina aritmetických inštrukcií

- minimálna: $+$, $-$, div 2
- obmedzená: $+$, $-$, div 2, $\lfloor x2^y \rfloor$ (aritmetické posuny, “shifts”)
- úplná: $+$, $-$, div 2, $\lfloor x2^y \rfloor$, $\star$, div
- rozšírená: $+$, $-$, div 2, $\lfloor x2^y \rfloor$, $\star$, div, x^y

Inštrukčná sada

množina aritmetických inštrukcií

- minimálna: $+$, $-$, div 2
- obmedzená: $+$, $-$, div 2, $\lfloor x2^y \rfloor$ (aritmetické posuny, “shifts”)
- úplná: $+$, $-$, div 2, $\lfloor x2^y \rfloor$, $\star$, div
- rozšírená: $+$, $-$, div 2, $\lfloor x2^y \rfloor$, $\star$, div, x^y

Väčšinou budeme používať obmedzenú sadu inštrukcií

Simulácia COMMON na EREW PRAM

Predpokladajme, že v určitom kroku výpočtu môže do istého registra $\bar{r}$ zapisovať jeden alebo viacej z procesorov $p_1, p_2, \ldots, p_n$. Zmeňte algoritmus tak, aby nedochádzalo k zapisovacím konfliktom!

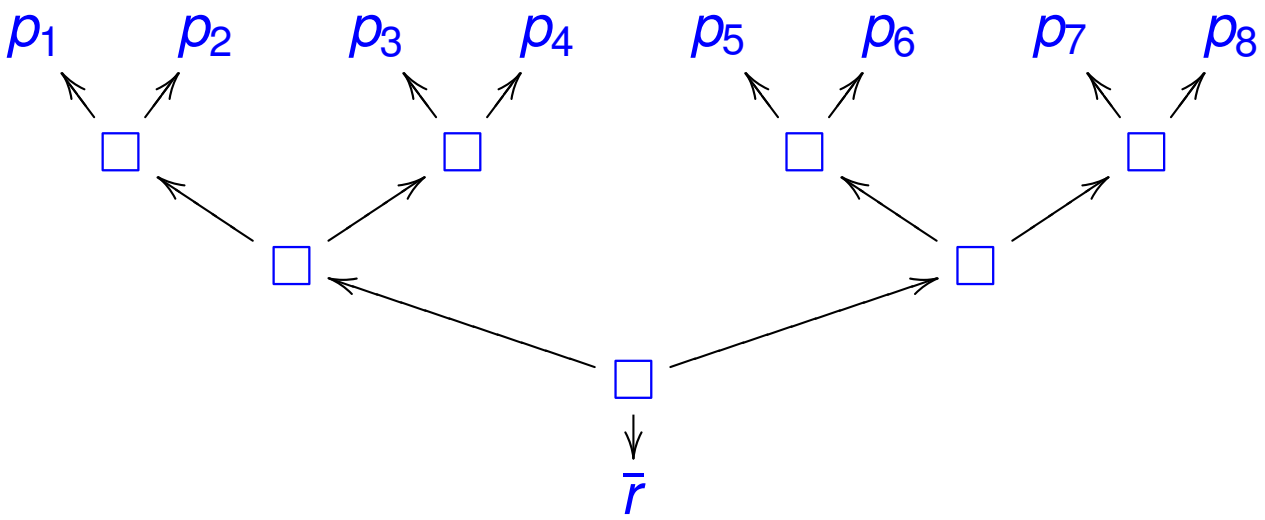
Simulácia COMMON na EREW PRAM

Predpokladajme, že v určitom kroku výpočtu môže do istého registra $\bar{r}$ zapisovať jeden alebo viacej z procesorov $p_1, p_2, \ldots, p_n$. Zmeňte algoritmus tak, aby nedochádzalo k zapisovacím konfliktom!

- pridáme stromovú štruktúru – každý procesor v strome dohliada na najviac 2 procesory pod ním, či nechcú zapisovať do registra $\bar{r}$

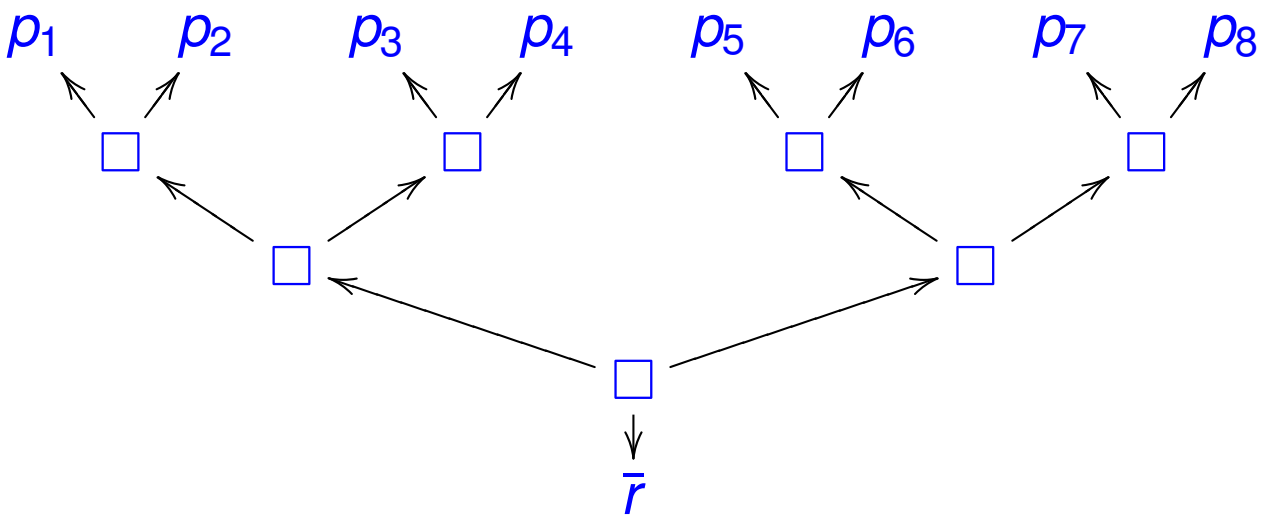
Simulácia COMMON na EREW PRAM

Predpokladajme, že v určitom kroku výpočtu môže do istého registra $\bar{r}$ zapisovať jeden alebo viacej z procesorov $p_1, p_2, \ldots, p_n$. Zmeňte algoritmus tak, aby nedochádzalo k zapisovacím konfliktom!

- pridáme stromovú štruktúru – každý procesor v strome dohliada na najviac 2 procesory pod ním, či nechcú zapisovať do registra $\bar{r}$
- 

Simulácia COMMON na EREW PRAM

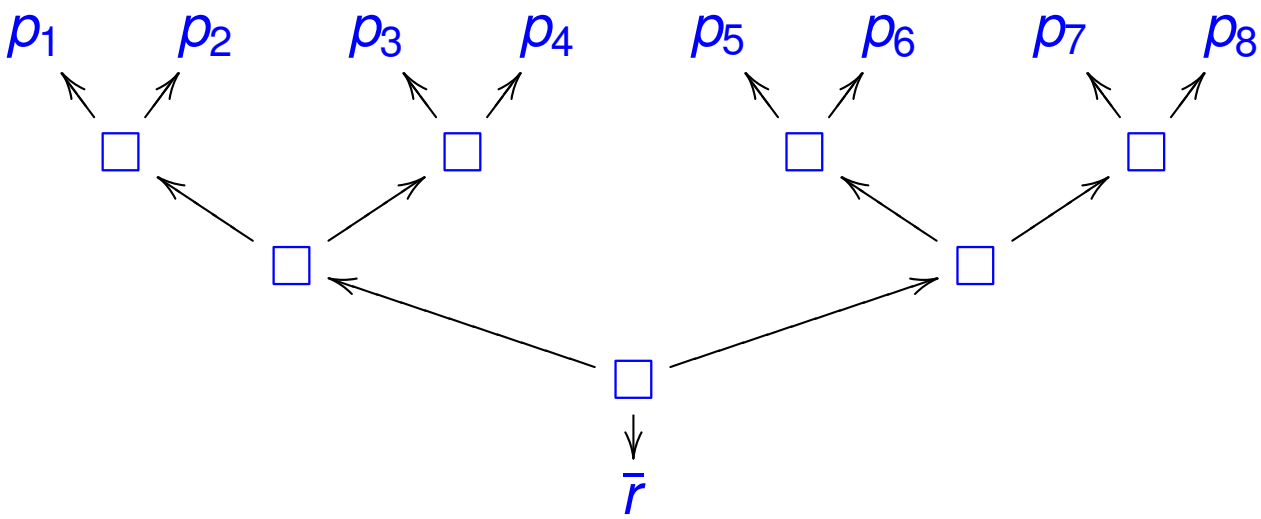
Predpokladajme, že v určitom kroku výpočtu môže do istého registra $\bar{r}$ zapisovať jeden alebo viacej z procesorov $p_1, p_2, \ldots, p_n$. Zmeňte algoritmus tak, aby nedochádzalo k zapisovacím konfliktom!

- pridáme stromovú štruktúru – každý procesor v strome dohliada na najviac 2 procesory pod ním, či nechcú zapisovať do registra $\bar{r}$
- 

- ak áno, tak ohlási do vyššieho poschodia, že chce zapisovať hodnotu h

Simulácia COMMON na EREW PRAM

Predpokladajme, že v určitom kroku výpočtu môže do istého registra $\bar{r}$ zapisovať jeden alebo viacej z procesorov $p_1, p_2, \ldots, p_n$. Zmeňte algoritmus tak, aby nedochádzalo k zapisovacím konfliktom!

- pridáme stromovú štruktúru – každý procesor v strome dohliada na najviac 2 procesory pod ním, či nechcú zapisovať do registra $\bar{r}$
- 

- ak áno, tak ohlási do vyššieho poschodia, že chce zapisovať hodnotu h
- spomalenie $O(\log n)$

Simulácia COMMON na EREW PRAM – pokračovanie

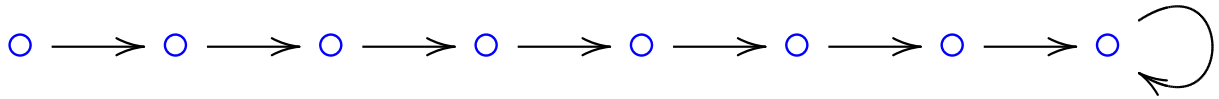
- ako získať pomocné procesory – napr. pôvodným procesorom pridelíme procesory s párnym PID a pomocným s nepárnym PID

Simulácia COMMON na EREW PRAM – pokračovanie

- ako získať pomocné procesory – napr. pôvodným procesorom pridelíme procesory s párnym PID a pomocným s nepárnym PID
- podobne sa dá “rozriediť” globálna pamäť $\rightarrow$ v podstate nepotrebujeme lokálne pamäte

Dynamicky alokovaná pamäť

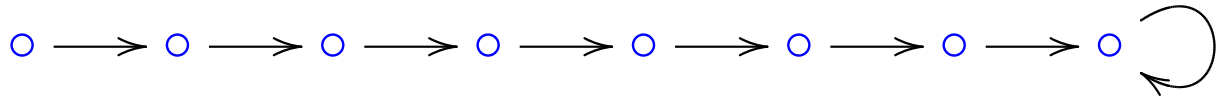
väzby medzi procesormi môžu byť vytvorené dynamicky. Napr. lineárny spojový zoznam:



- predpokladáme, že každý prvok k zoznamu má pridelený procesor a procesor má odkaz na následníka $P(k)$

Dynamicky alokovaná pamäť

väzby medzi procesormi môžu byť vytvorené dynamicky. Napr. lineárny spojový zoznam:

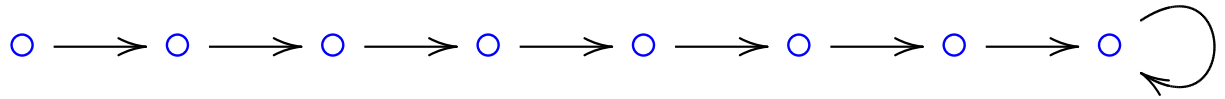


- predpokladáme, že každý prvok k zoznamu má pridelený procesor a procesor má odkaz na následníka $P(k)$
- **repeat** $\log n$ **times**

  if $P(P(k)) \neq P(k)$ **then** $P(k) := P(P(k))$

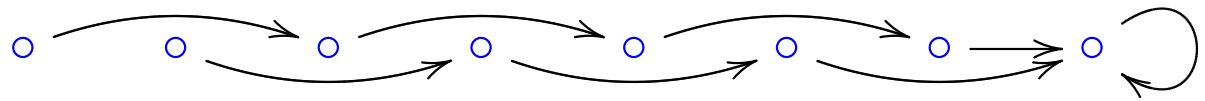
Dynamicky alokovaná pamäť

väzby medzi procesormi môžu byť vytvorené dynamicky. Napr. lineárny spojový zoznam:



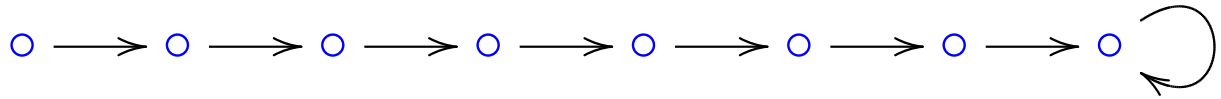
- predpokladáme, že každý prvok k zoznamu má pridelený procesor a procesor má odkaz na následníka $P(k)$
- **repeat** $\log n$ **times**

  if $P(P(k)) \neq P(k)$ **then** $P(k) := P(P(k))$

1. 

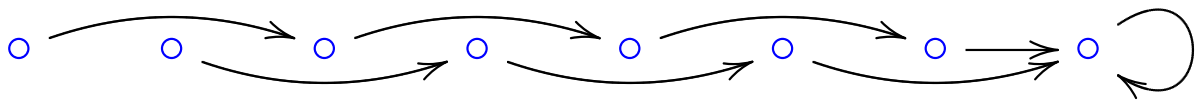
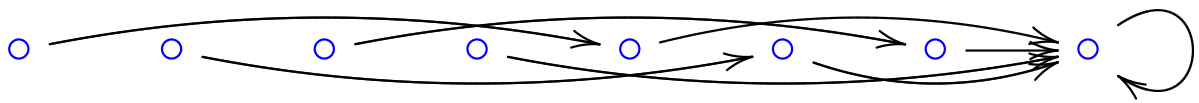
Dynamicky alokovaná pamäť

väzby medzi procesormi môžu byť vytvorené dynamicky. Napr. lineárny spojový zoznam:



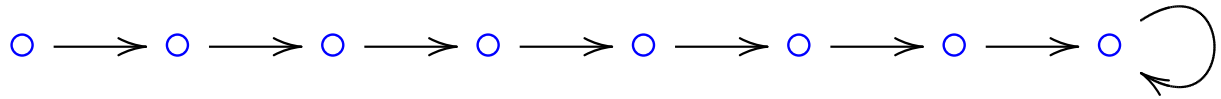
- predpokladáme, že každý prvok k zoznamu má pridelený procesor a procesor má odkaz na následníka $P(k)$
- **repeat** $\log n$ **times**

  if $P(P(k)) \neq P(k)$ **then** $P(k) := P(P(k))$

1. 
2. 

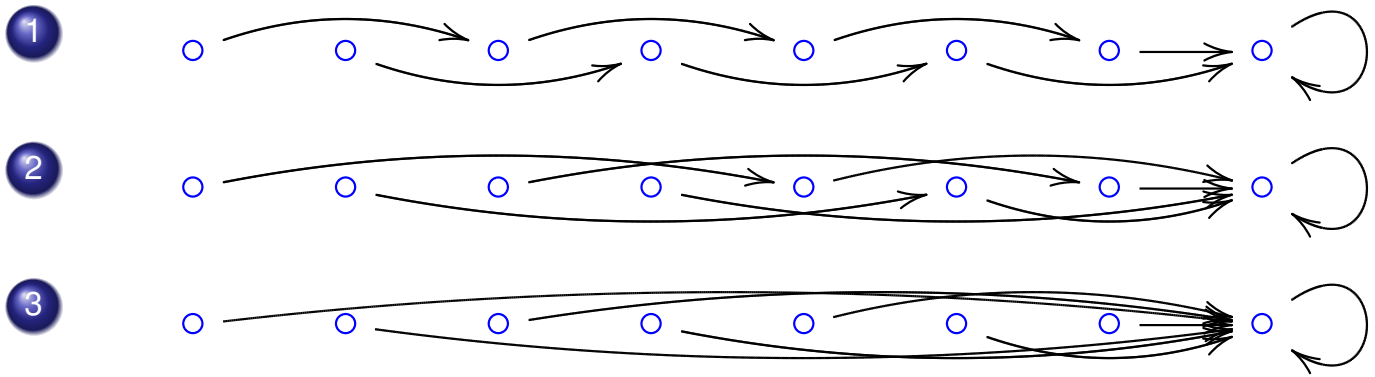
Dynamicky alokovaná pamäť

väzby medzi procesormi môžu byť vytvorené dynamicky. Napr. lineárny spojový zoznam:



- predpokladáme, že každý prvok k zoznamu má pridelený procesor a procesor má odkaz na následníka $P(k)$
- **repeat** $\log n$ **times**

  if $P(P(k)) \neq P(k)$ **then** $P(k) := P(P(k))$



Sekvenčné modely výpočtov (algoritmov)

- 1930 – Alan Turing – Turingov stroj (TS): rôzne modely líšiace sa počtom hláv, pások, ...
  zložitosť výpočtov:

Sekvenčné modely výpočtov (algoritmov)

- 1930 – Alan Turing – Turingov stroj (TS): rôzne modely líšiace sa počtom hláv, pások, ...
  zložitosť výpočtov:
  - časová – $\mathrm{SEQTIME}(n)$

Sekvenčné modely výpočtov (algoritmov)

- 1930 – Alan Turing – Turingov stroj (TS): rôzne modely líšiace sa počtom hláv, pások, ...
  zložitosť výpočtov:
  - časová – $\mathrm{SEQTIME}(n)$
  - priestorová – $\mathrm{SPACE}(n)$

Sekvenčné modely výpočtov (algoritmov)

- 1930 – Alan Turing – Turingov stroj (TS): rôzne modely líšiace sa počtom hláv, pások, ...
  zložitosť výpočtov:
  - časová – $\mathrm{SEQTIME}(n)$
  - priestorová – $\mathrm{SPACE}(n)$
- neskôr mnoho ďalších: Markovove algoritmy, PL-programy, čiastočne rekurzívne funkcie, RAM (s logaritmickou cenou inštrukcie), . . .
  sú rovnako silné, líšia sa najviac polynomiálne

Sekvenčné modely výpočtov (algoritmov)

- 1930 – Alan Turing – Turingov stroj (TS): rôzne modely líšiace sa počtom hláv, pások, ...
  zložitosť výpočtov:
  - časová – $\mathrm{SEQTIME}(n)$
  - priestorová – $\mathrm{SPACE}(n)$
- neskôr mnoho ďalších: Markovove algoritmy, PL-programy, čiastočne rekurzívne funkcie, RAM (s logaritmickou cenou inštrukcie), . . .
  sú rovnako silné, líšia sa najviac polynomiálne
- **Tvrdenie:** Ak TS rieši daný problém veľkosti n v čase $T(n)$, tak tento problém môže byť vyriešený na ľubovoľnom inom sekvenčnom modele M v čase $O([T(n)]^k)$ pre vhodnú konštantu k závislú iba na modeli M

Sekvenčné modely výpočtov (algoritmov)

- 1930 – Alan Turing – Turingov stroj (TS): rôzne modely líšiace sa počtom hláv, pások, ...
  zložitosť výpočtov:
  - časová – $\mathrm{SEQTIME}(n)$
  - priestorová – $\mathrm{SPACE}(n)$
- neskôr mnoho ďalších: Markovove algoritmy, PL-programy, čiastočne rekurzívne funkcie, RAM (s logaritmickou cenou inštrukcie), . . .
  sú rovnako silné, líšia sa najviac polynomiálne
- **Tvrdenie:** Ak TS rieši daný problém veľkosti n v čase $T(n)$, tak tento problém môže byť vyriešený na ľubovoľnom inom sekvenčnom modele M v čase $O([T(n)]^k)$ pre vhodnú konštantu k závislú iba na modeli M
- **Dôsledok:** Trieda P úloh riešiteľných na TS (RAMe s logaritmickou cenou inštrukcie) v polynomickom čase je pojem robustný voči zmene modelu z tejto triedy.

Sekvenčná a paralelná téza

- **Sekvenčná téza:** Všetky “rozumné” sekvenčné modely algoritmov sú polynomiálne zviazané.
  Pozn.: RAM s jednotkovou cenou inštrukcie nie je “rozumný”.

Sekvenčná a paralelná téza

- **Sekvenčná téza:** Všetky "rozumné" sekvenčné modely algoritmov sú polynomiálne zviazané.
  Pozn.: RAM s jednotkovou cenou inštrukcie nie je "rozumný".
- **Paralelná téza:** Všetky "rozumné" paralelné modely sa líšia iba polynomiálne a ich časová zložitosť je polynomiálne viazaná na priestorovú zložitosť sekvenčných algoritmov.

$$\begin{aligned} \mathrm{PARTIME}(T(n)) &\leq \mathrm{SPACE}(T(n)^k) \\ \mathrm{SPACE}(T(n)) &\leq \mathrm{PARTIME}(T(n)^{\widetilde{k}}) \end{aligned}$$

Sekvenčná a paralelná téza

- **Sekvenčná téza:** Všetky "rozumné" sekvenčné modely algoritmov sú polynomiálne zviazané.
  Pozn.: RAM s jednotkovou cenou inštrukcie nie je "rozumný".
- **Paralelná téza:** Všetky "rozumné" paralelné modely sa líšia iba polynomiálne a ich časová zložitosť je polynomiálne viazaná na priestorovú zložitosť sekvenčných algoritmov.

$$\begin{aligned} \mathrm{PARTIME}(T(n)) &\leq \mathrm{SPACE}(T(n)^k) \\ \mathrm{SPACE}(T(n)) &\leq \mathrm{PARTIME}(T(n)^{\widetilde{k}}) \end{aligned}$$

k a $\widetilde{k}$ konštanty

Sekvenčná a paralelná téza

- **Sekvenčná téza:** Všetky "rozumné" sekvenčné modely algoritmov sú polynomiálne zviazané.
  Pozn.: RAM s jednotkovou cenou inštrukcie nie je "rozumný".
- **Paralelná téza:** Všetky "rozumné" paralelné modely sa líšia iba polynomiálne a ich časová zložitosť je polynomiálne viazaná na priestorovú zložitosť sekvenčných algoritmov.

$$\mathrm{PARTIME}(T(n)) \leq \mathrm{SPACE}(T(n)^k)$$
$$\mathrm{SPACE}(T(n)) \leq \mathrm{PARTIME}(T(n)^{\widetilde{k}})$$

k a $\widetilde{k}$ konštanty

$\mathrm{SPACE}(f(n))$ úlohy riešiteľné sekvenčne v priestore $O(f(n))$, n je veľkosť úlohy

Sekvenčná a paralelná téza

- **Sekvenčná téza:** Všetky "rozumné" sekvenčné modely algoritmov sú polynomiálne zviazané.
  Pozn.: RAM s jednotkovou cenou inštrukcie nie je "rozumný".
- **Paralelná téza:** Všetky "rozumné" paralelné modely sa líšia iba polynomiálne a ich časová zložitosť je polynomiálne viazaná na priestorovú zložitosť sekvenčných algoritmov.

$$\mathrm{PARTIME}(T(n)) \leq \mathrm{SPACE}(T(n)^k)$$
$$\mathrm{SPACE}(T(n)) \leq \mathrm{PARTIME}(T(n)^{\widetilde{k}})$$

k a $\widetilde{k}$ konštanty

$\mathrm{SPACE}(f(n))$ úlohy riešiteľné sekvenčne v priestore $O(f(n))$, n je veľkosť úlohy

$\mathrm{PARTIME}(f(n))$ úlohy riešiteľné v paralelnom modeli v čase $O(f(n))$

Použitie paralelnej tézy

- kritérium “rozumnosti” modelu

Použitie paralelnej tézy

- kritérium “rozumnosti” modelu
- ak model spĺňa paralelnú tézu:

  Otvorený problém: $P \stackrel{?}{=} \mathrm{POLYLOGSPACE}$

  $\mathrm{POLYLOGSPACE}$ označuje triedu úloh riešiteľných v polylogaritmickom priestore $= \bigcup_k \mathrm{SPACE}(\log^k n)$

  iná formulácia problému: *Dá sa každá úloha riešiteľná v “rozumnom čase” (v P) exponenciálne zrýchliť na paralelnom modele?*

Použitie paralelnej tézy

- kritérium “rozumnosti” modelu
- ak model spĺňa paralelnú tézu:

  Otvorený problém: $P \stackrel{?}{=} \mathrm{POLYLOGSPACE}$

  $\mathrm{POLYLOGSPACE}$ označuje triedu úloh riešiteľných v polylogaritmickom priestore $= \bigcup_k \mathrm{SPACE}(\log^k n)$

  iná formulácia problému: *Dá sa každá úloha riešiteľná v “rozumnom čase” (v P) exponenciálne zrýchliť na paralelnom modele?*
- ak poznáme sekvenčnú priestorovú zložitosť problému, tak môžeme odhadovať paralelný čas

Optimálne a efektívne paralelné algoritmy

nech n označuje veľkosť problému

NC – Nick Pippenger's Class
trieda úloh riešiteľných v polylogaritmickom čase s polynomiálnym počtom procesorov (vzhľadom k n)

Optimálne a efektívne paralelné algoritmy

nech n označuje veľkosť problému

NC – Nick Pippenger's Class
trieda úloh riešiteľných v polylogaritmickom čase s polynomiálnym počtom procesorov (vzhľadom k n)

- nech S je problém, pre ktorý najlepší sekvenčný algoritmus má časovú zložitosť $T(n)$. Paralelný algoritmus A riešiaci problém S v čase $t(n)$ s $p(n)$ procesormi sa nazýva

Optimálne a efektívne paralelné algoritmy

nech n označuje veľkosť problému

NC – Nick Pippenger's Class
trieda úloh riešiteľných v polylogaritmickom čase s polynomiálnym počtom procesorov (vzhľadom k n)

- nech S je problém, pre ktorý najlepší sekvenčný algoritmus má časovú zložitosť $T(n)$. Paralelný algoritmus A riešiaci problém S v čase $t(n)$ s $p(n)$ procesormi sa nazýva

  optimálny ak

Optimálne a efektívne paralelné algoritmy

nech n označuje veľkosť problému

NC – Nick Pippenger's Class
trieda úloh riešiteľných v polylogaritmickom čase s polynomiálnym počtom procesorov (vzhľadom k n)

- nech S je problém, pre ktorý najlepší sekvenčný algoritmus má časovú zložitosť $T(n)$. Paralelný algoritmus A riešiaci problém S v čase $t(n)$ s $p(n)$ procesormi sa nazýva

  optimálny ak

  1. $t(n) = polylog(n)$ – exponenciálne zrýchlenie

Optimálne a efektívne paralelné algoritmy

nech n označuje veľkosť problému

NC – Nick Pippenger's Class
trieda úloh riešiteľných v polylogaritmickom čase s polynomiálnym počtom procesorov (vzhľadom k n)

- nech S je problém, pre ktorý najlepší sekvenčný algoritmus má časovú zložitosť $T(n)$. Paralelný algoritmus A riešiaci problém S v čase $t(n)$ s $p(n)$ procesormi sa nazýva

  optimálny ak

  1. $t(n) = polylog(n)$ – exponenciálne zrýchlenie
  2. $p(n) \cdot t(n) \in O(T(n))$ – práca vykonaná algoritmom A je ekvivalentná sekvenčnej zložitosti.

Optimálne a efektívne paralelné algoritmy

nech n označuje veľkosť problému

NC – Nick Pippenger's Class
trieda úloh riešiteľných v polylogaritmickom čase s polynomiálnym počtom procesorov (vzhľadom k n)

- nech S je problém, pre ktorý najlepší sekvenčný algoritmus má časovú zložitosť $T(n)$. Paralelný algoritmus A riešiaci problém S v čase $t(n)$ s $p(n)$ procesormi sa nazýva

  optimálny ak
  1. $t(n) = polylog(n)$ – exponenciálne zrýchlenie
  2. $p(n) \cdot t(n) \in O(T(n))$ – práca vykonaná algoritmom A je ekvivalentná sekvenčnej zložitosti.

  efektívny ak

Optimálne a efektívne paralelné algoritmy

nech n označuje veľkosť problému

NC – Nick Pippenger's Class
trieda úloh riešiteľných v polylogaritmickom čase s polynomiálnym počtom procesorov (vzhľadom k n)

- nech S je problém, pre ktorý najlepší sekvenčný algoritmus má časovú zložitosť $T(n)$. Paralelný algoritmus A riešiaci problém S v čase $t(n)$ s $p(n)$ procesormi sa nazýva

  optimálny ak
  1. $t(n) = polylog(n)$ – exponenciálne zrýchlenie
  2. $p(n) \cdot t(n) \in O(T(n))$ – práca vykonaná algoritmom A je ekvivalentná sekvenčnej zložitosti.

  efektívny ak
  1. $t(n) = polylog(n)$ – exponenciálne zrýchlenie

Optimálne a efektívne paralelné algoritmy

nech n označuje veľkosť problému

NC – Nick Pippenger's Class
trieda úloh riešiteľných v polylogaritmickom čase s polynomiálnym počtom procesorov (vzhľadom k n)

- nech S je problém, pre ktorý najlepší sekvenčný algoritmus má časovú zložitosť $T(n)$. Paralelný algoritmus A riešiaci problém S v čase $t(n)$ s $p(n)$ procesormi sa nazýva

  optimálny ak
  1. $t(n) = polylog(n)$ – exponenciálne zrýchlenie
  2. $p(n) \cdot t(n) \in O(T(n))$ – práca vykonaná algoritmom A je ekvivalentná sekvenčnej zložitosti.

  efektívny ak
  1. $t(n) = polylog(n)$ – exponenciálne zrýchlenie
  2. $p(n) \cdot t(n) \in O(T(n)) \cdot polylog(n)$ – práca vykonaná algoritmom A je ekvivalentná sekvenčnej zložitosti.

Optimálne a efektívne paralelné algoritmy

nech n označuje veľkosť problému

NC – Nick Pippenger's Class
trieda úloh riešiteľných v polylogaritmickom čase s polynomiálnym počtom procesorov (vzhľadom k n)

- nech S je problém, pre ktorý najlepší sekvenčný algoritmus má časovú zložitosť $T(n)$. Paralelný algoritmus A riešiaci problém S v čase $t(n)$ s $p(n)$ procesormi sa nazýva

  optimálny ak
  1. $t(n) = polylog(n)$ – exponenciálne zrýchlenie
  2. $p(n) \cdot t(n) \in O(T(n))$ – práca vykonaná algoritmom A je ekvivalentná sekvenčnej zložitosti.

  efektívny ak
  1. $t(n) = polylog(n)$ – exponenciálne zrýchlenie
  2. $p(n) \cdot t(n) \in O(T(n)) \cdot polylog(n)$ – práca vykonaná algoritmom A je ekvivalentná sekvenčnej zložitosti.

  pojem efektívneho paralelného algoritmu je robustný vzhľadom k "rozumným modelom", pretože jednotlivé paralelné modely sa dajú navzájom simulovať v polylogaritmickom čase