Multiple Tensor Times Matrix computation

Suraj Kumar

Inria & ENS Lyon

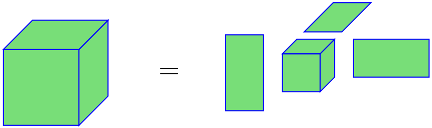
Email:*suraj.kumar@inria.fr*

CR12: October 2023

https://surakuma.github.io/courses/daamtc.html

Tucker decomposition of $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$

It represents a tensor with d matrices (usually orthonormal) and a small core tensor.



Tucker decomposition of a 3-dimensional tensor.

$$\mathcal{X} = \mathcal{Y} \times_1 \mathrm{A}^{(1)} \cdots \times_d \mathrm{A}^{(d)}$$

$$\mathcal{X}(i_1, \cdots, i_d) = \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_d=1}^{r_d} \mathcal{Y}(\alpha_1, \cdots, \alpha_d)\mathrm{A}^{(1)}(i_1, \alpha_1) \cdots \mathrm{A}^{(d)}(i_d, \alpha_d)$$

It can be concisely expressed as $\mathcal{X} = [\![\mathcal{Y}; \mathrm{A}^{(1)}, \cdots, \mathrm{A}^{(d)}]\!]$.

Here r_j for $1 \leq j \leq d$ denote a set of ranks. Matrices $\mathrm{A}^{(j)} \in \mathbb{R}^{n_j \times r_j}$ for $1 \leq j \leq d$ are usually orthonormal and known as factor matrices. The tensor $\mathcal{Y} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_d}$ is called the core tensor.

High Order SVD (HOSVD) for computing a Tucker decomposition

Algorithm 1 HOSVD method to compute a Tucker decomposition

Require: input tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, desired rank $(r_1, \cdots, r_d)$
Ensure: $\mathcal{X} = \mathcal{Y} \times_1 \mathsf{A}^{(1)} \times_2 \mathsf{A}^{(2)} \cdots \times_d \mathsf{A}^{(d)}$
1: **for** $k = 1$ to d **do**
2: $\quad \mathsf{A}^{(k)} \leftarrow r_k$ leading left singular vectors of $X_{(k)}$
3: **end for**
4: $\mathcal{Y} = \mathcal{X} \times_1 {\mathsf{A}^{(1)}}^{\mathsf{T}} \times_2 {\mathsf{A}^{(2)}}^{\mathsf{T}} \cdots \times_d {\mathsf{A}^{(d)}}^{\mathsf{T}}$

- When $r_i < rank(X_{(i)})$ for one or more i, the decomposition is called the truncated-HOSVD (T-HOSVD)
- The collective operation $\mathcal{X} \times_1 {\mathsf{A}^{(1)}}^{\mathsf{T}} \times_2 {\mathsf{A}^{(2)}}^{\mathsf{T}} \cdots \times_d {\mathsf{A}^{(d)}}^{\mathsf{T}}$ is known as Multiple Tensor-Times-Matrix (Multi-TTM) computation

Sequentially T-HOSVD (ST-HOSVD) for Tucker decomposition

- This method is more work efficient than T-HOSVD
- In each step, it reduces the size of one dimension of the tensor

---

Algorithm 2 ST-HOSVD method to compute a Tucker decomposition

---

Require: input tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, desired rank $(r_1, \cdots, r_d)$
Ensure: $[\![\mathcal{Y}; \mathsf{A}^{(1)}, \cdots, \mathsf{A}^{(d)}]\!]$: a $(r_1, \cdots, r_d)$-rank Tucker decomposition of $\mathcal{X}$

1: $\mathcal{W} \leftarrow \mathcal{X}$
2: **for** $k = 1$ to d **do**
3: $\quad \mathsf{A}^{(k)} \leftarrow r_k$ leading singular vectors of $W_{(k)}$
4: $\quad \mathcal{W} \leftarrow \mathcal{W} \times_k {\mathsf{A}^{(k)}}^{\mathsf{T}}$
5: **end for**
6: $\mathcal{Y} = \mathcal{W}$

We can note that ST-HOSVD also performs Multi-TTM computation by doing a sequence of TTM operations, i.e, $\mathcal{Y} = ((\mathcal{X} \times_1 {\mathsf{A}^{(1)}}^{\mathsf{T}}) \times_2 {\mathsf{A}^{(2)}}^{\mathsf{T}}) \cdots \times_d {\mathsf{A}^{(d)}}^{\mathsf{T}}$.

Bottlenecks for algorithms to compute Tucker decompositions

- Multi-TTM becomes the overwhelming bottleneck computation when
  - Matrix SVD costs are reduced using randomization via sketching or
  - U_k are computed with eigen value decompositions of $B_{(k)}B_{(k)}^T$

Multi-TTM computation

Let $\mathcal{Y} \in \mathbb{R}^{r_1 \times \cdots \times r_d}$ be the output tensor, $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ be the input tensor, and $\mathsf{A}^{(k)} \in \mathbb{R}^{n_k \times r_k}$ be the matrix of the kth mode. Then the Multi-TTM computation can be represented as

$$\mathcal{Y} = \mathcal{X} \times_1 {\mathsf{A}^{(1)}}^\mathsf{T} \cdots \times_d {\mathsf{A}^{(d)}}^\mathsf{T}$$

$$\text{or } \mathcal{X} = \mathcal{Y} \times_1 \mathsf{A}^{(1)} \cdots \times_d \mathsf{A}^{(d)}.$$

We will focus only on the first representation in this course. Our results and analysis extend straightforwardly to the latter case.

Two approaches to perform this computation:

- TTM-in-sequence approach – performed by a sequence of TTM operations

$$\mathcal{Y} = ((\mathcal{X} \times_1 {\mathsf{A}^{(1)}}^\mathsf{T}) \times_2 {\mathsf{A}^{(2)}}^\mathsf{T}) \cdots \times_d {\mathsf{A}^{(d)}}^\mathsf{T}$$

- All-at-once approach

$$\mathcal{Y}(r_1', \ldots, r_d') = \sum_{\{n_k' \in [n_k]\}_{k \in [d]}} \mathcal{X}(n_1', \ldots, n_d') \prod_{j \in [d]} \mathsf{A}^{(j)}(n_j', r_j')$$

$[d]$ denotes $\{1, 2, \cdots, d\}$. We represent $n_1 n_2 \cdots n_d$ and $r_1 r_2 \cdots r_d$ by n and r, respectively. We mainly focus on all-at-once approach.

All-at-once Multi-TTM pseudo code

$$
\begin{aligned}
&\text{for } n_1' = 1{:}n_1,\ \ldots,\ \text{for } n_d' = 1{:}n_d,\\
&\quad \text{for } r_1' = 1{:}r_1,\ \ldots,\ \text{for } r_d' = 1{:}r_d,\\
&\qquad \mathcal{Y}(r_1', \ldots, r_d') \mathrel{+}= \mathcal{X}(n_1', \ldots, n_d') \cdot \mathrm{A}^{(1)}(n_1', r_1') \cdot \cdots \cdot \mathrm{A}^{(N)}(n_d', r_d')
\end{aligned}
$$

Δ matrix for Multi-TTM

$$
\Delta = \begin{array}{cc}
 & \begin{array}{ccccccc} \mathrm{A}^{(1)} & \cdots & \mathrm{A}^{(i)} & \cdots & \mathrm{A}^{(d)} & \mathcal{X} & \mathcal{Y} \end{array} \\
\begin{array}{c} n_1' \\ \vdots \\ n_i' \\ \vdots \\ n_d' \\ r_1' \\ \vdots \\ r_i' \\ \vdots \\ r_d' \end{array} &
\begin{pmatrix}
1 & & & & & 1 & \\
 & \ddots & & & & \vdots & \\
 & & 1 & & & 1 & \\
 & & & \ddots & & \vdots & \\
 & & & & 1 & 1 & \\
1 & & & & & & 1 \\
 & \ddots & & & & & \vdots \\
 & & 1 & & & & 1 \\
 & & & \ddots & & & \vdots \\
 & & & & 1 & & 1
\end{pmatrix}
\end{array}
= \begin{pmatrix} \mathrm{I}_{d\times d} & 1_d & 0_d \\ \mathrm{I}_{d\times d} & 0_d & 1_d \end{pmatrix}
$$

Final assignment – deadline Oct 26

Question: Let $\mathcal{Y} \in \mathbb{R}^{r\times r\times r}$, $\mathcal{X} \in \mathbb{R}^{n\times n\times n}$ and $A \in \mathbb{R}^{n\times r}$. What are the different approaches to perform the following Multi-TTM computation?

$$\mathcal{Y} = \mathcal{X} \times_1 A^\mathsf{T} \times_2 A^\mathsf{T} \times_3 A^\mathsf{T}$$

Compute the exact number of arithmetic operations for each approach.

1. Parallel Multi-TTM computation

Settings to compute parallel communication lower bound

- Without loss of generality, we assume that $n_1 r_1 \leq n_2 r_2 \leq \cdots \leq n_d r_d$
- The input tensor is larger than the output tensor, i.e., $n \geq r$
- The algorithm load balances the computation – each processor performs $1/P$th number of loop iterations
- One copy of data is in the system
    - There exists a processor whose input data at the start plus output data at the end must be at most $\frac{n+r+\sum_{i=1}^{d} n_i r_i}{P}$ words – will analyze amount of data transfers for this processor
- Assume that the innermost computation is atomic – all the multiplications are performed on only one processor

Table of Contents

Optimization problems (Ballard et. al., 2023)

Lemma

Consider the following optimization problem:

$$\min_{x,y,z} x + y + z \textit{ such that}$$

$$\frac{nr}{P} \leq xyz, \quad 0 \leq \ x \ \leq n_1 r_1, \quad 0 \leq \ y \ \leq n_2 r_2, \quad 0 \leq \ z \ \leq n_3 r_3,$$

where $n_1 r_1 \leq n_2 r_2 \leq n_3 r_3$, *and* $n_1, n_2, n_3, r_1, r_2, r_3, P \geq 1$. *The optimal solution* (x^*, y^*, z^*) *depends on the relative values of the constraints, yielding three cases:*

1. *if* $P < \frac{n_3 r_3}{n_2 r_2}$, *then* $x^* = n_1 r_1$, $y^* = n_2 r_2$, $z^* = \frac{n_3 r_3}{P}$;
2. *if* $\frac{n_3 r_3}{n_2 r_2} \leq P < \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}$, *then* $x^* = n_1 r_1$, $y^* = z^* = \left(\frac{n_2 n_3 r_2 r_3}{P}\right)^{\frac{1}{2}}$;
3. *if* $\frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \leq P$, *then* $x^* = y^* = z^* = \left(\frac{nr}{P}\right)^{\frac{1}{3}}$;

which can be visualized as follows.

P

1 — $\frac{n_3 r_3}{n_2 r_2}$ — $\frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}$

$x^* = n_1 r_1$
$y^* = n_2 r_2$
$z^* = \frac{n_3 r_3}{P}$

$x^* = n_1 r_1$
$y^* = z^* = \left(\frac{n_2 n_3 r_2 r_3}{P}\right)^{1/2}$

$x^* = y^* = z^* = \left(\frac{nr}{P}\right)^{1/3}$

Optimization problems (Ballard et. al., 2023)

Lemma

Consider the following optimization problem:

$$\min_{u,v} u + v \text{ such that}$$

$$\frac{nr}{P} \le uv, \quad 0 \le u \le r, \quad 0 \le v \le n,$$

where $n \ge r$, *and* $n, r, P \ge 1$. *The optimal solution* (u^*, v^*) *depends on the relative values of the constraints, yielding two cases:*

1. *if* $P < \frac{n}{r}$, *then* $u^* = r$, $v^* = \frac{n}{P}$;
2. *if* $\frac{n}{r} \le P$, *then* $u^* = v^* = \left(\frac{nr}{P}\right)^{\frac{1}{2}}$;

which can be visualized as follows.

1 — $\frac{n}{r}$ — P

$u^* = r$, $v^* = \frac{n}{P}$ (for $1 \le P < \frac{n}{r}$)

$u^* = v^* = \left(\frac{nr}{P}\right)^{1/2}$ (for $P \ge \frac{n}{r}$)

Both lemma can be proved using the KKT conditions.

Communication lower bound

Theorem

Any computationally load balanced atomic Multi-TTM algorithm that starts and ends with one copy of the data distributed across processors involving 3-dimensional tensors with dimensions n_1, n_2, n_3 *and* r_1, r_2, r_3 *performs at least* $A + B - \left(\frac{n}{P} + \frac{r}{P} + \sum_{j=1}^{3} \frac{n_j r_j}{P}\right)$ *sends or receives where*

$$A = \begin{cases} n_1 r_1 + n_2 r_2 + \frac{n_3 r_3}{P} & \text{if } P < \frac{n_3 r_3}{n_2 r_2} \\ n_1 r_1 + 2\left(\frac{n_2 n_3 r_2 r_3}{P}\right)^{\frac{1}{2}} & \text{if } \frac{n_3 r_3}{n_2 r_2} \leq P < \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \\ 3\left(\frac{nr}{P}\right)^{\frac{1}{3}} & \text{if } \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \leq P \end{cases}$$

$$B = \begin{cases} r + \frac{n}{P} & \text{if } P < \frac{n}{r} \\ 2\left(\frac{nr}{P}\right)^{\frac{1}{2}} & \text{if } \frac{n}{r} \leq P. \end{cases}$$

Communication lower bound proof

Let F be the set of loop indices performed by a processor and $|F| = nr/P$. Define $\phi_{\mathfrak{X}}(F)$, $\phi_{\mathfrak{Y}}(F)$ and $\phi_j(F)$ to be the projections of F onto the indices of the arrays $\mathfrak{X}, \mathfrak{Y}$, and $\mathrm{A}^{(j)}$ for $1 \leq j \leq 3$. Δ matrix can be represented as,

$$\Delta = \begin{pmatrix} \mathrm{I}_{3\times 3} & 1_3 & 0_3 \\ \mathrm{I}_{3\times 3} & 0_3 & 1_3 \end{pmatrix}.$$

Let $\mathcal{C} = \left\{\mathrm{s} \in [0,1]^5 : \Delta \cdot \mathrm{s} \geq 1\right\}$. Here Δ is not full rank, we consider all vectors $\mathrm{v} = [a\ a\ a\ 1\text{-}a\ 1\text{-}a]^{\mathsf{T}} \in \mathcal{C}$ where $0 \leq a \leq 1$ such that $\Delta \cdot \mathrm{v} = 1$. From HBL inequality, we obtain

$$\frac{nr}{P} \leq \Big(\prod_{j\in[3]} |\phi_j(F)|\Big)^{a} \big(|\phi_{\mathfrak{X}}(F)||\phi_{\mathfrak{Y}}(F)|\big)^{1\text{-}a}.$$

This is equivalent to $\frac{nr}{P} \leq \prod_{j\in[3]} |\phi_j(F)|$ and $\frac{nr}{P} \leq |\phi_{\mathfrak{X}}(F)||\phi_{\mathfrak{Y}}(F)|$. We also have $|\phi_{\mathfrak{X}}(F)| \leq n$, $|\phi_{\mathfrak{Y}}(F)| \leq r$, and $|\phi_j(F)| \leq n_j r_j$ for $1 \leq j \leq 3$. We want to minimize $|\phi_{\mathfrak{X}}(F)| + |\phi_{\mathfrak{Y}}(F)| + \sum_{j\in[3]} |\phi_j(F)|$. Employing the previous two lemmas and subtracting the owned data of the processor yields the mentioned bound.

Multi-TTM with cubical tensors

Corollary

Any computationally load balanced atomic Multi-TTM algorithm that starts and ends with one copy of the data distributed across processors involving 3-dimensional cubical tensors with dimensions $n^{\frac{1}{3}} \times n^{\frac{1}{3}} \times n^{\frac{1}{3}}$ *and* $r^{\frac{1}{3}} \times r^{\frac{1}{3}} \times r^{\frac{1}{3}}$ *(with* $n \geq r$*) performs at least*

$$3\left(\frac{nr}{P}\right)^{\frac{1}{3}} + r - \frac{3(nr)^{\frac{1}{3}} + r}{P}$$

sends or receives when $P < \frac{n}{r}$ *and at least*

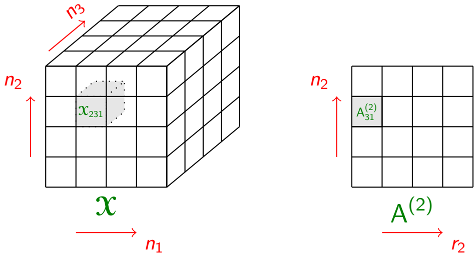
$$3\left(\frac{nr}{P}\right)^{\frac{1}{3}} + 2\left(\frac{nr}{P}\right)^{\frac{1}{2}} - \frac{n + 3(nr)^{\frac{1}{3}} + r}{P}$$

sends or receives when $P \geq \frac{n}{r}$.

We will manily focus on $P < \frac{n}{r}$ case throughout the slides.

Data distribution model

P processors are organized in a 6-dimensional $p_1 \times p_2 \times p_3 \times q_1 \times q_2 \times q_3$ logical processor grid.



Subtensor $\mathcal{X}_{231}$ is distributed evenly among processors $(2, 3, 1, *, *, *)$. Similarly, submatrix $A^{(2)}_{31}$ is distributed evenly among processors $(*, 3, *, *, 1, *)$.

Parallel Multi-TTM algorithm

Algorithm 3 Parallel Atomic 3-dimensional Multi-TTM

Require: $\mathcal{X}$, $\mathsf{A}^{(1)}$, $\mathsf{A}^{(2)}$, $\mathsf{A}^{(3)}$, $p_1 \times p_2 \times p_3 \times q_1 \times q_2 \times q_3$ logical processor grid

Ensure: $\mathcal{Y}$ such that $\mathcal{Y} = \mathcal{X} \times_1 {\mathsf{A}^{(1)}}^{\mathsf{T}} \times_2 {\mathsf{A}^{(2)}}^{\mathsf{T}} \times_3 {\mathsf{A}^{(3)}}^{\mathsf{T}}$

$(p'_1, p'_2, p'_3, q'_1, q'_2, q'_3)$ is my processor id

//All-gather input tensor $\mathcal{X}$

$\mathcal{X}_{p'_1 p'_2 p'_3} =$ All-Gather($\mathcal{X}$, $(p'_1, p'_2, p'_3, *, *, *)$)

//All-gather input matrices

$\mathsf{A}^{(1)}_{p'_1 q'_1} =$ All-Gather($\mathsf{A}^{(1)}$, $(p'_1, *, *, q'_1, *, *)$)

$\mathsf{A}^{(2)}_{p'_2 q'_2} =$ All-Gather($\mathsf{A}^{(2)}$, $(*, p'_2, *, *, q'_2, *)$)

$\mathsf{A}^{(3)}_{p'_3 q'_3} =$ All-Gather($\mathsf{A}^{(3)}$, $(*, *, p'_3, *, *, q'_3)$)

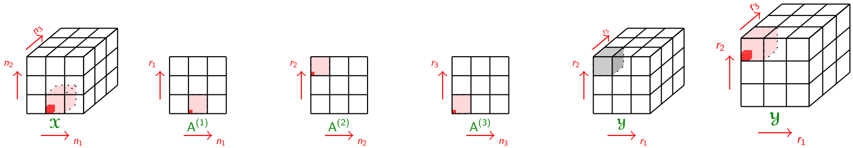
//Local computations in a temporary tensor $\mathcal{T}$

$\mathcal{T} =$ Local-Multi-TTM($\mathcal{X}_{p'_1 p'_2 p'_3}$, $\mathsf{A}^{(1)}_{p'_1 q'_1}$, $\mathsf{A}^{(2)}_{p'_2 q'_2}$, $\mathsf{A}^{(3)}_{p'_3 q'_3}$)

//Reduce-scatter the output tensor in $\mathcal{Y}_{q'_1 q'_2 q'_3}$

Reduce-Scatter($\mathcal{Y}_{q'_1 q'_2 q'_3}$, $\mathcal{T}$, $(*, *, *, q'_1, q'_2, q'_3)$)

Steps of the algorithm



(a) Perform All-Gather on processors $(2,1,1,*,*,*)$ to obtain $\mathcal{X}_{211}$.

(b) Perform All-Gather on processors $(2,*,*,1,*,*)$ to obtain $A^{(1)}_{21}$.

(c) Perform All-Gather on processors $(*,1,*,*,3,*)$ to obtain $A^{(2)}_{13}$.

(d) Perform All-Gather on processors $(*,*,1,*,*,1)$ to obtain $A^{(3)}_{11}$.

(e) Perform local Multi-TTM to compute partial $\mathcal{Y}_{131}$.

(f) Perform Reduce-Scatter on processors $(*,*,*,1,3,1)$ to compute/distribute $\mathcal{Y}_{131}$.

Steps of the algorithm for processor $(2,1,1,1,3,1)$, where $p_1 = p_2 = p_3 = q_1 = q_2 = q_3 = 3$. Highlighted areas correspond to the data blocks on which the processor is operating. The dark red highlighting represents the input/output data initially/finally owned by the processor, and the light red highlighting corresponds to received/sent data from/to other processors in All-Gather/Reduce-Scatter collectives to compute $\mathcal{Y}_{131}$.

Cost analysis

The bandwidth cost of the algorithm is

$$\frac{n}{p} + \frac{n_1 r_1}{p_1 q_1} + \frac{n_2 r_2}{p_2 q_2} + \frac{n_3 r_3}{p_3 q_3} + \frac{r}{q} - \left(\frac{n + n_1 r_1 + n_2 r_2 + n_3 r_3 + r}{P}\right).$$

Here $p = p_1 p_2 p_3$ and $q = q_1 q_2 q_3$. The algorithm is communication optimal when we select p_i and q_i based on lower bounds.

Arithmetic operations

The dimensions of $\mathcal{X}_{p_1' p_2' p_3'}$ and $\mathcal{T}$ are $\frac{n_1}{p_1} \times \frac{n_2}{p_2} \times \frac{n_3}{p_3}$ and $\frac{r_1}{q_1} \times \frac{r_2}{q_2} \times \frac{r_3}{q_3}$, respectively. The dimension of $\mathrm{A}^{(k)}_{p_k' q_k'}$ is $\frac{n_i}{p_i} \times \frac{r_i}{q_i}$ for $i = 1, 2, 3$.

- Local Multi-TTM can be performed as a sequence of TTM operations
- Assuming TTM operations are performed in their order, first with $\mathrm{A}^{(1)}$, then with $\mathrm{A}^{(2)}$, and in the end with $\mathrm{A}^{(3)}$,

$$\text{Total arithmetic operations } = 2\left(\frac{n_1 n_2 n_3 r_1}{p_1 p_2 p_3 q_1} + \frac{n_2 n_3 r_1 r_2}{p_2 p_3 q_1 q_2} + \frac{n_3 r_1 r_2 r_3}{p_3 q_1 q_2 q_3}\right).$$

Multi-TTM cost in TuckerMPI library

- State-of-the-art library for parallel Tucker decomposition
- Implements ST-HOSVD algorithm – employs TTM-in-sequence approach to perform Multi-TTM
- Assume TTMs are performed in increasing mode order

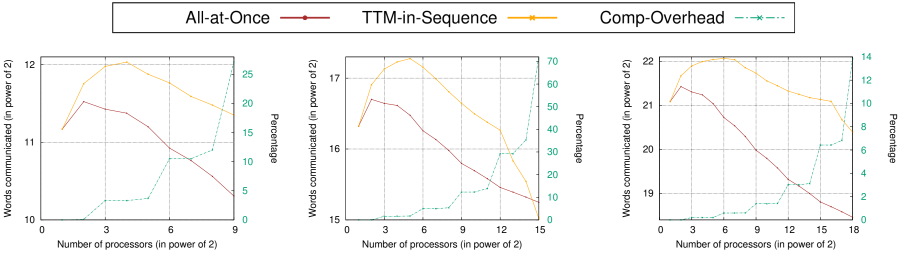
It uses a $\tilde{p}_1 \times \tilde{p}_2 \times \tilde{p}_3$ logical processor grid. The bandwidth cost is

$$\frac{r_1 n_2 n_3}{\tilde{p}_2 \tilde{p}_3} + \frac{n_1 r_1}{\tilde{p}_1} + \frac{r_1 r_2 n_3}{\tilde{p}_1 \tilde{p}_3} + \frac{n_2 r_2}{\tilde{p}_2} + \frac{r_1 r_2 r_3}{\tilde{p}_1 \tilde{p}_2} + \frac{n_3 r_3}{\tilde{p}_3} - \frac{r_1 n_2 n_3 + r_1 r_2 n_3 + r_1 r_2 r_3 + n_1 r_1 + n_2 r_2 + n_3 r_3}{P}.$$

The parallel computational cost is

$$2\left(\frac{r_1 n_1 n_2 n_3 + r_1 r_2 n_2 n_3 + r_1 r_2 r_3 n_3}{P}\right).$$

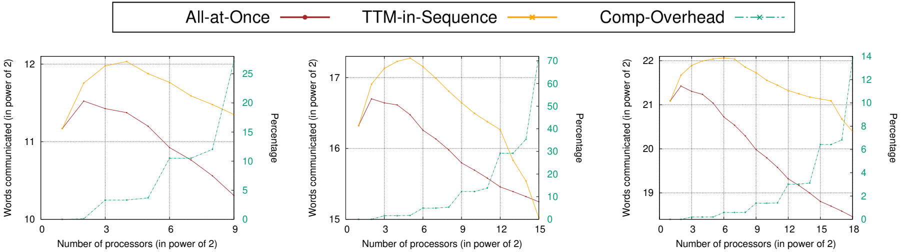
Comparison of All-at-once and TTM-in-sequence



(a) $n_i = 2^8,\ r_i = 2^3$. (b) $n_i = 2^{11},\ r_i = 2^5$. (c) $n_i = 2^{15},\ r_i = 2^6$.

Communication cost comparison of all-at-once approach (the presented algorithm) and TTM-in-sequence approach (of TuckerMPI). *Comp-Overhead* shows the percentage of computational overhead of the all-at-once approach with respect to the TTM-in-sequence approach. Cost of an approach represents the minimum cost among all possible processor configurations.

Comparison of All-at-once and TTM-in-sequence



(a) $n_i = 2^8,\ r_i = 2^3.$ (b) $n_i = 2^{11},\ r_i = 2^5.$ (c) $n_i = 2^{15},\ r_i = 2^6.$

- Not any clear winner for all settings
- All-at-once approach performs significantly less communication for small P
- Computational overhead of all-at-once approach is negligible for small P
- TTM-in-sequence approach is better for large P

Table of Contents

Communication lower bound

Theorem

Any computationally load balanced atomic Multi-TTM algorithm that starts and ends with one copy of the data distributed across processors and involves d-dimensional tensors with dimensions $n_1, n_2, \ldots, n_d$ and $r_1, r_2, \ldots, r_d$ performs at least $A + B - \left(\frac{n}{P} + \frac{r}{P} + \sum_{j=1}^{d} \frac{n_j r_j}{P}\right)$ sends or receives where

$$A = \begin{cases} \sum_{j=1}^{d\text{-}1} n_j r_j + \frac{N_1 R_1}{P} & \text{if } P < \frac{N_1 R_1}{n_{d\text{-}1} r_{d\text{-}1}}, \\ \sum_{j=1}^{(d\text{-}i)} n_j r_j + i\left(\frac{N_i R_i}{P}\right)^{\frac{1}{i}} & \text{if } \frac{N_{i\text{-}1} R_{i\text{-}1}}{(n_{d+1\text{-}i} r_{d+1\text{-}i})^{i\text{-}1}} \le P < \frac{N_i R_i}{(n_{d\text{-}i} r_{d\text{-}i})^{i}}, \\ & \quad \text{for some } 2 \le i \le d-1, \\ d\left(\frac{N_d R_d}{P}\right)^{\frac{1}{d}} & \text{if } \frac{N_{d\text{-}1} R_{d\text{-}1}}{(n_1 r_1)^{d\text{-}1}} \le P. \end{cases}$$

$$B = \begin{cases} r + \frac{n}{P} & \text{if } P < \frac{n}{r}, \\ 2\left(\frac{nr}{P}\right)^{\frac{1}{2}} & \text{if } \frac{n}{r} \le P. \end{cases}$$

Parallel Multi-TTM algorithm

Algorithm 4 Parallel Atomic d-dimensional Multi-TTM

Require: $\mathcal{X}$, $\mathrm{A}^{(1)}, \cdots, \mathrm{A}^{(d)}$, $p_1 \times \cdots \times p_d \times q_1 \times \cdots \times q_d$ logical processor grid

Ensure: $\mathcal{Y}$ such that $\mathcal{Y} = \mathcal{X} \times_1 {\mathrm{A}^{(1)}}^{\mathsf{T}} \cdots \times_d {\mathrm{A}^{(d)}}^{\mathsf{T}}$

$(p'_1, \cdots, p'_d, q'_1, \cdots, q'_d)$ is my processor id

//All-gather input tensor $\mathcal{X}$

$\mathcal{X}_{p'_1 \cdots p'_d} = \text{All-Gather}(\mathcal{X}, (p'_1, \cdots, p'_d, *, \cdots, *))$

//All-gather all input matrices

for $i = 1, \cdots, d$ **do**

    $\mathrm{A}^{(i)}_{p'_i q'_i} = \text{All-Gather}(\mathrm{A}^{(i)}, (*, \cdots, *, p'_i, * \cdots, *, q'_i, *))$

end for

//Perform local computations in a temporary tensor $\mathcal{T}$

$\mathcal{T} = \text{Local-Multi-TTM}(\mathcal{X}_{p'_1 \cdots p'_d}, \mathrm{A}^{(1)}_{p'_1 q'_1}, \cdots, \mathrm{A}^{(d)}_{p'_d q'_d})$

//Reduce-scatter the output tensor in $\mathcal{Y}_{q'_1 \cdots q'_d}$

$\text{Reduce-Scatter}(\mathcal{Y}_{q'_1 \cdots q'_d}, \mathcal{T}, (*, \cdots, *, q'_1, \cdots, q'_d))$

The algorithm is communication optimal when p_i and q_i are selected based on the lower bound.

Perspectives

- Cost analysis of several ways to perform Multi-TTM
  - Unifying all-at-once and sequence approaches
  - Study of communication-computation trade-off

- Optimal costs for algorithms to compute Tucker decompositions
- Design and implementation of parallel optimal algorithms