Project 5 Report

J. Patrick Lacher
jlacher1@nd.edu

James Marvin
jmarvin1@nd.edu

Overview

The purpose of these experiment was to compare the properties and performance of different page replacement algorithms when used with a series of sample programs that access memory in different patterns. Gathering the test data and plotting it allowed us to make the following observations detailed in the observation section below.

Experimental Setup

We ran our programs using a bash script that called the command

```
./virtmem 100 $x <program> <algorithm>
```

and outputted the results to a text file. We then took the data from the text file and created the graphs shown in Appendix 1. These experiments was run in the linux environment on the student machines.

Our Algorithm

Our algorithm, which is similar to an LRU, could be described as *Least Often Faulted*. We implemented a counter for each page in the page table that was incremented every time a fault occurred for any given page. When the frame was filled and a fault occurred, the page fault handler would return the page in the frame with the lowest fault count as the page to be overwritten by the incoming page.

Observations

Looking at the graph to the right (Fig. 1), which is characteristic of all of our data, rand (blue) and fifo (red) had very similar performances in these experiments, both in terms of number of faults and in the relative frequency of reads and writes (Fig. 2-3). Our custom algorithm (green) began with significantly more faults, but began to beat both fifo and rand after frame size increased over 40. The custom algorithm also had a much lower frequency of faults resulting in a disk write than the other two algorithms. In general the number of reads and writes went down as frames went up because less pages had to be overwritten.

Appendix 1:

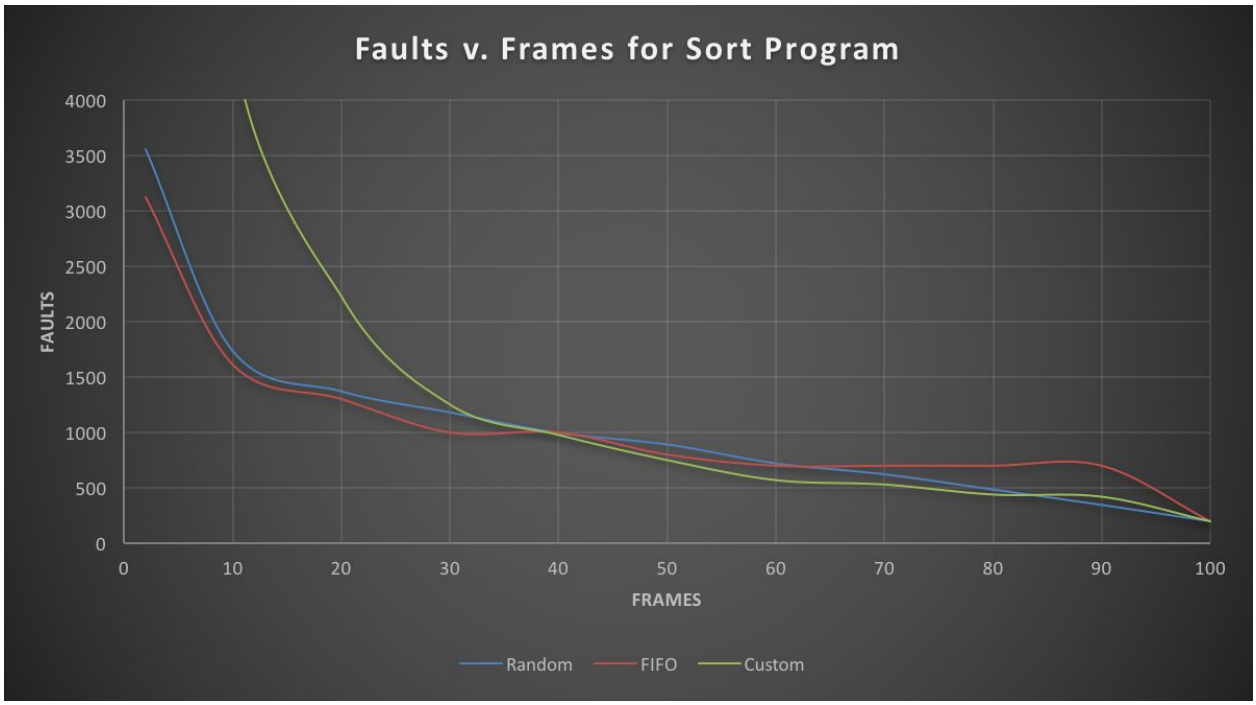

Fig. 1

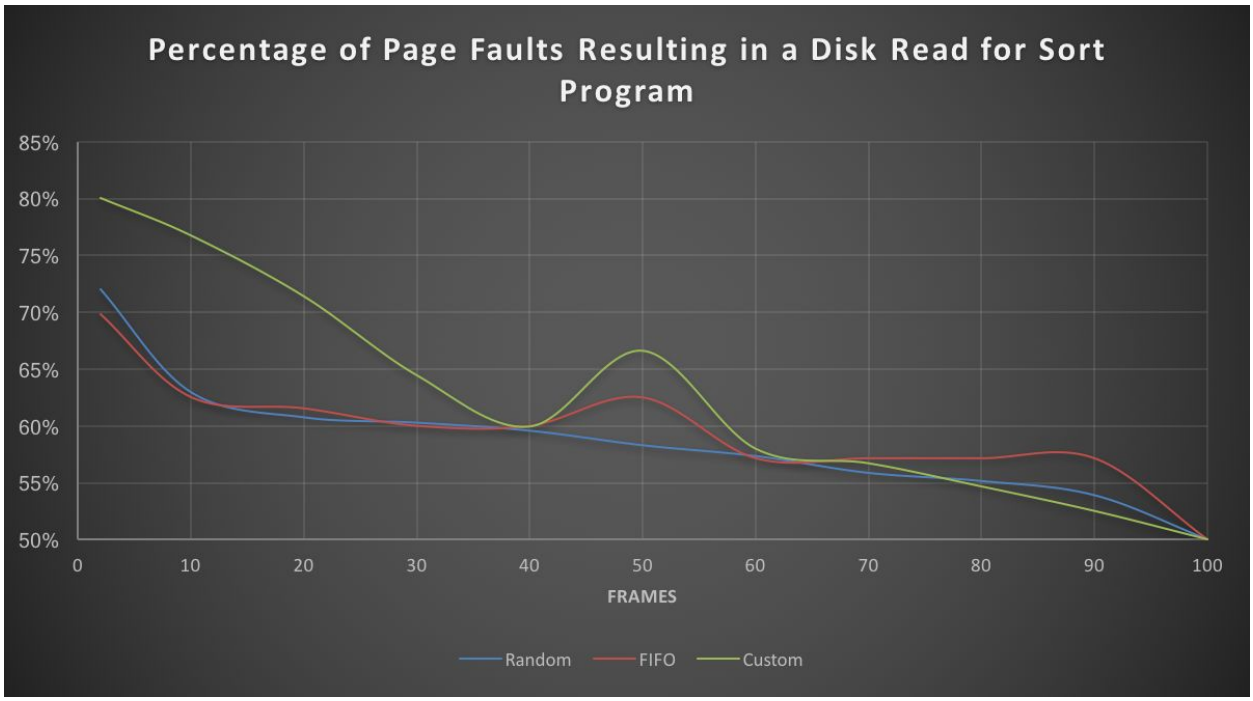

Fig. 2

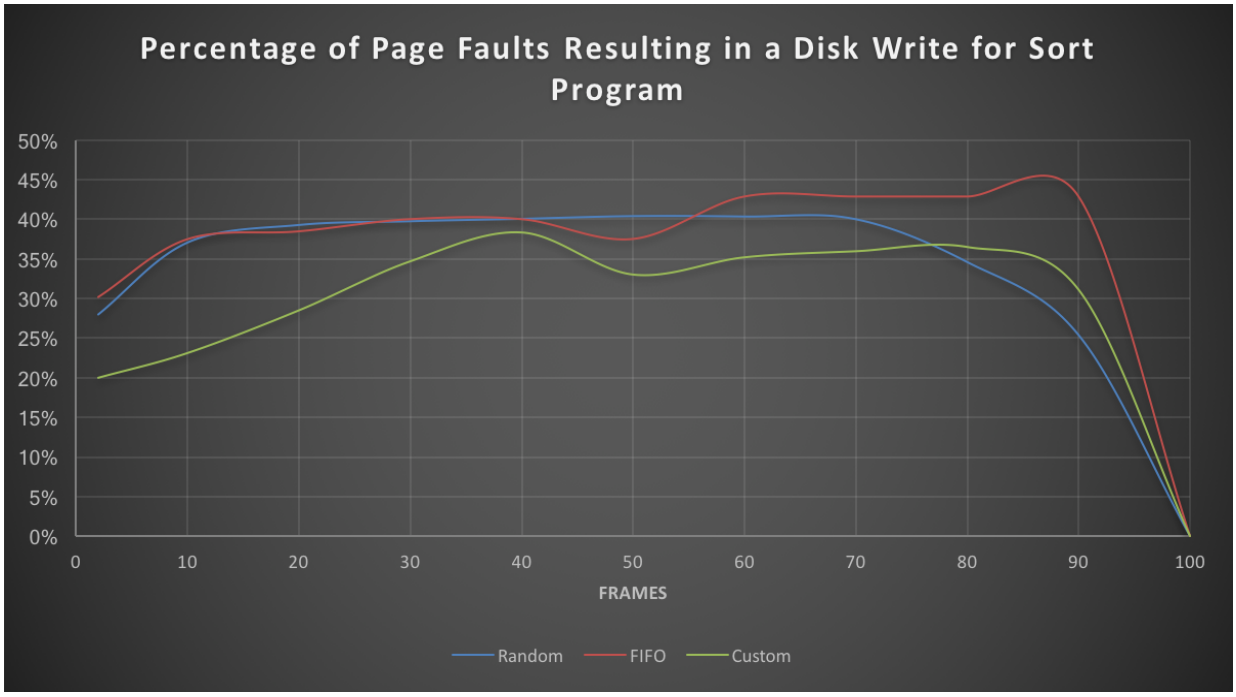
Percentage of Page Faults Resulting in a Disk Write for Sort Program
50%
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
0
10
20
30
40
50
60
70
80
90
100
FRAMES
Random
FIFO
Custom

Fig. 3

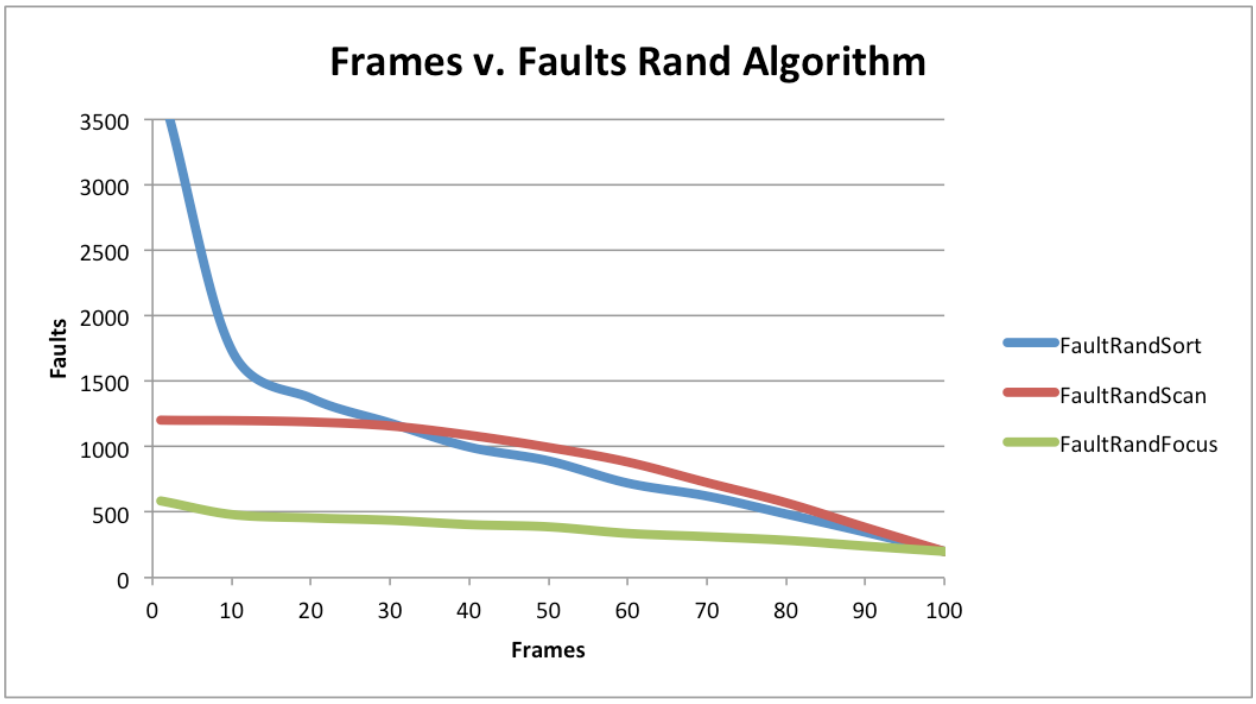
Frames v. Faults Rand Algorithm
3500
3000
2500
2000
1500
1000
500
0
Faults
0
10
20
30
40
50
60
70
80
90
100
Frames
FaultRandSort
FaultRandScan
FaultRandFocus

Fig. 4

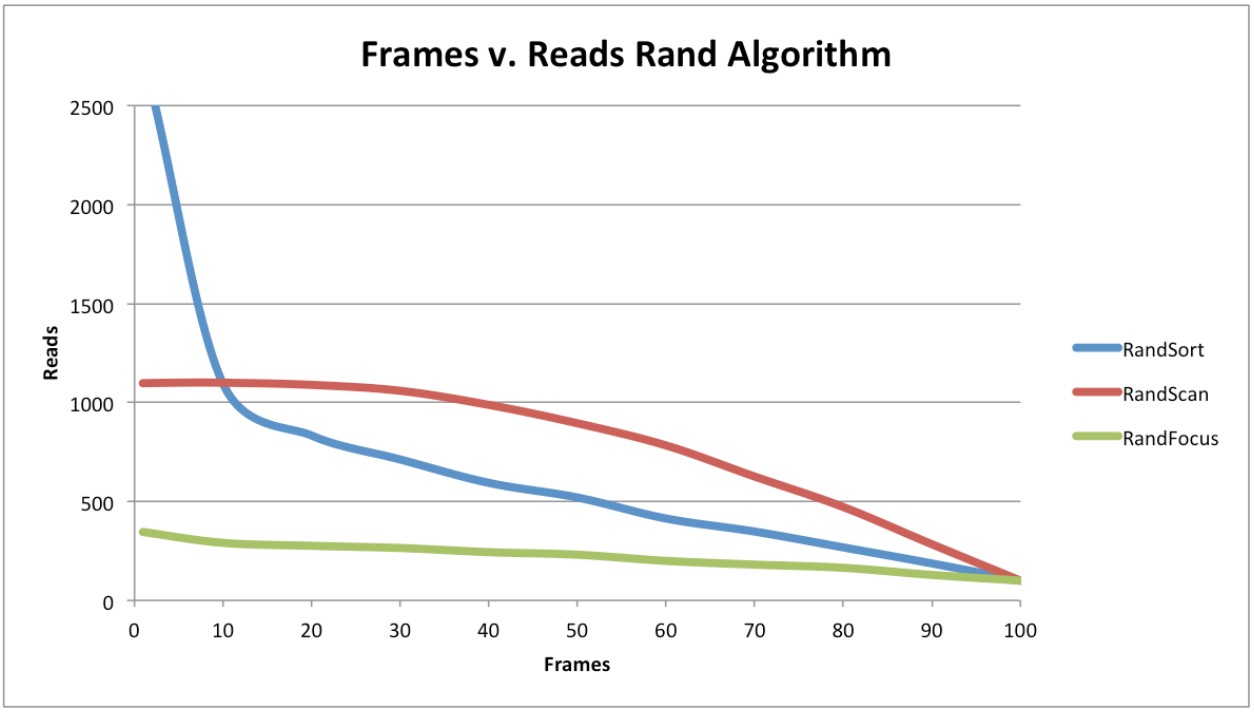
Frames v. Reads Rand Algorithm
Reads
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
80
90
100
Frames
RandSort
RandScan
RandFocus

Fig. 5

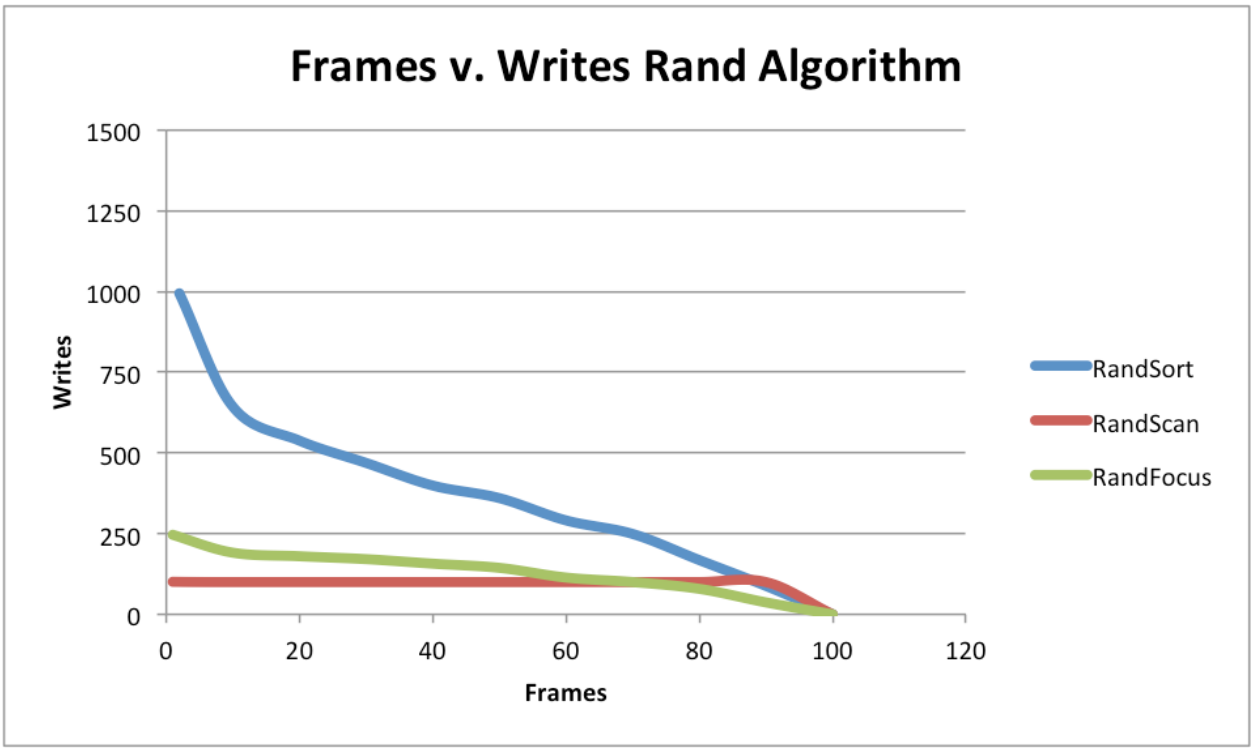
Frames v. Writes Rand Algorithm
Writes
1500
1250
1000
750
500
250
0
0
20
40
60
80
100
120
Frames
RandSort
RandScan
RandFocus

Fig. 6

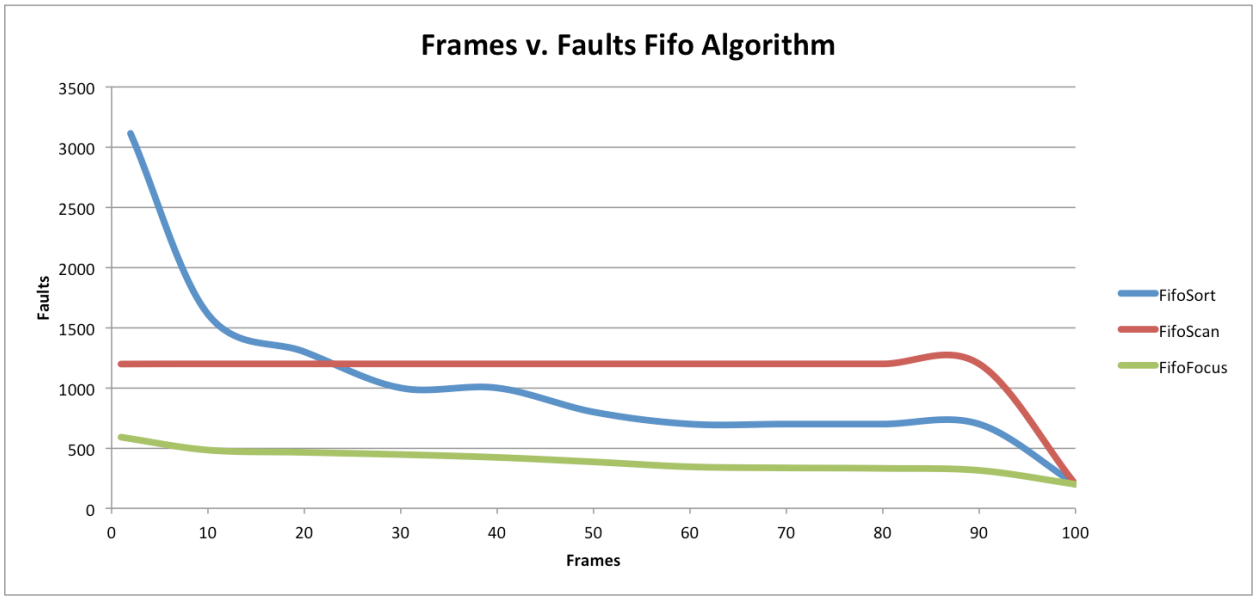
Frames v. Faults Fifo Algorithm
Faults
3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
80
90
100
Frames
FifoSort
FifoScan
FifoFocus

Fig. 7

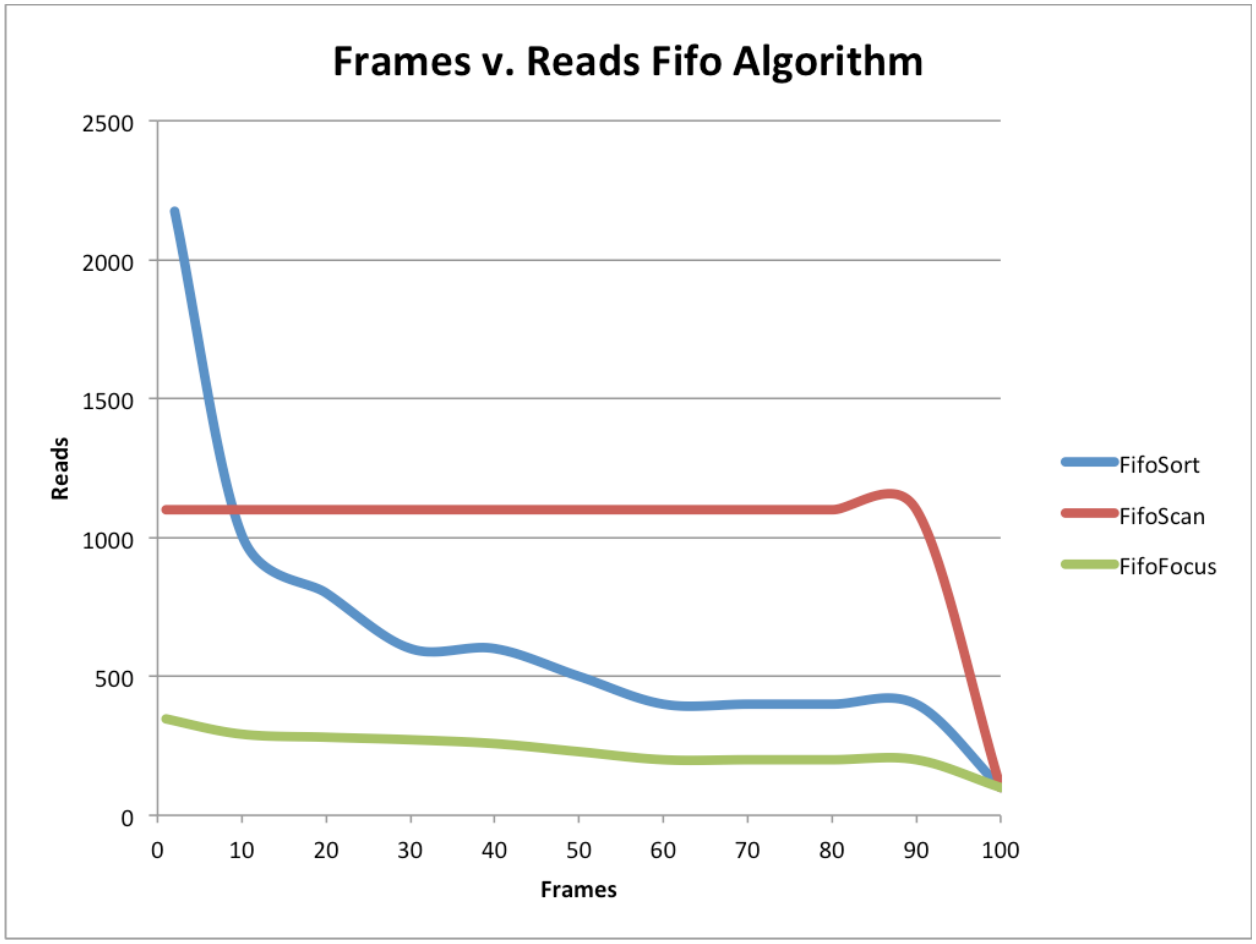
Frames v. Reads Fifo Algorithm
Reads
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
80
90
100
Frames
FifoSort
FifoScan
FifoFocus

Fig. 8

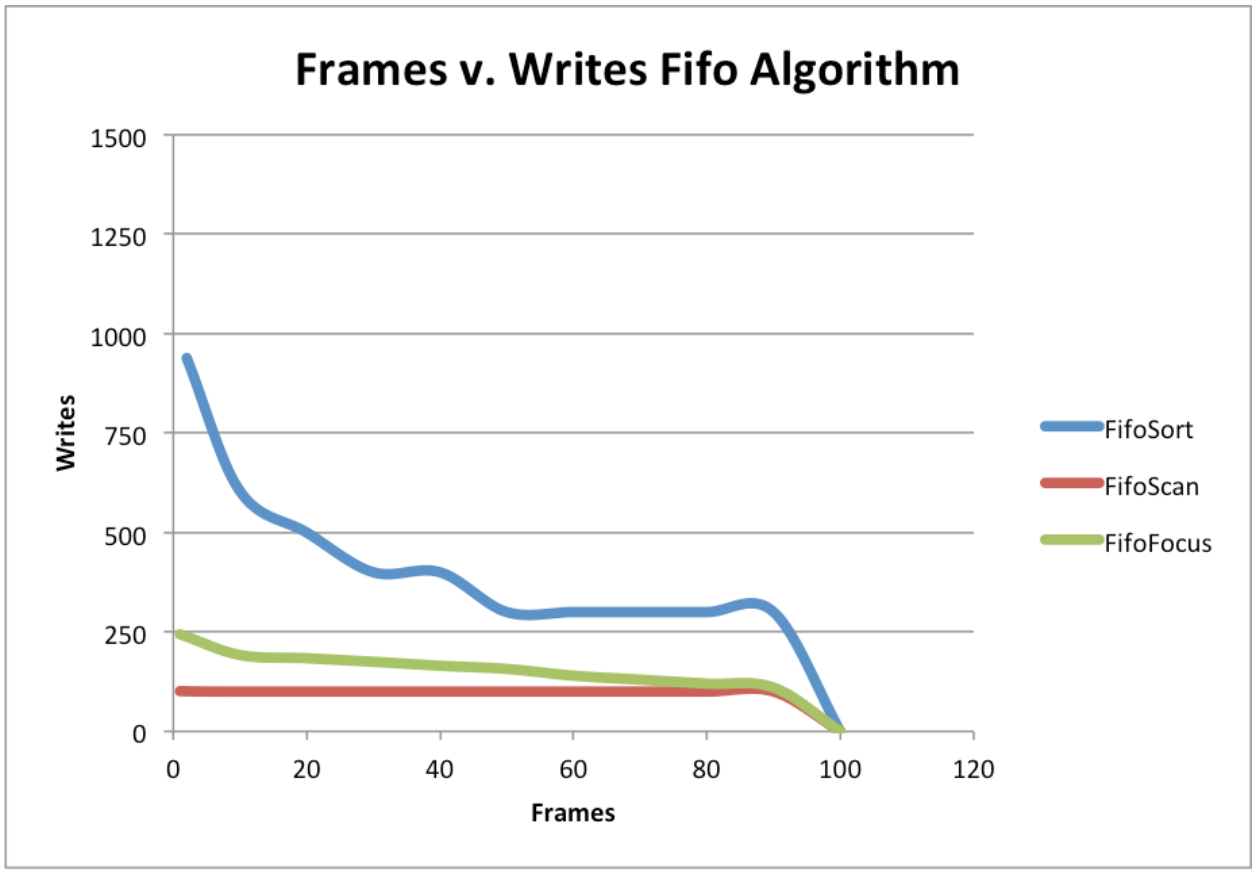
Frames v. Writes Fifo Algorithm
Writes
1500
1250
1000
750
500
250
0
0
20
40
60
80
100
120
Frames
FifoSort
FifoScan
FifoFocus

Fig. 9

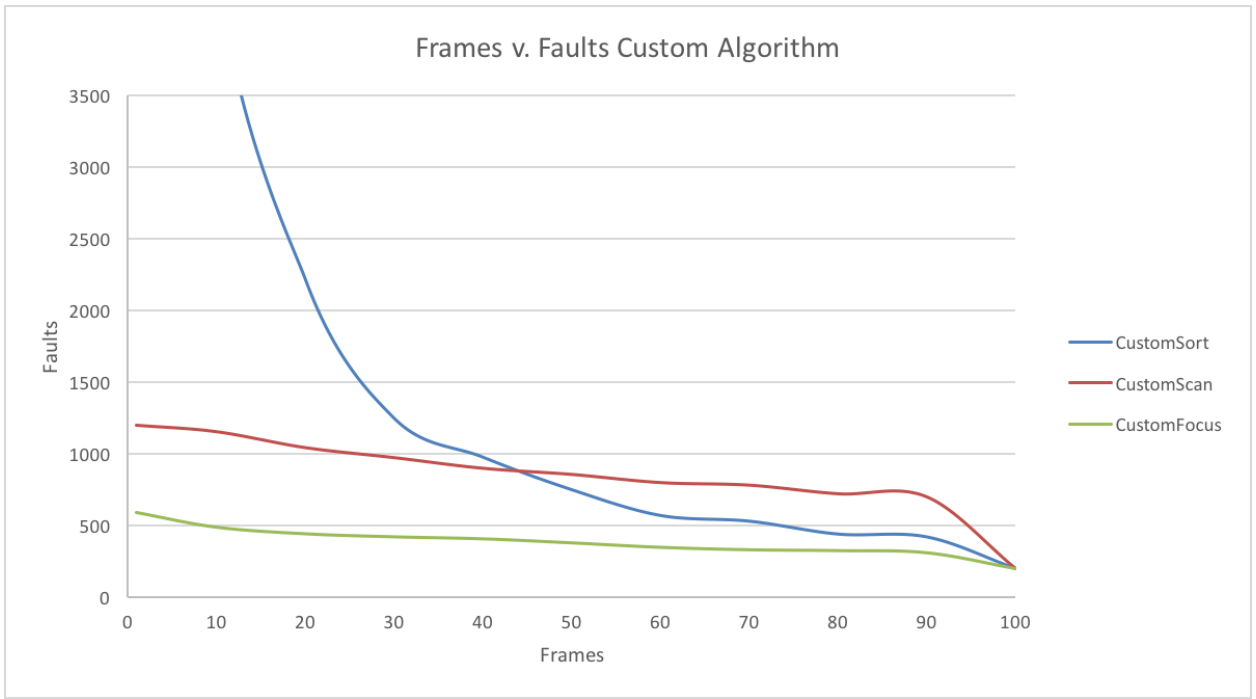
Frames v. Faults Custom Algorithm
Faults
3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
80
90
100
Frames
CustomSort
CustomScan
CustomFocus

Fig. 10

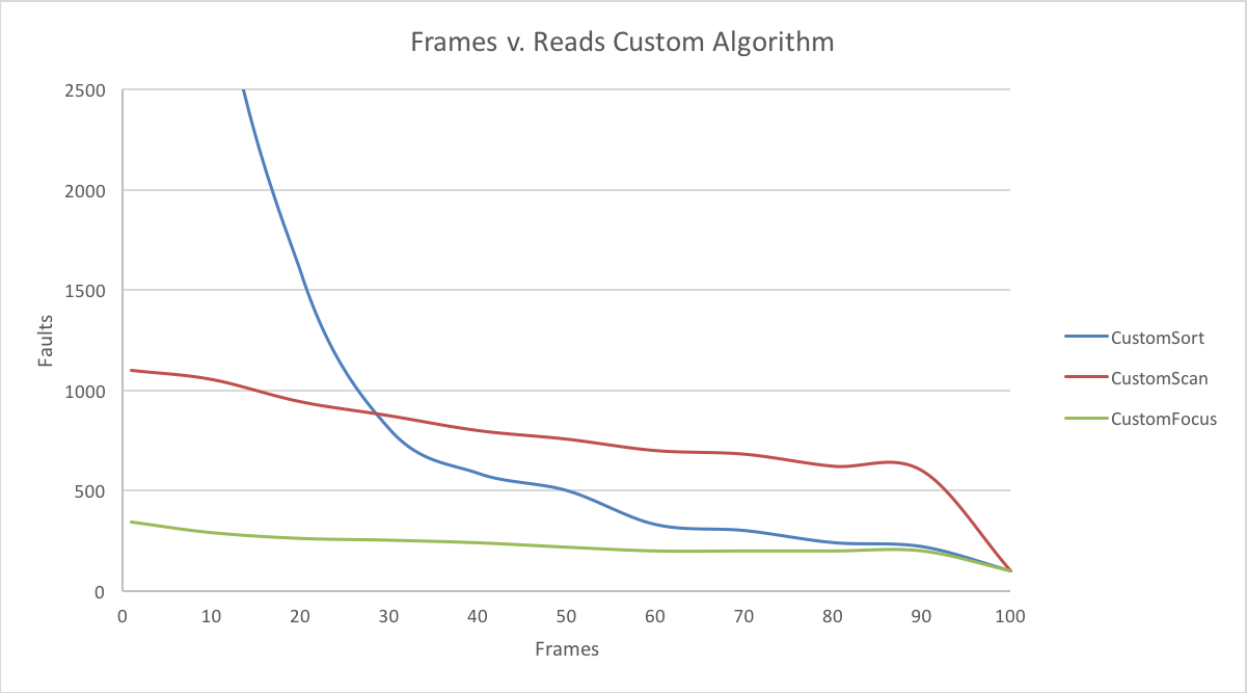
Frames v. Reads Custom Algorithm
Faults
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
80
90
100
Frames
CustomSort
CustomScan
CustomFocus

Fig. 11

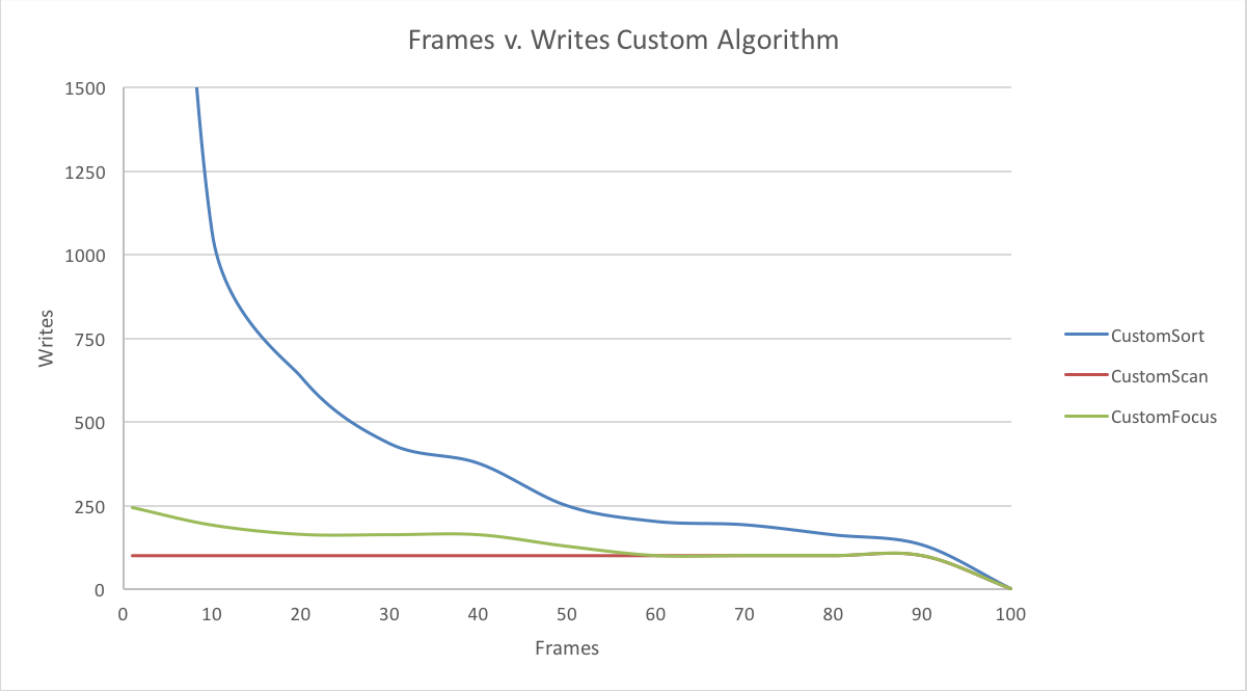
Frames v. Writes Custom Algorithm
Writes
1500
1250
1000
750
500
250
0
0
10
20
30
40
50
60
70
80
90
100
Frames
CustomSort
CustomScan
CustomFocus

Fig. 12