Rémi
Pellerin
M1IF

APPD january 2016

1/ Algorithms for PRAM.

1.1/ Matrix generation

1/ Some recalls on what we are dealing with:

PRAM = shared memory among processors, no communications between processors (what would be the interest since they can "communicate" through the shared memory?)

CRCW: concurrent write and concurrent read allowed. Intuitively, there is no problem with concurrent read, but for concurrent write, we have to deal with inconsistent writes (see your course for more details about fusion mode, priority mode etc... but here we don't care!)

Each processor will be given a matrix case to compute!

Let A be the matrix represented by an array of array, all of size n and indexed from 0 to n-1

Algo(A, x, y)
    for all p in // do
        i ← p / n   ($\in [\![0, n-1]\!]$)
        j ← p % n   ($\in [\![0, n-1]\!]$)
        A[i][j] ← x[i] * y[j]

Complexity : $O(1)$

Of course this algorithm is optimal. To show it properly let's compute its efficiency :

Recall that efficiency $e_p(n) = \frac{T_{seq}(n)}{p \times T_{//}(p,n)}$

where $T_{seq}(n)$ is the execution time of the best sequential algorithm and $T_{//}(p,n)$ the execution time for the best // algorithm using p processors.

$T_{seq}(n) = n^2$
$T_{//}(p,n) = 1$ and $p = n^2$ so $e_p(n) = 1$ which is what we can call "optimal".

2) This time we cannot use concurrent write. Its not a problem since the last algorithm doesn't use it!

3) However, our first algo uses a lot of concurrent reads!
Let's try to design an optimal algorithm.
The best we can do is to compute A in n steps reading all x and all y during each step.

We can assume that at each step, p_{ij} (where $i = p/n$ and $j = p \% n$) will always read $x[j]$

and $y[\sigma(i)]$ where σ is a permutation of $[\![0, n-1]\!]$.

By doing that, there won't be any concurrent read (and no concurrent write since p_{ij} will only write $A[i][j]$.

We have to find n permutations σ. We can take for example the permutation matrix

$$D_1 = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 1 & & & 0 \end{bmatrix} = \left(\delta_{i+1 [n], j}\right)$$

which corresponds to $\sigma(i) = i+1\ [n]$

One can show that $D_1^2 = \begin{bmatrix} 0 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 0 & 0 \\ 1 & & & & 0 \end{bmatrix}$: this is

kind of a shifting: $D_1^n = I_n = \begin{bmatrix} 1 & & (0) \\ & \ddots & \\ (0) & & 1 \end{bmatrix}$.

Algo (A, x, y):

$D_1 \leftarrow \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 1 & & & 0 \end{bmatrix}$

$D \leftarrow \begin{bmatrix} 1 & & (0) \\ & \ddots & \\ (0) & & 1 \end{bmatrix}$

for $k = 1$ to n do (we need n steps!)

$D \leftarrow D * D_1$ (can be computed easily by shifting 1's)

for all p in // do

$i \leftarrow p / n$

$j \leftarrow p \,\%\, n$

```
If  D[i][j] == 1 then
      A[i][j] ← x[i] * y[j]
Else
      pass
```

Complexity : $O(n)$

↳ I do not take into account the computation of D. Why? Because this can be "stored" in the algorithm since it do not depend on the inputs.

Efficiency : $e_p(n) = \dfrac{T_{seq}(n)}{p\,T_{//}(p,n)} = \dfrac{n^2}{n^2 \times n} = \dfrac{1}{n}$.

This algorithm is optimal in the sense that we cannot do better in EREW PRAM than computing n A[i][j]'s at each of the n (required) steps.

Q1 For the first algo, we will have to make each PU compute $\log(n)$ coefficients

$\rightarrow O(\log(n))$

Remark: We could have used Brent's theorem!
Indeed, our first algo runs in time $T = 1$ on a PRAM so it can be simulated with $p = \dfrac{n^2}{\log(n)}$ in

time $O\left(\dfrac{m}{p} + T\right) = O\left(\dfrac{n^2}{\frac{n^2}{\log(n)}} + 1\right) = O(\log(n))$.

← # of operations

Rémi
Bellenin
M1IF

For the second algo (EREW PRAM) we can keep the same strategy. Indeed, $p_1 = n^2$; $p_2 = \frac{n^2}{\log(n)}$

$0 < \log(n) \leq n$

$\frac{1}{n} \leq \frac{1}{\log(n)}$ so $n \leq \frac{n^2}{\log(n)} = p_2$

We have still enough processors to compute $x_{ik}[i][j]$ s each step. We just have to assign more coefficients to each PU.

This is once again confirmed by Brent's theorem:

$nn = n^2$; $p = \frac{n^2}{\log(n)}$

$T = n$

$\rightarrow O\left(\frac{n^2}{\frac{n^2}{\log(n)}} + n\right) = O(\log(n) + n) = O(n)$

5) Our algorithms are still optimal !

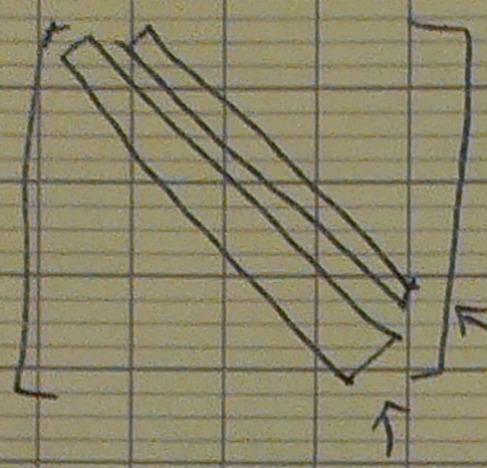
For the first one: $e_p(n) = \frac{n^2}{\frac{n^2}{\log(n)} \times \log(n)} = 1$

For the second one: $e_p(n) = \frac{n^2}{\frac{n^2}{\log(n)} \times n} = \frac{\log(n)}{n}$

which is greater than $\frac{1}{n}$ (for $p = n^2$). This means that the second algo do not require n^2 PU's (and this is intuitive ! In fact, n PU's are enough !)

6) We want to compute, for each $h \in [\![0, n-2]\!]$,
$S(h) = \sum_{i=0}^{n-2-h} A[i][i+h]$.

At each step, we will write in each $S(h)$, so exactly one PU will write in $S(h)$. Is it possible? Yes because we have enough PU (in fact, $\frac{n(n+1)}{2}$ would be enough)



Algo (A, S) S = array of n zeros
A = matrix (array of array)

```
for line = 0 to n-1 do
    for all p in // do
        i ← p/n
        j ← p %0 n
        if i == line then
(*)         S(j-i) ← S(j-i) + A[i][j]
        else
            pass
```

This algo computes line per line and at each step, only Pline.j writes in S(j-line).

(*) can be written in order to be strictly EREW.

$t \leftarrow S[j-i]$
$t' \leftarrow t + A[i][j]$
$S[j-i] \leftarrow t'$

Complexity : $O(n)$

1) $e_p(n) = \frac{\frac{n(n+1)}{2}}{n^2} = \frac{n+1}{2n} \sim \frac{1}{2n}$

If we only use $\frac{n(n+1)}{2}$ PU's,

$e_{\frac{n(n+1)}{2}}(n) = \frac{\frac{n(n+1)}{2}}{\frac{n(n+1)}{2} n} = \frac{1}{n}$ much better !

The first step is optimal but not the others. The design of an optimal algorithm seems to be a bit technical in PRAM!

Remark: In CRCW PRAM with fusion mode on +, we can do $O(1)$!

1.2) Changing the playground

This is just applications of Brent's theorem : Given an algorithm A performing m operations on a PRAM machine in time T (we do not need to know the number of PU's used), one can simulate (where one = Piene Prodic) A with p PU's in time

$$O\left(\frac{m}{p} + T\right).$$

1) $O\left(\frac{n \log(n)}{n} + O(\log(n))\right) = O(2\log(n)) = O(\log(n))$.

(We used that $\underset{n}{O}(O(u_n)) = O(u_n)$ and

$\underset{n}{O}(\lambda u_n) = O(u_n)$, $\lambda \in \mathbb{C}$.)

2) This time we cannot apply Brent's theorem directly because we switch from CREW PRAM to EREW PRAM...

However, we have the simulation theorem!

→ Any CRCW algorithm using p PU's has an execution time at most $O(\log(p))$ lower than the faster EREW algorithm using p PU's to solve the same problem.

Notice that a CREW is a CRCW!

So we can do $O(\log(\underset{\uparrow}{n^2})) \times O(\log(n))$

$= O(\log^2(n))$

(We used $O(u_n)\,O(v_n) = O(u_n v_n)$.)

3) In last question we have shown a result for $p = n^2$ PU's. Let's apply Brent's theorem,

$m = n\log(n)$

$T = O(\log^2(n))$

$p = n$

$\rightarrow O\left(\frac{n\log(n)}{n} + O(\log^2(n))\right) = O(\log^2(n))$