Itemrank

Φοιτητές
Εμμανουηλίδης Κωνσταντίνος 57315
Χατζηπαπάς Νίκος 57370

What we will see...

- What is ItemRank - Motivation
- Code
- Code Optimizations
- Optimization in Catapult
- Verification
- Testing

ItemRank

- Variation of Pagerank
- Used in recommendation systems
- Goal: recommend item based on previous users' ratings

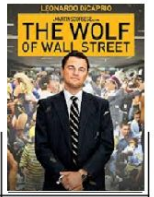
- Main Equation:

$$\begin{cases} \mathbf{IR}_{u_i}(0) = \frac{1}{|\mathcal{M}|} \cdot \mathbf{1}_{|\mathcal{M}|} \\ \mathbf{IR}_{u_i}(t+1) = \alpha \cdot \mathcal{C} \cdot \mathbf{IR}_{u_i}(t) + (1-\alpha) \cdot \mathbf{d}_{u_i} \end{cases}$$

- C : adjacency matrix of graph with movies
- IR : itemrank of each movie
- d : user rating of movies

Movies

Users

				
Bob	4	?	?	4
Alice	?	5	4	?
Joe	?	5	?	?
Sam	5	?	?	?

Code

Code

Problems/Dependencies:

- Scalability
- Dual Port

Optimizations

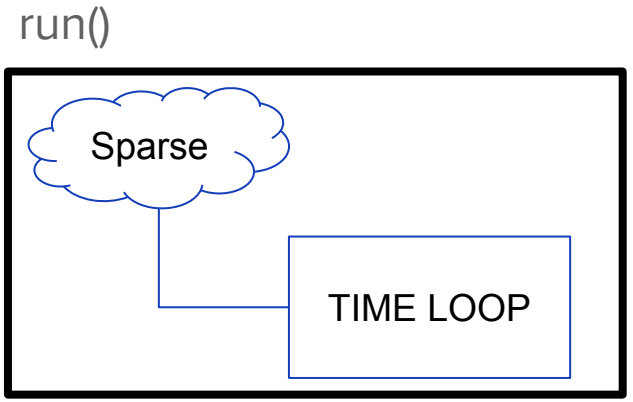
- Sparse Representation
- Tilling
- 128 bit representation
- unroll USER + MOVIE

```
#pragma hls_design interface
void run (data_type movies_correlation[m][m],data_type initial_critics[m][n],data_type IR_new[m][n]){

    USER:for(int user = 0; user < n; user++) { //for each user
      // initialize IR
      IR_COL_INIT:for (int j = 0; j < n; j++) {
          IR_col[j] = 1.0/m;
      }

      TIME:for (int t = 0; t < max_iter; t++) { //for each time

          MOVIE:for(int i = 0; i < m; i++) { //for each movie
              summation = 0;

              INNER_PRODUCT: for(int j = 0; j < m; j++) {
                  summation = summation + movies_correlation[i][j] * IR_col[j];
              }
              IR_col_new[i] = a * summation + (1-a)*initial_critics[i][user]; //rating of i-th movie of user
          }

          UPDATE_IR:for(int j = 0; j < m; j++) {
              IR_col[j] = IR_col_new[j];
          }
      }
      // write to output
      WRITE_OUTPUT:for(int i = 0; i < m; i++){
          IR_new[i][user] = IR_col[i];
      }
    }
}
```

Optimizations

Sparse Representation

- M = m x m ➔ S = m x num_elements
- C = n x m ➔ SC = n x num_elements
- Num_elements max elements of each row of M
- Find max elements: Heap

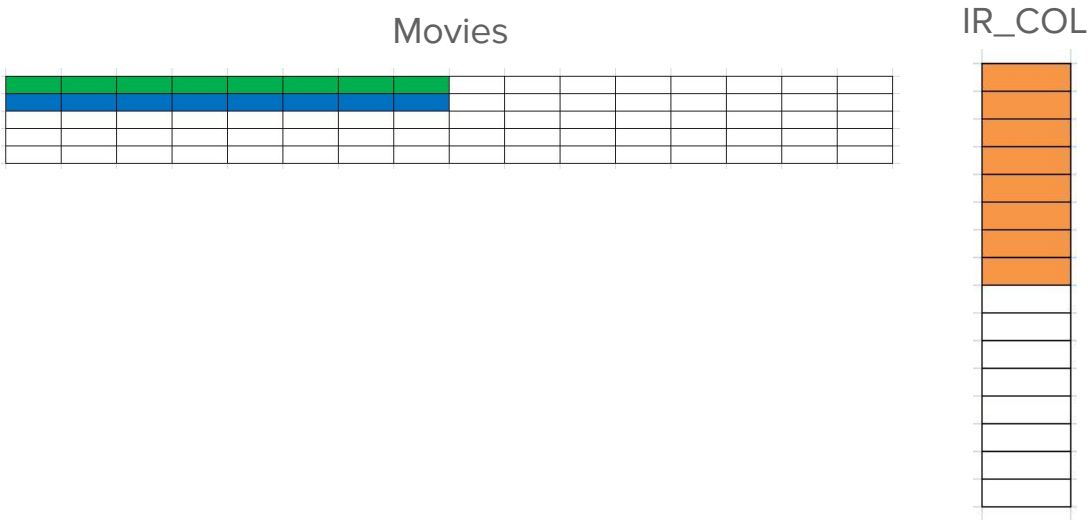


```
void preprocessing_movies (data_type movies_correlation[m][m], compressed_data_type sparce_movies[m][num_elements],
                           int sparce_movies_index[m][num_elements]){
    for(int i = 0; i < m; i++){
        MinHeap mheap(num_elements, movies_correlation[i],  sparce_movies_index[i]);
        //First k elements
        mheap.findkBiggest(num_elements, movies_correlation[i]);
        mheap.saveAs128bit(sparce_movies[i]);
    }
}
```

```
void preprocessing_critics (data_type initial_critics[n][m], int low_index, compressed_data_type sparce_critics[m][num_elements/8]
                            int sparce_critics_index[m][n]){

    for(int user = 0; user < 2; user++){
        //create MinHeap object and find the k biggest elements to make critics array sparce
        MinHeap mheap(num_elements,  input: initial_critics[low_index+user],   pointer_indices: sparce_critics_index[low_index+user]);
        mheap.findkBiggest(num_elements,  input: initial_critics[low_index+user]);

        critics_heap_inside_module[user] = mheap;
    }
}
```

Tilling

- Two rows simultaneously ➔ Movies Unroll 2
- IR_col once read for two rows ➔ Reusability data, Less memory reads
- Run for 2 users ➔ movies once read for 2 users



128 bit representation

- 8 words of 16 bits ➜ 1 word of 128 bits
- Sparse movies saved in 128 bits representation
- Better use of memory bus

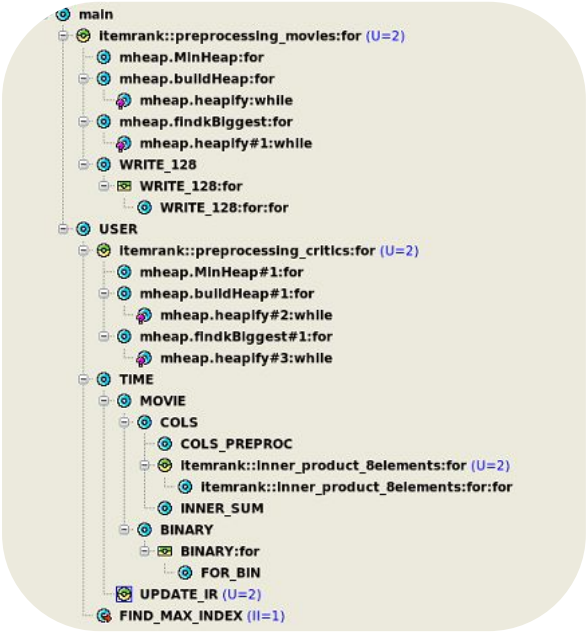
```
void saveAs128bit(compressed_data_type *pointer_heap){
    //compress 8 words of 16bits to one word of 128 bits
    WRITE_128:for(int i=0; i<compressed_size; i++){
    for(int l=0; l<8; l++){
        for(int j=total_num_digits-1; j>=0; j--){
            buffer[(16*l)+j] = arr[l][j];
        }
    }
    pointer_heap[i].set_slc( lsb: 0,buffer);
    }
}
```

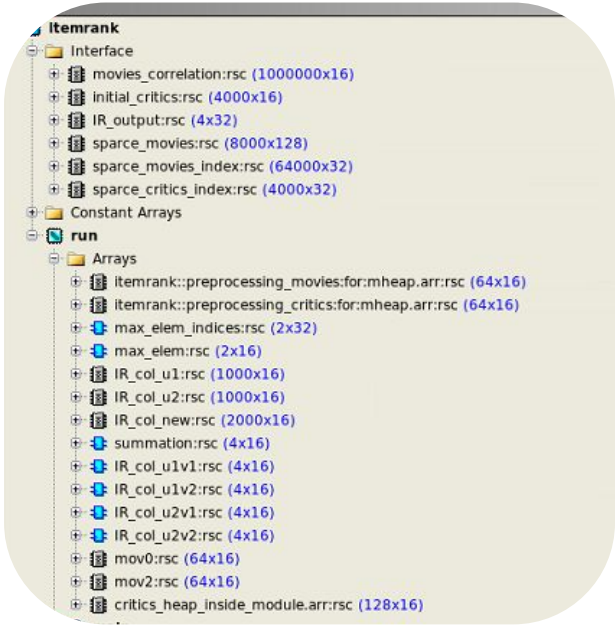
```
void inner_product_8elements(data_type IR_col1[4], data_type IR_col2[4],
                             data_type IR_col3[4], data_type IR_col4[4],
                             compressed_data_type sparce_movies[m][num_elements], int row_to_read

    //read 128-bit word and save it as 8 words of 16 bits
    inner_data1 = sparce_movies[row_to_read][last_read];
    inner_data2 = sparce_movies[row_to_read+1][last_read];
    for(int l=0; l<8; l++){
        mov0[l] = inner_data1.slc<16>( index: 16*l);
        mov01[l] = inner_data2.slc<16>( index: 16*l);

    }
    last_read++;
```

Catapult Optimizations

Catapult Optimizations

- Optimization over Latency (instead of area)





Tables

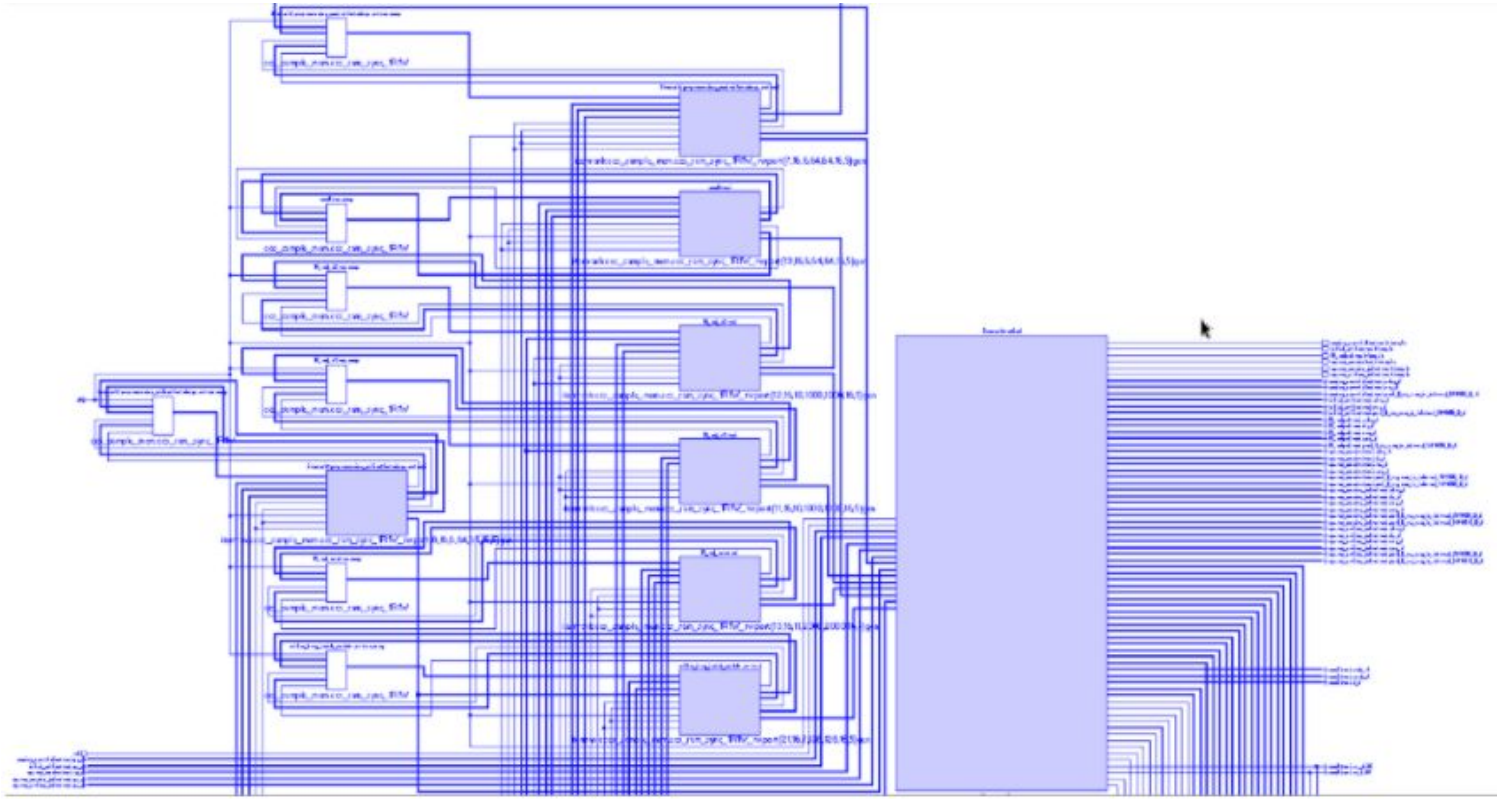
Initial Code

Solution /	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area
Itemrank.v2 (extract)	**160652105**	**321304210.00**	**160652118**	**321304236.00**	**4631.03**

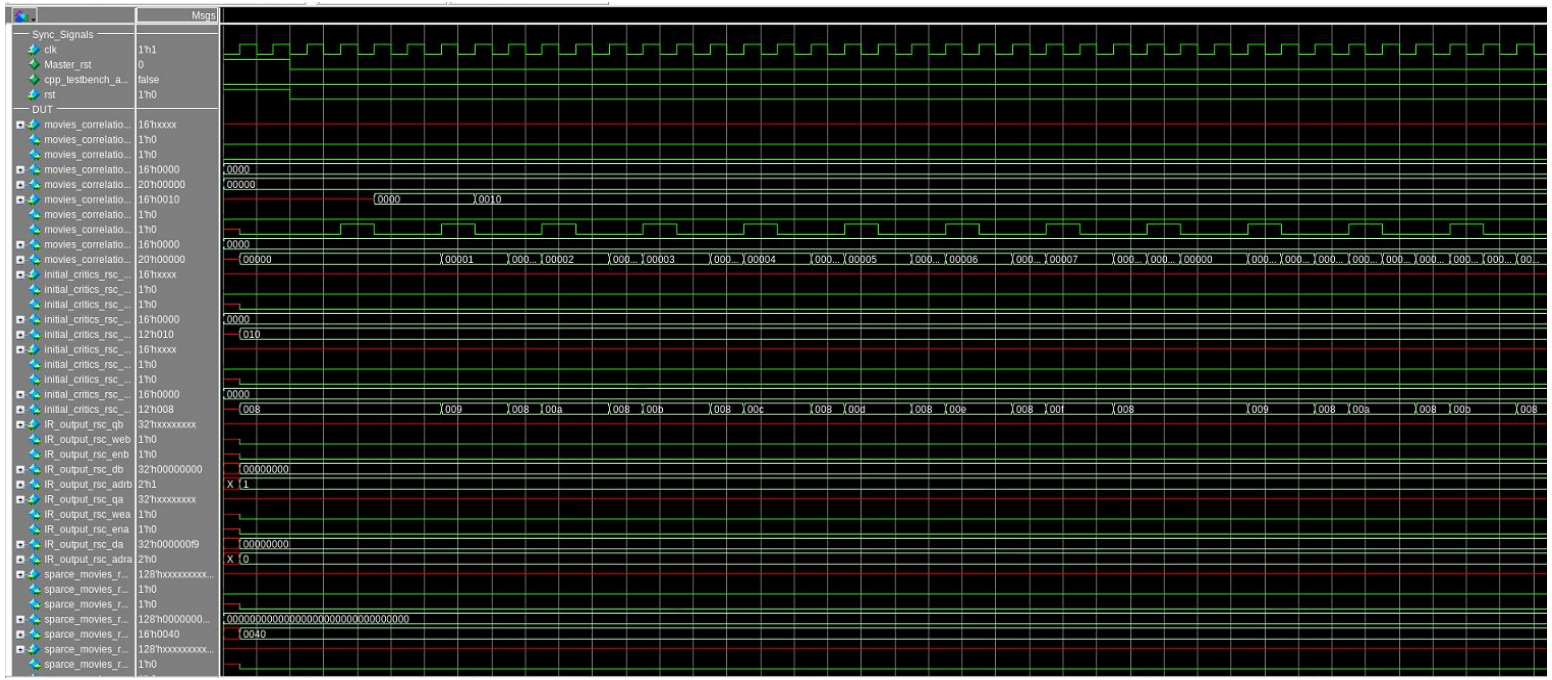
Code

Solution /	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area
Itemrank.v6 (extract)	33744457	67488914.00	33744462	67488924.00	22080.44
Itemrank.v7 (extract)	31742457	63484914.00	31742462	63484924.00	33234.93
Itemrank.v8 (extract)	**18370497**	**36740994.00**	**18370502**	**36741004.00**	**45679.87**

Architecture



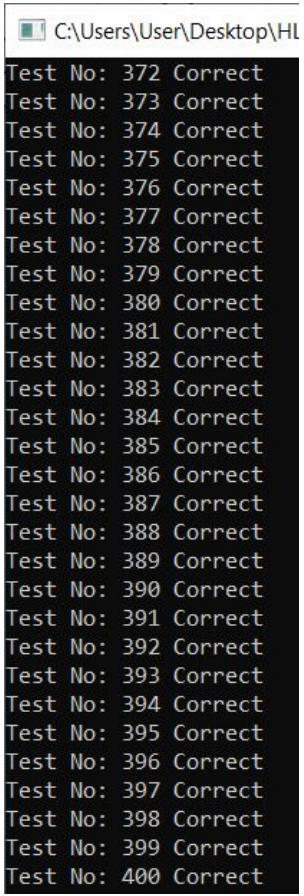
Verification



Testing

Testing

- There is no ground truth
- So, tests on:
  - small graphs of movies that are correlated arbitrarily
  - big graphs of movies that are not significantly correlated to each other



Thank you!
Questions..?