

國立臺灣大學理學院應用數學科學研究所

碩士論文

Institute of Applied Mathematical Sciences

College of Science

National Taiwan University

Master Thesis



利用高度平行之演算法整合多重隨機奇異值分解並應  
用於巨量資料分析

Highly Scalable Parallelism of Integrated Randomized  
Singular Value Decomposition with Big Data Applications

楊慕

Mu Yang

指導教授：王偉仲 博士

Advisor: Weichung Wang, Ph. D.

中華民國 106 年 7 月

July, 2017



國立臺灣大學碩士學位論文  
口試委員會審定書



利用高度平行之演算法整合多重隨機奇異值分解  
並應用於巨量資料分析

Highly Scalable Parallelism of Integrated  
Randomized Singular Value Decomposition with Big  
Data Applications

本論文係楊慕君 (R04246003) 在國立臺灣大學應用數學科學  
研究所完成之碩士學位論文，於民國 106 年 7 月 26 日承下列考試  
委員審查通過及口試及格，特此證明

口試委員：

王信仲

陳定立

陳孝正

所長：





## 誌謝

在此論文完成之際，必須要感謝在我研究路程上幫助我的所有人，沒有你們的幫助，我無法有現在的成果。

在研究過程中，給予我最多幫助的是我的指導教授 王偉仲博士，從認識他以來，他不遺餘力地指導我的學習，帶給我許多收穫。他的辦公室大門始終對我敞開，每當我的研究或寫作有新的進展或是遇到瓶頸時，總是不厭其煩的與我討論並給予全面性的回應。在王老師的帶領下，我獲得許多機會參與各個學術會議及參訪，得以與各領域的專家交流，對我的研究有相當程度的幫助。此外，也非常感謝中央研究院統計科學研究所的陳素雲博士與陳定立博士，他們的研究在這個領域上奠下基石，我能夠做出此研究並寫出論文，可說是站在巨人的肩膀上。

我衷心感謝我的同學張大衛，他熱情的參與和投入，極大地協助了我的研究工作，與他的討論啟發了我很多想法。在寫作上他也給了我很多意見，大大改善了論文的品質。此外，還要感謝東京大學情報基盤中心提供 Reedbush-U/H 超級電腦系統，以及周世恩學長提供臉書資料庫，幫助我完成這份論文的數值測試。也感謝我研究室的同伴們，不僅在研究上給予我許多幫助，也提供我最歡樂的研究環境，並在我遇到瓶頸給予我精神上的支持。

最後，感謝我的家人這些年來給予我的支持與關懷，讓我得以求學生涯中無後顧之憂，專心於學習與研究。謹以本文獻給我敬愛的家人與所有關心我的人，並將這份成果呈獻給你們。





# Acknowledgements

I'm glad to express my sincere gratitude to my thesis advisor Dr. Weichung Wang of the Institute of Applied Mathematical Sciences at National Taiwan University for his support of my study and research. The door to Prof. Wang's office was always open whenever I ran into a trouble spot or had a question about my research or writing.

I also like to show my gratitude to Dr. Su-Yun Huang and Dr. Ting-Li Chen at the Institute of Statistical Science of Academia Sinica. Without their insight and expertise, the research could not have been successfully conducted.

My sincere thanks also go to my colleague Dawei Chang for his passionate participation and input that greatly assisted the research. The discussion with him inspires me a lot. He also gave me many comments that greatly improved the manuscript.

I would also like to thank the Information Technology Center at the University of Tokyo for providing Reedbush-U/H supercomputer systems, and Mr. Shih-En Chou for providing the Facebook datasets. I am glad to thank my fellow labmates for the stimulating discussions, and for all the fun we have had.

Last but not the least, I would like to thank my family for supporting me spiritually throughout my life.







## 摘要

低秩近似在大數據分析中佔了重要的地位，整合奇異值分解（Integrated Singular Value Decomposition, iSVD）是一種用於計算大矩陣的低秩近似奇異值分解的演算法。iSVD 集成了從多個隨機子空間抽樣而獲得的不同的低秩奇異值分解，並達到更高的精準度和更好的穩定性。雖然多個隨機抽樣與合併的過程需要更高的計算成本，但由於這些操作可以平行化，iSVD 仍然可以節省計算時間。我們在多核心計算集群上平行此演算法，並對計算方法及資料結構進行了修改，以增加可擴展性並減少資料傳輸。透過平行化，iSVD 可以找到巨大矩陣的近似奇異值分解，達到相對於矩陣尺寸和機器數量接近線性的可擴展性，並透過使用 GPU 在抽樣的步驟達到四倍的加速。我們用 C++ 實作此演算法，並應用了幾種提高可維護性、可擴展性和可用性的技術。我們在使用混合 CPU-GPU 的超級電腦系統上使用 iSVD 求解一些大規模的應用問題。

**關鍵詞：**奇異值分解，平行演算法，分散式演算法，隨機演算法，圖形處理器，大數據分析





# Abstract

Low-rank approximation plays an important role in big data analysis. Integrated Singular Value Decomposition (iSVD) is an algorithm for computing low-rank approximate singular value decomposition of large size matrices. The iSVD integrates different low-rank SVDs obtained by multiple random subspace sketches and achieve higher accuracy and better stability. While iSVD takes higher computational costs due to multiple random sketches and the integration process, these operations can be parallelized to save computational time. We parallelize iSVD for multicore clusters, and modify the algorithms and data structures to increase the scalability and reduce communication. With parallelization, iSVD can find the approximate SVD of matrices with huge size, and achieve near-linear scalability with respect to the matrix size and the number of machines, and gained further 4X faster timing performance on sketching by using GPU. We implement the algorithms in C++, with several techniques for high maintainability, extensibility, and usability. The iSVD is applied on some huge size application using hybrid CPU-GPU supercomputer systems.

**Keywords.** Singular value decomposition, Parallel algorithms, Distributed algorithms, Randomized algorithms, Graphics processing units, Big data analysis

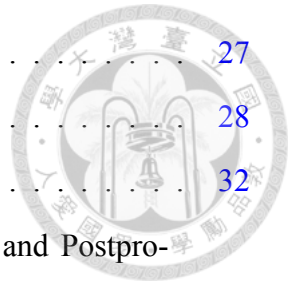




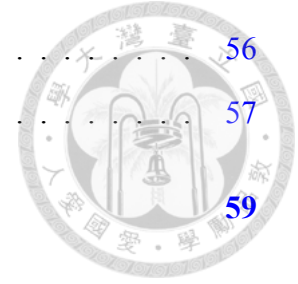
# Contents

口試委員會審定書	iii
誌謝	v
<b>Acknowledgements</b>	<b>vii</b>
摘要	ix
<b>Abstract</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Algorithms</b>	<b>5</b>
2.1 Integrated Singular Value Decomposition . . . . .	5
2.2 Stage I: Sketching . . . . .	6
2.3 Stage II: Orthogonalization . . . . .	7
2.4 Stage III: Integration . . . . .	8
2.5 Stage IV: Postprocessing . . . . .	11
<b>3 Improvements of Integration</b>	<b>15</b>
3.1 Optimization Problem . . . . .	15
3.2 Improvements of Kolmogorov-Nagumo Integration . . . . .	17
3.3 Improvements of Wen-Yin Integration . . . . .	19
3.4 Integration Using the Gramian of $\mathcal{Q}$ . . . . .	22

<b>4</b>	<b>Parallelism</b>	<b>27</b>
4.1	Naïve Parallelism . . . . .	27
4.2	Block-Row Parallelism for Integration . . . . .	28
4.3	Block-Row Parallelism for Gramian Integration . . . . .	32
4.4	Block-Row Parallelism for Sketching, Orthogonalization, and Postprocessing . . . . .	34
4.5	Block-Column Parallelism . . . . .	38
4.6	GPU Acceleration . . . . .	41
<b>5</b>	<b>Comparison of Algorithms</b>	<b>43</b>
<b>6</b>	<b>Implementation</b>	<b>45</b>
6.1	C++ Implementation . . . . .	45
6.1.1	Object Oriented Programming . . . . .	45
6.1.2	C++11 Standard . . . . .	46
6.1.3	Template Meta-Programming . . . . .	46
6.1.4	Curiously Recurring Template Pattern . . . . .	46
6.1.5	Data Storage and Linear Algebra Wrappers . . . . .	47
6.2	Parallelism . . . . .	47
6.2.1	MPI . . . . .	47
6.2.2	OpenMP . . . . .	47
6.2.3	GPU . . . . .	47
6.3	Tools . . . . .	48
6.3.1	CMake . . . . .	48
6.3.2	Google Test . . . . .	48
6.4	Package Structure . . . . .	49
<b>7</b>	<b>Numerical Results</b>	<b>51</b>
7.1	Comparison of Parallelisms . . . . .	51
7.2	Scalability of Integration Algorithms . . . . .	53
7.3	Implementation and Big Data Applications . . . . .	55



7.3.1	Comparison with MATLAB . . . . .	56
7.3.2	Facebook 100k . . . . .	56
7.3.3	1000 Genomes Project . . . . .	57



**8 Conclusion**

<b>Bibliography</b>	<b>61</b>
---------------------	-----------

**A Derivations** **65**

A.1	Derivation of Hierarchical Reduction Integration . . . . .	65
A.2	Derivation of Tall Skinny QR . . . . .	66

**B Complexity** **69**

B.1	Canonical Methods . . . . .	70
B.1.1	Orthogonalization . . . . .	71
B.2	Sequential Algorithms . . . . .	72
B.2.1	Stage I: Sketching . . . . .	72
B.2.2	Stage II: Orthogonalization . . . . .	72
B.2.3	Stage III: Integration . . . . .	73
B.2.4	Stage IV: Postprocessing . . . . .	76
B.3	Parallel Algorithms . . . . .	77
B.3.1	Stage I: Sketching (Parallelism) . . . . .	77
B.3.2	Stage II: Orthogonalization (Parallelism) . . . . .	78
B.3.3	Stage III: Integration (Parallelism) . . . . .	79
B.3.4	Stage IV: Postprocessing (Parallelism) . . . . .	84







# List of Figures

6.1	The Table of Implemented Algorithms . . . . .	48
7.1	Comparison of Naïve and Block-Row Parallelism . . . . .	52
7.2	Scalability of Integration Algorithms . . . . .	53
7.4	The Time of iSVD using MATLAB and C++ . . . . .	56
7.5	The Time of iSVD using CPU and GPU . . . . .	56





# List of Tables

1.1	Notations Used in This Article . . . . .	3
3.1	Notations and Formulas Used in Optimization Integrations . . . . .	16
4.1	Communication Tree of Tall Skinny QR . . . . .	37
5.1	The Complexity of Sequential Algorithms . . . . .	44
5.2	The Complexity of Block-Row Algorithms . . . . .	44
5.3	The Complexity of Block-Column Algorithms . . . . .	44
B.1	Complexity of Canonical Methods . . . . .	70
B.2	Complexity of $\mathfrak{Q}$ Orthogonalization . . . . .	71
B.3	Complexity of Sketching . . . . .	72
B.4	Complexity of Orthogonalization . . . . .	72
B.5	Complexity of Integration . . . . .	73
B.8	Complexity of Postprocessing . . . . .	76
B.9	Complexity of Sketching (Naïve Parallelism) . . . . .	77
B.10	Complexity of Sketching (Block-Row Parallelism) . . . . .	77
B.11	Complexity of Sketching (Block-Column Parallelism) . . . . .	77
B.12	Complexity of Orthogonalization (Naïve Parallelism) . . . . .	78
B.13	Complexity of Orthogonalization (Block-Row Parallelism) . . . . .	78
B.14	Complexity of Integration (Naïve Parallelism) . . . . .	79
B.15	Complexity of Integration (Block-Row Parallelism) . . . . .	80
B.19	Complexity of Postprocessing (Block-Row Parallelism) . . . . .	84
B.21	Complexity of Postprocessing (Block-Column Parallelism) . . . . .	86





# List of Algorithms

## Main Algorithms

1	Integrated Singular Value Decomposition . . . . .	6
---	---	---

## Sketching Algorithms

I-1	Gaussian Projection Sketching . . . . .	7
I-2	Gaussian Projection Sketching (Block-Row Parallelism) . . . . .	35
I-3	Gaussian Projection Sketching (Block-Column Parallelism) . . . . .	39

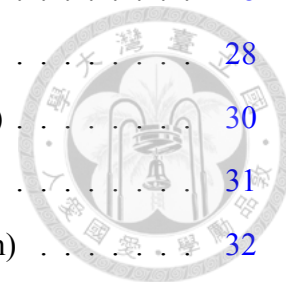
## Orthogonalization Algorithms

II-1	Canonical Orthogonalization . . . . .	7
II-2	Gramian Orthogonalization . . . . .	7
II-3	Gramian Orthogonalization (Block-Row Parallelism) . . . . .	35
II-4	TSQR Orthogonalization (Block-Row Parallelism) . . . . .	37

## Integration Algorithms

III-1	Kolmogorov-Nagumo Integration . . . . .	9
III-2	Wen-Yin Integration . . . . .	10
III-3	Hierarchical Reduction Integration . . . . .	11
III-4	Kolmogorov-Nagumo Integration (Optimized) . . . . .	19
III-5	Wen-Yin Integration (Optimized) . . . . .	22
III-6	Kolmogorov-Nagumo Integration (Gramian) . . . . .	24
III-7	Wen-Yin Integration (Gramian) . . . . .	25

III-8 Kolmogorov-Nagumo Integration (Naïve Parallelism) . . . . .	28
III-9 Hierarchical Reduction Integration (Naïve Parallelism) . . . . .	28
III-10 Kolmogorov-Nagumo Integration (Block-Row Parallelism) . . . . .	30
III-11 Wen-Yin Integration (Block-Row Parallelism) . . . . .	31
III-12 Hierarchical Reduction Integration (Block-Row Parallelism) . . . . .	32



### Postprocessing Algorithms

IV-1 Canonical Postprocessing . . . . .	12
IV-2 Gramian Postprocessing . . . . .	12
IV-3 Symmetric Postprocessing . . . . .	13
IV-4 Gramian Postprocessing (Block-Row Parallelism) . . . . .	36
IV-5 Symmetric Postprocessing (Block-Row Parallelism) . . . . .	36
IV-6 TSQR Postprocessing (Block-Row Parallelism) . . . . .	38
IV-7 Gramian Postprocessing (Block-Column Parallelism) . . . . .	39
IV-8 TSQR Postprocessing (Block-Column Parallelism) . . . . .	40
IV-9 Symmetric Postprocessing (Block-Column Parallelism) . . . . .	41

### Subroutines

S-1 Matrices Rearrangement (Block-Row Parallelism) . . . . .	33
S-2 Computing $\mathbf{B} = \mathbf{Q}^T \mathbf{Q}$ (Block-Row Parallelism) . . . . .	33
S-3 Computing $\mathbf{B}_c = \mathbf{Q}^T \mathbf{Q}_{\text{init}}$ (Block-Row Parallelism) . . . . .	33
S-4 Computing $\bar{\mathbf{Q}}_{\text{opt}} = \mathbf{Q}_{\text{init}} \tilde{\mathbf{F}} + \mathbf{Q} \tilde{\mathbf{E}}$ (Block-Row Parallelism) . . . . .	33
S-5 Matrices Gathering (Block-Row Parallelism) . . . . .	34



# Chapter 1

## Introduction

Big data analysis is one of the important fields in nowadays. We often use low-rank approximation for feature selection and dimension reduction. Singular value decomposition (SVD) is one of an essential tool for finding low-rank approximations. In this article, we consider the rank- $k$  singular value decomposition

$$\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \approx \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T, \quad (1.1)$$

where  $\mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$  is the full SVD and  $\mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$  is the truncated rank- $k$  SVD. However, traditional algorithms may take about  $12nm^2$  flops (assuming  $m < n$ ) for computing the SVD of an  $m \times n$  matrix  $\mathbf{A}$ , which lack scalability and leads to a computational problem when the matrix size is large. Randomized singular value decomposition (rSVD) [1, 2] reduces the computational cost to  $4nmk$  flops. Furthermore, Chen et al. proposed an efficient algorithm, integrated singular value decomposition (iSVD) (Algorithm 1) [3], which integrates the results from multiple rSVDs and achieve higher accuracy.

In this article, we focus on the implementation and the application of iSVD on large-scale clusters. The development of supercomputer allows us to handle huge scale problems. Multithread and multicore parallelization can speed up the computations. The computation is also benefited from GPU acceleration on hybrid CPU-GPU architecture. In these machine structures, data communication becomes an essential problem.

We modify the iSVD algorithms that reuse the matrices to reduce the computational

cost. For large-scale clusters, we propose new data structure for parallel algorithms to reduce data communication so that the communication cost is independent of the size of the problem. These algorithms are balanced and weak scalable. We compare the computational and communication complexity of each algorithm, and recommend the suitable choice for different shapes of the input matrix. The iSVD is implemented in C++ with several techniques for high maintainability, extensibility, and usability. We test our implementation on the Reedbush supercomputer system of the University of Tokyo with use up to 128 nodes and apply it to some huge size applications.

This article is organized as follows. The algorithms are introduced in [Chapter 2](#). The improvements of the algorithms and the parallelism ideas are discussed in [Chapters 3 to 5](#). [Chapter 6](#) describes the techniques of the implementation. Several numerical results are provided in [Chapter 7](#). The article ends with a discussion in [Chapter 8](#). The details of some formula are derived in [Appendix A](#). The tables of the complexity of all the algorithms are listed in [Appendix B](#).

For the notations, we use italic letters (e.g.  $m, \beta, N$ ) for scalars, bold italic uppercase letters (e.g.  $\mathbf{A}, \mathbf{\Omega}$ ) for matrices, italic sans serif uppercase letters (e.g.  $G, X$ ) for row/column-block submatrices, bold fraktur uppercase letters (e.g.  $\mathfrak{B}, \mathfrak{Q}$ ) for the matrices that keep unchanged in the iteration, under-script bracketed numbers (e.g.  $\mathbf{Q}_{[i]}, \mathbf{Y}_{[i]}$ ) for the matrices of the  $i$ -th sketches, super-script parenthesized numbers (e.g.  $\mathbf{U}^{(j)}, \mathbf{\Omega}^{(j)}$ ) for the  $j$ -th block-row in the  $j$ -th processor, super-script angled numbers (e.g.  $\mathbf{A}^{(j)}$ ) for the  $j$ -th block-column in the  $j$ -th processor, under-script  $c$  (e.g.  $\mathbf{Q}_c, \mathbf{X}_c$ ) for the matrices of the current iteration, and under-script plus sign (e.g.  $\mathbf{B}_+, \mathbf{D}_+$ ) for the matrices of the next iteration. [Table 1.1](#) lists the notations and the formulas used in this article.



$m, n$	Row and column dimensions of a matrix $\mathbf{A}$ with the assumption $m \leq n$ .
$k$	Desired rank of approximate SVD.
$p$	Oversampling parameter.
$l$	Dimension of randomized sketches, i.e., $l = (k + p) \ll m$ .
$N$	Number of random sketches.
$P$	Number of processors.
$m_b, n_b$	Row/column dimensions of a row/column-block, i.e., $m_b = \frac{m}{P}$ , $n_b = \frac{n}{P}$ .
$\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^\top$	An $m \times n$ matrix and its SVD.
$\mathbf{A} \approx \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$	Rank- $k$ SVD defined in <a href="#">eq. (1.1)</a> .
$\mathbf{A} \approx \hat{\mathbf{U}}_k \hat{\mathbf{\Sigma}}_k \hat{\mathbf{V}}_k^\top$	Rank- $k$ SVD computed by <a href="#">Algorithm 1</a> (iSVD).
$\mathbf{Q}_{[i]}$	The $i$ -th Gaussian random projection matrix.
$\mathbf{Y}_{[i]}$	The $i$ -th sketched matrix.
$\mathbf{Q}_{[i]}$	The $i$ th orthonormal basis of the sketched subspace.
$\bar{\mathbf{P}}$	The average of $\mathbf{Q}_{[i]} \mathbf{Q}_{[i]}^\top$ defined in <a href="#">eq. (3.2)</a> (stored implicitly).
$\bar{\mathbf{Q}}$	The integrated orthonormal basis of the sketched subspace.
$\bar{\mathbf{Q}}_{\text{opt}}$	The $\bar{\mathbf{Q}}$ of the optimization problem defined in <a href="#">eq. (3.2)</a> .
$\bar{\mathbf{Q}}_{\text{hr}}$	The $\bar{\mathbf{Q}}$ of hierarchical reduction integration defined in <a href="#">Algorithm III-3</a> .
$\ \bullet\ _F$	The Frobenius norm. $\ \mathbf{A}\ _F = \sqrt{\sum_i \sum_b a_{ij}^2} = \sqrt{\text{tr}(\mathbf{A}^\top \mathbf{A})}$ .
$\text{orth}(\bullet)$	Computes an orthonormal basis of given matrix using QR or SVD.
$\text{qr}(\bullet)$	Computes the QR decomposition.
$\text{eig}(\bullet)$	Computes the eigenvalue decomposition.
$\text{svd}(\bullet)$	Computes the singular value decomposition.

Table 1.1: Notations Used in This Article





# Chapter 2

## Algorithms

In this chapter, we briefly describe the Integrated Singular Value Decomposition algorithm, and the algorithms of each stage.

### 2.1 Integrated Singular Value Decomposition

Integrated Singular Value Decomposition (iSVD, [Algorithm 1](#)) [3] finds an approximate rank- $k$  SVD

$$\mathbf{A} \approx \hat{\mathbf{U}}_k \hat{\mathbf{\Sigma}}_k \hat{\mathbf{V}}_k^\top. \quad (2.1)$$

In the algorithm, we set  $\mathbf{A}$  as the matrix with size  $m \times n$ ,  $k$  as the desired rank of approximate SVD,  $p$  as the oversampling parameter,  $l = k + p$  as the dimension of the sketched column space, and  $N$  as the number of random sketches. We split iSVD into four stages.

- **Stage I: Sketching.** Sketches  $N$  rank- $l$  column subspaces of the input matrix  $\mathbf{A}$ . In other words, computes  $m \times l$  matrices  $\mathbf{Y}_{[i]}$  whose columns spans a column subspace of  $\mathbf{A}$ . A naïve way is multiplying  $\mathbf{A}$  by a random generated matrix  $\mathbf{\Omega}_{[i]}$ .
- **Stage II: Orthogonalization.** Computes an approximate basis for the range of the input matrix  $\mathbf{A}$  from those  $\mathbf{Y}_{[i]}$ ; that is, find orthogonal matrices  $\mathbf{Q}_{[i]}$  with

$$\mathbf{A} \approx \mathbf{Q}_{[i]} \mathbf{Q}_{[i]}^\top \mathbf{A}.$$

With  $\mathbf{Y}_{[i]}$ , one may directly orthogonalize them to obtain  $\mathbf{Q}_{[i]}$ .



- **Stage III: Integration.** Integrates  $\bar{\mathbf{Q}} \leftarrow \{\mathbf{Q}_{[i]}\}_{i=1}^N$ ; that is, find an orthonormal basis  $\bar{\mathbf{Q}}$  that best represent the  $\mathbf{Q}_{[i]}$ .
- **Stage IV: Postprocessing.** Computes a rank- $k$  approximate SVD  $\hat{\mathbf{U}}_k, \hat{\mathbf{\Sigma}}_k, \hat{\mathbf{V}}_k$  of  $\mathbf{A}$  in the range of  $\bar{\mathbf{Q}}$ ; i.e., find the SVD of

$$\bar{\mathbf{Q}}\bar{\mathbf{Q}}^\top \mathbf{A} = \hat{\mathbf{U}}_l \hat{\mathbf{\Sigma}}_l \hat{\mathbf{V}}_l^\top$$

and extract the largest  $k$  singular-pairs  $\hat{\mathbf{U}}_k, \hat{\mathbf{\Sigma}}_k, \hat{\mathbf{V}}_k$ .

---

**Algorithm 1** Integrated Singular Value Decomposition [3]

---

**Require:**  $\mathbf{A}$  (real  $m \times n$  matrix),  $k$  (desired rank of approximate SVD),  $p$  (oversampling parameter),  $l = k+p$  (dimension of the sketched column space),  $N$  (number of random sketches).

**Ensure:** Approximate rank- $k$  SVD of  $\mathbf{A} \approx \hat{\mathbf{U}}_k \hat{\mathbf{\Sigma}}_k \hat{\mathbf{V}}_k^\top$ .

- 1: **(Sketching.)** Compute  $m \times l$  matrices  $\mathbf{Y}_{[i]}$  whose columns spans a column subspace of  $\mathbf{A}$  for  $i = 1, \dots, N$ .
  - 2: **(Orthogonalization.)** Compute  $\mathbf{Q}_{[i]}$  whose columns are an orthonormal basis of  $\mathbf{Y}_{[i]}$  for  $i = 1, \dots, N$ .
  - 3: **(Integration.)** Integrate  $\bar{\mathbf{Q}} \leftarrow \{\mathbf{Q}_{[i]}\}_{i=1}^N$ .
  - 4: **(Postprocessing.)** Compute a rank- $k$  approximate SVD  $\hat{\mathbf{U}}_k \hat{\mathbf{\Sigma}}_k \hat{\mathbf{V}}_k^\top$  of  $\mathbf{A}$  in the range of  $\bar{\mathbf{Q}}$ .
- 

## 2.2 Stage I: Sketching

We use the same sketching algorithm as rSVD. [Algorithm I-1](#) (Gaussian Projection Sketching) multiples  $\mathbf{A}$  by some random matrices using Gaussian normal distribution.

---

**Algorithm I-1** Gaussian Projection Sketching

---

**Require:**  $\mathbf{A}$  (real  $m \times n$  matrix),  $l$  (dimension of the sketched column space),  $q$  (exponent of the power method),  $N$  (number of random sketches).

**Ensure:**  $\mathbf{Y}_{[i]}$  (real  $m \times l$  matrices) whose columns spans a column subspace of  $\mathbf{A}$  for  $i = 1, \dots, N$ .

- 1: Generate  $n \times l$  random matrices  $\mathbf{Q}_{[i]}$  using Gaussian normal distribution.
  - 2: Assign  $\mathbf{Y}_{[i]} \leftarrow (\mathbf{A}\mathbf{A}^\top)^q \mathbf{A}\mathbf{Q}_{[i]}$ .
- 

According to Halko et al. [1], while using this algorithm with  $q > 0$ , multiplying  $\mathbf{A}$  and  $\mathbf{A}^\top$  many times will cause rounding errors. They suggest orthogonalizing the columns between each multiplication of  $\mathbf{A}$  and  $\mathbf{A}^\top$ . In this article, we focus on the cases with  $q = 0$  so that there is no need to be concerned about this situation.

## 2.3 Stage II: Orthogonalization

In general, we can simply find the orthonormal basis using canonical QR or SVD of  $\mathbf{Y}_{[i]}$  (Algorithm II-1). Additional, we may also compute the orthonormal basis using eigenvalue decomposition of  $\mathbf{Y}_{[i]}^\top \mathbf{Y}_{[i]}$  — the Gramian of  $\mathbf{Y}_{[i]}$  (Algorithm II-2).

---

**Algorithm II-1** Canonical Orthogonalization

---

**Require:**  $\mathbf{Y}_{[i]}$  (real  $m \times l$  matrices).

**Ensure:**  $\mathbf{Q}_{[i]}$  (real  $m \times l$  matrices) whose columns are an orthonormal basis of  $\mathbf{Y}_{[i]}$ .

- 1: Compute  $\mathbf{Q}_{[i]}$  whose columns are an orthonormal basis of  $\mathbf{Y}_{[i]}$  using QR or SVD.
- 

In Canonical Orthogonalization, we can use both QR ( $\mathbf{Y}_{[i]} = \mathbf{Q}_{[i]}\mathbf{R}_{[i]}$ ) or SVD ( $\mathbf{Y}_{[i]} = \mathbf{Q}_{[i]}\mathbf{S}_{[i]}\mathbf{W}_{[i]}^\top$ ). Although these two  $\mathbf{Q}_{[i]}$  might not be exactly the same, they both span the same space; that is, the product  $\mathbf{Q}_{[i]}\mathbf{Q}_{[i]}^\top$  are exactly equal in both decompositions.

---

**Algorithm II-2** Gramian Orthogonalization

---

**Require:**  $\mathbf{Y}_{[i]}$  (real  $m \times l$  matrices).

**Ensure:**  $\mathbf{Q}_{[i]}$  (real  $m \times l$  matrices) whose columns are an orthonormal basis of  $\mathbf{Y}_{[i]}$ .

- 1: Compute  $\mathbf{W}_{[i]}\mathbf{S}_{[i]}^2\mathbf{W}_{[i]}^\top \leftarrow \text{eig}(\mathbf{Y}_{[i]}^\top \mathbf{Y}_{[i]})$ .
  - 2: Assign  $\mathbf{Q}_{[i]} \leftarrow \mathbf{Y}_{[i]}\mathbf{W}_{[i]}\mathbf{S}_{[i]}^{-1}$ .
-

Instead of computing the QR decomposition of a  $m \times l$  matrices  $\mathbf{Y}_{[i]}$  (Step 1 in Algorithm II-1), the Gramian Orthogonalization (Algorithm II-2) compute the eigenvalue decomposition of the  $l \times l$  matrices  $\mathbf{Y}_{[i]}^\top \mathbf{Y}_{[i]}$ . Denoting the SVD as  $\mathbf{Y}_{[i]} = \mathbf{Q}_{[i]} \mathbf{S}_{[i]} \mathbf{W}_{[i]}^\top$ , the eigenvalue decomposition of the Gramian matrix  $\mathbf{Y}_{[i]}^\top \mathbf{Y}_{[i]}$  can be written as

$$\mathbf{Y}_{[i]}^\top \mathbf{Y}_{[i]} = \mathbf{W}_{[i]}^\top \mathbf{S}_{[i]} \mathbf{Q}_{[i]}^\top \mathbf{Q}_{[i]} \mathbf{S}_{[i]} \mathbf{W}_{[i]} = \mathbf{W}_{[i]}^\top \mathbf{S}_{[i]}^2 \mathbf{W}_{[i]}. \quad (2.2)$$

Note that  $\mathbf{Q}_{[i]}^\top \mathbf{Q}_{[i]}$  is an identity matrix since  $\mathbf{Q}_{[i]}$  is orthonormal. With the eigenvalue decomposition, we can form the orthonormal basis by solving the equation

$$\mathbf{Q}_{[i]} = \mathbf{Y}_{[i]} \left( \mathbf{S}_{[i]} \mathbf{W}_{[i]}^\top \right)^{-1}. \quad (2.3)$$

Since  $\mathbf{W}_{[i]}$  is orthogonal and  $\mathbf{S}_{[i]}$  is diagonal, the inverse can be computed by multiplying the  $\mathbf{W}_{[i]}$  and dividing the columns by the diagonal elements of  $\mathbf{S}_{[i]}$ ; that is,

$$\mathbf{Q}_{[i]} = \mathbf{Y}_{[i]} \mathbf{W}_{[i]} \mathbf{S}_{[i]}^{-1}. \quad (2.4)$$

As shown in Chapter 5, the Gramain algorithm is faster.

## 2.4 Stage III: Integration

In the integration stage, we solve the optimization problem (see Section 3.1 for detail)

$$\bar{\mathbf{Q}}_{\text{opt}} := \arg \max_{\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}} f(\mathbf{Q}) \quad \text{with} \quad f(\mathbf{Q}) = \frac{1}{2} \text{tr} \left( \mathbf{Q}^\top \bar{\mathbf{P}} \mathbf{Q} \right). \quad (2.5)$$

There are two algorithms for this optimization problem. Algorithm III-1 uses the Kolmogorov-Nagumo-type average [4] and Algorithm III-2 uses the line search proposed by Wen and Yin [5].

---

**Algorithm III-1** Kolmogorov-Nagumo Integration [4]

---

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices),  $\mathbf{Q}_{\text{init}}$  (initial guess).

**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{opt}}$ .

- 1: Initialize the current iterate  $\mathbf{Q}_c \leftarrow \mathbf{Q}_{\text{init}}$ .
  - 2: **while** (not convergent) **do**
  - 3:   Assign  $\mathbf{X}_c \leftarrow (\mathbf{I} - \mathbf{Q}_c \mathbf{Q}_c^\top) \overline{\mathbf{P}} \mathbf{Q}_c$ .
  - 4:   Compute  $\mathbf{C} \leftarrow \left( \frac{\mathbf{I}}{2} + \left( \frac{\mathbf{I}}{4} - \mathbf{X}_c^\top \mathbf{X}_c \right)^{1/2} \right)^{1/2}$ .
  - 5:   Update  $\mathbf{Q}_c \leftarrow \mathbf{Q}_c \mathbf{C} + \mathbf{X}_c \mathbf{C}^{-1}$ .
  - 6: **end while**
  - 7: Output  $\overline{\mathbf{Q}}_{\text{opt}} \leftarrow \mathbf{Q}_c$ .
- 



In Kolmogorov-Nagumo Integration, we stop the iteration if  $(\mathbf{I} - \mathbf{Q}_+^\top \mathbf{Q}_c)$  is small enough. This condition measures the similarity of  $\mathbf{Q}_+$  and  $\mathbf{Q}_c$ . In the implementation, we use an equivalent condition  $\|\mathbf{C}\|_2 < \epsilon$  for some tolerance  $\epsilon$ .

---

**Algorithm III-2** Wen-Yin Integration [5]

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices),  $\mathbf{Q}_{\text{init}}$  (initial guess),  $\tau_0 \geq 0$  (initial step size),  $\beta \in (0, 1)$  (scaling parameter for step size searching),  $\sigma \in (0, 1)$  (parameter for step size searching),  $\eta \in (0, 1)$  (parameter for next step searching),  $\tau_{\max}, \tau_{\min}$  (maximum and minimum predicting step size).

**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{opt}}$ .

- 1: Initialize  $\mathbf{Q}_c \leftarrow \mathbf{Q}_{\text{init}}$ ,  $\tau_g \leftarrow \tau_0$ ,  $\zeta \leftarrow 1$ ,  $\phi \leftarrow f(\mathbf{Q}_c)$ .
- 2: **while** (not convergent) **do**
- 3:   Assign  $\mathbf{G}_c \leftarrow \overline{\mathbf{P}}\mathbf{Q}_c$ .
- 4:   Let  $\tau = \tau_g \beta^t$ . Find the smallest integer  $t \geq 0$  satisfying the inequality

$$\tilde{\phi} \geq \phi + \tau \sigma \frac{1}{2} \|\mathbf{M}\|_F^2,$$

where  $\tilde{\phi} = f(\mathbf{Q}_+)$ ,  $\mathbf{Q}_+ = \left(\mathbf{I} - \frac{\tau}{2}\mathbf{M}\right)^{-1} \left(\mathbf{I} + \frac{\tau}{2}\mathbf{M}\right) \mathbf{Q}_c$ , and  $\mathbf{M} = \mathbf{G}_c \mathbf{Q}_c^\top - \mathbf{Q}_c \mathbf{G}_c^\top$ .

- 5:   Update  $\phi \leftarrow \frac{\eta \zeta \phi + \tilde{\phi}}{\eta \zeta + 1}$  and then  $\zeta \leftarrow \eta \zeta + 1$ .
- 6:   Compute the differences  $\mathbf{\Delta}_1 = \mathbf{Q}_+ - \mathbf{Q}_c$  and  $\mathbf{\Delta}_2 = \mathbf{X}_+ - \mathbf{X}_c$ , where

$$\begin{aligned} \mathbf{X}_c &= (\mathbf{I} - \mathbf{Q}_c \mathbf{Q}_c^\top) \overline{\mathbf{P}} \mathbf{Q}_c \\ \mathbf{X}_+ &= (\mathbf{I} - \mathbf{Q}_+ \mathbf{Q}_+^\top) \overline{\mathbf{P}} \mathbf{Q}_+ \end{aligned}$$

- 7:   Update  $\tau_g \leftarrow \max(\min(\tau_{\text{guess}}, \tau_{\max}), \tau_{\min})$ , where

$$\tau_{\text{guess}} = \frac{\text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_1)}{|\text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_2)|} \quad \text{or} \quad \frac{|\text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_2)|}{\text{tr}(\mathbf{\Delta}_2^\top \mathbf{\Delta}_2)}.$$

- 8:   Assign  $\mathbf{Q}_c \leftarrow \mathbf{Q}_+$ .
  - 9: **end while**
  - 10: Output  $\overline{\mathbf{Q}}_{\text{opt}} \leftarrow \mathbf{Q}_c$ .
- 

In Wen-Yin Integration, we stop the iteration if  $\mathbf{X}_c$  small enough [5]. In the implementation, we use an equivalent condition  $\|\mathbf{X}_c\|_F < \epsilon$  for some tolerance  $\epsilon$ . Note that  $\|\mathbf{X}_c\|_F^2 = \frac{1}{2} \|\mathbf{M}\|_F^2$ , which is already computed in Step 4.

Instead of solving the optimization problem, Chang proposed a divide and conquer algorithm (Algorithm III-3) [6]. It integrates every two  $\mathbf{Q}_{[i]}$  recursively (see Appendix A.1 for detail).



---

**Algorithm III-3** Hierarchical Reduction Integration [6]

---

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices).

**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{hr}}$ .

- 1: Set  $\tilde{N} \leftarrow N$ .
  - 2: **while**  $\tilde{N} > 1$  **do**
  - 3:   Set  $h \leftarrow \left\lfloor \frac{\tilde{N}}{2} \right\rfloor$
  - 4:   **for**  $t = 1$  **to**  $h$  **do**
  - 5:     Compute  $\mathbf{WST}^\top \leftarrow \text{svd}(\mathbf{Q}_{[i]}^\top \mathbf{Q}_{[i+h]})$ .
  - 6:     Update  $\mathbf{Q}_{[i]} \leftarrow (\mathbf{Q}_{[i]}\mathbf{W} + \mathbf{Q}_{[i+h]}\mathbf{T}) (2(\mathbf{I} + \mathbf{S}))^{-1/2}$ .
  - 7:   **end for**
  - 8:   Update  $\tilde{N} \leftarrow \left\lfloor \frac{\tilde{N}}{2} \right\rfloor$ .
  - 9: **end while**
  - 10: Output  $\overline{\mathbf{Q}}_{\text{hr}} \leftarrow \mathbf{Q}_{[1]}$ .
- 



As shown in [Chapter 5](#), the Hierarchical Reduction Integration much faster than solving the optimization problem. It costs  $O(Nml^2)$  only, which is roughly the same the complexity of a single iteration in Kolmogorov-Nagumo Iteration and the Wen-Yin Iteration. However, according to Chang [6], the result is less accurate but still better than any one  $\mathbf{Q}_{[i]}$ . He suggests using this algorithm as a preprocessing of finding  $\mathbf{Q}_{\text{init}}$  of the Kolmogorov-Nagumo Iteration and the Wen-Yin Iteration to reduce the number of iteration.

## 2.5 Stage IV: Postprocessing

There are several methods for postprocessing. [Algorithm IV-1](#), the canonical method, forms the decomposition using SVD of  $\overline{\mathbf{Q}}^\top \mathbf{A}$ . Similar to the Gramian Orthogonalization ([Algorithm II-2](#)), [Algorithm IV-2](#) compute the eigenvalue decomposition of  $\overline{\mathbf{Q}}^\top \mathbf{A} \mathbf{A}^\top \overline{\mathbf{Q}}$  (the Gramian of  $\mathbf{A}^\top \overline{\mathbf{Q}}$ ) instead of computing SVD.

---

**Algorithm IV-1** Canonical Postprocessing

---

**Require:**  $\mathbf{A}$  (real  $m \times n$  matrix),  $\overline{\mathbf{Q}}$  (real  $m \times l$  orthogonal matrix),  $k$  (desired rank of approximate SVD).

**Ensure:** Approximate rank- $k$  SVD of  $\mathbf{A} \approx \widehat{\mathbf{U}}_k \widehat{\mathbf{\Sigma}}_k \widehat{\mathbf{V}}_k^\top$ .

- 1: Compute  $\widehat{\mathbf{W}}_l \widehat{\mathbf{\Sigma}}_l \widehat{\mathbf{V}}_l^\top \leftarrow \text{svd}(\overline{\mathbf{Q}}^\top \mathbf{A})$ .
  - 2: Extract the largest  $k$  singular pairs from  $\widehat{\mathbf{W}}_l, \widehat{\mathbf{\Sigma}}_l, \widehat{\mathbf{V}}_l$  to obtain  $\widehat{\mathbf{W}}_k, \widehat{\mathbf{\Sigma}}_k, \widehat{\mathbf{V}}_k$ .
  - 3: Assign  $\widehat{\mathbf{U}}_k \leftarrow \overline{\mathbf{Q}} \widehat{\mathbf{W}}_k$ .
- 

Since the size of the projected matrix  $\overline{\mathbf{Q}} \overline{\mathbf{Q}}^\top \mathbf{A}$  is equal to the input matrix  $\mathbf{A}$ , computing the SVD of  $\overline{\mathbf{Q}} \overline{\mathbf{Q}}^\top \mathbf{A}$  is unwise. Canonical Postprocessing computes the SVD of the smaller matrix  $\overline{\mathbf{Q}}^\top \mathbf{A}$  in order to reduce the computing complexity. Denoting the SVD as  $\overline{\mathbf{Q}}^\top \mathbf{A} = \widehat{\mathbf{W}}_l \widehat{\mathbf{\Sigma}}_l \widehat{\mathbf{V}}_l^\top$ , the SVD of  $\overline{\mathbf{Q}} \overline{\mathbf{Q}}^\top \mathbf{A}$  can be written as

$$\overline{\mathbf{Q}} \overline{\mathbf{Q}}^\top \mathbf{A} = \overline{\mathbf{Q}} (\widehat{\mathbf{W}}_l \widehat{\mathbf{\Sigma}}_l \widehat{\mathbf{V}}_l^\top) = (\overline{\mathbf{Q}} \widehat{\mathbf{W}}_l) \widehat{\mathbf{\Sigma}}_l \widehat{\mathbf{V}}_l^\top = \widehat{\mathbf{U}}_l \widehat{\mathbf{\Sigma}}_l \widehat{\mathbf{V}}_l^\top. \quad (2.6)$$

Note that the product  $\widehat{\mathbf{U}}_l$  is a orthogonal matrix since  $\overline{\mathbf{Q}}$  and  $\widehat{\mathbf{W}}_l$  are orthogonal.

---

**Algorithm IV-2** Gramian Postprocessing

---

**Require:**  $\mathbf{A}$  (real  $m \times n$  matrix),  $\overline{\mathbf{Q}}$  (real  $m \times l$  orthogonal matrix),  $k$  (desired rank of approximate SVD).

**Ensure:** Approximate rank- $k$  SVD of  $\mathbf{A} \approx \widehat{\mathbf{U}}_k \widehat{\mathbf{\Sigma}}_k \widehat{\mathbf{V}}_k^\top$ .

- 1: Assign of  $\mathbf{Z} \leftarrow \mathbf{A}^\top \overline{\mathbf{Q}}$ .
  - 2: Compute  $\widehat{\mathbf{W}}_l \widehat{\mathbf{\Sigma}}_l^2 \widehat{\mathbf{W}}_l^\top \leftarrow \text{eig}(\mathbf{Z}^\top \mathbf{Z})$ .
  - 3: Extract the largest  $k$  eigen-pairs from  $\widehat{\mathbf{W}}_l, \widehat{\mathbf{\Sigma}}_l$  to obtain  $\widehat{\mathbf{W}}_k, \widehat{\mathbf{\Sigma}}_k$ .
  - 4: Assign  $\widehat{\mathbf{U}}_k \leftarrow \overline{\mathbf{Q}} \widehat{\mathbf{W}}_k$ .
  - 5: Assign  $\widehat{\mathbf{V}}_k \leftarrow \mathbf{Z} \widehat{\mathbf{W}}_k \widehat{\mathbf{\Sigma}}_k^{-1}$ .
- 

For symmetric  $\mathbf{A}$ , Halko, Martinsson and Tropp proposed an elegant algorithm [1] (Algorithm IV-3) for this situation. The algorithm is much faster than the canonical method and keeps the symmetry of the result, with about twice error than the canonical algorithm.

---

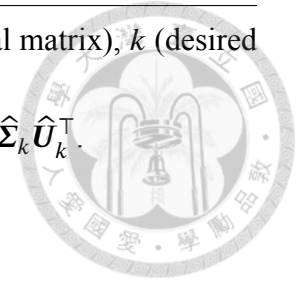
**Algorithm IV-3** Symmetric Postprocessing [1]

---

**Require:**  $\mathbf{A}$  (real symmetric  $m \times m$  matrix),  $\overline{\mathbf{Q}}$  (real  $m \times l$  orthogonal matrix),  $k$  (desired rank of approximate SVD).

**Ensure:** Approximate rank- $k$  eigenvalue decomposition of  $\mathbf{A} \approx \hat{\mathbf{U}}_k \hat{\Sigma}_k \hat{\mathbf{U}}_k^\top$ .

- 1: Compute  $\widehat{\mathbf{W}}_l \widehat{\Sigma}_l \widehat{\mathbf{W}}_l^\top \leftarrow \text{eig}(\overline{\mathbf{Q}}^\top \mathbf{A} \overline{\mathbf{Q}})$ .
  - 2: Extract the largest  $k$  eigen-pairs from  $\widehat{\mathbf{W}}_l, \widehat{\Sigma}_l$  to obtain  $\widehat{\mathbf{W}}_k, \widehat{\Sigma}_k$ .
  - 3: Assign  $\hat{\mathbf{U}}_k \leftarrow \overline{\mathbf{Q}} \widehat{\mathbf{W}}_k$ .
- 







# Chapter 3

## Improvements of Integration

In this chapter, we optimize the Kolmogorov-Nagumo Integration ([Algorithm III-1](#)) and the Wen-Yin Integration ([Algorithm III-2](#)) for better performance, and propose algorithms based on the Gramian idea target for the case with many iterations. [Table 3.1](#) lists the notations and the formulas used in this chapter. Here, we use bold italic uppercase letters (e.g.  $\mathbf{A}$ ,  $\mathbf{Q}$ ) for matrices, bold fraktur uppercase letters (e.g.  $\mathfrak{B}$ ,  $\mathfrak{Q}$ ) for the matrices that keep unchanged in the iteration. Under-script  $c$  (e.g.  $\mathbf{Q}_c$ ,  $\mathbf{X}_c$ ) are used for the matrices of the current iteration, and under-script plus sign (e.g.  $\mathbf{B}_+$ ,  $\mathbf{D}_+$ ) are used for the matrices of the next iteration. Moreover, we use matrices with super-script  $g$  for the  $\mathbf{G}_c$  terms. For example, in the updating of  $\mathbf{Q}_+$ ,  $F_c$  is the coefficient of  $\mathbf{Q}_c$ , and  $F_c^g$  is the coefficient of  $\mathbf{G}_c$ .

### 3.1 Optimization Problem

The integration stage finds an orthogonal matrix  $\bar{\mathbf{Q}}$  that best represent the orthonormal basis  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$ . Here, we define such best  $\bar{\mathbf{Q}}_{\text{opt}}$  as

$$\bar{\mathbf{Q}}_{\text{opt}} := \arg \min_{\mathbf{Q}^T \mathbf{Q} = \mathbf{I}} \frac{1}{N} \sum_{i=1}^N \left\| \mathbf{Q}_{[i]} \mathbf{Q}_{[i]}^T - \mathbf{Q} \mathbf{Q}^T \right\|_F^2. \quad (3.1)$$


$\mathfrak{Q} = [\mathbf{Q}_{[1]} \quad \mathbf{Q}_{[2]} \quad \cdots \quad \mathbf{Q}_{[N]}]$ $\mathfrak{B} = \mathfrak{Q}^\top \mathfrak{Q}$	
$\mathbf{G}_c = \overline{\mathbf{P}} \mathbf{Q}_c = \frac{1}{N} \mathfrak{Q} \mathbf{B}_c$ $\mathbf{X}_c = (\mathbf{I} - \mathbf{Q}_c \mathbf{Q}_c^\top) \overline{\mathbf{P}} \mathbf{Q}_c = \mathbf{G}_c - \mathbf{Q}_c \mathbf{D}_c$	
$\mathbf{B}_c = \mathfrak{Q}^\top \mathbf{Q}_c$ $\mathbf{B}_c^g = \mathfrak{Q}^\top \mathbf{G}_c = \frac{1}{N} \mathfrak{B} \mathbf{B}_c$ $\mathbf{D}_c = \mathbf{Q}_c^\top \overline{\mathbf{P}} \mathbf{Q}_c = \mathbf{Q}_c^\top \mathbf{G}_c = \mathbf{G}_c^\top \mathbf{Q}_c = \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c$ $\mathbf{D}_c^g = \mathbf{Q}_c^\top \overline{\mathbf{P}}^2 \mathbf{Q}_c = \mathbf{G}_c^\top \mathbf{G}_c = \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c^g = \frac{1}{N^2} \mathbf{B}_c^\top \mathfrak{B} \mathbf{B}_c$	
$\mathbf{C}$ defined in Step 4 of Algorithm III-1	
$\mathbf{C}_{11}^{-1}, \mathbf{C}_{12}^{-1}$ $\mathbf{C}_{21}^{-1}, \mathbf{C}_{22}^{-1}$ defined in Eq. (3.19)	
$\mathbf{F}_c = \begin{cases} \mathbf{C} - \mathbf{D}_c \mathbf{C}^{-1} & \text{in Kolmogorov-Nagumo Integration} \\ \mathbf{I} - \mathbf{C}_{22}^{-1} \mathbf{D}_c - \mathbf{C}_{21}^{-1} & \text{in Wen-Yin Integration} \end{cases}$ $\mathbf{F}_c^g = \begin{cases} \mathbf{C}^{-1} & \text{in Kolmogorov-Nagumo Integration} \\ \mathbf{C}_{12}^{-1} \mathbf{D}_c - \mathbf{C}_{11}^{-1} & \text{in Wen-Yin Integration} \end{cases}$ $\mathbf{E}_c = \frac{1}{N} \mathbf{B}_c \mathbf{F}_c^g$	
$\mathbf{Q}_+ = \mathbf{Q}_c \mathbf{F}_c + \mathbf{G}_c \mathbf{F}_c^g = \mathbf{Q}_c \mathbf{F}_c + \mathfrak{Q} \mathbf{E}_c$ $\mathbf{B}_+ = \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g = \mathbf{B}_c \mathbf{F}_c + \mathfrak{B} \mathbf{E}_c$ $\overline{\mathbf{Q}}_{\text{opt}} = \mathbf{Q}_{\text{init}} \tilde{\mathbf{F}} + \mathfrak{Q} \tilde{\mathbf{E}}$ $\tilde{\mathbf{F}}_+ = \tilde{\mathbf{F}}_c \mathbf{F}_c$ $\tilde{\mathbf{E}}_+ = \tilde{\mathbf{E}}_c \mathbf{F}_c + \mathbf{E}_c$	

Table 3.1: Notations and Formulas Used in Optimization Integrations

The optimization problem is equivalent to a maximization problem

$$\overline{\mathbf{Q}}_{\text{opt}} := \arg \max_{\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}} \frac{1}{2} \text{tr}(\mathbf{Q}^\top \overline{\mathbf{P}} \mathbf{Q}) \quad \text{with} \quad \overline{\mathbf{P}} := \frac{1}{N} \sum_{i=1}^N \mathbf{Q}_{[i]} \mathbf{Q}_{[i]}^\top. \quad (3.2)$$

Here, we define

$$f(\mathbf{Q}) = \frac{1}{2} \text{tr}(\mathbf{Q}^\top \overline{\mathbf{P}} \mathbf{Q}) \quad (3.3)$$

as the objective function.

## 3.2 Improvements of Kolmogorov-Nagumo Integration

In the implementation, instead of explicitly forming  $m \times m$  matrices such as  $\mathbf{Q}_c \mathbf{Q}_c^\top$  (with  $2m^2l$  flops), we compute  $l \times l$  matrices such as  $\mathbf{Q}_{[i]}^\top \mathbf{Q}_c$  (with  $2ml^2$  flops) to reduce computational cost and memory usage. For example, [Step 3 in Algorithm III-1](#) (Kolmogorov-Nagumo Integration) can be rewritten as

$$\begin{aligned}
 \mathbf{X}_c &= (\mathbf{I} - \mathbf{Q}_c \mathbf{Q}_c^\top) \bar{\mathbf{P}} \mathbf{Q}_c = (\mathbf{I} - \mathbf{Q}_c \mathbf{Q}_c^\top) \left( \frac{1}{N} \sum_{i=1}^N \mathbf{Q}_{[i]} \mathbf{Q}_{[i]}^\top \right) \mathbf{Q}_c \\
 &= \frac{1}{N} \sum_{i=1}^N (\mathbf{I} - \mathbf{Q}_c \mathbf{Q}_c^\top) \mathbf{Q}_{[i]} \mathbf{Q}_{[i]}^\top \mathbf{Q}_c \\
 &= \frac{1}{N} \sum_{i=1}^N \mathbf{Q}_{[i]} \mathbf{Q}_{[i]}^\top \mathbf{Q}_c - \frac{1}{N} \sum_{i=1}^N \mathbf{Q}_c \mathbf{Q}_c^\top \mathbf{Q}_{[i]} \mathbf{Q}_{[i]}^\top \mathbf{Q}_c \\
 &= \frac{1}{N} \sum_{i=1}^N \mathbf{Q}_{[i]} (\mathbf{Q}_{[i]}^\top \mathbf{Q}_c) - \frac{1}{N} \mathbf{Q}_c \sum_{i=1}^N (\mathbf{Q}_{[i]}^\top \mathbf{Q}_c)^\top (\mathbf{Q}_{[i]}^\top \mathbf{Q}_c) \\
 &= \frac{1}{N} \sum_{i=1}^N \mathbf{Q}_{[i]} \mathbf{B}_{[i]} - \frac{1}{N} \mathbf{Q}_c \sum_{i=1}^N \mathbf{B}_{[i]}^\top \mathbf{B}_{[i]},
 \end{aligned} \tag{3.4}$$

where  $\mathbf{B}_{[i]} = \mathbf{Q}_{[i]}^\top \mathbf{Q}_c$  are  $l \times l$  matrices. Moreover, those matrix products can be accelerated by combining the matrices

$$\mathfrak{N}_{[1]} \mathfrak{Q}_{[1]} + \mathfrak{N}_{[2]} \mathfrak{Q}_{[2]} + \cdots + \mathfrak{N}_{[N]} \mathfrak{Q}_{[N]} = \begin{bmatrix} \mathfrak{N}_{[1]} & \mathfrak{N}_{[2]} & \cdots & \mathfrak{N}_{[N]} \end{bmatrix} \begin{bmatrix} \mathfrak{Q}_{[1]} \\ \mathfrak{Q}_{[2]} \\ \vdots \\ \mathfrak{Q}_{[N]} \end{bmatrix}. \tag{3.5}$$

Therefore, [eq. \(3.4\)](#) can be rewritten as

$$\mathbf{X}_c = \mathbf{G}_c - \mathbf{Q}_c \mathbf{D}_c, \tag{3.6}$$

where  $\mathbf{G}_c = \frac{1}{N} \mathfrak{Q} \mathbf{B}_c$ ,  $\mathbf{D}_c = \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c$ ,

$$\mathfrak{Q} = \begin{bmatrix} \mathbf{Q}_{[1]} & \mathbf{Q}_{[2]} & \cdots & \mathbf{Q}_{[N]} \end{bmatrix} \quad \text{and} \quad \mathbf{B}_c = \begin{bmatrix} \mathbf{B}_{[1]} & \mathbf{B}_{[2]} & \cdots & \mathbf{B}_{[N]} \end{bmatrix}. \tag{3.7}$$

Note that we may compute  $\mathbf{B}_c$  as  $\mathbf{B}_c = \mathfrak{Q}^\top \mathbf{Q}_c$ . Hence, the updating of  $\mathbf{Q}_+$  (Step 5 in Algorithm III-1 Kolmogorov-Nagumo Integration) can be written as

$$\mathbf{Q}_+ = \mathbf{Q}_c \mathbf{C} + \mathbf{X}_c \mathbf{C}^{-1} = \mathbf{Q}_c \mathbf{C} + \mathbf{G}_c \mathbf{C}^{-1} - \mathbf{Q}_c \mathbf{D}_c \mathbf{C}^{-1} = \mathbf{Q}_c \mathbf{F}_c + \mathbf{G}_c \mathbf{F}_c^g, \quad (3.8)$$

where

$$\mathbf{F}_c = \mathbf{C} - \mathbf{D}_c \mathbf{C}^{-1}, \quad \mathbf{F}_c^g = \mathbf{C}^{-1}. \quad (3.9)$$

Similarly, we can update  $\mathbf{B}_+ = \mathfrak{Q}^\top \mathbf{Q}_+$  as

$$\mathbf{B}_+ = \mathfrak{Q}^\top \mathbf{Q}_+ = \mathfrak{Q}^\top \mathbf{Q}_c \mathbf{F}_c + \mathfrak{Q}^\top \mathbf{G}_c \mathbf{F}_c^g = \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g, \quad (3.10)$$

where  $\mathbf{B}_c^g = \mathfrak{Q}^\top \mathbf{G}_c$ . Furthermore, instead of forming  $\mathbf{X}_c$ , we may compute  $\mathbf{\Xi} = \mathbf{X}_c^\top \mathbf{X}_c$  directly as

$$\begin{aligned} \mathbf{\Xi} &= \mathbf{X}_c^\top \mathbf{X}_c = (\mathbf{G}_c^\top - \mathbf{D}_c \mathbf{Q}_c^\top)(\mathbf{G}_c - \mathbf{Q}_c \mathbf{D}_c) \\ &= \mathbf{G}_c^\top \mathbf{G}_c - \mathbf{G}_c^\top \mathbf{Q}_c \mathbf{D}_c - \mathbf{D}_c \mathbf{Q}_c^\top \mathbf{G}_c + \mathbf{D}_c \mathbf{Q}_c^\top \mathbf{Q}_c \mathbf{D}_c \\ &= \mathbf{G}_c^\top \mathbf{G}_c - \mathbf{D}_c \mathbf{D}_c - \mathbf{D}_c \mathbf{D}_c + \mathbf{D}_c \mathbf{D}_c \\ &= \mathbf{D}_c^g - \mathbf{D}_c^2, \end{aligned} \quad (3.11)$$

where  $\mathbf{D}_c^g = \mathbf{G}_c^\top \mathbf{G}_c = \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c^g$ . Note that  $\mathbf{Q}_c^\top \mathbf{G}_c = \mathbf{G}_c^\top \mathbf{Q}_c = \mathbf{D}_c$  and  $\mathbf{Q}_c^\top \mathbf{Q}_c = \mathbf{I}$ .



---

**Algorithm III-4** Kolmogorov-Nagumo Integration (Optimized)

---

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices),  $\mathbf{Q}_{\text{init}}$  (initial guess).

**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{opt}}$ .

- 1: Combine  $\mathfrak{Q} = \begin{bmatrix} \mathbf{Q}_{[1]} & \mathbf{Q}_{[2]} & \dots & \mathbf{Q}_{[N]} \end{bmatrix}$ .
  - 2: Initialize the current iterate  $\mathbf{Q}_c \leftarrow \mathbf{Q}_{\text{init}}$ .
  - 3: Assign  $\mathbf{B}_c \leftarrow \mathfrak{Q}^\top \mathbf{Q}_c$ .
  - 4: **while** (not convergent) **do**
  - 5: Assign  $\mathbf{G}_c \leftarrow \frac{1}{N} \mathfrak{Q} \mathbf{B}_c$ ,  $\mathbf{B}_c^g \leftarrow \mathfrak{Q}^\top \mathbf{G}_c$ ,  $\mathbf{D}_c \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c$ ,  $\mathbf{D}_c^g \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c^g$ .
  - 6: Compute  $\mathbf{C} \leftarrow \left( \frac{\mathbf{I}}{2} + \left( \frac{\mathbf{I}}{4} - \mathbf{\Xi} \right)^{1/2} \right)^{1/2}$ , where  $\mathbf{\Xi} = \mathbf{D}_c^g - \mathbf{D}_c^2$ .
  - 7: Assign  $\mathbf{F}_c \leftarrow \mathbf{C} - \mathbf{D}_c \mathbf{C}^{-1}$  and  $\mathbf{F}_c^g \leftarrow \mathbf{C}^{-1}$ .
  - 8: Update  $\mathbf{Q}_c \leftarrow \mathbf{Q}_c \mathbf{F}_c + \mathbf{G}_c \mathbf{F}_c^g$  and  $\mathbf{B}_c \leftarrow \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g$ .
  - 9: **end while**
  - 10: Output  $\overline{\mathbf{Q}}_{\text{opt}} \leftarrow \mathbf{Q}_c$ .
- 



### 3.3 Improvements of Wen-Yin Integration

Similar to the Optimized Kolmogorov-Nagumo Integration ([Algorithm III-4](#)), we combine the matrices  $\mathfrak{Q} = \begin{bmatrix} \mathbf{Q}_{[1]} & \mathbf{Q}_{[2]} & \dots & \mathbf{Q}_{[N]} \end{bmatrix}$  and define

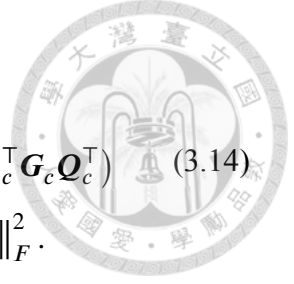
$$\begin{aligned} \mathbf{G}_c &= \overline{\mathbf{P}} \mathbf{Q}_c = \frac{1}{N} \mathfrak{Q} \mathbf{B}_c, \\ \mathbf{B}_c &= \mathfrak{Q}^\top \mathbf{Q}_c, \\ \mathbf{B}_c^g &= \mathfrak{Q}^\top \mathbf{G}_c, \\ \mathbf{D}_c &= \mathbf{Q}_c^\top \overline{\mathbf{P}} \mathbf{Q}_c = \mathbf{Q}_c^\top \mathbf{G}_c = \mathbf{G}_c^\top \mathbf{Q}_c = \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c, \\ \mathbf{D}_c^g &= \mathbf{Q}_c^\top \overline{\mathbf{P}}^2 \mathbf{Q}_c = \mathbf{G}_c^\top \mathbf{G}_c = \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c^g. \end{aligned} \tag{3.12}$$

We observed that  $f(\mathbf{Q}_c)$  in [Step 4](#) of [Algorithm III-2](#) (Wen-Yin Integration) can be computed by

$$f(\mathbf{Q}_c) = \frac{1}{2} \text{tr} \left( \mathbf{Q}_c^\top \overline{\mathbf{P}} \mathbf{Q}_c \right) = \frac{1}{2} \text{tr} \left( \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c \right) = \frac{1}{2N} \|\mathbf{B}_c\|_F^2. \tag{3.13}$$

Moreover,  $\|\mathbf{M}\|_F^2$  can be written as

$$\begin{aligned}
\|\mathbf{M}\|_F^2 &= \text{tr}(\mathbf{M}^\top \mathbf{M}) = \text{tr}((\mathbf{Q}_c \mathbf{G}_c^\top - \mathbf{G}_c \mathbf{Q}_c^\top)(\mathbf{G}_c \mathbf{Q}_c^\top - \mathbf{Q}_c \mathbf{G}_c^\top)) \\
&= \text{tr}(\mathbf{Q}_c \mathbf{G}_c^\top \mathbf{G}_c \mathbf{Q}_c^\top + \mathbf{G}_c^\top \mathbf{Q}_c^\top \mathbf{Q}_c \mathbf{G}_c^\top - \mathbf{Q}_c \mathbf{G}_c^\top \mathbf{Q}_c \mathbf{G}_c^\top - \mathbf{G}_c \mathbf{Q}_c^\top \mathbf{G}_c \mathbf{Q}_c^\top) \quad (3.14) \\
&= 2 \text{tr}(\mathbf{G}_c^\top \mathbf{G}_c) - 2 \text{tr}(\mathbf{Q}_c^\top \mathbf{G}_c \mathbf{Q}_c^\top \mathbf{G}_c) = 2 \text{tr}(\mathbf{D}_c^g) - 2 \|\mathbf{D}_c\|_F^2.
\end{aligned}$$



To compute  $\mathbf{Q}_+$ , instead of explicitly forming  $m \times m$  matrix  $\mathbf{M} = \mathbf{G}_c \mathbf{Q}_c^\top - \mathbf{Q}_c \mathbf{G}_c^\top$ , we construct two low-rank matrices [5]

$$\mathbf{L} = \begin{bmatrix} \mathbf{G}_c & \mathbf{Q}_c \end{bmatrix} \quad \text{and} \quad \mathbf{R} = \begin{bmatrix} \mathbf{Q}_c & -\mathbf{G}_c \end{bmatrix} \quad (3.15)$$

with  $\mathbf{M} = \mathbf{L}\mathbf{R}^\top$ . Using Woodbury matrix identity, the inverse can be rewritten as

$$\left(\mathbf{I} - \frac{\tau}{2} \mathbf{M}\right)^{-1} = \left(\mathbf{I} - \frac{\tau}{2} \mathbf{L}\mathbf{R}^\top\right)^{-1} = \mathbf{I} - \mathbf{L} \left(\mathbf{R}^\top \mathbf{L} - \frac{2}{\tau} \mathbf{I}\right)^{-1} \mathbf{R}^\top. \quad (3.16)$$

Therefore,

$$\begin{aligned}
\mathbf{Q}_+ &= \left(\mathbf{I} - \frac{\tau}{2} \mathbf{M}\right)^{-1} \left(\mathbf{I} + \frac{\tau}{2} \mathbf{M}\right) \mathbf{Q}_c = \left(2 \left(\mathbf{I} - \frac{\tau}{2} \mathbf{M}\right)^{-1} - \mathbf{I}\right) \mathbf{Q}_c \\
&= \mathbf{Q}_c - \mathbf{L} \left(\frac{1}{2} \mathbf{R}^\top \mathbf{L} - \frac{1}{\tau} \mathbf{I}\right)^{-1} \mathbf{R}^\top \mathbf{Q}_c \\
&= \mathbf{Q}_c - \begin{bmatrix} \mathbf{G}_c & \mathbf{Q}_c \end{bmatrix} \mathbf{C}^{-1} \begin{bmatrix} \mathbf{Q}_c^\top \\ -\mathbf{G}_c^\top \end{bmatrix} \mathbf{Q}_c \quad (3.17) \\
&= \mathbf{Q}_c - \begin{bmatrix} \mathbf{G}_c & \mathbf{Q}_c \end{bmatrix} \mathbf{C}^{-1} \begin{bmatrix} \mathbf{I} \\ -\mathbf{D}_c \end{bmatrix},
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{C} &= \frac{1}{2} \mathbf{R}^\top \mathbf{L} - \frac{1}{\tau} \mathbf{I} = \frac{1}{2} \begin{bmatrix} \mathbf{Q}_c^\top \\ -\mathbf{G}_c^\top \end{bmatrix} \begin{bmatrix} \mathbf{G}_c & \mathbf{Q}_c \end{bmatrix} - \frac{1}{\tau} \mathbf{I} \\
&= \frac{1}{2} \begin{bmatrix} \mathbf{Q}_c^\top \mathbf{G}_c & \mathbf{Q}_c^\top \mathbf{Q}_c \\ -\mathbf{G}_c^\top \mathbf{G}_c & -\mathbf{G}_c^\top \mathbf{Q}_c \end{bmatrix} - \frac{1}{\tau} \mathbf{I} = \frac{1}{2} \begin{bmatrix} \mathbf{D}_c - \frac{2}{\tau} \mathbf{I} & \mathbf{I} \\ -\mathbf{D}_c^g & -\mathbf{D}_c - \frac{2}{\tau} \mathbf{I} \end{bmatrix} \quad (3.18)
\end{aligned}$$

is a  $2l \times 2l$  matrix, which is much smaller than  $\mathbf{M}$ . Denoting

$$\mathbf{C}^{-1} = \begin{bmatrix} \mathbf{C}_{11}^{-1} & \mathbf{C}_{12}^{-1} \\ \mathbf{C}_{21}^{-1} & \mathbf{C}_{22}^{-1} \end{bmatrix}, \quad (3.19)$$



eq. (3.17) becomes

$$\begin{aligned} \mathbf{Q}_+ &= \mathbf{Q}_c - \begin{bmatrix} \mathbf{G}_c & \mathbf{Q}_c \end{bmatrix} \begin{bmatrix} \mathbf{C}_{11}^{-1} & \mathbf{C}_{12}^{-1} \\ \mathbf{C}_{21}^{-1} & \mathbf{C}_{22}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} \\ -\mathbf{D}_c \end{bmatrix} \\ &= \mathbf{Q}_c - \mathbf{G}_c \mathbf{C}_{11}^{-1} + \mathbf{G}_c \mathbf{C}_{12}^{-1} \mathbf{D}_c - \mathbf{Q}_c \mathbf{C}_{21}^{-1} + \mathbf{Q}_c \mathbf{C}_{22}^{-1} \mathbf{D}_c \\ &= \mathbf{Q}_c (\mathbf{C}_{22}^{-1} \mathbf{D}_c - \mathbf{C}_{21}^{-1} + \mathbf{I}) + \mathbf{G}_c (\mathbf{C}_{12}^{-1} \mathbf{D}_c - \mathbf{C}_{11}^{-1}) \\ &= \mathbf{Q}_c \mathbf{F}_c + \mathbf{G}_c \mathbf{F}_c^g, \end{aligned} \quad (3.20)$$

where

$$\mathbf{F}_c = \mathbf{C}_{22}^{-1} \mathbf{D}_c - \mathbf{C}_{21}^{-1} + \mathbf{I}, \quad \mathbf{F}_c^g = \mathbf{C}_{12}^{-1} \mathbf{D}_c - \mathbf{C}_{11}^{-1}. \quad (3.21)$$

Therefore, we can update  $\mathbf{B}_+ = \mathfrak{Q}^\top \mathbf{Q}_+$  as

$$\mathbf{B}_+ = \mathfrak{Q}^\top \mathbf{Q}_+ = \mathfrak{Q}^\top \mathbf{Q}_c \mathbf{F}_c + \mathfrak{Q}^\top \mathbf{G}_c \mathbf{F}_c^g = \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g. \quad (3.22)$$

---

**Algorithm III-5** Wen-Yin Integration (Optimized)

---

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices),  $\mathbf{Q}_{\text{init}}$  (initial guess),  $\tau_0 \geq 0$  (initial step size),  $\beta \in (0, 1)$  (scaling parameter for step size searching),  $\sigma \in (0, 1)$  (parameter for step size searching),  $\eta \in (0, 1)$  (parameter for next step searching),  $\tau_{\max}, \tau_{\min}$  (maximum and minimum predicting step size).

**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{opt}}$ .

- 1: Combine  $\mathfrak{Q} = \begin{bmatrix} \mathbf{Q}_{[1]} & \mathbf{Q}_{[2]} & \cdots & \mathbf{Q}_{[N]} \end{bmatrix}$ .
  - 2: Initialize the current iterate  $\mathbf{Q}_c \leftarrow \mathbf{Q}_{\text{init}}$ .
  - 3: Assign  $\mathbf{B}_c \leftarrow \mathfrak{Q}^\top \mathbf{Q}_c$ ,  $\mathbf{D}_c \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c$ ,  $\mathbf{G}_c \leftarrow \frac{1}{N} \mathfrak{Q} \mathbf{B}_c$ ,  $\mathbf{X}_c \leftarrow \mathbf{G}_c - \mathbf{Q}_c \mathbf{D}_c$ .
  - 4: Initialize  $\tau_g \leftarrow \tau_0$ ,  $\zeta \leftarrow 1$ ,  $\phi \leftarrow \frac{1}{2N} \|\mathbf{B}_c\|_F^2$ .
  - 5: **while** (not convergent) **do**
  - 6:   Assign  $\mathbf{B}_c^g \leftarrow \mathfrak{Q}^\top \mathbf{G}_c$ ,  $\mathbf{D}_c^g \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c^g$ .
  - 7:   Compute  $\mu \leftarrow \text{tr}(\mathbf{D}_c^g) - \|\mathbf{D}_c\|_F^2$ .
  - 8:   **repeat for**  $t = 0, 1, \dots$  **do**
  - 9:     Let  $\tau = \tau_g \beta^t$ . Compute  $\mathbf{C}$  by eq. (3.18) and find its inverse.
  - 10:    Compute  $\mathbf{F}_c$  and  $\mathbf{F}_c^g$  by eq. (3.21).
  - 11:    Assign  $\mathbf{B}_+ \leftarrow \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g$ .
  - 12:    Assign  $\tilde{\phi} \leftarrow \frac{1}{2N} \|\mathbf{B}_+\|_F^2$ .
  - 13:    **until**  $\tilde{\phi} \geq \phi + \tau \sigma \mu$
  - 14:    Update  $\phi \leftarrow \frac{\eta \zeta \phi + \tilde{\phi}}{\eta \zeta + 1}$  and then  $\zeta \leftarrow \eta \zeta + 1$ .
  - 15:    Assign  $\mathbf{D}_+ \leftarrow \frac{1}{N} \mathbf{B}_+^\top \mathbf{B}_+$ ,  $\mathbf{Q}_+ \leftarrow \mathbf{Q}_c \mathbf{F}_c + \mathbf{G}_c \mathbf{F}_c^g$ ,  $\mathbf{G}_+ \leftarrow \frac{1}{N} \mathfrak{Q} \mathbf{B}_+$ ,  $\mathbf{X}_+ \leftarrow \mathbf{G}_+ - \mathbf{Q}_+ \mathbf{D}_+$ .
  - 16:    Compute the differences  $\mathbf{\Delta}_1 = \mathbf{Q}_+ - \mathbf{Q}_c$  and  $\mathbf{\Delta}_2 = \mathbf{X}_+ - \mathbf{X}_c$ .
  - 17:    Update  $\tau_g \leftarrow \max(\min(\tau_{\text{guess}}, \tau_{\max}), \tau_{\min})$ , where
$$\tau_{\text{guess}} = \frac{\text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_1)}{|\text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_2)|} \quad \text{or} \quad \frac{|\text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_2)|}{\text{tr}(\mathbf{\Delta}_2^\top \mathbf{\Delta}_2)}.$$
  - 18:    Update  $\mathbf{Q}_c \leftarrow \mathbf{Q}_+$ ,  $\mathbf{G}_c \leftarrow \mathbf{G}_+$ ,  $\mathbf{X}_c \leftarrow \mathbf{X}_+$ ,  $\mathbf{B}_c \leftarrow \mathbf{B}_+$ ,  $\mathbf{D}_c \leftarrow \mathbf{D}_+$ .
  - 19: **end while**
  - 20: Output  $\overline{\mathbf{Q}}_{\text{opt}} \leftarrow \mathbf{Q}_c$ .
- 

### 3.4 Integration Using the Gramian of $\mathfrak{Q}$

The complexity of the optimized algorithms are  $O(mNl^2 \cdot \#\text{Iter})$  flops, whereas computing  $\mathfrak{B} = \mathfrak{Q}^\top \mathfrak{Q}$  cost  $mN^2l^2$  flops (see Chapter 5 for detail). For the cases with many iteration,

explicitly forming  $\mathfrak{B}$  is worthwhile. Although we need to compute an extra  $Nl \times Nl$  matrix  $\mathfrak{B} = \mathfrak{Q}^\top \mathfrak{Q}$  (with  $2mN^2l^2$  flops), since  $\mathfrak{B}$  does not change in the iteration, we only need to do it once at the beginning. With  $\mathfrak{B}$ , we are able to compute  $\bar{\mathbf{Q}}_{\text{opt}}$  without explicitly forming any  $m \times l$  matrices such as  $\mathbf{Q}_c$ ,  $\mathbf{G}_c$  and  $\mathbf{X}_c$ . We found that  $\mathbf{B}_c^g$  can be computed without  $\mathbf{Q}_c$  and  $\mathbf{G}_c$ .

$$\mathbf{B}_c^g = \mathfrak{Q}^\top \mathbf{G}_c = \frac{1}{N} \mathfrak{Q}^\top \mathfrak{Q} \mathbf{B}_c = \mathfrak{B} \mathbf{B}_c. \quad (3.23)$$

However, we still need to compute  $\mathbf{Q}_+$  when the algorithm converges. The updating of  $\mathbf{Q}_+$  can be rewritten as

$$\mathbf{Q}_+ = \mathbf{Q}_c \mathbf{F}_c + \mathbf{G}_c \mathbf{F}_c^g = \mathbf{Q}_c \mathbf{F}_c + \frac{1}{N} \mathfrak{Q} \mathbf{B}_c \mathbf{F}_c^g = \mathbf{Q}_c \mathbf{F}_c + \mathfrak{Q} \mathbf{E}_c, \quad (3.24)$$

where  $\mathbf{E} = \frac{1}{N} \mathbf{B}_c \mathbf{F}_c^g$ . Assuming

$$\mathbf{Q}_c = \mathbf{Q}_{\text{init}} \tilde{\mathbf{F}}_c + \mathfrak{Q} \tilde{\mathbf{E}}_c, \quad (3.25)$$

the updating of  $\mathbf{Q}_+$  becomes

$$\mathbf{Q}_+ = \mathbf{Q}_c \mathbf{F}_c + \mathfrak{Q} \mathbf{E}_c = \mathbf{Q}_{\text{init}} \tilde{\mathbf{F}}_c \mathbf{F}_c + \mathfrak{Q} \left( \tilde{\mathbf{E}}_c \mathbf{F}_c + \mathbf{E}_c \right). \quad (3.26)$$

Note that  $\tilde{\mathbf{F}}_c = \mathbf{I}$  and  $\tilde{\mathbf{E}}_c = \mathbf{O}$  at the beginning. Hence, we get the recurrence relations

$$\begin{aligned} \tilde{\mathbf{F}}_+ &= \tilde{\mathbf{F}}_c \mathbf{F}_c \\ \tilde{\mathbf{E}}_+ &= \tilde{\mathbf{E}}_c \mathbf{F}_c + \mathbf{E}_c \end{aligned} \quad (3.27)$$

with  $\mathbf{Q}_+ = \mathbf{Q}_{\text{init}} \tilde{\mathbf{F}}_+ + \mathfrak{Q} \tilde{\mathbf{E}}_+$ . With  $\mathfrak{B}$ , we propose the Gramian Kolmogorov-Nagumo Integration ([Algorithm III-6](#)) without forming any  $m \times l$  matrices in the iteration.

---

**Algorithm III-6** Kolmogorov-Nagumo Integration (Gramian)
 

---

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices),  $\mathbf{Q}_{\text{init}}$  (initial guess).

**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{opt}}$ .

- 1: Combine  $\mathfrak{Q} = \begin{bmatrix} \mathbf{Q}_{[1]} & \mathbf{Q}_{[2]} & \cdots & \mathbf{Q}_{[N]} \end{bmatrix}$ .
  - 2: Compute  $\mathfrak{B} = \mathfrak{Q}^\top \mathfrak{Q}$ .
  - 3: Initialize  $\mathbf{B}_c \leftarrow \mathfrak{Q}^\top \mathbf{Q}_{\text{init}}$ ,  $\tilde{\mathbf{F}} \leftarrow \mathbf{I}_l$ ,  $\tilde{\mathbf{E}} \leftarrow \mathbf{O}_{Nl \times l}$ .
  - 4: **while** (not convergent) **do**
  - 5: Assign  $\mathbf{B}_c^g = \frac{1}{N} \mathfrak{B} \mathbf{B}_c$ ,  $\mathbf{D}_c \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c$ ,  $\mathbf{D}_c^g \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c^g$ .
  - 6: Compute  $\mathbf{C} \leftarrow \left( \frac{\mathbf{I}}{2} + \left( \frac{\mathbf{I}}{4} - \mathfrak{E} \right)^{1/2} \right)^{1/2}$ , where  $\mathfrak{E} = \mathbf{D}_c^g - \mathbf{D}_c^2$ .
  - 7: Assign  $\mathbf{F}_c \leftarrow \mathbf{C} - \mathbf{D}_c \mathbf{C}^{-1}$ ,  $\mathbf{F}_c^g \leftarrow \mathbf{C}^{-1}$ ,  $\mathbf{E}_c \leftarrow \frac{1}{N} \mathbf{B}_c \mathbf{F}_c^g$ .
  - 8: Update  $\tilde{\mathbf{F}} \leftarrow \tilde{\mathbf{F}} \mathbf{F}_c$  and  $\tilde{\mathbf{E}} \leftarrow \tilde{\mathbf{E}} \mathbf{F}_c + \mathbf{E}_c$ .
  - 9: Update  $\mathbf{B}_c \leftarrow \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g$ .
  - 10: **end while**
  - 11: Output  $\overline{\mathbf{Q}}_{\text{opt}} \leftarrow \mathbf{Q}_{\text{init}} \tilde{\mathbf{F}} + \mathfrak{Q} \tilde{\mathbf{E}}$ .
- 



Similarly, we also propose the Gramian Wen-Yin Integration ([Algorithm III-7](#)). In the same way, we do not need to compute  $\mathbf{Q}$  and  $\mathbf{X}$  in [Step 6](#) in [Algorithm III-2](#). We parameterize  $\mathbf{\Delta}_1$  and  $\mathbf{\Delta}_2$  as

$$\begin{aligned} \mathbf{\Delta}_1 &= \mathbf{Q}_+ - \mathbf{Q}_c = \mathbf{Q}_c \mathbf{\Phi}_1 + \mathfrak{Q} \mathbf{Y}_1, \\ \mathbf{\Delta}_2 &= \mathbf{X}_+ - \mathbf{X}_c = \mathbf{Q}_c \mathbf{\Phi}_2 + \mathfrak{Q} \mathbf{Y}_2, \end{aligned} \quad (3.28)$$

where

$$\begin{aligned} \mathbf{\Phi}_1 &= \mathbf{F}_c - \mathbf{I}, & \mathbf{Y}_1 &= \mathbf{E}_c, \\ \mathbf{\Phi}_2 &= \mathbf{D}_c - \mathbf{F}_c \mathbf{D}_+, & \mathbf{Y}_2 &= \frac{1}{N} (\mathbf{B}_+ - \mathbf{B}_c) - \mathbf{E}_c \mathbf{D}_+. \end{aligned} \quad (3.29)$$

Hence we may compute [Step 7](#) in [Algorithm III-2](#) using

$$\begin{aligned} \text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_1) &= \text{tr}(\mathbf{\Phi}_1^\top \mathbf{\Phi}_1) + 2 \text{tr}(\mathbf{\Phi}_1^\top \mathbf{B}_c^\top \mathbf{Y}_1) + \text{tr}(\mathbf{Y}_1^\top \mathfrak{B} \mathbf{Y}_1), \\ \text{tr}(\mathbf{\Delta}_2^\top \mathbf{\Delta}_2) &= \text{tr}(\mathbf{\Phi}_2^\top \mathbf{\Phi}_2) + 2 \text{tr}(\mathbf{\Phi}_2^\top \mathbf{B}_c^\top \mathbf{Y}_2) + \text{tr}(\mathbf{Y}_2^\top \mathfrak{B} \mathbf{Y}_2), \\ \text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_2) &= \text{tr}(\mathbf{\Phi}_1^\top \mathbf{\Phi}_2) + \text{tr}(\mathbf{\Phi}_1^\top \mathbf{B}_c^\top \mathbf{Y}_2) + \text{tr}(\mathbf{\Phi}_2^\top \mathbf{B}_c^\top \mathbf{Y}_1) + \text{tr}(\mathbf{Y}_1^\top \mathfrak{B} \mathbf{Y}_2) \end{aligned} \quad (3.30)$$

without explicitly forming  $\mathbf{Q}$  and  $\mathbf{X}$ .

---

**Algorithm III-7** Wen-Yin Integration (Gramian)

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices),  $\mathbf{Q}_{\text{init}}$  (initial guess),  $\tau_0 \geq 0$  (initial step size),  $\beta \in (0, 1)$  (scaling parameter for step size searching),  $\sigma \in (0, 1)$  (parameter for step size searching),  $\eta \in (0, 1)$  (parameter for next step searching),  $\tau_{\max}, \tau_{\min}$  (maximum and minimum predicting step size).

**Ensure:** Integrated orthogonal basis  $\bar{\mathbf{Q}}_{\text{opt}}$ .

- 1: Combine  $\mathfrak{Q} = \begin{bmatrix} \mathbf{Q}_{[1]} & \mathbf{Q}_{[2]} & \cdots & \mathbf{Q}_{[N]} \end{bmatrix}$ .
- 2: Compute  $\mathfrak{B} = \mathfrak{Q}^\top \mathfrak{Q}$ .
- 3: Initialize  $\mathbf{B}_c \leftarrow \mathfrak{Q}^\top \mathbf{Q}_{\text{init}}$ ,  $\tilde{\mathbf{F}} \leftarrow \mathbf{I}_l$ ,  $\tilde{\mathbf{E}} \leftarrow \mathbf{O}_{Nl \times l}$ ,  $\tau_g \leftarrow \tau_0$ ,  $\zeta \leftarrow 1$ ,  $\phi \leftarrow \frac{1}{2N} \|\mathbf{B}_c\|_F^2$ .
- 4: Assign  $\mathbf{D}_c \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c$ .
- 5: **while** (not convergent) **do**
- 6:   Assign  $\mathbf{B}_c^g = \frac{1}{N} \mathfrak{B} \mathbf{B}_c$  and  $\mathbf{D}_c^g \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c^g$ .
- 7:   Compute  $\mu \leftarrow \text{tr}(\mathbf{D}_c^g) - \|\mathbf{D}_c\|_F^2$ .
- 8:   **repeat for**  $t = 0, 1, \dots$  **do**
- 9:     Let  $\tau = \tau_g \beta^t$ . Compute  $\mathbf{C}$  by eq. (3.18) and find its inverse.
- 10:    Compute  $\mathbf{F}_c$  and  $\mathbf{F}_c^g$  by eq. (3.21).
- 11:    Assign  $\mathbf{B}_+ \leftarrow \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g$ .
- 12:    Assign  $\tilde{\phi} \leftarrow \frac{1}{2N} \|\mathbf{B}_+\|_F^2$ .
- 13:    **until**  $\tilde{\phi} \geq \phi + \tau \sigma \mu$
- 14:    Assign  $\mathbf{D}_+ \leftarrow \frac{1}{N} \mathbf{B}_+^\top \mathbf{B}_+$  and  $\mathbf{E}_c \leftarrow \frac{1}{N} \mathbf{B}_c \mathbf{F}_c^g$ .
- 15:    Update  $\phi \leftarrow \frac{\eta \zeta \phi + \tilde{\phi}}{\eta \zeta + 1}$  and then  $\zeta \leftarrow \eta \zeta + 1$ .
- 16:    Update  $\tilde{\mathbf{F}} \leftarrow \tilde{\mathbf{F}} \mathbf{F}_c$  and  $\tilde{\mathbf{E}} \leftarrow \tilde{\mathbf{E}} \mathbf{F}_c + \mathbf{E}_c$ .
- 17:    Update  $\tau_g \leftarrow \max(\min(\tau_{\text{guess}}, \tau_{\max}), \tau_{\min})$  using eq. (3.30), where

$$\tau_{\text{guess}} = \frac{\text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_1)}{|\text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_2)|} \quad \text{or} \quad \frac{|\text{tr}(\mathbf{\Delta}_1^\top \mathbf{\Delta}_2)|}{\text{tr}(\mathbf{\Delta}_2^\top \mathbf{\Delta}_2)}.$$

- 18:   Update  $\mathbf{B}_c \leftarrow \mathbf{B}_+$  and  $\mathbf{D}_c \leftarrow \mathbf{D}_+$ .
  - 19: **end while**
  - 20: Output  $\bar{\mathbf{Q}}_{\text{opt}} \leftarrow \mathbf{Q}_{\text{init}} \tilde{\mathbf{F}} + \mathfrak{Q} \tilde{\mathbf{E}}$ .
-







# Chapter 4

## Parallelism

In this chapter, we use  $P$  as the number of processors, and assume that  $N = P$  for simplicity. For  $N > P$ , one may just handle more matrices in a processor. Note that in the Block-Row/Column Parallelism,  $N$  and  $P$  are independent so that one may use more processors.

We propose three parallelism ideas — Naïve Parallelism, Block-Row Parallelism, and Block-Column Parallelism. The Block-Row/Column Parallelism split the matrix into row/column-blocks. These data structures are similar to the one-dimensional block row/column distribution in ScaLAPACK [7]. These block parallelism ideas are proposed to reduce communication cost. Precisely, the Block-Row Parallelism is targeted to the problems with near-square or square  $\mathbf{A}$ , and the Block-Column Parallelism is specialized for short and fat  $\mathbf{A}$ . All of the block-row/column algorithms are balanced, so that the parallel overheads are limited.

### 4.1 Naïve Parallelism

Sketching and orthogonalization can be easily parallelized by using  $N$  processors — computes  $\mathbf{\Omega}_{[i]}$ ,  $\mathbf{Y}_{[i]}$ ,  $\mathbf{Q}_{[i]}$  in the  $i$ -th processor — without any communication. Also, the Kolmogorov-Nagumo Integration ([Algorithm III-1](#)) can also be easily parallelized using the same idea.

---

**Algorithm III-8** Kolmogorov-Nagumo Integration (Naïve Parallelism)

---

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}, \mathbf{Q}_{\text{init}}$ .**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{opt}}$ .

- 1: Initialize the current iterate  $\mathbf{Q}_c \leftarrow \mathbf{Q}_{\text{init}}$ .
  - 2: **while** (not convergent) **do**
  - 3: Assign  $\mathbf{B}_{[i]} \leftarrow \mathbf{Q}_{[i]}^\top \mathbf{Q}_c$  in processor  $i$ .
  - 4: Assign  $\mathbf{X}_{[i]} \leftarrow \frac{1}{N} \left( \mathbf{Q}_{[i]} \mathbf{B}_{[i]} - \mathbf{Q}_c \mathbf{B}_{[i]}^\top \mathbf{B}_{[i]} \right)$  in processor  $i$ .
  - 5: Sum  $\mathbf{X} \leftarrow \sum_{i=1}^N \mathbf{X}_{[i]}$  to all processors (MPI\_Allreduce).
  - 6: Compute  $\mathbf{C} \leftarrow \left( \frac{\mathbf{I}}{2} + \left( \frac{\mathbf{I}}{4} - \mathbf{X}^\top \mathbf{X} \right)^{1/2} \right)^{1/2}$ .
  - 7: Update  $\mathbf{Q}_c \leftarrow \mathbf{Q}_c \mathbf{C} + \mathbf{X} \mathbf{C}^{-1}$ .
  - 8: **end while**
  - 9: Output  $\overline{\mathbf{Q}}_{\text{opt}} \leftarrow \mathbf{Q}_c$ .
- 



Similarly, the Hierarchical Reduction Integration ([Algorithm III-3](#)) can be implemented as follow. However, the algorithm is unbalanced due to the reduction structure.

---

**Algorithm III-9** Hierarchical Reduction Integration (Naïve Parallelism)

---


**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices).**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{hr}}$ .

- 1: Set  $\tilde{N} \leftarrow N$ .
  - 2: **while**  $\tilde{N} > 1$  **do**
  - 3: Set  $h \leftarrow \left\lfloor \frac{\tilde{N}}{2} \right\rfloor$
  - 4: Transfer  $\mathbf{Q}_{[i+h]}$  from processor  $i + h$  to processor  $i$  (MPI\_Send and MPI\_Receive).
  - 5: **if** the processor ID  $i \leq h$  **then**
  - 6: Compute  $\mathbf{W} \mathbf{S} \mathbf{T}^\top \leftarrow \text{svd} \left( \mathbf{Q}_{[i]}^\top \mathbf{Q}_{[i+h]} \right)$  in processor  $i$ .
  - 7: Update  $\mathbf{Q}_{[i]} \leftarrow \left( \mathbf{Q}_{[i]} \mathbf{W} + \mathbf{Q}_{[i+h]} \mathbf{T} \right) \left( 2(\mathbf{I} + \mathbf{S}) \right)^{-1/2}$  in processor  $i$ .
  - 8: **end if**
  - 9: Update  $\tilde{N} \leftarrow \left\lfloor \frac{\tilde{N}}{2} \right\rfloor$ .
  - 10: **end while**
  - 11: Output  $\overline{\mathbf{Q}}_{\text{hr}} \leftarrow \mathbf{Q}_{[1]}$  in processor 1.
- 

## 4.2 Block-Row Parallelism for Integration

We proposed block-row algorithms for lower communication cost. The Naïve Parallelism ([Algorithm III-8](#)) requires an  $m \times l$  data communication ([Step 5](#)) in each iteration. It will be

very slow when  $m$  is huge. Therefore, we propose a new algorithm base on the following idea

$$\begin{bmatrix} \mathcal{N}^{(1)} & \mathcal{N}^{(2)} & \dots & \mathcal{N}^{(P)} \end{bmatrix} \begin{bmatrix} \mathfrak{r}^{(1)} \\ \mathfrak{r}^{(2)} \\ \vdots \\ \mathfrak{r}^{(P)} \end{bmatrix} = \mathcal{N}^{(1)}\mathfrak{r}^{(1)} + \mathcal{N}^{(2)}\mathfrak{r}^{(2)} + \dots + \mathcal{N}^{(P)}\mathfrak{r}^{(P)}. \quad (4.1)$$


In [Algorithm III-10](#) (Block-Row Parallelism of the Optimized Kolmogorov-Nagumo Integration, [Algorithm III-4](#)), we split the matrices into  $m_b \times l$  row-blocks ( $m_b = m/P$ ) and host them in different processors.

$$\mathfrak{Q} = \begin{bmatrix} \mathfrak{Q}^{(1)} \\ \mathfrak{Q}^{(2)} \\ \vdots \\ \mathfrak{Q}^{(P)} \end{bmatrix}, \quad \mathbf{Q}_c = \begin{bmatrix} Q_c^{(1)} \\ Q_c^{(2)} \\ \vdots \\ Q_c^{(P)} \end{bmatrix}, \quad \mathbf{G}_c = \begin{bmatrix} G_c^{(1)} \\ G_c^{(2)} \\ \vdots \\ G_c^{(P)} \end{bmatrix}. \quad (4.2)$$

Therefore,  $\mathbf{B}_c$  and  $\mathbf{B}_c^g$  can be computed as

$$\mathbf{B}_c = \mathfrak{Q}^\top \mathbf{Q}_c = \begin{bmatrix} \mathfrak{Q}^{(1)\top} & \mathfrak{Q}^{(2)\top} & \dots & \mathfrak{Q}^{(P)\top} \end{bmatrix} \begin{bmatrix} Q_c^{(1)} \\ Q_c^{(2)} \\ \vdots \\ Q_c^{(P)} \end{bmatrix} = \sum_{j=1}^P \mathfrak{Q}^{(j)\top} Q_c^{(j)}, \quad (4.3)$$

$$\mathbf{B}_c^g = \mathfrak{Q}^\top \mathbf{G}_c = \begin{bmatrix} \mathfrak{Q}^{(1)\top} & \mathfrak{Q}^{(2)\top} & \dots & \mathfrak{Q}^{(P)\top} \end{bmatrix} \begin{bmatrix} G_c^{(1)} \\ G_c^{(2)} \\ \vdots \\ G_c^{(P)} \end{bmatrix} = \sum_{j=1}^P \mathfrak{Q}^{(j)\top} G_c^{(j)}.$$

---

**Algorithm III-10** Kolmogorov-Nagumo Integration (Block-Row Parallelism)

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}, \mathbf{Q}_{\text{init}}$  (real  $m \times l$  orthogonal matrices),  $P$  (number of processors).

**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{opt}}$ .

- 1: Rearrange  $\mathbf{Q}_{[i]}$  to  $\mathbf{Q}^{(j)}$  in processor  $j$  (MPI\_Alltoall).
  - 2: Initialize the current iterate  $\mathbf{Q}_c^{(j)} \leftarrow \mathbf{Q}_{\text{init}}^{(j)}$  in processors  $j$ .
  - 3: Sum  $\mathbf{B}_c \leftarrow \sum_{j=1}^P \mathbf{Q}^{(j)\top} \mathbf{Q}_c^{(j)}$  to all processors (MPI\_Allreduce).
  - 4: **while** (not convergent) **do**
  - 5:   Assign  $\mathbf{G}_c^{(j)} \leftarrow \frac{1}{N} \mathbf{Q}^{(j)} \mathbf{B}_c$  in processor  $j$ .
  - 6:   Sum  $\mathbf{B}_c^g \leftarrow \sum_{j=1}^P \mathbf{Q}^{(j)\top} \mathbf{G}_c^{(j)}$  to all processors (MPI\_Allreduce).
  - 7:   Assign  $\mathbf{D}_c \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c$  and  $\mathbf{D}_c^g \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c^g$ .
  - 8:   Compute  $\mathbf{C} \leftarrow \left( \frac{\mathbf{I}}{2} + \left( \frac{\mathbf{I}}{4} - \mathbf{\Xi} \right)^{1/2} \right)^{1/2}$ , where  $\mathbf{\Xi} \leftarrow \mathbf{D}_c^g - \mathbf{D}_c^2$ .
  - 9:   Assign  $\mathbf{F}_c \leftarrow \mathbf{C} - \mathbf{D}_c \mathbf{C}^{-1}$  and  $\mathbf{F}_c^g \leftarrow \mathbf{C}^{-1}$ .
  - 10:   Update  $\mathbf{Q}_c^{(j)} \leftarrow \mathbf{Q}_c^{(j)} \mathbf{F}_c + \mathbf{G}_c^{(j)} \mathbf{F}_c^g$  and  $\mathbf{B}_c \leftarrow \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g$  in processor  $j$ .
  - 11: **end while**
  - 12: Gather  $\overline{\mathbf{Q}}_{\text{opt}} \leftarrow \left[ \mathbf{Q}_c^{(1)\top} \quad \mathbf{Q}_c^{(2)\top} \quad \dots \quad \mathbf{Q}_c^{(P)\top} \right]^\top$  (MPI\_Gather).
- 



Similarly, we propose [Algorithm III-11](#) (Block-Row Parallelism of the Optimized Wen-Yin Integration, [Algorithm III-5](#)) using the block-row idea by splitting the following matrices.

$$\mathbf{X}_c = \begin{bmatrix} \mathbf{X}_c^{(1)} \\ \mathbf{X}_c^{(2)} \\ \vdots \\ \mathbf{X}_c^{(P)} \end{bmatrix}, \quad \mathbf{\Delta}_1 = \begin{bmatrix} \mathbf{\Delta}_1^{(1)} \\ \mathbf{\Delta}_1^{(2)} \\ \vdots \\ \mathbf{\Delta}_1^{(P)} \end{bmatrix}, \quad \mathbf{\Delta}_2 = \begin{bmatrix} \mathbf{\Delta}_1^{(1)} \\ \mathbf{\Delta}_1^{(2)} \\ \vdots \\ \mathbf{\Delta}_2^{(P)} \end{bmatrix}. \quad (4.4)$$

---

**Algorithm III-11** Wen-Yin Integration (Block-Row Parallelism)

---

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices),  $\mathbf{Q}_{\text{init}}$  (initial guess),  $\tau_0 \geq 0$  (initial step size),  $\beta \in (0, 1)$  (scaling parameter for step size searching),  $\sigma \in (0, 1)$  (parameter for step size searching),  $\eta \in (0, 1)$  (parameter for next step searching),  $\tau_{\max}, \tau_{\min}$  (maximum and minimum predicting step size),  $P$  (number of processors).

**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{opt}}$ .

- 1: Rearrange  $\mathbf{Q}_{[j]}$  to  $\mathbf{\Omega}^{(j)}$  in processor  $j$  (MPI\_Alltoall).
  - 2: Initialize the current iterate  $\mathbf{Q}_c^{(j)} \leftarrow \mathbf{Q}_{\text{init}}^{(j)}$  in processors  $j$ .
  - 3: Sum  $\mathbf{B}_c \leftarrow \sum_{j=1}^P \mathbf{\Omega}^{(j)\top} \mathbf{Q}_c^{(j)}$  to all processors (MPI\_Allreduce).
  - 4: Assign  $\mathbf{D}_c \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c$ ,  $\mathbf{G}_c^{(j)} \leftarrow \frac{1}{N} \mathbf{\Omega}^{(j)} \mathbf{B}_c$ , and  $\mathbf{X}_c^{(j)} \leftarrow \mathbf{G}_c^{(j)} - \mathbf{Q}_c^{(j)} \mathbf{D}_c$  in processor  $j$ .
  - 5: Initialize  $\tau_g \leftarrow \tau_0$ ,  $\zeta \leftarrow 1$ ,  $\phi \leftarrow \frac{1}{2N} \|\mathbf{B}_c\|_F^2$ .
  - 6: **while** (not convergent) **do**
  - 7: Sum  $\mathbf{B}_c^g \leftarrow \sum_{j=1}^P \mathbf{\Omega}^{(j)\top} \mathbf{G}_c^{(j)}$  to all processors (MPI\_Allreduce).
  - 8: Assign  $\mathbf{D}_c^g \leftarrow \frac{1}{N} \mathbf{B}_c^\top \mathbf{B}_c^g$ .
  - 9: Compute  $\mu \leftarrow \text{tr}(\mathbf{D}_c^g) - \|\mathbf{D}_c\|_F^2$ .
  - 10: **repeat for**  $t = 0, 1, \dots$  **do**
  - 11: Let  $\tau = \tau_g \beta^t$ . Compute  $\mathbf{C}$  by eq. (3.18) and find its inverse.
  - 12: Compute  $\mathbf{F}_c$  and  $\mathbf{F}_c^g$  by eq. (3.21).
  - 13: Assign  $\mathbf{B}_+ \leftarrow \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g$ .
  - 14: Assign  $\tilde{\phi} \leftarrow \frac{1}{2N} \|\mathbf{B}_+\|_F^2$ .
  - 15: **until**  $\tilde{\phi} \geq \phi + \tau \sigma \mu$
  - 16: Update  $\phi \leftarrow \frac{\eta \zeta \phi + \tilde{\phi}}{\eta \zeta + 1}$  and then  $\zeta \leftarrow \eta \zeta + 1$ .
  - 17: Assign  $\mathbf{D}_+ \leftarrow \frac{1}{N} \mathbf{B}_+^\top \mathbf{B}_+$ ,  $\mathbf{Q}_+^{(j)} \leftarrow \mathbf{Q}_c^{(j)} \mathbf{F}_c + \mathbf{G}_c^{(j)} \mathbf{F}_c^g$ ,  $\mathbf{G}_+^{(j)} \leftarrow \frac{1}{N} \mathbf{\Omega}^{(j)} \mathbf{B}_+$ ,  $\mathbf{X}_+^{(j)} \leftarrow \mathbf{G}_+^{(j)} - \mathbf{Q}_+^{(j)} \mathbf{D}_+$  in processor  $j$ .
  - 18: Compute the differences  $\Delta_1^{(j)} = \mathbf{Q}_+^{(j)} - \mathbf{Q}_c^{(j)}$  and  $\Delta_2^{(j)} = \mathbf{X}_+^{(j)} - \mathbf{X}_c^{(j)}$  in processor  $j$ .
  - 19: Sum  $t_{11} \leftarrow \sum_{j=1}^P \text{tr}(\Delta_1^{(j)\top} \Delta_1^{(j)})$ ,  $t_{12} \leftarrow \sum_{j=1}^P \text{tr}(\Delta_1^{(j)\top} \Delta_2^{(j)})$ , and  $t_{22} \leftarrow \sum_{j=1}^P \text{tr}(\Delta_2^{(j)\top} \Delta_2^{(j)})$  to all processors (MPI\_Allreduce).
  - 20: Update  $\tau_g \leftarrow \max(\min(\tau_{\text{guess}}, \tau_{\max}), \tau_{\min})$ , where
$$\tau_{\text{guess}} = \frac{t_{11}}{t_{12}} \quad \text{or} \quad \frac{t_{12}}{t_{22}}.$$
  - 21: Update  $\mathbf{Q}_c^{(j)} \leftarrow \mathbf{Q}_+^{(j)}$ ,  $\mathbf{G}_c^{(j)} \leftarrow \mathbf{G}_+^{(j)}$ ,  $\mathbf{X}_c^{(j)} \leftarrow \mathbf{X}_+^{(j)}$ ,  $\mathbf{B}_c \leftarrow \mathbf{B}_+$ ,  $\mathbf{D}_c \leftarrow \mathbf{D}_+$  in processor  $j$ .
  - 22: **end while**
  - 23: Gather  $\overline{\mathbf{Q}}_{\text{opt}} \leftarrow [\mathbf{Q}_c^{(1)\top} \quad \mathbf{Q}_c^{(2)\top} \quad \dots \quad \mathbf{Q}_c^{(P)\top}]^\top$  (MPI\_Gather).
-

Algorithm III-12 (Block-Row Parallelism of the Hierarchical Reduction Integration, Algorithm III-3) also use the block-row idea. Unlike the Naïve Parallelism (Algorithm III-9), this algorithm is balanced.

---

**Algorithm III-12** Hierarchical Reduction Integration (Block-Row Parallelism)

---

**Require:**  $\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}, \dots, \mathbf{Q}_{[N]}$  (real  $m \times l$  orthogonal matrices),  $P$  (number of processors).

**Ensure:** Integrated orthogonal basis  $\overline{\mathbf{Q}}_{\text{hr}}$ .

- 1: Rearrange  $\mathbf{Q}_{[i]}$  to  $\mathfrak{Q}^{(j)}$  in processor  $j$  (MPI\_Alltoall).
  - 2: Set  $\tilde{N} \leftarrow N$ .
  - 3: **while**  $\tilde{N} > 1$  **do**
  - 4:   Set  $h \leftarrow \lfloor \frac{\tilde{N}}{2} \rfloor$
  - 5:   **for**  $t = 1$  **to**  $h$  **do**
  - 6:     Sum  $\mathbf{B}_{[i, i+h]} \leftarrow \sum_{j=1}^P \mathbf{Q}_{[i]}^{(j)\top} \mathbf{Q}_{[i+h]}^{(j)}$  to all processors (MPI\_Allreduce).
  - 7:     Compute  $\mathbf{WST}^\top \leftarrow \text{svd}(\mathbf{B}_{[i, i+h]})$ .
  - 8:     Update  $\mathbf{Q}_{[i]}^{(j)} \leftarrow \left( \mathbf{Q}_{[i]}^{(j)} \mathbf{W} + \mathbf{Q}_{[i+h]}^{(j)} \mathbf{T} \right) (2(\mathbf{I} + \mathbf{S}))^{-1/2}$  in processor  $j$ .
  - 9:   **end for**
  - 10:   Update  $\tilde{N} \leftarrow \lfloor \frac{\tilde{N}}{2} \rfloor$ .
  - 11: **end while**
  - 12: Output  $\overline{\mathbf{Q}}_{\text{hr}} \leftarrow \left[ \mathbf{Q}_{[1]}^{(1)\top} \quad \mathbf{Q}_{[1]}^{(2)\top} \quad \dots \quad \mathbf{Q}_{[1]}^{(P)\top} \right]^\top$  (MPI\_Gather).
- 

### 4.3 Block-Row Parallelism for Gramian Integration

With the improvement on the Gramian integration (Algorithm III-6 and Algorithm III-7), we only need to handle tiny matrices (with size  $l$  or  $Nl$ ) in the iteration, which cost  $O(N^2 l^3) \cdot \#\text{Iter}$  flops only. Since  $l$  is extremely small, the iteration does not need parallelism. Therefore, we focus computing  $\mathfrak{B} = \mathfrak{Q}^\top \mathfrak{Q}$ . To reduce data communication, we use the block-row idea. Define

$$\mathfrak{Q} = \begin{bmatrix} \mathfrak{Q}^{(1)} \\ \mathfrak{Q}^{(2)} \\ \vdots \\ \mathfrak{Q}^{(P)} \end{bmatrix} \quad (4.5)$$

and compute  $\mathfrak{B}$  using

$$\mathfrak{B} = \sum_{j=1}^P \mathfrak{Q}^{(j)\top} \mathfrak{Q}^{(j)}. \quad (4.6)$$

We proposed several subroutines by splitting

$$\mathbf{Q}_{\text{init}} = \begin{bmatrix} Q_{\text{init}}^{(1)} \\ Q_{\text{init}}^{(2)} \\ \vdots \\ Q_{\text{init}}^{(P)} \end{bmatrix}, \quad \bar{\mathbf{Q}}_{\text{opt}} = \begin{bmatrix} \bar{Q}_{\text{opt}}^{(1)} \\ \bar{Q}_{\text{opt}}^{(2)} \\ \vdots \\ \bar{Q}_{\text{opt}}^{(P)} \end{bmatrix}. \quad (4.7)$$



Subroutines S-1 to S-3 are used before the iteration, and Subroutines S-4 to S-5 are used after the iteration. Furthermore, if we pick the  $\mathbf{Q}_{[1]}$  as the initial guess (i.e.  $\mathbf{Q}_{\text{init}} = \mathbf{Q}_{[1]}$ ), we can skip Subroutine S-3 and copy  $\mathbf{B}_c$  from the first column-block of  $\mathfrak{B}$ .

---

**Subroutine S-1** Matrices Rearrangement (Block-Row Parallelism)

---

**Require:**  $\mathbf{Q}_{[i]}$ ,  $\mathbf{Q}_{\text{init}}$ ,  $P$  (number of processors).

**Ensure:**  $\mathfrak{Q}^{(j)}$ ,  $Q_{\text{init}}^{(j)}$ .

- 1: Split the  $\mathbf{Q}_{[i]}$  into  $P$  row-blocks and send  $Q_{[i]}^{(j)}$  to processor  $j$  (MPI\_Alltoall).
  - 2: Combine  $\mathfrak{Q}^{(j)} = \begin{bmatrix} Q_{[1]}^{(j)} & Q_{[2]}^{(j)} & \cdots & Q_{[N]}^{(j)} \end{bmatrix}$  in processor  $j$ .
  - 3: Split the  $\mathbf{Q}_{\text{init}}$  into  $P$  row-blocks and send  $Q_{\text{init}}^{(j)}$  to processor  $j$  (MPI\_Scatter).
- 

---

**Subroutine S-2** Computing  $\mathfrak{B} = \mathfrak{Q}^\top \mathfrak{Q}$  (Block-Row Parallelism)

---

**Require:**  $\mathfrak{Q}^{(j)}$ .

**Ensure:**  $\mathfrak{B}$ .

- 1: Sum  $\mathfrak{B} \leftarrow \sum_{j=1}^P \mathfrak{Q}^{(j)\top} \mathfrak{Q}^{(j)}$  to all processors (MPI\_Allreduce).
- 

---

**Subroutine S-3** Computing  $\mathbf{B}_c = \mathfrak{Q}^\top \mathbf{Q}_{\text{init}}$  (Block-Row Parallelism)

---

**Require:**  $\mathfrak{Q}^{(j)}$ ,  $Q_{\text{init}}^{(j)}$ .

**Ensure:**  $\mathbf{B}_c$ .

- 1: Sum  $\mathbf{B}_c \leftarrow \sum_{j=1}^P \mathfrak{Q}^{(j)\top} Q_{\text{init}}^{(j)}$  to all processors (MPI\_Allreduce).
- 

---

**Subroutine S-4** Computing  $\bar{\mathbf{Q}}_{\text{opt}} = \mathbf{Q}_{\text{init}} \tilde{\mathbf{F}} + \mathfrak{Q} \tilde{\mathbf{E}}$  (Block-Row Parallelism)

---

**Require:**  $\mathfrak{Q}^{(j)}$ ,  $Q_{\text{init}}^{(j)}$ .

**Ensure:**  $\bar{Q}_{\text{opt}}^{(j)}$ .

- 1: Assign  $\bar{Q}_{\text{opt}}^{(j)} \leftarrow Q_{\text{init}}^{(j)} \tilde{\mathbf{F}} + \mathfrak{Q}^{(j)} \tilde{\mathbf{E}}$ .
-

---

**Subroutine S-5** Matrices Gathering (Block-Row Parallelism)

---

**Require:**  $\overline{Q}_{\text{opt}}^{(j)}$ .**Ensure:**  $\overline{Q}_{\text{opt}}$ .1: Gather  $\overline{Q}_{\text{opt}} \leftarrow \left[ \overline{Q}_{\text{opt}}^{(1)\top} \quad \overline{Q}_{\text{opt}}^{(2)\top} \quad \dots \quad \overline{Q}_{\text{opt}}^{(P)\top} \right]^\top$  (MPI\_Gather or MPI\_Allgather).

## 4.4 Block-Row Parallelism for Sketching, Orthogonalization, and Postprocessing

The block-row idea can be also used on sketching, orthogonalization, and postprocessing. With the idea, the rearrangement (Subroutine S-1) and the gathering (Subroutine S-5) are unnecessary. Also, splitting the input matrix  $\mathbf{A}$  can reduce memory usage so that we can handle larger problems. We split The following matrices into  $m_b \times n$  and  $m_b \times l$  row-blocks

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}^{(1)} \\ \mathbf{A}^{(2)} \\ \vdots \\ \mathbf{A}^{(P)} \end{bmatrix}, \quad \mathbf{Y}_{[i]} = \begin{bmatrix} Y_{[i]}^{(1)} \\ Y_{[i]}^{(2)} \\ \vdots \\ Y_{[i]}^{(P)} \end{bmatrix}, \quad \mathbf{Q}_{[i]} = \begin{bmatrix} Q_{[i]}^{(1)} \\ Q_{[i]}^{(2)} \\ \vdots \\ Q_{[i]}^{(P)} \end{bmatrix}, \quad \overline{\mathbf{Q}} = \begin{bmatrix} \overline{Q}^{(1)} \\ \overline{Q}^{(2)} \\ \vdots \\ \overline{Q}^{(P)} \end{bmatrix}. \quad (4.8)$$

For sketching, we split  $\mathbf{A}$  to row-blocks  $\mathbf{A}^{(j)}$  and computes  $Y_{[i]}^{(j)}$  in the  $j$ -th processors.

The only communication is sending the random seed to all processors (with  $q = 0$ ).



---

**Algorithm I-2** Gaussian Projection Sketching (Block-Row Parallelism)

---

**Require:**  $A^{(j)}$  (real  $m_b \times n$  matrix, row-block),  $l$  (dimension of the sketched column space),  $q$  (exponent of the power method),  $N$  (number of random sketches),  $P$  (number of processors).

**Ensure:**  $Y_{[i]}^{(j)}$  (real  $m_b \times l$  matrices, row-blocks), where columns of  $Y_{[i]}^{(j)}$  spans a column subspace of  $\mathbf{A}$  for  $i = 1, \dots, N$ .

- 1: Broadcast the random seed to all processors (MPI\_Bcast).
  - 2: Generate  $n \times l$  random matrices  $\Omega_{[i]}$  using Gaussian normal distribution in all processors.
  - 3: Assign  $Y_{[i]}^{(j)} \leftarrow A^{(j)} \Omega_{[i]}$  in processor  $j$ .
  - 4: **loop**  $q$  times **do**
  - 5: Sum  $\tilde{Y}_{[i]} \leftarrow \sum_{j=1}^P A^{(j)\top} Y_{[i]}^{(j)}$  to all processors (MPI\_Allreduce).
  - 6: Update  $Y_{[i]}^{(j)} \leftarrow A^{(j)} \tilde{Y}_{[i]}$  in processor  $j$ .
  - 7: **end loop**
- 

We also use the block-row idea on Gramian orthogonalization ([Algorithm II-2](#)). Here we only need to communicate  $l \times l$  matrices.

---

**Algorithm II-3** Gramian Orthogonalization (Block-Row Parallelism)

---

**Require:**  $Y_{[i]}^{(j)}$  (real  $m_b \times l$  matrices, row-blocks),  $P$  (number of processors).

**Ensure:**  $Q_{[i]}^{(j)}$  (real  $m_b \times l$  matrices, row-blocks), where columns of  $Q_{[i]}^{(j)}$  are an orthonormal basis of  $Y_{[i]}^{(j)}$ .

- 1: Sum  $\mathbf{M}_{[i]} \leftarrow \sum_{j=1}^P Y_{[i]}^{(j)\top} Y_{[i]}^{(j)}$  to all processors (MPI\_Allreduce).
  - 2: Compute  $\mathbf{W}_{[i]} \mathbf{S}_{[i]}^2 \mathbf{W}_{[i]}^\top \leftarrow \text{eig}(\mathbf{M}_{[i]})$ .
  - 3: Assign  $Q_{[i]}^{(j)} \leftarrow Y_{[i]}^{(j)} \mathbf{W}_{[i]} \mathbf{S}_{[i]}^{-1}$  in processor  $j$ .
- 

To parallel the Gramian Postprocessing ([Algorithm IV-2](#)) and the symmetric postprocessing ([Algorithm IV-3](#)), we split  $\mathbf{Z} = \mathbf{A}^\top \bar{\mathbf{Q}}$  into  $n_b \times l$  row-blocks

$$\mathbf{Z} = \begin{bmatrix} Z^{(1)} \\ Z^{(2)} \\ \vdots \\ Z^{(P)} \end{bmatrix} \quad (4.9)$$

with an extra communication using MPI\_Reduce\_scatter\_block.

---

**Algorithm IV-4** Gramian Postprocessing (Block-Row Parallelism)

---

**Require:**  $A^{(j)}$  (real  $m_b \times n$  matrix, row-block),  $\overline{Q}^{(j)}$  (real  $m_b \times l$  orthogonal matrix, row-block),  $k$  (desired rank of approximate SVD),  $P$  (number of processors).

**Ensure:** Approximate rank- $k$  SVD of  $\mathbf{A} \approx \widehat{\mathbf{U}}_k \widehat{\Sigma}_k \widehat{\mathbf{V}}_k^\top$ .

- 1: Compute  $A^{(j)\top} \overline{Q}^{(j)}$  in each processor and sum the  $j$ -th row-blocks to  $Z^{(j)}$  in processor  $j$  (MPI\_Reduce\_scatter\_block).
  - 2: Sum  $\mathbf{M} \leftarrow \sum_{j=1}^P Z^{(j)\top} Z^{(j)}$  to all processors (MPI\_Allreduce).
  - 3: Compute  $\widehat{\mathbf{W}}_l \widehat{\Sigma}_l^2 \widehat{\mathbf{W}}_l^\top \leftarrow \text{eig}(\mathbf{M})$ .
  - 4: Extract the largest  $k$  eigen-pairs from  $\widehat{\mathbf{W}}_l, \widehat{\Sigma}_l$  to obtain  $\widehat{\mathbf{W}}_k, \widehat{\Sigma}_k$ .
  - 5: Assign  $\widehat{\mathbf{U}}_k^{(j)} \leftarrow \overline{Q}^{(j)} \widehat{\mathbf{W}}_k$ .
  - 6: Assign  $\widehat{\mathbf{V}}_k^{(j)} \leftarrow Z^{(j)} \widehat{\mathbf{W}}_k \widehat{\Sigma}_k^{-1}$ .
  - 7: Gather  $\widehat{\mathbf{U}}_k \leftarrow \left[ \widehat{\mathbf{U}}_k^{(1)\top} \quad \widehat{\mathbf{U}}_k^{(2)\top} \quad \dots \quad \widehat{\mathbf{U}}_k^{(P)\top} \right]^\top$  (MPI\_Gather).
  - 8: Gather  $\widehat{\mathbf{V}}_k \leftarrow \left[ \widehat{\mathbf{V}}_k^{(1)\top} \quad \widehat{\mathbf{V}}_k^{(2)\top} \quad \dots \quad \widehat{\mathbf{V}}_k^{(P)\top} \right]^\top$  (MPI\_Gather).
- 

---

**Algorithm IV-5** Symmetric Postprocessing (Block-Row Parallelism)

---

**Require:**  $A^{(j)}$  (real  $m_b \times m$  matrix, row-block),  $\overline{Q}^{(j)}$  (real  $m_b \times l$  orthogonal matrix, row-block),  $k$  (desired rank of approximate SVD),  $P$  (number of processors).

**Ensure:** Approximate rank- $k$  SVD of  $\mathbf{A} \approx \widehat{\mathbf{U}}_k \widehat{\Sigma}_k \widehat{\mathbf{U}}_k^\top$ .

- 1: Compute  $A^{(j)\top} \overline{Q}^{(j)}$  in each processor and sum the  $j$ -th row-blocks to  $Z^{(j)}$  in processor  $j$  (MPI\_Reduce\_scatter\_block).
  - 2: Sum  $\mathbf{M} \leftarrow \sum_{j=1}^P Z^{(j)\top} \overline{Q}^{(j)}$  to all processors (MPI\_Allreduce).
  - 3: Compute  $\widehat{\mathbf{W}}_l \widehat{\Sigma}_l \widehat{\mathbf{W}}_l^\top \leftarrow \text{eig}(\mathbf{M})$ .
  - 4: Extract the largest  $k$  eigen-pairs from  $\widehat{\mathbf{W}}_l, \widehat{\Sigma}_l$  to obtain  $\widehat{\mathbf{W}}_k, \widehat{\Sigma}_k$ .
  - 5: Assign  $\widehat{\mathbf{U}}_k^{(j)} \leftarrow \overline{Q}^{(j)} \widehat{\mathbf{W}}_k$ .
  - 6: Gather  $\widehat{\mathbf{U}}_k \leftarrow \left[ \widehat{\mathbf{U}}_k^{(1)\top} \quad \widehat{\mathbf{U}}_k^{(2)\top} \quad \dots \quad \widehat{\mathbf{U}}_k^{(P)\top} \right]^\top$  (MPI\_Gather).
- 

Demmel et al. proposed Block-Row Parallelism based Tall Skinny QR (TSQR) [8] for computing QR decomposition (see Appendix A.2 for detail). Algorithm II-4 is the orthogonalization using TSQR.

Table 4.1: Communication Tree of Tall Skinny QR

Level					
1	1&2	3&4	5&6	7&8	...
2	1&3	2&4	5&7	6&8	...
3	1&5	2&6	3&7	4&8	...
⋮	⋮	⋮	⋮	⋮	⋮




---

**Algorithm II-4** TSQR [8] Orthogonalization (Block-Row Parallelism)
 

---

**Require:**  $Y_{[i]}^{(j)}$  (real  $m_b \times l$  matrix, row-block),  $P$  (number of processors).

**Ensure:**  $Q_{[i]}^{(j)}$  (real  $m_b \times l$  matrices, row-blocks), where columns of  $Q_{[i]}$  are an orthonormal basis of  $Y_{[i]}$ .

- 1: Denote  $H = \lceil \log_2 P \rceil$ .
  - 2: Compute  $Q_{[i],0}^{(j)} R_{[i],0}^{(j)} \leftarrow \text{qr} \left( Y_{[i]}^{(j)} \right)$ , where  $Q_{[i],0}^{(j)} \in \mathbb{R}^{m_b \times l}$  and  $R_{[i],0}^{(j)} \in \mathbb{R}^{l \times l}$ .
  - 3: **for**  $t = 1$  **to**  $H$  **do**
  - 4:   **if** processor  $j$  has neighbor at level  $t$  **then**
  - 5:     Combine  $Y_{[i],t}^{(j)}$  from  $R_{[i],t-1}^{(j)}$  and  $R_{[i],t-1}^{(\check{j})}$ , where  $\check{j}$  denote the neighbor processor (see Table 4.1). (MPI\_Sendrecv).
  - 6:     Compute  $Q_{[i],t}^{(j\check{j})} R_{[i],t}^{(j\check{j})} \leftarrow \text{qr} \left( Y_{[i],t}^{(j\check{j})} \right)$ , where  $Q_{[i],t}^{(j\check{j})} \in \mathbb{R}^{2l \times l}$  and  $R_{[i],t}^{(j\check{j})} \in \mathbb{R}^{l \times l}$ .
  - 7:     Assign  $Q_{[i],t}^{(j)}$  as the correspond row-block of  $Q_{[i],t}^{(j\check{j})}$  (upper/lower block if  $j$  is smaller/larger than  $\check{j}$ ).
  - 8:     Denote  $R_{[i],t}^{(j)} \leftarrow R_{[i],t}^{(j\check{j})}$ .
  - 9:   **else**
  - 10:     Assign  $R_{[i],t}^{(j)} \leftarrow R_{[i],t-1}^{(j)}$  and  $Q_{[i],t} \leftarrow I_l$  (the latter is stored implicitly).
  - 11:   **end if**
  - 12: **end for**
  - 13: Output  $Q_{[i]}^{(j)} \leftarrow Q_{[i],0}^{(j)} Q_{[i],1}^{(j)} \cdots Q_{[i],H}^{(j)}$ .
- 

Also, we can compute the SVD of  $Z = A^T \overline{Q}$  by applying in TSQR and computing the small SVD of the upper triangular part. Using the idea, we propose Algorithm IV-6.

---

**Algorithm IV-6** TSQR [8] Postprocessing (Block-Row Parallelism)

---

**Require:**  $A^{(j)}$  (real  $m_b \times n$  matrix, row-block),  $\overline{Q}^{(j)}$  (real  $m_b \times l$  orthogonal matrix, row-block),  $k$  (desired rank of approximate SVD),  $P$  (number of processors).

**Ensure:** Approximate rank- $k$  SVD of  $\mathbf{A} \approx \widehat{\mathbf{U}}_k \widehat{\mathbf{\Sigma}}_k \widehat{\mathbf{V}}_k^\top$ .

- 1: Denote  $H = \lceil \log_2 P \rceil$ .
  - 2: Compute  $A^{(j)\top} \overline{Q}^{(j)}$  in each processor and sum the  $j$ -th row-blocks to  $Z^{(j)}$  in processor  $j$  (MPI\_Reduce\_scatter\_block).
  - 3: Compute  $V_0^{(j)} R_0^{(j)} \leftarrow \text{qr}(Z^{(j)})$ , where  $V_0^{(j)} \in \mathbb{R}^{m_b \times l}$  and  $R_0^{(j)} \in \mathbb{R}^{l \times l}$ .
  - 4: **for**  $t = 1$  **to**  $H$  **do**
  - 5:   **if** processor  $j$  has neighbor at level  $t$  **then**
  - 6:     Combine  $\tilde{Z}_t$  from  $R_{t-1}^{(j)}$  and  $R_{t-1}^{(\check{j})}$ , where  $\check{j}$  denote the neighbor processor (see Table 4.1). (MPI\_Sendrecv).
  - 7:     Compute  $\tilde{V}_t R_t \leftarrow \text{qr}(\tilde{Z}_t)$ , where  $\tilde{V}_t \in \mathbb{R}^{2l \times l}$  and  $R_t \in \mathbb{R}^{l \times l}$ .
  - 8:     Assign  $V_t^{(j)}$  as the correspond row-block of  $\tilde{V}_t$  (upper/lower block if  $j$  is smaller/larger than  $\check{j}$ ).
  - 9:   **else**
  - 10:     Assign  $R_t \leftarrow R_{t-1}$  and  $V_t \leftarrow I_l$  (the latter is stored implicitly).
  - 11:   **end if**
  - 12: **end for**
  - 13: Compute  $\widehat{\mathbf{W}}_l \widehat{\mathbf{\Sigma}}_l \widehat{\mathbf{T}}_l^\top \leftarrow \text{svd}(R_H^\top)$ .
  - 14: Extract the largest  $k$  singular-pairs from  $\widehat{\mathbf{W}}_l, \widehat{\mathbf{\Sigma}}_l, \widehat{\mathbf{T}}_l$  to obtain  $\widehat{\mathbf{W}}_k, \widehat{\mathbf{\Sigma}}_k, \widehat{\mathbf{T}}_k$ .
  - 15: Assign  $\widehat{\mathbf{U}}_k^{(j)} \leftarrow \overline{Q}^{(j)} \widehat{\mathbf{W}}_k$ .
  - 16: Assign  $\widehat{\mathbf{V}}_k^{(j)} \leftarrow V_0^{(j)} V_1^{(j)} \dots V_H^{(j)} \widehat{\mathbf{T}}_k$ .
  - 17: Gather  $\widehat{\mathbf{U}}_k \leftarrow \left[ \widehat{\mathbf{U}}_k^{(1)\top} \quad \widehat{\mathbf{U}}_k^{(2)\top} \quad \dots \quad \widehat{\mathbf{U}}_k^{(P)\top} \right]^\top$  (MPI\_Gather).
  - 18: Gather  $\widehat{\mathbf{V}}_k \leftarrow \left[ \widehat{\mathbf{V}}_k^{(1)\top} \quad \widehat{\mathbf{V}}_k^{(2)\top} \quad \dots \quad \widehat{\mathbf{V}}_k^{(P)\top} \right]^\top$  (MPI\_Gather).
- 

## 4.5 Block-Column Parallelism

The Block-Column Parallelism is applied only on the input matrix  $\mathbf{A}$  since it is the only short and fat matrix in the algorithm. Therefore, we only apply Block-Column Parallelism on sketching and postprocessing since  $\mathbf{A}$  is only used in this two stages.

For the cases with extremely large  $n$ , Step 1 in Algorithm IV-4 (Block-Row Parallelism of the Gramian Postprocessing) cost  $nl$  words communication. For  $m \ll n$  (more precisely,

$Nm < n$ ), we choose to parallelize the input matrix  $\mathbf{A}$  into column blocks

$$\mathbf{A} = \left[ \mathbf{A}^{(1)} \quad \mathbf{A}^{(2)} \quad \dots \quad \mathbf{A}^{(P)} \right], \quad (4.10)$$

and use `MPI_Reduce_scatter_block` to convert the result into row-blocks. Here [Step 2](#) in [Algorithm I-3](#) only cost  $Nml$  words. (See [Chapter 5](#) for more detail.)




---

**Algorithm I-3** Gaussian Projection Sketching (Block-Column Parallelism)

---

**Require:**  $\mathbf{A}^{(j)}$  (real  $m \times n_b$  matrix, column-block),  $l$  (dimension of the sketched column space),  $N$  (number of random sketches),  $P$  (number of processors).

**Ensure:**  $\mathbf{Y}_{[i]}^{(j)}$  (real  $m_b \times l$  matrices, row-blocks), where columns of  $\mathbf{Y}_{[i]}$  spans a column subspace of  $\mathbf{A}$  for  $i = 1, \dots, N$ .

- 1: Generate  $n_b \times l$  random matrices  $\mathbf{\Omega}_{[i]}^{(j)}$  using Gaussian normal distribution in all processors (with different random seed).
  - 2: Compute  $\mathbf{A}^{(j)}\mathbf{\Omega}_{[i]}^{(j)}$  in each processor and sum the  $j$ -th row-blocks to  $\mathbf{Y}_{[i]}^{(j)}$  in processor  $j$  (`MPI_Reduce_scatter_block`).
- 

With column blocks  $\mathbf{A}^{(j)}$ , [Step 2](#) in [Algorithm IV-7](#) can be done without communication. Through we need to gather  $\overline{\mathbf{Q}}$  (using [Subroutine S-5](#), require  $ml$  words communication) before postprocessing, the communication is still reduced since  $m \ll n$ . Furthermore, with column blocks, if one only needs left singular vectors  $\widehat{\mathbf{U}}_k$ , iSVD can be done without any commutation related to  $n$ .

---

**Algorithm IV-7** Gramian Postprocessing (Block-Column Parallelism)

---

**Require:**  $\mathbf{A}^{(j)}$  (real  $m_b \times n$  matrix, column-block),  $\overline{\mathbf{Q}}^{(j)}$  (real  $m_b \times l$  orthogonal matrix, row-block),  $k$  (desired rank of approximate SVD),  $P$  (number of processors).

**Ensure:** Approximate rank- $k$  SVD of  $\mathbf{A} \approx \widehat{\mathbf{U}}_k \widehat{\mathbf{\Sigma}}_k \widehat{\mathbf{V}}_k^\top$ .

- 1: Gather  $\overline{\mathbf{Q}} \leftarrow \left[ \overline{\mathbf{Q}}^{(1)\top} \quad \overline{\mathbf{Q}}^{(2)\top} \quad \dots \quad \overline{\mathbf{Q}}^{(P)\top} \right]^\top$  to all processors (`MPI_Allgather`).
  - 2: Assign  $\mathbf{Z}^{(j)} \leftarrow \mathbf{A}^{(j)\top} \overline{\mathbf{Q}}$  in processor  $j$ .
  - 3: Sum  $\mathbf{M} \leftarrow \sum_{j=1}^P \mathbf{Z}^{(j)\top} \mathbf{Z}^{(j)}$  to all processors (`MPI_Allreduce`).
  - 4: Compute  $\widehat{\mathbf{W}}_l \widehat{\mathbf{\Sigma}}_l^2 \widehat{\mathbf{W}}_l^\top \leftarrow \text{eig}(\mathbf{M})$ .
  - 5: Extract the largest  $k$  eigen-pairs from  $\widehat{\mathbf{W}}_l$ ,  $\widehat{\mathbf{\Sigma}}_l$  to obtain  $\widehat{\mathbf{W}}_k$ ,  $\widehat{\mathbf{\Sigma}}_k$ .
  - 6: Assign  $\widehat{\mathbf{U}}_k \leftarrow \overline{\mathbf{Q}} \widehat{\mathbf{W}}_k$ .
  - 7: Assign  $\widehat{\mathbf{V}}_k^{(j)} \leftarrow \mathbf{Z}^{(j)} \widehat{\mathbf{W}}_k \widehat{\mathbf{\Sigma}}_k^{-1}$ .
  - 8: Gather  $\widehat{\mathbf{V}}_k \leftarrow \left[ \widehat{\mathbf{V}}_k^{(1)\top} \quad \widehat{\mathbf{V}}_k^{(2)\top} \quad \dots \quad \widehat{\mathbf{V}}_k^{(P)\top} \right]^\top$  (`MPI_Gather`).
-

The TSQR Postprocessing ([Algorithm IV-8](#)) can also use column block  $A^{(j)}$  in the same way as [Algorithm IV-7](#).




---

**Algorithm IV-8** TSQR [[8](#)] Postprocessing (Block-Column Parallelism)

---

**Require:**  $A^{(j)}$  (real  $m_b \times n$  matrix, column-block),  $\overline{Q}^{(j)}$  (real  $m_b \times l$  orthogonal matrix, row-block),  $k$  (desired rank of approximate SVD),  $P$  (number of processors).

**Ensure:** Approximate rank- $k$  SVD of  $\mathbf{A} \approx \widehat{\mathbf{U}}_k \widehat{\mathbf{\Sigma}}_k \widehat{\mathbf{V}}_k^\top$ .

- 1: Denote  $H = \lceil \log_2 P \rceil$ .
  - 2: Gather  $\overline{\mathbf{Q}} \leftarrow \left[ \overline{Q}^{(1)\top} \quad \overline{Q}^{(2)\top} \quad \dots \quad \overline{Q}^{(P)\top} \right]^\top$  to all processors (MPI\_Allgather).
  - 3: Assign  $Z^{(j)} \leftarrow A^{(j)\top} \overline{\mathbf{Q}}$  in processor  $j$ .
  - 4: Compute  $V_0^{(j)} R_0^{(j)} \leftarrow \text{qr}(Z^{(j)})$ , where  $V_0^{(j)} \in \mathbb{R}^{m_b \times l}$  and  $R_0^{(j)} \in \mathbb{R}^{l \times l}$ .
  - 5: **for**  $t = 1$  **to**  $H$  **do**
  - 6:   **if** processor  $j$  has neighbor at level  $t$  **then**
  - 7:     Combine  $\tilde{Z}_t$  from  $R_{t-1}^{(j)}$  and  $R_{t-1}^{(\check{j})}$ , where  $\check{j}$  denote the neighbor processor (see [Table 4.1](#)). (MPI\_Sendrecv).
  - 8:     Compute  $\tilde{V}_t R_t \leftarrow \text{qr}(\tilde{Z}_t)$ , where  $\tilde{V}_t \in \mathbb{R}^{2l \times l}$  and  $R_t \in \mathbb{R}^{l \times l}$ .
  - 9:     Assign  $V_t^{(j)}$  as the correspond row-block of  $\tilde{V}_t$  (upper/lower block if  $j$  is smaller/larger than  $\check{j}$ ).
  - 10:   **else**
  - 11:     Assign  $R_t \leftarrow R_{t-1}$  and  $V_t \leftarrow I_l$  (the latter is stored implicitly).
  - 12:   **end if**
  - 13: **end for**
  - 14: Compute  $\widehat{\mathbf{W}}_l \widehat{\mathbf{\Sigma}}_l \widehat{\mathbf{T}}_l^\top \leftarrow \text{svd}(R_H^\top)$ .
  - 15: Extract the largest  $k$  singular-pairs from  $\widehat{\mathbf{W}}_l, \widehat{\mathbf{\Sigma}}_l, \widehat{\mathbf{T}}_l$  to obtain  $\widehat{\mathbf{W}}_k, \widehat{\mathbf{\Sigma}}_k, \widehat{\mathbf{T}}_k$ .
  - 16: Assign  $\widehat{\mathbf{U}}_k \leftarrow \overline{\mathbf{Q}} \widehat{\mathbf{W}}_k$ .
  - 17: Assign  $\widehat{V}_k^{(j)} \leftarrow V_0^{(j)} V_1^{(j)} \dots V_H^{(j)} \widehat{\mathbf{T}}_k$ .
  - 18: Gather  $\widehat{\mathbf{V}}_k \leftarrow \left[ \widehat{V}_k^{(1)\top} \quad \widehat{V}_k^{(2)\top} \quad \dots \quad \widehat{V}_k^{(P)\top} \right]^\top$  (MPI\_Gather).
- 

We also propose the block-column version of the Symmetric Postprocessing ([Algorithm IV-3](#)). Although the matrix  $\mathbf{A}$  is not short and fat (due to the symmetry of  $\mathbf{A}$ ), since  $\mathbf{A}$  is symmetric, we can use Block-Row Parallelism in sketching and Block-Column Parallelism in forming to minimize the communication.

---

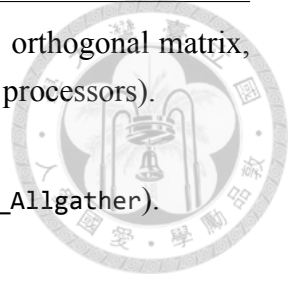
**Algorithm IV-9** Symmetric Postprocessing (Block-Column Parallelism)

---

**Require:**  $A^{(j)}$  (real  $m_b \times m$  matrix, column-block),  $\overline{Q}^{(j)}$  (real  $m_b \times l$  orthogonal matrix, row-block),  $k$  (desired rank of approximate SVD),  $P$  (number of processors).

**Ensure:** Approximate rank- $k$  SVD of  $\mathbf{A} \approx \widehat{\mathbf{U}}_k \widehat{\mathbf{\Sigma}}_k \widehat{\mathbf{U}}_k^\top$ .

- 1: Gather  $\overline{\mathbf{Q}} \leftarrow \left[ \overline{Q}^{(1)\top} \quad \overline{Q}^{(2)\top} \quad \dots \quad \overline{Q}^{(P)\top} \right]^\top$  to all processors (MPI\_Allgather).
  - 2: Assign  $\mathbf{Z}^{(j)} \leftarrow A^{(j)\top} \overline{\mathbf{Q}}$  in processor  $j$ .
  - 3: Sum  $\mathbf{M} \leftarrow \sum_{j=1}^P \mathbf{Z}^{(j)\top} \overline{Q}^{(j)}$  to all processors (MPI\_Allreduce).
  - 4: Compute  $\widehat{\mathbf{W}}_l \widehat{\mathbf{\Sigma}}_l \widehat{\mathbf{W}}_l^\top \leftarrow \text{eig}(\mathbf{M})$ .
  - 5: Extract the largest  $k$  eigen-pairs from  $\widehat{\mathbf{W}}_l, \widehat{\mathbf{\Sigma}}_l$  to obtain  $\widehat{\mathbf{W}}_k, \widehat{\mathbf{\Sigma}}_k$ .
  - 6: Assign  $\widehat{\mathbf{U}}_k \leftarrow \overline{\mathbf{Q}} \widehat{\mathbf{W}}_k$ .
- 



## 4.6 GPU Acceleration

Sketching and postprocessing stages are the only algorithms using the huge input matrix  $\mathbf{A}$ , and cost square to the input size ( $O(mnl)$  flops). Fortunately, all the operations require  $\mathbf{A}$  are matrix-matrix multiplication, which can be highly accelerated using GPU [9].

We use GPU to compute  $\mathbf{Y}_{[i]} \leftarrow \mathbf{A} \mathbf{Q}_{[i]}$  in sketching and  $\mathbf{Z} \leftarrow \mathbf{A}^\top \overline{\mathbf{Q}}$  in postprocessing stages. For  $\mathbf{A}$  with extremely large sizes that exceed the GPU memory limit, we split the matrices into column/row blocks and compute the multiplication successively.







## Chapter 5

# Comparison of Algorithms

In this chapter, we denote  $I$  as the number of iteration, and  $I_s$  as the total number of iteration in finding step size ( $I_s \geq I$ ). Here we assume  $m \gg Nl$ ,  $l = k$ , and  $q = 0$ . The following tables only show the leading terms.<sup>1</sup> In the table, we abbreviate the Kolmogorov-Nagumo Iteration, the Wen-Yin Iteration, and the Hierarchical Reduction Integration as KN, WY, and HR respectively.

Tables 5.1 to 5.3 summarize the computational and communication complexity of all the algorithms. For integration, we obtain that the optimized algorithms indeed reduce the computational complexity. We do not recommend the Gramian algorithms, unless one expects the number of iteration to be larger than  $\frac{N}{4}$ . For orthogonalization and postprocessing, we recommend the Gramian algorithms, which are fast and well-parallelized.

By comparing the parallelisms, we obtain that the block-row postprocessing requires an extra  $nl$  communication, and the block-column sketching requires an extra  $Nml$  communication. Therefore we recommend using Block-Column Parallelisms on sketching and postprocessing if and only if the input matrix  $\mathbf{A}$  is short and fat enough (that is,  $Nm < n$ ).

---

<sup>1</sup>See [Appendix B](#) for detail.

Table 5.1: The Complexity of Sequential Algorithms

Stage	Name	#flops
Sketching	<a href="#">Algorithm I-1</a> Gaussian Projection	$2Nnml$
Orthogonalization	<a href="#">Algorithm II-1</a> Canonical QR	$4Nml^2$
	<a href="#">Algorithm II-2</a> Gramian	$3Nml^2$
Integration	<a href="#">Algorithm III-1</a> KN (Original)	$(4N + 10)ml^2 I$
	<a href="#">Algorithm III-4</a> KN (Optimized)	$(4N + 4)ml^2 I$
	<a href="#">Algorithm III-6</a> KN (Gramian)	$(N^2 + 2N + 2)ml^2$
	<a href="#">Algorithm III-2</a> WY (Original)	$(2N + 8)ml^2 I_s + (4N + 4)ml^2 I$
	<a href="#">Algorithm III-5</a> WY (Optimized)	$(4N + 6)ml^2 I$
	<a href="#">Algorithm III-7</a> WY (Gramian)	$(N^2 + 2N + 2)ml^2$
Postprocessing	<a href="#">Algorithm III-3</a> HR	$6Nml^2$
	<a href="#">Algorithm IV-1</a> SVD with QR	$2nml + 6nl^2 + 2ml^2$
	<a href="#">Algorithm IV-2</a> Gramian	$2nml + 3nl^2 + 2ml^2$
	<a href="#">Algorithm IV-3</a> Symmetric	$2m^2l + 3ml^2$

Table 5.2: The Complexity of Block-Row Algorithms

Stage	Name	#flops	#words
Sketching	<a href="#">Algorithm I-2</a> Gaussian Projection	$2\frac{N}{P}nml$	-
Orthogonalization	<a href="#">Algorithm II-4</a> TSQR	$4\frac{N}{P}ml^2$	$(\log P)Nl^2$
	<a href="#">Algorithm II-3</a> Gramian	$3\frac{N}{P}ml^2$	$(\log P)Nl^2$
Integration	<a href="#">Algorithm III-10</a> KN	$\frac{1}{P}(4N + 4)ml^2 I$	$(\log P)Nl^2 I$
	<a href="#">Algorithm III-6</a> KN (Gramian)	$\frac{1}{P}(N^2 + 2N + 2)ml^2$	$(\log P)N^2 l^2$
	<a href="#">Algorithm III-11</a> WY	$\frac{1}{P}(4N + 6)ml^2 I$	$(\log P)Nl^2 I$
	<a href="#">Algorithm III-7</a> WY (Gramian)	$\frac{1}{P}(N^2 + 2N + 2)ml^2$	$(\log P)N^2 l^2$
	<a href="#">Algorithm III-12</a> HR	$6\frac{N}{P}ml^2$	$(\log P)Nl^2$
Postprocessing	<a href="#">Algorithm IV-6</a> TSQR	$\frac{1}{P}(2nml + 4nl^2 + 2ml^2)$	$2nl + ml$
	<a href="#">Algorithm IV-4</a> Gramian	$\frac{1}{P}(2nml + 3nl^2 + 2ml^2)$	$2nl + ml$
	<a href="#">Algorithm IV-5</a> Symmetric	$\frac{1}{P}(2m^2l + 3ml^2)$	$2ml$

Table 5.3: The Complexity of Block-Column Algorithms

Stage	Name	#flops	#words
Sketching	<a href="#">Algorithm I-3</a> Gaussian Projection	$2\frac{N}{P}nml$	$Nml$
Postprocessing	<a href="#">Algorithm IV-8</a> TSQR	$\frac{1}{P}(2nml + 4nl^2) + 2ml^2$	$nl + ml$
	<a href="#">Algorithm IV-7</a> Gramian	$\frac{1}{P}(2nml + 3nl^2) + 2ml^2$	$nl + ml$
	<a href="#">Algorithm IV-9</a> Symmetric	$\frac{1}{P}(2m^2l + ml^2) + 2ml^2$	$ml$



# Chapter 6

## Implementation

The iSVD algorithm is implemented in C++ with more than 20000 lines of codes. The package provides multinode, multithread, and GPU support. We also use some tools for automatic installation and correctness test.

### 6.1 C++ Implementation

We design and implement iSVD package in C++ with objective oriented programming model. It is paralleled using MPI for multicore clusters. Intel Math Kernel Library [10] is used for BLAS/LAPACK [11] routines. The package is highly benefited from adopting object-oriented programming, C++11 standard [12], and template meta-programming. We also wrap the BLAS/LAPACK routines that delegate the selection of the correct type of routines to the compiler. With these techniques, the package can be easily used with different types of input matrix and different iSVD algorithms without modifying exist codes.

#### 6.1.1 Object Oriented Programming

Object-oriented programming (OOP) is a programming paradigm based on objects, which combines data and functions. The benefit to adopt OOP is huge in our implementation. With OOP, we warp data structures into object, and design linear algebra routines based on these object instead of using raw data directly. This technique makes programming

easier and less fallible. The iSVD algorithms are also implemented with OOP. The user can use them easily without dealing with parameter transfer and memory control.



### 6.1.2 C++11 Standard

Our implementation benefited plenty from C++11 standard [12]. This new standard is a great improvement from C++03, with many new features such as right value reference helps developer manipulate temporary objects generated by some syntax and avoid performance traps, `constexpr` specifier allows the compiler to do more compile-time decision and gain better optimization. The standard library of C++11 provided a smart pointer classes such as `shared_ptr` with features such as automatic memory management or bounds checking. All of the data in our package are controlled by smart pointers. One no need to concerned about the dynamic memory allocations.

### 6.1.3 Template Meta-Programming

Template, one of an important feature of C++, allows function and classes to operate with generic types. In iSVD Package, almost all the codes, such as storage structures, linear algebra functions, and iSVD drivers, are implemented with template. With template usage, we can combine write a single code for all data types and storage layout instead of having multiple similar codes.

The implementation only contains header files. One can easily use it without any binary library to link to, and any configured header file. Our package is a pure template library defined in the headers.

### 6.1.4 Curiously Recurring Template Pattern

The curiously recurring template pattern (CRTP) was formalized in the 1980s as “F-bounded quantification” [13], and was named by Jim Coplien [14] in 1995. CRTP is an idiom in C++ in which a class `x` derives from a class template instantiation using `x` itself as template argument [15]. With CRTP, we implement interfaces without virtual functions to reduce run-time overheads.

## 6.1.5 Data Storage and Linear Algebra Wrappers

We implement the matrix structures using OOP, and implement methods so that one may use it similar to the usage of MATLAB matrices. The expressions such as transpose, sub-matrix access, and data type changes are done in compile time. We also warp BLAS/LAPACK routines with our data structures using template to decide the correct type of routines in compile-time. These wrappers are written in intuitive interfaces. With above these technique, one does not need to concern for the complicated matrix memory structure and BLAS/LAPACK routines.

## 6.2 Parallelism

### 6.2.1 MPI

The support of MPI allows iSVD package to access more computing resources. In [Chapter 4](#), we discussed the parallelism using multiple computing nodes. Like BLAS/LAPACK, MPI routines are also wrapped using template for convenience.

### 6.2.2 OpenMP

The Intel Math Kernel Library [10] provided multi-threaded BLAS/LAPACK libraries based on OpenMP threading. However, since the vector statistics library of MKL does not support multi-threading, we use OpenMP [16] to parallel the random generating and gain near-linear acceleration.

### 6.2.3 GPU

The support of GPU accelerates the BLAS-3 matrix-matrix multiplication of the input huge matrix  $\mathbf{A}$ . The MAGMA library [9] provides efficient linear algebra routines implemented on GPU. Our implementation benefited plenty from MAGMA library and gains further 4X faster timing performance.

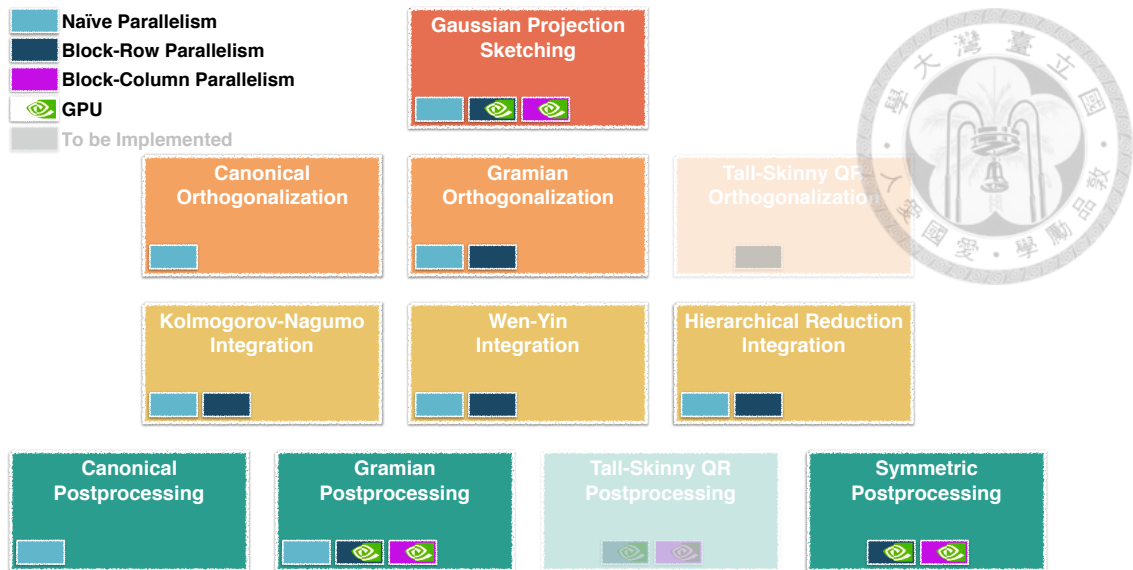


Figure 6.1: The Table of Implemented Algorithms

## 6.3 Tools

### 6.3.1 CMake

CMake is a cross-platform tool to build and test package. It allows the user to install the package without modifying any configuration codes. It can automatically determine the compile flags for linking libraries. With CMake, one can use a simple GUI to choose the library, and switch the options in our package.

### 6.3.2 Google Test

iSVD uses Google Test, a unit testing library for C++, to test the correctness of the package. We also combine it with CMake's test system. It allows us to make sure that each modification of our code does not harm existing features, and allows the user to check if his/her system is compatible with our code.

## 6.4 Package Structure

[Figure 6.1](#) lists the algorithms implemented in iSVD package. The red blocks are the sketching algorithms, the orange ones are the orthogonalization algorithms, the yellow ones are the integration algorithms, and the green ones are the postprocessing algorithms. The light blue stickers represent that the algorithm is implemented with Naïve Parallelism, the dark blue ones represent the Block-Row Parallelism, and the purple ones represent the Block-Column Parallelism. The NVIDIA logos represent that the algorithm has GPU support. The translucent items are not finished yet but planned to be done in the future updates. We also provide some routines for converting data between naïve storage, block-row storage, and block-column storage.

The Canonical Orthogonalization and the Canonical Postprocessing are implemented in neither Block-Row Parallelism nor Block-Column Parallelism since the canonical QR and SVD are difficult to be paralleled. Instead, we provide Tall-Skinny QR as the block-row/column version. The Block-Column Parallelism is only available in the sketching and postprocessing stages since they are the only stages that contain the short and fat matrix **A**. (See [Section 4.5](#) for detail)







## Chapter 7

# Numerical Results

We use two computer systems to test our program. The single node tests use a computer with two Intel Xeon E5-2650 v4 CPU (Broadwell-EP 2.20GHz, 12cores, 30MB L3-cache) and 512GB RAM (DDR4-2400, 153.6 GB/sec). We also use the Reedbush supercomputer system of the University of Tokyo. Each node of the Reedbush-U has two Intel Xeon E5-2695 v4 CPU (Broadwell-EP 2.10GHz, 18cores, 45MB L3-cache) with 256GB RAM (DDR4-2400, 153.6 GB/sec). The connection between nodes uses InfiniBand EDR 4x (100 Gbps/node). The Reedbush-H is an extension of the Reedbush-U. It contains two NVIDIA Tesla P100 GPU (16 GB Memory) per node.

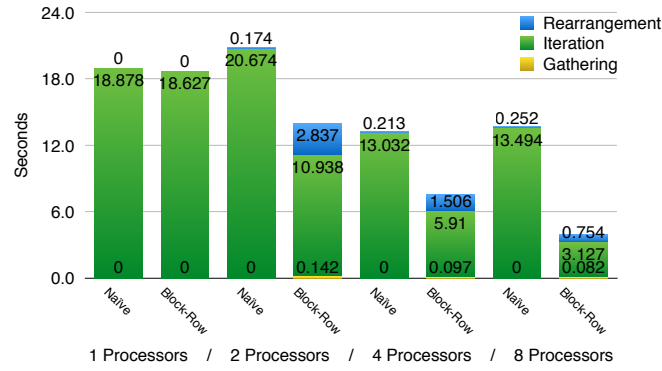
In this chapter, we first (in [Section 7.1](#)) compare the Naïve Parallelism and the Block-Row Parallelism to show that the advantage of the Block-Row Parallelism. In [Section 7.2](#), we show the scalability of the integration algorithms. Last, we display our implementation and apply it on two real big data application in [Section 7.3](#).

### 7.1 Comparison of Parallelisms

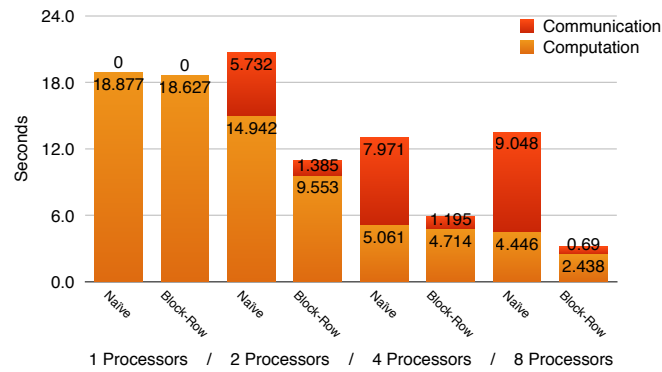
In this section, we compare the Naïve Parallelism ([Algorithm III-8](#)) and the Block-Row Parallelism ([Algorithm III-10](#)) on the Reedbush-U supercomputer system<sup>1</sup>. Here we use Kolmogorov-Nagumo Integration as an example since the algorithm structure of the in-

---

<sup>1</sup>Each node contains Intel Xeon E5-2695 v4 CPU ×2 (Broadwell-EP 2.10GHz, 18cores, 45MB L3-cache), 256GB RAM (DDR4-2400, 153.6 GB/sec), InfiniBand EDR 4x (100 Gbps/node).



(a) Kolmogorov-Nagumo Integration

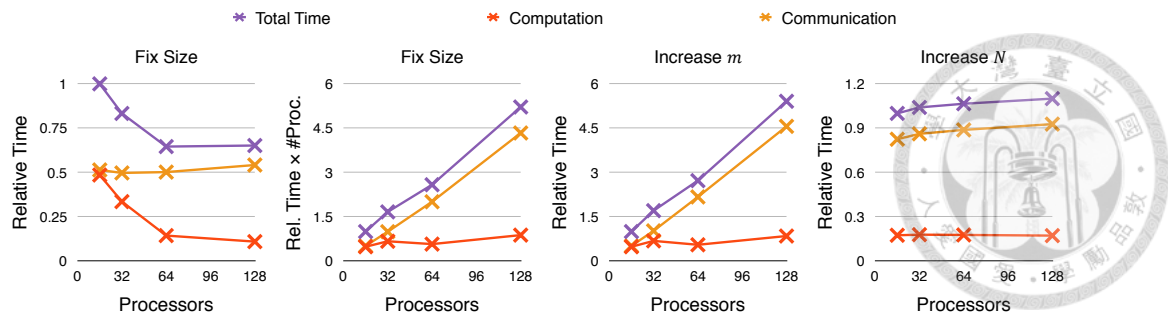


(b) Iteration Part of Kolmogorov-Nagumo Integration

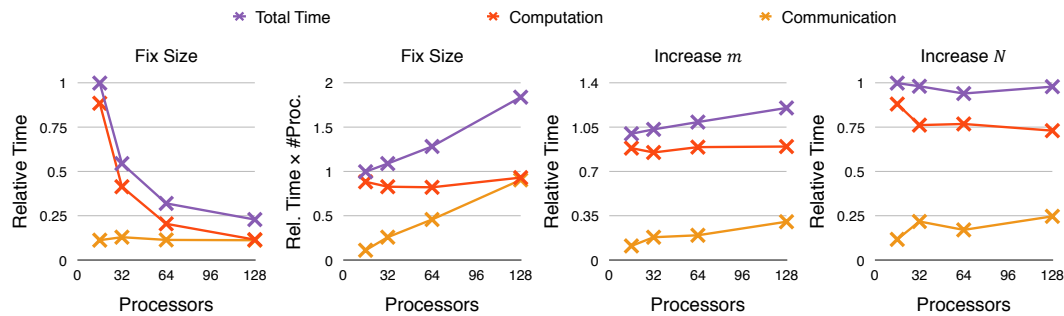
Figure 7.1: Comparison of Naïve and Block-Row Parallelism

tegration algorithms are similar. For the parameters, we set  $m = 10^6$ ,  $l = 100$ ,  $N = 8$ , randomly generate the orthogonal matrices  $\mathbf{Q}_{[i]}$  and force the algorithm to run 16 iterations. Here we use 1, 2, 4, 8 processors. Figure 7.1 shows the average time of 10 repeats.

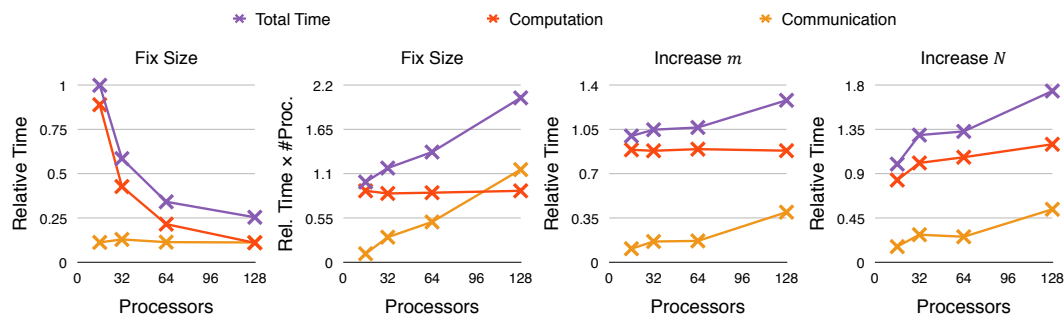
Figure 7.1a are the costs of each part of Kolmogorov-Nagumo Integration, and Figure 7.1b are the computing and communication costs of the iteration part. We find that the Naïve Parallelism has bad scalability. From Figure 7.1a, we find that most of the execution time is used in the iteration. Therefore, we focus on the behavior of the iteration. In Figure 7.1b, we conclude that the bottleneck of the naïve version is the communication. The Naïve Parallelism with two processors is even slower than using single processor since the extra communication waste too much time. It is shown that both computing and communication costs are reduced in Block-Row Parallelism.



(a) Naïve Parallelism of Kolmogorov-Nagumo Integration



(b) Block-Row Parallelism of Kolmogorov-Nagumo Integration

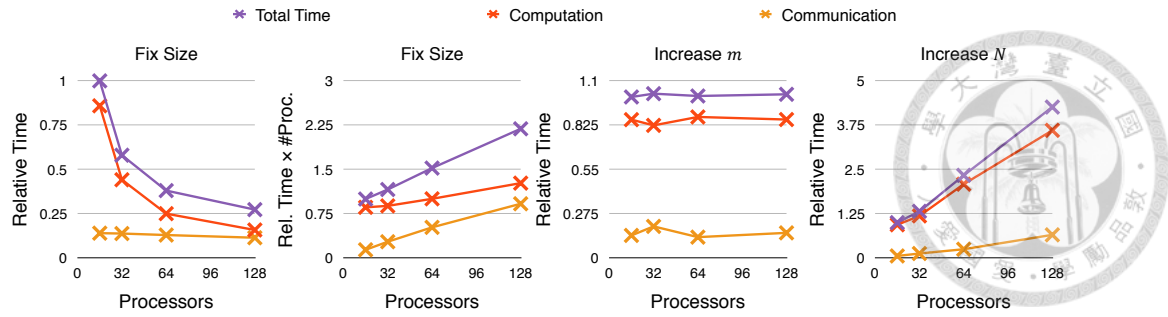


(c) Block-Row Parallelism of Wen-Yin Integration

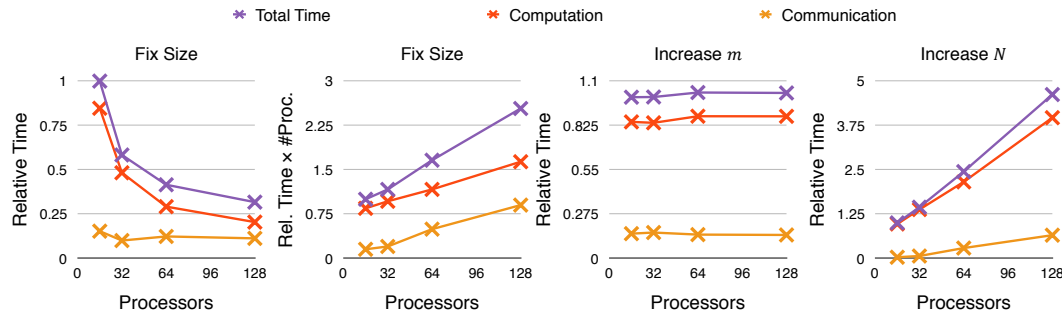
Figure 7.2: Scalability of Integration Algorithms

## 7.2 Scalability of Integration Algorithms

We test the scalability on the Reedbush-U supercomputer system. Here, we compare the Naïve Parallelism (Kolmogorov-Nagumo Integration [Algorithm III-8](#)) and the Block-Row Parallelism (Block-Row Kolmogorov-Nagumo Integration [Algorithm III-10](#), Block-Row Wen-Yin Integration [Algorithm III-11](#)). We also test the scalability of the Gramian algorithms with Block-Row Parallelism (Block-Row Gramian Kolmogorov-Nagumo Integration [Algorithm III-6](#) and Block-Row Gramian Wen-Yin Integration [Algorithm III-7](#)).



(d) Block-Row Parallelism of Gramian Kolmogorov-Nagumo Integration



(e) Block-Row Parallelism of Gramian Wen-Yin Integration

Figure 7.3: Scalability of Integration Algorithms (Conti.)

For each algorithm, we test the strong scalability — increase  $P$  and fix the problem size — and the weak scalability — scale both  $P$  and the problem size [17]. The meaning of strong scalability is to minimize the time-to-solution for a fixed size problem, and the meaning of weak scalability is to achieve constant time-to-solution for a large problem.

In the strong scalability test, we set  $P = 16, 32, 64, 128$  with fixing  $N = 128, m = 10^6$ . We have two tests for weak scalability. First we test for  $P = 16, 32, 64, 128$  and  $m = 1 \cdot 10^6, 2 \cdot 10^6, 4 \cdot 10^6, 8 \cdot 10^6$  with fixing  $N = 128$ . In the second test, we set  $P = 16, 32, 64, 128$  and  $N = 16, 32, 64, 128$  with fixing  $m = 8 \cdot 10^6$ . In all of the tests, we set  $l = 100$ , randomly generate the orthogonal matrices  $\mathbf{Q}_{[i]}$  and force the algorithm to run 16 iterations. Here we ignore the rearrangement (Subroutine S-1) and the gathering (Subroutine S-5) since these subroutines do not appear in the implementation. (See Section 4.4 for detail)

Figures 7.2 to 7.3 are the scalability test results. The x-axis is the number of processors ( $P$ ), and the y-axis is the relative time. The first and second columns are the tests for strong scalability, and the third and the fourth columns are the weak scalability ones. In the strong

scalability, we expect the total execution time is inversely proportional to the number of processor  $P$ . Therefore, in the second column, we show the summation of the execution time of all the processors (i.e. the product of  $P$  and the execution time). The algorithms are scalable if the lines are flat in the second, the third, and the fourth columns.

From Figure 7.2a, we find that the Naïve Parallelism is only scalable while increasing  $N$ . However, this parallelism is not recommended since the communication cost is too expensive (almost 4x than the computational cost). Figure 7.2b and Figure 7.2c are the results of the Block-Row Parallelisms. We find that both of the algorithms are weak scalable. As shown from Figure 7.3d and Figure 7.3e that the Gramian algorithms are weak scalable while increasing  $m$ . However, they are not scalable with  $N$  increasing since the complexity is  $O(N^2ml^2/P)$ , which increases too rapidly with  $N$  increasing.

All of the Block-Row Parallelisms are not strong scalable since the computational cost decreased too rapidly with more processors, while the communication cost is almost unchanged. Therefore the communication cost becomes a bottleneck while using many processors.

### 7.3 Implementation and Big Data Applications

We use iSVD to find the first 20 principal components of two huge size dataset, Facebook 100k and 1000 Genomes Project. We test these dataset using MATLAB and C++ on the single node computer<sup>2</sup> by MATLAB and C++. The multinode tests are tested on the Reedbush-H supercomputer system<sup>3</sup>. For computational resources, we use 4 node, with 8 GPUs. In each node, we use two MPI ranks. Each MPI rank handles a GPU. We use iSVD to find the first 20 principal components of two huge size dataset, Facebook 100k and 1000 Genomes Project.

---

<sup>2</sup>Intel Xeon E5-2650 v4 CPU ×2 (Broadwell-EP 2.20GHz, 12cores, 30MB L3-cache), 512GB RAM (DDR4-2400, 153.6 GB/sec).

<sup>3</sup>Each node contains Intel Xeon E5-2695 v4 CPU ×2 (Broadwell-EP 2.10GHz, 18cores, 45MB L3-cache), 256GB RAM (DDR4-2400, 153.6 GB/sec), InfiniBand EDR 4x (100 Gbps/node), NVIDIA Tesla P100 GPU ×2 (15GB Memory).

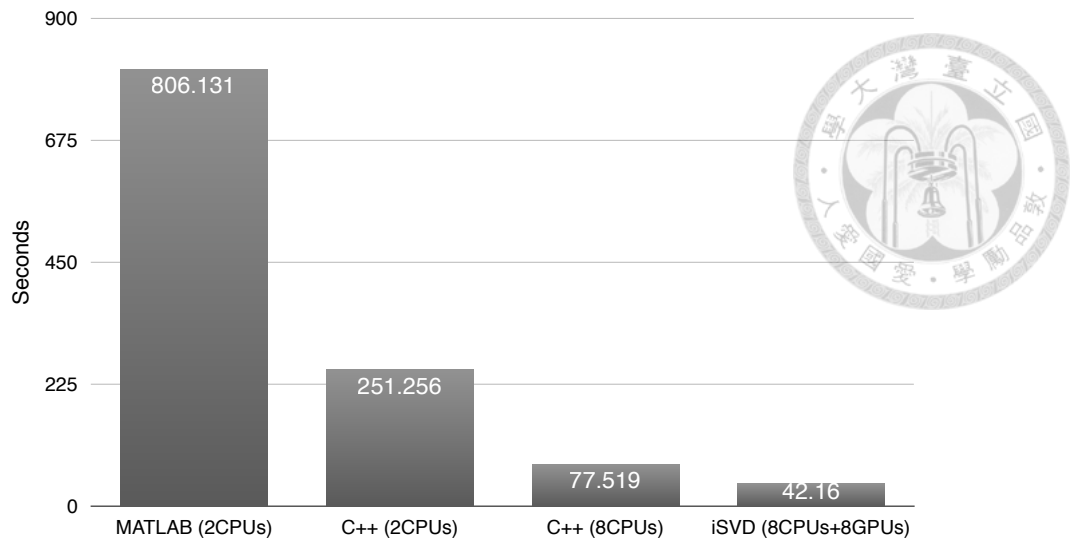


Figure 7.4: The Time of iSVD using MATLAB and C++

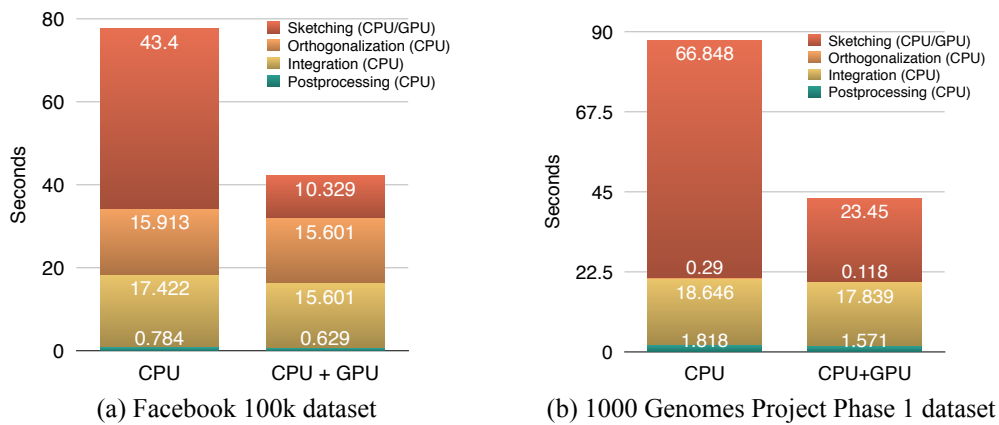


Figure 7.5: The Time of iSVD using CPU and GPU

### 7.3.1 Comparison with MATLAB

We compare our program with MATLAB using the Facebook 100k dataset (see [Section 7.3.2](#) for detail). From [Section 7.3.1](#), we find that our C++ program is 3X faster than the MATLAB code. With more computing resources, the program can be 10X faster using 8 CPUs, and 19X faster with using extra 8 GPUs.

### 7.3.2 Facebook 100k

Facebook 100k<sup>4</sup> is a dataset of social network connection. The contains the information of 108,585 fan pages. The  $(i, j)$  element of the symmetric matrix  $\mathbf{A}$  stores the common

<sup>4</sup>Provided by Shih-En Chou.

fans of  $i$ -th and the  $j$ -th fan pages.

We use block-row iSVD on this dataset. Here we use the Block-Row Gaussian Projection Sketching (Algorithm I-2,  $q = 0$ ), the Block-Row Gramian Orthogonalization (Algorithm II-3), the Block-Row Wen-Yin Integration (Algorithm III-11), and the Block-Row Gramian Postprocessing (Algorithm IV-4). We use GPU in sketching stage. The parameters are set as  $k = 20, p = 12, N = 256, \epsilon = 10^{-3}$ . Both cases converges in 49 iterations in the integration stage.

As shown in Figure 7.5a, the sketching stage with GPU is about 4X faster than pure CPU version (from 43 to 10 seconds). The total time is reduced from 78 to 42 seconds. The acceleration of GPU saves almost half of the execution time.

### 7.3.3 1000 Genomes Project

1000 Genomes Project [18] is a dataset of people's genotype. Here we use the Phase 1 dataset, which contains 36,781,560 variants of 1,092 individuals. All the data are stored in a  $1,092 \times 36,781,560$  matrix  $\mathbf{A}$ . Due to the size of the problem, the original MATLAB code require more than an hour (3769.141745 seconds).

We use block-column iSVD on this dataset. Here we use the Block-Column Gaussian Projection Sketching (Algorithm I-3,  $q = 0$ ), the Block-Row Gramian Orthogonalization (Algorithm II-3), the Block-Row Wen-Yin Integration (Algorithm III-11), and the Block-Column Gramian Postprocessing (Algorithm IV-7). We use GPU in sketching stage. The parameters are set as  $k = 20, p = 12, N = 64, \epsilon = 10^{-3}$ . The CPU version converges in 86 iterations in the integration stage, and the hybrid CPU-GPU version converges in 96 iterations.

As shown in Figure 7.5b, the sketching stage with GPU is about 3X faster than pure CPU version (from 67 to 23 seconds). The total time is reduced from 88 to 43 seconds. The acceleration of GPU saves almost half of the execution time. Compare to the original MATLAB code, our GPU version achieve about 88X faster timing performance.<sup>5</sup>

---

<sup>5</sup>However, we do not recommend to use iSVD on such short matrix. The total dimension of sketching  $Nl$  is 2048, which is larger than the number of rows in  $\mathbf{A}$ . In this case, computing the eigenvalue decomposition  $\mathbf{A}\mathbf{A}^T$  directly is faster and more accurate.







## Chapter 8

### Conclusion

Integrated Singular Value Decomposition is an algorithm that finds low-rank SVD efficiently with good accuracy. It integrates different low-rank SVDs obtained by multiple random subspace sketches. While iSVD takes higher computational costs due to multiple random sketches and the integration process, these operations can be parallelized to save computational time. Due to the size of the problem, data communication becomes a bottleneck.

We optimize the Kolmogorov-Nagumo Integration and Wen-Yin Integration for better performance. The optimized algorithms reuse some matrices and explicitly generate fewer matrices than the original matrices to reduce the computational cost. We also propose algorithms of these two algorithms based on the Gramian idea, which is faster if the number of iteration is larger than the quarter of the number of random sketches.

In order to use high performance computer for acceleration, we propose the Block-Row Parallelism and the Block-Column Parallelism. The block-row algorithms are aimed to the problems with near-square or square input matrix, and the block-column algorithms are recommended if the input matrix is short and fat. We also use GPU acceleration and achieve about 4X faster timing performance on the sketching stage. With these parallelism schemes, these algorithms are balanced, and the communication cost is reduced to a small number which is independent of the size of the problem.

The iSVD algorithm is carefully designed and implemented in C++. We use several techniques for high maintainability, extensibility, and usability. The package is de-

signed with multinode, multithread, and GPU support. CMake and Google Test are used for automatic installation and correctness test. Our package is 3X faster than the MATLAB version. From our numerical results, the Block-Row Parallelism of the Optimized Kolmogorov-Nagumo Integration and the Optimized Wen-Yin Integration are shown to be weak scalable respect to the size of the problem and the number of random sketches on the Reedbush supercomputer system of the University of Tokyo with up to 128 nodes. The iSVD package is applied on two kinds of huge size applications, and compute the result within a minutes using 4 computing nodes with 8 GPUs.

Low-rank approximation plays an important role in big data analysis. With our implementation, Integrated Singular Value Decomposition solves huge size applications on hybrid CPU-GPU supercomputer systems efficiently and scalably.



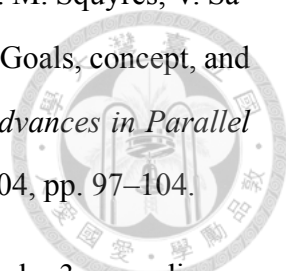
# Bibliography

- [1] N. Halko, P.-G. Martinsson, and J. A. Tropp, “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions,” *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.
- [2] V. Rokhlin, A. Szlam, and M. Tygert, “A randomized algorithm for principal component analysis,” *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 3, pp. 1100–1124, 2009.
- [3] T.-L. Chen, D. D. Chang, S.-Y. Huang, H. Chen, C. Lin, and W. Wang, “Integrating multiple random sketches for singular value decomposition,” *arXiv preprint arXiv:1608.08285*, 2016.
- [4] S. Fiori, T. Kaneko, and T. Tanaka, “Mixed maps for learning a Kolmogoroff-Nagumo-type average element on the compact Stiefel manifold,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 4518–4522.
- [5] Z. Wen and W. Yin, “A feasible method for optimization with orthogonality constraints,” *Mathematical Programming*, vol. 142, no. 1-2, pp. 397–434, 2013.
- [6] D. D. Chang, “Theoretical and performance analysis for integrated randomized singular value decomposition,” Master’s thesis, National Taiwan University, 2017, unpublished.
- [7] L. S. Blackford, J. Choi, A. Cleary, E. D’Azevedo, J. Demmel, I. Dhillon, J. Don-

garra, S. Hammarling, G. Henry, A. Petit, *et al.*, *ScaLAPACK Users' Guide*. SIAM, 1997.



- [8] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, “Communication-optimal parallel and sequential QR and LU factorizations,” *SIAM Journal on Scientific Computing*, vol. 34, no. 1, pp. A206–A239, 2012.
- [9] S. Tomov, J. Dongarra, and M. Baboulin, “Towards dense linear algebra for hybrid GPU accelerated manycore systems,” *Parallel Computing*, vol. 36, no. 5, pp. 232–240, 2010.
- [10] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, “Intel math kernel library,” in *High-Performance Computing on the Intel® Xeon Phi™*. Springer, 2014, pp. 167–188.
- [11] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, *et al.*, *LAPACK Users' Guide*. SIAM, 1999.
- [12] C. S. Committee and Others, “ISO/IEC 14882: 2011, standard for programming language C++,” Technical report, 2011. <http://www.open-std.org/jtc1/sc22/wg21>, Tech. Rep., 2011.
- [13] P. Canning, W. Cook, W. Hill, W. Olthoff, and J. C. Mitchell, “F-bounded polymorphism for object-oriented programming,” in *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*. ACM, 1989, pp. 273–280.
- [14] J. O. Coplien, “Curiously recurring template patterns,” *C++ Report*, vol. 7, no. 2, pp. 24–27, 1995.
- [15] D. Abrahams and A. Gurtovoy, *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond, Portable Documents*. Pearson Education, 2004.

- 
- [16] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, *et al.*, “Open MPI: Goals, concept, and design of a next generation MPI implementation,” in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2004, pp. 97–104.
- [17] I. T. Todorov, W. Smith, K. Trachenko, and M. T. Dove, “Dl\_poly\_3: new dimensions in molecular dynamics simulations via massive parallelism,” *Journal of Materials Chemistry*, vol. 16, no. 20, pp. 1911–1918, 2006.
- [18] 1000 Genomes Project Consortium and others, “A global reference for human genetic variation,” *Nature*, vol. 526, no. 7571, p. 68, 2015.
- [19] E. Anderson, J. Dongarra, and S. Ostrouchov, *LAPACK Working Note 41: Installation Guide for LAPACK*. University of Tennessee, Computer Science Department, 1992.





# Appendix A

## Derivations

### A.1 Derivation of Hierarchical Reduction Integration

Hierarchical Reduction Integration ([Algorithm III-3](#)) computes the integrated orthonormal basis using divide and conquer. Here we describe the algorithm using an example with  $N = 4$ . First, we compute the orthonormal basis of  $\{\mathbf{Q}_{[1]}, \mathbf{Q}_{[2]}\}$  and  $\{\mathbf{Q}_{[3]}, \mathbf{Q}_{[4]}\}$ ; that is,

$$\bar{\mathbf{Q}}_{[12]} = \text{orth}_l\left(\begin{bmatrix} \mathbf{Q}_{[1]} & \mathbf{Q}_{[2]} \end{bmatrix}\right) \quad \text{and} \quad \bar{\mathbf{Q}}_{[34]} = \text{orth}_l\left(\begin{bmatrix} \mathbf{Q}_{[3]} & \mathbf{Q}_{[4]} \end{bmatrix}\right). \quad (\text{A.1})$$

Next, we compute the orthonormal basis of these basis

$$\bar{\mathbf{Q}}_{[1234]} = \text{orth}_l\left(\begin{bmatrix} \bar{\mathbf{Q}}_{[12]} & \bar{\mathbf{Q}}_{[34]} \end{bmatrix}\right) \quad (\text{A.2})$$

and set  $\bar{\mathbf{Q}}_{\text{hr}} = \bar{\mathbf{Q}}_{[1234]}$  as the output.

We found that the orthonormal basis of  $\mathbf{M} = \begin{bmatrix} \mathbf{Q}_{[i]} & \mathbf{Q}_{[j]} \end{bmatrix}$  can be compute analytically using the SVD of  $\mathbf{Q}_{[i]}^\top \mathbf{Q}_{[j]}$ . Denote the SVDs as  $\mathbf{L}\mathbf{E}\mathbf{R}^\top = \mathbf{M}$  and  $\mathbf{W}\mathbf{S}\mathbf{T}^\top = \mathbf{Q}_{[i]}^\top \mathbf{Q}_{[j]}$ .

Therefore, the eigenvalue decomposition of  $\mathbf{M}^\top \mathbf{M}$  is

$$\begin{aligned} \mathbf{M}^\top \mathbf{M} &= \begin{bmatrix} \mathbf{I} & \mathbf{Q}_{[i]}^\top \mathbf{Q}_{[j]} \\ \mathbf{Q}_{[j]}^\top \mathbf{Q}_{[i]} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{W} \mathbf{S} \mathbf{T}^\top \\ \mathbf{T} \mathbf{S} \mathbf{W}^\top & \mathbf{I} \end{bmatrix} \\ &= \left( \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{T} & \mathbf{T} \\ \mathbf{W} & -\mathbf{W} \end{bmatrix} \right) \begin{bmatrix} \mathbf{I} + \mathbf{S} & \\ & \mathbf{I} - \mathbf{S} \end{bmatrix} \left( \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{T} & \mathbf{T} \\ \mathbf{W} & -\mathbf{W} \end{bmatrix} \right)^\top. \end{aligned} \quad (\text{A.3})$$

Since  $\mathbf{Q}_{[i]}$  and  $\mathbf{Q}_{[j]}$  are orthogonal matrices, the singular values of  $\mathbf{Q}_{[i]}^\top \mathbf{Q}_{[j]}$  are less than 1. Therefore,  $\mathbf{I} - \mathbf{S}$  is positive, and all the diagonal elements in  $\mathbf{I} + \mathbf{S}$  are greater than those in  $\mathbf{I} - \mathbf{S}$ , which implies that the leading  $l$  eigenvalues of  $\mathbf{M}^\top \mathbf{M}$  are  $\mathbf{I} + \mathbf{S}$ . Hence,  $\mathbf{E}_l = (\mathbf{I} + \mathbf{S})^{1/2}$ ,  $\mathbf{R}_l = 1/\sqrt{2} \begin{bmatrix} \mathbf{T}^\top & \mathbf{W}^\top \end{bmatrix}^\top$ , and

$$\mathbf{L}_l = \mathbf{M} \mathbf{R}_l \mathbf{E}_l^{-1} = (\mathbf{Q}_{[i]} \mathbf{W} + \mathbf{Q}_{[j]} \mathbf{T}) (2(\mathbf{I} + \mathbf{S}))^{-1/2}. \quad (\text{A.4})$$

## A.2 Derivation of Tall Skinny QR

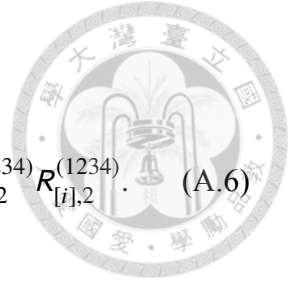
Tall Skinny QR (TSQR) [8] is a block-row algorithm for computing the QR decomposition. Eq. (A.5) shows an example with  $P = 4$ .

$$\begin{aligned} \mathbf{Y}_{[i]} &= \begin{bmatrix} \mathbf{Y}_{[i]}^{(1)} \\ \mathbf{Y}_{[i]}^{(2)} \\ \mathbf{Y}_{[i]}^{(3)} \\ \mathbf{Y}_{[i]}^{(4)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_{[i,0]}^{(1)} \mathbf{R}_{[i,0]}^{(1)} \\ \mathbf{Q}_{[i,0]}^{(2)} \mathbf{R}_{[i,0]}^{(2)} \\ \mathbf{Q}_{[i,0]}^{(3)} \mathbf{R}_{[i,0]}^{(3)} \\ \mathbf{Q}_{[i,0]}^{(4)} \mathbf{R}_{[i,0]}^{(4)} \end{bmatrix}, \\ \begin{bmatrix} \mathbf{R}_{[i,0]}^{(1)} \\ \mathbf{R}_{[i,0]}^{(2)} \\ \mathbf{R}_{[i,0]}^{(3)} \\ \mathbf{R}_{[i,0]}^{(4)} \end{bmatrix} &= \begin{bmatrix} \mathbf{R}_{[i,0]}^{(1)} \\ \mathbf{R}_{[i,0]}^{(2)} \\ \mathbf{R}_{[i,0]}^{(3)} \\ \mathbf{R}_{[i,0]}^{(4)} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{[i,1]}^{(12)} \\ \mathbf{Y}_{[i,1]}^{(34)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_{[i,1]}^{(12)} \mathbf{R}_{[i,1]}^{(12)} \\ \mathbf{Q}_{[i,1]}^{(34)} \mathbf{R}_{[i,1]}^{(34)} \end{bmatrix}, \\ \begin{bmatrix} \mathbf{R}_{[i,1]}^{(12)} \\ \mathbf{R}_{[i,1]}^{(34)} \end{bmatrix} &= \mathbf{Y}_{[i,2]}^{(1234)} = \mathbf{Q}_{[i,2]}^{(1234)} \mathbf{R}_{[i,2]}^{(1234)}. \end{aligned} \quad (\text{A.5})$$



Therefore,

$$\mathbf{Y}_{[i]} = \begin{bmatrix} \gamma_{[i]}^{(1)} \\ \gamma_{[i]}^{(2)} \\ \gamma_{[i]}^{(3)} \\ \gamma_{[i]}^{(4)} \end{bmatrix} = \begin{bmatrix} \mathcal{Q}_{[i],0}^{(1)} & & & \\ & \mathcal{Q}_{[i],0}^{(2)} & & \\ & & \mathcal{Q}_{[i],0}^{(3)} & \\ & & & \mathcal{Q}_{[i],0}^{(4)} \end{bmatrix} \begin{bmatrix} \mathcal{Q}_{[i],1}^{(12)} \\ \mathcal{Q}_{[i],1}^{(34)} \end{bmatrix} \mathcal{Q}_{[i],2}^{(1234)} \mathbf{R}_{[i],2}^{(1234)}. \quad (\text{A.6})$$



Note that  $\mathbf{R} = \mathbf{R}_{[i],2}^{(1234)}$ , and the orthonormal basis is

$$\begin{aligned} \mathbf{Q}_{[i]} &= \begin{bmatrix} \mathcal{Q}_{[i]}^{(1)} \\ \mathcal{Q}_{[i]}^{(2)} \\ \mathcal{Q}_{[i]}^{(3)} \\ \mathcal{Q}_{[i]}^{(4)} \end{bmatrix} = \begin{bmatrix} \mathcal{Q}_{[i],0}^{(1)} & & & \\ & \mathcal{Q}_{[i],0}^{(2)} & & \\ & & \mathcal{Q}_{[i],0}^{(3)} & \\ & & & \mathcal{Q}_{[i],0}^{(4)} \end{bmatrix} \begin{bmatrix} \mathcal{Q}_{[i],1}^{(12)} \\ \mathcal{Q}_{[i],1}^{(34)} \end{bmatrix} \mathcal{Q}_{[i],2}^{(1234)} \\ &= \begin{bmatrix} \mathcal{Q}_{[i],0}^{(1)} & & & \\ & \mathcal{Q}_{[i],0}^{(2)} & & \\ & & \mathcal{Q}_{[i],0}^{(3)} & \\ & & & \mathcal{Q}_{[i],0}^{(4)} \end{bmatrix} \begin{bmatrix} \mathcal{Q}_{[i],1}^{(1)} \\ \mathcal{Q}_{[i],1}^{(2)} \\ \mathcal{Q}_{[i],1}^{(3)} \\ \mathcal{Q}_{[i],1}^{(4)} \end{bmatrix} \begin{bmatrix} \mathcal{Q}_{[i],2}^{(1)} \\ \mathcal{Q}_{[i],2}^{(3)} \end{bmatrix} \\ &= \begin{bmatrix} \mathcal{Q}_{[i],0}^{(1)} \mathcal{Q}_{[i],1}^{(1)} \mathcal{Q}_{[i],2}^{(1)} \\ \mathcal{Q}_{[i],0}^{(2)} \mathcal{Q}_{[i],1}^{(2)} \mathcal{Q}_{[i],2}^{(2)} \\ \mathcal{Q}_{[i],0}^{(3)} \mathcal{Q}_{[i],1}^{(3)} \mathcal{Q}_{[i],2}^{(3)} \\ \mathcal{Q}_{[i],0}^{(4)} \mathcal{Q}_{[i],1}^{(4)} \mathcal{Q}_{[i],2}^{(4)} \end{bmatrix}, \end{aligned} \quad (\text{A.7})$$

where

$$\begin{aligned} \mathcal{Q}_{[i],1}^{(12)} = \mathcal{Q}_{[i],1}^{(21)} &= \begin{bmatrix} \mathcal{Q}_{[i],1}^{(1)} \\ \mathcal{Q}_{[i],1}^{(2)} \end{bmatrix}, & \mathcal{Q}_{[i],1}^{(34)} = \mathcal{Q}_{[i],1}^{(43)} &= \begin{bmatrix} \mathcal{Q}_{[i],1}^{(3)} \\ \mathcal{Q}_{[i],1}^{(4)} \end{bmatrix}, \\ \mathcal{Q}_{[i],2}^{(1234)} = \mathcal{Q}_{[i],2}^{(13)} = \mathcal{Q}_{[i],2}^{(31)} = \mathcal{Q}_{[i],2}^{(24)} = \mathcal{Q}_{[i],2}^{(42)} &= \begin{bmatrix} \mathcal{Q}_{[i],2}^{(1)} \\ \mathcal{Q}_{[i],2}^{(3)} \end{bmatrix} = \begin{bmatrix} \mathcal{Q}_{[i],2}^{(2)} \\ \mathcal{Q}_{[i],2}^{(4)} \end{bmatrix}. \end{aligned} \quad (\text{A.8})$$





## Appendix B

### Complexity

In this chapter, denote  $\mathfrak{Y} = \begin{bmatrix} Y_{[1]} & Y_{[2]} & \dots & Y_{[N]} \end{bmatrix}$ ,  $I$  as the number of iteration, and  $I_s$  as the total number of iteration in finding step size ( $I_s \geq I$ ). Here we assume  $m \gg Nl$  and  $l = k$ . Note that the following tables only show the leading terms. The complexity of BLAS/LAPACK routines are referred from LAPACK Working Note 41 [19]. In the tables, we use *#flops* for the number of floating-point operations, *#words* for the number of data communication, and *#messages* for the number of MPI calls during communication.

## B.1 Canonical Methods

Table B.1: Complexity of Canonical Methods

Name	Routine	#flops
Full SVD		$12nm^2 + 5\frac{1}{3}m^3$
$U\Sigma V^T \leftarrow \text{svd}(A)$	GEBRD	$4nm^2 - \frac{4}{3}m^3$
	ORGBR	$\frac{4}{3}m^3$
		$2nm^2 - \frac{2}{3}m^3$
	BDSQR	$6m^3$
		$6nm^2$
Full SVD with QR ( $m \ll n$ )		$6nm^2 + 16m^3$
$LQ \leftarrow \text{qr}(A)$	GELQF	$2nm^2 - \frac{2}{3}m^3$
	ORGLQ	$2nm^2 - \frac{2}{3}m^3$
$U\Sigma W^T \leftarrow \text{svd}(L)$	GEBRD	$\frac{8}{3}m^3$
	ORGBR	$\frac{4}{3}m^3 + \frac{4}{3}m^3$
	BDSQR	$6m^3 + 6m^3$
$V \leftarrow WQ^T$	GEMM	$2nm^2$
Randomized SVD (Sketch $l$ Columns)		$4nml + 12nl^2 + 6ml^2 + Rnl + 4l^3$
Algorithm I-1 Gaussian Projection ( $q = 0, l$ columns)		$2nml + Rnl$
$Q \leftarrow \text{orth}(Y)$	GEQRF	$2ml^2 - \frac{2}{3}l^3$
	ORGQR	$2ml^2 - \frac{2}{3}l^3$
Algorithm IV-1 Canonical SVD		$2nml + 12nl^2 + 2ml^2 + 5\frac{1}{3}l^3$
Randomized SVD (Sketch $Nl$ Columns)		$(2N + 2)nml + 12nl^2 + (4N^2 + 2N + 2)ml^2 + RNnl + \left(8\frac{2}{3}N^3 + 5\frac{1}{3}\right)l^3$
Algorithm I-1 Gaussian Projection ( $q = 0, Nl$ columns)		$2Nnml + RNnl$
$Q \leftarrow \text{orth}_l(\mathfrak{Y})$	SVD with QR (see Appendix B.1.1)	$4N^2ml^2 + 2Nml^2 + 8\frac{2}{3}N^3l^3$
Algorithm IV-1 Canonical SVD		$2nml + 12nl^2 + 2ml^2 + 5\frac{1}{3}l^3$

## B.1.1 $\mathfrak{H}$ Orthogonalization

Table B.2: Complexity of  $\mathfrak{H}$  Orthogonalization

Name	Routine	#flops
Canonical SVD		$12N^2ml^2 - 3N^3l^3$
$Q \leftarrow \text{orth}_{\text{svd}}(\mathfrak{H})$	GEBRD	$4N^2ml^2 - \frac{4}{3}N^3l^3$
	ORGBR	$2N^2ml^2 - \frac{2}{3}N^3l^3$
	BDSQR	$6N^2ml^2$
SVD with QR		$4N^2ml^2 + 2Nml^2 + 8\frac{2}{3}N^3l^3$
$\tilde{Q}R \leftarrow \text{qr}(\mathfrak{H})$	GEQRF	$2N^2ml^2 - \frac{2}{3}N^3l^3$
	ORGQR	$2N^2ml^2 - \frac{2}{3}N^3l^3$
$U \leftarrow \text{orth}_{\text{svd}}(R)$	GEBRD	$\frac{8}{3}N^3l^3$
	ORGBR	$\frac{4}{3}N^3l^3$
	BDSQR	$6N^3l^3$
$Q \leftarrow \tilde{Q}U_l$	GEMM	$2Nml^2$
Gramian		$N^2ml^2 + 2Nml^2 + 9\frac{2}{3}Nl^3$
$M \leftarrow \mathfrak{H}^T \mathfrak{H}$	SYRK	$N^2ml^2$
$W S^2 W^T \leftarrow \text{eig}(M)$	SYTRD	$\frac{4}{3}N^3l^3$
	ORGTR	$\frac{4}{3}N^3l^3$
	STEQR	$7N^3l^3$
$Q \leftarrow \mathfrak{H} W_l S_l^{-1}$	DIASM	-
	GEMM	$2Nml^2$



## B.2 Sequential Algorithms

### B.2.1 Stage I: Sketching

Table B.3: Complexity of Sketching

Name	Routine	#flops
<b>Algorithm I-1</b> Gaussian Projection		$(4q + 2)Nml$
$Y_{[i]} \leftarrow (AA^T)^q A Q_{[i]}$	GEMM $\times(2q + 1)N$	$(4q + 2)Nml$



### B.2.2 Stage II: Orthogonalization

Table B.4: Complexity of Orthogonalization

Name	Routine	#flops
<b>Algorithm II-1</b> Canonical QR		$4Nml^2 - \frac{4}{3}Nl^3$
$Q_{[i]} \leftarrow \text{orth}_{\text{qr}}(Y_{[i]})$	GEQRF $\times N$	$2Nml^2 - \frac{2}{3}Nl^3$
	ORGQR $\times N$	$2Nml^2 - \frac{2}{3}Nl^3$
<b>Algorithm II-1</b> Canonical SVD		$12Nml^2 - 2Nl^3$
$Q_{[i]} \leftarrow \text{orth}_{\text{svd}}(Y_{[i]})$	GEBRD $\times N$	$4Nml^2 - \frac{4}{3}Nl^3$
	QRGBR $\times N$	$2Nml^2 - \frac{2}{3}Nl^3$
	BDSQR $\times N$	$6Nml^2$
<b>Algorithm II-1</b> SVD with QR		$6Nml^2 + 8\frac{2}{3}Nl^3$
$\tilde{Q}_{[i]} R_{[i]} \leftarrow \text{qr}(Y_{[i]})$	GEQRF $\times N$	$2Nml^2 - \frac{2}{3}Nl^3$
	ORGQR $\times N$	$2Nml^2 - \frac{2}{3}Nl^3$
$\tilde{R}_{[i]} \leftarrow \text{orth}_{\text{svd}}(R_{[i]})$	GEBRD $\times N$	$\frac{8}{3}Nl^3$
	ORGBR $\times N$	$\frac{4}{3}Nl^3$
	BDSQR $\times N$	$6Nl^3$
$Q_{[i]} \leftarrow \tilde{Q}_{[i]} \tilde{R}_{[i]}$	GEMM $\times N$	$2Nml^2$
<b>Algorithm II-2</b> Gramian		$3Nml^2 + 9\frac{2}{3}Nl^3$
$M_{[i]} \leftarrow Y_{[i]}^T Y_{[i]}$	SYRK $\times N$	$Nml^2$
$W_{[i]} S_{[i]}^2 W_{[i]}^T \leftarrow \text{eig}(M_{[i]})$	SYTRD $\times N$	$\frac{4}{3}Nl^3$
	ORGTR $\times N$	$\frac{4}{3}Nl^3$
	STEQR $\times N$	$7Nl^3$
$Q_{[i]} \leftarrow Y_{[i]} W_{[i]} S_{[i]}^{-1}$	DIASM $\times N$	-
	GEMM $\times N$	$2Nml^2$

## B.2.3 Stage III: Integration

Table B.5: Complexity of Integration

	Name	Routine	#flops	
	Algorithm III-4 Kolmogorov-Nagumo		$(4N + 4)ml^2 I + 2Nml^2 + (7N + 15\frac{2}{3})l^3 I$	
	$B_c \leftarrow \mathfrak{Q}^\top Q_c$	GEMM	$2Nml^2$	
ITERATION $\times I$	$G_c \leftarrow \frac{1}{N} \mathfrak{Q} B_c$	GEMM	$2Nml^2$	
	$B_c^g \leftarrow \mathfrak{Q}^\top G_c$	GEMM	$2Nml^2$	
	$D_c \leftarrow B_c^\top B_c$	GEMM	$2Nl^3$	
	$D_c^g \leftarrow B_c^\top B_c^g$	GEMMT	$Nl^3$	
	$\Xi \leftarrow D_c^g - D_c^2$	SYRK	$l^3$	
	eig( $\Xi$ )	SYTRD	$\frac{4}{3}l^3$	
		ORGTR	$\frac{4}{3}l^3$	
		STEQR	$7l^3$	
	Compute $C$	GEMM	$2l^3$	
	Compute $C^{-1}$	SYRK	$l^3$	
	$F_c \leftarrow C - D_c C^{-1}$	SYMM	$2l^3$	
	$Q_+ \leftarrow Q_c F_c + G_c F_c^g$	GEMM $\times 2$	$4ml^2$	
$B_+ \leftarrow B_c F_c + B_c^g F_c^g$	GEMM $\times 2$	$4Nl^3$		
	Algorithm III-5 Wen-Yin		$(4N + 6)ml^2 I + (4N + 2)ml^2 + (4N + 20)l^3 I_s + 4Nl^3 I + 2Nl^3$	
	$B_c \leftarrow \mathfrak{Q}^\top Q_c$	GEMM	$2Nml^2$	
	$D_c \leftarrow B_c^\top B_c$	GEMM	$2Nl^3$	
	$G_c \leftarrow \frac{1}{N} \mathfrak{Q} B_c$	GEMM	$2Nml^2$	
	$X_c \leftarrow G_c - Q_c D_c$	GEMM	$2ml^2$	
ITERATION $\times I$	LOOP $\times I_s$	$B_c^g \leftarrow \mathfrak{Q}^\top G_c$	GEMM	$2Nml^2$
		$D_c^g \leftarrow B_c^\top B_c^g$	GEMM	$2Nl^3$
	Compute $C^{-1}$	GETRF	$5\frac{1}{3}l^3$	
		GETRI	$10\frac{2}{3}l^3$	
	$F_c \leftarrow C_{22}^{-1} D_c - C_{21}^{-1} + I$	GEMM	$2l^3$	
	$F_c^g \leftarrow C_{12}^{-1} D_c - C_{11}^{-1}$	GEMM	$2l^3$	
	$B_+ \leftarrow B_c F_c + B_c^g F_c^g$	GEMM $\times 2$	$4Nl^3$	
	$D_+ \leftarrow B_+^\top B_+$	GEMM	$2Nl^3$	
	$Q_+ \leftarrow Q_c F_c + G_c F_c^g$	GEMM $\times 2$	$4ml^2$	
	$G_+ \leftarrow \frac{1}{N} \mathfrak{Q} B_+$	GEMM	$2Nml^2$	
	$X_+ \leftarrow G_+ - Q_+ D_+$	GEMM	$2ml^2$	
	tr( $\Delta_1^\top \Delta_1$ ) and tr( $\Delta_1^\top \Delta_2$ )	MINUS $\times 2$	-	
DOT $\times 2$		-		

Table B.6: Complexity of Integration (Conti.)

	Name	Routine	#flops
	<b>Algorithm III-3</b> Hierarchical Reduction		$6Nml^2 + 17\frac{1}{3}Nl^3$
LOOP $\times N$	$\mathbf{B}_{[i,i+h]} \leftarrow \mathbf{Q}_{[i]}^\top \mathbf{Q}_{[i+h]}$	GEMM	$2ml^2$
	$\mathbf{WST}^\top \leftarrow \text{svd}(\mathbf{B}_{[i,i+h]})$	GEBRD	$\frac{8}{3}l^3$
		ORGBR $\times 2$	$\frac{8}{3}l^3$
		BDSQR	$12l^3$
	$\mathbf{Q}_{[i]} \leftarrow \mathbf{Q}_{[i]} \mathbf{W} + \mathbf{Q}_{[i+h]} \mathbf{T}$	GEMM $\times 2$	$4ml^2$
$\mathbf{Q}_{[i]} \leftarrow \mathbf{Q}_{[i]} (2(\mathbf{I} + \mathbf{S}))^{-1/2}$	DIASM	-	
	<b>Algorithm III-6</b> Kolmogorov-Nagumo (Gramian)		$(N^2 + 2N + 2)ml^2 + (2N^2 + 11N + 17\frac{2}{3})l^3 I$
	$\mathfrak{B} \leftarrow \sum_{j=1}^p \mathfrak{Q}^\top \mathfrak{Q}$	SYRK	$N^2 ml^2$
	$\mathbf{B}_c \leftarrow \sum_{j=1}^p \mathfrak{Q}^\top \mathbf{Q}_{\text{init}}^*$	COPY	-
ITERATION $\times I$	$\mathbf{B}_c^g \leftarrow \frac{1}{N} \mathfrak{B} \mathbf{B}_c$	SYMM	$2N^2 l^3$
	$\mathbf{D}_c \leftarrow \mathbf{B}_c^\top \mathbf{B}_c$	GEMM	$2Nl^3$
	$\mathbf{D}_c^g \leftarrow \mathbf{B}_c^\top \mathbf{B}_c^g$	GEMMT	$Nl^3$
	$\Xi \leftarrow \mathbf{D}_c^g - \mathbf{D}_c^2$	SYRK	$l^3$
	eig( $\Xi$ )	SYTRD	$\frac{4}{3}l^3$
		ORGTR	$\frac{4}{3}l^3$
		STEQR	$7l^3$
	Compute $\mathbf{C}$	GEMM	$2l^3$
	Compute $\mathbf{C}^{-1}$	SYRK	$l^3$
	$\mathbf{F}_c \leftarrow \mathbf{C} - \mathbf{D}_c \mathbf{C}^{-1}$	SYMM	$2l^3$
	$\mathbf{E}_c \leftarrow \frac{1}{N} \mathbf{B}_c \mathbf{F}_c^g$	GEMM	$2Nl^3$
	$\tilde{\mathbf{F}} \leftarrow \tilde{\mathbf{F}} \mathbf{F}_c$	GEMM	$2l^3$
	$\tilde{\mathbf{E}} \leftarrow \tilde{\mathbf{E}} \mathbf{F}_c + \mathbf{E}_c$	GEMM	$2Nl^3$
	$\mathbf{B}_+ \leftarrow \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g$	GEMM $\times 2$	$4Nl^3$
$\bar{\mathbf{Q}} \leftarrow \mathbf{Q}_{\text{init}} \tilde{\mathbf{F}} + \mathfrak{Q} \tilde{\mathbf{E}}$	GEMM	$2ml^2$	
	GEMM	$2Nml^2$	

\* Requires  $2Nml^2$  flops using GEMM if  $\mathbf{Q}_{\text{init}} \neq \mathbf{Q}_{[1]}$ .



Table B.7: Complexity of Integration (Conti.)

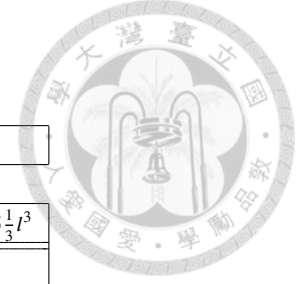
	Name	Routine	#flops
	Algorithm III-7 Wen-Yin (Gramian)		$(N^2 + 2N + 2) ml^2$ $+ (4N^2 + 11N + 4) l^3 I$ $+ (4N + 20) l^3 I_s + 2Nl^3$
	$\mathfrak{B} \leftarrow \sum_{j=1}^P \mathfrak{Q}^\top \mathfrak{Q}$	SYRK	$N^2 ml^2$
	$B_c \leftarrow \sum_{j=1}^P \mathfrak{Q}^\top Q_{\text{init}}^\dagger$	COPY	-
	$D_c \leftarrow B_c^\top B_c$	GEMM	$2Nl^3$
	$B_c^g \leftarrow \frac{1}{N} \mathfrak{B} B_c$	SYMM	$2N^2 l^3$
	$D_c^g \leftarrow B_c^\top B_c^g$	GEMMT	$Nl^3$
LOOP $\times l^3$	Compute $C^{-1}$	GETRF	$5\frac{1}{3} l^3$
		GETRI	$10\frac{2}{3} l^3$
	$F_c \leftarrow C_{22}^{-1} D_c - C_{21}^{-1} + I$	GEMM	$2l^3$
	$F_c^g \leftarrow C_{12}^{-1} D_c - C_{11}^{-1}$	GEMM	$2l^3$
	$B_+ \leftarrow B_c F_c + B_c^g F_c^g$	GEMM $\times 2$	$4Nl^3$
	$E_c \leftarrow \frac{1}{N} B_c F_c^g$	GEMM	$2Nl^3$
	$\tilde{F} \leftarrow \tilde{F} F_c$	GEMM	$2l^3$
	$\tilde{E} \leftarrow \tilde{E} F_c + E_c$	GEMM	$2Nl^3$
	$\Phi_2 \leftarrow D_c - F_c D_+$	GEMM	$2l^3$
	$Y_2 \leftarrow \frac{1}{N} (B_+ - B_c) - E_c D_+$	GEMM	$2Nl^3$
ITERATION $\times l$	Compute $B_c^\top Y_1$ and $B_c^\top Y_2$	GEMM $\times 2$	$4Nl^3$
	Compute $\mathfrak{B} Y_1$ or $\mathfrak{B} Y_2$	SYMM	$2N^2 l^3$
	$\sum_{j=1}^P \mathbf{A}_1^\top \mathbf{A}_1$ and $\sum_{j=1}^P \mathbf{A}_1^\top \mathbf{A}_2$	DOT $\times 5$	-
		DOT $\times 2$	-
	$\bar{Q}^{(j)} \leftarrow Q_{\text{init}}^{(j)} \tilde{F} + \mathfrak{Q}^{(j)} \tilde{E}$	GEMM	$2ml^2$
		GEMM	$2Nml^2$

$\dagger$  Requires  $2Nml^2$  flops using GEMM if  $Q_{\text{init}} \neq Q_{[1]}$ .

## B.2.4 Stage IV: Postprocessing

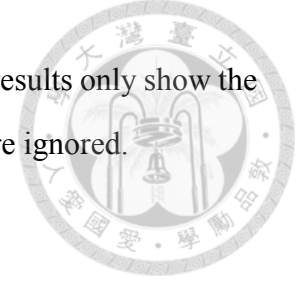
Table B.8: Complexity of Postprocessing

Name	Routine	#flops
<b>Algorithm IV-1 Canonical SVD</b>		$2nml + 12nl^2 + 2ml^2 + 5\frac{1}{3}l^3$
$Q^T A$	GEMM	$2nml$
$W \Sigma V^T \leftarrow \text{svd}(Q^T A)$	GEBRD	$4nl^2 - \frac{4}{3}l^3$
	ORGBR $W$	$\frac{4}{3}l^3$
	$V$	$2nl^2 - \frac{2}{3}l^3$
	BDSQR $W$	$6l^3$
$V$	$6nl^2$	
$U \leftarrow QW$	GEMM	$2ml^2$
<b>Algorithm IV-1 SVD with QR</b>		$2nml + 6nl^2 + 2ml^2 + 16l^3$
$Z \leftarrow A^T Q$	GEMM	$2nml$
$\tilde{Q}R \leftarrow \text{qr}(Z)$	GEQRF	$2nl^2 - \frac{2}{3}l^3$
	ORGQR	$2nl^2 - \frac{2}{3}l^3$
$W \Sigma \tilde{V}^T \leftarrow \text{svd}(R^T)$	GEBRD	$\frac{8}{3}l^3$
	ORGBR $W$ & $\tilde{V}$	$\frac{4}{3}l^3 + \frac{4}{3}l^3$
	ORGBR $W$ & $\tilde{V}$	$6l^3 + 6l^3$
$U \leftarrow QW$	GEMM	$2ml^2$
$V \leftarrow \tilde{Q}\tilde{V}$	GEMM	$2nl^2$
<b>Algorithm IV-2 Gramian</b>		$2nml + 3nl^2 + 2ml^2 + 9\frac{2}{3}l^3$
$Z \leftarrow A^T Q$	GEMM	$2nml$
$M \leftarrow Z^T Z$	SYRK	$nl^2$
$W \Sigma^2 W^T \leftarrow \text{eig}(M)$	SYTRD	$\frac{4}{3}l^3$
	ORGTR	$\frac{4}{3}l^3$
	STEQR	$7l^3$
$U \leftarrow QW$	GEMM	$2ml^2$
$V \leftarrow ZW \Sigma^{-1}$	DIASM	$l^2$
	GEMM	$2nl^2$
<b>Algorithm IV-3 Symmetric</b>		$2m^2l + 3ml^2 + 9\frac{2}{3}l^3$
$Z \leftarrow A^T Q$	GEMM	$2m^2l$
$M \leftarrow Z^T Q$	GEMMT	$ml^2$
$W \Sigma^2 W^T \leftarrow \text{eig}(M)$	SYTRD	$\frac{4}{3}l^3$
	ORGTR	$\frac{4}{3}l^3$
	STEQR	$7l^3$
$U \leftarrow QW$	GEMM	$2ml^2$



## B.3 Parallel Algorithms

For parallel algorithms, instead of compute the total complexity, the results only show the complexity of each processor. The rearrangements between stages are ignored.



### B.3.1 Stage I: Sketching (Parallelism)

Table B.9: Complexity of Sketching (Naïve Parallelism)

Name	Routine	#flops	#words	#messages
Algorithm I-1 Gaussian Projection		$(4q + 2) \frac{N}{P} nml$	-	-
$Y_{[i]} \leftarrow (AA^T)^q A Q_{[i]}$	GEMM $\times (2q + 1) \frac{N}{P}$	$(4q + 2) \frac{N}{P} nml$	-	-

Table B.10: Complexity of Sketching (Block-Row Parallelism)

Name	Routine	#flops	#words	#messages
Algorithm I-2 Gaussian Projection		$(4q + 2) \frac{N}{P} nml$	$q(\log P) \frac{N}{P} ml$	$q + 1$
Random seed	Bcast	-	1	1
$Y_{[i]}^{(j)} \leftarrow A^{(j)} Q_{[i]}$	GEMM $\times N$	$2 \frac{N}{P} nml$	-	-
$\tilde{Y}_{[i]} \leftarrow \sum_{j=1}^P A^{(j)T} Y_{[i]}^{(j)}$	GEMM $\times Nq$	$2q \frac{N}{P} nml$	-	-
	AllReduce $\times q$	-	$q(\log P) \frac{N}{P} ml$	$q$
$Y_{[i]}^{(j)} \leftarrow A^{(j)} \tilde{Y}_{[i]}$	GEMM $\times Nq$	$2q \frac{N}{P} nml$	-	-

Table B.11: Complexity of Sketching (Block-Column Parallelism)

Name	Routine	#flops	#words	#messages
Algorithm I-3 Gaussian Projection ( $q = 0$ )		$2 \frac{N}{P} nml$	$Nml$	$\log P$
$A^{(j)} Q_{[i]}^{(j)}$	GEMM	$2 \frac{N}{P} nml$	-	-
$Y_{[i]}^{(j)} \leftarrow (A Q_{[i]})^{(j)}$	ReduceScatterBlock	-	$Nml$	$\log P$

### B.3.2 Stage II: Orthogonalization (Parallelism)

Table B.12: Complexity of Orthogonalization (Naïve Parallelism)

Name	Routine	#flops	#words	#messages
<b>Algorithm II-1</b> Canonical QR		$4\frac{N}{P}ml^2 - \frac{4}{3}\frac{N}{P}l^3$	-	-
$Q_{[i]} \leftarrow \text{orth}_{\text{qr}}(Y_{[i]})$	GEQRF $\times \frac{N}{P}$	$2\frac{N}{P}ml^2 - \frac{2}{3}\frac{N}{P}l^3$	-	-
	ORGQR $\times \frac{N}{P}$	$2\frac{N}{P}ml^2 - \frac{2}{3}\frac{N}{P}l^3$	-	-
<b>Algorithm II-1</b> Canonical SVD		$12\frac{N}{P}ml^2 - 2\frac{N}{P}l^3$	-	-
$Q_{[i]} \leftarrow \text{orth}_{\text{svd}}(Y_{[i]})$	GEBRD $\times \frac{N}{P}$	$4\frac{N}{P}ml^2 - \frac{4}{3}\frac{N}{P}l^3$	-	-
	QRGBR $\times \frac{N}{P}$	$2\frac{N}{P}ml^2 - \frac{2}{3}\frac{N}{P}l^3$	-	-
	BDSQR $\times \frac{N}{P}$	$6\frac{N}{P}ml^2$	-	-
<b>Algorithm II-1</b> SVD with QR		$6\frac{N}{P}ml^2 + 8\frac{2}{3}\frac{N}{P}l^3$	-	-
$\tilde{Q}_{[i]}R \leftarrow \text{qr}(Y_{[i]})$	GEQRF $\times \frac{N}{P}$	$2\frac{N}{P}ml^2 - \frac{2}{3}\frac{N}{P}l^3$	-	-
	ORGQR $\times \frac{N}{P}$	$2\frac{N}{P}ml^2 - \frac{2}{3}\frac{N}{P}l^3$	-	-
$\tilde{R}_{[i]} \leftarrow \text{orth}_{\text{svd}}(R)$	GEBRD $\times \frac{N}{P}$	$\frac{8}{3}\frac{N}{P}l^3$	-	-
	QRGBR $\times \frac{N}{P}$	$\frac{4}{3}\frac{N}{P}l^3$	-	-
	BDSQR $\times \frac{N}{P}$	$6\frac{N}{P}l^3$	-	-
$Q_{[i]} \leftarrow \tilde{Q}_{[i]}\tilde{R}_{[i]}$	GEMM $\times \frac{N}{P}$	$2\frac{N}{P}ml^2$	-	-

Table B.13: Complexity of Orthogonalization (Block-Row Parallelism)

Name	Routine	#flops	#words	#messages
<b>Algorithm II-3</b> Gramian		$3\frac{N}{P}ml^2 + 9\frac{2}{3}Nl^3$	$(\log P)Nl^2$	$\log P$
$M_{[i]} \leftarrow \sum_{j=1}^P Y_{[i]}^{(j)\top} Y_{[i]}^{(j)}$	SYRK $\times N$	$\frac{N}{P}ml^2$	-	-
	AllReduce	-	$(\log P)Nl^2$	$\log P$
$W_{[i]}S_{[i]}^2W_{[i]}^\top \leftarrow \text{eig}(M_{[i]})$	SYTRD $\times N$	$\frac{4}{3}Nl^3$	-	-
	ORGTR $\times N$	$\frac{4}{3}Nl^3$	-	-
	STEQR $\times N$	$7Nl^3$	-	-
$Q_{[i]}^{(j)} \leftarrow Y_{[i]}W_{[i]}S_{[i]}^{-1}$	SCAL $\times Nl$	-	-	-
	GEMM $\times N$	$2\frac{N}{P}ml^2$	-	-
<b>Algorithm II-4</b> TSQR		$4\frac{N}{P}ml^2 + 7\frac{1}{3}(\log P)Nl^3 - 5\frac{1}{3}Nl^3$	$(\log P)Nl^2$	$\log P$
$Q_{[i],0}^{(j)}R_{[i],0}^{(j)} \leftarrow \text{qr}(Y_{[i]}^{(j)})$	GEQRF $\times N$	$2\frac{N}{P}ml^2 - \frac{2}{3}Nl^3$	-	-
LOOP	Combine $\tilde{Y}_{[i],t-1}$	Sendrecv $\times N \log P$	-	$(\log P)Nl^2$
	$\tilde{Q}_{[i],t}R_{[i],t} \leftarrow \text{qr}(\tilde{Y}_{[i],t-1})$	GEQRF $\times N \log P$	$3\frac{1}{3}(\log P)Nl^3$	-
$Q_{[i]}^{(j)} \leftarrow Q_{[i],0}^{(j)} \cdots Q_{[i],H}^{(j)}$	ORGQR $\times N$	$4Nl^3 - \frac{2}{3}Nl^3$	-	-
	ORMQR $\times N(\log P - 1)$	$4(\log P - 1)Nl^3$	-	-
	ORMQR $\times N$	$2\frac{N}{P}ml^2$	-	-

### B.3.3 Stage III: Integration (Parallelism)

Table B.14: Complexity of Integration (Naïve Parallelism)

	Name	Routine	#flops	#words	#messages
	Algorithm III-8 Kolmogorov-Nagumo		$\left(6\frac{N}{P} + (\log P)\frac{N}{P} + 4\right)ml^2I$ $+ \left(\frac{N}{P} + 11\frac{2}{3}\right)l^3I$	$(\log P)\frac{N}{P}mlI$	$(\log P)I$
ITERATION $\times I$	$B_{[i],c} \leftarrow Q_{[i]}^T Q_c$	GEMM $\times \frac{N}{P}$	$2\frac{N}{P}ml^2$	-	-
	$B_{[i]}^T B_{[i]}$	SYRK $\times \frac{N}{P}$	$\frac{N}{P}l^3$	-	-
	$X_{[i]} \leftarrow \frac{1}{N} \left( Q_{[i]} B_{[i]} - Q_c B_{[i]}^T B_{[i]} \right)$	GEMM $\times \frac{N}{P}$	$2\frac{N}{P}ml^2$	-	-
		SYMM $\times \frac{N}{P}$	$2\frac{N}{P}ml^2$	-	-
	$X \leftarrow \sum_{i=1}^N X_{[i]}$	AllReduce	-	$(\log P)\frac{N}{P}ml$	$\log P$
	$\Xi \leftarrow X^T X$	SYRK	$ml^2$	-	-
	eig( $\Xi$ )	SYTRD	$\frac{4}{3}l^3$	-	-
		ORGTR	$\frac{4}{3}l^3$	-	-
		STEQR	$7l^3$	-	-
	Compute $C$ and $C^{-1}$	SYRK $\times 2$	$2l^3$	-	-
$Q_c \leftarrow Q_c C + X C^{-1}$	SYMM $\times 2$	$4ml^2$	-	-	
	Algorithm III-3 Hierarchical Reduction		$6(\log N)ml^2 + 17\frac{1}{3}(\log N)l^3$	$(\log N)ml$	$N^\ddagger$
LOOP $\times \log N$	$B_{[i,i+h]} \leftarrow Q_{[i]}^T Q_{[i+h]}$	Send & Receive	-	$ml$	1
		GEMM	$2ml^2$	-	-
	$WST^T \leftarrow \text{svd}(B_{[i,i+h]})$	GEBRD	$\frac{8}{3}l^3$	-	-
		ORGBR $\times 2$	$\frac{8}{3}l^3$	-	-
		BDSQR	$12l^3$	-	-
	$Q_{[i]} \leftarrow Q_{[i]}W + Q_{[i+h]}T$	GEMM $\times 2$	$4ml^2$	-	-
$Q_{[i]} \leftarrow Q_{[i]}(2(I+S))^{-1/2}$	DIASM	-	-	-	

$\ddagger$ Each  $Q_{[i]}$  (except  $Q_{[1]}$ ) should be transferred.

Table B.15: Complexity of Integration (Block-Row Parallelism)

	Name	Routine	#flops	#words	#messages
	Algorithm III-10 Kolmogorov-Nagumo		$\left(4\frac{N}{P} + 4\frac{1}{P}\right)ml^2I + 2\frac{1}{P}ml^2 + \left(7N + 15\frac{2}{3}\right)l^3I$	$(\log P)Nl^2(I + 1)$	$(\log P)(I + 1)$
	$\mathbf{B}_c \leftarrow \sum_{j=1}^P \mathbf{\Omega}^{(j)\top} \mathbf{Q}_c^{(j)}$	GEMM	$2\frac{N}{P}ml^2$	-	-
		AllReduce	-	$(\log P)Nl^2$	$\log P$
ITERATION $\times I$	$\mathbf{G}_c^{(j)} \leftarrow \frac{1}{N}\mathbf{\Omega}^{(j)}\mathbf{B}_c$	GEMM	$2\frac{N}{P}ml^2$	-	-
	$\mathbf{B}_c^g \leftarrow \sum_{j=1}^P \mathbf{\Omega}^{(j)\top} \mathbf{G}_c^{(j)}$	GEMM	$2\frac{N}{P}ml^2$	-	-
		AllReduce	-	$(\log P)Nl^2$	$\log P$
	$\mathbf{D}_c \leftarrow \mathbf{B}_c^\top \mathbf{B}_c$	GEMM	$2Nl^3$	-	-
	$\mathbf{D}_c^g \leftarrow \mathbf{B}_c^\top \mathbf{B}_c^g$	GEMMT	$Nl^3$	-	-
	$\mathbf{\Xi} \leftarrow \mathbf{D}_c^g - \mathbf{D}_c^2$	SYRK	$l^3$	-	-
	eig( $\mathbf{\Xi}$ )	SYTRD	$\frac{4}{3}l^3$	-	-
		ORGTR	$\frac{4}{3}l^3$	-	-
		STEQR	$7l^3$	-	-
	Compute $\mathbf{C}$	GEMM	$2l^3$	-	-
	Compute $\mathbf{C}^{-1}$	SYRK	$l^3$	-	-
	$\mathbf{F}_c \leftarrow \mathbf{C} - \mathbf{D}_c\mathbf{C}^{-1}$	SYMM	$2l^3$	-	-
	$\mathbf{Q}_+^{(j)} \leftarrow \mathbf{Q}_c^{(j)}\mathbf{F}_c + \mathbf{G}_c^{(j)}\mathbf{F}_c^g$	GEMM $\times 2$	$4\frac{1}{P}ml^2$	-	-
$\mathbf{B}_+ \leftarrow \mathbf{B}_c\mathbf{F}_c + \mathbf{B}_c^g\mathbf{F}_c^g$	GEMM $\times 2$	$4Nl^3$	-	-	

Table B.16: Complexity of Integration (Block-Row Parallelism) (Conti.)

	Name	Routine	#flops	#words	#messages		
	Algorithm III-11 Wen-Yin		$\left(4\frac{N}{P} + 6\frac{1}{P}\right)ml^2I$ $+ \left(4\frac{N}{P} + 2\frac{1}{P}\right)ml^2$ $+ (4N + 20)l^3I_s$ $+ 4Nl^3I + 2Nl^3$	$(\log P)Nl^2(I + 1)$	$(\log P)(2I + 1)$		
	$B_c \leftarrow \sum_{j=1}^P \mathfrak{Q}^{(j)\top} Q_c^{(j)}$	GEMM	$2\frac{N}{P}ml^2$	-	-		
		AllReduce	-	$(\log P)Nl^2$	$\log P$		
		$D_c \leftarrow B_c^\top B_c$	GEMM	$2Nl^3$	-	-	
		$G_c^{(j)} \leftarrow \frac{1}{N}\mathfrak{Q}^{(j)} B_c$	GEMM	$2\frac{N}{P}ml^2$	-	-	
	$X_c^{(j)} \leftarrow \frac{1}{N}G_c^{(j)} - Q_c^{(j)} D_c$	GEMM	$2\frac{1}{P}ml^2$	-	-		
	$B_c^g \leftarrow \sum_{j=1}^P \mathfrak{Q}^{(j)\top} G_c^{(j)}$	GEMM	$2\frac{N}{P}ml^2$	-	-		
		AllReduce	-	$(\log P)Nl^2$	$\log P$		
	$D_c^g \leftarrow B_c^\top B_c^g$	GEMM	$2Nl^3$	-	-		
		Compute $C^{-1}$	GETRF	$5\frac{1}{3}l^3$	-	-	
			GETRI	$10\frac{2}{3}l^3$	-	-	
	LOOP $\times I_s$	$F_c \leftarrow C_{22}^{-1} D_c - C_{21}^{-1} + I$	GEMM	$2l^3$	-	-	
		$F_c^g \leftarrow C_{12}^{-1} D_c - C_{11}^{-1}$	GEMM	$2l^3$	-	-	
		$B_+ \leftarrow B_c F_c + B_c^g F_c^g$	GEMM $\times 2$	$4Nl^3$	-	-	
		ITERATION $\times I$	$D_+ \leftarrow B_+^\top B_+$	GEMM	$2Nl^3$	-	-
			$Q_+^{(j)} \leftarrow Q_c^{(j)} F_c + G_c^{(j)} F_c^g$	GEMM $\times 2$	$4m\frac{1}{P}l^2$	-	-
	$G_+^{(j)} \leftarrow \frac{1}{N}\mathfrak{Q}^{(j)} B_+$		GEMM	$2\frac{N}{P}ml^2$	-	-	
	$X_+^{(j)} \leftarrow G_+^{(j)} - Q_+^{(j)} D_+$		GEMM	$2ml^2$	-	-	
	$\sum_{j=1}^P \text{tr}(\Delta_1^{(j)\top} \Delta_1^{(j)})$ and $\sum_{j=1}^P \text{tr}(\Delta_1^{(j)\top} \Delta_2^{(j)})$	MINUS $\times 2$	-	-	-		
		DOT $\times 2$	-	-	-		
		AllReduce	-	$2 \log P$	$\log P$		
	Algorithm III-12 Hierarchical Reduction		$6\frac{N}{P}ml^2 + 17\frac{1}{3}Nl^3$	$(\log P)Nl^2$	$(\log P)N$		
	LOOP $\times N$	$B_{[i,i+h]} \leftarrow \sum_{j=1}^N Q_{[i]}^{(j)\top} Q_{[i+h]}^{(j)}$	GEMM	$2\frac{1}{P}ml^2$	-	-	
			AllReduce	-	$(\log P)l^2$	$\log P$	
		$WST^\top \leftarrow \text{svd}(B_{[i,i+h]})$	GEBRD	$\frac{8}{3}l^3$	-	-	
			ORGBR $\times 2$	$\frac{8}{3}l^3$	-	-	
			BDSQR	$12l^3$	-	-	
		$Q_{[i]} \leftarrow Q_{[i]}W + Q_{[i+h]}T$	GEMM $\times 2$	$4\frac{1}{P}ml^2$	-	-	
	$Q_{[i]} \leftarrow Q_{[i]}(2(I+S))^{-1/2}$	DIASM	-	-	-		

Table B.17: Complexity of Integration (Block-Row Parallelism) (Conti.)

	Name	Routine	#flops	#words	#messages
	Algorithm III-6 Kolmogorov-Nagumo (Gramian)		$\left(\frac{N^2}{P} + 2\frac{N}{P} + 2\frac{1}{P}\right)ml^2 + I\left(2N^2 + 11N + 17\frac{2}{3}\right)l^3$	$(\log P)N^2l^2$	$\log P$
	$\mathfrak{B} \leftarrow \sum_{j=1}^P \mathfrak{Q}^{(j)\top} \mathfrak{Q}^{(j)}$	SYRK	$\frac{N^2}{P}ml^2$	-	-
		AllReduce	-	$(\log P)N^2l^2$	$\log P$
	$\mathbf{B}_c \leftarrow \sum_{j=1}^P \mathfrak{Q}^{(j)\top} \mathbf{Q}_{\text{init}}^{(j)\S}$	COPY	-	-	-
ITERATION $\times I$	$\mathbf{B}_c^g \leftarrow \frac{1}{N} \mathfrak{B} \mathbf{B}_c$	SYMM	$2N^2l^3$	-	-
	$\mathbf{D}_c \leftarrow \mathbf{B}_c^\top \mathbf{B}_c$	GEMM	$2Nl^3$	-	-
	$\mathbf{D}_c^g \leftarrow \mathbf{B}_c^\top \mathbf{B}_c^g$	GEMMT	$Nl^3$	-	-
	$\mathbf{\Xi} \leftarrow \mathbf{D}_c^g - \mathbf{D}_c^2$	SYRK	$l^3$	-	-
	eig( $\mathbf{\Xi}$ )	SYTRD	$\frac{4}{3}l^3$	-	-
		ORGTR	$\frac{4}{3}l^3$	-	-
		STEQR	$7l^3$	-	-
	Compute $\mathbf{C}$	GEMM	$2l^3$	-	-
	Compute $\mathbf{C}^{-1}$	SYRK	$l^3$	-	-
	$\mathbf{F}_c \leftarrow \mathbf{C} - \mathbf{D}_c \mathbf{C}^{-1}$	SYMM	$2l^3$	-	-
	$\mathbf{E}_c \leftarrow \frac{1}{N} \mathbf{B}_c \mathbf{F}_c^g$	GEMM	$2Nl^3$	-	-
	$\tilde{\mathbf{F}} \leftarrow \tilde{\mathbf{F}} \mathbf{F}_c$	GEMM	$2l^3$	-	-
	$\tilde{\mathbf{E}} \leftarrow \tilde{\mathbf{E}} \mathbf{F}_c + \mathbf{E}_c$	GEMM	$2Nl^3$	-	-
	$\mathbf{B}_+ \leftarrow \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g$	GEMM $\times 2$	$4Nl^3$	-	-
$\overline{\mathbf{Q}}^{(j)} \leftarrow \mathbf{Q}_{\text{init}}^{(j)} \tilde{\mathbf{F}} + \mathfrak{Q}^{(j)} \tilde{\mathbf{E}}$	GEMM	$2\frac{1}{P}ml^2$	-	-	
	GEMM	$2\frac{N}{P}ml^2$	-	-	

$\S$ Requires  $2\frac{N}{P}ml^2$  flops without communication using GEMM and  $(\log P)Nl^2$  flops,  $(\log P)Nl^2$  words, and  $\log P$  messages using AllReduce if  $\mathbf{Q}_{\text{init}} \neq \mathbf{Q}_{[1]}$ .



Table B.18: Complexity of Integration (Block-Row Parallelism) (Conti.)

	Name	Routine	#flops	#words	#messages
	Algorithm III-7 Wen-Yin (Gramian)		$\left(\frac{N^2}{P} + 2\frac{N}{P} + 2\frac{1}{P}\right)ml^2$ $+ (4N^2 + 11N + 4)l^3I$ $+ (4N + 20)l^3I_s + 2Nl^3$	$(\log P)N^2l^2$	$\log P(I + 1)$
	$\mathfrak{B} \leftarrow \sum_{j=1}^P \mathfrak{Q}^{(j)\top} \mathfrak{Q}^{(j)}$	SYRK AllReduce	$\frac{N^2}{P}ml^2$ -	- $(\log P)N^2l^2$	- $\log P$
	$\mathbf{B}_c \leftarrow \sum_{j=1}^P \mathfrak{Q}^{(j)\top} \mathbf{Q}_{\text{init}}^{(j)\dagger}$	COPY	-	-	-
	$\mathbf{D}_c \leftarrow \mathbf{B}_c^\top \mathbf{B}_c$	GEMM	$2Nl^3$	-	-
	$\mathbf{B}_c^g \leftarrow \frac{1}{N} \mathfrak{B} \mathbf{B}_c$	SYMM	$2N^2l^3$	-	-
	$\mathbf{D}_c^g \leftarrow \mathbf{B}_c^\top \mathbf{B}_c^g$	GEMMT	$Nl^3$	-	-
LOOP $\times I_s$	Compute $\mathbf{C}^{-1}$	GETRF GETRI	$5\frac{1}{3}l^3$ $10\frac{2}{3}l^3$	-	-
	$\mathbf{F}_c \leftarrow \mathbf{C}_{22}^{-1} \mathbf{D}_c - \mathbf{C}_{21}^{-1} + \mathbf{I}$	GEMM	$2l^3$	-	-
	$\mathbf{F}_c^g \leftarrow \mathbf{C}_{12}^{-1} \mathbf{D}_c - \mathbf{C}_{11}^{-1}$	GEMM	$2l^3$	-	-
	$\mathbf{B}_+ \leftarrow \mathbf{B}_c \mathbf{F}_c + \mathbf{B}_c^g \mathbf{F}_c^g$	GEMM $\times 2$	$4Nl^3$	-	-
	$\mathbf{E}_c \leftarrow \frac{1}{N} \mathbf{B}_c \mathbf{F}_c^g$	GEMM	$2Nl^3$	-	-
	$\tilde{\mathbf{F}} \leftarrow \tilde{\mathbf{F}} \mathbf{F}_c$	GEMM	$2l^3$	-	-
	$\tilde{\mathbf{E}} \leftarrow \tilde{\mathbf{E}} \mathbf{F}_c + \mathbf{E}_c$	GEMM	$2Nl^3$	-	-
	$\Phi_2 \leftarrow \mathbf{D}_c - \mathbf{F}_c \mathbf{D}_+$	GEMM	$2l^3$	-	-
	$\mathbf{Y}_2 \leftarrow \frac{1}{N} (\mathbf{B}_+ - \mathbf{B}_c) - \mathbf{E}_c \mathbf{D}_+$	GEMM	$2Nl^3$	-	-
	Compute $\mathbf{B}_c^\top \mathbf{Y}_1$ and $\mathbf{B}_c^\top \mathbf{Y}_2$	GEMM $\times 2$	$4Nl^3$	-	-
ITERATION $\times I$	Compute $\mathfrak{B} \mathbf{Y}_1$ or $\mathfrak{B} \mathbf{Y}_2$	SYMM	$2N^2l^3$	-	-
	$\sum_{j=1}^P \text{tr}(\Delta_1^\top \Delta_1)$ and $\sum_{j=1}^P \text{tr}(\Delta_1^\top \Delta_2)$	DOT $\times 5$ DOT $\times 2$ AllReduce	- - -	- - $2 \log P$	- - $\log P$
	$\bar{\mathbf{Q}}^{(j)} \leftarrow \mathbf{Q}_{\text{init}}^{(j)} \tilde{\mathbf{F}} + \mathfrak{Q}^{(j)} \tilde{\mathbf{E}}$	GEMM GEMM	$2\frac{1}{P}ml^2$ $2\frac{N}{P}ml^2$	- -	- -

<sup>†</sup>Requires  $2\frac{N}{P}ml^2$  flops without communication using GEMM and  $(\log P)Nl^2$  flops,  $(\log P)Nl^2$  words, and  $\log P$  messages using AllReduce if  $\mathbf{Q}_{\text{init}} \neq \mathbf{Q}_{[1]}$ .

### B.3.4 Stage IV: Postprocessing (Parallelism)

Table B.19: Complexity of Postprocessing (Block-Row Parallelism)

Name	Routine	#flops	#words	#messages
<b>Algorithm IV-4 Gramian</b>		$2\frac{1}{P}nml + 3\frac{1}{P}nl^2 + 2\frac{1}{P}ml^2 + 9\frac{2}{3}l^3$	$2nl + ml + (\log P)l^2$	$4 \log P$
$A^{(j)\top} \overline{Q}^{(j)}$	GEMM	$2\frac{1}{P}nml$	-	-
$Z^{(j)} \leftarrow (A^\top \overline{Q})^{(j)}$	ReduceScatterBlock	-	$nl$	$\log P$
$M^{(j)} \leftarrow Z^{(j)\top} Z^{(j)}$	SYRK	$\frac{1}{P}nl^2$	-	-
$M \leftarrow \sum_{j=1}^P M^{(j)}$	AllReduce	-	$(\log P)l^2$	$\log P$
$W \Sigma^2 W^\top \leftarrow \text{eig}(M)$	SYTRD	$\frac{4}{3}l^3$	-	-
	ORGTR	$\frac{4}{3}l^3$	-	-
	STEQR	$7l^3$	-	-
$U^{(j)} \leftarrow \overline{Q}^{(j)} W$	GEMM	$2\frac{1}{P}ml^2$	-	-
$V^{(j)} \leftarrow Z^{(j)} W \Sigma^{-1}$	DIASM	-	-	-
	GEMM	$2\frac{1}{P}nl^2$	-	-
Gather $U$	Gather	-	$ml$	$\log P$
Gather $V$	Gather	-	$nl$	$\log P$
<b>Algorithm IV-5 Symmetric</b>		$2\frac{1}{P}m^2l + 3\frac{1}{P}ml^2 + 9\frac{2}{3}l^3$	$2ml + (\log P)l^2$	$3 \log P$
$A^{(j)\top} \overline{Q}^{(j)}$	GEMM	$2\frac{1}{P}m^2l$	-	-
$Z^{(j)} \leftarrow (A^\top \overline{Q})^{(j)}$	ReduceScatterBlock	-	$ml$	$\log P$
$M^{(j)} \leftarrow Z^{(j)\top} \overline{Q}^{(j)}$	GEMMT	$\frac{1}{P}ml^2$	-	-
$M \leftarrow \sum_{j=1}^P M^{(j)}$	AllReduce	-	$(\log P)l^2$	$\log P$
$W \Sigma^2 W^\top \leftarrow \text{eig}(M)$	SYTRD	$\frac{4}{3}l^3$	-	-
	ORGTR	$\frac{4}{3}l^3$	-	-
	STEQR	$7l^3$	-	-
$U^{(j)} \leftarrow \overline{Q}^{(j)} W$	GEMM	$2\frac{1}{P}ml^2$	-	-
Gather $U$	Gather	-	$ml$	$\log P$

Table B.20: Complexity of Postprocessing (Block-Row Parallelism) (Conti.)

	Name	Routine	#flops	#words	#messages
	<b>Algorithm IV-6 TSQR</b>		$2\frac{1}{P}nml + 4\frac{1}{P}nl^2 + 2\frac{1}{P}ml^2 + 7\frac{1}{3}(\log P)l^3 + 16l^3$	$2nl + ml + (\log P)l^2$	$4 \log P$
	$A^{(j)\top} \bar{Q}^{(j)}$	GEMM	$2\frac{1}{P}nml$	-	-
	$Z^{(j)} \leftarrow (A^\top \bar{Q})^{(j)}$	ReduceScatterBlock	-	$nl$	$\log P$
	$V_0^{(j)} R_0^{(j)} \leftarrow \text{qr}(Z_t^{(j)})$	GEQRF	$2\frac{1}{P}nl^2 - \frac{2}{3}l^3$	-	-
LOOP	Combine $\tilde{Z}_{t-1}^{(j)}$	Sendrecv $\times \log P$	-	$(\log P)l^2$	$\log P$
	$\tilde{V}_t^{(j)} R_t^{(j)} \leftarrow \text{qr}(\tilde{Z}_{t-1}^{(j)})$	GEQRF $\times \log P$	$3\frac{1}{3}(\log P)l^3$	-	-
$W \Sigma T^\top \leftarrow \text{svd}(R_H^\top)$	GEBRD		$\frac{8}{3}l^3$	-	-
	ORGBR $W$ & $T$		$\frac{8}{3}l^3$	-	-
	BDSQR $W$ & $T$		$12l^3$	-	-
$U^{(j)} \leftarrow \bar{Q}^{(j)} W$	GEMM		$2\frac{1}{P}ml^2$	-	-
$V^{(j)} \leftarrow V_0^{(j)} \dots V_H^{(j)} T$	ORMQR $\times \log P$		$4(\log P)l^3$	-	-
	ORMQR		$2\frac{1}{P}nl^2$	-	-
Gather $U$	Gather		-	$ml$	$\log P$
Gather $V$	Gather		-	$nl$	$\log P$

Table B.21: Complexity of Postprocessing (Block-Column Parallelism)

Name	Routine	#flops	#words	#messages
<b>Algorithm IV-7 Gramian</b>		$2\frac{1}{P}nml + 3\frac{1}{P}nl^2 + 2ml^2 + 9\frac{2}{3}l^3$	$nl + ml + (\log P)l^2$	$3 \log P$
Gather $\bar{Q}$	AllGather	-	$ml$	$\log P$
$Z^{(j)} \leftarrow A^{(j)\top} \bar{Q}$	GEMM	$2\frac{1}{P}nml$	-	-
$M^{(j)} \leftarrow Z^{(j)\top} Z^{(j)}$	SYRK	$\frac{1}{P}nl^2$	-	-
$M \leftarrow \sum_{j=1}^P M^{(j)}$	AllReduce	-	$(\log P)l^2$	$\log P$
$W \Sigma^2 W^\top \leftarrow \text{eig}(M)$	SYTRD	$\frac{4}{3}l^3$	-	-
	ORGTR	$\frac{4}{3}l^3$	-	-
	STEQR	$7l^3$	-	-
$U \leftarrow \bar{Q}W$	GEMM	$2ml^2$	-	-
$V^{(j)} \leftarrow Z^{(j)}W \Sigma^{-1}$	DIASM	-	-	-
	GEMM	$2\frac{1}{P}nl^2$	-	-
Gather $V$	Gather	-	$nl$	$\log P$
<b>Algorithm IV-9 Symmetric</b>		$2\frac{1}{P}m^2l + \left(2 + \frac{1}{P}\right)ml^2 + 9\frac{2}{3}l^3$	$ml + (\log P)l^2$	$2 \log P$
Gather $\bar{Q}$	AllGather	-	$ml$	$\log P$
$Z^{(j)} \leftarrow A^{(j)\top} \bar{Q}$	GEMM	$2\frac{1}{P}m^2l$	-	-
$M^{(j)} \leftarrow Z^{(j)\top} \bar{Q}^{(j)}$	GEMMT	$\frac{1}{P}ml^2$	-	-
$M \leftarrow \sum_{j=1}^P M^{(j)}$	AllReduce	-	$(\log P)l^2$	$\log P$
$W \Sigma^2 W^\top \leftarrow \text{eig}(M)$	SYTRD	$\frac{4}{3}l^3$	-	-
	ORGTR	$\frac{4}{3}l^3$	-	-
	STEQR	$7l^3$	-	-
$U \leftarrow \bar{Q}W$	GEMM	$2ml^2$	-	-

Table B.22: Complexity of Postprocessing (Block-Column Parallelism) (Conti.)

	Name	Routine	#flops	#words	#messages
	Algorithm IV-8 TSQR		$2\frac{1}{P}nml + 4\frac{1}{P}nl^2 + 2ml^2 + 7\frac{1}{3}(\log P)l^3 + 16l^3$	$nl + ml + (\log P)l^2$	$3 \log P$
	Gather $\bar{Q}$	AllGather	-	$ml$	$\log P$
	$Z^{(j)} \leftarrow A^{(j)\top} \bar{Q}$	GEMM	$2\frac{1}{P}nml$	-	-
	$V_0^{(j)} R_0^{(j)} \leftarrow \text{qr}(Z_t^{(j)})$	GEQRF	$2\frac{1}{P}nl^2 - \frac{2}{3}l^3$	-	-
LOOP	Combine $\tilde{Z}_{t-1}^{(j)}$	Sendrecv $\times \log P$	-	$(\log P)l^2$	$\log P$
	$\tilde{V}_t^{(j)} R_t^{(j)} \leftarrow \text{qr}(\tilde{Z}_{t-1}^{(j)})$	GEQRF $\times \log P$	$3\frac{1}{3}(\log P)l^3$	-	-
$W \Sigma T^\top \leftarrow \text{svd}(R_H^\top)$	GEBRD		$\frac{8}{3}l^3$	-	-
	ORGBR $W$ & $T$		$\frac{8}{3}l^3$	-	-
	BDSQR $W$ & $T$		$12l^3$	-	-
$U \leftarrow \bar{Q}W$	GEMM		$2ml^2$	-	-
$V^{(j)} \leftarrow V_0^{(j)} \dots V_H^{(j)} T$	ORMQR $\times \log P$		$4(\log P)l^3$	-	-
	ORMQR		$2\frac{1}{P}nl^2$	-	-
Gather $V$	Gather		-	$nl$	$\log P$

