# Report and graph plotting

March 31, 2020

The purpose of the experiments is to demonstrate how different algorithms in page replacements effectively reduce page faults in different scenarios. Using trace files from virtual memory accesses, we can simulate how physical frames changed and count the number of page faults.

So, we simulate physical frame changes from the trace files. Moreover, we assumed that this trace file is created from a process from a 64-bit machine and everytime we run the test, we run a test script ("test_script.sh") which you can simply use the make file to compile the code quickly.

The run-time environment is Ubuntu 18.04.4 LTS and code is compiled using gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0.

**Command line arguments:** virtmem <trace file> <page size> <physical frames> <rand|fifo|lru>

**RESULT**

For each trace generated, the sort.trace is generated from the scenario when we sorted the data, the focus.trace is generated from the scenario when the data is manipulated randomly but the calulation is focused within a range of 25 which has a high locality when manipulating in focused mode, and the scan.trace is generated from the scenario when we scan the data many times from the first index to the last index.

So, we run the test and we got interesting graphs below.

From the result, LRU replacement policy always have the least number of page faults in most scenarios. Frankly, we remove the least recently used frame which is logically good for every scenario in the give trace files. The only weakness that LRU has is when data are accessed from different frames often. Luckily, that scenario is not included in here.

When the memory are accessed in the same place often (high locality, since we accessing the same data multiple time) shown in the sort.trace and focus.trace, FIFO replacement policy performs worst compared to other policies, because FIFO will evict the oldest frame out of the frame table, which could be a problem when the evicted frame is likely to be accessed again. So, increasing frames could be effective but not as much as shown in other policies.

**OBSERVATION: The chance of evicting good frame is low when the number of frames are large enough in RANDOM policy**

Though FIFO performs better than RANDOM policy with few frames of physical frames, but RANDOM policy appeared to be better than FIFO when given more than 10 physical frames in the same scenarios, except for scanning. RANDOM policy seems ridiculous, but with the randomness, we can avoid removing the frame that is likely to be used again while taking the risk of removing it at same time. Compared to FIFO, RANDOM policy will make every frame has $\frac{1}{\text{number of physical frames}}$

chance of getting evicted which means the frame which is most likely to be used again also has $\frac{1}{\text{number of physical frames}}$ chance. Especially when there are 10 frames or more, the chance is less than 10% which is a lot better than FIFO, which prefer evicting the old frames and cause more page faults.

RANDOM policy was terrible for scanning data program, since we need to acquire the data in an order, which we should not randomly evict the physical frame.

In conclusion, LRU is the best policy among other polices, follow by RANDOM policy and FIFO policy. When accessing the same data often, FIFO policy is terrible as shown in sort.trace and focus.trace. When accessing data in an order, RANDOM policy is terrible as shown in scan.trace.

[2]:
```
from matplotlib import pyplot as plt
import numpy
```

[7]:
```
xss = [i for i in range(3,21)]
```

[9]:
```
#I did not do this often, so do not worry though. I was just not in a coding␣
↪mood.

f256lru   = "545918 425444 380697 376786 323695 312444 278760 219591 200225␣
↪176935 105393 61960 40408 33829 33286 33003 32699 32451"
f256fifo  = "607141 535787 450001 419904 382328 343450 323483 308238 305010␣
↪302527 199476 177704 63742 37123 34823 34383 34106 33843"
f256rand  = "644535 553311 488405 435909 388377 344748 301506 258773 217301␣
↪175475 134995 95627 61062 38760 36186 35699 35022 34838"
f1024lru  = "495762 377678 323103 281598 220460 189253 158544 117615 67179␣
↪26395 6337 5713 5490 5305 5172 5042 4973 4839"
f1024fifo = "561662 452436 406011 383822 295853 256148 252929 249892 139485␣
↪137639 8399 7155 6881 6412 6209 6094 5914 5828"
f1024rand = "594392 497855 424642 363001 307324 254382 203543 153084 104609␣
↪56313 9130 7707 7450 7120 6900 6614 6347 6105"
s256lru   = "202786 180556 159396 135189 119186 101246 74248 52241 37491 33898␣
↪31285 25667 18376 15941 15455 15118 14814 14530"
s256fifo  = "229338 196252 163483 150502 144764 141807 111488 62482 49690 47959␣
↪46563 43245 25606 18455 17083 16607 16305 16033"
s256rand  = "254693 210725 178650 152450 127499 103318 79474 59879 48550 41767␣
↪35476 29272 24014 20026 18469 17993 17333 17028"
s1024lru  = "153819 127624 92908 71644 45871 28182 18580 14241 10767 8459 6135␣
↪5768 5539 5355 5221 5091 5014 4882"
s1024fifo = "170581 134191 115708 109386 76282 40798 31084 21945 21470 14252␣
↪7510 7236 6948 6474 6265 6148 5964 5877"
s1024rand = "195874 150544 115714 86207 57429 39310 27091 18826 15219 11831␣
↪8285 7804 7520 7215 6963 6670 6419 6159"
sc256lru  = "39367 30575 25006 21196 18257 17022 16436 15707 15401 14882 14392␣
↪13875 13475 13214 13038 12826 12550 12325"
```

```
sc256fifo = "50428 37859 29132 24087 20311 19308 18169 17633 17104 16568 15958␣
 ↪15465 15064 14710 14317 14063 13892 13705"
sc256rand = "51724 38901 31117 26596 23453 21742 20392 19366 18671 17950 17264␣
 ↪16706 16308 15931 15478 15360 14876 14640"
sc1024lru = "23273 15839 11472 10273 9431 8436 7725 7089 6657 6372 6064 5699␣
 ↪5476 5294 5162 5033 4958 4827"
sc1024fifo= "29053 20096 13972 12286 10894 9932 9213 8672 8266 7621 7388 7134␣
 ↪6863 6399 6201 6083 5901 5819"
sc1024rand= "32369 22136 16151 13987 12468 11316 10377 9645 9044 8710 8166 7696␣
 ↪7407 7090 6871 6595 6330 6068"
```

**Data referenced from test_script.sh output please do not remove this. This is my sanity check.**

file: focus.trace pagesize: 256

algo: lru 545918 425444 380697 376786 323695 312444 278760 219591 200225 176935 105393 61960 40408 33829 33286 33003 32699 32451

algo: fifo 607141 535787 450001 419904 382328 343450 323483 308238 305010 302527 199476 177704 63742 37123 34823 34383 34106 33843

algo: rand 644535 553311 488405 435909 388377 344748 301506 258773 217301 175475 134995 95627 61062 38760 36186 35699 35022 34838

pagesize: 1024 algo: lru 495762 377678 323103 281598 220460 189253 158544 117615 67179 26395 6337 5713 5490 5305 5172 5042 4973 4839

algo: fifo 561662 452436 406011 383822 295853 256148 252929 249892 139485 137639 8399 7155 6881 6412 6209 6094 5914 5828

algo: rand 594392 497855 424642 363001 307324 254382 203543 153084 104609 56313 9130 7707 7450 7120 6900 6614 6347 6105

file: sort.trace pagesize: 256

algo: lru 202786 180556 159396 135189 119186 101246 74248 52241 37491 33898 31285 25667 18376 15941 15455 15118 14814 14530

algo: fifo 229338 196252 163483 150502 144764 141807 111488 62482 49690 47959 46563 43245 25606 18455 17083 16607 16305 16033

algo: rand 254693 210725 178650 152450 127499 103318 79474 59879 48550 41767 35476 29272 24014 20026 18469 17993 17333 17028

pagesize: 1024

algo: lru 153819 127624 92908 71644 45871 28182 18580 14241 10767 8459 6135 5768 5539 5355 5221 5091 5014 4882

algo: fifo 170581 134191 115708 109386 76282 40798 31084 21945 21470 14252 7510 7236 6948 6474 6265 6148 5964 5877

algo: rand 195874 150544 115714 86207 57429 39310 27091 18826 15219 11831 8285 7804 7520 7215 6963 6670 6419 6159

file: scan.trace pagesize: 256

algo: lru 39367 30575 25006 21196 18257 17022 16436 15707 15401 14882 14392 13875 13475 13214 13038 12826 12550 12325

algo: fifo 50428 37859 29132 24087 20311 19308 18169 17633 17104 16568 15958 15465 15064 14710 14317 14063 13892 13705

algo: rand 51724 38901 31117 26596 23453 21742 20392 19366 18671 17950 17264 16706 16308 15931 15478 15360 14876 14640

pagesize: 1024

algo: lru 23273 15839 11472 10273 9431 8436 7725 7089 6657 6372 6064 5699 5476 5294 5162 5033 4958 4827

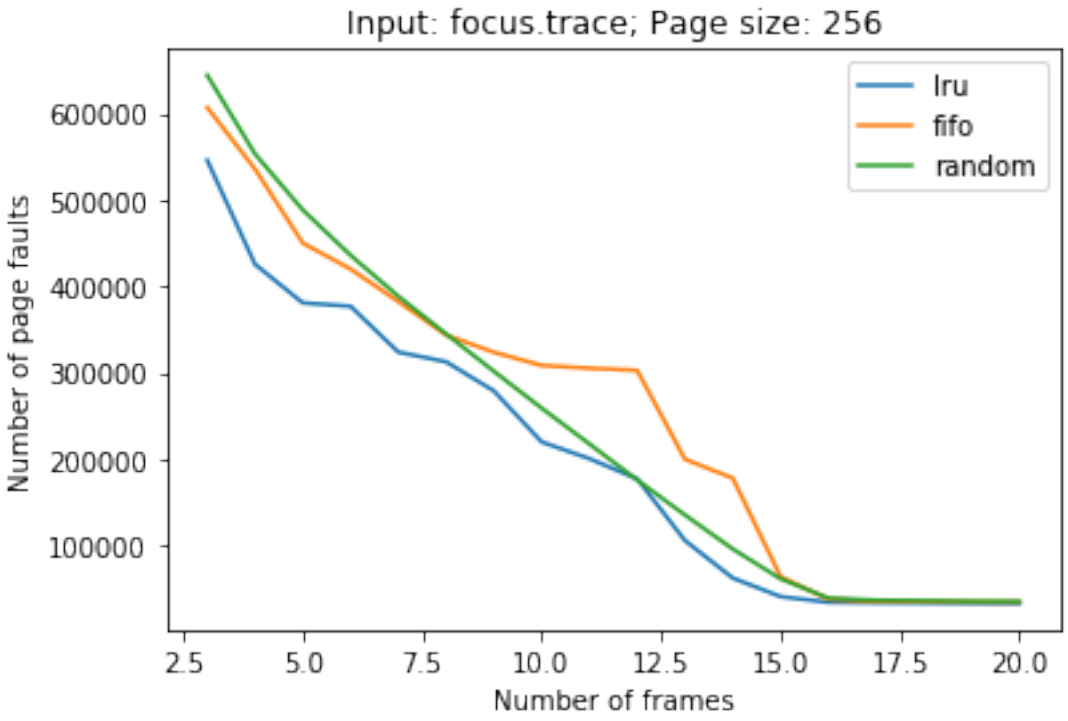
algo: fifo 29053 20096 13972 12286 10894 9932 9213 8672 8266 7621 7388 7134 6863 6399 6201 6083 5901 5819

algo: rand 32369 22136 16151 13987 12468 11316 10377 9645 9044 8710 8166 7696 7407 7090 6871 6595 6330 6068

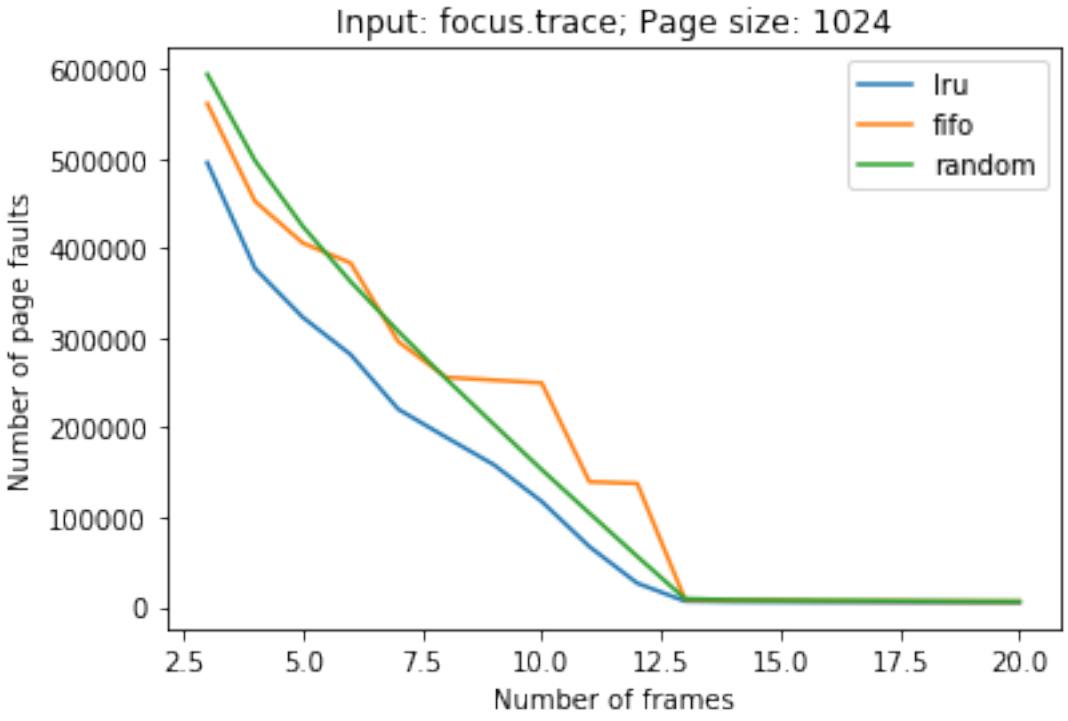
[21]:
```
def plotter(lru, fifo, rands, pagesize, file):
    plt.plot(xss,lru,label='lru')
    plt.plot(xss,fifo,label='fifo')
    plt.plot(xss,rands,label='random')
    plt.title("Input: "+file+"; Page size: "+str(pagesize))
    plt.xlabel("Number of frames")
    plt.ylabel("Number of page faults")
    plt.legend()
    plt.savefig(file+str(pagesize)+".png")

def prepare_array(array):
    split = array.split(' ')
    return [int(i) for i in split]
```

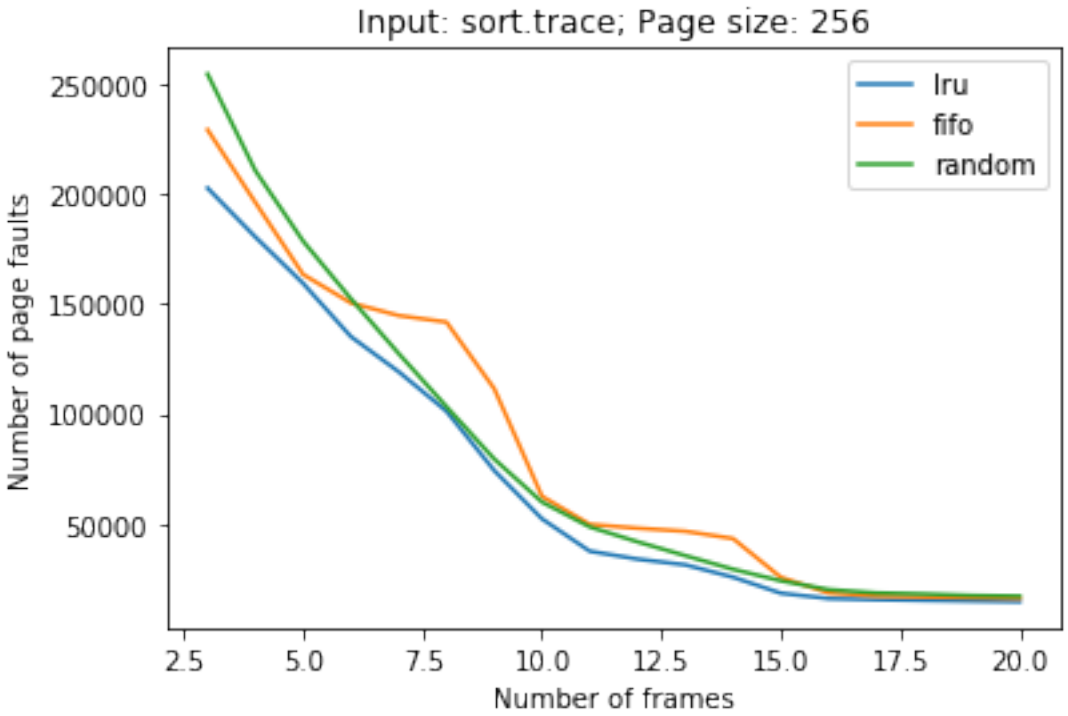
[22]:
```
plotter(prepare_array(f256lru), prepare_array(f256fifo),␣
↪prepare_array(f256rand), 256, "focus.trace")
```



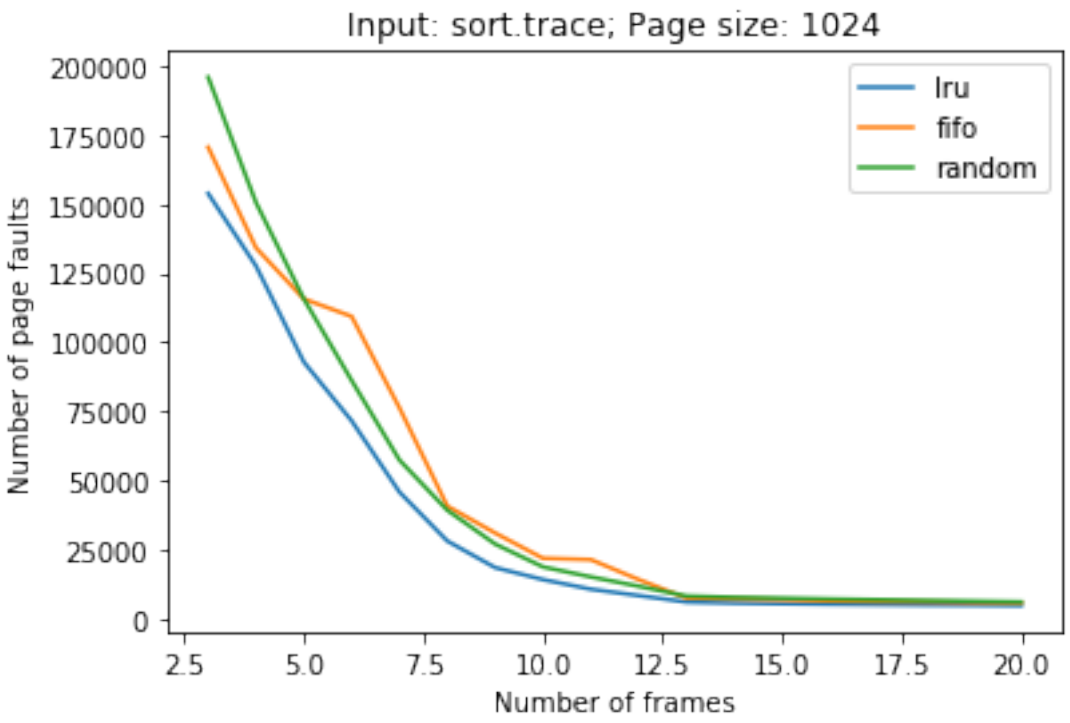

[23]:
```
plotter(prepare_array(f1024lru), prepare_array(f1024fifo),␣
↪prepare_array(f1024rand), 1024, "focus.trace")
```



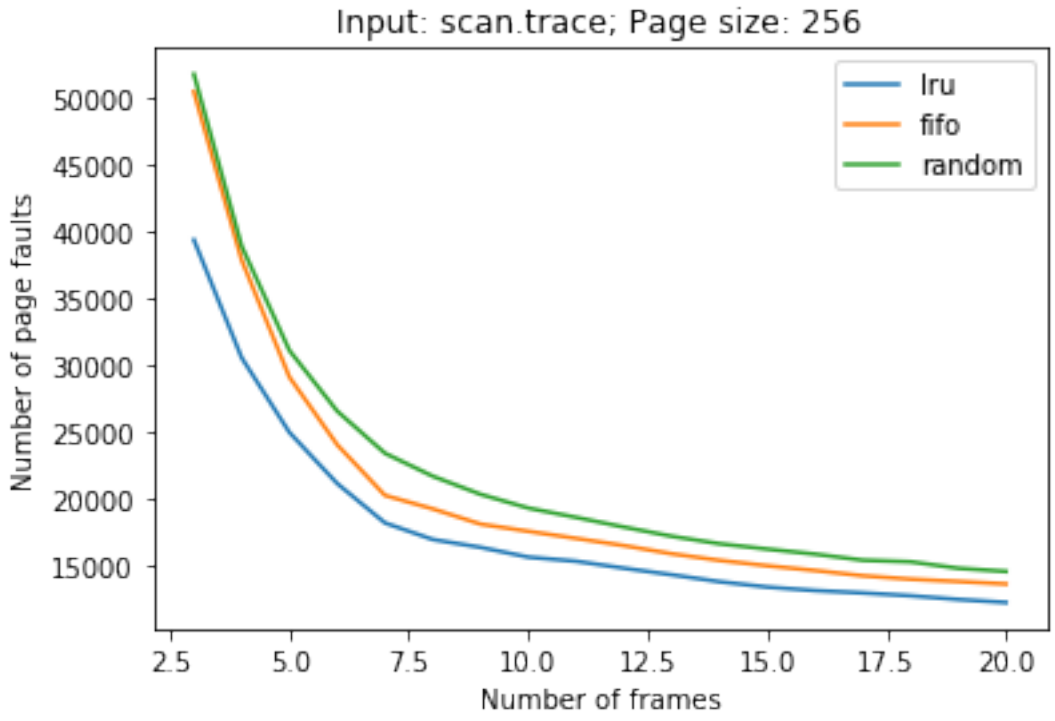
[24]:
```
plotter(prepare_array(s256lru), prepare_array(s256fifo),␣
↪prepare_array(s256rand), 256, "sort.trace")
```



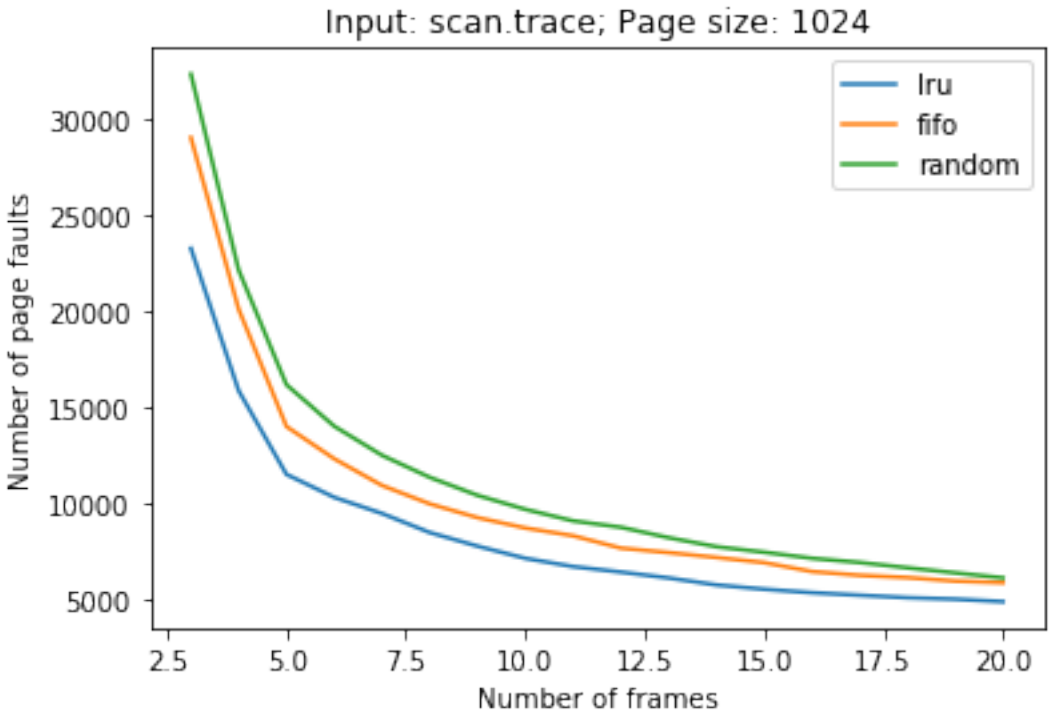

[25]:
```
plotter(prepare_array(s1024lru), prepare_array(s1024fifo),␣
↪prepare_array(s1024rand), 1024, "sort.trace")
```




[26]:
```
plotter(prepare_array(sc256lru), prepare_array(sc256fifo),␣
↪prepare_array(sc256rand), 256, "scan.trace")
```



[27]:
```
plotter(prepare_array(sc1024lru), prepare_array(sc1024fifo),␣
↪prepare_array(sc1024rand), 1024, "scan.trace")
```




[ ]: