

 Open access • Proceedings Article • DOI:10.1109/GLOBALSIP.2016.7905910

Decentralized beamforming for massive MU-MIMO on a GPU cluster

— [Source link](#) 

Kaipeng Li, Riski Skaran, Yujun Chen, Joseph R. Cavallaro ...+2 more authors

Institutions: Rice University, Cornell University, University of Maryland, College Park

Published on: 01 Dec 2016 - IEEE Global Conference on Signal and Information Processing

Topics: Precoding, WSDMA, Beamforming, Multi-user MIMO and GPU cluster

Related papers:


- [Decentralized data detection for massive MU-MIMO on a Xeon Phi cluster](#)
- [Massive MIMO for next generation wireless systems](#)
- [Decentralized Baseband Processing for Massive MU-MIMO Systems](#)
- [What Will 5G Be](#)
- [On the achievable rates of decentralized equalization in massive MU-MIMO systems](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/decentralized-beamforming-for-massive-mu-mimo-on-a-gpu-vuhvv4tq6k>

Decentralized beamforming for massive MU-MIMO on a GPU cluster

Conference Paper**Author(s):**

Li, Kaipeng; Skaran, Riski; Chen, Yujun; Cavallaro, Joseph R.; Goldstein, Tom; [Studer, Christoph](#) 

Publication date:

2016

Permanent link:

<https://doi.org/10.3929/ethz-b-000455405>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)

Originally published in:

<https://doi.org/10.1109/GlobalSIP.2016.7905910>

DECENTRALIZED BEAMFORMING FOR MASSIVE MU-MIMO ON A GPU CLUSTER

Kaipeng Li¹, Rishi Sharan², Yujun Chen¹, Joseph R. Cavallaro¹, Tom Goldstein³, and Christoph Studer²

¹Department of Electrical and Computer Engineering, Rice University, Houston, TX

²School of Electrical and Computer Engineering, Cornell University, Ithaca, NY

³Department of Computer Science, University of Maryland, College Park, MD

ABSTRACT

In the massive multi-user multiple-input multiple-output (MU-MIMO) downlink, traditional centralized beamforming (or precoding), such as zero-forcing (ZF), entails excessive complexity for the computing hardware, and generates raw baseband data rates that cannot be supported with current interconnect technology and chip I/O interfaces. In this paper, we present a novel *decentralized beamforming* approach that partitions the base-station (BS) antenna array into separate clusters, each associated with independent computing hardware. We develop a decentralized beamforming algorithm that requires only local channel state information and minimum exchange of consensus information among the clusters. We demonstrate the efficacy and scalability of decentralized ZF beamforming for systems with hundreds of BS antennas using a reference implementation on a GPU cluster.

1. INTRODUCTION

Massive MU-MIMO is believed to be a key technology for realizing high spectral efficiency and link reliability in 5G wireless systems [1]. In the massive MU-MIMO downlink, data is transmitted from a base-station (BS) with hundreds or thousands of antennas to tens of user terminals simultaneously and in the same frequency band [2, 3]. In the downlink, the BS must perform beamforming to mitigate multi-user interference (MUI), which can be accomplished with linear beamforming algorithms that use channel state information acquired in the uplink (users transmit pilot signals to the BS).

1.1. Limits of Centralized Baseband Processing

Existing algorithms that realize the full benefits of massive MU-MIMO in systems with realistic antenna configurations, such as zero-forcing (ZF) beamforming [4], rely on *centralized baseband processing*. This approach requires that all channel state information must be available at a centralized processing node where a beamforming algorithm computes all baseband

signals that are transmitted to the radio-frequency (RF) chains. Such a traditional centralized processing approach poses significant practical challenges for massive MU-MIMO systems.

Consider a 128 BS antenna system with 40 MHz bandwidth. For such a system, the raw baseband data rates to be transmitted from the centralized processing node to the RF chains easily exceed 200 Gb/s. Such high data rates not only pose severe implementation challenges for the computing hardware that carries out the beamforming algorithms, but the resulting baseband data stream also exceeds the I/O bandwidth of integrated circuits and of high-speed interconnects, such as the common public radio interface (CPRI) [5]. In fact, existing massive MU-MIMO testbeds, such as the Argos testbed [6], have shown that centralized baseband processing required for ZF beamforming is infeasible with current computing hardware and interconnect technology. Hence, existing testbeds use maximum ratio combining (MRC), which enables fully decentralized beamforming at the antenna elements at significantly reduced spectral efficiency [4].

1.2. Contributions

This paper proposes a novel, decentralized beamforming architecture for massive MU-MIMO systems. Our method partitions the BS antennas into independent clusters, which perform beamforming using the alternating direction method of multipliers (ADMM) [7, 8] in a decentralized manner. Each antenna cluster performs beamforming locally for the associated RF elements, with only little information exchange between the clusters. Furthermore, each cluster only needs access to local channel state information. To demonstrate the efficacy of our approach, we provide reference implementation results on a GPU cluster. Our results show that throughputs in the Gb/s regime can be achieved in a decentralized architecture, while providing superior error-rate performance compared to MRC.

2. DECENTRALIZED BEAMFORMING

2.1. Downlink System Model

We consider a massive MU-MIMO OFDM downlink system, where the BS is equipped with B antennas and serves $U \leq B$

This work was supported by Xilinx Inc., the US NSF under grants CNS-1265332, ECCS-1232274, ECCS-1408370, ECCS-1408006, CCF-1535902, and the US Office of Naval Research under grant N00014-15-1-2676. The authors would like to thank the DoD High Performance Computing Center.

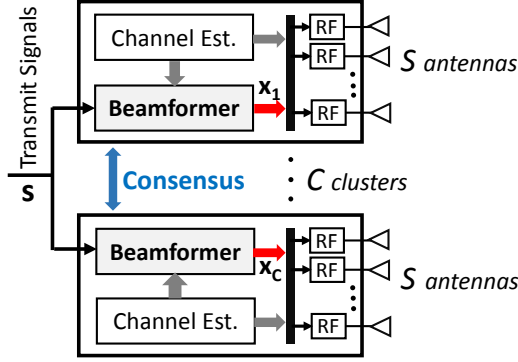


Fig. 1: Proposed decentralized beamforming architecture.

users. The BS performs beamforming to mitigate MUI and computes $\mathbf{x} \in \mathbb{C}^B$ on each OFDM subcarrier using the associated transmit data symbol $\mathbf{s} \in \mathcal{O}^U$ where \mathcal{O} is a constellation (e.g., 16-QAM). The beamformed signal \mathbf{x} is then transmitted over the downlink channel, which is modeled as $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}$ per subcarrier. Here, $\mathbf{H} \in \mathbb{C}^{U \times B}$ is the downlink channel matrix, which is the transpose of the uplink channel matrix and acquired via training; the vectors $\mathbf{y} \in \mathbb{C}^U$ and $\mathbf{n} \in \mathbb{C}^U$ contain the received signals at each user and noise, respectively.

2.2. Decentralized Beamforming Architecture

We solve the following beamforming (or precoding) problem:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathbb{C}^B} \|\mathbf{x}\|_2 \quad \text{subject to } \mathbf{s} = \mathbf{H}\mathbf{x}, \quad (1)$$

which minimizes the instantaneous energy of the transmit signal $\hat{\mathbf{x}}$ while satisfying the so-called *beamforming constraint* $\mathbf{s} = \mathbf{H}\hat{\mathbf{x}}$. By transmitting $\hat{\mathbf{x}}$, the equivalent downlink input-output relation is given by $\mathbf{y} = \mathbf{s} + \mathbf{n}$, which contains no interference across users. We note that (1) is the well-known ZF beamformer, which can be computed in closed form by $\hat{\mathbf{x}} = \mathbf{H}^H(\mathbf{H}\mathbf{H}^H)^{-1}\mathbf{s}$ assuming that \mathbf{H} is full rank.

To avoid a centralized computation of $\hat{\mathbf{x}}$, we propose to compute the solution to (1) in a decentralized fashion. Figure 1 shows the proposed architecture. We partition the BS antenna array into C clusters of equal size, such that $B = CS$, where S is number of BS antennas associated with each cluster. Each of the C clusters performs decentralized beamforming using only local¹ channel state information $\mathbf{H}_c \in \mathbb{C}^{U \times S}$, $c = 1, 2, \dots, C$, where $\mathbf{H} = [\mathbf{H}_1 \ \mathbf{H}_2 \ \dots \ \mathbf{H}_C]$, and with a minimum amount of consensus information exchange across the clusters.

2.3. Decentralized Beamforming via ADMM

By introducing C auxiliary variables $\mathbf{z}_c = \mathbf{H}_c\mathbf{x}_c$, $c = 1, 2, \dots, C$, we can rewrite the beamforming problem (1) as

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathbb{C}^B} \|\mathbf{x}\|_2 \quad \text{subject to } \mathbf{s} = \sum_{c=1}^C \mathbf{z}_c. \quad (2)$$

¹Each cluster performs independent channel estimation in the uplink and channel state information remains at each cluster and will *not* be distributed.

Algorithm 1 Decentralized Beamforming Algorithm

- 1: **Input:** $\mathbf{s}, \mathbf{H}_c, c = 1, 2, \dots, C, \rho, \gamma$
- 2: **if** $S \leq U$ **then**
- 3: $\mathbf{A}_c^{-1} = (\mathbf{H}_c^H \mathbf{H}_c + \rho^{-1} \mathbf{I}_S)^{-1}$
- 4: $\mathbf{Q}_c = \mathbf{A}_c^{-1} \mathbf{H}_c^H$, $\mathbf{P}_c = \mathbf{H}_c \mathbf{Q}_c$
- 5: **else**
- 6: $\mathbf{B}_c^{-1} = (\mathbf{H}_c \mathbf{H}_c^H + \rho^{-1} \mathbf{I}_U)^{-1}$
- 7: $\mathbf{Q}_c = \mathbf{H}_c^H \mathbf{B}_c^{-1}$, $\mathbf{P}_c = \mathbf{H}_c \mathbf{Q}_c$
- 8: **Init:** $\mathbf{z}_c^{(1)} = \max\{U/B, 1/C\}\mathbf{s}$, $\lambda_c^{(1)} = \mathbf{0}$, $\mathbf{x}_c^{(1)} = \mathbf{Q}_c \mathbf{z}_c^{(1)}$
- 9: **for** $t = 2, 3, \dots, T$ **do** /* $T = \max.$ iteration number */
- 10: $\mathbf{m}_c = \mathbf{P}_c(\mathbf{z}_c^{(t-1)} + \lambda_c^{(t-1)})$
- 11: $\mathbf{w}_c = \mathbf{m}_c - \lambda_c^{(t-1)}$
- 12: $\mathbf{w} = \sum_{c=1}^C \mathbf{w}_c$ /* Consensus */
- 13: $\mathbf{z}_c^{(t)} = \mathbf{w}_c + C^{-1}(\mathbf{s} - \mathbf{w})$
- 14: $\lambda_c^{(t)} = \lambda_c^{(t-1)} - \gamma(\mathbf{m}_c - \mathbf{z}_c^{(t)})$
- 15: $\mathbf{x}_c^{(t)} = \mathbf{Q}_c(\mathbf{z}_c^{(t)} + \lambda_c^{(t)})$
- 16: **Output:** $\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \dots; \mathbf{x}_C]$

The solution to (2) corresponds to a saddle point of the so-called scaled augmented Lagrangian function [9] defined as

$$\mathcal{L}(\mathbf{s}, \mathbf{z}, \boldsymbol{\lambda}) = \frac{1}{2} \|\mathbf{x}\|_2^2 + \sum_{c=1}^C \frac{\rho}{2} \|\mathbf{H}_c \mathbf{x}_c - \mathbf{z}_c - \lambda_c\|_2^2 + \mathcal{X}(\mathbf{z}), \quad (3)$$

where $\rho > 0$ is a regularization parameter, the vector $\boldsymbol{\lambda} = [\lambda_1; \lambda_2; \dots; \lambda_C]$ contains C Lagrange multipliers, and $\mathcal{X}(\mathbf{z})$ is the characteristic function for the affine constraint in (2), i.e., $\mathcal{X}(\mathbf{z}) = 0$ if $\mathbf{s} = \sum_{c=1}^C \mathbf{z}_c$ and $\mathcal{X}(\mathbf{z}) = \infty$ otherwise.

To solve (3) in a decentralized fashion, we use the ADMM framework [9]. We initialize $\mathbf{z}_c^{(1)} = \max\{U/B, 1/C\}\mathbf{s}$ and $\lambda_c^{(1)} = \mathbf{0}$, $c = 1, 2, \dots, C$. We then perform the following three-step procedure for the iterations $t = 1, 2, \dots$ until convergence or a maximum number of iterations has been reached:

$$\mathbf{x}_c^{(t+1)} = \arg \min_{\mathbf{x}_c \in \mathbb{C}^S} \frac{1}{2} \|\mathbf{x}_c\|_2^2 + \frac{\rho}{2} \|\mathbf{H}_c \mathbf{x}_c - \mathbf{z}_c^{(t)} - \lambda_c^{(t)}\|_2^2 \quad (4)$$

$$\mathbf{z}^{(t+1)} = \arg \min_{\mathbf{z}_c \in \mathbb{C}^U} \sum_{c=1}^C \frac{\rho}{2} \|\mathbf{H}_c \mathbf{x}_c^{(t+1)} - \mathbf{z}_c - \lambda_c^{(t)}\|_2^2 + \mathcal{X}(\mathbf{z}) \quad (5)$$

$$\lambda_c^{(t+1)} = \lambda_c^{(t)} - \gamma(\mathbf{H}_c \mathbf{x}_c^{(t+1)} - \mathbf{z}_c^{(t+1)}), \quad c = 1, \dots, C. \quad (6)$$

Here, \mathbf{z} is the consensus vector defined as $\mathbf{z} = [\mathbf{z}_1; \mathbf{z}_2; \dots; \mathbf{z}_C]$, and $\gamma > 0$ is a suitably-chosen stepsize parameter.

Equation (4) is a least-squares problem that can be solved independently in every cluster in closed-form as follows:

$$\mathbf{x}_c^{(t+1)} = \mathbf{A}_c^{-1} \mathbf{H}_c^H (\mathbf{z}_c^{(t)} + \lambda_c^{(t)}), \quad (7)$$

where $\mathbf{A}_c^{-1} = (\mathbf{H}_c^H \mathbf{H}_c + \rho^{-1} \mathbf{I}_S)^{-1}$, which requires the computation of an $S \times S$ matrix inverse. To reduce the amount of recurrent computations, we can precompute $\mathbf{A}_c^{-1} \mathbf{H}_c^H$ and reuse the result during the algorithm iterations. For situations where the cluster size S is larger than the number of users U , we can perform an alternative update that requires the inversion of a smaller matrix. In particular, we have the following

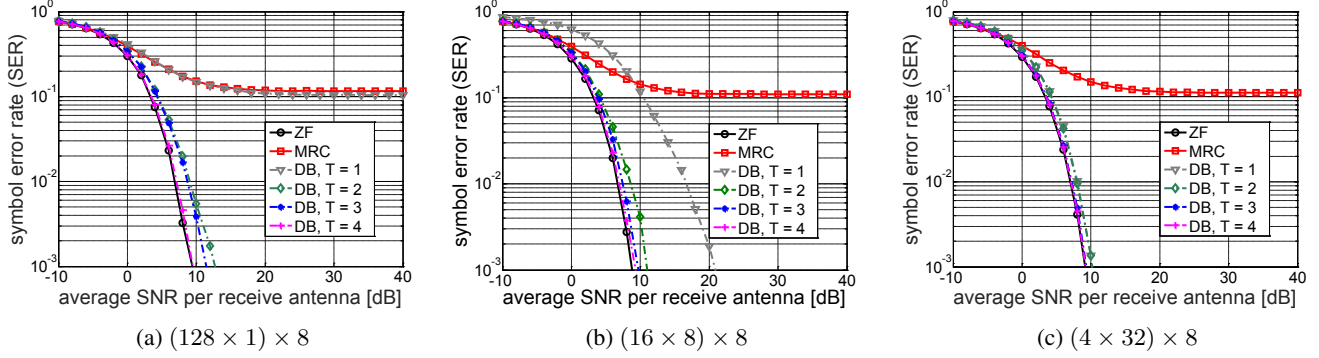


Fig. 2: Symbol error-rate (SER) performance of decentralized beamforming (DB); we use the notation $(C \times S) \times U$.

equivalent update:

$$\mathbf{x}_c^{(t+1)} = \mathbf{H}_c^H \mathbf{B}_c^{-1} (\mathbf{z}_c^{(t)} + \lambda_c^{(t)}). \quad (8)$$

Here, $\mathbf{B}_c^{-1} = (\mathbf{H}_c \mathbf{H}_c^H + \rho^{-1} \mathbf{I}_U)^{-1}$, which requires the computation of an $U \times U$ matrix inverse.

Equation (5) can be carried out efficiently in our decentralized architecture, but requires the exchange of consensus information. To show this, we rewrite (5) as follows:

$$\mathbf{z}^{(t+1)} = \arg \min_{\mathbf{z} \in \mathcal{C}^{UC}, \mathbf{s} = \mathbf{D}\mathbf{z}} \frac{1}{2} \|\mathbf{w} - \mathbf{z}\|_2^2, \quad (9)$$

where we define $\mathbf{D} = \mathbf{1}_{1 \times C} \otimes \mathbf{I}_U$ with the Kronecker product \otimes , and $\mathbf{w}^T = [\mathbf{w}_1^T \cdots \mathbf{w}_C^T]$ with $\mathbf{w}_c = \mathbf{H}_c \mathbf{x}_c^{(t+1)} - \lambda_c^{(t)}$. The problem (9) denotes the orthogonal projection of \mathbf{w} onto the constraint $\mathbf{s} = \mathbf{D}\mathbf{z}$, which has a closed-form solution [10]:

$$\mathbf{z}^{(t+1)} = \mathbf{w} + \mathbf{D}^H (\mathbf{D}\mathbf{D}^H)^{-1} (\mathbf{s} - \mathbf{D}\mathbf{w}). \quad (10)$$

Since $(\mathbf{D}\mathbf{D}^H)^{-1} = C^{-1} \mathbf{I}_U$ and $\mathbf{D}^H \mathbf{D} = \mathbf{1}_{C \times C} \otimes \mathbf{I}_U$, we have the following equivalent result

$$\mathbf{z}^{(t+1)} = \mathbf{w} + (C^{-1} \mathbf{D}^H \mathbf{s} - C^{-1} (\mathbf{1}_{C \times C} \otimes \mathbf{I}_U) \mathbf{w}), \quad (11)$$

which can be rewritten using per-cluster variables as

$$\mathbf{z}_c^{(t+1)} = \mathbf{w}_c + (C^{-1} \mathbf{s} - \mathbf{v}) \quad (12)$$

with $\mathbf{v} = C^{-1} \sum_{c=1}^C \mathbf{w}_c$ and $\mathbf{w}_c = \mathbf{H}_c \mathbf{x}_c^{(t+1)} - \lambda_c^{(t)}$. Evidently, (12) only involves simple averaging across the clusters, which will be carried out in the consensus phase.

Equation (6) is straightforward and can be carried out independently at every cluster.

The resulting decentralized beamforming procedure is summarized in Algorithm 1. To arrive at an efficient implementation of the above decentralized beamforming algorithm, we compute the initial values of $\mathbf{z}_c^{(1)}$, $\lambda_c^{(1)}$, $\mathbf{x}_c^{(1)}$ and perform ADMM iterations in the order of \mathbf{z}_c , λ_c , \mathbf{x}_c considering \mathbf{x}_c is the final output of the local beamformer. To avoid redundant computations during ADMM iterations, we also compute intermediate variables \mathbf{P}_c and \mathbf{Q}_c defined at lines 4 and 7.

2.4. Simulation Results

We simulate the symbol error-rate (SER) of the proposed decentralized beamformer (DB) in a massive MIMO system with 128 total BS antennas and 8 user antennas with 16-QAM modulation. Figure 2 compares the SER for different algorithm iterations and various antenna configurations. We also compare DB with centralized ZF beamforming and fully decentralized MRC beamforming as a baseline. We see that our DB achieves near-optimal SER performance for a very small number of iterations for various combinations of C and S , even with the fully distributed configuration ($C = 128$, $S = 1$). For larger cluster sizes, e.g., $S = 32$, a single iteration is sufficient to approach the performance of centralized ZF beamforming. These results demonstrate the effectiveness of our approach.

3. IMPLEMENTATION ON A GPU CLUSTER

We now describe an implementation of Algorithm 1 on a GPU cluster for the architecture shown in Figure 1. Here, we generate C total processes on the GPU cluster, with each controlling a node for accelerating local beamforming computations on the GPU using compute unified device architecture (CUDA) [11]. Those GPU nodes are connected with high-bandwidth networking interfaces and communicate using the message passing interface (MPI) [12] among their controlling processes for local information collection and consensus sharing. Our software-defined implementation demonstrates the potential data-rate performance and design scalability of decentralized beamforming on modern hardware platforms.

3.1. Accelerating Kernel Computation

We implement the local beamforming computations corresponding to S local BS antennas by GPU kernel functions, which can be launched with thousands of parallel threads on thousands of GPU computing cores within a certain node for acceleration. As shown in Algorithm 1, the dominant computations are matrix-matrix or matrix-vector multiplications and matrix inversions, which can be efficiently implemented using the *cuBLAS* library—a

CUDA-based basic linear algebra subprograms (BLAS) library [13] targeting GPUs, with automatically tuned number of threads and thread-blocks according to computing workloads. Specifically, we use `cublasCgemvBatched` function for fast matrix-matrix or matrix-vector multiplications and use `cublasCgetrfBatched` function followed by `cublasCgetriBatched` function for fast matrix inversion based on the Cholesky decomposition. Here, we select the *Batched* version of *cuBLAS* so that the function call can perform the computation for a batch of matrix computations, which, for example, corresponds to a batch of subcarriers considering that the local beamforming is performed on a per-subcarrier basis, in order to achieve high utilization of GPU resources and to enable high throughput.

We define N_{sym} OFDM symbols, each including N_{sc} subcarriers, as the total local beamforming workload in each execution of beamforming kernel flow. As mentioned before, to avoid redundant computations in ADMM iterations, we calculate matrix \mathbf{P}_c and \mathbf{Q}_c (line 4 or 7 of Algorithm 1) as intermediate results before the iteration starts. By assuming that the channel \mathbf{H}_c is static across every N_{sym} symbols within channel coherence time, we can calculate matrix \mathbf{P}_c and \mathbf{Q}_c , which depend on \mathbf{H}_c , only for N_{sc} subcarriers in an OFDM symbol using the above *cuBLAS* functions with `batchsize = N_{sc}` , and then, broadcast the results to N_{sym} OFDM symbols inside GPU device memory to save computing complexity and latency. We note, however, that for batched matrix-matrix or matrix-vector multiplications during ADMM iterations (line 9-15 of Algorithm 1), we must launch *cuBLAS* functions with `batchsize = $N_{sc} \times N_{sym}$` since those computations are dependent not only on \mathbf{H}_c , but also on the transmit symbols.

The parameter update procedure during each ADMM iteration requires some other types of local computations such as vector addition, subtraction and scaling. Instead of resorting to *cuBLAS* functions and exchanging results in between via slow GPU device memory, we design customized kernel functions, where we can combine several steps of computation, for example, the vector operations on lines 13-15 of Algorithm 1, into a single `par_update` kernel, to utilize local registers to store and share intermediate results. We launch the customized kernels with $N_{sc} \times N_{sym} \times U$ threads to perform those vector operations during ADMM iteration in parallel under a per-sample basis to exploit the data-level parallelism.

3.2. Reducing Message Passing Overhead

During the ADMM iteration, C MPI processes running on the GPU cluster need to perform collective communication with message size of $N_{sc} \times N_{sym} \times U$ complex samples on each node, for gathering local \mathbf{w}_c from C GPU nodes, summing up to consensus \mathbf{w} , and broadcasting \mathbf{w} back to all the nodes. Those three steps can be realized by a single MPI function call, `MPI_Allreduce` with sum operation, or equivalently by `MPI_Reduce` for gathering and summing up

Table 1: Latency (L) and throughput (T) performance.

$U=16$	$C=8, B=64$	$C=16, B=128$	$C=32, B=256$
Iter.	L(ms) / T(Mb/s)	L(ms) / T(Mb/s)	L(ms) / T(Mb/s)
1	0.747 / 1079.5	0.747 / 1079.5	0.749 / 1076.6
2	2.930 / 275.2	3.032 / 266.0	3.106 / 259.6
3	4.964 / 162.4	5.153 / 156.5	5.289 / 152.5
4	7.006 / 115.1	7.244 / 111.3	7.473 / 107.9

followed by `MPI_Bcast` for broadcasting [12]. Those MPI functions typically operate on the CPU’s memory, requiring extra GPU-to-CPU memory copy before function call and extra CPU-to-GPU memory copy afterwards. Here, to reduce the message passing overhead, in our design, we utilize *CUDA-aware MPI* [14] and *GPUDirect* remote device memory access (RDMA) [15] techniques, where MPI functions, for example, `MPI_Allreduce`, can operate on GPU memories to realize direct GPU-to-GPU memory copy via modern high bandwidth network interfaces, such as Infiniband, or Cray Aries [16], etc., eliminating unnecessary GPU-to-CPU and CPU-to-GPU memory copy operations and reducing the total latency.

4. PERFORMANCE RESULTS

We implemented our design on a Cray XC30 cluster [17], hosted by the Navy DoD Supercomputing Resource Center, which has a total of 32 GPU nodes. The GPU nodes are connected with Cray Aries network interface, and each node includes a 10-core Intel Xeon E5-2670v2 CPU and an Nvidia Tesla K40 GPU card with 2880 CUDA cores and 12GB GDDR5 memory, running with Cray Linux OS. The hybrid CUDA and MPI source code is compiled with the Nvidia nvcc compiler and the Cray compiler, linked with CUDA’s runtime library, *cuBLAS* library and Cray MPICH2 library. Timing characteristics are measured by CPU wall-clock time with necessary synchronization of CPU timer and GPU kernels.

Table I summarizes latency and throughput performance for different configurations with 64-QAM modulation and beamforming workload of $N_{sym} = 7$ and $N_{sc} = 1200$ corresponding to a “slot” of a 20 MHz LTE frame. We can scale up the total number of BS antennas $B = CS$ by increasing the value of C . Here, we record the data rate performance for $C = 8, C = 16$, and $C = 32$ cases at various ADMM iteration numbers. As it can be seen, our GPU cluster implementation achieves over 1.075 Gb/s at 1 iteration, and over 255 Mb/s at 2 iterations, which is sufficient for near-optimal error-rate performance as discussed in Section 2.4. Interestingly, by increasing the number of clusters C , our design suffers only from very little throughput degradation, demonstrating that our decentralized architecture provides scalability on modern hardware platforms to support hundreds to thousands of BS antennas in practice. We conclude by noting that the throughput of our approach can potentially be increased by an order of magnitude with decentralized FPGA or ASIC implementations.

5. REFERENCES

- [1] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. Soong, and J. C. Zhang, "What will 5G be?," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065–1082, June 2014.
- [2] F. Rusek, D. Persson, B. K. Lau, E. G. Larsson, T. L. Marzetta, O. Edfors, and F. Tufvesson, "Scaling up MIMO: Opportunities and challenges with very large arrays," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 40–60, Jan 2013.
- [3] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta, "Massive MIMO for next generation wireless systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 186–195, Feb. 2014.
- [4] J. Hoydis, S. Ten Brink, and M. Debbah, "Massive MIMO in the UL/DL of cellular networks: How many antennas do we need?," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 2, pp. 160–171, Feb. 2013.
- [5] <http://www.cpri.info>, *Common public radio interface*.
- [6] C. Shepard, H. Yu, N. Anand, E. Li, T. Marzetta, R. Yang, and L. Zhong, "Argos: Practical Many-antenna Base Stations," in *Proc. of the 18th Annual International Conference on Mobile Computing and Networking*, Aug. 2012, pp. 53–64.
- [7] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.
- [8] T. Goldstein, B. O’Donoghue, S. Setzer, and R. G. Baraniuk, "Fast alternating direction optimization methods," *SIAM Journal on Imaging Sciences*, vol. 7, no. 3, pp. 1588–1623, Aug. 2014.
- [9] S. Boyd and L. Vandenberghe, *Convex optimization*, Cambridge university press, 2004.
- [10] C. Studer, T. Goldstein, W. Yin, and R. G. Baraniuk, "Democratic representations," *arXiv preprint: 1401.3420*, Apr. 2015.
- [11] <http://docs.nvidia.com/cuda>, *Nvidia CUDA programming guide*.
- [12] <https://computing.llnl.gov/tutorials/mpi>, *Message passing interface*.
- [13] <http://docs.nvidia.com/cuda/cublas>, *Nvidia cuBLAS library*.
- [14] <https://devblogs.nvidia.com/parallelforall/introduction-cuda-aware-mpi/>, *CUDA-aware MPI*.
- [15] <http://docs.nvidia.com/cuda/gpudirect-rdma>, *GPU Direct RDMA*.
- [16] Bob Alverson, Edwin Froese, Larry Kaplan, and Duncan Roweth, "Cray XC Series Network," *Cray Inc., White Paper WP-Aries01-1112 (2012)*.
- [17] <https://navydsrsrc.hpc.mil>, *Navy DSRC, Cray XC30 User Guide*.