ECONOMICALLY EFFICIENT CACHE FRIENDLY ADJACENCY LIST FOR GRAPH PROCESSING ALGORITHMS

Arman MARTIROSYAN
Doctor of Science, Professor, YSU
Rafayel KHACHATRYAN
R&D Engineer, NPUA, MA
ICTEI, Chair of Microelectronic circuits and systems
Rafayel VEZIRYAN
Junior Researcher, NPUA, MA
Informatics and applied mathematics faculty

Keywords: data structures, computational efficiency, cache optimization, adjacency list, graph

Introduction. Graphs are a fundamental part of computer science and various scientific fields. They are used in numerous applications, from social networks to electronic design automation. Efficient graph representation is important for modern algorithms, especially in economical contexts [Wang, 2022, 1-6], where resource-efficient implementations play a significant role by reducing the resource consumption. This is essential in environments where computing resources are a high priority. There are many representations of graphs as data structure. One of these types is adjacency list: it is a collection of linked lists which shows the connection of source and destination vertices. Adjacency lists are more memory-efficient than adjacency matrices because they only allocate memory for vertices with at least one edge in the graph. However, traditional implementations of adjacency lists do not fully use the features provided by modern CPUs, such as the cache line. Our design of Adjacency List is cache friendly and is keeping the main structure of traditional implementation. The main idea is derived from C#'s dictionary implementtation for hash tables, which uses one array combining all data that should be in linked lists. Our results show that cache friendly implementation of Adjacency list outperforms it in lookup and insertion operations.

Literature review. The effective representation of graph data structures is a challenging problem nowadays. There are a few fundamental simple implementations of graphs [Berge, 1964, 186-187]. One of these is Adjacency list [Cormen et al., 2009, 610-613], which instead of Adjacency Matrix [Cormen et al., 2009, 610-613] requires less memory. Traditional adjacency list structures often lead to a high number of cache misses. This is primarily due to their non-linear memory allocation, which disturbs the efficient utilization of the cache. There are many optimal cache friendly variants of graph representations, but we will discuss data structures which are at most keeping graph's adjacency list structure. There are several representation techniques such as DCRS [King et al., 2016, 61-80] and PCRS [Wheatman et al., 2018, 1-7], which use the CRS (Compressed Sparse Row) technique to effectively represent graph data structure. Another approach is to imp-

lement adjacency list keeping stacks instead of linked lists [Yadav et al., 2013, 1-5]. Although it was more effective than traditional implementation, stacks will be separated from each other in heap memory. Our solution is based on C#'s Dictionary data structure implementation [Microsoft] which is done by keeping two separate arrays, which will collect the graph data in two contentious memories.

Methodology. Our methodology is based on cache friendly data structure design. We modified the adjacency list implementation mechanism to be more cache friendly but kept this structure. Our approach is based on C# programming language's Dictionary data structure design technique, which is modified version of hash table's separate chaining structure. They are used to separate vectors instead of list and vector combination and do mapping mechanism of linked list with vector indexes. By using this technique, we will be modifying graph's adjacency list representation by making him cheche friendly and keeping his structure.

Scientific novelty. In nowadays the importance of high efficiencies algorithms plays the big role in the market products, which are supporting business and giving the computing more efficient. One of the base data structures that is used in big percentage of algorithms theory are graphs and their efficient representation is a challenging problem. We have designed and implemented a new representation for adjacency list data structure for graphs, which is more efficient than traditional representation due to its cache friendly representation. In addition to that this structure keeps the adjacency list's main structure.

Analysis. Let's start with C#'s Dictionary implementation mechanism. C#'s Dictionary is analogous to the traditional hash table Separate Chaining data structure. Instead of keeping pointers to linked lists, it uses two arrays. The first array stores the indexes of the second array's elements plus one. The second array illustrates the linked list mechanism by keeping all data in one contiguous memory. Now let's illustrate the main working mechanism for element insertion in a hash table and its structure. In Figure 1, the main structure of a Dictionary is illustrated. The keys of C and A elements make a collision, and as can be seen in the figure, the element with index 2 in the entries vector points to the 0th element by its _next member. If we add a new element, the sizes of the vectors will be increased, and the new element will first be added to the entries vector. Then, if no collision was made, the new index will be kept in the buckets vector with the index of the entries element plus one. If there is a collision, the new element's _next member will point to the element to which the bucket's index is pointing, and the bucket's index will be changed to the new element's index in the entries vector plus one.

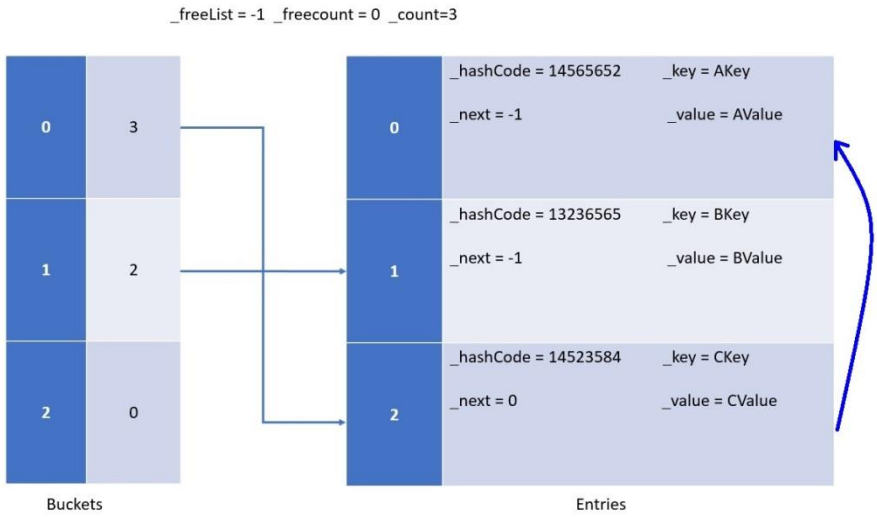

Figure 1. C#'s Dictionary Structure

The same technique will be used in the graph's adjacency list representation. Instead of keeping linked lists for each vertex, we will use two vectors, one for source vertexes and second for destination vertexes. When we will add the new edge, by adding new vertex to source vertex, we will add the new element in destination vector and if source vertex points to another vertexes, just set the new element's next member to index in which is pointing source vertex, and update source vertex as new element's index plus one.

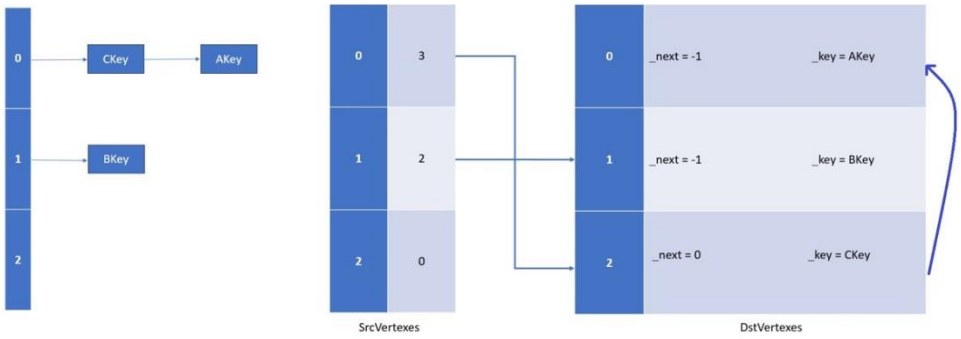

Figure 2. Adjacency List and 3Cache Friendly Adjacency List

In Figure 2 is illustrated traditional adjacency list structure and cache friendly adjacency list structure. Instead of A, B and C keys will be vertex numbers.

We have designed and implemented our proposed graph representation and tested it using Google's benchmark library. The analysis is node on creating and inserting ele-

ments in graph, the comparation is done on simple adjacency implementation and out cache friendly adjacency list's representation.

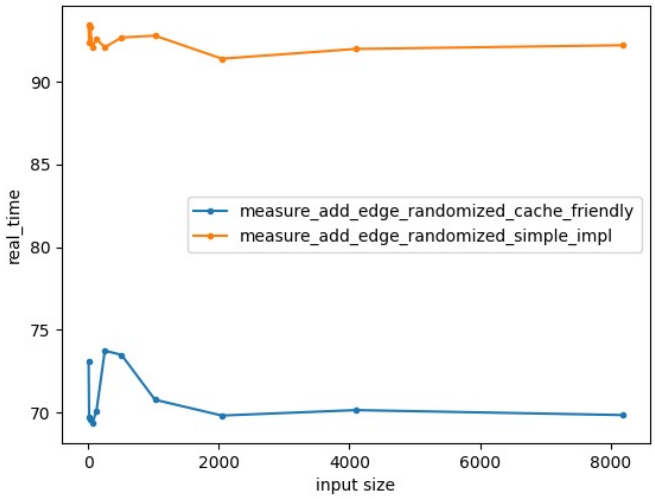

Figure 4. Performance Analisis of addEdge operation

The analysis shows that we outperformed by 29.71 percents.

Our proposed data structure shows significant computational savings. By reducing CPU time and using memory effectively, it reduces energy and consumption which is become to economic savings and keeps the global initiatives for energy efficiency.

Conclusions. In this paper we have discussed graph data structure's adjacency list implementation types and proposed our design to cache friendly adjacency list keeping the adjacency list's structure. Our design is based on C#'s Dictionary implementation mechanism, which tries to replace the linked list with vector with additional indexes for implementing the pointer mechanism in liked list. By storing the data in contiguous memory, we have improved the efficiency of our data structure, making it more economically efficient. The cost-effectivity of our data structure leads to less energy consumption which is critical to nowadays economics and is staying the high priority important aspects for researchers. Future research will focus on discussing more effective mechanisms for graph representation and modifications to graph algorithms based on these representations.

References:

1. Shuai Wang. On the Analysis of Large Integrated Knowledge Graphs for Economics, Banking and Finance, 2022, pp 1-6.
2. Anshu Yadav, Aruna Bhat, Rajni Jindal. Stack Implementation of Adjacency List for Representation of Graphs, 2013, pp 1-5.
3. Claude Berge, The Theory of Graphs and it's Applications, Wiley, 1964, pp 186-187.
4. James King, Thomas Gilray, Robert M Kirby, and Matthew Might. Dynamic sparse-matrix allocation on gpus. In International Conference on High Performance Computing, Springer, 2016, pages 61–80.
5. Brian Wheatman, Helen Xu. Packed Compressed Sparse Row: A Dynamic Graph Representation 2018, pp 1-7
6. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. Introduction to Algorithms. MIT Press, 2009, pp 610-613.
7. (2023) C#'s Dictionary Data Structure. [online] available: Dictionary<TKey,TValue> Class (System.Collections.Generic) | Microsoft Learn

Arman MARTIROSYAN, Rafayel KHACHATRYAN, Rafayel VEZIRYAN
Economically efficient cache friendly adjacency list for graph processing algorithms
Key words: data structures, computational efficiency, cache optimization, adjacency list, graph

Graphs are a fundamental part of computer science and various scientific fields. They are used in numerous applications, from social networks to electronic design automation. Efficient graph representation is important for modern algorithms, especially in economical contexts, where resource-efficient implementations play a significant role by reducing the resource consumption. This is essential in environments where computing resources are a high priority. There are many representations of graphs as data structure. One of these types is adjacency list: it is collection of linked lists which shows the connection of source and destination vertices. Adjacency lists are more memory-efficient than adjacency matrices because they only allocate memory for vertices with at least one edge in the graph. However, traditional implementations of adjacency lists do not fully use the features provided by modern CPUs, such as the cache line. Our design of Adjacency List is cache friendly and is keeping the main structure of traditional implementation. The main idea is derived from C#'s dictionary implementation for hash tables, which uses one array combining all data that should be in linked lists. Our results show that cache friendly implementation of Adjacency list outperforms it in lookup and insertion operations.