Magic Memory: A Programming Model for Interactive Data Analysis

Carnegie Mellon
Electrical & Computer ENGINEERING

Eric Tang, Franz Franchetti

Abstract

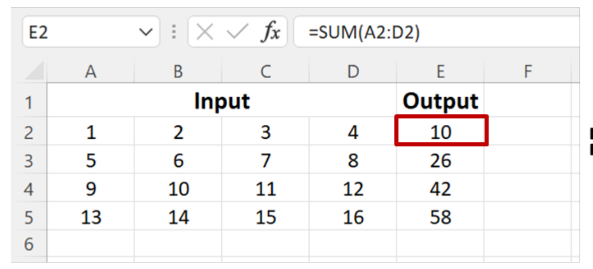
Magic Memory is a new programming model for big data analysis. In this model, persistent invariants or functional dependencies can be established between memory regions. These functional dependencies will always be held true when this memory region is being read. Recent technological advancements enable Magic Memory at the hardware level, providing performance that is hard to achieve with a software-only solution. Our ongoing work seeks to explore an implementation of Magic Memory on a CPU-FPGA system, where the CPU runs the host code while the FPGA provides hardware acceleration.

Interactive Data Analysis

- Domain experts explore a wide range of questions on the data set
  1. Formulate question for the data set
  2. Modify the dataset
  3. Perform computation in question
  4. Repeat based upon results

Problem:

- Slow iterative process which distracts from expert's main focus



E2 =SUM(A2:D2)

	A	B	C	D	E	F
1		Input			Output	
2	1	2	3	4	10	
3	5	6	7	8	26	
4	9	10	11	12	42	
5	13	14	15	16	58	
6						

=

Address	Value
0x01	1
0x02	2
0x03	3
0x04	4
0x05	10
0x06	5
0x07	6
0x08	7
0x09	8
0x0a	26

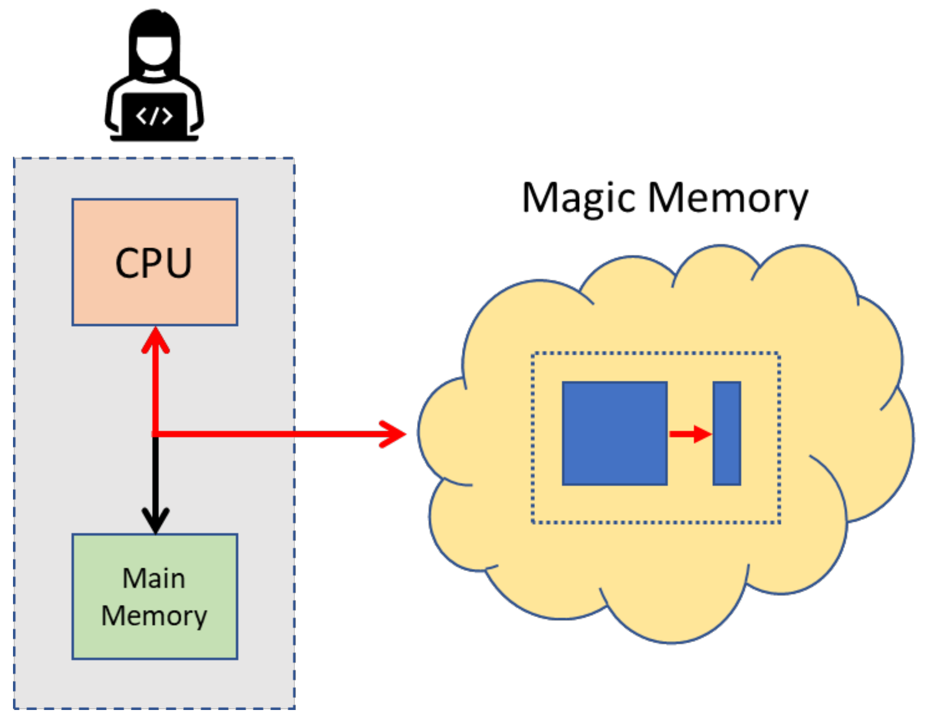
Address	Value
0x0b	9
0x0c	10
0x0d	11
0x0e	12
0x0f	42
0x10	13
0x11	14
0x12	15
0x13	16
0x14	58

Solution

- Create an intuitive simple API to allow domain experts to explore various properties of a dataset
- Design a programming model that allows users to utilize a heterogeneous architecture to its full potential

Programming Model

- Memory maintains a persistent invariant that is declared by the user
- User writes program as though memory region will "magically" be computed
- Separate program to be written to describe the invariant to be maintained. This program may target other hardware platforms.
- Upon any write to the magic memory region, the invariant is automatically recalculated. Reads to this memory region will always return a value that is consistent with the current state.



Example: Calculating PageRank using Magic Memory

Using this programming model on a CPU-FPGA system
- CPU allocates memory on FPGA NUMA domain

Example: Exploring the PageRank of large graphs

Input: Graph (millions of nodes)
Invariant: PageRank of the input graph

x – vector containing PageRank of each node
α – damping factor (probability of a random jump)
A – input graph
N – number of nodes

$$x_{i+1} = \frac{(1-\alpha)}{N} + \alpha A x_i$$

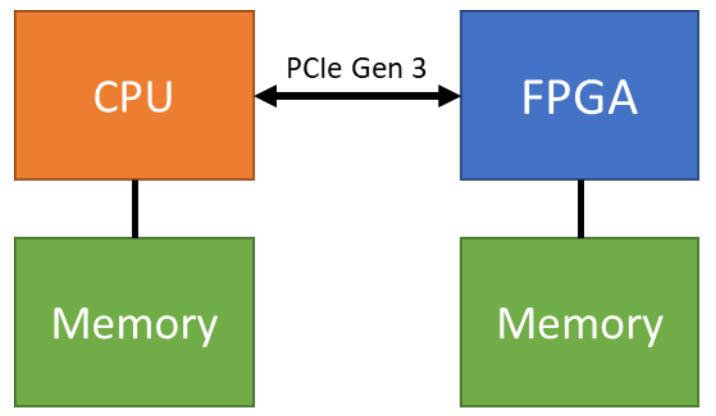
Questions to be explored

Upon modifying an input node / edge, what is the new PageRank of the graph?
- Does the node with the highest PageRank change?
- How has a specific node's PageRank changed?

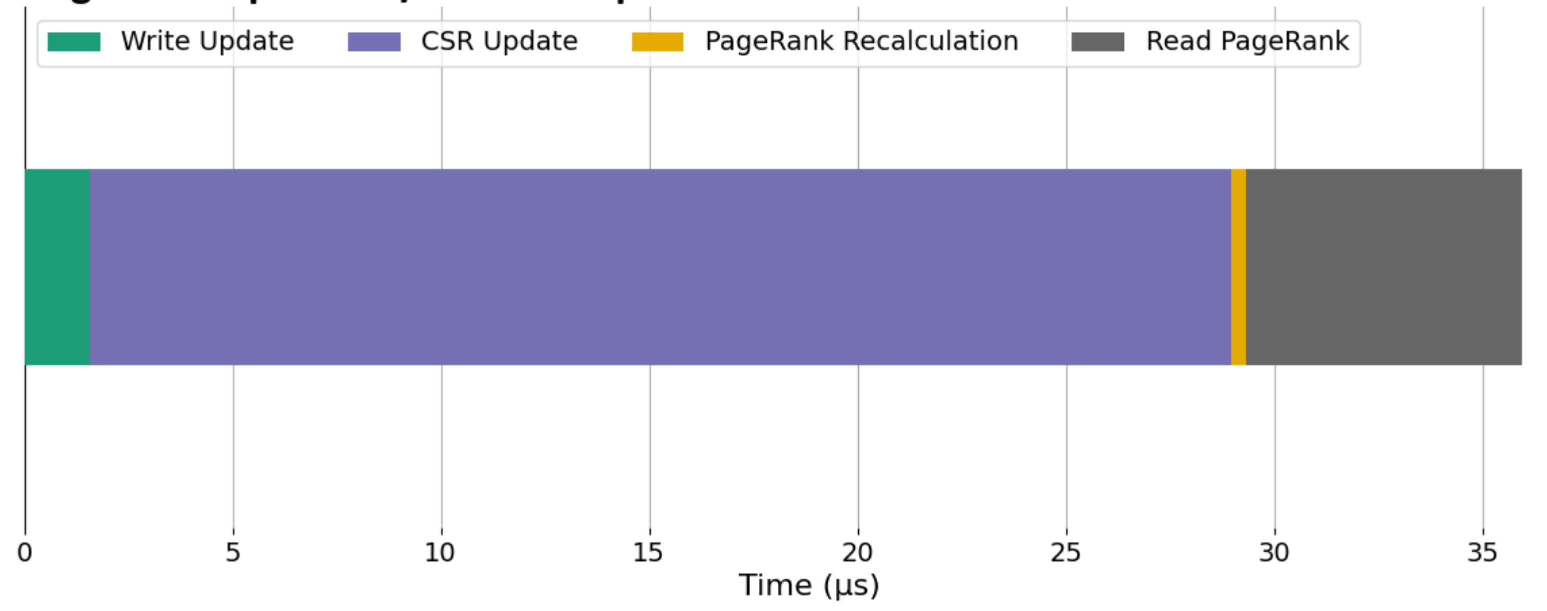
```
int magic_mem_PageRank(){

    int N = 10000000;

    int dims_op1[3] = {N, N, 0};
    int dims_op2[2] = {N, 0};

    // Adjacency Matrix
    uint64_t* a = magic_mem_in_alloc(dims_op1);

    // PageRank Vector
    uint64_t* c = magic_mem_out_alloc(dims_op2);

    // Write values to matrix A
    int row, col;
    for(int i = 0; i < NNZ; i++) {
        row = rand() % N;
        col = rand() % N;
        a[row][col] = 1;
    }

    // Check output
    uint64_t page_rank;
    for(int j = 0; j < N; j++)
        page_rank = c[j];

    return 1;
}
```

Results

CPU: Intel i7-10700
FPGA: Stratix 10 MX
Graph: web-NotreDame [3]
Nodes: 0.33 million
Edges: 1.45 million
Avg. Degree: 4.61



PageRank Update w/ Efficient SpMV Hardware Performance Breakdown



- From modeling more efficient SpMV hardware [3], the latency for performing an update can be seen to be around 30 µs

Future Work

- Expand Magic Memory API to allow for different techniques for accessing data
  - Get all neighbors
  - Traverse edges
  - Get row / col
- Allow users to describe users in a high-level language and use SPIRAL to generate the appropriate hardware

Spiral
Software/Hardware Generation for Performance

References

[1] J. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. Haas, "Interactive data analysis: the control project," Computer, vol. 32, no. 8, pp. 51–59, 1999.
[2] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999
[3] F. Sadi, J. Sweeney, T. M. Low, J. C. Hoe, L. Pileggi, and F. Franchetti, "Efficient spmv operation for large and highly sparse matrices using scalable multi-way merge parallelization," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019