© September 2024 | IJIRT | Volume 11 Issue 4 | ISSN: 2349-6002

Optimizing SVD for Object Tracking: A Hardware-Software Co-Design Methodology

Priyanka Singh[1], Mohd.Murtaja[2], and Gaurav Verma[3]
[1,2] *Chaudhary Charan Singh University Campus, Meerut-250005, UP, India*
[3]*National Institute of Technology Kurukshetra-136119, Haryana, India*

***Abstract*—The present study suggests the development of a customized processor architecture designed specifically for tracking applications, taking into account the complex nature of tracking in nonlinear systems. We design the prototype platform to enhance tracking capabilities by incorporating singular value decomposition (SVD), a crucial component of the tracking algorithm. By including SVD in the processor design, the suggested architecture can solve the problems that come up with real-time vision algorithms used in object tracking, smart cameras, and automated security systems.**

***Index Terms*—Singular value decomposition, Tracking Algorithm, Profiling, Co-Design**

I. INTRODUCTION

This paper proposes a method to analyse and develop real-time vision algorithms for tracking objects. Apart from standard accuracy requirements, the algorithms must satisfy the target processing architecture's memory, frame rate, and latency constraints [1]. A closed-loop control system uses latency, the time it takes to produce results for a frame, as a critical constraint for purposeful motion control of the camera. Developing real-time algorithms for the processing architecture to extract high-level descriptors from streaming video poses key challenges. These issues necessitate the development of special-purpose processor architectures different from traditional DSPs [2]. This paper focuses on tracking applications. Tracking creates an interface that provides information on an object's motion based on a series of photographs. Motion capture, motion recognition, surveillance, targeting, and other applications that use tracking are a few examples [3]. An inference problem more accurately describes the tracking process. Each frame necessitates determining the internal status of the moving object. To estimate the state of the object, it is imperative to optimize the integration of our measurements to accurately determine the condition of the object. There are two important cases. The first scenario occurs when both the dynamics and measurements are linear, resulting in a straightforward inference problem with a standard solution. Even minor deviations in system dynamics have a significant impact on the outcome when dealing with nonlinear dynamics. As a result, drawing inferences can be challenging and may even seem unattainable. If the dimension of the state space is low, there is a useful algorithm that often works. We have chosen tracking as the application for processor design. Within the framework of the tracking algorithm, the singular value decomposition (SVD) function is an essential component. We aim to propose a prototype platform development strategy that utilizes SVD for tracking applications [4]. We can approach SVD from three complementary perspectives. It can be considered a technique for converting correlated variables into a collection of uncorrelated variables that exhibits a more accurate representation of the various relationships that exist between the initial data segments. On the other hand, we can use SVD to identify and arrange the dimensions where an individual data point exhibits the highest variance. The third way of looking at SVD connects to the possibility of obtaining the best approximation of the original data points by using fewer dimensions. Discovering the most variance allows us to make the best approximation. Therefore, we can consider SVD as a technique for data reduction. To illustrate these concepts, examine the 2-dimensional data elements in Figure 1(a). The regression line that passes through them represents the most accurate approximation of the original data using a one-dimensional object (a line). Because the line reduces the space between the original points and the line, it is the most accurate approximation [5].

To estimate the true data point, we can construct a perpendicular line connecting every point to the regression line, and then use the point where these lines join as the approximation statistic. This approach would aim to capture the maximum amount of the original deviation possible.

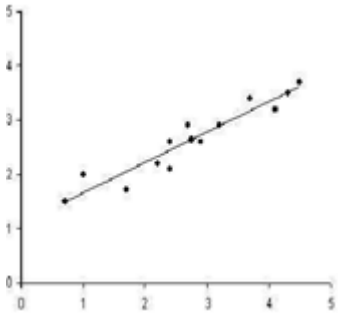

Figure 1(a)

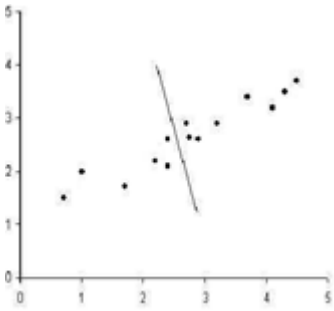

Figure 1 (b)

Figure 1(a) shows that the best-fit regression line integrates two-dimensional data into one. Figure 1(b) shows that the second dimension causes the regression line to capture less variation in the original data.

As we can see from figure 1(b) that displays a second regression line perpendicular to the first one. To the best of our knowledge, this line represents the original data set's second dimension. Because it pertains to a dimension that initially had less variance, it does not perform as well in approximating the original data. Using these regression lines, one can create a dataset devoid of correlation, revealing hidden patterns in the original data that might not be immediately apparent.

The fundamental concepts of SVD are as follows: first, we take a high-dimensional, highly variable dataset and reduce it to a lower-dimensional space. This makes the original data's substructure more apparent and sorts it from most variable to least variable. By ignoring variation below a certain threshold, you can drastically compress your data using SVD for natural language processing applications while maintaining the essential associations of interest. A linear algebraic theorem, the foundation of SVD, asserts that we can divide a rectangular matrix A into three matrices: the transpose of an orthogonal matrix V, an orthogonal matrix U, and a diagonal matrix S. The theorem typically takes the following form:

In this case, UTU = I, VTV = I. The columns U and V represent the orthonormal eigenvectors of AAT and ATA, respectively. S is a diagonal matrix that has the square roots of the singular values or eigenvalues from V or U arranged in decreasing order [6].

Amn = Umm Smn (Vnn) T

Where UTU = I; VTV = I; the columns of U are orthonormal eigenvectors of AAT, the columns of V are orthonormal eigenvectors of ATA, and S is a diagonal matrix containing the square roots of Eigenvalues or singular values from U or V in descending order. The decomposition of the matrix into different matrices aids many image processing applications in analysing various features in the image independently at the same time with reduced noise.

II. WORKING OF SVD

The application's matrix data, a memory-stored image matrix, initiates the SVD operation. The householder function will take the data in an array and convert it from an ordinal matrix to a bidiagonal matrix. After converting the data into a bidiagonal matrix, the right-hand and left-hand accumulation functions utilize the bidiagonal matrix to compute the U and V matrices. Following this, we utilize the calculated data to compute the diagonal matrix from the bidiagonal matrix and subsequently determine the Eigenvalues from the data.

III. METHODOLOGY: HW/SW PARTITIONING

The hardware and software practitioner determines which part of the application should be implemented in hardware and which one in software, specifically identifying which function requires dedicated hardware. In this manner, when the processor executes certain functions, it directs them to the appropriate hardware, and after completing that function.

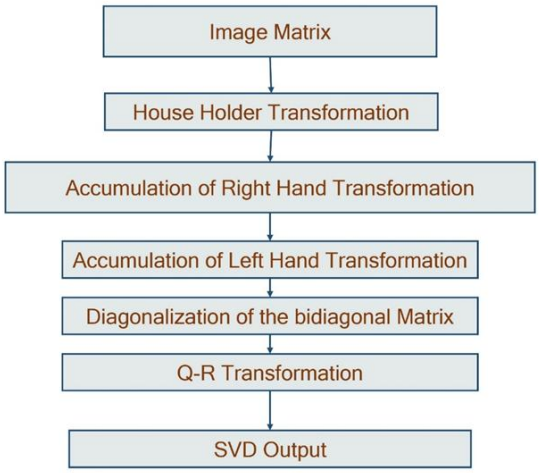

Figure 2. SVD Flowcharts

It proceeds to execute other functions using its own set of instructions [7]. The steps used for the HW/SW partitioning in this project are as follows:

i. For a functional modular implementation, we have modified the code to include some essential functions.

ii. In the Linux operating system, we use the "gprof" profiler tool to profile the C code and estimate the execution time of each function.

iii. Based on each function's execution time, we can determine which function requires dedicated hardware based on the execution time of each function.

A.Hardware-Software Co-Design

A primary objective of co-design is to expedite the time-to-market by minimizing the design work and expenses associated with the resulting products. As a result, the designer must take advantage of the diversity of the chosen architecture. The benefit of employing processors is multifaceted, as software offers greater flexibility and cost-effectiveness compared to raw hardware. The software's versatility allows for the inclusion of late design modifications and easier debugging. Moreover, the potential for software reuse through iterative portability to different processors decreases both the time required to bring the product to market and the design work [8]. Primarily, the utilization of processors is generally more cost-effective than the expenses incurred in developing ASICs due to the large-scale production of processors, resulting in a substantial price decrease. However, when the CPUs fail to achieve the required performance, the designer always employs hardware. This trade-off between the hardware and the software demonstrates the optimization part of the co-design challenge. Co-design is an example of an interdisciplinary activity that combines ideas and concepts from many fields, such as software engineering, hardware design, and system-level modelling. Figure 3 shows the overall co-design process flow. The first step in a co-design project is to define the system-level behavior specifications. The next step involves breaking down the system specification into smaller components, referred to as granules, akin to building blocks. During the cost-estimating phase, we determine the grain values for several cost indicators. Projected expenses can comprise both software and hardware investments. Some examples of hardware cost metrics include runtime, power consumption, area on the chip, and testability. Metrics for software costs could include runtime data and program memory requirements.

After calculating the cost, the next step is to divide the granules into hardware or software groups. We accomplish this by determining the optimal method for mapping the granules to the appropriate hardware or software and then implementing the groups of granules on the appropriate hardware and software [9]. The mapping necessitates extra interface components between ASICs and CPUs to implement the system on a heterogeneous target architecture. The specification refinement stage converts the system specification, which is independent of implementation, into specifications for both hardware and software components. Every specification inherently incorporates communication protocols that facilitate the transfer of data between CPUs and ASICs. The selected processor compiles the software specification and synthesizes the hardware based on the provided specification. The co-synthesis phase produces a collection of integrated circuits (ASICs) and assembler programs designed for the systems. Finally, the processors co-simulate the ASICs while they execute the assembler instructions they designed. After all performance restrictions and design costs have been considered, the co-design process comes to a halt. If not, the design optimization through repartitioning continues until an adequate system implementation is discovered.

Moreover, apart from the aforementioned issues, there are extra challenges in the domain of hardware-software co-design. The issue of co-validation in system-level design involves several approaches to identify faults at various levels of abstraction

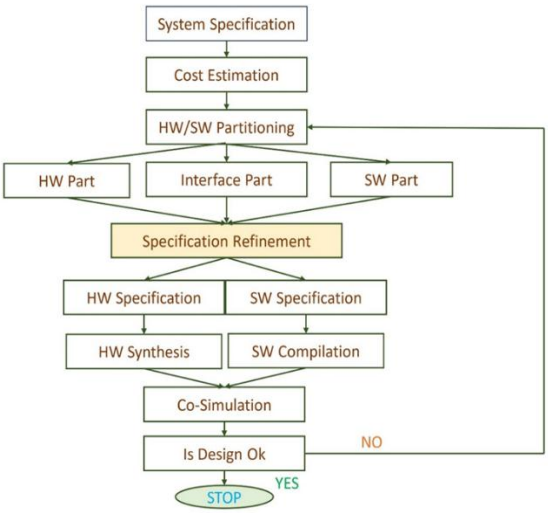

Figure 3. The Complete Flow of Hardware-Software Co-Design

B. Profiling of SVD Algorithm

We can design the various basic computing blocks of SVD using VHDL or the System C language and implement the processor on an FPGA [12]. Before implementation, hardware and software partitioning is required. At this moment, we are using profiling. Profiling enables us to ascertain the allocation of time and the invocation of corresponding functions by our software throughout its execution. Moreover, it can furnish us with insights into whether our software invokes a function at a frequency higher or lower than expected. This methodology may facilitate the identification of bugs that have previously been undetected. The profiler's reliance on data gathered during program execution makes it applicable to programs too large or intricate for source analysis [13]. Nevertheless, the execution of our program will impact the content observed in the profile data. Failure to use a certain feature of our application during the profiling process will result in no profile information for that feature. The GNU Gprof 2.15.92.02 is the specific Gprof profiler version used

C.H*ardware Software Co-Design: Platform-Based Design*

In this paper, we propose an idea for the development and implementation of an algorithm using a platform-based design, as shown in Figure 4. As depicted in Figure 4, an application developer will construct his application using the available library elements. We will use the API to compile the application code and then map the generated bit pattern into the FPGA device. Before implementation, hardware and software partitioning are required. For this purpose, we are currently using profiling. The Power PC will receive the software portion of the code, while the logic and block RAMs will receive the hardware portion of the code. The proposed prototype platform-based design aims to achieve all the aforementioned goals [11]. The block diagram for the hardware-software co-design system is shown below.

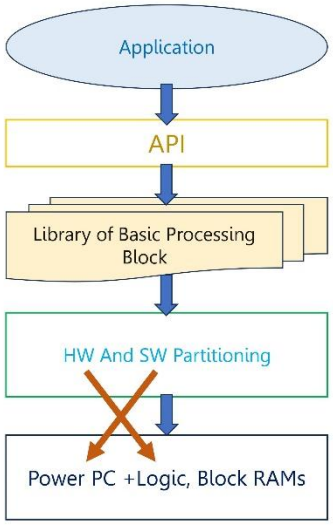

Figure 4: Hardware Software Co-design System

C. Performance Analysis

The original SVD code is plain and has only one important function, "main." Modifying the code to include a few critical functions will create a functional modular implementation. Implementing this code directly on the hardware with a synthesis tool will result in a large chip or hardware area filled with numerous redundant hardware functions [14]. The most effective method for implementing a code or algorithm is to break it down into smaller

components, allowing for reusable functions without the need for repeated synthesis.

IV. CONCLUSION & FUTURE PROSPECTIVE

We have proposed a methodology for implementing the SVD function. This work provides recommendations for implementing the SVD function in both software and hardware to achieve desired results. The modified SVD code runs successfully on a Linux-based system. We have profiled the system for hardware-software partitioning. Based on the analysis results, we can identify certain computation-intensive functions that the application frequently requires. By selecting the appropriate hardware devices and implementing the software on a Power PC, we have designed the embedded system of SVD. System-based SVD code implementation has been done to generate dedicated hardware for tracking applications in smart camera systems. This is done so that it can be added as a library element for the smart camera system in the proposed platform-based design. The software-hardware partitioning has been done by profiling. Based on performance analysis results, we identified functions that are computationally intensive and frequently required by the application.

The application of object tracking is central to this research, which involves generating information about an object's motion from a sequence of images. This capability is vital in various fields, including motion capture, recognition from motion, surveillance, and targeting. Tracking, conceptualized as an inference problem, requires accurately estimating the internal state of a moving object based on sequential measurements. In scenarios where system dynamics and measurements are linear, standard solutions are applicable. However, even slight nonlinearities can complicate the inference process, making it challenging and sometimes infeasible. By leveraging the proposed prototype platform-based implementation of SVD in Smart Camera Systems, Embedded Vision Systems, Medical Imaging, and Data Compression areas, systems can achieve higher efficiency, reduced latency, and improved performance, particularly in applications where real-time processing is essential.

REFERENCES

[1]. Harris S, Harris D. Digital Design and Computer Architecture, RISC-V Edition. Morgan Kaufmann; 2021 Jul 12.

[2]. Liu D. Embedded DSP processor design: Application-specific instruction set processors. Morgan Kaufmann; 2008 May 30.

[3]. Sankaranarayanan AC, Veeraraghavan A, Chellappa R. Object detection, tracking and recognition for multiple smart cameras. Proceedings of the IEEE. 2008 Oct 17;96(10):1606-24.

[4]. Doblander A, Zoufal A, Rinner B. A novel software framework for embedded multiprocessor smart cameras. ACM Transactions on Embedded Computing Systems (TECS). 2009 Apr 22;8(3):1-30.

[5]. Klema V, Laub A. The singular value decomposition: Its computation and some applications. IEEE Transactions on automatic control. 1980 Apr;25(2):164-76.

[6]. Lange K, Lange K. Singular value decomposition. Numerical analysis for statisticians. 2010:129-42.

[7]. Clements A. Principles of computer hardware. Oxford University Press, USA; 2006 Feb 9.

[8]. Anasuodei M, Akpofure N. Software Reusability: Approaches and Challenges. International Journal of Research and Innovation in Applied Science. 2021;6(05).

[9]. Cirstea M, Benkrid K, Dinu A, Ghiriti R, Petreus D. Digital Electronic System-on-Chip Design: Methodologies, Tools, Evolution, and Trends. Micromachines. 2024 Feb 7;15(2):247.

[10]. FORNACIARI W. Co-synthesis and co-simulation of control-dominated embedded systems. Readings in Hardware/Software Co-Design. 2001 Jun 19:395.

[11]. Xue S. Overview of Computer Architecture Development Direction Breaking Through Von Neumann Architecture. International Core Journal of Engineering. 2021 Aug 1;7(8):330-4.

[12]. Fummi F, Loghi M, Perbellini G, Poncino M. SystemC co-simulation for core-based embedded systems. Design Automation for Embedded Systems. 2007 Sep;11:141-66.

[13]. Gangarapu S, Chilukoori SS. Profiling-Driven Performance Enhancement: Accelerating Embedded Firmware Execution. memory. 2024 Jul;11(07).

[14]. Gottschall B. Time-Proportional Performance Analysis for Out-of-Order Processors. 2024