# Exploring Fine-grained Memory Analysis for PIM Offloading

Thomas M. Papka[1], Nanda Velugoti[2] (advisor), Kyle Hale[2] (advisor)

[1]Loyola University, Chicago, IL 60660

[2]Illinois Institute of Technology, Chicago, IL 60616

## Objective

- Develop an analysis pipeline that determines when to offload memory-intensive tasks to **Processing-in-Memory (PIM)** devices.
- Enhance computational efficiency and cost-effectiveness by optimizing memory usage.
- Eventually integrate the pipeline with **Globus Compute**, a Function-as-a-Service (FaaS) platform.
- Leverage **Globus Compute's Python-based** environment to facilitate easy adoption and usage by Python programmers.

## Memory Bound Applications

- Applications often become **memory-bound**, where performance is limited by memory access speed rather than CPU speed.
- The CPU waits for data transfers between memory and the processor, causing **idle CPU cycles**.
- **Memory retrieval** (RAM) is slower than CPU processing, making programs like simulations, data analysis, or graphics processing more susceptible to being memory-bound.
- **High memory access latency** causes delays as processors wait for data.
- **Limited cache utilization** worsens the problem, as effective cache use reduces memory bottlenecks by storing frequently accessed data closer to the CPU.
- **Cache misses** occur when the working set size exceeds cache size or access patterns are irregular, requiring additional access to slower main memory.
- **Bandwidth constraints** arise when the memory subsystem cannot supply data to the CPU fast enough, further limiting performance.

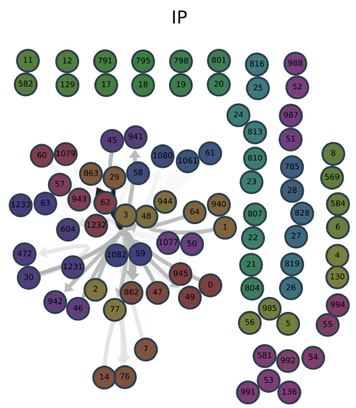
## Background

- **Processing in Memory (PIM)** is a promising solution to memory-bound system limitations.
- Traditional architectures face inefficiencies in data transfer between CPU and memory, causing bottlenecks.
- PIM reduces data movement by enabling computation directly within the memory, saving time and energy.
- **Mutlu et al.** highlight how PIM addresses issues like high memory access latency and limited cache utilization.
- **Chen et al.** introduce a framework that integrates PIM into existing systems with minimal code changes.
- PIM improves performance, reducing latency and increasing throughput, making it ideal for memory-bound applications.
- Boosting RAM capacity could help, but using PIM is often more practical and cost-effective.
- The **key challenge** is deciding when to transfer tasks to PIM devices for optimal performance.

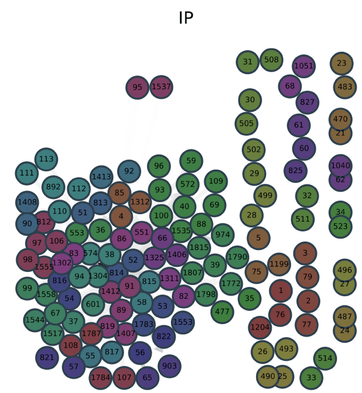
## Optimizing Memory Usage with MemGaze

- **Our Workflow:** The process begins by writing the application function in **Python**, then deploying it using **Globus Compute FaaS** (currently under development). To ensure compatibility with MemGaze, the Python function is converted into **C** using Cython. The function is executed with MemGaze to collect memory trace data. Finally, the trace files are analyzed to generate **cluster graphs** that reveal memory usage patterns and potential optimization points.
- **MemGaze Analysis:** MemGaze evaluates memory reuse patterns across both **temporal** and **spatial** dimensions, highlighting areas for optimization. The generated **clustered graphs** visualize these memory usage patterns, helping developers pinpoint sections of the code that can benefit from PIM offloading. This process reduces memory bottlenecks and enhances the overall performance of memory-bound applications.

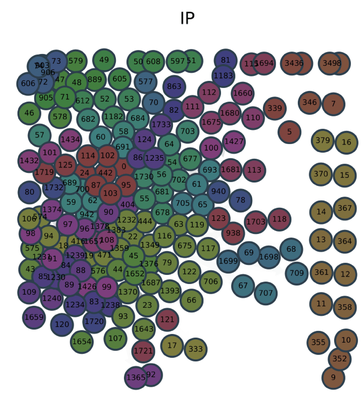
## Results



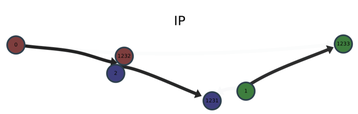

Unfiltered Vector Addition Results



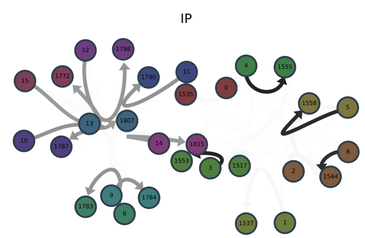

Unfiltered Breadth-First Search Results



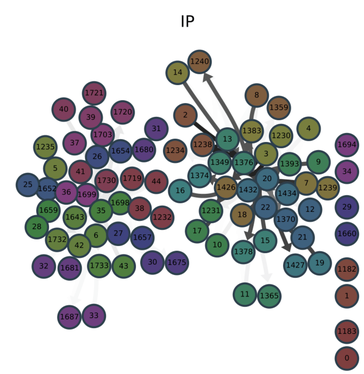

Unfiltered Reduction Results




Filtered Vector Addition Results




Filtered Breadth-First Search Results




Filtered Reduction Results

## Results

- The trace text files must first be **filtered** to generate meaningful graphs for analysis.
- This filtering process isolates the **original Python code**, excluding additional Cython-generated code.
- A custom **Python script** allows users to search for and focus on original Python functions.
- Ensures that the analysis pertains solely to Python code, avoiding interference from extraneous Cython functions.
- Proper filtering results in more **accurate and meaningful graphs**, facilitating clearer insights during analysis.

## Conclusion

- The project streamlines the process of identifying memory usage in applications.
- Helps determine whether portions should be offloaded to a **Processing-in-Memory (PIM)** device.
- Provides tools for programmers to make informed decisions about PIM utilization.
- Future work includes automating the process and improving **graph analysis** capabilities.
- More detailed explanations and guidance on offloading decisions are needed.
- The pipeline design could be integrated into a **Function-as-a-Service (FaaS)** platform like Globus Compute.
- The current process is simple, allowing any programmer to analyze the memory usage of **Python programs**. It is not limited to only Python programs, it can also be used for **C programs**.

## References

[1] André Bauer, Haochen Pan, Ryan Chard, Yadu Babuji, Josh Bryan, Devesh Tiwari, Ian Foster, and Kyle Chard. The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud. *Future Generation Computer Systems*, 153:558–574, 2024.

[2] Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, Yuxin Guo, and Onur Mutlu. Simplepim: A software framework for productive and efficient processing-in-memory. In *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 99–111. IEEE, 2023.

[3] Ozgur O Kilic, Nathan R Tallent, Yasodha Suriyakumar, Chenhao Xie, Andrés Marquez, and Stephane Eranian. Memgaze: Rapid and effective load-level memory trace analysis. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 484–495. IEEE, 2022.

[4] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. A modern primer on processing in memory. In *Emerging computing: from devices to systems: looking beyond Moore and Von Neumann*, pages 171–243. Springer, 2022.

## Acknowledgments

This work was funded by the Research Experiences for Undergraduates (REU) program, supported by the National Science Foundation under grant OAC-2150500. I want to thank my co-authors and REU advisors, Nanda Velugoti and Kyle Hale, for their invaluable guidance and support throughout this project. I also wish to thank my fellow REU students for their collaboration and camaraderie during this summer's research experience.

LOYOLA UNIVERSITY CHICAGO

ILLINOIS TECH