MICCO: An Enhanced Multi-GPU Scheduling Framework for Many-Body Correlation Functions

Qihan Wang*, Bin Ren*, Jie Chen†, and Robert G. Edwards†

*Department of Computer Science, William & Mary, Williamsburg, VA
Email: qwang19@email.wm.edu, bren@wm.edu
†Jefferson Lab, Newport News, VA
Email: {chen, edwards}@jlab.org

2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 978-1-6654-8106-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPDPS53621.2022.00022

***Abstract*—Calculation of many-body correlation functions is one of the critical kernels utilized in many scientific computing areas, especially in Lattice Quantum Chromodynamics (Lattice QCD). It is formalized as a sum of a large number of contraction terms each of which can be represented by a graph consisting of vertices describing quarks inside a hadron node and edges designating quark propagations at specific time intervals. Due to its computation- and memory-intensive nature, real-world physics systems (e.g., multi-meson or multi-baryon systems) explored by Lattice QCD prefer to leverage multi-GPUs. Different from general graph processing, many-body correlation function calculations show two specific features: a large number of computation-/data-intensive kernels and frequently repeated appearances of original and intermediate data. The former results in expensive memory operations such as tensor movements and evictions. The latter offers data reuse opportunities to mitigate the data-intensive nature of many-body correlation function calculations. However, existing graph-based multi-GPU schedulers cannot capture these *data-centric* features, thus resulting in a sub-optimal performance for many-body correlation function calculations.**

To address this issue, this paper presents a multi-GPU scheduling framework, MICCO, to accelerate contractions for correlation functions particularly by taking the *data dimension* (e.g., data reuse and data eviction) into account. This work first performs a comprehensive study on the interplay of *data reuse* and *load balance*, and designs two new concepts: *local reuse pattern* and *reuse bound* to study the opportunity of achieving the optimal trade-off between them. Based on this study, MICCO proposes a heuristic scheduling algorithm and a machine-learning-based regression model to generate the optimal setting of reuse bounds. Specifically, MICCO is integrated into a real-world Lattice QCD system, Redstar, *for the first time* running on multiple GPUs. The evaluation demonstrates MICCO outperforms other state-of-art works, achieving up to $2.25\times$ speedup in synthesized datasets, and $1.49\times$ speedup in real-world correlation functions.

I. INTRODUCTION

Calculation of many-body correlation functions is a key kernel widely used in many scientific physics systems (such as Lattice Quantum Chromodynamics(QCD)) [5], [6], [7], [8], [26], [2]. *Hadronic correlation function* in complex multi-meson and multi-baryon systems is a typical example of many-body correlation function, which involves quarks enclosed in mesons and baryons. Calculating hadronic correlation functions converts a series of quark propagations describing interactions among hadrons into many undirected graphs that have quarks of the hadrons as vertices and quark propagations as edges, followed by performing a graph contraction on every graph that reduces graph edges one after another until only two hadrons are left. Each reduction of an edge is a *tensor contraction* between hadron nodes which is dubbed hadron contraction.

Calculation of many-body correlation functions is computation and memory-intensive because it usually involves many thousands even millions of contractions resulting in extremely large numbers of tensor contractions. Graph contractions also generate a large amount of intermediate data, requiring significant memory resources. Thus, real-world physics systems commonly rely on high-end computing devices like many-core GPUs to compute many-body correlation functions. Specifically, due to the limited memory size of a single GPU, multi-GPU systems are preferred.

However, accelerating the calculation of many-body correlation functions on multi-GPUs is challenging. In contrast to general graph-based applications that process a huge graph on multi-GPUs [4], [9], [16], many-body correlation function calculations are featured with two specific characteristics: First, the entire calculation consists of many computation-/data-intensive kernels that are represented by graph edges. Second, repeated hadron nodes appear frequently because of overlapped reduction paths among multiple contraction graphs. The former shifts the scheduling bottleneck from optimizing graph partition and reducing partition synchronization (as shown frequently in general graph processing) to improving the GPU assignment of these computation kernels to avoid expensive memory operations such as tensor evictions in memory oversubscription situations, or tensor movements. The latter offers unique (and many) data reuse opportunities that potentially mitigate the data-intensive nature of many-body correlation function calculations. Unfortunately, existing multi-GPU scheduling frameworks [29], [4], [1], [15], [18], [24] mainly focus on workload balance without considering the above *data dimension* that is critical to the execution performance of many-body correlation, thus resulting in sub-optimal system performance if they are adopted directly.

To address this issue, this work presents a new multi-GPU scheduling framework, MICCO, to accelerate calculating many-body correlation functions. The key innovation of MICCO is that it brings the *data dimension* into the whole scheduling picture, particularly by studying the impact of a

1530-2075/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPS53621.2022.00022

data reuse-load balance interplay on the scheduling and leveraging this interplay to find the optimal scheduling scheme. The key insight of this study is that data reuse and load balance form a trade-off relationship in scheduling scheme exploring and multiple factors affect this trade-off, rendering it very challenging to find a global optimal scheduling solution within a practical time budget for real-world systems.

Fortunately, this study demonstrates that it is possible to create a highly effective local optimal scheduling with the help of two newly designed concepts including *local reuse pattern* and associated simplified but effective mapping analysis, and *reuse bound* that characterizes the allowed level of load imbalance when exploring data reuse opportunities. Based on both new concepts, this work proposes a heuristic scheduling algorithm that toggles between leveraging data reuse and pursuing load balance particularly by taking memory evictions into account, and designs a machine-learning-based regression model to determine the optimal setting of reuse bounds. MICCO is integrated into a well-known Lattice QCD system, Redstar [6], [7], [8], *for the first time* running it on multiple GPUs.

The main contributions can be summarized as follows:

- For the first time performing a comprehensive study on the interplay between data reuse and load balance in multi-GPU scheduling of many-body correlation function calculations, particularly introducing two new concepts that are critical in multi-GPU scheduler design, *local reuse pattern* and *reuse bound*.
- Based on the previous study, presenting a multi-GPU scheduling framework, MICCO, to accelerate the calculation of many-body correlation functions that consists of a heuristic scheduling algorithm and a regression model to generate optimal settings to balance the impact of data reuse and load balance, particularly by considering memory oversubscription situations.
- Integrating MICCO into a real-world Lattice QCD system, Redstar, and for the first time running it on a multi-GPU environment.

MICCO is extensively evaluated with both synthesized datasets and real-world datasets with varied settings. The evaluation demonstrates that MICCO outperforms other state-of-art works in all situations, achieving up to $2.25\times$ speedup.

II. BACKGROUND

A. Many-body Correlation Function

Hadronic correlation functions are the central quantities to be calculated when determining the properties and interactions of quarks directly from Lattice QCD simulations. Calculation of correlation functions is crucial for generating physics observables and is relevant to experiments planned for Jlab, FAIR, and J-PARC facilities [5], [26], [2]. However, the computational cost of constructing such correlators is, however, known to be exceptionally enormous. The reason for such a high cost comes from computing all required quark propagation diagrams [6] resulting from *Wick contractions* [6], [7], [8]. The number of such diagrams grows factorially as

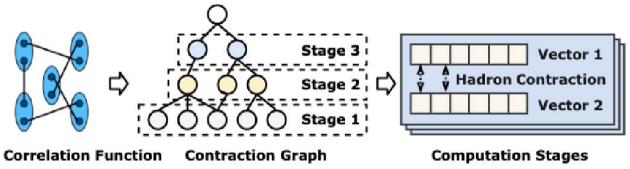

Fig. 1: **Topology Representations of many-body Correlation.** Correlation functions are represented as multiple contraction graphs. Each contraction graph consists of multiple computation stages. Each stage consists of two vectors of independent hadron nodes. Each pair of hadron nodes conducts hadron contractions.

the number of quarks and the total number of freedom of the hadronic systems under consideration increase. A quark propagation diagram can be represented as a graph consisting of a set of hadron nodes each of which has vertices (V) representing the quarks inside a hadron node and undirected edges (E) describing quark propagations at specific time intervals. Especially, the number of unique graphs can be potentially huge approaching in the order of $500,000$. The graph contraction of a graph, which is defined as deleting one edge after another, consists of a series of hadron contractions involving batched matrix multiplications for a meson system or batched tensor contractions for a baryon system[1]. A large number of contraction graphs on many time-slices and the size of matrices/tensors ($\approx 100s$) associated with the hadron nodes present extreme computing challenges. It is paramount to utilize modern computing accelerators such as GPUs to speed up the calculations of hadron contractions.

B. Topological Representations

To translate the statistic definition into a formalized computational problem, Fig.1 illustrates many-body correlation function calculations as a topological representation, in the form of contraction graphs. It is worth noting that a many-body correlation may involve **thousands of contraction graphs** while this figure only shows one for simplicity. In each contraction graph, vertices represent hadron nodes, while edges describe the interactions between hadron nodes. Hadron nodes are formalized as batched matrices or tensors, with different ranks of tensors representing different types of hadron nodes (e.g., matrices in meson systems and three-dimensional tensors in baryon systems, respectively). The associated interactions between two hadron nodes are formalized as matrix multiplications or tensor contractions.

A well-known Lattice QCD system, Redstar [6], [7], [8] first translates each correlation function into a set of unique contraction graphs, and then produces a sequence of hadron contractions from the generated contraction graphs. One correlation function can produce many thousands of contraction graphs. Each graph undergoes a graph contraction process during which one edge after another is reduced until only two

[1]This paper uses *tensor* in the following discussion to refer to both two-dimensional *matrix* and higher dimensional *tensor*.

nodes are left. Again, each reduction of an edge corresponds to a matrix multiplication or tensor contraction.

To leverage the concurrency of many-body correlation calculations, pre-processing, based on dependency analysis is used to partition the computation into several stages (e.g., stages 1, 2, and 3 in Fig.1) with these stages executing sequentially. Each stage contains two vectors and each vector contains independent hadron nodes. Each pair of associated hadron nodes in these two vectors accomplishes hadron contractions. Since the hadron nodes are independent, the hadron contractions can execute concurrently.

C. Challenges and Opportunities

Many-body correlation calculation introduces multiple new (and interesting) challenges to multi-GPU scheduling due to its unique computation patterns:

Calculation consists of many computation-intensive kernels. In contrast to conventional graph processing applications (e.g., BFS, PageRank, and Shortest Path) [29], [25], many-body correlation comprises a large number of small contraction graphs that construct a backbone computation structure, and the overall correlation function consists of many computation-intensive kernels, e.g., matrix multiplications or tensor contractions that are represented by edges of these contraction graphs. The multi-GPU scheduling bottleneck is shifted from graph partition and partition synchronization reduction to proper GPU assignment of these computation-intensive kernels to avoid frequent memory oversubscription and intensive tensor movement.

Contraction graphs may share hadron nodes. A hadron node may appear multiple times in more than one contraction graph in a random manner. The tensors belonging to this hadron participate in multiple computations if this hadron is shared by multiple contraction graphs. This key observation demonstrates that many-body correlation offers many *data reuse* opportunities during its computation. It is critical to take data reuse into account during multi-GPU task allocation. This work particularly studies the interplay between data reuse and load balance and proposes an enhanced scheduler based on this study.

System cannot afford a heavy scheduler. Finding the optimal scheduling scheme for the entire many-body correlation computation is time-consuming because it consists of thousands of kernel computations that involve many matrices and tensors. The resulted searching space is huge. However, due to the computation- and memory-intensive nature, the real-world systems cannot afford a heavy scheduling mechanism. A lightweight approach with a reduced scheduling search space and the limited cost is desired.

III. Interplay between Data Reuse and Load Balance

This section carefully studies the interplay between two scheduling metrics, data reuse and load balance, and their effects on multi-GPU scheduling of many-body correlation calculations. This section further analyzes the impact of multiple key factors on this interplay. This study aims to guide the design of MICCO.

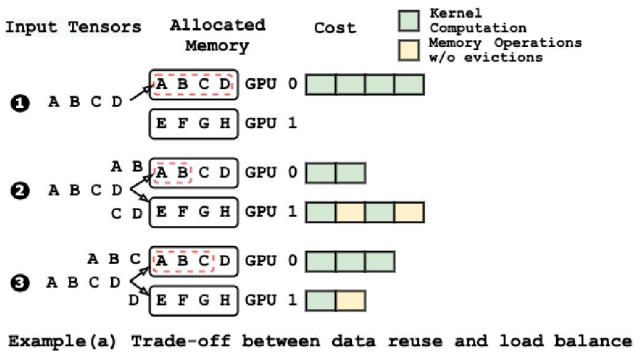

Fig. 2: **Example (a): Trade-off between data reuse and load balance.** Input tensors are A, B, C, and D. Case ❶ only considers data reuse; Case ❷ only cares about load balance; Case ❸ trades off data reuse and load balance. Red dotted frames label reused data. The green bars mean kernel computation cost, and the yellow bars mean memory operation cost (allocation and communication) without memory evictions.

A. Data Reuse and Load Balance Trade-off Analysis

Although improving load balance and data reuse can both lead to better multi-GPU system performance, the multi-GPU scheduler may not be able to achieve optimal for both, simultaneously, e.g., optimizing data reuse may result in unbalanced computation. An interesting trade-off relationship exists between these two metrics. Fig.2 illustrates a detailed example. Assume input data are four tensors (A, B, C, and D) in a vector. If at present time GPU 0 has fetched a copy of these tensors from CPU and GPU 1 stores another set of tensors (E, F, G, and H). In the next step, if only considering data reuse, all input tensors should be assigned to GPU 0 (as shown in case ❶); while if only caring about workload balance, GPU 0 and GPU 1 should fetch the identical amount of tensors (as shown in case ❷). However, both cases result in sub-optimal system performance. Case ❶ only keeps GPU 0 busy, while case ❷ incurs extra memory operations for two tensors (C and D), including two tensor allocations and two tensor movements from CPU to GPU. In contrast to both cases, we point out case ❸ specifically that trades off data reuse and load balance, i.e., assigning three tensors (A, B, and C) to GPU 0 and one tensor (D) to GPU 1. This case results in the best system performance among three schedule schemes.

Fig.3 shows that concerning memory oversubscription, both data reuse (Example (b)) and load balance (Example (c)) are able to reduce memory evictions. Leveraging data reuse decreases the total new memory allocations to avoid memory oversubscription. In Example (b), assume each GPU memory can hold up to four input tensors. Both scheduling cases have balanced workloads, but case ❷ does not reuse the repeated tensors and causes two extra memory operations (including two memory allocations and two tensor movements), and two

TABLE I: **Definition and Impact of Data Characteristics.**

Data Characteristics	Description	Impact on Performance
Tensor Size	Dimension length of a tensor	Computation and Allocation
Vector Size	Number of tensors in one vector	Computation, Allocation, and Communication
Data Distribution	Repeated data follows biased or unbiased distribution	Allocation and Communication
Repeated Rate	Ratio of the repeated data by total data per vector	Allocation and Communication

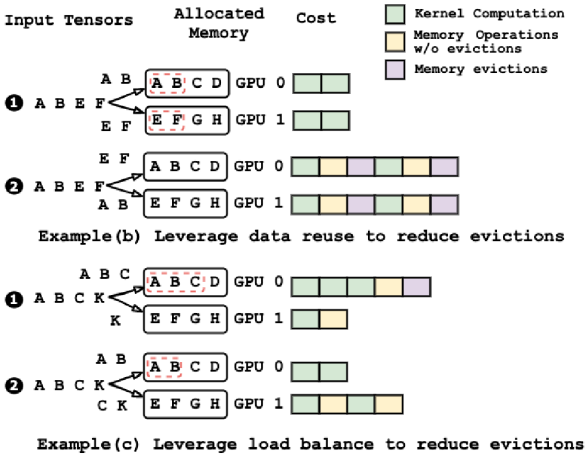

Fig. 3: **Examples: Trade-off between data reuse and load balance regarding memory evictions.**

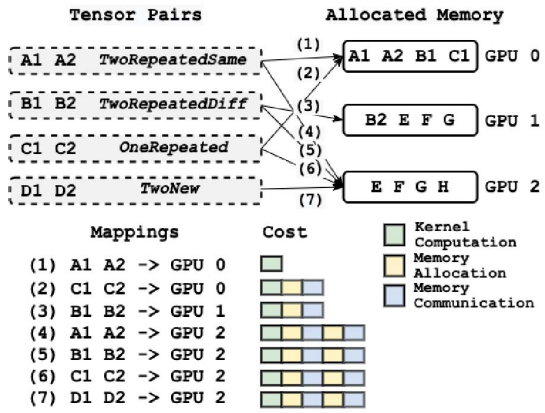

Fig. 4: **Example: Local reuse patterns and task assignments.** Classify tensor pairs based on four local reuse patterns: $TwoRepeatedSame$, $TwoRepeatedDiff$, $OneRepeated$, and $TwoNew$. Mappings between tensor pairs and GPUs can be categorized into seven cases. Mapping (1) represents two reused tensors, assigned to the re-utilized GPU with the least overhead. Mappings (2) and (3) contain one reused tensor, and the rest four mappings have two new tensors, resulting in the most expensive cost.

memory evictions for each GPU. In Example (c), assume each GPU memory can hold two more output tensors. Example (c) compares two cases to show load balance can also help oversubscription: case ❶ has better reusability with three reused tensors, and case ❷ has only two reused tensors but better workload balance. In case ❶, a memory eviction occurs when tensor C results a new output tensor. Case ❷ achieves no evictions and better performance than case ❶.

Remarks: A proper trade-off between data reuse and workload balance results in the optimal task allocation and helps avoiding memory evictions in GPU oversubscription situations that frequently happen in large-scale scientific computations with memory-intensive kernels like many-body correlation.

B. Factors Impacting the Data Reuse-Load Balance Trade-off

The execution of many-body correlation function calculation consists of three main parts: kernel computation, memory allocation, and data communication (i.e., data movement between CPU and GPU or between two GPUs). The latter two are referred to as memory operations in this paper. Data reuse mainly reduces memory operation cost, while workload balance is critical to kernel computation performance. Our study discovers that multiple factors influence this data reuse-load balance trade-off, and our multi-GPU scheduler design can benefit from a careful study of them.

1) The Impact of Local Reuse Pattern on the Trade-off: Theoretically, if the scheduler can capture all data reuses and conduct an exhaustive search by targeting the best data reuse-load balance combination, it can find the optimal task scheduling scheme. However, two major issues exist: First, it assumes the global knowledge of all contraction graphs that may not be available for many cases, particularly when (partial) contraction graphs are generated dynamically. Second, the search space is too large and this exhaustive search is easy to be proved an NP problem as other task scheduling problems. To address this issue, this work proposes to leverage *local reuse pattern* information to dynamically search the local optimal scheduling scheme based on a key study as follows.

Each tensor contraction involves two tensors. The tensor pair of each (incoming) tensor contraction can be categorized into one of four local reuse patterns (Fig.4 shows an example):

- $twoRepeatedSame$: Both tensors in this pair already exist in the current memory of the same GPUs. $A1$ and $A2$ already exist in the memory of GPU 0 when the new tensor pair (with $A1$ and $A2$) comes.
- $twoRepeatedDiff$: Two tensors exist in the current memory of different GPUs. $B1$ and $B2$ already exist on GPU 0 and 1, respectively.
- $oneRepeated$: One tensor of this tensor pair exists in current GPU memory. $C1$ presents in GPU 0.
- $twoNew$: Neither of them exist in current GPUs' memory. $D1$ and $D2$ are two new tensors.

Based on this *local reuse pattern* classification, Fig.4 also demonstrates and analyzes the cost of seven typical task assignments/mappings[2]: Mapping (1) assigns both tensors to the GPU that stores $A1$ and $A2$, previously. Mappings (2)

[2]The costs of other mappings that are not shown in this figure have been covered by these cases.

TABLE II: **Description of Reuse Bounds.** Reuse bounds manage different tensor pairs and mappings, representing the allowed level of load imbalance.

Name	Tensor Pairs	Mappings
Reuse_bound_1	$TwoRepeatedSame$	(1)
Reuse_bound_2	$TwoRepeatedDiff,OneRepeated$	(2) (3)
Reuse_bound_3	$TwoNew$	(4)-(7)

and (3) assign only one reused tensor in the tensor pair to the GPU with this tensor before, producing one memory allocation and one memory communication. Mappings (4) - (7) incur the most expensive cost: two memory allocations and two memory communications.

Remarks: Although it is challenging to find the global optimal scheduling scheme, it is possible to create a local optimal one with our insights on the *local reuse patterns* and *mapping study* aforementioned, particularly by designing a heuristic approach (introduced in Section IV). This approach has demonstrated its high efficacy in our evaluation.

2) The Impact of Reuse Bounds on the Trade-off: Another key factor that impacts the data reuse-load balance trade-off is *the level of allowed load imbalance*, i.e., the scheduler allows a certain level of load imbalance to leverage the potential data reuse. This work defines this factor as a special term called **reuse bound**. For example, assume assigning eight tensors to two GPUs. If the reuse bound is zero, each GPU must receive four tensors (i.e., with a perfect load balance). If the reuse bound is two, each GPU can receive up to six tensors, i.e., each GPU allows to exceed the average allocation by two.

Considering the tensor pairs with different local reuse patterns and mappings impact the schedule differently (e.g., a tensor pair with $twoRepeatedSame$ and mapping type (1) brings more data reuse benefits while others bring less), this work specifically maintains three *reuse bounds* according to the local reuse patterns and mappings of incoming tensor pairs. Table II explains these reuse bounds in detail.

Besides local reuse patterns (and mappings), multiple data characteristics also influence the setting of reuse bounds. Table I characterizes them in detail, particularly specifying their performance impact on either computation and/or memory operations. Because of these factors, it is challenging to set a uniform set of reuse bound values. An auto-tuning or machine learning approach is desired for finding reuse bound values.

To further support the above claim, this work leverages Spearman's rank correlation coefficient [22], a widely used approach to explore the relationships among data characteristics, three reuse bounds, and performance. The Spearman correlation unveils the correlation (whether linear or not) between two variables. All seven factors have positive impacts on the GFLOPS, as shown in Fig.5. *Data Distribution* and *Repeated Rate* benefit data reuse to improve the GFLOPS. Larger *Vector Size* and *Tensor Size* bring more kernel computations, resulting in higher GFLOPS. Reuse bounds represent better data reuse and workload unbalance. The positive coefficients of reuse bounds illustrate that data reuse is slightly more important than workload balance. *Data Distribution* and *Repeated Rate* have

Fig. 5: **Heatmap of the Spearman correlation coefficients.** The correlation coefficients are among data characteristics (*Data Distribution*, *Vector Size*, *Repeat Rate*, and *Tensor Size*), three reuse bounds, and GFLOPS.

positive coefficients with reuse bounds, due to the benefits of data reuse. *Vector Size* and *Tensor Size* are sensitive with workload imbalance, having negative coefficients with reuse bounds.

Remarks: Reuse bounds are critical to trade-off data reuse benefits and load imbalance costs; however, many factors influence the setting of reuse bounds. This fact guides us to design an approach to find the optimal reuse bounds efficiently (e.g., our regression model in Section IV-C).

IV. Multi-GPU Scheduling Framework

This section introduces the design and optimization of our multi-GPU scheduling framework, MICCO. MICCO's design focuses on these aspects: 1) exploring data reuse opportunities for repeated tensors, 2) improving load balance to keep GPUs busy, and 3) achieving optimal data reuse-load balance trade-off with considerations of memory evictions.

A. System Overview

Fig. 6 shows an overview of MICCO that mainly consists of two components: a heuristic scheduling algorithm, and a pre-trained lightweight regression model. Fig. 6 also illustrates the workflow that MICCO calculates many-body correlation functions. In the first step, MICCO fetches input vectors from the upstream module of a scientific application (e.g., Lattice QCD) and feeds each vector to the pre-trained regression model (❶). In the second step, the pre-trained regression model prepares for its online inference input (e.g., the data characteristics of tensor pairs in a given vector), conducts an online inference, and outputs a set of reuse bounds for this vector (❷). Because this regression model is small, this step is lightweight incurring negligible overhead. In the third step, the heuristic scheduling algorithm takes each tensor pair (❸) in a given vector and the associated reuse bounds to assign the related tensor contraction (and its tensor pair) to a specific GPU. Particularly, the heuristic scheduling algorithm toggles among three policies to assign tensors: *data-centric policy*, *computation-centric policy*, and *memory-eviction-sensitive policy*.

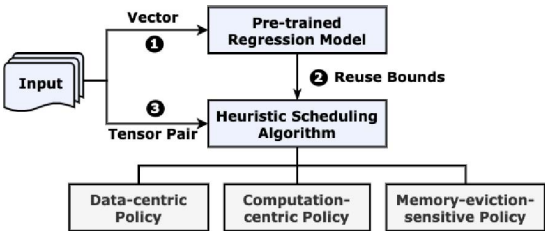

Fig. 6: **System overview of MICCO.** Input data is tensors in vectors. MICCO dynamically handles vectors and generates GPU assignments for each vector. MICCO consists of a regression model and a heuristic scheduling algorithm. MICCO extracts data characteristics of each vector to the regression model (❶). The regression model generates optimal reuse bounds (❷). The heuristic algorithm classifies tensor pairs (❸) and jointly manages three policies.

TABLE III: **Definitions of Variables.**

Variable Name	Descriptions
$Tensor1$,$Tensor2$	Input tensor of one tensor pair
$reuseBd$	A vector of three reuse bounds
$tensorsGPU$	A pair between tensors and GPU
$mapGPUTensor$	GPU-Tensor pairs mappings
$mapGPUCom$	GPU-Computation cost mappings
$mapGPUMem$	GPU-Memory cost mappings
$numGPU$	The number of GPUs
$numTensor$	The number of tensors
$balanceNum$	$\lvert numTensor$ / $numGPU\rvert$
$candiQueue$	A queue of candidate GPUs
$GPUMaxMemory$	The maximal memory size of GPU

B. Heuristic Scheduling Algorithm

To trade-off data reuse and load balance, this heuristic algorithm toggles among three scheduling policies. ***Data-centric policy*** emphasizes data reuse, and assigns tensors based on the aforementioned tensor pair local reuse pattern classification and mapping strategy. ***Computation-centric policy*** emphasizes workload balance, and ensures each GPU handles the identical number of tensor pairs. ***Memory-eviction-sensitive policy*** emphasizes reducing memory cost to avoid evictions.

As claimed in Sec. III, data reuse is the principal factor to alleviate expensive memory operations, thus benefiting the data-intensive nature of many-body correlation functions. Therefore, the data-centric policy first dominates MICCO's scheduling to find *available* GPUs that hold the incoming tensors already, and then MICCO stores these GPUs' IDs in a queue ($candiQueue$). To decide if a GPU is available, MICCO compares a GPU's computing utilization with the reuse bounds from the regression model, i.e., if assigning the incoming pair to a given GPU results in severe load imbalance, this GPU is *unavailable*. Next, the computation-centric policy dominates MICCO's scheduling to select the GPU with least computation from $candiQueue$ to further balance workload. If the former scheduling causes any oversubscription of a GPU in $candiQueue$, the memory-eviction-sensitive policy kicks in to select the GPU with the most available memory in $candiQueue$ to avoid data evictions.

Alg.1 illustrates the designed heuristic scheduling algorithm

Algorithm 1: Heuristic Scheduling Algorithm

Input: Tensor $Tensor1$, $Tensor2$, Vector $reuseBd$, Map $mapGPUTensor$, $mapGPUCom$, $mapGPUMem$, Integer $balanceNum$
Output: A pair $tensorsGPU$

```
1  Initialize candiQueue;
2  GPUsofTensor1 = mapGPUTensor.find(Tensor1);
3  GPUsofTensor2 = mapGPUTensor.find(Tensor2);
4  if (GPUsofTensor1 ∩ GPUsofTensor2) ≠ NULL then
5      for it1 : GPUsofTensor1 do
6          if it1 ∈ GPUsofTensor2 ∩
             mapGPUTensor.at(it1).size() <
             reuseBd[0] + balanceNum then
7              Add it1 to candiQueue;
8  if candiQueue = NULL ∩ (GPUsofTensor1 ≠
     NULL ∪ GPUsofTensor2 ≠ NULL) then
9      for it1 : GPUsofTensor1 do
10         if mapGPUTensor.at(it1).size() <
             reuseBd[1] + balanceNum then
11             Add it1 to candiQueue;
12     for it2 : GPUsofTensor2 do
13         if mapGPUTensor.at(it2).size() <
             reuseBd[1] + balanceNum then
14             Add it2 to candiQueue;
15 if candiQueue = NULL then
16     for it = 1; it ≤ numGPU; it++ do
17         if mapGPUTensor.at(it).size() <
             reuseBd[2] + balanceNum then
18             Add it to candiQueue;
19 Call Alg.2 to determine tensorsGPU;
20 Update mapGPUTensor, mapGPUCom, mapGPUMem;
21 return tensorsGPU;
```

that processes tensor pairs one after another. Alg.2 shows the generation of an assignment between a tensor pair and a GPU by designed scheduling policies. Tab. III explains the variables in both algorithms. The heuristic algorithm is *greedy* with $O(n^2)$ time complexity, where n is the number of tensor pairs. The outer n loop is to traverse all tensor pairs, while the inner n loop is to check previous tensor pairs in $mapGPUTensor$. The main steps of this algorithm are clarified as follows:

- *Step-I*: Alg.1 figures out the local reuse pattern of an incoming tensor pair by checking $mapGPUTensor$ (line 2-3). If the pair belongs to $twoRepeatedSame$ (line 4), Alg.1 finds all available GPUs that hold this pair already and puts their IDs in $candiQueue$ (line 5-7).
- *Step-II*: If the $twoRepeatedSame$ pair cannot find any available GPUs or the pair belongs to $twoRepeatedDiff$ or $oneRepeated$ (line 8), Alg.1 fills in $candiQueue$ with the available GPUs containing one tensor of the pair (line 9-14). Otherwise, Alg.1 fills in $candiQueue$ with all available GPUs (line 15-18).
- *Step-III*: Alg.2 monitors memory cost and recognizes memory oversubscription (line 3-5). If no memory eviction occurs (line 6), Alg.2 selects the GPU with the least computation from the $candiQueue$ (line 7-11). If memory eviction appears, Alg.2 selects the GPU with the most memory capacity from the $candiQueue$ (line 12-17).

- *Step-IV*: Alg.2 creates a tensor assignment ($tensorGPU$) by combining the selected GPU ID with this tensor pair (line 18), and passes it to Alg.1 (line 19). Alg.1 dynamically updates $mapGPUTensor$ after handling each tensor pair (line 20).

Algorithm 2: Tensor Assignment Algorithm

```
   Input: Map mapGPUCom, mapGPUMem,
          mapGPUTensor, Vector candiQueue, Tensor Tensor1,
          Tensor2, Integer GPUMaxMemory
   Output: A pair tensorsGPU
1  Initialize vector GPUSelect, Integer GPUID, Bool evictFlag;
2  Integer candidateNum = candiQueue.size();
3  for id = 1; id ≤ candidateNum; id++ do
4      if mapGPUMem.at(id).size() > GPUMaxMemory then
5          evictFlag = TRUE;
6  if evictFlag ≠ TRUE then
7      GPUSelect.add(min (mapGPUCom.at(id).size(),
         id ∈ candiQueue));
8      if GPUSelect.size()>1 then
9          GPUID =random (min (mapGPUMem.at(id).size(),
             id ∈ candiQueue));
10     else
11         GPUID = GPUSelect.at(0);
12 else
13     GPUSelect.add (min (mapGPUMem.at (id).size(),
         id ∈ candiQueue));
14     if GPUSelect.size()>1 then
15         GPUID =random(min (mapGPUCom.at(id).size(),
             id ∈ candiQueue));
16     else
17         GPUID = GPUSelect.at(0);
18 tensorsGPU = make_pair (Tensor1, Tensor2, GPUID);
19 return tensorsGPU;
```

C. Regression Model

To determine the optimal setting of the three reuse bounds, MICCO builds a regression model to explore the correlation between data characteristics and reuse bounds. Input (feature variables) is data characteristics and output (response labels) is optimal reuse bound setting. Data characteristics include vector size, tensor size, data distribution, and repeated rate. Vector size and tensor size are given variables by input data. Data distribution is judged to be uniform or biased. Repeated rate is calculated dynamically for each vector. The regression model is offline trained once in the beginning. In offline training, the total data size is 300, 20% of which is test data to evaluate the prediction. For each set of feature variables, we measure GFLOPS of all possible values of reuse bounds and set the optimal reuse bounds to be the response labels. Reuse bounds range from 0 to $numTensor - balanceNum$ (i.e., assigning all data to one GPU). During online scheduling, MICCO extracts data characteristics of each vector and executes the inference of the pre-trained regression model to generate optimal reuse bound values.

Tab. IV compares the precision of three regression models [21], including Linear Regression, Gradient Boosting, and Random Forest. R^2 Score [17] is a well-known statistical metric to measure the regression predicting quality. R^2 Score is closer to 1, the regression model is more accurate. The results in Tab. IV illustrate that the correlation among data characteristics, reuse bounds, and GFLOPS is non-linear. It is also difficult to predict the optimal reuse bound setting by a policy-based approach. Thus, building a non-linear regression model is necessary. MICCO selects Random Forest [23] as its regression model because of its high accuracy (95%). Here are more details about this model: the learning rate of Gradient Boosting and Random Forest is 0.1, the number of boosting stages in Gradient Boosting is 150, and the number of trees in Random Forest is 150.

TABLE IV: R^2 **Score of Regression Models**

$LinearRegression$	$GradientBoosting$	$RandomForest$
0.57	0.91	0.95

V. EVALUATION

This section aims to evaluate MICCO, particularly with the following objectives: (1) proving MICCO outperforms state-of-art schedulers with varied vector sizes and data repeated rates for both uniform and non-uniform data distributions; (2) exploring the impact of reuse bounds and demonstrating MICCO obtains stable improvements with varied numbers of GPUs (scalability), tensor size, and memory oversubscription rate; (3) showing that MICCO can be integrated into a real-world system, Redstar [6], [7], [8], and yields obvious benefits on real problem sizes and datasets.

A. Experiment Setup

Platforms. MICCO is evaluated on eight AMD MI100 GPUs, each with 32G GPU memory[3]. The compiler is Rocm-4.3.0 based on clang 13.0.0. These GPUs are connected to an AMD EPYC 7502 32-Core CPU Processor.

Baseline and optimized versions. This evaluation compares MICCO with a state-of-art work, `Groute` [4], a popular and efficient multi-GPU scheduling framework. `Groute` assigns jobs and associated data on the earliest available device to achieve good load balance similar to many other frameworks [16], [9] Two versions of MICCO are evaluated including `MICCO-naive` and `MICCO-optimal`. `MICCO-naive` does not benefit from reuse bounds (by setting these values as zero) while `MICCO-optimal` leverages reuse bounds produced from the regression model.

Evaluation setups. Our experiments extensively evaluate MICCO by changing multiple data characteristics including vector size, repeated rate, tensor size, and memory oversubscription rate. MICCO is also evaluated with varied numbers of GPUs for scalability. To evaluate the impact of data distribution, our experiments synthesize both unbiased and biased datasets. The selection of repeated data from the previous data follows two distributions: Uniform and Gaussian.

[3]Although MICCO is evaluated on the latest AMD GPUs, it can also run on other GPUs like other generations of AMD GPUs and NVIDIA GPUs because its design is general, and not bound with specific GPU hardware implementations.

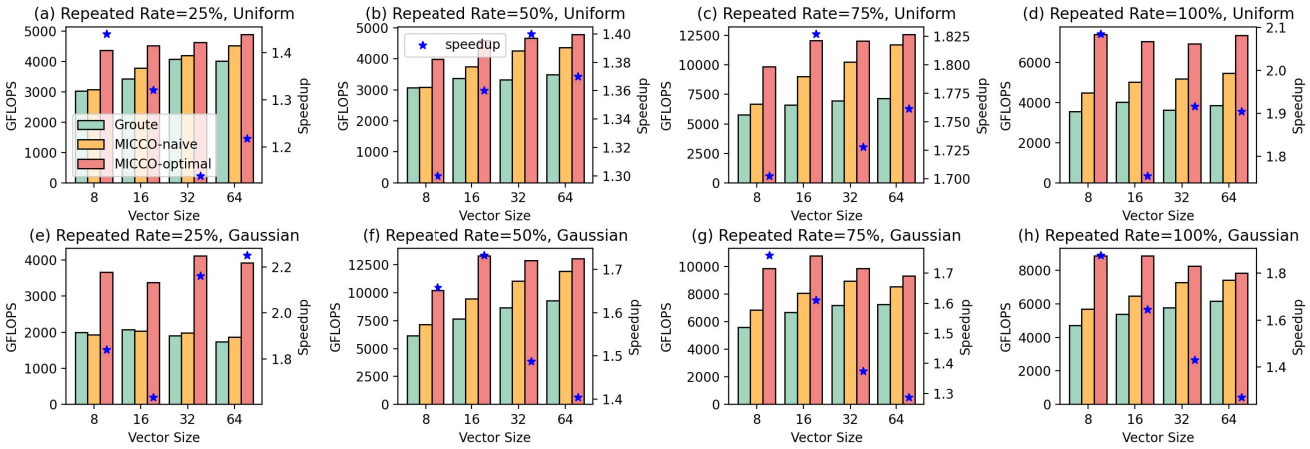

Fig. 7: **Overall Performance.** Two distributions: Uniform (a)-(d) and Gaussian (e)-(h). **Blue stars denote speedup of MICCO-optimal / `Groute`.** Repeated rate varies from 25% to 100%. Vector size varies from 8 to 64. Tensor size is 384. The utilized GPU number is eight.

TABLE V: **Execution Time (ms).** Tensor size is 384. Vector size is 64. Repeated rate is 50%. Sum of 10 vectors.

Distribution	Scheduling Overhead	Total Time
Uniform	8.27	4925.73
Gaussian	8.52	1550.88

Real-world system and datasets. To further validate the practical performance, MICCO[4] is also integrated into Redstar and evaluated on three real-world correlation functions.

B. Overall Performance Evaluation

Fig.7 illustrates the overall performance improvements by comparing MICCO with `Groute` in two distributions: Uniform and Gaussian. We measure four vector sizes from 8 to 64, and the tensor size is 384. The speedup of MICCO over `Groute` is also shown in Fig.11, labeled as blue stars.

Experiment results demonstrate that MICCO outperforms `Groute` in all cases, achieving up to 2.25× speedup. Fig.7 (a)-(d) show throughput in Uniform distribution. The optimal version of MICCO (`MICCO-optimal`) is able to achieve 1.57× geometric mean speedup than `Groute`. Fig.7 (e)-(h) show throughput in Gaussian distribution, and the geometric mean speedup is 1.65× than `Groute`. Compared with `MICCO-naive`, `MICCO-optimal` achieves up to 1.89× speedup. These results show the great benefits of MICCO's heuristic scheduling algorithm and regression model.

One interesting observation is growing repeated rate cannot keep improving performance, further validating the trade-off between data reuse and load balance. The best performance appears with 75% repeated rate in Uniform, and 50% repeated rate in Gaussian. Please note that repeated rate describes initial characteristics of input data rather than real reused data in calculations. Considering load balance, some repeated data has to be assigned to new GPUs for the optimal performance according to our heuristic scheduling, so improving repeated rate does not necessarily mean more data reuse.

[4]https://github.com/JeffersonLab/hadron.

When comparing data distributions, the following two observations also support that data reuse and load balance jointly impact the performance. One is that to reach the best throughput, the repeated rate of Uniform is higher than Gaussian. This is because biased distribution (like Gaussian that determines the selection of repeated data) leads to more load imbalance than Uniform distribution, and this load imbalance particularly increases with the growth of biased degree. Another observation is that a larger vector size may degrade the performance of Gaussian (as shown in Fig.7 (g) and (h)). This is because the increasing vector size and large repeated rate (more than 50%) produce many biased repeated data and cause increasingly severe load imbalance.

Tab.V demonstrates the extremely **low scheduling overhead** of MICCO (`MICCO-optimal`), particularly compared with the total executing time (5.4‰ and 1.6‰).

C. Performance analysis

This section further studies MICCO's performance from multiple aspects. Please notice that `MICCO` in this section denotes `MICCO-optimal`.

Exploring the impacts of reuse bounds. Fig.8 shows the impact of changing reuse bounds on the performance. The experiments include three cases: Case (1) vector size = 64, repeated rate = 50%; Case (2) vector size = 16, repeated rate = 25%; Case (3) vector size = 32, repeated rate = 75%. This work measures thirteen values of three reuse bounds. The reuse bound values of the best performance vary when changing vector size, repeated rate, and data distribution. In Case (1) of Fig.8 (a), the best performance is 9753 GFLOPS with (0,2,0), but in Case (3) of Fig.8 (b), the best performance is 5869 GFLOPS with (0,2,2). The evaluation results explicitly show that multiple data characteristics influence data reuse-load balance trade-off and the optimal values of reuse bounds, which are hard to predict by a policy-based approach or linear regression. This observation is consistent with Tab. IV,

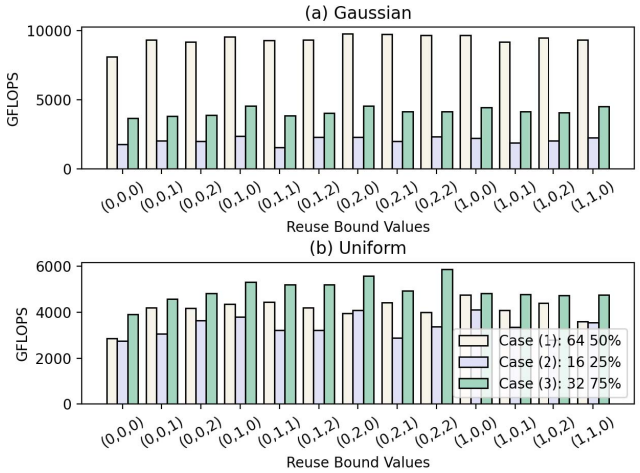

Fig. 8: **Impact of Reuse Bounds.** Case (1) vector size = 64, repeated rate = 50%; Case (2) vector size = 16, repeated rate = 25%; Case (3) vector size = 32, repeated rate = 75%; Tensor size is 384. 13 sets of three reuse bounds are measured, and the ranging from 0 to 2.

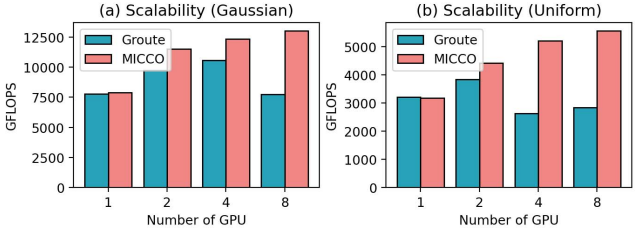

Fig. 9: **Scalability.** Tensor size is 384. Vector size is 64.

and further supports the necessity of building a non-linear regression model to generate optimal reuse bounds.

Exploring scalability. This work compares MICCO with `Groute` and changes the number of GPUs from 1 to 8 in Uniform and Gaussian distributions. As shown in Fig.9, MICCO outperforms `Groute`, achieving up to 1.96× speedup. One observation is the slow growth of GFLOPS with an increasing number of GPUs. e.g., GFLOPS increases from 7877 GFLOPS on 1 GPU, to 13043 GFLOPS on 8 GPUs in Fig.9 (a). One reason is when computing small tensors (tensor size is 384), memory operation impacts more than computation on GFLOPS. Another reason is more GPUs bring more computation capacity but make data reuse harder. One GPU can reuse all repeated tensors, while multiple GPUs cannot achieve full data reuse concerning load balance. The speedup improves from 1.18× on 2 GPUs to 1.68× on 8 GPUs, showing MICCO yields great benefits on leveraging data reuse and reducing memory operations.

Exploring the impact of tensor size. This work compares `Groute` and MICCO with varying tensor sizes including 128, 256, 384, 768 in two distributions. As shown in Fig.10, MICCO outperforms `Groute`, achieving speedup from 1.35× to 1.92×. The performance is sensitive to the tensor size, which determines the kernel computation cost. Overall, MICCO obtains better performance than `Groute` in

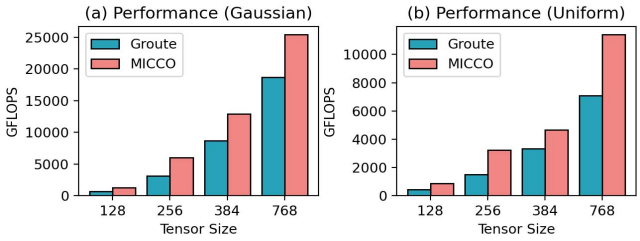

Fig. 10: **Impact of Tensor Size.** Tensor size varies from 128 to 768. Vector size is 64. Repeated rate is 50%.

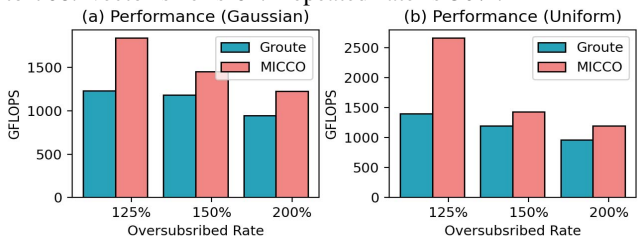

Fig. 11: **Memory Oversubscription.** Oversubscription rate increases from 125% to 200%. Vector size is 64. Tensor size is 384. Repeated rate is 50%.

all cases as tensor size varies.

Exploring memory oversubscription. The experiments measure two data distributions when vector size is 64, tensor size is 384, and the repeated rate is 50%. MICCO achieves a speedup up to 1.9× over `Groute`. The GFLOPS decreases with the increasing memory oversubscription. For instance, GFLOPS decreases from 1841 to 1224 in Gaussian and 2663 to 1194 in Uniform as the subscription rate increases from 125% to 200%. This observation shows that the performance is sensitive to memory evictions. The geometric mean speedup of MICCO over `Groute` is 1.4× in Gaussian and 1.2× in Uniform. Memory evictions have slightly more impacts on Uniform than Gaussian distribution.

D. Case Study: Real-world Datasets in Redstar System

TABLE VI: **Real Many-body Correlation Functions.** Total memory represents the total device memory about input and intermediate output data. 'Speedup' is based on the `Groute`.

Function	Tensor Size	Memory Cost	Speedup
`a1_rhopi`	128	56.05G	**1.49x**
`f0d2`	256	4645.12G	**1.41x**
`f0d4`	256	4064.48G	**1.36x**

In order to evaluate practical scenarios, this work measures three real physics correlation functions in the Redstar system. The correlation functions are `a1_rhopi` in a_1 system, and `f0d2` and `f0d4` in f_0 system. All of them belong to meson systems and consist of two-particle and single-particle constructions. Their tensor size and total device memory cost are shown in Tab. VI. The memory cost is the sum of sixteen time slices, including initial input data and intermediate output data. The utilized number of GPUs is eight. Vector size, repeated rate, and data distribution vary dynamically. Compared with `Groute`, the speedup achieves up to 1.49×. The experiment results demonstrate the practical significance of MICCO.

VI. RELATED WORKS

Recently, many multi-GPU scheduling frameworks have been developed to support different types of applications [10], [14], [4], [12], [15], [19], [35], [34], [31], [30], [32]. The efforts closely related to this work include some general data-aware multi-GPU schedulers, graph processing schedulers, schedulers that support other irregular computations, streaming task schedulers aiming to process a sequence of computation kernels, and schedulers for machine learning models [33].

General data-aware Multi-GPU schedulers Although the current general data-aware GPU schedulers (as [3], [11], [27]) that consider data locality have shown efficacy on many applications based on large matrices (e.g., Matrix Multiplication, LU, etc.), it is not easy to directly apply them to our application due to multiple unique features of many-body correlation calculations. Augonnet *et al.* [3] mainly focus on reducing transfer latency and overlapping with kernel computation, while reducing data movement counts brings more significant performance gains for our application because of the large number of kernels (with reusable data). Gonthier *et al.* [11] assume the knowledge of all tasks and dependencies, but our application requires online scheduling for dynamic graphs. This work also mainly focuses on single-GPU scheduling. Teodoro *et al.* [27] mainly focus on optimizing CPU-GPU transfers instead of multi-GPUs.

Multi-GPU schedulers in graph applications and other irregular applications. Many multi-GPU graph processing schedulers [4], [9], [16] adopt greedy-based strategies that assign jobs to the earliest available devices and mainly consider workload balance. Ben *et al.* [4] present an asynchronous and runtime multi-GPU programming model for graph and irregular applications. Chen *et al.* [9] propose a task-based dynamic load-balancing solution for both single- and multi-GPU systems. These efforts mainly focus on optimizing workload balance to improve the GPU utilization without considering data reusability and the interplay between load balance and data reusability. More close to our work, Kim *et al.* [18] develop CODA that enables co-placement of compute and data for fine-grained interleaved memory with a low-cost method. Although this work takes data placement into account, it pays more attention to data locations rather than reusing data.

Multi-GPU schedulers for streaming tasks and machine learning models. Huynh *et al.* [15] propose a code generation framework mapping streaming applications onto a multi-GPU system. Melot *et al.* [20] present a crown scheduling to improve the efficiency of energy utilization. Udupa *et al.* [28] propose an efficient technique to execute stream programs on GPUs. These efforts do not consider data reuse-load balance trade-off in terms of memory oversubscription as MICCO. Many multi-GPU schedulers for machine learning workloads have been proposed recently [35], [19], [12], [13]. Gandiva [35] is a domain-specific scheduler, accelerating deep learning models by packing jobs on multiple GPUs. CROSSBOW [19] proposes a multi-GPU scheduler for deep learning with small batch sizes. These efforts have different focuses and do not study the trade-off of load balance and data reuse as MICCO, either. Additionally, many works, e.g., Tiresias [12] and Marble [13], apply preemptive scheduling approaches, which are not suitable for this work, due to the heavy overhead of suspending and resuming in many kernel computations.

VII. CONCLUSION AND FUTURE WORK

This work presents MICCO, a multi-GPU scheduling framework to accelerate calculating many-body correlation functions, integrated in a real-world Lattice QCD system, Redstar system. This work extensively studies the data reuse-load balance interplay, and further brings up *local reuse pattern* and *reuse bounds*. MICCO proposes a heuristic scheduling algorithm toggling data reuse and load balance regarding memory oversubscription. Moreover, MICCO builds a regression model to predict optimal reuse bounds. In evaluation, MICCO achieves up to $2.25\times$ speedup in synthesized datasets and $1.49\times$ speedup in real correlation functions. In the future, we plan to extend the design of MICCO to a multi-node cluster with GPUs and implement it on an NVIDIA GPU cluster upon the availability of these devices. Additionally, we are exploring further optimizations on both intra-node and inter-node communications, including asynchronous data copy and prefetching data.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for making innumerable helpful suggestions and comments. This work is partially supported by the NSF award CCF-2047516 (CAREER), and the US Department Of Energy, Office of Science, Offices of Nuclear Physics and Advanced Scientific Computing Research, through the SciDAC program under contract DE-AC05-06OR23177 under which JSA LLC operates and manages Jefferson Lab, and under the 17-SC-20-SC Exascale Computing Project.

REFERENCES

[1] Ahmad Al Badawi, Bharadwaj Veeravalli, Jie Lin, Nan Xiao, Matsumura Kazuaki, and Aung Khin Mi Mi. Multi-gpu design and performance evaluation of homomorphic encryption on gpu clusters. *TPDS*, 2020.

[2] Y Aoki, T Blum, N Christ, C Dawson, K Hashimoto, T Izubuchi, JW Laiho, L Levkova, M Lin, R Mawhinney, et al. Lattice qcd with two dynamical flavors of domain wall fermions. *Physical Review D*, 2005.

[3] Cédric Augonnet, Jérôme Clet-Ortega, Samuel Thibault, and Raymond Namyst. Data-aware task scheduling on multi-accelerator based platforms. In *2010 IEEE 16th International Conference on Parallel and Distributed Systems*, pages 291–298. IEEE, 2010.

[4] Tal Ben-Nun, Michael Sutton, Sreepathi Pai, and Keshav Pingali. Groute: An asynchronous multi-gpu programming model for irregular computations. *ACM SIGPLAN Notices*, 2017.

[5] Evan Berkowitz, Thorsten Kurth, Amy Nicholson, Bálint Joó, Enrico Rinaldi, Mark Strother, Pavlos M Vranas, and André Walker-Loud. Two-nucleon higher partial-wave scattering from lattice qcd. *Physics Letters B*, 2017.

[6] Jie Chen, Robert Edwards, and Frank Winter. Graph-based contractions with optimal evaluation strategies. *ADSE03-LatticeQCD Application Strategy WBS 1.2.1.03*, (Milestone ADSE03-7), 2017.

[7] Jie Chen, Robert Edwards, and Frank Winter. Performance enhancement to the graph-based contraction calculations. *ADSE03-LatticeQCD Application Strategy WBS 1.2.1.03*, (Milestone ADSE03-7), 2018.

[8] Jie Chen, Robert Edwards, and Frank Winter. Enabling graph based contraction calculations for multi-nucleon systems. *ADSE03-LatticeQCD Application Strategy WBS 1.2.1.03*, (Milestone ADSE03-14), 2019.

[9] Long Chen, Oreste Villa, Sriram Krishnamoorthy, and Guang R Gao. Dynamic load balancing on single-and multi-gpu systems. In *IPDPS*. IEEE, 2010.

[10] Trilce Estrada, David A Flores, Michela Taufer, Patricia J Teller, Andre Kerstens, and David P Anderson. The effectiveness of threshold-based scheduling policies in boinc projects. In *e-Science'06*. IEEE, 2006.

[11] Maxime Gonthier, Loris Marchal, and Samuel Thibault. *Locality-Aware Scheduling of Independant Tasks for Runtime Systems*. PhD thesis, Inria, 2021.

[12] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A {GPU} cluster manager for distributed deep learning. In {*NSDI*}, pages 485–500, 2019.

[13] Jingoo Han, M Mustafa Rafique, Luna Xu, Ali R Butt, Seung-Hwan Lim, and Sudharshan S Vazhkudai. Marble: A multi-gpu aware job scheduler for deep learning on hpc systems. In *CCGRID*. IEEE, 2020.

[14] Stephen Herbein, Dong H Ahn, Don Lipari, Thomas RW Scogland, Marc Stearman, Mark Grondona, Jim Garlick, Becky Springmeyer, and Michela Taufer. Scalable i/o-aware job scheduling for burst buffer enabled hpc clusters. In *HPDC*, 2016.

[15] Huynh Phung Huynh, Andrei Hagiescu, Weng-Fai Wong, and Rick Siow Mong Goh. Scalable framework for mapping streaming applications onto multi-gpu systems. In *PPoPP*, 2012.

[16] Zhihao Jia, Yongkee Kwon, Galen Shipman, Pat McCormick, Mattan Erez, and Alex Aiken. A distributed multi-gpu system for fast graph processing. *Proceedings of the VLDB Endowment*, 11(3):297–310, 2017.

[17] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 2015.

[18] Hyojong Kim, Ramyad Hadidi, Lifeng Nai, Hyesoon Kim, Nuwan Jayasena, Yasuko Eckert, Onur Kayiran, and Gabriel Loh. Coda: Enabling co-location of computation and data for multiple gpu systems. *ACM TACO*, 2018.

[19] Alexandros Koliousis, Pijika Watcharapichat, Matthias Weidlich, Luo Mai, Paolo Costa, and Peter Pietzuch. Crossbow: scaling deep learning with small batch sizes on multi-gpu servers. *arXiv preprint arXiv:1901.02244*, 2019.

[20] Nicolas Melot, Christoph Kessler, Jörg Keller, and Patrick Eitschberger. Fast crown scheduling heuristics for energy-efficient mapping and scaling of moldable streaming tasks on manycore systems. *ACM TACO*, 2015.

[21] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[22] Philip Sedgwick. Spearman's rank correlation coefficient. *Bmj*, 2014.

[23] Tao Shi and Steve Horvath. Unsupervised learning with random forest predictors. *Journal of Computational and Graphical Statistics*, 2006.

[31] Qiong Wu, Adam Hare, Sirui Wang, Yuwei Tu, Zhenming Liu, Christopher G Brinton, and Yanhua Li. Bats: A spectral biclustering approach

[24] Yogesh Simmhan, Alok Kumbhare, Charith Wickramaarachchi, Soonil Nagarkar, Santosh Ravi, Cauligi Raghavendra, and Viktor Prasanna. Goffish: A sub-graph centric framework for large-scale graph analytics. In *Euro-Par*. Springer, 2014.

[25] Markus Steinberger, Michael Kenzel, Pedro Boechat, Bernhard Kerbl, Mark Dokter, and Dieter Schmalstieg. Whippletree: Task-based scheduling of dynamic workloads on the gpu. *TOG*, 2014.

[26] SN Syritsyn, JD Bratt, MF Lin, HB Meyer, JW Negele, AV Pochinsky, M Procura, M Engelhardt, Ph Hägler, TR Hemmert, et al. Nucleon electromagnetic form factors from lattice qcd using 2+ 1 flavor domain wall fermions on fine lattices and chiral perturbation theory. *Physical Review D*, 2010.

[27] George Teodoro, Tony Pan, Tahsin M Kurc, Jun Kong, Lee AD Cooper, Norbert Podhorszki, Scott Klasky, and Joel H Saltz. High-throughput analysis of large microscopy image datasets on cpu-gpu cluster platforms. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 103–114. IEEE, 2013.

[28] Abhishek Udupa, R Govindarajan, and Matthew J Thazhuthaveetil. Software pipelined execution of stream programs on gpus. In *CGO*. IEEE, 2009.

[29] Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D Owens. Gunrock: A high-performance graph processing library on the gpu. In *PPoPP*, 2016.

[30] Qiong Wu, G. Brinton Christopher, Zhang Zheng, Pizzoferrato Andrea, LIU Zhenming, and Cucuringu Mihai. Equity2vec: End-to-end deep learning framework for cross-sectional asset pricing. *International Conference on AI in Finance*, 2021.
to single document topic modeling and segmentation. *ACM Transactions on Intelligent Systems and Technology*, 2021.

[32] Qiong Wu, Wen-Ling Hsu, Tan Xu, Zhenming Liu, George Ma, Guy Jacobson, and Shuai Zhao. Speaking with actions-learning customer journey behavior. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, pages 279–286. IEEE, 2019.

[33] Qiong Wu and Zhenming Liu. Rosella: A self-driving distributed scheduler for heterogeneous clusters. *International Conference on Mobility, Sensing and Networking (MSN)*, 2021.

[34] Qiong Wu, Felix M Wong, Yanhua Li, Zhenming Liu, and Varun Kanade. Adaptive reduced rank regression. *Advances in Neural Information Processing Systems*, 33:4103–4114, 2020.

[35] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In {*OSDI*}, pages 595–610, 2018.