VLSI Hardware Architecture for Gaussian Process

Chunhua Deng[1], Yongbin Gong[2], Feng Han[2], Siyu Liao[1], Jingang Yi[2], Bo Yuan[1]
[1]Department of Electrical and Computer Engineering, Rutgers University
[2]Department of Mechanical and Aerospace, Rutgers University
{chunhua.deng, siyu.liao, jgyi}@rutgers.edu; {yg283, fh233}@scarletmail.rutgers.edu; bo.yuan@soe.rutgers.edu

***Abstract*—Gaussian process (GP) is a popular machine learning technique that is widely used in many application domains, especially in robotics. However, GP is very computation intensive and time consuming during the inference phase, thereby bringing severe challenges for its large-scale deployment in real-time applications. In this paper, we propose two efficient hardware architecture for GP accelerator. One architecture targets for general GP inference, and the other architecture is specifically optimized for the scenario when the data point is gradually observed. Evaluation results show that the proposed hardware accelerator provides significant hardware performance improvement than the general-purpose computing platform.**

***Index Terms*—Gaussian Process, VLSI**

I. Introduction

Gaussian process (GP) [1]–[3] is a widely used machine learning approach in many applications, such as robotics, bioinformatics, finance etc. In the today's "deep learning era", GP, as an elegant kernel method, is still a very important and useful machine learning technique because of its unique non-parametric flexibility and inherent interpretability. In particularly, the recent discovery of the inherent equivalence between GP and infinitely wide fully-connected neural network further motivates the active research on GP technique.

However, a well-known challenge of GP is the scalability. Due to the matrix inversion operation in the inference phase, GP is inherently computation intensive and storage intensive, which therefore hinders the potential large-scale deployment of GP in many real-time applications. To date, though many algorithm-level optimizations have been proposed to accelerate GP inference and improve its scalability, the hardware acceleration support, which is a key for the current success of neural network, is still missing for GP.

In order to accelerate GP inference and promote its further adoption in the wide range of applications, in this paper we propose to develop customized hardware architecture of GP accelerator. Considering the matrix inversion is the bottleneck step in GP inference, we propose two hardware architecture targeting to two different cases. One architecture, which is equipped with explicit matrix inversion module, is used for general scenario. And another architecture, which utilize the recursive property of covariance matrix, is specifically optimized for the scenario when the data point is observed gradually. Evaluation results show that our proposed accelerator provides significant hardware performance improvement over the general-purpose computing platforms.

The rest of this paper is organized as follows. Section II reviews the overall computing procedure of GP and also describes the optimization technique for computation saving. The hardware architecture exploration is presented in Section III. Section IV shows the evaluation results, and the conclusions are drawn in Section V.

II. Gaussian Process Computing Procedure

A. Overall Computing Procedure of GP

A typical GP computation procedure consists of covariance matrix calculation, matrix inversion, and matrix multiplication. In general, the inputs of GP are a vector $\mathbf{x}$ and its corresponding state vector $\mathbf{y}$, and x_* as the data point to be interpolated. In the first step of GP, the covariance matrices $\mathbf{K}$ and $\mathbf{K}_*$ are calculated as follows:

$$\mathbf{K}(i,j) = \sigma_f^2 e^{-\frac{1}{2l^2}(x_i - x_j)^2}, \quad (1)$$

$$\mathbf{K}_*(i) = \sigma_f^2 e^{-\frac{1}{2l^2}(x_i - x_*)^2}, \quad (2)$$

where i and j are within the range of N, which represents the number of data samples. Notice that here the covariance matrix $\mathbf{K}$ is inherently symmetrical, a useful property that we will leverage to faciliate hardware design. In addition, σ_f and l are the hyperparameters that are determined from GP training phase. Once covariance matrix $\mathbf{K}$ is obtained, it will be used to calculate matrix $\mathbf{K}_y = \mathbf{K} + \sigma_n^2 \mathbf{I}$, where σ_n is another hyperparameter and $\mathbf{I}$ is the identity matrix. Finally, the mean y_* and variance Σ_* of the outputs are estimated as follows [3]:

$$y_* = \mathbf{K}_*{}^T \mathbf{K}_y^{-1} \mathbf{y}, \quad (3)$$

$$\Sigma_* = \sigma_f^2 - y_*. \quad (4)$$

B. Reduced-complexity Matrix Inversion for GP

From Eq. 3 it is seen that the kernel computations of GP are matrix multiplication, matrix inversion and matrix-vector multiplication. Among them, matrix inversion is the bottleneck since it requires $O(n^3)$ computational cost. Therefore, a reduced-complexity matrix inversion is the key for accelerating GP and make it more attractive for real-time applications.

To that end, we propose to utilize the inherent characteristic of the covariance matrix of GP to reduce the computational cost of matrix inversion. More specifically, for the overall matrix inversion scheme we adopt the block matrix inversion algorithm [4]. Assume $\mathbf{R}$ is the to-be-inverted matrix that can be represented in the block format as:

$$\mathbf{R} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}. \quad (5)$$

Then the inverse of $\mathbf{R}$ can be calculated as

$$\mathbf{R}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\mathbf{M}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{M} \\ -\mathbf{M}\mathbf{C}\mathbf{A}^{-1} & \mathbf{M} \end{bmatrix}, \quad (6)$$

where $\mathbf{M} = (\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}$. As described in Eq. 6, the inverse of a block matrix $\mathbf{R}$ can be expressed with the inverse of its submatrix $\mathbf{A}$. Such property is especially useful for the matrix inversion in GP since in some practical applications, such as robotics, the information involved with the observation points is added to the covariance matrix gradually. In other words, the covariance matrix of GP grows from $k \times k$ to $(k+1)\times(k+1)$ each time when a new point is observed. Therefore, in Eq. 5 when the $(k+1) \times (k+1)$ matrix $\mathbf{R}$ is partitioned to make $\mathbf{A}$ as a $k \times k$ matrix, $\mathbf{A}$ is just the old covariance matrix before observing the current new data point.

Based on such recursive construction principle, we can utilize the inverse of $\mathbf{A}$ to quickly calculate $\mathbf{R}$ via Eq. 6. Notice in such setting the matrices $\mathbf{D}$, $\mathbf{B}$ and $\mathbf{C}$ in Eq. 6 are scalar, column and row vectors, respectively. In addition, since $\mathbf{R}$ is a symmetric matrix, $\mathbf{B}$ equals $\mathbf{C}^T$, it can be represented as:

$$\mathbf{R} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^T & d, \end{bmatrix}. \quad (7)$$

where $\mathbf{b}$ is the vector and d is the scalar. Therefore, Eq. 6 can be further simplified as:

$$\mathbf{R}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{b}\mathbf{b}^T\mathbf{A}^{-1}/k & -\mathbf{A}^{-1}\mathbf{b}/k \\ -\mathbf{b}^T\mathbf{A}^{-1}/k & 1/k \end{bmatrix} \quad (8)$$

$$= \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{s}\mathbf{s}^T/k & -\mathbf{s}/k \\ -\mathbf{s}^T/k & 1/k \end{bmatrix}, \quad (9)$$

where $k = d - \mathbf{b}^T\mathbf{s}$ and $\mathbf{s} = \mathbf{A}^{-1}\mathbf{b}$. Based on Eq. 9, we can recursively calculate $\mathbf{R}^{-1}$ via using $\mathbf{A}^{-1}$ without explicitly performing matrix inversion for multiple times, thereby significantly reducing computational costs.

III. Hardware Architecture Exploration

In this section we explore the efficient hardware architecture for GP. We first propose a matrix inversion-based hardware architecture, which is a general solution when the covariance matrix is not gradually increased. We then propose another more specialized hardware architecture that is customized designed for the scenario when the inverse of covariance matrix can be efficiently calculated as Eq. 9.

A. Matrix Inversion-based Hardware Architecture

Overall Architecture. Fig. 1 shows the overall hardware architecture of the general GP accelerator, which contains three major modules corresponding to three major operations of GP computing procedure. First, the input vector $\mathbf{x}$ and the point to be interpolated x_* are sent to the Covariance Generation module to calculate the covariance, and the result is stored in the Covariance SRAM and the K_* SRAM. Then, a matrix inversion unit (MIU) computes the matrix inversion of the covariance matrix. After that, a post-processing module is used to calculate y_* and Σ_* according to Eq. 3 and Eq. 4, respectively.

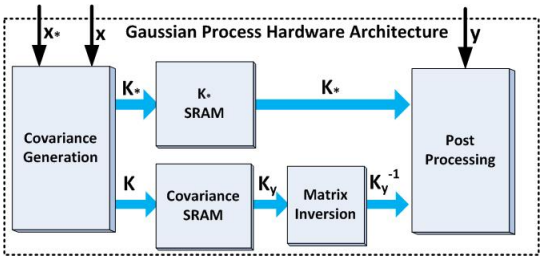

Fig. 1. The overall hardware architecture of Gaussian Process in general case.

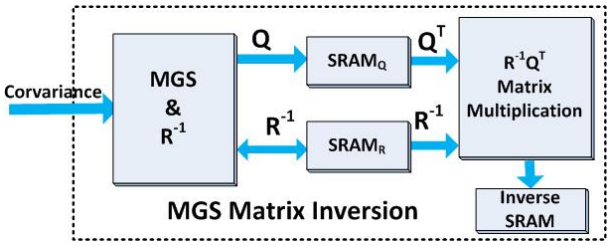

Fig. 2. Hardware architecture of MGS-based matrix inversion.

Matrix Inversion Module. Considering matrix inversion is the bottleneck module, in this subsection we focus on this component. In general, matrix inversion can be performed using various methods, and one common approach is based on QR decomposition (QRD), which also can be realized in many hardware-friendly ways, such as Gram-Schmidt (GS) [5], Givens rotation (GR) [6], Householder [7], etc. In this work, we adopt Modified Gram-Schmidt (MGS) [8], [9], which is proven to be numerically equivalent to GR but needs less number of operations when the target matrix is square [10]. Fig. 2 shows the hardware architecture of MGS QRD. The MGS QRD module performs the QR decomposition and generates an orthogonal matrix $\mathbf{Q}$ and a upper triangular matrix $\mathbf{R}$. After obtaining these two matrices, the inverse of input matrix can be calculated as $\mathbf{R}^{-1}\mathbf{Q}^T$, where the inversion of the triangular matrix is quite straightforward [10]. Algorithm 1 describes the details of MGS algorithm, and its corresponding hardware architecture is shown in Fig. 3. From Fig. 3, it is seen that the main computations in MGS QRD are inner product, $\mathbf{W}$ matrix update and norm calculation, which are accelerated by three corresponding hardware blocks as inner-product unit (IPU), $\mathbf{W}$ matrix update unit (WUU), and norm calculate unit (NCU), respectively.

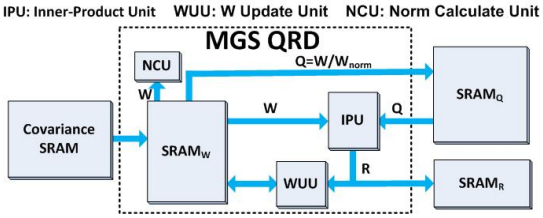

Fig. 3. Hardware architecture of MGS-based QR decomposition.

Algorithm 1: Modified Gram-Schmidt QR Decomposition Algorithm

Input : $\mathbf{A}_{n\times n}$
Output: $\mathbf{Q}_{n\times n}$, $\mathbf{R}_{n\times n}$

$\mathbf{R} = \mathbf{0}$
for $j = 1$ **to** n **do**
  $\mathbf{W}$=$\mathbf{A}_{:j}$
  for $i = 1$ **to** $j-1$ **do**
    $\mathbf{R}_{ij} = \langle \mathbf{W}, \mathbf{Q}_{:i} \rangle$
    $\mathbf{W}$ = $\mathbf{W}$ - $\mathbf{R}_{ij}\mathbf{Q}_{:i}$
  $\mathbf{Q}_{:j} = \mathbf{W} / \|\mathbf{W}\|_2$
  $\mathbf{R}_{jj} = \|\mathbf{W}\|_2$

B. Matrix Inversion-free Hardware Architecture

Recall that as described in Eq. 9, the inverse of $\mathbf{R}$ can be simply updated using $\mathbf{A}^{-1}, \mathbf{b}$, and d when the data points are gradually increased. Therefore for such special case the corresponding hardware architecture is significantly simplified. Fig. 4 shows the overall hardware architecture for this matrix inversion-free design. Here the data read from the inverse SRAM and covariance SRAM are used to calculate the vector $\mathbf{s}$ and $\mathbf{ss^T}$. These calculated information, together with k, are used to update the inverse SRAM for the next iteration. More specifically, Fig. 5 describes the detailed procedure of this matrix inversion-free operation via a 4-state finite state machine (FSM) as follows:

1) IDLE state: When a start signal is detected in IDLE state, the FSM enters into Cal_s state.
2) Cal_s state: The FSM reads the data in the Inverse SRAM and the data in the Covariance SRAM to calculate vector $\mathbf{s}$, and writes the vector in the SRAMs.
3) Cal_k state: The FSM reads the data in the SRAMs and data in the Covariance SRAM to get k.
4) Update Inverse SRAM state: Fig. 6 shows the four regions of the Inverse SRAM (logically). For Update Region, the data is read from the Inverse SRAM, added the corresponding element of $\mathbf{ss}^T/k$, and then written back the Inverse SRAM. For the Right Vector Region, the Bottom Vector Region, and the New Element Region, the element value is generated according to Eq. 9 and written into their corresponding address of the Inverse SRAM.

Computational Cost Analysis. The matrix inversion-free operation brings great benefits of computational costs reduction. Assume the matrix $\mathbf{A}^{-1}$ is of size $n \times n$. First, it takes n^2 multiplications to calculate vector $\mathbf{s}$. Then, n multiplications are needed for calculating k. With $\mathbf{s}$ and k, it takes $2n(n+1)$ multiplications to update the inverse matrix $\mathbf{R}^{-1}$. Therefore, the total number of multiplication operations is $3n(n+1)$. On the other hand, following the similar analysis principle, the number of multiplications for MGS-based matrix inversion is roughly $2n^3 + (3n^2 + n)/2$. Comparing the two roughly estimated computational costs, it is seen that using recursive property of $\mathbf{R}$ leads to very significant computation saving.

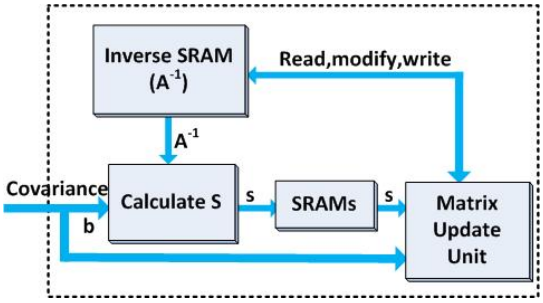

Fig. 4. Hardware architecture of matrix inversion-free design.

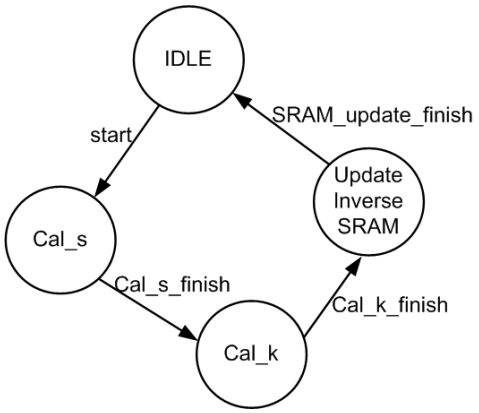

Fig. 5. 4-state FSM for matrix inversion-free operation.

It should be noted that the matrix inversion-free scheme does not reduce computational cost, but also reduces storage requirement. In MGS-based solution, besides the inversion SRAM, two additional SRAMs are needed to store the matrix $\mathbf{Q}$ and matrix $\mathbf{R}$, and one SRAM is needed to store matrix $\mathbf{W}$ in the MGS hardware module. Instead, as shown in Fig. 4 only one additional SRAM is needed to store the internal vector $\mathbf{s}$ for matrix inversion-free scheme, thereby significantly reducing SRAM cost and hence improving the energy efficiency.

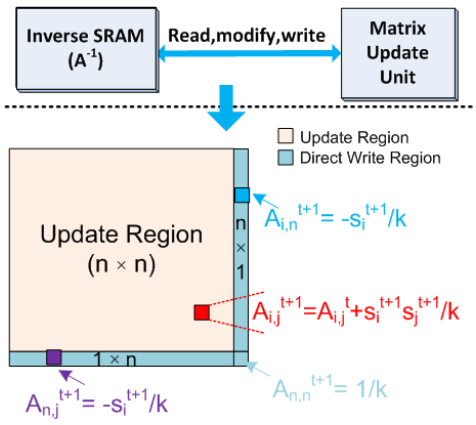

Fig. 6. The regions of Inverse SRAM (logically).

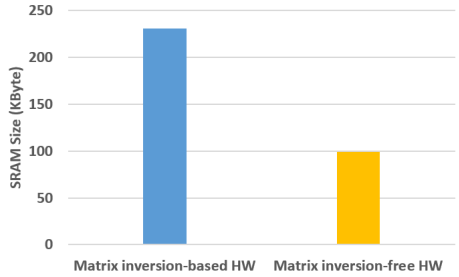

Fig. 7. The SRAM size comparison between MGS-based architecture and matrix inversion-free architecture.

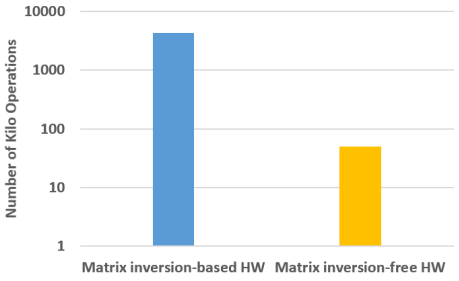

Fig. 8. The number of operations comparison between MGS-based architecture and matrix inversion-free architecture.

IV. Evaluation

Evaluation Setting. In our evaluation the size of on-chip SRAM is configured to support the number of observation points as large as 128. For those applications which need more data samples, external DRAM is necessary. The Inversion SRAM and SRAM_Q in Fig. 2 has the same size to the Covariance SRAM, and SRAM_R, as the memory stores the upper triangular matrix, only requires half the size of Covariance SRAM. The detailed SRAM cost is analyzed as follows.

1) For MGS-based matrix inversion, it needs two 16384×32 SRAM to store matrix **Q**, and the final result, respectively. It also needs a 8256×32 SRAM to store matrix **R** and a 128×32 SRAM to store matrix **W**.
2) For matrix inversion-free solution, it only needs a 16384×32 SRAM to store the final result and a 128×32 intermediate SRAM to store vector **s**.
3) Both of these two architectures need a common Covariance SRAM and a **K*** SRAM, whose sizes are 16384×32 and 128×32, respectively.

In overall, it is seen that the GP accelerator based on MGS matrix inversion takes 230654 Bytes SRAM, and the accelerator using matrix inversion-frees scheme only takes 99072 Bytes. Fig. 7 and 8 compares the SRAM cost and operation cost, respectively, according to the evaluation setting. It is seen that utilizing the matrix inversion-free scheme brings more than half reduction on SRAM cost and about $85\times$ reduction on operations.

We further develop the hardware architecture using Verilog HDL, and syntheis this RTL model using Synopsys Design Compiler with 28nm CMOS technology. The synthesis result

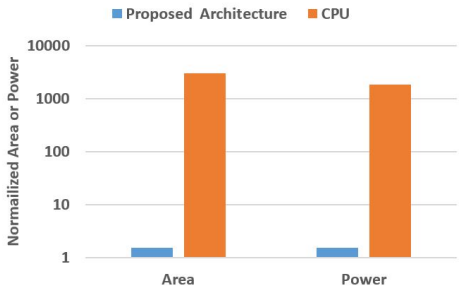

Fig. 9. The comparison with CPU.

shows that the entire hardware architecture has an area of 0.47 mm^2 and consumes 49 mW power. As shown in Fig. 9, the proposed hardware accelerator achieves $2991\times$ and $1857\times$ less area and power consumption, respectively, as compared to conventional software implementation using Intel Core i7-7700K CPU.

V. Conclusion

In this paper we develop two hardware architecture Gaussian Process. One architecture is a general solution and another one is specifically optimized for the scenario when data point is gradually observed. Evaluation result shows that the proposed hardware accelerator provides significant hardware performance improvement than the general-purpose CPU.

VI. Acknowledgement

This work is funded by National Science Foundation Award CNS-1932370.

References

[1] J. Quiñonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate gaussian process regression," *Journal of Machine Learning Research*, vol. 6, no. Dec, pp. 1939–1959, 2005.

[2] R. Urtasun, D. J. Fleet, and P. Fua, "3d people tracking with gaussian process dynamical models," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1. IEEE, 2006, pp. 238–245.

[3] J. Wang, A. Hertzmann, and D. J. Fleet, "Gaussian process dynamical models," in *Advances in neural information processing systems*, 2006, pp. 1441–1448.

[4] T.-T. Lu and S.-H. Shiou, "Inverses of $2\times$ 2 block matrices," *Computers & Mathematics with Applications*, vol. 43, no. 1-2, pp. 119–129, 2002.

[5] P. Luethi, C. Studer, S. Duetsch, E. Zgraggen, H. Kaeslin, N. Felber, and W. Fichtner, "Gram-schmidt-based qr decomposition for mimo detection: Vlsi implementation and comparison," in *APCCAS 2008-2008 IEEE Asia Pacific Conference on Circuits and Systems*. IEEE, 2008, pp. 830–833.

[6] M. Karkooti, J. R. Cavallaro, and C. Dick, "Fpga implementation of matrix inversion using qrd-rls algorithm," in *Asilomar Conference on Signals, Systems, and Computers*, 2005, pp. 1625–1629.

[7] S. Aslan, E. Oruklu, and J. Saniie, "Realization of area efficient qr factorization using unified division, square root, and inverse square root hardware," in *2009 IEEE International Conference on Electro/Information Technology*. IEEE, 2009, pp. 245–250.

[8] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2012, vol. 3.

[9] A. Bjorck, *Numerical methods for least squares problems*. Siam, 1996, vol. 51.

[10] C. K. Singh, S. H. Prasad, and P. T. Balsara, "Vlsi architecture for matrix inversion using modified gram-schmidt based qr decomposition," in *20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07)*, 2007, pp. 836–841.