Abstract
We propose PIMMiner, a high-performance PIM architecture graph mining framework.

- We first identify that current PIM architecture cannot be fully utilized by graph mining applications.
- Next, we propose a set of optimizations that enhance the locality, and internal bandwidth utilization and reduce remote bank accesses and load imbalance through cohesive algorithm and architecture co-designs.
- We compare PIMMiner with several state-of-the-art graph mining frameworks and show that PIMMiner is able to outperform all of them significantly.

Background
- Graph pattern mining (GPMI) needs to find all subgraphs with different patterns that meet the application requirements.
- GPMI applications are considered as a new class of data-intensive applications that generate massive irregular computation workloads and memory accesses, which degrade the performance significantly.
- Processing-in-Memory (PIM) integrates processing units inside the memory to reduce the overhead of frequent data movement and achieve high-performance and energy-efficient computation.
- Samsung has recently started manufacturing HBM-PIM chips. The HBM-PIM incorporates PIM cores inside of memory banks. There are three ways for a PIM unit to access memory: (1) near-core bank access, (2) intra-channel bank access; (3) inter-channel remote bank access.

Motivation
- Directly offload the GPMI execution kernel to PIM cannot achieve desired performance. We observe high load imbalance and lots of inter-channel remote bank accesses.

<table>
<thead>
<tr>
<th>TABLE 1: Performance between 96-threads CPU (L3 cache) and 128-core PIM (L3 cache)</th>
<th>CC</th>
<th>CPU</th>
<th>PIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Processing Time</td>
<td>3.710x</td>
<td>3.690x</td>
<td>3.630x</td>
</tr>
<tr>
<td>Execution Time (s)</td>
<td>2.150</td>
<td>1.900</td>
<td>1.990</td>
</tr>
<tr>
<td>Peak/Execution Speedup</td>
<td>4.07x</td>
<td>4.15x</td>
<td>3.66x</td>
</tr>
</tbody>
</table>

- We compare PIMMiner with several state-of-the-art graph mining frameworks and show that PIMMiner achieves even higher performance with the hardware GPMI accelerators.

PIMMiner Framework Design
- **Conditional Access Filter (Filter):** PIMMiner adds lightweight hardware dedicated to filter unnecessary data from memory. According to AutoMine, PIM units only need the nodes $v_1 < v_2$.
- **Local-First Data Mapping (Remap):** PIMMiner proposes a new address mapping method to exploit low-latency and high-bandwidth local memory bank accesses for PIM units. The new mapping maps the data in the same neighbor list to the same bank group.

HBM-PIM Architecture

**Fig1:** Performance of PIMMiner with the effectiveness of proposed optimizations. In each bar, we show the average time across cores (the solid line) and the total execution time (top of bar).

**Evaluation Results**
- Overall, by enabling all optimizations, PIMMiner achieves 16.81x average speedup and 137.32x maximum speedup over the baseline PIM.
- The performance improvement from PIMMiner optimizations:
  - Filter: 2.01x average, 17.57x maximum speedup
  - Remap: 1.39x average, 2.47x maximum speedup
  - Duplication: 1.84x average, 3.05x maximum speedup
  - Stealing: 3.01x average, 28.87x maximum speedup
- 1.49x average, 2.25x maximum speedup

**References**

Acknowledgement
This work is collaboratively done with Dr. Peng Jiang at the University of Iowa. Our research is generously supported by the National Science Foundation under grants CCF-2048014 and CCF-2028925.