Gebze Technical University

Computer Engineering Faculty

Final Project Report

Virtual Memory Systems and Paging

Student: Muhammed Okumuş

No: 151044017

Contents

1 Part 1: Overview

This report focus on explaining the page table structure design and references to its implementations in part 2 of the project. Also comments on part 3 are included.

2 Page Table Structure

The page tables consists of entries that defines a frame and it's location(physical and virtual). This entries help retrieving information, replacing page frames and deciding which frame to replace.

```
typedef struct{
    unsigned int addr_virtual;          // VM address space
    int addr_physical;                  // RAM address space
    int modified;                       // Modified bit
    int present;                        // Present bit
    int referenced;                     // Referenced bit
    int owner;                          // Owner process ID
    int age;                            // Age bit
    struct timespec reference_time;
    struct timespec load_time;
}Entry;              // main.c line 28 at Part_2 folder
```

Physical and **virtual** address fields are used for mapping a frame to the physical memory. Virtual address stays unchanged as program executes. Physical address is likely to change during execution as a result of page faults. If the page is not mapped to the physical memory, addr_physical field will -1. There are 2 possible scenarios on how to read this frame in to physical memory. These scenarios are discussed in the get/set method section.

The **modified** bit(a.k.a dirty bit) is to keep track of a frame in case its changed and requires write back. **Present** bit is only set if the page is in physical memory. **Referenced** bit is set if the frame is referenced in the last cycle. Cycle simulation is discussed later in the report. **Owner** is an optional field to use in case of local allocation is requested. **Age** field is an also an optional field added for WSClock algortithm.

Reference time and **load time** are simulation fields. They are utilized in some of the page replacement algorithms(**LRU** and **FIFO**). The page table is made of multiple entries as it's defined below:

```
typedef struct{
    int page_size;
    Entry* page_table;
}VirtualMemory;      // main.c line 43 at Part_2 folder
```

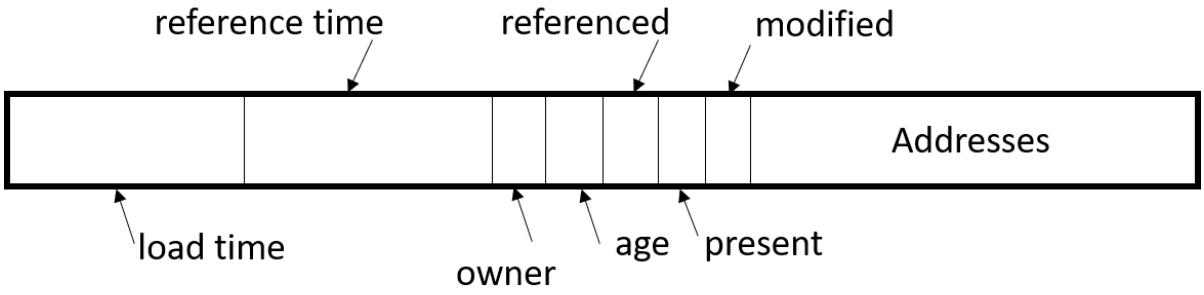


Figure 1: *A table entry, specifically designed for a simulation without hardware support.*

3 Function Prototypes

```
// Misc Functions
void print_usage();
int compare_time(struct timespec t1, struct timespec t2);
Stats* whos_stats(char *tName);
void print_stats(Stats s);

// Debug Functions
void print_entry(Entry e);

// Error Checking Functions
int pr_validity(char* type);
int ap_validity(char* type);

// Integer Array Utility Functions
void print_array(int* arr, int n);

// VM Functions
void initilize_vm();
void print_pt();
int to_addr_space(unsigned int i);
int find_free_addr();

// Get/Set Functions
void set(unsigned int index, int value, char * tName);
int get(unsigned int index, char * tName);
```

```
// Page Replacement Functions
int algorithm(int owner);
int NRU(int owner);
int FIFO(int owner);
int SC(int owner);
int LRU(int owner);
int WSClock(int owner);

// Clock Interrupt Routines
void reset_r_bit();
void apply_aging();

// Sorting
void print_disk(int s, int e);
int is_sorted(int s, int e);
void bubble_sort(int s, int e, char* c);
void merge(int start, int mid, int end, char* c);
void merge_sort(int left, int right, char* c);
void quick_sort(int low, int high, char* c);
int partition(int low, int high, char* c);
void swap(int i, int j, char* c);
void index_sort(int s, int e, char* c);

// Threads
void *thread_bubble_sort(void *arg);
void *thread_merge_sort(void *arg);
void *thread_quick_sort(void *arg);
void *thread_index_sort(void *arg);
void *thread_clock_interrupt(void *arg);
```

4 Get/Set Functions

These functions abstracts memory access. They are similar in most part. They access the disk file using fseek, fwrite and fread. The disk file is binary file for easier offset adjustment.

- Entry field updates, like referenced, present etc.
- Handle page faults.
- Write back if necessary.
- Simulation statistics update.

4.1 Get Function Pseudo-code

Below is the pseudo-code with code line reference from main.c file in part 2 folder.

1. Lock access mutex. (Shared with set function)[411]
2. Check if index is out of range, exit program if so.[416]
3. Get statistics pointer to the caller, update accordingly.[418]
4. Calculate frame address which request index falls in.[423]
5. If it's present in memory, simply return it.[433]
6. If it's not in memory(page fault), first check if there available space in memory, if so place it there.[442]
7. If there is no space in memory, run the page replacement algorithm. Perform write back if necessary and place requested frame into the memory.[464]
8. Unlock access mutex.[510]

4.2 Set Function

Set function is very similar to get function in sense of handling write backs, calling page replacement algorithms etc. They differ when calculating statics and modifying page entries. For example, a frame can only be set modified in set function. Also the statistic they calculate differs, disk page reads can only happen in get function while disk page writes can only happen in set function. In addition to all this, the set function performs a write on the inputted index.

5 Page Replacement Functions

5.1 Not Recently Used

This algorithm utilizes both **referenced** and **modified** bits to decide which pages to replace. Each entry falls into one class like mentioned in the text book. First it search for best case, not referenced and not modified, and goes until the worst case(modified and referenced). Procedure stops as soon as finds a suitable entry.[line 641]

5.2 FIFO

Here instead of using a linked to create and actual FIFO structure, time stamps are utilized. First a dummy time stamp is created at maximum time possible using the defines from limit.h library. Then it simply loop trough present entries and records the the one with the earliest **load time** and mark it for replacement for the caller(get or set function).[line 704]

5.3 Second Chance

This is an improved version of the FIFO algorithm. In reference to comparing the **load time** of the frames, it also check the **referenced** bit. This way an old page used frequent is prevented from being discarded.[line 732]

5.4 Least Recently Used

LRU makes use of the **reference time** field. In theory pages are represent like linked list structure to act like FIFO to queue reference times. In the simulation, linked list structure is not required since we have time stamps and we can find the oldest reference easily.[line 792]

5.5 WSClock

Not implemented. Had hard time with simulating interrupts.

6 Allocation Policies

Global allocation is easier to implement and faster to replace pages compared to local allocation. While local allocation lacks what global allocation provides, it offers safety. But local allocation yields better results in tests in part 3, it's mentioned later in the report.

In the simulation, threads are created for each sorting job. Initially each threads tries to grab a piece of the physical memory. While this is fine for global allocation, local allocation will need extra logic to be realized. For this reason each entry has an **owner field**. This way we can pass the owner ID to the page replacement algorithm and it will try to allocate a frame that which already assigned to the same owner. If it fails to allocate a frame, might happen if its thread doesn't get a chance to execute, it will fall back on global allocation.

I choose to go with this design since it will clearly result in less page faults and it will more robust than just relaying on local allocation since we do not worry about safety in this simulation.

7 Backing Store(Swap Space)

The simulation doesn't require backing store. Maximum amount of page entries are calculated and allocated at the virtual memory initialization.[line 248]

8 Simulating Interrupts

To simulate interrupts, a separate thread is created[line 295]. This thread only runs if the chosen page replacement algorithm is either NRU or WSClock. It shares the same mutex with get and set functions and utilize POSIX nanosleep(2) to simulate interrupts. For NRU it clears referenced bits[line 885] at each 40 ms interval[line 871] and for WSClock it applies aging and clears reference bits[line 891].

9 Finding Optimal Page Size

In part 3 main.c file, a testing program is coded. There are 4 page replacement algorithms(NRU, LRU, SC and FIFO) and 2 allocation policies(local and global) used in the tests. Program outputs all the statistics required for part 2 for all sorting algorithms and PR-AP combinations. Program output is directed to a file in terminal and the data is organized in excel for analyzing. Graphs can be viewed in full size in the excel file inside the project folder. Raw data can be viewed under Part 3 folder, test1.txt.

9.1 A Note About Tests

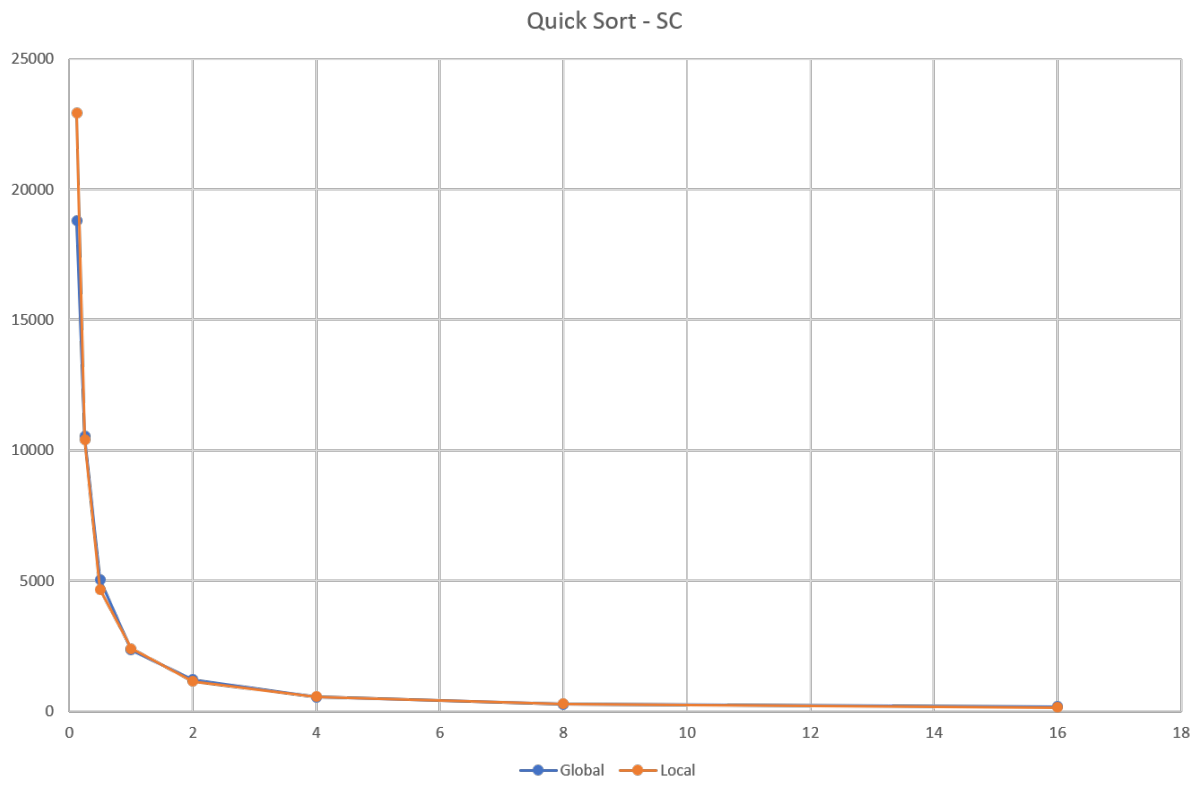
Over all 9 tests made for each PR-AP combinations. Each test step frame size is factored by 2 and virtual memory and ram is adjusted to stay 64KB and 1024KB respectively. Test 1 starts from 0.06KB and goes up to 16KB at test 9. Test 1 was disregarded because a bug resulted from such small frame size, resulting in unsorted data. But other tests(2 to 9) are completely valid and all the sorting functions completed successfully. In this version of the program no user input is taken and page table doesn't get printed to make test results more readable. Also bubble sorts operation area is limited to a smaller range so that it wont take forever. This caused bubble sort algorithm to result in 0 misses in most tests, because **initial memory load is not considered a miss in the design**. In the bigger frame sizes bubble sort can fit all it's required memory into a single page resulting 0 misses. Meanwhile other algorithms works on relatively huge data, about 250.000 data points for each sort thread.

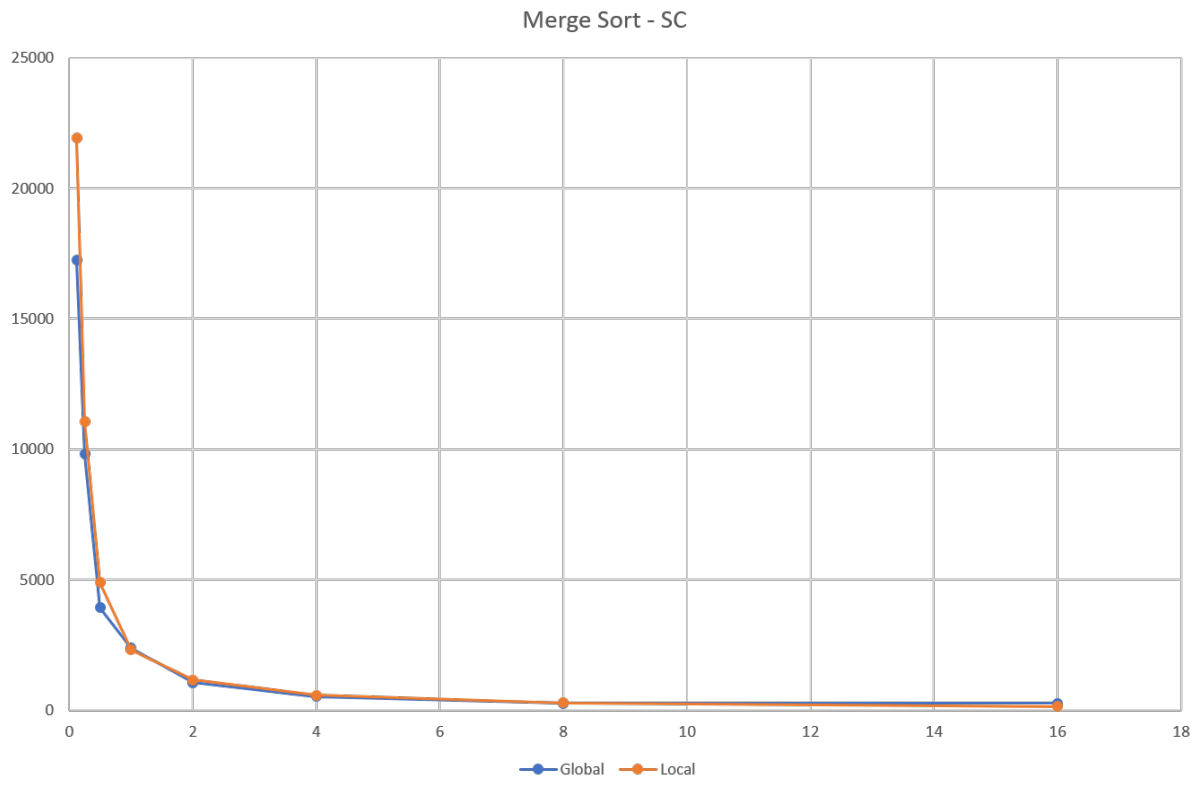
9.2 Best Combinations For Each Sort Algorithm

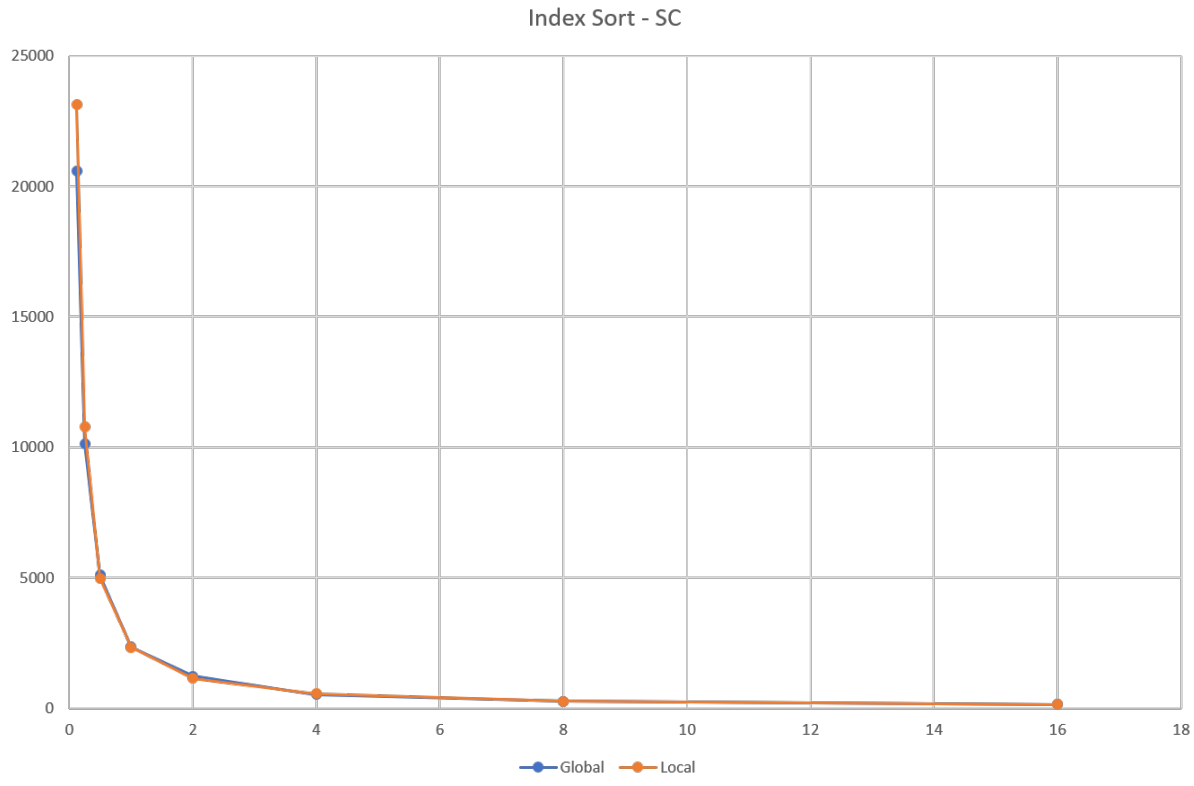
Looking at the data in excel sheet, we can clearly see that the local allocation policy for each type of PR algorithm yield consistently better results. This is because global policy can replace other processes frequently used page in some cases.

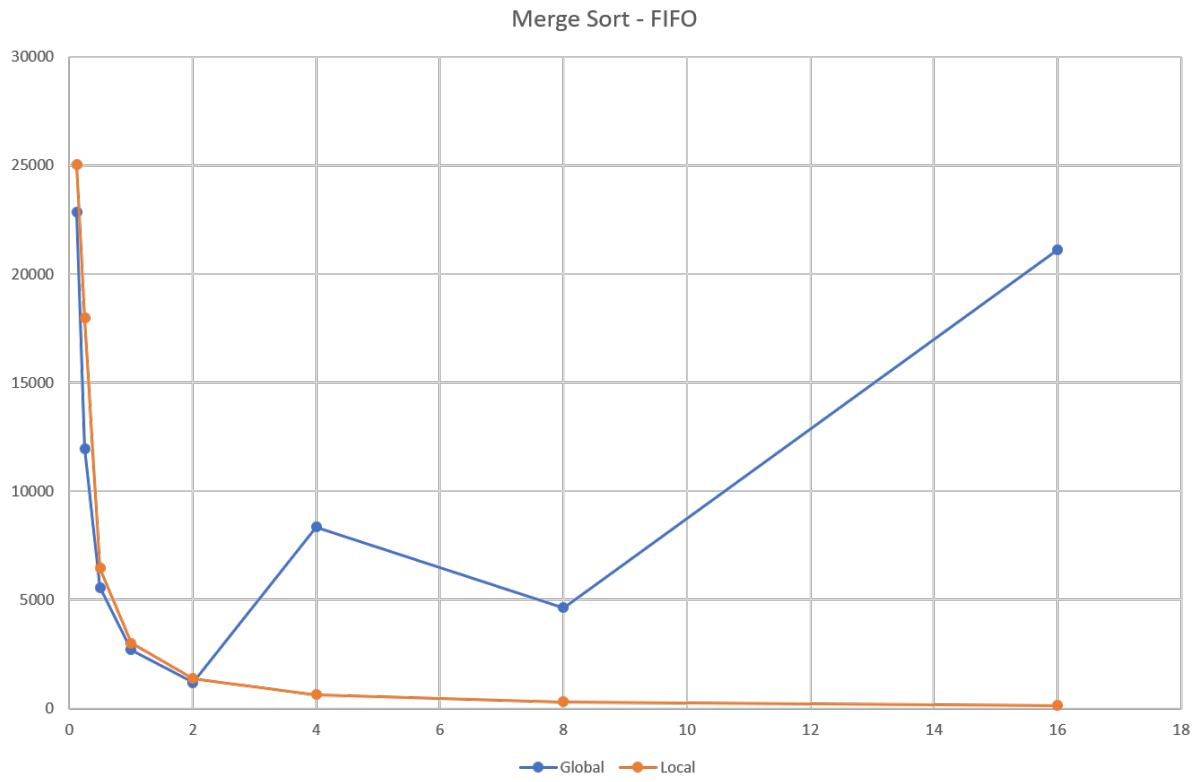
9.2.1 Winner Page Replacement Type for 3 Sorting Methods

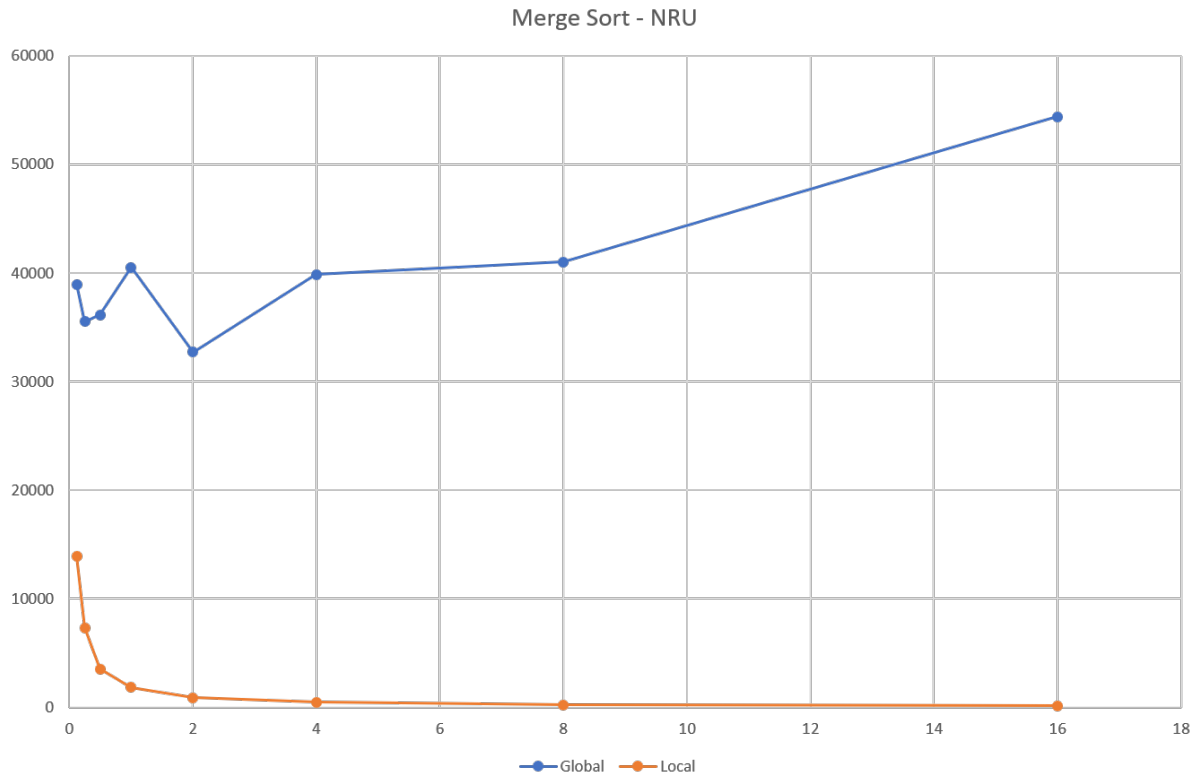
Second chance algorithm with both local and global allocation works great for merge, quick and index sorts. Mean while other algorithms are very close with local policy equipped, SC algorithm resulted in less misses in general.

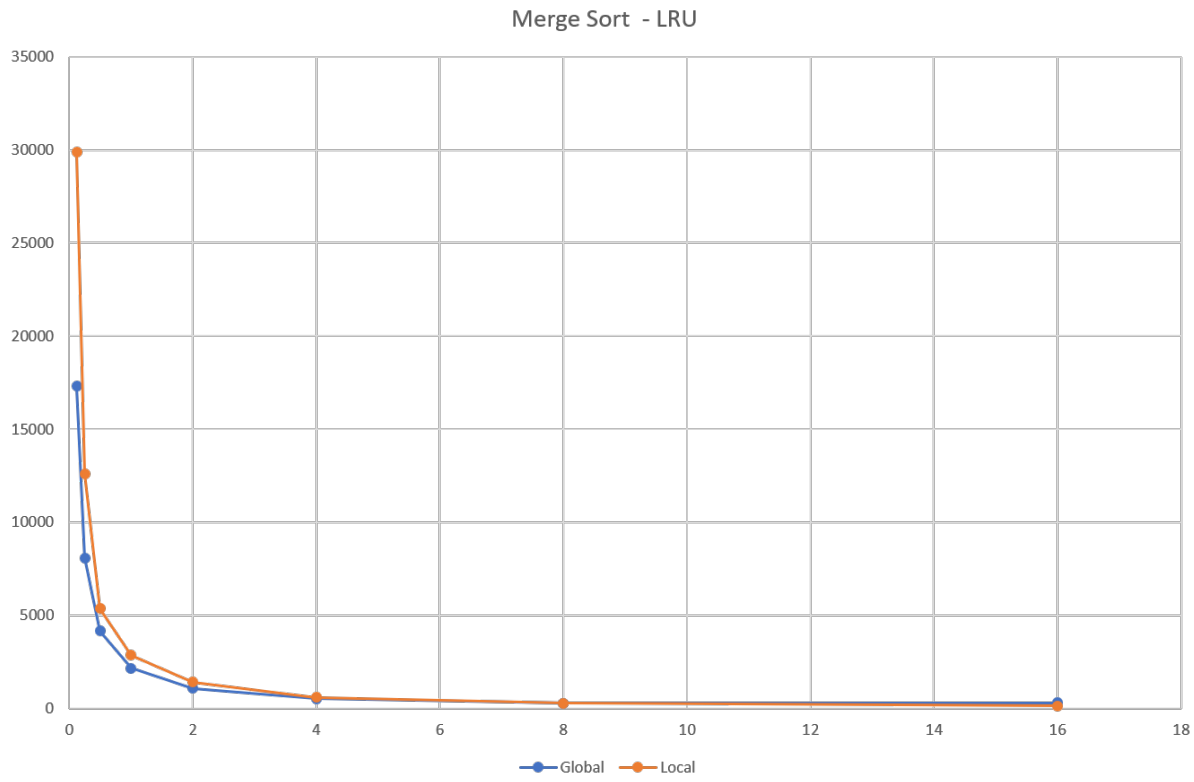


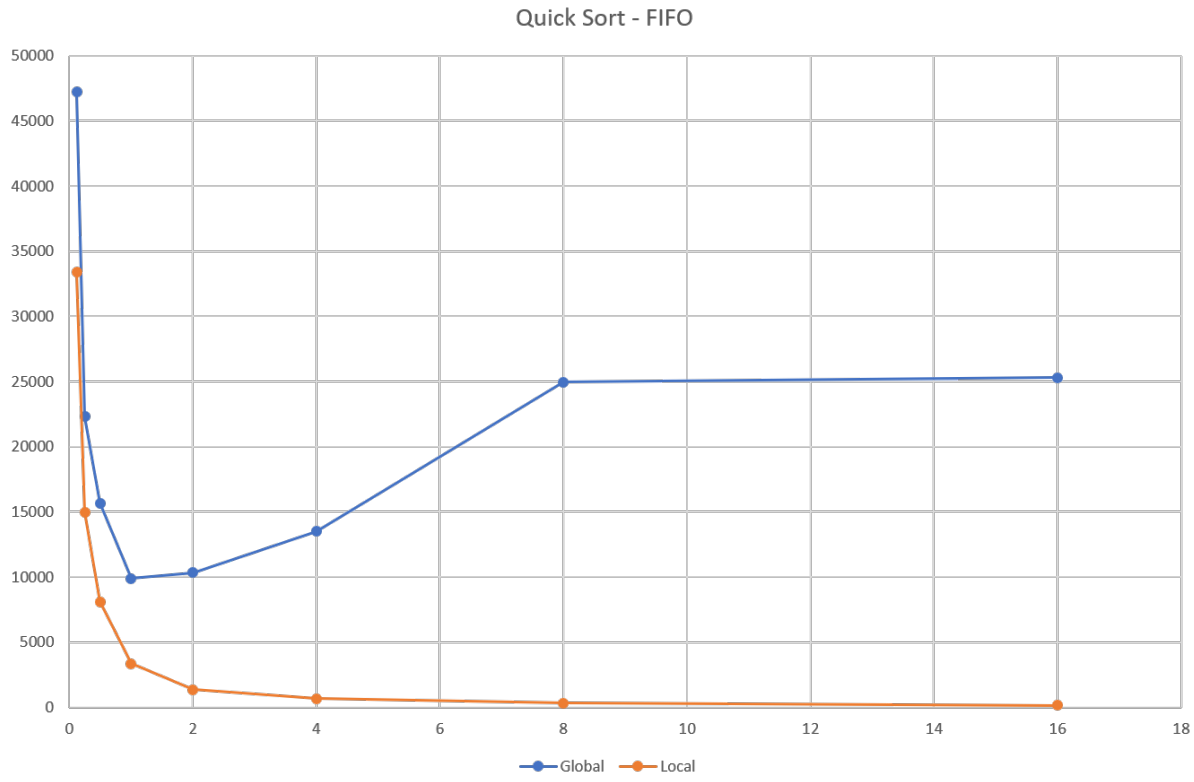

Merge Sort - SC
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16
18
Global
Local

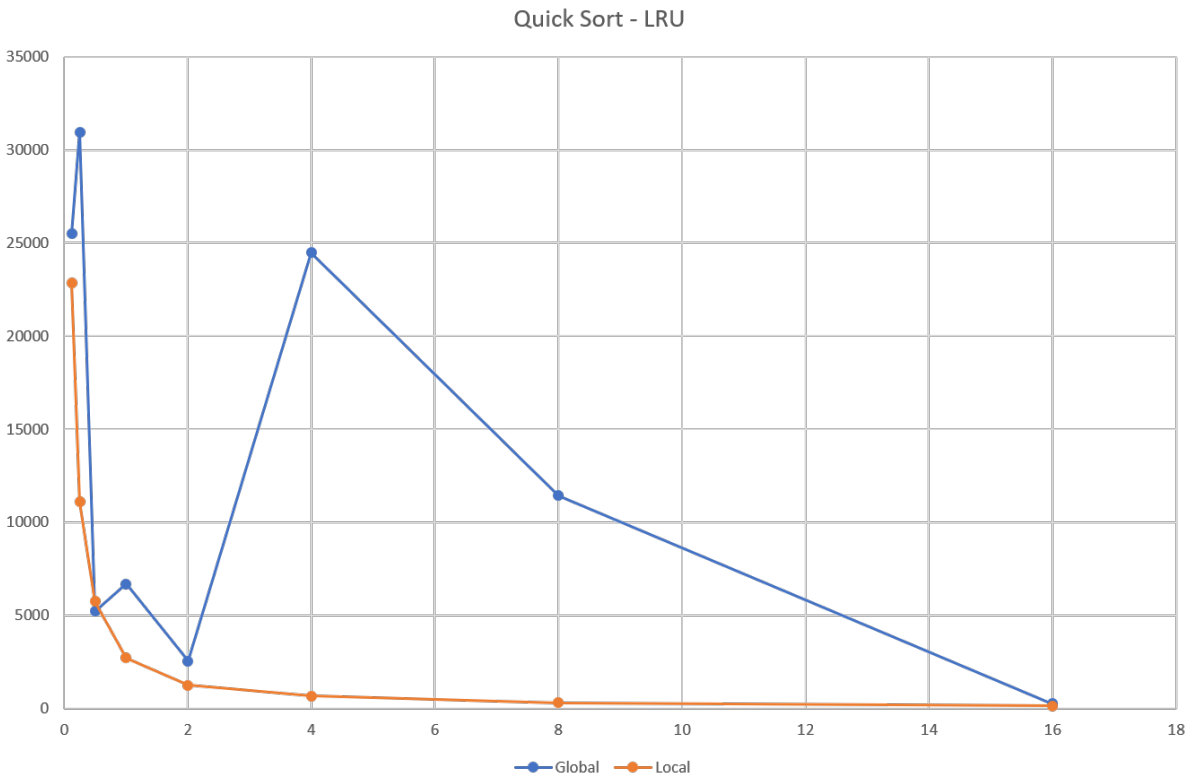

Index Sort - SC
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16
18
Global
Local

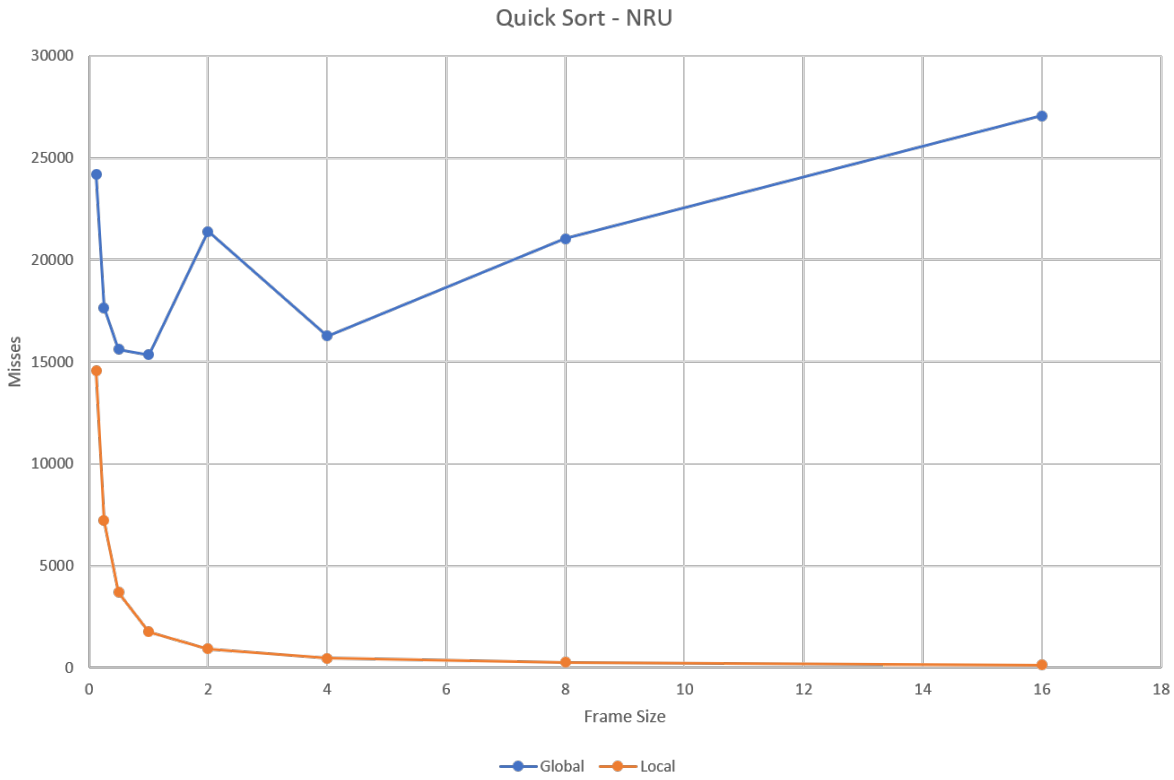

Merge Sort - FIFO
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16
18
Global
Local

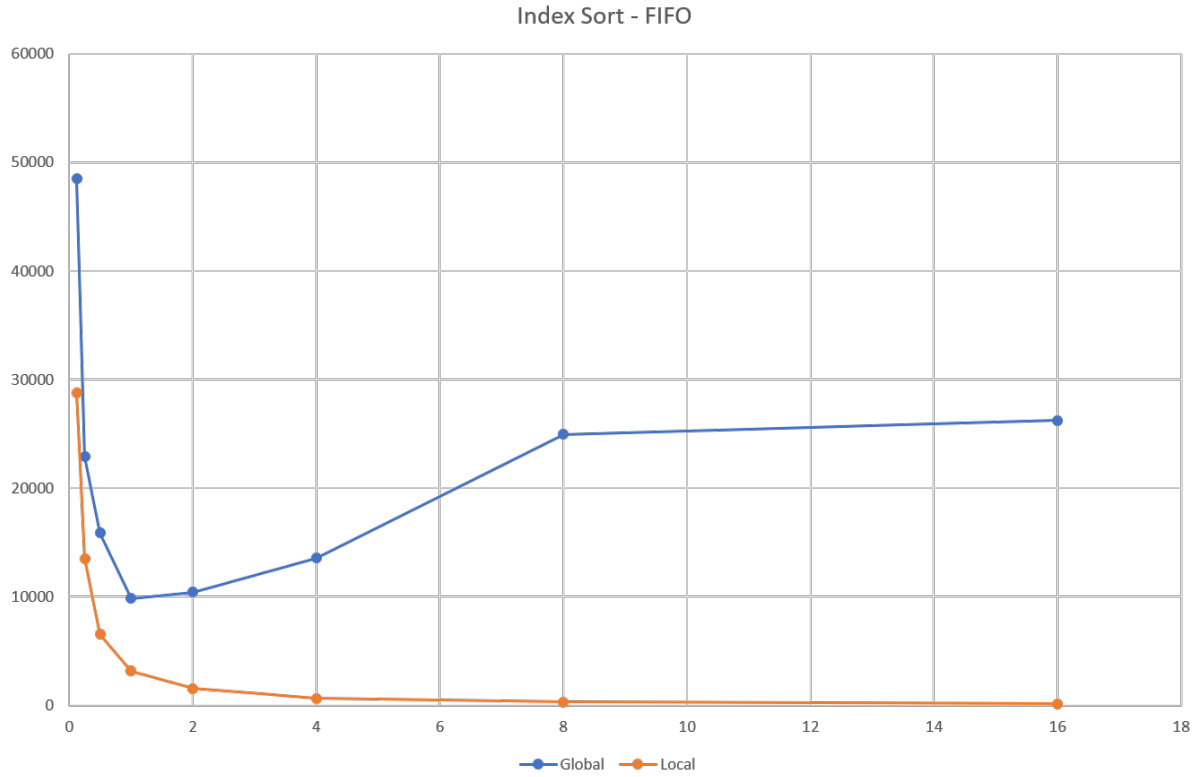

Merge Sort - NRU
60000
50000
40000
30000
20000
10000
0
0
2
4
6
8
10
12
14
16
18
Global
Local

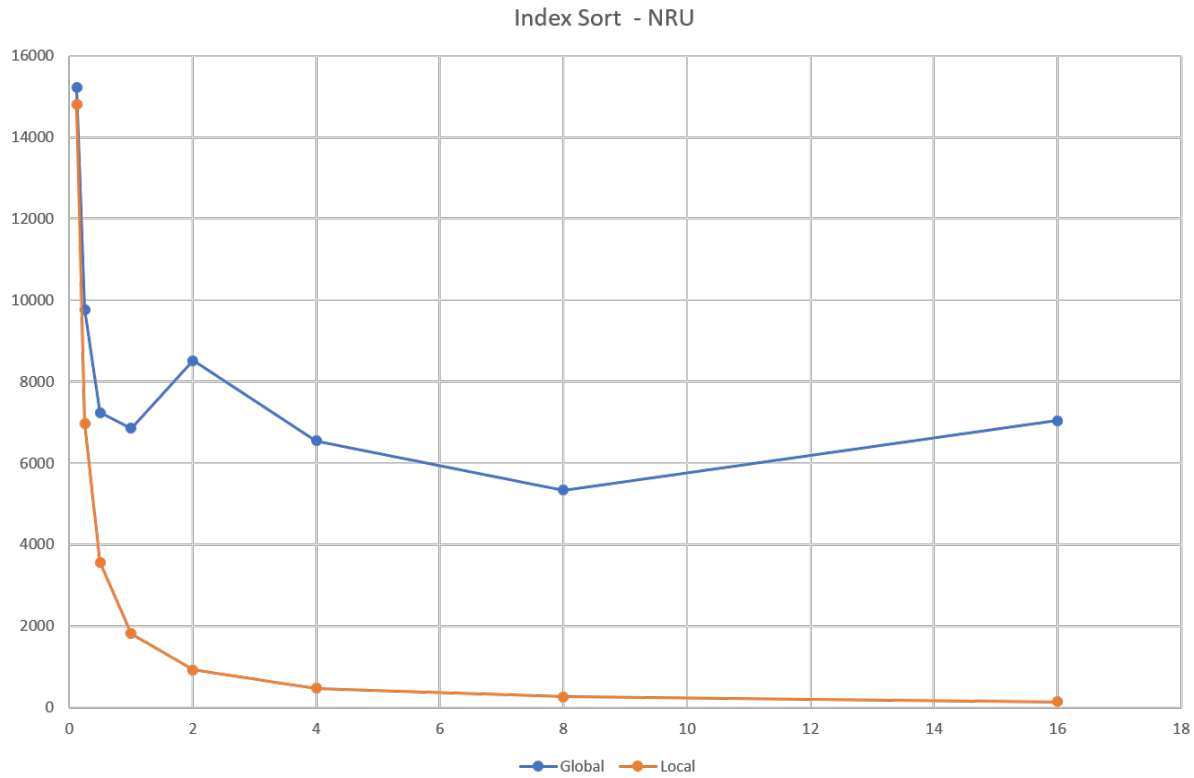

Merge Sort - LRU
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16
18
Global
Local

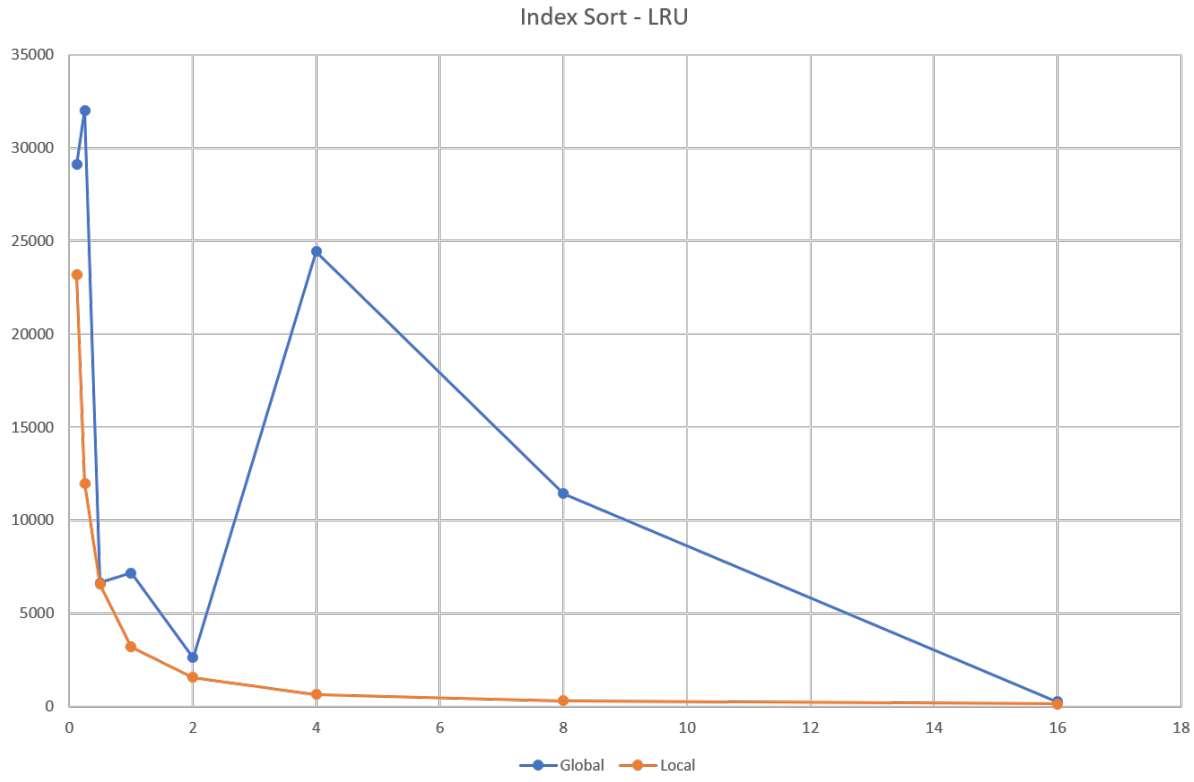

Quick Sort - FIFO
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16
18
Global
Local


Quick Sort - LRU
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16
18
Global
Local


Quick Sort - NRU
30000
25000
20000
15000
10000
5000
0
Misses
0
2
4
6
8
10
12
14
16
18
Frame Size
Global
Local


Index Sort - FIFO
60000
50000
40000
30000
20000
10000
0
0
2
4
6
8
10
12
14
16
18
Global
Local


Index Sort - NRU
16000
14000
12000
10000
8000
6000
4000
2000
0
0
2
4
6
8
10
12
14
16
18
Global
Local


Index Sort - LRU
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16
18
Global
Local