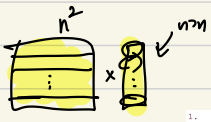
Matrix-Vector Multiplication

Problem
- multiply dense $n \times n$ matrix A with $n \times 1$ vector x $(A \times x)$

dense: useless entry가 거의없는 상태

output
- $n \times 1$ result vector y



Serial Complexity
- serial run time $= n^2 = n \times n$
  - # rows
  - n operations in each row

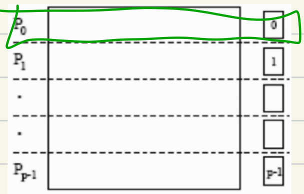
```
1.  procedure MAT_VECT ( A, x, y)
2.  begin
3.      for i := 0 to n - 1 do
4.      begin
5.          y[i]:=0;
6.          for j := 0 to n - 1 do
7.              y[i] := y[i] + A[i, j] x x[j];
8.      endfor;
9.  end MAT_VECT
```

Parallel approach
- row wise 1D
- Column wise 1D
- 2D partitioning

Matrix-Vector Multiplication : 1D Partitioning

Case of $p=n$

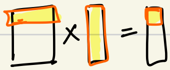
Partitioning

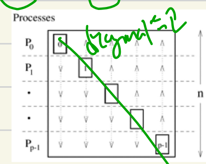


- each processor stores
  - row of matrix ($1 \times n$)
  - 1 element of vector

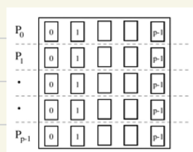
Computation

① y의 element 계산하려면 x vector 전체 element가 필요하다.

- 
- all-to-all broadcast → each node gets entire elements of vector
  → append해야됨



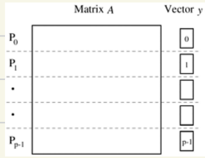
all-to-all broadcast



after broadcast

① Broadcast Time = $\Theta(p-1)$ → $\Theta(n)$

② Process P_i computes $y[i] = \sum_{j=0}^{p-1} A[i,j] \times x[j]$



Parallel Run Time

- all-to-all broadcast = $\Theta(n)$ time
- Computation of $y[i] = \Theta(n)$ time

} total $\Theta(n)$ time per node

serial은 n^2 ★

serial work에 비해 n배 줄었음!

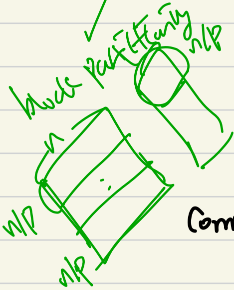
- total parallel work = $\Theta(p^2) = \Theta(n^2)$

↕ serial이랑 같으면 speed up 있음?

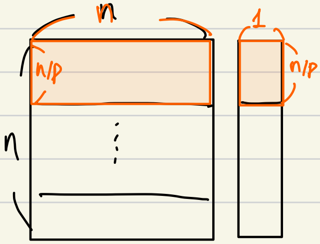
Matrix-Vector Multiplication : 1D Partitioning (Cont.)

Case of $p<n$

block partitioning



Partitioning



- n/p complete rows of matrix A } per node
- n/p elements of vector x

Computation

① 각자 own 하고있는 n/p vector elements all-to-all broadcast

② 각자 n/p개의 length n의 row에 대해서 local dot product

broadcast 필요없고 All-to-All

$(t_s + t_w \cdot m \cdot (n/p))(p-1)$

Parallel Run Time

- 2D mesh protocol

$$T_p = n^2/p + 2t_s(\sqrt{p}-1) + t_w(n/p)(p-1)$$

→ Computation : $n^2/p = (n/p) * n$

→ Communication

- Phase 1 : $(t_s + t_w(n/p))(\sqrt{p}-1)$
- Phase 2 : $(t_s + t_w(n\sqrt{p}/p))(\sqrt{p}-1)$
- total : $2t_s(\sqrt{p}-1) + t_w(n/p)(\sqrt{p}-1+p-\sqrt{p})$

$$= 2t_s(\sqrt{p}-1) + t_w(n/p)(p-1)$$

halving 안함

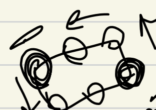
mesh는 halving 안함

$(t_s + t_w \cdot m \cdot (n/\sqrt{p}))$

- ring protocol 고민해보니까...

broadcast ~~recursive~~ doubling. msg 크기 같음

$(t_s + t_w \cdot m \cdot n/\sqrt{p})(p-1)$

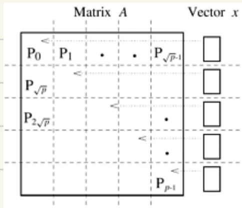


Matrix-Vector Multiplication: 2D Partitioning

개인이 다 갖는 경우

Case of $p=n^2$

Partitioning



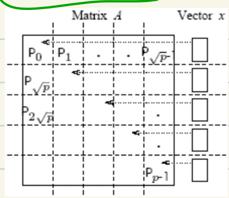
- each node stores
  - 1 element of matrix A
- only last node of each row holds 1 element of x

broadcast 를 어떻게 하지

그래도 overlap 안되?

Computation

① last column node는 vector x element를 diagonal에 locate 시킨다

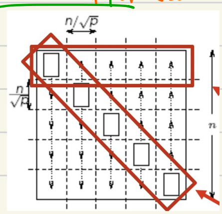


→ one-to-one communication

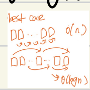
→ cost : constant $O(1)$ $\theta(t_s+t_w m)$

→ recursive doubling 하면 $O(\log n)$

② one-to-all broadcast from diagonal



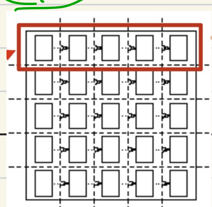
- 각 diagonal에서 rowwise로 broadcast 한다
- cost : $\theta(\log n)$ time



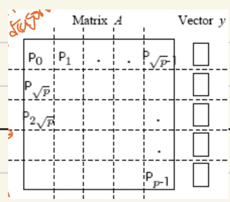
recursive doubling 사용

reduction element 선들수 있음

③ All-to-one reduction in each row to compute y[i]



→



- cost : $\theta(\log n)$ time

cost ✓

Total parallel work $\theta(n^2 \log n)$ … serial은 $\theta(n^2)$ 보다 overhead는 있지만 speedup이 그만큼 있어서! (ideal)

parallel time

eh?

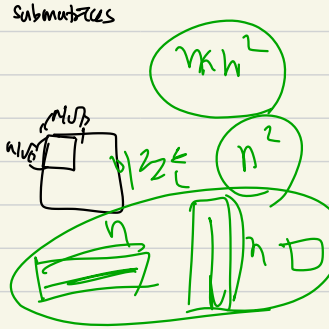
nodes 수

Matrix-Matrix Multiplication

Matrix-Matrix Multiplication

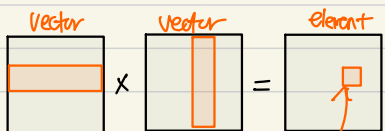
- Compute $C = A \times B$ for A, B, C $n \times n$ dense square matrices
- serial complexity, $O(n^3)$
- block operations are useful

  - q^3 matrix multiplication
  - each using $(n/q) \times (n/q)$ submatrices



Block distribution

- decompose A and B matrices into p blocks
  - each block size = $(n/p) \times (n/p)$
- Process P_{ij}
  - stores A_{ij}, B_{ij}
  - computes C_{ij} of result matrix



Computation

① all-to-all broadcast for row data

- cost = $t_s \cdot \log\sqrt{p} + t_w (n^2/p)(\sqrt{p}-1)$ (hypercube)

② all-to-all broadcast for column data

- cost = $t_s \cdot \log\sqrt{p} + t_w (n^2/p)(\sqrt{p}-1)$ (hypercube)

③ Compute local

- cost = $\sqrt{p} * (n/\sqrt{p})^3 = n^3/p$

$\sqrt{p} * (n/\sqrt{p})^3 = n^3/p$

Total parallel run time

- $T_p = n^3/p + 2t_s \cdot \log\sqrt{p} + 2t_w (n^2/p)(\sqrt{p}-1)$

per thread $\sqrt{p} * (n/\sqrt{p})^3$

$\sqrt{p} * (n/\sqrt{p})^3$

Memory overhead

- each node needs memory space at least $2n^2/\sqrt{p} + n^2/p$

input A, B

output C

output

$n/\sqrt{p} \times n$

Matrix-Matrix Multiplication (Cont.)

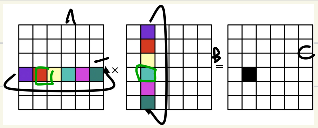
Memory overhead 줄이기 위함?

Cannon's Algorithm

: memory-efficient version of MM algorithm

- Circulate the blocks of A and B with each process multiplying local submatrices at each step

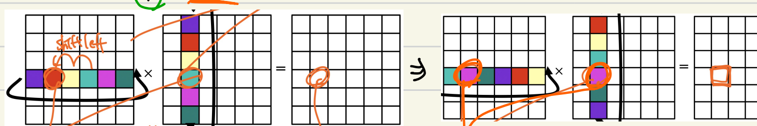
Steps



(initial state)

① perform alignment until same color on target element
- shift A_{ij} left by i
- shift B_{ij} up by j

각각 shift amount 다름



$i=3$ $j=1$

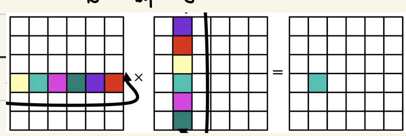
same color aligned
여기서 보면 모두 same color

* Cannon의 경우 local computation 끝은 partial reduction 후의 discard 할 수 없다.
즉, (n^2/p) memory만 필요함
기존방법은 $2n^2/p$
1 for input (temp)
1 for output

- cost : Maximum shift amount $\sqrt{p}-1$
  each shift communication $t_s + t_w n^2/p$ (blocksize → block)

② Perform local block multiplication and add to partial result (reduction)
- shift A_{ij} left by 1
- shift B_{ij} up by 1

} same shift amount … total 6 shift needed



→ Multiplication step $= \sqrt{p}$

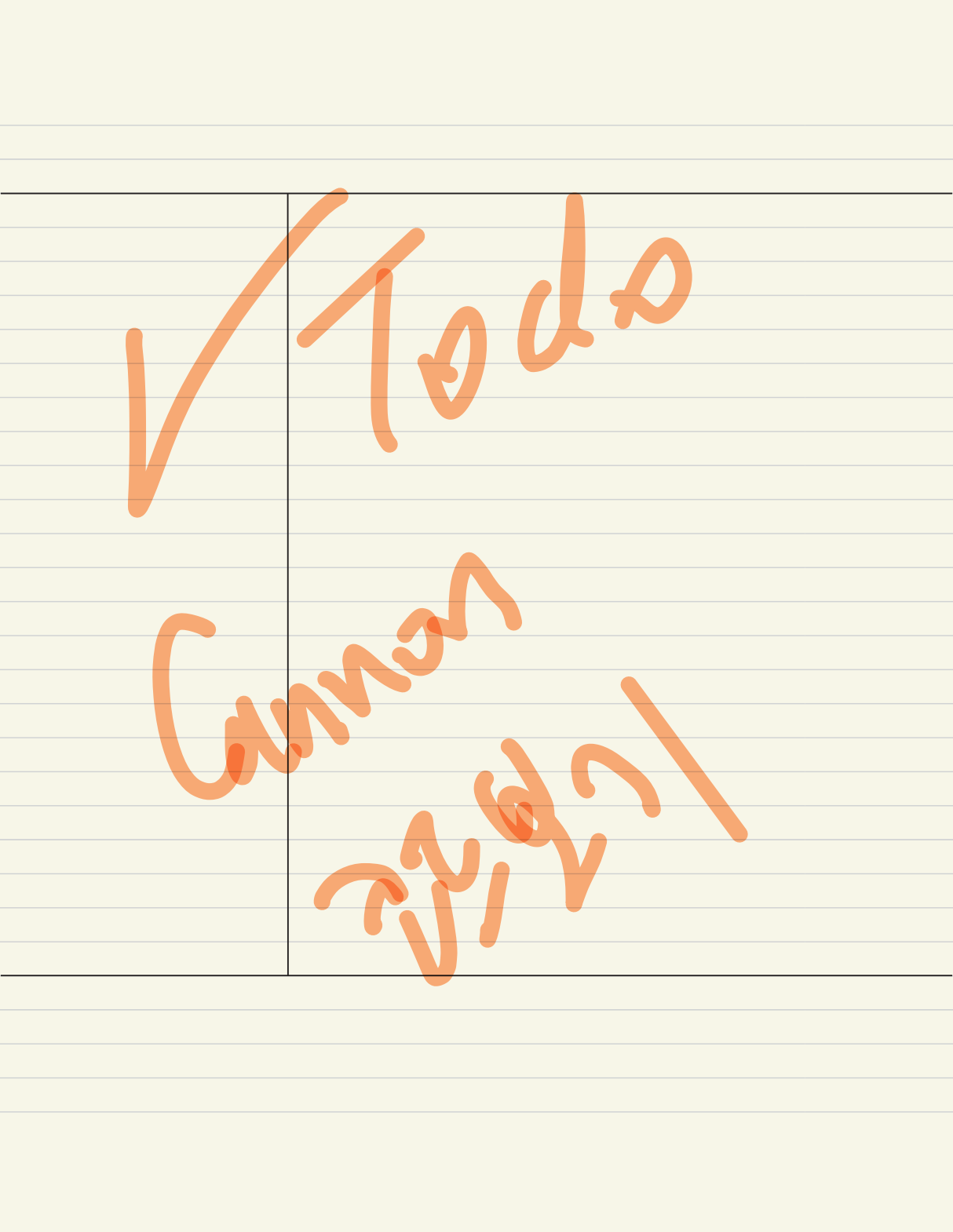
- cost :
  → Communication : $\sqrt{p}$ shifts and each shift takes $t_s + t_w n^2/p$
  → Computation : n^3/p

alignment, $n\#(n/\sqrt{p} \times n/\sqrt{p})$, comm, computation

more works than serial but Memory efficient

Parallel time

$2(\sqrt{p}-1)(t_s + t_w n^2/p) + 2\sqrt{p}(t_s + t_w n^2/p) + n^3/p \approx 3n^2/p$


Todo
Cumar
2021

3D Matrix Multiplication

Algorithm	Comm	Compute	Memory per processor
Original MM	$O(n^2/\sqrt{p})$	$\Theta(n^3/p)$	$\cong 2n^2/p^{1/2} + n^2/p$
Cannon's MM	$O(n^2/\sqrt{p})$	$\Theta(n^3/p)$	$\cong 3n^2/p$
3D MM	$O(n^2/p^{2/3} \log p)$	$\Theta(n^3/p)$	$\cong 3n^2/p^{2/3}$