

Distributed Dense Matrix Multiply

Introduction

1. Supercomputers are ranked using a matrix multiply benchmark
 - Matrix multiply is relatively easy to analyze

Matrix Multiply Basic Definitions

1. $C \leftarrow C + A * B$
 - C is m by n
 - A is m by k
 - B is k by n
 - Compute dot product between rows of A and columns of B and sum

```
for i <- 1 to m do
  for j <- 1 to n do
    for l <- 1 to k do
      C[i,j] <- C[i,j] + A[i,l] * B[l,j]
```

2. Complexity
 - $T(m,n,k) = O(mnk) = O(n^3)$
 - $W(n) = O(n^3)$
 - $D(n) = O(\log(n))$
 - The first two loops can be converted to parfor loops
 - You can block the computation to improve cache coherence

```
for i <- 1 to m do
  for j <- 1 to n do
    let T[1:k] = temp array
    for l <- 1 to k do
      T[l] <- A[i,l] * B[l,j]
    C[i,j] <- C[i,j] + reduce(T[:])
```

Definitions Check

1. Determine the state of matrices A, B, and C

$$\underline{C \leftarrow C + A \cdot B}$$

Your task: Fill in the blanks to reflect the state once the matmul completes.

A	<div></div>	4	1
	4	<div></div>	1

<div></div>	20
27	31

C

3	5
6	2
3	

B

(Assume $C = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ initially.)

Definitions Quiz

2. $A = \begin{bmatrix} 1 & 4 & 1 \end{bmatrix}; \begin{bmatrix} 4 & 2 & 1 \end{bmatrix}$;
3. $B = \begin{bmatrix} 3 & 5 \end{bmatrix}; \begin{bmatrix} 6 & 2 \end{bmatrix}; \begin{bmatrix} 3 & 7 \end{bmatrix}$;
4. $C = \begin{bmatrix} 30 & 20 \end{bmatrix}; \begin{bmatrix} 27 & 31 \end{bmatrix}$;

A Geometrical View

1. You can think of the matrix multiply computation as a cube that is m by n by k
2. For any integer cube of points, suppose I give you a subset of its surfaces S_a, S_b, S_c
 - There may be some volume of intersection in the interior, I
 - $I \leq \sqrt{S_a * S_b * S_c}$
 - Loomis and Whitney, 1949

Applying Loomis Whitney

1. Consider the following:
 - S_a : Some 3×5 piece of A
 - S_b : Some 5×4 piece of B
 - S_c : Some 2×2 piece of C
2. The number of multiplies is between X and Y inclusive
 - Maximum is $\sqrt{((3 * 5) * (5 * 4) * (2 * 2))} = 34$
 - Minimum is 0 if slices don't intersect

1D Algorithms

1. Assume A,B,C are $n \times n$ and $P \mid n$

```

let Ahat[1:n/p][1:n] = local part of A
let Bhat[1:n/p][1:n] = local part of B
let Chat[1:n/p][1:n] = local part of C
let B0[1:n/p][1:n] = temporary storage
let rnext <- (RANK+1) % P
let rprev <- (RANK+P-1) % P

```

```

for L <- 0 to P-1 do
  Chat[:,L] += Ahat[:,L] * Bhat[L,:]
  sendAsync(Bhat -> rnext)
  recvAsync(B0 <- rprev)
  wait(*)
  swap(Bhat, B0)

```

1D Algorithm Cost Part 1

1. τ = time per FLOP (1 multiply or 1 add)
2. $T_{\text{comp}}(n;P) = 2 * \tau * n^3 / P$
3. What is $T_{\text{net}}(n;P)$? Use a for α and b for β
 - Each iteration sends $n/p * n$ words = n^2 / P
 - There are P rounds of communication
 - $T_{\text{net}}(n;P) = a * P + b * n^2$

1D Algorithm Cost Part 2

1. $T1D(n;P) = 2 * \tau * n^3 / P + a * P + B * n^2$
2. How can we rearrange the statements in the body of the loop to get a factor of 2x improvement in the best case?
 - `sendAsync(Bhat -> rnext)`
 - `recvAsync(B0 <- rprev)`
 - `Chat[:,L] += Ahat[:,L] * Bhat[L,:]`
 - `waitAll()`
 - `swap(Bhat, B0)`
 - This overlaps the computation and communication
 - $T1D(n;P) = \max(2 * \tau * n^3 / P, a * P + B * n^2)$
 - $a + b \leq 2 * \max(a, b)$

Efficiency and the 1D Algorithm

1. Speedup: $S1D(n;P) = T(n) / T1D(n;P)$
 - $S1D(n;P) = \max(1, a * P^2 / (2 * \tau * n^3) + B * P / (2 * \tau * n))$
 - $n = \omega(P)$ for this system to be efficient
 - This is called the isoefficiency function
2. Speedup / P = Parallel efficiency
 - A parallel system is efficient if its parallel efficiency is a constant
 - This means the system scales well as P grows
 - Otherwise, we see diminishing returns as we increase the parallelism of the system
3. Temporary storage: $M(n;P) = (3 + 1) * (n/p) * n = 4 * n^2 / P$

Isoefficiency

1. Consider a tree-based all-to-one reduce
 - $T_{\text{tree}}(n;P) = \tau * n * \log(P) + a * \log(P) + B * n * \log(P)$
2. Which of the following best describes the isoefficiency function of a tree-based all-to-one reduce?
 - $\log(P)$
 - $O(P)$
 - $P * \log(P)$
 - P^2
 - none of these (true)
3. Efficiency: $E(n;P) = S(n;P) / P = T(n) / (P * T(n;P))$
 - $E(n;P) = (\tau * n * P) / P * (\tau * n * \log(P) + a * \log(P) + B * n * \log(P))$

- $E(n;P) = 1 / ((1 + P/\tau) * \log(P) + (a/\tau) * (\log(P)/n))$
 - Because the $(1 + P/\tau) * \log(P)$ term scales with P , there is no value of n that causes this to tend to 0

A 2D Algorithm SUMMA

1. Intuitively, a 2D mesh or torus should be a better topology for matrix multiply
2. SUMMA gives each node a block of the output matrix C to update
 - Then, we provide a row of A of height s and a column of B of height s to each node (call this strip index l)
 - Owner of strip can simply broadcast
 - All processes execute the following for loop

```
for l <- 1 to n/s do
  broadcast(row, owner)
  broadcast(col, owner)
  matmul
```

3. SUMMA complexity
 - $T_{\text{summa}}(n;P,s) = n/s * (2 * \tau * n^2 * s / P) + T_{\text{het}}(n;P,s)$
 - $T_{\text{summa}}(n;P,s) = 2 * \tau * n^3/P + T_{\text{het}}(n;P,s)$

SUMMA Communication Time

1. Choose the best options for each term to satisfy the T_{het} complexity
 - $a * n / s$
 - $\log(P)$
 - \sqrt{P}
 - P
 - $B * n^2 / \sqrt{P}$
 - 1
 - $\log(P)$
 - \sqrt{P}
2. The answer depends on how the broadcast is implemented (bucket vs tree)
 - $T_{\text{tree}} = O(a * \log(P) + B * m * \log(P))$
 - $\log(P), \log(P)$
 - $T_{\text{bucket}} = O(a * P + B * m)$
 - $P, 1$

Efficiency of a 2D SUMMA

1. Is the 2D SUMMA scheme intrinsically more scalable than the 1D block-row scheme?
 - Quite possibly
2. Consider the efficiency with a tree-based broadcast
 - $E_{\text{tree}} = 1 / (1 + (aP\log(P))/(2\tau * n^2) + (B\sqrt{P}\log(P))/(2\tau * n))$
 - Isoefficiency function: $n_{\text{tree}}(P) = O(\sqrt{P} * \log(P))$
 - $n_{1D}(P) = O(P)$
 - $n_{\text{bucket}}(P) = O(P^{5/6})$ (this trades lower communication volume for higher latency)

SUMMA Memory

1. How much memory does SUMMA need compared to the 1D scheme?
 - More than 1D
 - Less than 1D
 - Same as 1D
2. The size of additional memory depends on s , so it could be any of them

- $M_{\text{summa}} = 3n^2/P + 2ns/\sqrt{P}$
 - Each node must store one block of A, B, C
 - Must also store a row and column of size ns/\sqrt{P}
- SUMMA is the gold standard for dense matrix multiply due to the simplicity of the algorithm and tuning parameter that allows the tradeoff between time and storage

A Lower Bound on Communication

1. Consider a network of P nodes connected with some topology
 - For one specific node i, how many words must i communicate?
 - W multiplies, M words of memory
 - A node will alternate between periods of communication and computation
 - S_a, S_b, S_c : Set of elements of A seen in this phase
 - $|S_a| \leq 2 * M$
 - Maximum multiplies per phase: $\sqrt{|S_a| * |S_b| * |S_c|} \leq 2 * \sqrt{2} * M^{3/2}$
 - How many phases, L, does the computation take?
 - $L \geq \lceil W / \text{Max multiplies per phase} \rceil = \lceil W / (2 * \sqrt{2} * M^{3/2}) \rceil$
 - Lower bound of transfers is number of phases times M

Words communicated by 1 node $\geq W / (2 * \sqrt{2} * \sqrt{M}) - M$

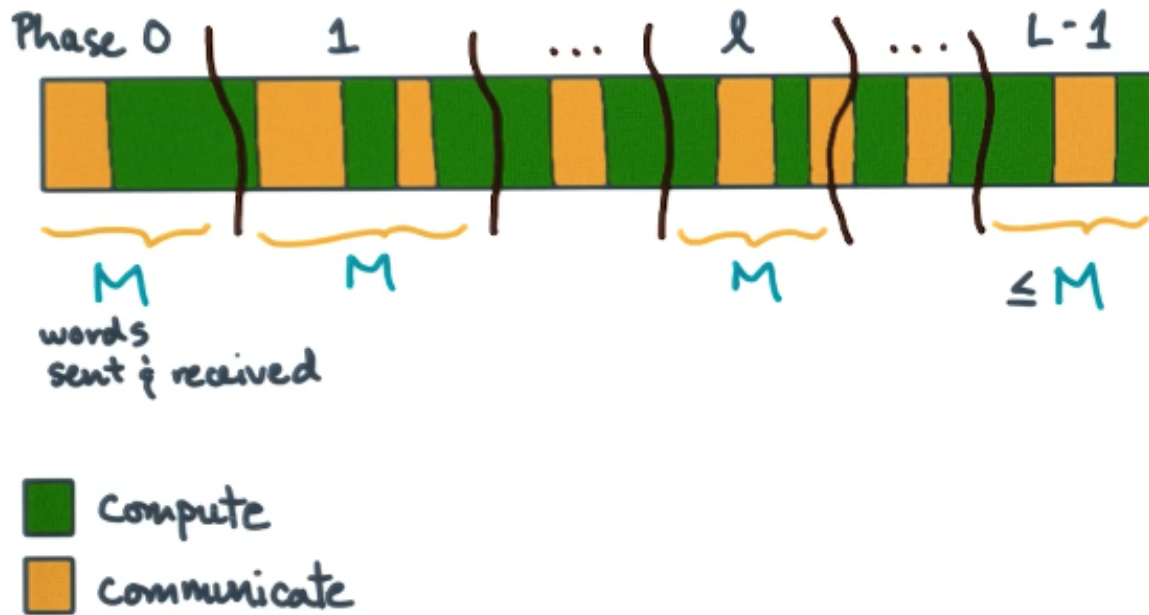
- Number of multiplies $W \geq mnk / P$
- 2. Lower bound on volume of communication by one node

words $\geq n^3 / (2 * \sqrt{2} * P * \sqrt{M}) - M$

- $M = O(n^2/P)$
-

words $\geq n^2 / \sqrt{P}$

- $T_{\text{net}}(n;P) \geq a + B * n^2/\sqrt{P}$
 - What is the factor for the alpha term?



Lower Bound on Communication

A Lower Bound on Communication Quiz

1. What is the lower bound on the number of messages a node must send?
 - n^2/\sqrt{P} is the minimum volume sent by a node
 - $M(n;P) = O(n^2/P)$ is the largest message a node can send
 -

$$\text{messages} = n^2/\sqrt{P} / M(n;P) = O(\sqrt{P})$$

Matching (Or Beating!) The Lower Bounds

1. SUMMA is off by a factor of $\log(P)$ in the alpha term
2. Cannon's algorithm beats SUMMA because it has a communication time that exactly matches the lower bound (1969)
 - Isn't practical to implement
3. The lower bound analysis assumes $M = n^2/P$
 - This assumption relates to distributing surfaces of the cube across nodes
 - If we distribute the volume instead of the surfaces (2D vs 3D), can we duplicate some data and reduce communication
4. 3D algorithm
 - $M_{3D} = M_{2D} * P^{(1/3)}$
 - $T_{3Dnet} = T_{2Dnet} / P^{(1/3)}$
 - Review this before the test (full vs partial replication)

$$T_{1D,net}(n; P) = \alpha \cdot P + \beta \cdot n^2$$

$$T_{summa,net}(n; P, s) = \begin{cases} \alpha \frac{n}{s} \log P + \beta \frac{n^2}{\sqrt{P}} \log P & (\text{tree}) \\ \alpha \frac{n}{s} \sqrt{P} + \beta \frac{n^2}{\sqrt{P}} & (\text{bucket}) \end{cases}$$

$$1 \leq s \leq \frac{n}{\sqrt{P}}$$

$$\geq \alpha \sqrt{P} \log P + \beta \frac{n^2}{\sqrt{P}}$$

$$T_{lower}(n; P) = \Omega\left(\alpha \sqrt{P} + \beta \frac{n^2}{\sqrt{P}}\right) \quad \text{assume: } M = \Theta\left(\frac{n^2}{P}\right)$$

Lower Bounds

Conclusion

1. Matrix multiply lends itself to different analysis techniques (1D vs 2D vs 3D) as well as analyzing the lower bound on communication
 - Other algorithms aren't necessarily as easy to analyze
2. Supercomputers are tuned to do problems that are computation-intensive
 - More communication-intensive algorithms might not scale as well