SCS-led CS550 Term Projects

* Students may pick one of the following SCS-led projects and build their term project proposals accordingly.
* Students should contact the designated PhD lead before and after preparing their term project proposals.
* Ideally, the proposals should be submitted to Blackboard after both the student(s) and the PhD lead agree that the proposal is ready.
* The proposals may still get rejected and need redo in the acceptance process.
* Feel free to submit your questions on Piazza under the ‘project’ folder.

## Building a distributed SQL engine

**Description**: At the SCS lab, we have built a small prototype for an SQL engine. This engine instead of operating on top of databases works around log-stores and has been enhanced to perform time-based queries. For example, what was the traffic on this road the last 8 Mondays of this year? Or what was the value of this stock a month ago? This early prototype has been developed in C++ and supports parsing and executing SQL language. Further, it supports the indexing of the log. Yet, all of these features are only designed for a single log residing on a single node.

Students applying for this project will receive documentation on the current project and access to the code. After familiarizing themselves with the codebase, they will be asked to further enhance the engine to be able to either perform multi-log queries, like joins, or multi-node queries, where a single log resides on two nodes.

**Prerequisite:** C++, Scripting, SQL/Databases (ideally)

**Recommended team size:** 1-2

**PhD lead:** Jaime Cernuda ([jcernudagarcia@hawk.iit.edu](mailto:jcernudagarcia@hawk.iit.edu))

## Survey on distributed clock algorithms

**Description**: When embarking in the field of distributed systems, a number of problems that were easy to solve suddenly become immensely complex. One such problem is the synchronization of distributed clocks.

In traditional computing, clocks are typically implemented as a simple count of the number of ticks that have transpired since some arbitrary starting date. Yet, these “ticks” are anything but uniform, as heat, CPU voltage and a number of factors can significantly modify the time between ticks, and thus the accuracy of the clock. This is not of significant concern on single node problems, where an external reference can be used every so often to return to the correct time.

But in distributed systems, this is much harder, as there is a need to not only correct every node, but also ensure that all nodes in the cluster have the same time for synchronization and coherent operations.

To solve this problem, a number of algorithms have been proposed for the last 50 years. Some deal with logical vs global clocks. Others are centralized vs decentralized, etc. The student that takes this project will embark in an exploration of this field through the reading and summarization of papers in the field. Starting papers will be provided.

**Prerequisite:** N/A

**Recommended team size:** 1

**PhD lead:** Jaime Cernuda ([jcernudagarcia@hawk.iit.edu](mailto:jcernudagarcia@hawk.iit.edu))

## Implementing an ML data streaming pipeline

**Description**: AI has been a revolution to the anyu field dealing with data. A traditional AI workload is split between the training phase, where data is given to a model to define its behavior; and an inference phase, where new data is given to the model.

Streaming ML is the application of an ML model to a streaming data pipeline, which is a workflow that ingests and transforms data in real-time increments between a source and a target. The machine learning model provides the logic that helps the streaming data pipeline generate real-time responses to the data.

This field of big data has been in development for a few years, but only recently are systems reaching maturity. The aim of this project is to explore some of the solutions available in the market and implement a small, distributed streaming ML pipeline.

**Prerequisite:** Python or Java, Data Streaming (ideally)

**Recommended team size:** 1-2

**PhD lead:** Jaime Cernuda ([jcernudagarcia@hawk.iit.edu](mailto:jcernudagarcia@hawk.iit.edu))

## Benchmarking Data Messaging Frameworks.

**Description**: Messaging systems are a core feature of a significant number of distributed architectures, messaging queues or data brokers have become indispensable for the operation of a great number of modern-day operations. These systems stand in the middle of applications and serve as a relay of information between other systems. Naturally, many tools have come up to provide users with an understanding of the different systems' capabilities. The OpenMessaging Benchmark Framework (OMBF) stands at the forefront of these tools. Publicly available and developed in Java, the OMBF provides a framework for deploying and testing these systems under different workloads. Yet, the core functionality of the benchmarks makes it uniquely suited to cloud providers, with deployment scripts and testing models uniquely developed to integrate with them. In this project, we want students to develop directly into these systems, learn to install them, deploy them, and run tests with them. Once done, we aim to modify the OMBF to detach it from its inherent cloud nature and allow the benchmark to function on more traditional cluster systems such as High-Performance Computing resources. Finally, the students will execute their benchmarks in a small cluster and provide results and analysis comparing a few of the data messaging systems.

**Prerequisite:** Java, Linux, Scripting, Data Streaming/Message Queues (ideally)

**Recommended team size:** 1-2

**PhD lead:** Jaime Cernuda ([jcernudagarcia@hawk.iit.edu](mailto:jcernudagarcia@hawk.iit.edu))

## Build an algorithm to create an I/O identity for an application

**Description**: This project aims to improve our understanding of I/O patterns in distributed environments by classifying I/O applications. By analyzing I/O traces from various cluster sites, students will gain insight into the challenges of multiprocess contention for I/O resources and the potential solutions for addressing them. The goal of this project is to develop a framework for classifying I/O applications, which will enable transfer of knowledge across applications and improve the performance of distributed systems.

**Prerequisite:** Python, Data Analysis and Classification Algorithms, Linux, Scripting

**Recommended team size:** 1-2

**PhD lead:** Neeraj Rajesh ([nrajesh@hawk.iit.edu](mailto:nrajesh@hawk.iit.edu))

## Survey of fast convergent neural networks

**Description**: This project aims to conduct an in-depth survey of the state-of-the-art techniques for fast convergence of neural networks. The research will involve a comprehensive literature review of the current methods and techniques used to improve the speed of convergence in neural networks. The students will also implement and test some of these techniques on benchmark datasets to evaluate their performance. The project will culminate in a comparison of the various techniques in terms of their convergence speed and accuracy. This project provides an opportunity for students to gain a deeper understanding of the challenges and potential solutions in the field of deep learning optimization.

**Prerequisite:** N/A

**Recommended team size:** 1-2

**PhD lead:** Neeraj Rajesh ([nrajesh@hawk.iit.edu](mailto:nrajesh@hawk.iit.edu))

## Deploy a distributed Queue using Erlang

**Description**: This project aims to demonstrate the advantages of using a language specifically designed for distributed systems, such as Erlang, by reimplementing an in-house distributed queue. Erlang is well suited for distributed environments because of its lightweight processes, message-passing concurrency model, and built-in fault tolerance. These features make it easy to design and implement highly concurrent and fault-tolerant systems. In contrast, languages such as C++, C, and Java, which are not specifically designed for distributed systems, often require more complex and error-prone implementations. The students will be tasked with writing the necessary code in Erlang to deploy a functional and efficient distributed queue system. They will also measure and compare the performance of this implementation with the existing one, using various metrics such as scalability, throughput and latency. This project is suitable for students with an interest in distributed systems, Erlang programming and performance optimization.

**Prerequisite:** C++, Erlang (or other functional programming languages), Linux, Scripting, Data Streaming/Message Queues (ideally)

**Recommended team size:** 1-2

**PhD lead:** Neeraj Rajesh ([nrajesh@hawk.iit.edu](mailto:nrajesh@hawk.iit.edu))

## Worker Scheduling Algorithms

**Description**:I/O is commonly performed in a synchronous paradigm where the user calls a read or write function and then waits for the read or write to complete. However, some systems such as [Labios](http://www.cs.iit.edu/~scs/assets/files/kougkas2019labios.pdf) enable a task-based approach to I/O where I/O is submitted as a task and then eventually executed on some worker. For this work, students will explore algorithms for scheduling I/O tasks to a collection of workers. This will require an examination of existing general task-based systems such as Charm++ and Legion, and the development of and comparison of several new algorithms for the purposes of scheduling I/O tasks.

**Prerequisite:** C, Algorithms

**Recommended team size:** 1-2

**PhD lead:** Keith Bateman ([kbateman@hawk.iit.edu](mailto:kbateman@hawk.iit.edu))

## Source Code Transformation Techniques

**Description**: Scientific applications cover a broad range of behavior, from physics simulations to weather visualizations and more. In all cases, performance of the High Performance Computing software stack is very important, but in order to ensure that the stack performs well, it is important to benchmark scientific applications. Particularly for I/O, a common technique is to use an I/O kernel: a simplified version of the application which simulates its I/O behavior. Often, the I/O kernel will run faster than the real application. For this project, students will explore and implement several techniques for transforming the source code of an application or I/O kernel to improve its runtime. Potentially, these techniques may change the behavior of the application and therefore not represent a completely accurate simulation of application behavior. You will be expected to benchmark the tradeoff between application performance and accuracy for different techniques and compare and contrast them.

**Prerequisite:** C, Python

**Recommended team size:** 2-3

**PhD lead:** Keith Bateman ([kbateman@hawk.iit.edu](mailto:kbateman@hawk.iit.edu))

## Data Provenance Survey

**Description**: Memory is typically allocated dynamically using explicit calls such as malloc and free, but storage is more complicated. Storage has write and read, but nothing to really delete stored data, because once data is stored there is some expectation of permanence. This survey will explore data provenance in several fields: data staging, cloud storage provisioning, and fault tolerance. For data staging, HPC often requires data to be moved from more permanent storage to a scratch storage while a computing job is being scheduled so that the job can access it conveniently during its run. For cloud storage, data provenance is even more important, as it is expected that data will be hosted on a platform which is remote from the user. Finally, fault tolerance represents an interesting edge case of data provenance, which is that data needs to be retained in the event of a fault. The student will have to explore all these facets of data provenance and provide a comprehensive survey of techniques for moving, freeing, and maintaining data within cloud and HPC systems.

**Prerequisite:** N/A

**Recommended team size:** 1

**PhD lead:** Keith Bateman ([kbateman@hawk.iit.edu](mailto:kbateman@hawk.iit.edu))

## I/O Prefetching for Deep Learning Applications:

**Description**: High-performance computing centers execute various deep learning applications relating to scientific simulation. Cosmic Tagger, for example, is a convolutional network used to separate cosmic pixels, background pixels, and neutrino pixels in a neutrinos dataset. However, the datasets used to train these models are massive, and the training phase may iterate over the dataset numerous times to reduce bias. This results in significant I/O overhead for these applications. Intelligent use of storage multi-tiering can reduce this overhead significantly by elevating important data to higher tiers of storage. In this project, students will analyze the I/O patterns of certain deep learning applications using the Deep Learning I/O Benchmark (DLIO) and develop an algorithm to prefetch data in a storage hierarchy. Students will have to consider factors such as random seeds and other model parameters to determine how to best adapt to these different applications. Overall, students will gain a deeper understanding of the behavior of deep learning applications and how to utilize a storage hierarchy to improve I/O performance.

**Prerequisite:** Python 3.x

**Recommended team size:** 2-3 (Students will analyze different deep-learning I/O applications in-depth)

**PhD lead:** Luke Logan

## Scalable, NUMA-Aware Shared-Memory Data Structures

**Description**: Process-to-process communication is widely used in HPC. MPI, for example, may use shared memory to communicate between processes on the same node to reduce data copying. Microkernels use low-latency shared-memory queues to implement system calls. However, Non-Uniform Memory Access (NUMA) has a significant impact on the performance of shared-memory data structures. Not all CPU cores can access the same memory regions at the same speed. NUMA can cause as much as 2x slow-down on these memory-sensitive workloads. With the advent of the upcoming Compute Express Link (CXL), which enables CPUs to place data directly in the memory of accelerators (e.g., GPU), NUMA effects are exacerbated. However, many shared-memory libraries (e.g., Boost, HCL) do not strongly consider the impact of NUMA in their implementation of important data structures such as the unordered\_map and unordered\_set. In this project, students will implement a few shared memory data structures while specifically considering the effects of NUMA. Students will quantify the impact of their NUMA-aware data structures compared to non-NUMA-aware data structures, such as Boost.

**Prerequisite:** C++

**Recommended team size:** 2-3 (Students will implement different data structures)

**PhD lead:** Luke Logan

## A Survey of OS Process Scheduling:

**Description**: Supercomputers currently rely on their OS for providing network, I/O, and process scheduling primitives. However, recent research has discovered that the default software provided by the OS can cause significant performance ramifications due to software overhead and workload-agnostic policies. For this reason, supercomputers are moving towards userspace I/O and networking stacks, which completely bypass the OS using technologies such as the Storage Performance Development Kit (SPDK) and the Data Plane Development Kit (DPDK). However, process schedulers have yet to fully adapt to this change in paradigm, which can result in significant overheads due to unintelligent context switching. In this work, students will conduct a survey of existing process scheduling methodologies, taking special consideration into how the process schedulers utilize knowledge of networking and I/O patterns to make decisions. In this project, students will gain a deep understanding of modern process scheduling and its performance implications.

**Prerequisite:** N/A

**Recommended team size:** 1

**PhD lead:** Luke Logan

## Survey on the common causes of I/O bottlenecks in HPC systems

**Description**: An academic paper survey on the common causes of I/O bottlenecks in high-performance computing (HPC) systems is needed to better understand and address this prevalent problem. HPC systems are essential for a wide range of scientific research and engineering applications, but their performance can be limited by I/O bottlenecks. These occur when the rate at which data is transferred to or from the system is slower than the rate at which the system can process it. A survey of existing research on the causes of I/O bottlenecks in HPC systems would provide valuable insight into the most common issues and potential solutions, such as studies like "IOMiner", "UMAMI" and "TOKIO" which have been proposed to address this problem. This knowledge could then be used to improve the design and performance of HPC systems, leading to more efficient and effective use of these resources.

**Prerequisite:** N/A

**Recommended team size:** 1

**PhD lead:** Izzet Yildirim ([iyildirim@hawk.iit.edu](mailto:iyildirim@hawk.iit.edu))

## Benchmarking Pandas with various optimization techniques

**Description**: This research project aims to conduct a comprehensive evaluation of the performance of the widely used Python library Pandas when utilizing Cython and Numba, two popular tools for optimizing Python code. The study will also compare the performance of Pandas with Cython and Numba to that of cuDF, a GPU DataFrame library. The benchmarking process will involve conducting a series of experiments on a diverse set of datasets, measuring the execution time and memory usage of each library. The results will be analyzed and discussed, with a focus on identifying the strengths and weaknesses of each library and determining their suitability for different types of data processing tasks. The findings of this study will provide valuable insights for practitioners and researchers in the field of data science and software development, as it will inform them on how to optimize the performance of their code when working with large datasets.

**Prerequisite:** Python 3.x, Pandas, Cython/Numba, CUDA Programming

**Recommended team size:** 2

**PhD lead:** Izzet Yildirim ([iyildirim@hawk.iit.edu](mailto:iyildirim@hawk.iit.edu))

## Set up a Dask cluster on Ares using SLURM

**Description**: This research project aims to investigate the potential benefits of utilizing Dask, a parallel computing library, in conjunction with the SLURM resource management system on an internal system called Ares. The study will involve the implementation and configuration of a Dask cluster on Ares, in order to assess its performance and scalability in comparison to traditional computing methods. Additionally, the project will include an analysis of the cluster's efficiency and effectiveness in handling large datasets and providing insights on the optimal configuration for such clusters. The findings of this study will provide valuable insights for practitioners and researchers in the field of computer science and distributed computing.

**Prerequisite:** Python 3.x, SLURM, Bash, Cluster Management

**Recommended team size:** 2

**PhD lead:** Izzet Yildirim ([iyildirim@hawk.iit.edu](mailto:iyildirim@hawk.iit.edu))

## A Survey of Data Prefetching for HPC Workflow

**Description**: Researchers and scientists solve real-world problems via the workflows of applications executing on HPC infrastructures. These applications usually have thousands to millions of interdependent tasks and can transfer terabytes or even petabytes of data. The de facto way to manage these data during workflow is through a shared file system. However, data movement between the simulation and analysis components of the workflow quickly becomes a serious bottleneck due to resource contention issues as well as network and storage device latencies. Intelligent data prefetching based on access patterns can hide disk access and data transfer latency by moving the data to DRAM before it is requested. However, the effectiveness of such prefetching depends upon the ability to recognize data access patterns and identify appropriate data to be prefetched. In this project, the student will undergo a deep dive into this field, exploring the publications surrounding this topic and generating a report summarizing the state of the art of the field.

**Prerequisite:** N/A

**Recommended team size:** 1-2

**PhD lead:** Meng Tang([mtang11@hawk.iit.edu](mailto:mtang11@hawk.iit.edu))

## Kubernetes Dynamic Volume Provisioning Benchmark

**Description**: Container technology has been rapidly growing and widely adopted in cloud computing because of its ability to create customizable, portable, and isolated environments for running applications. The development of container orchestration tools like Kubernetes further enhances its ability to dynamically provision containers to run large-scale applications. However, the storage performance of Kubernetes is not well studied. In this project, we want students to delve into Kubernetes Volumes, and learn to use the different Container Storage Interface (CSI) to run Kubernetes Volumes. Benchmark and performance analysis should be done on the Dynamic Volume Provisioning function. The two storage backends we want to compare are BeeGFS and Google Cloud Storage, and they both have available CSI for Kubernetes. The performance should be compared against the default Linux file system, which usually is XFS, BTRFS, EXT4, or as documented.

**Prerequisite:** Linux, Scripting, Docker/Kubernetes (ideally)

**Recommended team size:** 2 (available for 2 teams, one for using BeeGFS CSI and another for using Google Cloud Storage CSI)

**PhD lead:** Meng Tang([mtang11@hawk.iit.edu](mailto:mtang11@hawk.iit.edu))

## Parallel Sorting Algorithms in Python

**Description**: Sorting algorithms are widely used in many programs. With the growing data volume in many scientific applications and the increasing number of available processors in supercomputers, distributed and parallel processing becomes important for program scalability. Parallelization of sorting algorithms, especially merge sort, has been well-studied in the past, with many existing designs and implementations. On the other hand, the simplicity and extensive selection of libraries make Python programming the most popular scientific programming language. In particular, the python library Dask, a flexible library for parallel algorithms and out-of-core computation. In this project, we want students to 1) learn how to install and use Dask, 2) build a parallel multithreaded or multi-process sorting algorithm using Dask, and 3) evaluate the performance against a sequential sorting algorithm. The algorithm can be any sorting algorithm not limited to merge sort, students should justify the choice. The Dask cluster can be set up on a single computer.

**Prerequisite:** Python, Multithread/Multiprocess Computing

**Recommended team size:** 2

**PhD lead:** Meng Tang([mtang11@hawk.iit.edu](mailto:mtang11@hawk.iit.edu))

## A Survey of Distributed Inference Serving Systems for Deep Learning Models

**Description**: In recent years, deep learning has shown significant improvement in many fields, such as computer vision, image recognition, and climate forecast. A deep learning project often needs to deploy the trained model in production for inference serving and the number of applications relying on inference from machine learning models keeps growing. With abundance of existing data, continuous new data generation, and training larger deep learning models, the models have high performance at the expense of requiring bigger machines in terms of CPU/GPU and memory. Thus, there is increasing demand for providing distributed inference serving for these large-scale models. Therefore, it is highly required and helpful to study state-of-the-art distributed inference serving for deep learning models, summarizing how they manage different models and provide distributed inference serving.

**Prerequisite:** Deep Learning, Inference

**Recommended team size:** 1

**PhD lead:** Jie Ye (jye20@hawk.iit.edu)

## Implement and evaluate asynchronous checkpointing for DNN models

**Description**: Checkpoints play an important role in training long running machine learning (ML) models. Checkpoints take a snapshot of an ML model at regular intervals while model training and store it in persistent storage so that they can be used to recover from failures to ensure rapid training progress. Typically, there are two different approaches to implement checkpoints for DNN models, which are synchronous checkpointing and asynchronous checkpointing respectively. In synchronous mode, frequent checkpointing during training will slow down the training performance, which means that it needs to take longer to train a DNN model. Thus, it is necessary to design and implement an efficient, low-overhead asynchronous checkpointing approach for DNN models to hide the I/O overhead.

**Prerequisite:** Python, Tensorflow or Pytorch

**Recommended team size:** 1

**PhD lead:** Jie Ye (jye20@hawk.iit.edu)

## Evaluate Storage Performance for Deep Learning Workloads

**Description**: Parallel file systems are frequently deployed on leadership High Performance Computing (HPC) systems to ensure efficient I/O, persistent storage and scalable performance. There are many well-known PFSs, such as OrangeFS, BeeGFS, Lustre, and Ceph. However, emerging Deep Learning (DL) applications incur new I/O and storage requirements to HPC systems with batched input of small random files. Contrasting from the traditional well-structured HPC I/O pattern (e.g., checkpoint/restart, multi-dimensional I/O access), the DL training often gives rise to highly random small file accesses. In addition, it is said that object stores can provide high scalability, rich metadata, and the ability to store unstructured data. It is interesting to explore the optimal storage system for storing DL data and the possible trade-off. Therefore, there is a need to benchmark different storage systems with emerging DL applications and compare their performance results.

**Prerequisite:** N/A

**Recommended team size:** 2

**PhD lead:** Jie Ye (jye20@hawk.iit.edu)

## CaL-Driven Dynamic Data Block Size

**Description**: Memory performance optimizations generally fall into two categories, optimizations to improve data locality and optimizations to improve data concurrency. Locality-based optimization is a well-studied topic and has been the focus of data access optimization for many years. Concurrency-based technologies have been deployed in modern memory systems to overlap or hide data access latency. CaL[[1]](#footnote-0) is a performance model to quantify locality with the consideration of data access concurrency. CaL can be used to optimize memory performance. In this study, we will consider using CaL to decide the optimal data block size. In hierarchical memory systems, a larger cache block generally consumes more bandwidth but can present more reuse opportunities for later accesses if the locality is strong. Moreover, because different workloads can have distinct behaviors and even the same workload can have different phases, a fixed cache block size generally cannot achieve good matching between data access patterns and the underlying memory system. Your task is to understand the CaL model and use CaL to dynamically find the optimal data block size. You need to implement your design in the ChampSim simulator[[2]](#footnote-1). This project provides students an opportunity to learn and gain hands-on experience in architecture design.

**Prerequisite:** C, C++, Algorithms

**Recommended team size:** 1

**PhD lead:** Xiaoyang Lu ([xlu40@hawk.iit.edu](mailto:xlu40@hawk.iit.edu))

## Scalable and High-Performance Scheduler for Multi-Core Systems

**Description**: In modern systems, the high latency of accessing large-capacity off-chip memory and limited memory bandwidth have made main memory a critical performance bottleneck. In a multicore system, main memory is typically shared by applications running on different cores. Applications running on different cores interfere at main memory. In this project, you will implement and evaluate two memory scheduling policies: ATLAS[[3]](#footnote-2) and BLISS[[4]](#footnote-3). To this end, you will extend ChampSim[[5]](#footnote-4), which is a publicly available architectural memory simulator, to model the memory scheduling policy. The key idea of ATLAS is to periodically order threads based on the service they have attained from the memory controllers and prioritize threads that have attained the least service compared to the others in each period. BLISS scheduler classifies applications into interference-causing and vulnerable-to-interference groups by monitoring the number of consecutive requests served from each application at the memory controller. During scheduling, vulnerable-to-interference applications’ requests are given higher priority over requests of interference-causing applications. BLISS reduces the interference between the two groups and improves system throughput and fairness.

**Prerequisite:** C, C++, Algorithms

**Recommended team size:** 1

**PhD lead:** Xiaoyang Lu ([xlu40@hawk.iit.edu](mailto:xlu40@hawk.iit.edu))

## Bingo Spatial Data Prefetcher

**Description**: Applications extensively use data objects with a regular and fixed layout, which leads to the recurrence of access patterns over memory regions. Spatial data prefetching techniques exploit this phenomenon to prefetch future memory references and hide the long latency of DRAM accesses. While state-of-the-art spatial data prefetchers effectively reduce the number of data misses, there is still significant room for improvement. Since they often associate the observed spatial patterns to just a single event significantly, which limits the effectiveness of spatial data prefetchers. Bingo[[6]](#footnote-5) is a spatial data prefetch that associates the observed spatial patterns to more events to achieve high accuracy while not losing prediction opportunities. Your task is to extend the ChampSim simulator[[7]](#footnote-6) with the Bingo prefetch algorithm, as described in Section IV in the paper that proposed Bingo. You should make sure your implementation is functionally equivalent to the mechanism described in the paper. This project allows students to learn and gain hands-on experience in architecture design.

**Prerequisite:** C, C++, Algorithms

**Recommended team size:** 1

**PhD lead:** Xiaoyang Lu ([xlu40@hawk.iit.edu](mailto:xlu40@hawk.iit.edu))

1. Liu, Yuhang, and Xian-He Sun. "CaL: Extending data locality to consider concurrency for performance optimization." IEEE Transactions on Big Data 4.2 (2017): 273-288 [↑](#footnote-ref-0)
2. ChampSim simulator: https://gitlab.ethz.ch/rahbera/champsim [↑](#footnote-ref-1)
3. Yoongu Kim, Dongsu Han, Onur Mutlu, and Mor Harchol-Balter. ATLAS: A Scalable and High Performance Scheduling Algorithm for Multiple Memory Controllers. In HPCA, 2010 [↑](#footnote-ref-2)
4. Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu. BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling. TPDS, 2016 [↑](#footnote-ref-3)
5. ChampSim Simulator. https://github.com/ChampSim/ChampSim [↑](#footnote-ref-4)
6. Bakhshalipour, Mohammad, et al. "Bingo spatial data prefetcher." 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2019 [↑](#footnote-ref-5)
7. ChampSim simulator: https://gitlab.ethz.ch/rahbera/champsim [↑](#footnote-ref-6)