Acceleration of Data Analytics using SVD on Heterogeneous Supercloud Systems

Georgios Zacharopoulos
Computing Systems Lab
Huawei
Zurich Research Center
Zurich, Switzerland

Ilias Bournias
Computing Systems Lab
Huawei
Zurich Research Center
Zurich, Switzerland

Lukas Cavigelli
Computing Systems Lab
Huawei
Zurich Research Center
Zurich, Switzerland

Abstract—**Emerging Heterogeneous Supercloud systems are redefining data analytics by enabling dynamic scaling and efficient distribution of tasks across diverse resources. This paper introduces innovative computational models and performance estimation techniques tailored for accelerating Singular Value Decomposition (SVD)-based data analytics on Heterogeneous Supercloud systems. Our approach optimizes all three phases of SVD computation, demonstrating a remarkable speedup of up to 126x compared to a highly optimized 48-core ARM CPU implementation using LAPACK. This breakthrough illustrates the significant potential of Heterogeneous Supercloud systems in advancing data analytics performance.**

Index Terms—**cloud computing, SVD, data analytics, heterogeneous**

I. Introduction

Heterogeneous computing has emerged as a vital paradigm in modern computing, harnessing a mix of processor types within a unified system to enhance performance. The concept of the Supercloud, as introduced by McColl [13], represents an advanced heterogeneous cloud architecture specifically engineered to meet the demanding processing needs of AI, big data, and high-performance computing (HPC) applications.

Supervised and unsupervised learning are fundamental to contemporary machine learning, playing vital roles in processing and interpreting complex data sets. A key strategy to enhance training efficiency for both types of learning is the reduction of features and dimensions. In supervised learning, focusing on the most pertinent data features through feature reduction not only streamlines the learning process but also curtails model complexity. This approach speeds up the computations and reduces the likelihood of overfitting. In the context of unsupervised learning, particularly in applications like clustering or anomaly detection, dimensionality reduction techniques such as Principal Component Analysis (PCA) are employed. PCA simplifies the data set while preserving critical features, facilitating more efficient data analysis.

Singular value decomposition (SVD) plays a pivotal role in executing PCA for feature and dimensionality reduction. SVD identifies the orthogonal axes, or principal components, where data variation is most significant. These components have wide-ranging applications, including data visualization, feature engineering, and noise reduction. By highlighting the most informative features and compressing the data, SVD-based PCA significantly boosts the efficiency of both supervised and unsupervised learning algorithms. This leads to faster training, better interpretability, and enhanced overall algorithm performance. Furthermore, SVD finds extensive use in natural language processing (NLP) tasks [14] and has been a critical component in advanced recommendation systems, such as the award-winning Netflix algorithm [8].

Developing an automated methodology that expedites SVD computations across varied heterogeneous systems is vital to improving the efficiency of machine learning training processes. This methodology needs to ensure scalability and applicability across different systems.

Our key contributions are as follows:

- We investigate two computation models—iterative and blocked—tailored for automating the mapping of SVD computations on heterogeneous Supercloud systems.
- We introduce novel performance estimation models that encompass both computation and communication latency for each computation model.
- We conduct comprehensive evaluations of all computation models and their optimizations on a 48-core ARM-based Huawei Kunpeng CPU [15] and an Ascend 910 AI hardware accelerator [10], providing insights into their efficacy and applicability in real-world scenarios.

II. Related Work

Recent years have seen increased interest in leveraging heterogeneous systems for accelerating SVD computations [5], [11]. Typically, these approaches divide the SVD process into two stages, optimizing the use of heterogeneous resources.

The initial task involves transforming a full matrix into a bidiagonal form. Faverge *et al.* [3] investigate tiled algorithms for this transformation, focusing on orthogonal transformations and providing a critical path analysis for each algorithm. Ltaief *et al.* [12] explore the influence of tile size on execution time, linking it to the matrix bandwidth size post-reduction. They use a brute force mechanism to retrieve the optimal tile size of a specific problem size. It is also pointed out that the optimal tile size is not the same for every sub-task of the problem. Gates *et al.* [5] develop a 2-stage reduction algorithm to transform a full matrix to a bidiagonal one, accelerating the process by offloading matrix multiplications to GPUs and

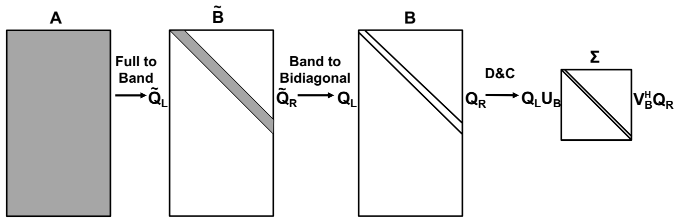

Fig. 1: 2-stage reduction to bidiagonal form followed by Divide and Conquer (D&C) SVD.

overlapping them with LQ and QR computations on the CPU. However, these studies lack automated tools for performance estimation based on hardware resources and granularity level to achieve maximum throughput.

Additionally, compiler-based methodologies have been introduced for performance estimation in heterogeneous systems [16], [17], focusing on general approaches and specific parallelism types [18], however not addressing diverse computation models. Design space exploration (DSE) methods have also been explored [2], [4], focusing on strategies for synthesizing accelerators and implementing optimizations.

III. Accelerated SVD for Heterogeneous Systems

The traditional SVD algorithm employs a *1-stage* reduction to bidiagonal form, relying on matrix-vector multiplications. This approach is often hindered by limited memory bandwidth, adversely affecting performance. To address this issue, a *2-stage* reduction algorithm to bidiagonal form has been proposed [5], among others. As illustrated in Figure 1, this algorithm reduces the matrix to a band form utilizing high-performance Level 3 BLAS functions, followed by reducing the band matrix to bidiagonal form using cache-optimized kernels with dynamic scheduling.

The SVD of an $m \times n$ matrix A is defined as $A = U\Sigma V^H$. Here, U and V are unitary matrices, and Σ is a real diagonal matrix with non-negative elements σ_i, which represent the singular values of A. The first $\min(m,n)$ columns of U and V constitute the left and right singular vectors of A, respectively. The Golub–Kahan–Reinsch algorithm [6], [7] performs SVD in three steps:

1) Bidiagonalization: Transform A into a bidiagonal matrix B such that $A = Q_L B Q_R$ with Q_L, Q_R unitary matrices,
2) Bidiagonal SVD: Employ QR or Divide and Conquer methods [9] to obtain $B = U_B \Sigma V_B^H$,
3) Extraction of singular vectors with back-transformation, $A = Q_L B Q_R = Q_L U_B \Sigma V_B^H Q_R = U\Sigma V^H$.

A. 2-Stage Bidiagonalization

The 2-stage reduction process begins by converting the full matrix into an upper band form, then further reducing it to upper bidiagonal. Specifically,

$$A = \tilde{Q}_L \tilde{B} \tilde{Q}_R = \tilde{Q}_L \hat{Q}_L B \hat{Q}_R \tilde{Q}_R = Q_L B Q_R,$$

with all matrices Q unitary, $\tilde{B}$ the upper band matrix, and B the upper bidiagonal matrix.

*1) **Full-to-Band**:* The first phase involves transforming the matrix to band form, crucially removing data dependencies with the trailing matrix during panel factorization. This is key to overcoming the bottleneck caused by matrix-vector operations in the panel.

The procedure involves a QR panel factorization of a block column to remove entries below the diagonal, followed by an update of the trailing matrix. This is succeeded by an LQ panel factorization of a block row to eliminate entries right of the upper band width of the matrix, and then another update of the trailing matrix.

*2) **Band-to-Bidiagonal**:* In the second stage, the upper band matrix is further reduced to upper bidiagonal form. Due to the limited parallelism, memory bandwidth constraints, and the need for CPU cache optimizations, using accelerators for this stage offers only marginal benefits. Hence, this stage's computations are primarily performed on the host CPU. It is important to note that the bulk of the computational work occurs in the initial stage (from full to band form), which consequently lessens the computational load in this later stage.

B. Divide and Conquer (D&C)

The D&C algorithm is employed for computing the SVD of a bidiagonal matrix B after its transformation into bidiagonal form. The matrix B is divided into two bidiagonal submatrices, B_1 and B_2, as illustrated below (e_k is the k-th column of an identity matrix). On each sub-matrix, the SVD is computed separately:

$$B = \begin{pmatrix} B_1 & \alpha_k e_k & 0 \\ 0 & \beta_k e_1 & B_2 \end{pmatrix}, \quad B_i = \begin{pmatrix} Q_i & q_i \end{pmatrix} \begin{pmatrix} \Sigma_i \\ 0 \end{pmatrix} W_i^H.$$

The D&C algorithm is recursively applied to B_1 and B_2 and their subsequent sub-matrices. However, at a certain recursion depth, the D&C algorithm becomes less beneficial due to excessively fine granularity, leading to the employment of a classic QR SVD approach for bidiagonal matrices [5].

C. Back-Transformation of Singular Vectors

To derive the singular vectors of matrix A, a three-stage back-transformation process is essential. The initial stage originates from the full matrix to band matrix reduction, where the block Householder reflectors, utilized in the band reduction and stored in compact WY format, are multiplied [5]. The second stage processes he matrices resulting from the band-to-bidiagonal transformation. The final stage utilizes U_B and V_B^H, the singular vectors of B, for the last back-transformation.

IV. Models of Computation

Our objective is to enhance the efficiency of all three stages of the SVD algorithm: bidiagonal reduction, D&C bidiagonal SVD, and back-transformation of singular vectors. We leverage optimized BLAS and Lapack functions [1] for the host CPU and accelerate parallelizable tasks in each stage.

We investigate two primary models of computation, each tailored to break down the SVD problem into tasks suitable for offloading to accelerators with varying granularity levels:

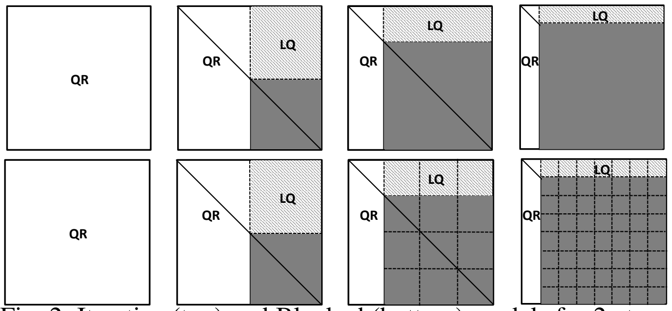

Fig. 2: Iterative (top) and Blocked (bottom) models for 2-stage SVD (Iteration: 0, 1, 2 and 3 Refinement Level: 1, 2, 4 and 8 respectively). Each dark grey rectangle (square) represents the area of gemm computation that is offloaded to an accelerator. The white rectangles of QR and LQ represent the part of the computation that remains in the host CPU.

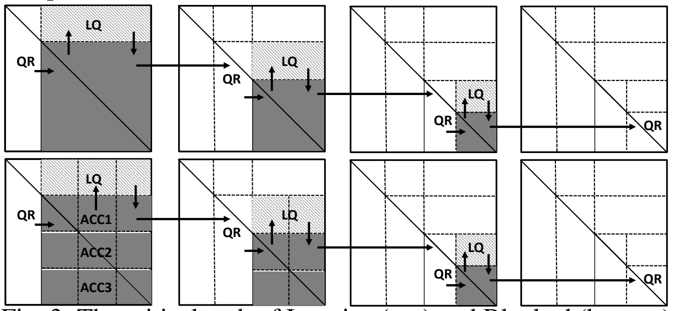

Fig. 3: The critical path of Iterative (top) and Blocked (bottom) models for 2-stage SVD (Refinement: 4). Dependencies are annotated with black arrows.

a) the iterative model and b) the blocked model for the first SVD stage (bidiagonalization).

A. Bidiagonalization

*1) **Iterative**:* The iterative approach to 2-stage bidiagonalization is depicted in Figure 2. For clarity, we illustrate only the first panel of the bidiagonalization process at each iteration or level of *refinement*. The concept of "refinement" here refers to a task's decomposition level, allowing for progressively finer granularity with each iteration.

We define the **Refinement Level** for iteration $i \in \{0, \dots, N\}$ as $r(i) = 2^i$. For each iteration i of the bidiagonalization decomposition/refinement, the dimensions of the initial QR matrix $n \times n$ (at $i = 0$) are halved from the size of the preceding iteration (thus, $n \times n/2$ for $i = 1$, $n \times n/4$ for $i = 2$, and so on).

The decision to proceed to the next refinement stage is based on a performance-related **condition**. This condition is applied *only* if advancing to the next refinement level $i + 1$ is projected to enhance performance. The following inequality quantitatively defines the criterion for proceeding with refinement in the iterative model of bidiagonalization:

$$\sum_{j=0}^{r(i+1)-1} t_{QR}^{(i+1,j)} + t_{BD}^{(i+1)} < \sum_{j=0}^{r(i)-1} t_{QR}^{(i,j)} + t_{BD}^{(i)}$$

where $t_{QR}^{(i,j)}$ represents the latency of the task QR for iteration i, and $t_{BD}^{(i)}$ is the latency of bidiagonalization. For example, Figure 2 demonstrates the computational equivalence for refinement levels 1 (no decomposition), 2, 4, and 8. With increasing refinement, the computational load of gemm tasks (depicted as dark grey rectangles) also increases, providing more opportunities for acceleration through offloading to available hardware resources.

Notably, the more computation is executed in the first stage (full-to-band), the less computational effort is needed in the second stage (band-to-bidiagonal), which mainly takes place on the host CPU and is not as amenable to acceleration. Therefore, our goal is to offload most of the computational load required for bidiagonalization to the first stage, thus expediting the overall process.

To estimate the computation and communication latencies in a heterogeneous Supercloud architecture, the following models are used:

$$t_{comp}^{(i)} = \sum_{j=0}^{r(i)-1} t_{QR}^{(i,j)} + \sum_{j=0}^{r(i)-2} t_{LQ}^{(i,j)} + \sum_{j=0}^{r(i)-2} \sum_{k=0}^{N-1} t_{gemm}^{(i,j,k)},$$

$$t_{comm}^{(i)} = \sum_{j=0}^{r(i)-2} \sum_{k=0}^{N-1} \left(t_{comm,H2D}^{(i,j,k)} + t_{comm,D2H}^{(i,j,k)} \right).$$

*2) **Blocked**:* Several advantages emerge in the blocked approach, applied after fine-tuning the refinement level and computation granularity for accelerator offloading. These include a) improved load balancing, b) optimized hardware accelerator utilization, c) enhanced scheduling capabilities, and d) potential task overlapping, aligning computation closer to the theoretical critical path for peak performance.

The computation and communication latency for a heterogeneous Supercloud in the blocked model are estimated as follows:

$$t_{comp}^{(i)} = \sum_{j=0}^{r(i)-1} t_{QR}^{(i,j)} + \sum_{j=0}^{r(i)-2} t_{LQ}^{(i,j)} + \sum_{j=0}^{r(i)-2} (r(i) - j - 1)^2 \cdot t_{gemm}^{(i,j)},$$

$$t_{comm}^{(i)} = \sum_{j=0}^{r(i)-2} \sum_{k=0}^{N-1} \left(t_{comm,H2D}^{(i,j,k)} + t_{comm,D2H}^{(i,j,k)} \right).$$

*3) **Blocked optimized**:* This variant of the blocked approach allows the use of multiple accelerators for simultaneous computation of gemm(i) tasks. As shown in Figure 3, a parallel implementation requires at least $r(2) - 1 = 3$ accelerators to match the theoretical critical path. The performance model for the parallel blocked approach is given by:

$$t_{comp}^{(i)} = \sum_{j=0}^{r(i)-1} t_{QR}^{(i,j)} + \sum_{j=0}^{r(i)-2} t_{LQ}^{(i,j)} + (r(i) - 1) \cdot \frac{r(i)}{2} \cdot t_{gemm}^{(i)}.$$

V. Experimental Setup

Our methodology and computation models were evaluated using a setup emulating the hardware resources of a heterogeneous Supercloud. The hardware configuration includes a 48-core ARM CPU, specifically the Huawei Kunpeng 920 [15], paired with a Huawei Ascend 910 AI processor [10], which

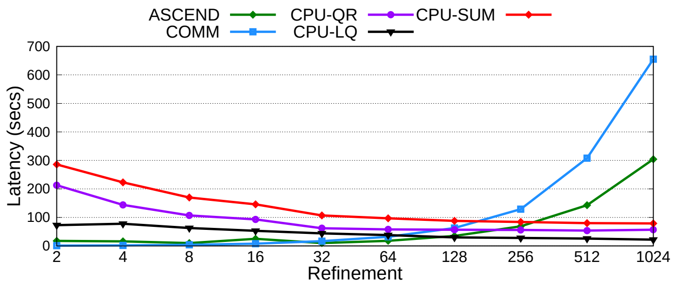

Fig. 4: First stage of bidiagonalization (full-to-band): Latency of the gemm computation offloaded to the Ascend AI accelerator, communication between accelerator and host, and CPU time as the refinement increases.

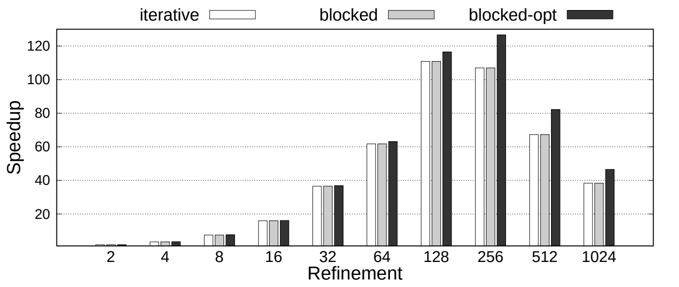

Fig. 5: Speedup obtained for SVD bidiagonalization increasing the refinement. Comparison of the iterative, blocked, and blocked-opt. (parallel) models.

features 32 Da Vinci AI cores capable of reaching a peak performance of 320 TFLOPS. Communication between these devices is facilitated via a PCIe link. For comparison, the CPU implementations in our experiments are multi-threaded, fully optimized SVD functions and subroutines from LAPACK [1].

VI. Experimental Results

We present results for all three SVD computation stages (bidiagonalization, D&C, and back-transformation) using iterative, blocked, and recursive models, tailored to each stage's requirements. The overall speedup achieved by integrating the most effective strategies for each stage is compared to a fully optimized multi-core CPU implementation. The input for these experiments is a square matrix of size 32000, resulting in 32000^2 entries. Our study examines the effects of refining task granularity and distribution, while offloading more computation to the first stage of the 2-stage bidiagonalization process.

Figure 4 illustrates the computation and communication dynamics between the CPU and the accelerator during the first bidiagonalization stage. We observe a gradual decrease in the host CPU's processing time, while the latency on the Ascend AI accelerator and communication overhead surpass the CPU time only at refinement levels 128 and 256. Beyond these levels, communication costs and accelerator latency increase exponentially, whereas CPU time plateaus.

Figure 5 compares the speedup of the overall bidiagonalization phase across the iterative, blocked, and blocked optimized models. Each model employs the Ascend AI accelerator(s) in conjunction with the host CPU, benchmarked against a fully optimized LAPACK 48-core ARM CPU version. The peak performance, with speedups of **116x** and **126x**, is observed at refinement levels 128 and 256 in the optimized parallel version of the blocked model. Beyond these levels, performance declines sharply due to the escalating latencies of AI accelerators and communication costs, while CPU time remains stable.

VII. Conclusions

Our study introduces novel models for performance estimation and explores computation models to accelerate SVD-based data analytics on heterogeneous Supercloud systems. By automating task mapping, scheduling and parallelization, our methodology achieves up to an 126x speedup on a system comprising a 48-core ARM CPU and an Ascend accelerator.

References

[1] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, third edition, 1999.

[2] Iulian Brumar et al. Early DSE and automatic generation of coarse grained merged accelerators. *ACM Trans. Embed. Comput. Syst.*, 2022.

[3] Mathieu Faverge, Julien Langou, Yves Robert, and Jack Dongarra. Bidiagonalization and r-bidiagonalization: Parallel tiled algorithms, critical paths and distributed-memory implementation. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017.

[4] Lorenzo Ferretti et al. Graph neural networks for High-Level Synthesis Design Space Explor. *ACM Trans. on Des. Autom. of Electr. Syst.*, 2022.

[5] Mark Gates, Stanimire Tomov, and Jack Dongarra. Accelerating the SVD two stage bidiagonal reduction and divide and conquer using gpus. *Parallel Computing*, 74:3–18, 2018.

[6] Gene Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics*, 2(2):205–224, 1965.

[7] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. In *Handbook for Automatic Computation: Volume II: Linear Algebra*, pages 134–151. Springer, 1971.

[8] Stephen Gower. Netflix prize and SVD. *University of Puget Sound*, 2014.

[9] Ming Gu and Stanley C Eisenstat. A divide-and-conquer algorithm for the bidiagonal svd. *SIAM Journal on Matrix Analysis and Applications*, 16(1):79–92, 1995.

[10] Heng Liao et al. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing. In *HPCA*, 2021.

[11] Ding Liu, Ruixuan Li, David J. Lilja, and Weijun Xiao. A divide-and-conquer approach for solving singular value decomposition on a heterogeneous system. In *Proceedings of the ACM International Conference on Computing Frontiers*, New York, NY, USA, 2013.

[12] Hatem Ltaief, Piotr Luszczek, and Jack Dongarra. High-performance bidiagonal reduction using tile algorithms on homogeneous multicore architectures. *ACM Trans. Math. Softw.*, 39(3), may 2013.

[13] Bill McColl. Superclouds: Scalable high performance nonstop infrastructure for AI and smart societies. In *SCITA*. Springer, 2017.

[14] Kaiz Merchant and Yash Pande. NLP based latent semantic analysis for legal text summarization. In *2018 international conference on advances in computing, communications and informatics (ICACCI)*, pages 1803–1807. IEEE, 2018.

[15] Jing Xia, Chuanning Cheng, Xiping Zhou, Yuxing Hu, and Peter Chun. Kunpeng 920: The first 7-nm chiplet-based 64-core arm soc for cloud services. *IEEE Micro*, 41(5):67–75, 2021.

[16] Georgios Zacharopoulos et al. Compiler-assisted selection of hardware acceleration candidates from application source code. *Proceedings of the International Conference on Computer Design*, pages 1–9, 2019.

[17] Georgios Zacharopoulos et al. RegionSeeker: Automatically identifying and selecting accelerators from application source code. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[18] Georgios Zacharopoulos et al. Trireme: Exploration of hierarchical multi-level parallelism for hardware acceleration. *ACM Trans. Embed. Comput. Syst.*, 22(3), apr 2023.