Týden T13
Datově-paralelní programování

Datově-paralelní programování (DP)

Rysy:

- pracuje s *poli* a *seznamy*, zpracování ve stylu SPMD (SIMD)
- kompilátor provádí rozčlenění, vkládá komunikace
- stručné programy, bez smyček přes prvky polí, přenositelnější, srozumitelnější
- nevhodné když se struktura komunikací a zátěž uzlů mění nepředvídatelně při běhu programu

Užitečné pro:

- stroje SIMD (implicitní synchronizace)
- vektorové (ko)procesory
- MIMD v režimu SPMD
- hybridní architektury SIMD/MIMD
- svazky pracovních stanic

Jazyky pro datově-paralelní programování

Hlavně pro vědecké výpočty:

- F90 http://www.fee.vutbr.cz /MIRROR/DBPP/
- HPF http://www.fee.vutbr.cz /MIRROR/DBPP/
- PC++ http://www.extreme.indiana.edu/sage/overview.html
- Paralaxis http://www.ee.uwa.edu.au/~braunl/parallaxis
  ftp://ftp.informatik.uni-stuttgart.de/pub/p3
- F95, F2001
- HPF-2

Predikce výkonnosti (HPF):
Journal of Parallel and Distributed Computing, Jan. 2000, pp.17-47.

Použitá notace vycházející z Fortranu (HPF)

- indexování podrobně: *začátek : konec : rozestup,*
  ve Fortranu začíná indexování od 1 !!
- deklarace: real : : A(10, 20), B(10)
  integer : : V(5)
- přístup k elementům pole:

5. řádek A:	A(5, :)
sloupce 1,5,9,13,17 pole A:	A(: , 1: 20 : 4)
jen sudé sloupce pole A:	A(1: 10 : 1, 2 : 20 : 2)

- čtení nebo zápis do pole je nedělitelná operace :

! inicializuj V na [10, 20, 30, 40, 50]
V(2 : 4) = V(1 : 3) + V(3 : 5)
! [10, 20, 30, 40, 50]
! + [10, 20, 30, 40, 50]
! V = [10, 40, 60, 80, 50]

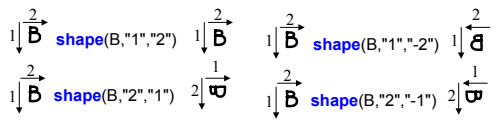
! nikoliv takto:
[10, 40, 80, 40, 50]
+ [10, 20, 30, 40, 50]
V = [10, 40, 80, 130, 50]

Datově-paralelní funkce

real : : A(10,20), B(10)

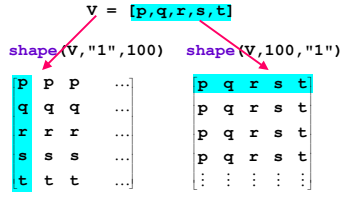
Funkce

index(n) vytvoří vektor (1, 2,…, n)
scalar(a, n) vytvoří vektor (a, a,…, a)
size(A, "1") vrátí 10 (rozměr v 1. dimenzi)
A = 5 vytvoří matici 10 x 20 samých "5"
(nafouknutí skaláru, *scalar promotion*)
shape( ) znaménko minus = reverzuj podél příslušné osy

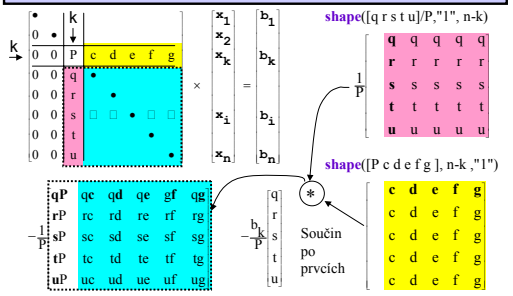


Tvarování (Shaping)

Funkce `shape(A,"2","1")`
vyrobí transpozici matice A (sloupce ←→ řádky)



Gaussova eliminace v maticovém tvaru



Gaussova eliminace v DP notaci

! Redukce matice A na horní trojúhelníkovou matici:

```
do k = 1, n -1
    A = A - shape [ A( k+1 : n, k ) /A(k,k), "1", n - k ] *
            shape [ A( k, k+1: n ), n - k, "1" ]
    B = B - A( k+1 : n, k ) / A(k,k) * B(k)
```

Součin po prvcích

! Zpětná substituce -
! redukce horní trojúhelníkové matice:

```
do  i = n, 2, -1
    X( i ) = B( i ) / A( i, i )
    B(1: i -1) = B(1: i -1) - A(1: i -1, i ) * X( i )
```

! Násob sloupec A(: , i) známou hodnotou X(i)
! a odečti od pravé strany B(:)

Paralelní podmíněný příkaz (maskování)

```
where  TRUE
   execute
end where

do i = 1, m                  →
   do j = 1, n
      if A( i, j ) >= 0  then
         A( i, j ) = sqrt ( A( i, j ) )
      end if
   end do
end do
```

```
where A >= 0
   A = sqrt (A)
end where
```


Paralelní podmíněné příkazy - pokrač.

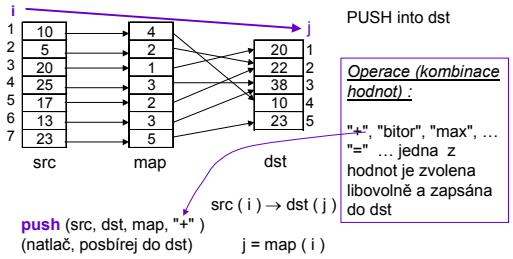
```
where    A < MIN_VAL
   A = MIN_VAL
elsewhwere   A > MAX_VAL
   A = MAX_VAL
end where
```

Vnořování:

```
where X1   do
   A1
   where Y   do
      A2
   end where
elsewhere   X2   do
      A3
end where
```

Where	do:
X1	A1
X1.and.Y	A2
.not.X1.and.X2	A3

Mapování jednoho pole na jiné

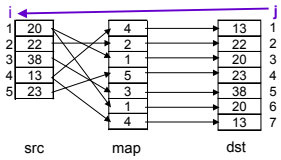


Operace (kombinace hodnot) :

"+", "bitor", "max", …
"=" … jedna z hodnot je zvolena libovolně a zapsána do dst

push (src, dst, map, "+")
(natlač, posbírej do dst)

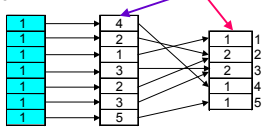
Mapování jednoho pole na jiné - pokrač.



pull (src, dst, map)
(pošli, rozhoď do dst)

Příklad: histogram

```
procedure histogram (map, h)
    integer : : map( : )
    integer : : h( : )
    h = 0
    call push( scalar(1, size(map,"1" )), h, map, "+")
end procedure
```



Složitější příklad převodu do DP programu

```
real : : A(100,100), V(20)
integer : : i, j
do i = 1, 100, 2
    do j = 1, 100, 2
        if A( i, j ) > = 0 then
            A( i, j ) = A( i, j ) - V( ( i -1)/5 +1)
        end if
    end do
end do
```

```
real : : temp(100)
pull( V, temp, ( index(100)-1) / 5 + 1)   ! z V do temp
where A > = 0 do
   A(1:100:2,1:100:2) = A(: , :) - shape( temp,"1",100)
end where
```


Posouvání

c, kruhový posuv — kladný posuv : ↓ → rostoucí indexy
p, planar shift
n, žádný posuv — záporný posuv : ↑ ← klesající

shift (A, "p", 2, "c", -1) :

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \xrightarrow{p} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ a & b & c & d \\ e & f & g & h \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ b & c & d & a \\ f & g & h & e \end{bmatrix}$$

Křivení matic (skewing)

křivení podle +1. osy: -2. osy:

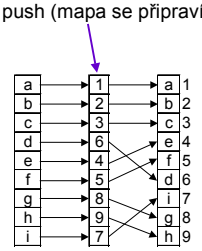
$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \rightarrow A^{(1)} = \begin{bmatrix} a & h & f \\ d & b & i \\ g & e & c \end{bmatrix}, \quad A^{(-2)} = \begin{bmatrix} a & b & c \\ e & f & d \\ i & g & h \end{bmatrix}$$

$A^{(-2)}$ (1 : i) = A(1: i) ! první řádek beze změny
$B^{(-1)}$ (1 : i) = B(1: i) ! první sloupec beze změny
do i = 2, N
  $A^{(-2)}$ (i , :) = **shift** (A(i, :), "c", 1- i)
  $B^{(-1)}$ (: , i) = **shift** (B(:, i), "c", 1- i)
end do

Křivení matic v jednom kroku

použitím funkce push (mapa se připraví dopředu):

A $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$ $A^{(-2)}$ $\begin{bmatrix} a & b & c \\ e & f & d \\ i & g & h \end{bmatrix}$



Redukce

= operace push do jedoprvkového pole
- kombinuje prvky pole použitím komutativní a asociativní operace

  * , + , or , bitand , min, …
- funkce *reduce* :
  např. skalární součin c dvou vektorů A(:), B(:)

  c = **reduce** (A(i)*B(i), "+")
- implementace:
  sekvenční: vnořené smyčky
  paralelní: redukční strom, ve 3D polích
  $\log_2 n_1 + \log_2 n_2 + \log_2 n_3 = \log_2 n_1 n_2 n_3$ kroků

Implementace DP redukce

```
real : : A(16)
stride = 2              ! stride = rozestup
do 1, 4
    A(1: 16: stride) =
    A(1: 16: stride) + A( (1+stride/2) : 16 : stride)
    stride = 2 * stride
end do
```

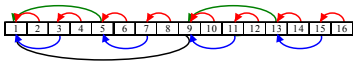
vzdálenost:

1: A(1:16: 2) = A(1:16: 2) + A(2:16: 2)

2: A(1:16: 4) = A(1:16: 4) + A(3:16: 4)

4: A(1:16: 8) = A(1:16: 8) + A(5:16: 8)

8: A(1:16: 16) = A(1:16:16) + A(9:16:16)



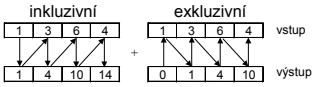
Paralelní prefix

Počítá posloupnost částečných výsledků určité operace na všech podposloupnostech elementů nějakého vektoru;

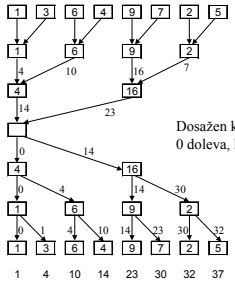
redukce = vlastně speciální případ paralelního prefixu;

funkce **scan** (vektor, operace, inc/exc, up/dn) - vrací vektor (T F, T F)

Příklad: Paralelní prefixový součet = paralelní scan:



Stromový paralelní scan za 2 log n kroků



Fáze průchodu nahoru:
ulož hodnotu z levého předchůdce,
pošli výsledek nahoru

Dosažen kořen stromu:
0 doleva, hodnota přijatá zleva jde doprava

Fáze průchodu dolů:
hodnota shora jde doleva,
součet z uloženou hodnotou doprava

Dosaženy listy:
sečti hodnotu shora a uloženou hodnotu

Paralelní řazení "Radix Sort"

n kroků jestliže jsou hodnoty reprezentovány n bity.
Příklad:
n = 4, podtržená hodnota má bit $2^i = 0$;

← podtržené hodnoty, zbylé hodnoty →

bit 2^0 (LSB)	2	7	5	6	9
2^1	2	6	7	5	9
2^2	5	9	2	6	7
2^3	9	2	5	6	7
	2	5	6	7	9

DP řazení radix sort, 1. krok

i	1	2	3	4	5
src	2	7	5	6	9
binárně	0010	0111	0101	0110	1001
maska	T	F	F	T	F
žádoucí index	1	3	4	2	5
pozice nul	1	0	0	1	0
scan →	1	1	1	2	2
pozice jedniček	0	1	1	0	1
scan ← w	3	3	2	1	1
invertuj 5-w[i])+1	3	3	4	5	5
mapa	1	3	4	2	5
push					
dst	2	6	7	5	9

DP procedura radixsort

```
Procedure radixsort (orig)
   integer : : orig ( : ), one(dim), zero(dim), dim = size (orig,1)
   logical : : mask (dim)
   do k = 1, bits_per_value
      mask = (orig .bitand. shift (1, k - 1) ==0) ! ← posuv "1" o 0,1,2,... míst
      where mask do   zero = 1; one = 0        ! pozice nul
          else        zero = 0; one = 1        ! pozice jedniček
      end where
      zero = scan( zero,"+", .true. , .true.)   ! inkluzivně, →
      one  = scan( one, "+", .true. , .false.)  ! inkluzivně, ←
          where .not. mask  do
              zero = dim + 1 - one              ! mapa  pro push
          end where
          call push (orig., one, zero, "=" )    ! push orig pres zero do one
          orig = one                            ! seřazený vektor do orig
   end do
end procedure
```

Segmentovaný prefix

- oddělený paralelní prefix je proveden na každém segmentu vektoru

5F	3T	4F	6F	2T	8F	hodnoty, příznaky

5	3	7	13	2	10	inkluzivní +
0	0	3	7	0	2	exkluzivní +

T = značka začátku segmentu;
implementace: k hodnotám je připojen tzv. bariérový bit

25

Četnost výskytů

Kolikrát se určitá hodnota vyskytuje v poli ?
Nalezení počtu maximálních hodnot:

```
real function num_max_1D(A)
    real : : A( : )
    integer : : temp (size(A,1))

    where A == reduce(A, ".max.") do
            temp = 1
    else
            temp = 0
    end where
    return  reduce (temp, "+")
end function
```

26

Lokalizace hodnoty v poli

Pro nalezení místa výskytu určité hodnoty v poli A(:, :, :) :

function **locate** (array_name, value, d(3))

uloží souřadnice do d(3) a vrací .true., nebo .false. (není-li tam taková hodnota).

Lokalizace hodnoty v 1D-poli v čase O (log n) :

```
integer function locate_1D ( V )

    logical : : V ( : )              ! vytvořeno z  A( : ) použitím locate
    integer : : n = size (V,1)
    integer : :  i, step

    V = scan( V, "or", .true., .true.)        ! inkluzivně, →
```

27

Lokalizace v 1D-poli - pokrač.

```
if V(1) then
    i = 1
else                                  ! binární hledání:
    i = n/2
    step = n/4            ! V = FFFFFFTTTTTTTTTTT
    do while .not. (.not. V( i - 1) .and. V( i ) )
        if V ( i ) then            !  jsme v části TTTT
            i = i - step
        else                  !  jsme v části FFFF
            i = i + step
        end if
        step = step/2
    end while
end if
return i
```

28

Konec PPP !
Zkouška 6. 6. 2002,
10:00 –12:00 hod
- s pomůckami

29