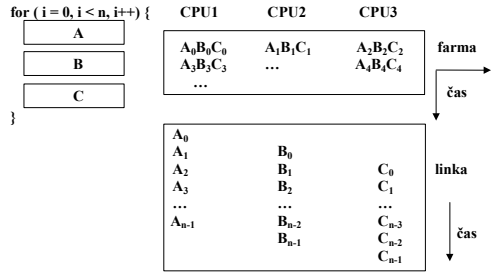
Týden 9
Vzory programů: linka, farma, D&C

Vzory MP programů: linka a farma

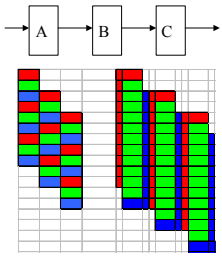


Kdy je účelné použít linku

Jestliže je dán problém, který lze rozdělit do řady sekvenčních úloh, pak pomocí řetězeného zpracování linkou získáme vyšší rychlost

1. Jestliže se má zpracovat řada datových položek, každá vyžadující násobné operace
2. Jestliže doba zpracování v jednotlivých stupních je souměřitelná
3. Jestliže komunikace mezi stupni je nízká nebo může běžet současně s výpočtem

SW řetězení na lince procesorů



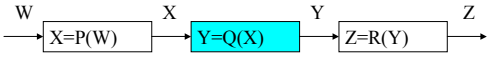
Každý stupeň *sekvenčně:*
- čte data zleva
- pracuje s daty
- posílá výsledek doprava

Překrytí komunikace s výpočtem *softwarovým řetězením*:

Každý stupeň *paralelně*:
- čte nová data zleva
- pracuje s aktuálními daty
- posílá výsledek zpracování předchozích dat doprava

Je třeba, aby stupně linky měly vyváženou zátěž !

Řetězená linka v MPI a Transimu



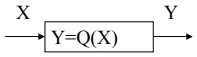
MPI

```
While(Not_Done) {
    MPI_Irecv(NextX, .........);
    MPI_Isend( PreviousY, ........);
    CurrentY = Q( CurrentX );
}
```

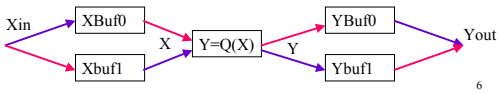
Transim

```
WHILE  notdone
    PRI PAR
      SEQ | high
        PAR
          SEQ | rx
              chin ? NextX
          SEQ | tx
              chout ! PreviousY
      SEQ | low
          CurrentY := Q(CurrentX)
```

Stupeň linky s dvojitými bufery




```
While(Not_Done) {
    if (X==XBuf0)  {X=XBuf1; Y=YBuf1; Xin=XBuf0; Yout=YBuf0;}
    else           {X=XBuf0; Y=YBuf0; Xin=XBuf1; Yout=YBuf1;}
    MPI_Irecv(Xin,..........., recv_handle);
    MPI_Isend(Yout,........, send_handle);
    Y=Q(X);
    MPI_Wait(recv_handle, recv_status);
    MPI_Wait(send_handle, send_status);
}
```



Příklad: n-tá mocnina matice A linkou

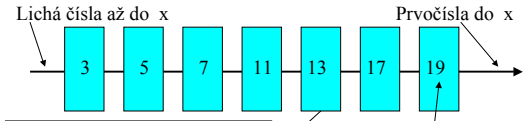
Dáno: 3-stupňová linka, vypočítat 10-tou mocninu čtvercové matice A [n×n], použít sw řetězení na překrytí komunikace a výpočtů.

Vyrovnání zátěže stupňů:
- 8. mocnina A je snadná, každý stupeň počítá čtverec matice na vstupu
- 10. mocnina $A^{10} = A^2 * A^4 * A^4$

Je třeba vyvážit komunikaci mezi stupni i zátěž jednotlivých stupňů:

A--> cpu[1]	---------->	cpu[2]	---------->	cpu[3]	---->A^{10}
1. A^2	A^2	A^4, $^1/_2 A^8$	A^2, A^4, $^1/_2 A^8$	$^1/_2 A^8$, A^{10}	59%
2. A^2, $^1/_2 A4$	A^2, $^1/_2 A4$	$^1/_2 A4$, $^1/_2 A8$	A^2, A^4, $^1/_2 A^8$	$^1/_2 A^8$, A^{10}	59%
3. A^2, $^1/_2 A4$	A^2, $^1/_2 A4$	$^1/_2 A4$, A^8	A^2, A^8	A^{10}	74%
4. A^2, $^1/_3 A4$	A^2, $^1/_3 A4$	$^2/_3 A4$, $^2/_3 A8$	A^2, A^4, $^2/_3 A^8$	$^1/_3 A^8$, A^{10}	56%

Příklad: Linka procesů jako síto na prvočísla

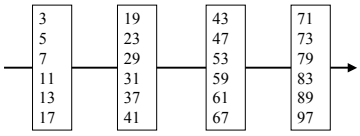



```
WHILE TRUE
  SEQ
    chin ? next
    IF
      next REM p = 0
        SKIP -- neni prvocislo
      TRUE
        chout ! next -- muze byt
```

Prvočísla do $\sqrt{x}$

x = 10 000: 24 prvočísel do 100 (3,5,7,..., 97)

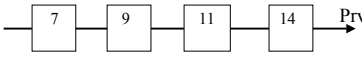
Linka procesorů na prvočísla (do 10 000)



4 procesory, každý 6 prvočísel

Algoritmus bez dělení:
číslo x na vstupu filtru se porovnává s
p*p, (p+2)*p, (p+4)*p,...(tj. postupně rostoucími lichými násobky p).
- Je-li x větší než aktuální násobek, zvýší se násobek o tolik, aby x bylo právě rovno nebo menší než jeho nová hodnota;
- je-li x menší než aktuální násobek, přejde se na další prvočíslo;
- Je-li x právě rovno aktuálnímu násobku , x není prvočíslo.

Vyrovnání zátěže procesorů (do 32 761)



Prvočísla do 32761

4 procesory, celkem 7 + 9 + 11 + 14 = 41 prvočísel

Odhad zátěže: z teorie čísel plyne, že zátěž filtru x je přibližně 1/log x.

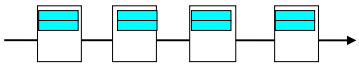
Celková zátěž celé linky

$$Z = \frac{1}{\log 3} + \frac{1}{\log 5} + \frac{1}{\log 7} + \ldots + \frac{1}{\log 173} + \frac{1}{\log 179} + \frac{1}{\log 181}$$

se musí rozdělit co nejrovnoměrněji na 4 procesory, tj. každý procesor si vezme tolik prvočísel, aby jeho zátěž byla ≈ *Z*/4.

Prvočísla do 10^6 na lince 4 procesorů - vliv různých technik na účinnost

	S	E%
1. Stejný počet prvočísel na procesor => různá zátěž, každé prvočíslo jeden proces	2,62	66
2. Statické vyrovnání zátěže, relativní E	3,66	92
3. je-li *next* > p^2, netestuje se; absolutní E	1,58	40
Vyloučení paralelismu na procesoru:		
4. Jeden proces testuje blokem prvočísel, vynechání testu *next*	2,79	70
5. Zařazení 2 buferů (příp. delší zprávy)	3,19	80



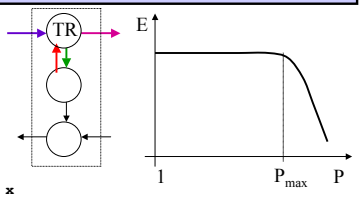
Největší známé prvočíslo
Elektronický časopis HPCwire, 7.prosince 2001

- Nejnovější prvočíslo lze zapsat ve tvaru 2^13,466,917 -1, obsahuje 4,053,946 číslic a jeho ruční zápis by trval kolem 3 týdnů. (Na nalezení prvočísla s více než 10^7 číslic je vypsána odměna USD 100 000).
- bylo objeveno Michaelem Cameronem, 20-letým účastníkem projektu Great Internet Mersenne Prime Search (Gimps).
- 130,000 dobrovolníků z řad domácích uživatelů, studentů, škol, univerzit a podniků z celého světa přispělo do Gimps; nalezení nového prvočísla spotřebovalo 13,000 roků strojového času během dvou let nepřetržité práce.
- Mersennova prvočísla jsou důležitá pro teorii čísel a mohou napomoci při vývoji neporazitelných kódů a šifrování zpráv.

Farma procesorů

- po inicializaci rozděluje řídící procesor (farmář) *identické* úlohy s rozdílnými daty (může jít o iterace smyčky) na požádání (při odevzdání výsledku) ostatním procesorům; doba úlohy je proměnlivá;
- V lineární farmě není třeba v balíčku dat udat adresu, balíček dat bude doručen prvnímu čekajícímu dělníku v řadě;
- aby se snížily prostoje, obsahuje každý dělník bufer pro další balíček (balíčky) dat;
- v systému není nikdy víc úloh než je prostoru pro jejich uložení;
- aby se zkrátily komunikační cesty a počet kopírování zpráv, lze použít stromovou konfiguraci farmy. Pak je ale třeba do balíčku dat přidat adresu (identitu cílového procesoru) a někteří dělníci musí kromě své práce i směrovat zprávy. Tím se komplikuje implementace.

Lineární farma procesorů bez buferů



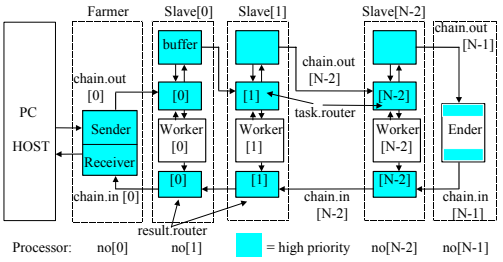

```
INT x,v:
SEQ |task.router
  WHILE TRUE
    SEQ
      input[i] ? x
      PRI ALT
        work.req ? v
          work.out ! x |task.size
        TRUE&SKIP
          output[i+1] ! x |task.size
```


Možnosti zlepšení účinnosti farmy

- řídící procesor musí také pracovat
- dva (nebo více) buferů pro malé balíčky dat mohou být lepší než jen jeden
- snížení granularity úloh, posílání více úloh a též více výsledků v jednom balíčku
- všechny komunikace by měly být ve vysoké prioritě na obou koncích (PRI PAR)
- směrovač úloh přednostně vyřizuje požadavky lokálního dělníka a teprve pak žádosti z buferu (pořadí příkazů v PRI ALT)

Změny ve výkonnosti - zhoršení - lze pozorovat
- při změně PRI PAR na pouhé PAR
- přehozením pořadí komponent PRI ALT

Lineární farma - graf procesů



Stromová konfigurace farmy

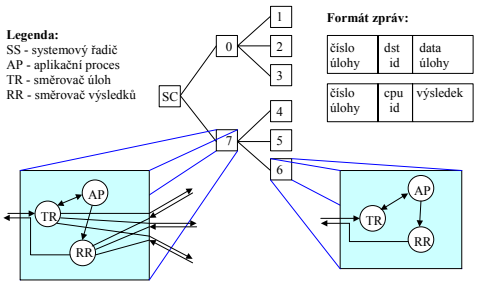


Schéma aplikace farmář-dělníci (MPI)

```
main(int argc, argv)
int argc;
char *argv[ ];
{
int myrank
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank==0)  {
    farmer( );
   } else {
     worker( );
   }
```

Příklad na farmu: Mandelbrotova množina

Mandelbrotova množina M je množina bodů *c*, v komplexní rovnině, které jsou kvasi-stabilní (tj. $|z_n| < \infty$ pro libovolné n) při iteracích

$z_0 = 0, \quad z_1 = z_0^2 + c,$

$z_2 = z_1^2 + c = c^2 + c,$

$z_3 = c^4 + 2c^3 + c^2 + c,$

$...., \quad z_{n+1} = z_n^2 + c$

$z = a + bi$

$z^2 = a^2 - b^2 + 2ab$

$nové\ \mathrm{Re}\, z = a^2 - b^2 + \mathrm{Re}\, c$

$nové\ \mathrm{Im}\, z = 2ab + \mathrm{Im}\, c$

- Platí:
  Jestliže ∃ n takové, že $|z_n| > 2$, pak $c \notin M$.
- Počet iterací nutných pro tento test lze odlišit barevně, čímž vzniká v komplexní rovině Mandelbrotův fraktál.
- Test pro různé body trvá různě dlouho
- Farmář rozesílá jednotlivé řádky oblasti volným dělníkům ve farmě

Sekvenční výpočet jednoho bodu, vrací počet iterací

```
int cal_pixel(complex c)
{
int count, max;
complex z;
float temp, modul2;
max = 256;
z.real = 0; z.imag = 0;
count = 0; /* number of iterations */
do {
   temp = z.real * z.real - z.imag * z.imag + c.real;
   z.imag = 2 * z.real * z.imag + c.imag;
   z.real = temp;
   modul2 = z.real * z.real + z.imag * z.imag;
   count++;
} while ((modul2 < 4.0) && (count < max));
return count;
}
```

```
structure complex {
   float real;
   float imag;
};
```


Programovací vzor: Brašna úloh

- Podobný vzor jako farma pro zasílání zpráv, ale procesy si berou úlohy z brašny a mohou je tam i dávat (např. metodou divide and conquer (DC, rozděl a panuj) se rekurzivně generují úlohy menší velikosti, řešitelné paralelně).
- Brašna je sdílena dvěma nebo více procesy worker.
- Vhodné pro dynamické vyrovnávání zátěže když zpracování podúloh trvá různě dlouho.
- schedule (dynamic, chunk_size) v OpenMP je také brašna úloh!

```
while (true) {
   vezmi si úlohu z brašny;
   if (nejsou další úlohy)
      break;     #vyskoč ze smyčky while
   proveď úlohu, případně generuj nové;
}
```


Paralelní MMM (brašna úloh)

Úlohy představují výpočet jednoho (případně několika) řádků výsledné matice c; na počátku obsahuje brašna n úloh, ty se mohou brát v libovolném pořadí. Stačí počítat hotové řádky.

```
int nextRow = 0;  #inicializace brašny
double a[n,n], b[n,n], c[n,n];

process Worker[w = 1 to P] {
  int row;
  double sum; #pro skalární součiny
  while (true) {
    <row = nextRow; nextRow++;> #vezmi si úlohu
    if (row >= n)
      break;
    z řádku a[row,*] a matice b[n,n] spočti řádek c[row,*];
  }
}
```

<kritická sekce, exkluzivní přístup>

Divide and Conquer, D&C (Rozděl a panuj)

- problém je rozdělen na b podproblémů (b, větvící faktor)
- r je poměr velikostí původního problému a podproblému
- f(n) je počet kroků potřebných na rozdělení původního problému na b podproblémů velikosti n a na kombinaci b dílčích řešení do jednoho

Musí platit:

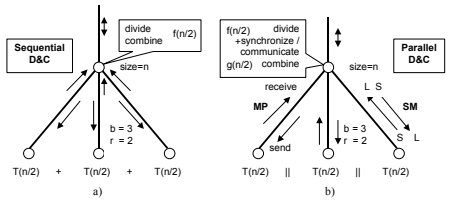
$T^{seq}_{DC}(n) = b\, T(n/r) + f(n/r) < T^{seq}(n);$

Je-li potom $T^{seq}_{DC}(n/r) < T(n/r)$, můžeme aplikovat strategii D&C strategy rekurzivně na b podproblémů o velikosti n/r, podobně na b^2 podproblémů o velikosti n/r^2, atd.

D&C, generická rekurzivní procedura

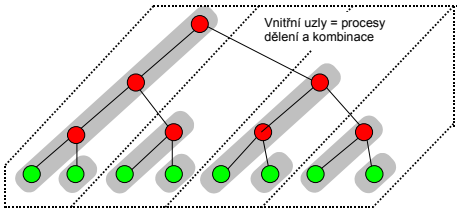
```
procedure D&C (n: input size)
  begin
    if n <= n0 then
      solve a problem without further subdivision;
    else
      split into b sub-problems each of size n/r;
      for each of b sub-problems do D&C(n/r);
      combine the resulting b sub-solutions to
      produce the solution to the original problem;
    end if;
end D&C;
```

D&C sekvenčně a paralelně



$T^{seq}_{DC}(n) = b\,T(n/r) + f(n/r)$ $\qquad$ $T^{par}_{DC}(n) = T(n/r) + f(n/r) + g(n/r)$,

D&C, alokace procesů



listy = výpočetní procesy

Mapování 8 procesů na 8 procesorů (šedé pruhy) nebo na 4 procesory (čárkovaně)

Příklad: násobení polynomů podle Karatsuby

Původní násobení 2 polynomů je převedeno na 3 násobení polynomů poloviční délky; 3 součiny jsou vhodně posunuty a sečteny, aby vytvořily žádaný součin.

$T^{seq}_{DC}(n) = b\,T(n/r) + f(n/r) = 3c\,(n/2)^2 + dn$, $\quad b = 3, r = 2$,

c = cena násobení, d = cena slučování

$T^{seq}_{DC}(n) = 3c\,(n/2)^2 + dn < T(n) = cn^2$, $n > 4d/c$

$T^{seq}_{DC}(n/2) < T(n/2)$,

$T^{seq}_{DC}(n/4) < T(n/4)$,

..... dělení pokračuje až

$T^{seq}_{DC}(n/2^i) \geq T(n/2^i)$ neboli $i \geq \log_2 nc/(4d)$;

$n_0 = n/2^i \leq 4d/c$ je kritická prahová velikost problému, pod ní další dělení nepřináší zisk

Násobení polynomů (Karatsuba)

$$g = g_l B^l + g_{l-1} B^{l-1} + \ldots + g_0, \qquad g = aB^l + b,$$

$$h = h_l B^l + h_{l-1} B^{l-1} + \ldots + h_0, \qquad h = cB^k + d;$$

$$x = (a+b)(c+d), \qquad y = ac, \qquad z = bd;$$

$$gh = yB^{2k} + (x - y - z)B^k + z.$$

$T^{seq}_{DC}(n) = O(n^{1.59})$

Paralelní D&C častější než sekvenční

Příklad: $T(n) = An$, $\quad f(n) = Bn$, $\quad r = 2$, $\quad b = 2$

$T^{seq}_{DC}(n) = 2\,T(n/2) + f(n/2) = (2A + B)n/2 > An = T(n)$,

čili nemá smysl, ale paralelní D&C je užitečné,

$$T^{par}_{DC}(n) = A\tfrac{n}{P} + Bn\left(\tfrac{1}{P} + \tfrac{1}{P/2} + \ldots + \tfrac{1}{2}\right) = A\tfrac{n}{P} + Bn\left(1 - \tfrac{1}{P}\right)$$

$$S = P\frac{A}{A + B(P-1)} \qquad E = \frac{A}{A + B(P-1)}.$$

Pro $E \geq 50$ %, počet procesorů P je omezen na $P \leq 1 + A/B$.

Tam kde naivní sekvenční D&C nevede ke zrychlení, paralelní D&C zrychlení dává !

Násobení matic D&C na 8 procesorech

Jedno násobení matic o rozměru n × n lze nahradit 8 násobeními a 4 součty matic o rozměru n/2 × n/2. Zanedbáme-li komunikaci, je

$$S = \frac{n^3 MAC}{(\frac{n}{2})^3 MAC + 4(\frac{n}{2})^2 ADD} = 8\frac{n}{n + 8\frac{ADD}{MAC}}$$

Další sekvenční dělení problému na jednotlivých procesorech nemá smysl, protože $T^{seq}_{DC}(n/2) > T^{seq}(n/2)$. Chytřejší Strassenův algoritmus používá jen 7 součinů matic n/2 × n/2 a další dělení umožňuje. Jeho složitost je $T^{seq}_{DC}(n) = O(n^{2.81})$ místo tradičního $O(n^3)$.

Násobení matic – Strassenův algoritmus

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \quad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \quad C = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$P_1 = A_{1,1}(B_{1,2} - B_{2,2}), \quad P_2 = (A_{1,1} + A_{1,2})B_{2,2},$$

$$P_3 = (A_{2,1} + A_{2,2})B_{1,1}, \quad P_4 = A_{2,2}(B_{2,1} - B_{1,1}),$$

$$P_5 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}),$$

$$P_6 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2}),$$

$$P_7 = (A_{1,1} - A_{2,1})(B_{1,1} - B_{1,2})$$

$$C_{1,1} = -P_2 + P_4 + P_5 + P_6, \qquad C_{1,2} = P_1 + P_2,$$

$$C_{2,1} = P_3 + P_4, \qquad C_{2,2} = P_1 - P_3 + P_5 - P_7$$

31