FPGA-Accelerated ADMM Distributed Utilizing RDMA

Bavly Shehata (bsheh002@ucr.edu)
University of California, Riverside

Prithviraj Yuvaraj (pyuva001@ucr.edu)
University of California, Riverside

Supervisor: Professor Philip Brisk (philip.brisk@ucr.edu)
University of California, Riverside

1 Introduction

Basis Pursuit, solved via the Alternating Direction Method of Multipliers (ADMM) first discussed in [1], is an optimization problem widely used in sparse signal recovery, compressed sensing, and optimization problems. However, the iterative matrix–vector multiplications and convergence checks make it computationally intensive.

The ADMM formulation of Basis Pursuit is

$$\begin{aligned} \text{minimize} \quad & f(x) + \|z\|_1 \\ \text{subject to} \quad & x - z = 0 \end{aligned}$$

where f is the indicator function of $\{x \in \mathbb{R}^n \mid Ax = b\}$ with variable $x \in \mathbb{R}^n$ and user-provided constant inputs $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$, with $m < n$.

Two matrices and one vector are computed prior to iterating:

$$Q = A^T(AA^T)^{-1} \in \mathbb{R}^{n \times m}, \quad P = I - QA \in \mathbb{R}^{m \times m}, \quad Qb \in \mathbb{R}^n.$$

Only P and Qb are retained. Each ADMM iteration is performed as:

$$\begin{aligned} x^{k+1} &= P(z^k - u^k) + Qb \\ z^{k+1} &= S_{1/\rho}(x^{k+1} + u^k) \\ u^{k+1} &= u^k + x^{k+1} - z^{k+1}. \end{aligned}$$

2 Project Overview

Our project introduces an FPGA-accelerated solution using the AMD Alveo U280 for efficient ADMM-based Basis Pursuit designed and implemented in Vitis HLS 2023.2. The original MATLAB implementation [2] was used as reference. Our approach focuses on chunked processing of large problem sizes, parallelizing matrix–vector operations across multiple HBM memory banks, and optimizing kernel dataflow for high throughput. Future extensions will include Remote Direct Memory Access (RDMA) to enable distributed multi-FPGA scaling.

3 Key Highlights

- **Chunked ADMM Solver Implementation:** A kernel (`krnl_bp`) performs iterative ADMM updates on sub-blocks of the data, allowing matrices larger than on-chip memory to be streamed sequentially.

- **Optimized Memory-Mapped Buffers:** Host buffers are allocated with alignment and partitioning across U280 HBM channels, improving bandwidth utilization.
- **Performance Improvements:** HLS pragmas such as pipelining, dataflow, and array partitioning significantly reduce per-iteration latency.

4 How to Run

Prerequisites

- Xilinx Vitis 2023.2
- make
- Platform file: `xilinx_u280_xdma_201920_3`

Steps

1. Clone the repository:

```
git clone https://github.com/bavlyshehata/ADMM_BP/
cd ADMM_BP
```

2. Navigate to the Alveo folder:

```
cd alveo
```

3. Set environment variables in the Makefile:

```
TARGET=hw or hw_emu
PLATFORM=xilinx_u280_xdma_201920_3
```

4. Build xclbin and run:

```
make run TARGET=hw DEVICE=/opt/xilinx/platforms/\
xilinx_u280_gen3x16_xdma_1_202211_1/\
xilinx_u280_gen3x16_xdma_1_202211_1.xpfm
```

5 Implementation Details

5.1 Chunked Processing

Input matrices are partitioned row-wise into configurable `chunk_size` blocks. Each FPGA kernel processes a chunk sequentially, streams partial results, and merges them on host memory.

5.2 Iterative ADMM Updates

Kernels compute updates of the form:

$$x^{k+1} = \arg\min_x \ \tfrac{1}{2}\|Ax - b\|^2 + 2\rho\|x - z^k + u^k\|^2$$

5.3 Multi-Bank Memory Mapping

Data is mapped across HBM banks (0:31), ensuring parallel access for vector operations.

6 Memory Interface

Precomputation

On the host, we compute:

$$G = A^T A, \quad G_\rho = G + \rho I,$$

and then invert G_ρ to compute

$$P = \rho G_\rho^{-1}, \quad Q = G_\rho^{-1} A^T b.$$

These are transferred to FPGA HBM memory. Each ADMM iteration reduces to:

$$x = P(z - u) + Q$$

Host

Large aligned buffers are allocated and partitioned across HBM banks (0:31).

Kernel

The kernel maps buffers to HBM channels via `m_axi` ports, enabling parallel matrix–vector operations for each chunk.

7 Hardware Architecture

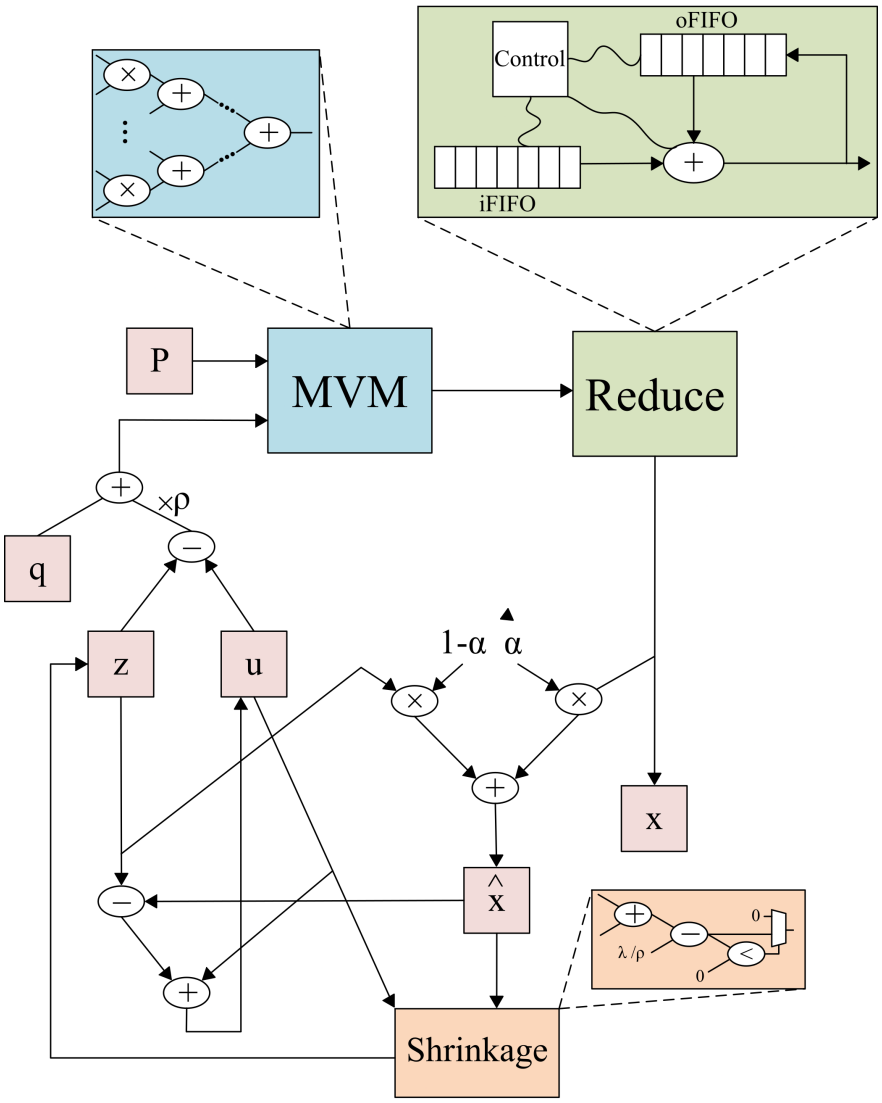

Figure 1: Hardware architecture of the FPGA ADMM solver.

8 Performance

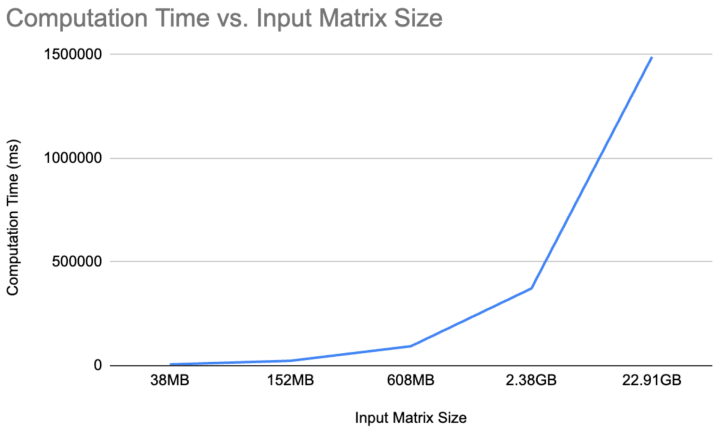

Figure 2: Performance of FPGA ADMM solver on the U280.

Size	N	Migrate to Device (ms)	Kernel Execution (ms)	Total
38MB	2048	9	5818	5827
152MB	4096	17	23269	23286
608MB	8192	43	93608	93651
2.38GB	16384	145	372252	372397
22.91GB	32768	535	1488970	1489505

Table 1: Measured runtime performance. Transfer back to host is negligible.

9 Future Work

- **RDMA:** Extend solver to multi-FPGA setups with low-latency inter-node communication.
- **Termination Check:** Implement early stopping based on convergence to reduce wasted iterations and improve adaptability.

10 Citations

References

[1] Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2010). *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Foundations and Trends in Machine Learning, 3(1), 1–122. https://doi.org/10.1561/2200000016

[2] Boyd, S. *Basis Pursuit MATLAB implementation*. https://stanford.edu/~boyd/papers/admm/basis_pursuit/basis_pursuit.html