System Setup:

The experiments were conducted on a native Linux system running Ubuntu 18.04, deployed on CloudLab resources. System memory details were obtained using the `cat /proc/meminfo` command, providing insights into the available memory, virtual memory, and huge page configuration. Key memory parameters are as follows:

```
MemTotal:       65835584 kB
MemFree:        60201116 kB
MemAvailable:   63830432 kB
…………………………
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
…………………………
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:      295420 kB
DirectMap2M:    13203456 kB
DirectMap1G:    55574528 kB
```

(Note: All .sh files require `chmod +x .sh` for permissions)

PART I:

Code Overview:

```
for (j = 0; j < CHUNK; ) {
      int step_size = (rand() % (64 - 1) + 1) * 4096;  // Random step between 4KB and 64KB
      if (rand() % 2 == 0) { // Randomly deallocate to create fragmentation
            madvise(data[i] + j, 4096, MADV_DONTNEED); // Reclaimable page
            printf("Fragmenting chunk %d at offset %d\n", i, j);
            }
      j += step_size;
}
```

This code snippet creates memory fragmentation by iterating over a memory chunk and randomly deallocating small portions. It works as follows:

- A random step size between 4KB and 64KB is chosen for each iteration.
- On each step, there is a 50% chance that a 4KB page will be deallocated using `madvise()`, marking it as reclaimable.

- The loop continues across the chunk, fragmenting it by freeing random pages at varying offsets.

This is done over 64Gb/24Mb = ~ 2731 Chunks. The allocation size is chosen based on the MemTotal value which indicates the total usable Ram size. The large allocation size is also due to there being no set limit on the max virtual memory size as indicated by `ulimit -v`. As such the overall motivation for 64G is to highlight the effects of fragmentation.

Experimental Steps:

1. **Ensure no other memory-intensive programs are running**: This isolates the impact of your fragmentation program.
2. **Run `./compact.sh` to compact the memory**: This initializes your experiment by compacting the memory before fragmentation occurs.
3. **Wait for 1 minute for the OS to compact the memory**: Allow sufficient time for compaction.
4. **Run `./check-fmfi.sh` and note the fragmentation score**: This gives you a baseline fragmentation score.
5. **Use `./frag-program 64G (no_thp`**: This will create fragmentation in memory.
6. **While `frag-program` is running**, run `./compact.sh`:
   - This allows you to see how the system attempts to compact fragmented memory in real-time, even while fragmentation is occurring.
   - Observe any changes in memory behavior or performance as a result of compaction.
7. **Check the fragmentation score again**: After running `./compact.sh`, use `./check-fmfi.sh` to see how compaction has affected fragmentation while `frag-program` is still running.

Fragmentation Scores (THP Disabled):

Compact - Frag Score

```
Node 0, zone      DMA 0.000 0.000 0.001 0.003 0.005 0.009 0.017 0.033 0.033 0.098 0.226
Node 0, zone    DMA32 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.004 0.008 0.013
Node 0, zone   Normal 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.002 0.004 .009
```

Fragmentation - Frag Score
Golden:

```
Node 0, zone      DMA 0.000 0.000 0.001 0.003 0.005 0.009 0.017 0.033 0.033 0.098 0.226
Node 0, zone    DMA32 0.000 0.000 0.000 0.001 0.002 0.003 0.005 0.008 0.013 0.023 0.037
Node 0, zone   Normal 0.000 0.004 0.017 0.038 0.065 0.098 0.141 0.185 0.243 0.307 0.378
```

Our Test:

```
Node 0, zone      DMA 0.000 0.000 0.001 0.003 0.005 0.009 0.017 0.033 0.033 0.098 0.226
Node 0, zone    DMA32 0.000 0.001 0.003 0.004 0.004 0.008 0.011 0.015 0.015 0.023 0.023
Node 0, zone   Normal 0.000 0.003 0.011 0.030 0.087 0.136 0.174 0.186 0.248 0.304 0.406
```

Compact After Frag - Frag Score
Golden:

```
Node 0, zone      DMA 0.000 0.000 0.001 0.003 0.005 0.009 0.017 0.033 0.033 0.098 0.226
Node 0, zone    DMA32 0.000 0.000 0.000 0.001 0.003 0.005 0.007 0.010 0.016 0.029 0.037
Node 0, zone   Normal 0.000 0.000 0.000 0.001 0.002 0.004 0.006 0.009 0.011 0.111 0.245
```

Our Test:

```
Node 0, zone      DMA 0.000 0.000 0.001 0.003 0.005 0.009 0.017 0.033 0.033 0.098 0.226
Node 0, zone    DMA32 0.000 0.001 0.003 0.004 0.004 0.008 0.012 0.016 0.016 0.020 0.020
Node 0, zone   Normal 0.000 0.005 0.016 0.039 0.107 0.165 0.209 0.226 0.241 0.259 0.272
```

Conclusion:

The fragmentation program proved effective in significantly increasing memory fragmentation, particularly in the **Normal** zone, where the fragmentation score rose from **0.007** to **0.406**. This demonstrates the program's capacity to heavily fragment system memory. After running the `compact.sh` script, the memory was partially defragmented, with the fragmentation score in the **Normal** zone dropping to **0.272**. However, this was less effective compared to the golden baseline, where the fragmentation score was reduced to **0.245**. This suggests that while the compaction script helps mitigate fragmentation, its effectiveness may vary depending on system conditions and the extent of fragmentation.

PART II:

(Note: Needed to update kernel with `sudo apt-get install linux-tools-common linux-tools-generic linux-tools-(uname -r)`)

File Organization:

```
px4_mp2.zip
|----> MP2Report.pdf
|----> fragmentation/
       |----> frag-program
       |----> frag-program-gold
       |----> check-fmfi.sh
       |----> compact.sh
|----> hugepage/
       |----> logs/
              |----> canneal/
              |----> fmm/
              |----> streamcluster/
       |----> results/
              |----> canneal/
              |----> fmm/
              |----> streamcluster/
       |----> parsec-3.0/

       |----> disable-thp-hugepage.sh
       |----> enable-thp-hugepage.sh
       |----> perf-all.sh
       |----> run-all.sh
```

Within the `hugepage/logs` directory, the total execution time for each benchmark is stored in separate bench folders, with results categorized for both THP-enabled and THP-disabled runs. Similarly, the `hugepage/results` directory contains CSV files with detailed data on cache and TLB misses, organized by benchmark in the same way, with both THP and no-THP results grouped accordingly. These changes are implemented in the `run-all.sh` and `perf-all.sh` scripts.

Results:

Execution time (THP Enabled)

Canneal:

```
real    3m59.511s
user   3m59.059s
sys    0m0.452s
```

Fmm:

```
real 1m38.981s
user  1m38.764s
sys  0m0.218s
```

Streamcluster:

```
real   7m9.417s
user   7m9.404s
sys   0m0.008s
```

Execution time (THP Disabled)

Canneal:

```
real   4m9.881s
user   4m9.412s
sys   0m0.468s
```

Fmm:

```
real  1m42.412s
user  1m41.640s
sys   0m0.773s
```

Streamcluster:

```
real  7m54.771s
user  7m54.721s
sys   0m0.048s
```

# of iTLB misses, dTLB misses, and Cache misses (THP Enabled)

Canneal:

```
524074,,dTLB-load-misses,239317377674,100.00,,
243742,,iTLB-load-misses,239317371268,100.00,,
12672653,,cache-misses,239317364042,100.00,,
```

Fmm:

```
231039,,dTLB-load-misses,100397434915,100.00,,
110001,,iTLB-load-misses,100397432434,100.00,,
5722100,,cache-misses,100397430464,100.00,,
```

Streamcluster:

```
704424,,dTLB-load-misses,435749706378,100.00,,
373907,,iTLB-load-misses,435749707867,100.00,,
19129316,,cache-misses,435749704399,100.00,,
```

# of iTLB misses, dTLB misses, and Cache misses (THP Disabled)
Canneal:
```
705221,,dTLB-load-misses,250068948346,100.00,,
319231,,iTLB-load-misses,250068952271,100.00,,
18212537,,cache-misses,250068953524,100.00,,
```
Fmm:
```
571794,,dTLB-load-misses,103529403382,100.00,,
422157,,iTLB-load-misses,103529411696,100.00,,
13047344,,cache-misses,103529414744,100.00,,
```
Streamcluster:
```
1092582,,dTLB-load-misses,477740738763,100.00,,
528868,,iTLB-load-misses,477740734388,100.00,,
28123182,,cache-misses,477740726777,100.00,,
```

Timing Improvements (End-to-End):
Canneal:
THP Enabled: 3m59.511s, THP Disabled: 4m9.881s
~4.2% faster with THP
Fmm:
THP Enabled: 1m38.981s, THP Disabled: 1m42.412s
~3.4% faster with THP
Streamcluster:
THP Enabled: 7m9.417s, THP Disabled: 7m54.771s
~9.6% faster with THP

TLB Hit Rate Improvements:
Canneal:
- dTLB misses reduced by ~25.7% ((705,221 - 524,074) / 705,221 * 100)
- iTLB misses reduced by ~23.7% ((319,231 - 243,742) / 319,231 * 100)

Fmm:
- dTLB misses reduced by ~59.6% ((571,794 - 231,039) / 571,794 * 100)
- iTLB misses reduced by ~73.9% ((422,157 - 110,001) / 422,157 * 100)

Streamcluster:
- dTLB misses reduced by ~35.5% ((1,092,582 - 704,424) / 1,092,582 * 100)
- iTLB misses reduced by ~29.3% ((528,868 - 373,907) / 528,868 * 100)

Analysis:
Given the results shown above there are noticeable improvements across each benchmark with varying effect. The primary reason for the performance gain is likely the reduced TLB miss overhead, as fewer TLB misses result in fewer page table walks, improving overall memory access efficiency.
However, the improvements are modest because:

- THP mainly optimizes memory management and TLB usage, which might not be the main bottleneck for these particular workloads.
- Cache performance does not show as much improvement with THP. In fact, cache misses remain relatively high in both THP-enabled and THP-disabled cases, suggesting that other factors, such as cache utilization, may be limiting further performance gains.

THP improves both the TLB hit rate and the execution time of the benchmarks, though the overall performance boost is modest. The primary benefit of THP is reducing TLB misses, which is most effective for benchmarks like Fmm that benefit significantly from improved memory management. The relatively small performance improvements suggest that memory access might not be the only limiting factor for these applications.

Fragmentation Score (THP Enabled):

(Utilized same approach as Part I without Gold Binary)

Compact - Frag Score

```
Node 0, zone      DMA 0.000 0.000 0.001 0.003 0.005 0.009 0.017 0.033 0.033 0.098 0.226
Node 0, zone    DMA32 0.000 0.000 0.000 0.001 0.002 0.003 0.005 0.008 0.013 0.023 0.037
Node 0, zone   Normal 0.000 0.000 0.000 0.000 0.000 0.001 0.002 0.003 0.005 0.008 0.018
```

Fragmentation - Frag Score
Our Test:

```
 Node 0, zone      DMA 0.000 0.000 0.001 0.003 0.005 0.009 0.017 0.033 0.033 0.098 0.226
Node 0, zone    DMA32 0.000 0.105 0.117 0.137 0.174 0.240 0.352 0.501 0.689 0.927 1.000
Node 0, zone   Normal 0.000 0.007 0.037 0.223 0.315 0.560 0.788 0.918 0.960 1.000 1.000
```

Compact After Frag - Frag Score
Our Test:

```
Node 0, zone      DMA 0.000 0.000 0.001 0.003 0.005 0.009 0.017 0.033 0.033 0.098 0.226
Node 0, zone    DMA32 0.000 0.000 0.001 0.002 0.004 0.008 0.017 0.025 0.031 0.043 0.254
Node 0, zone   Normal 0.000 0.025 0.054 0.090 0.126 0.210 0.315 0.379 0.428 0.442 0.442
```

Analysis:
Before memory compaction, the **DMA32** and **Normal zones** exhibited significant fragmentation, with scores reaching **1.0**, indicating maximum fragmentation, especially in memory regions handling larger allocations. After running the compaction script, fragmentation was significantly reduced in these zones, with **DMA32** dropping to **0.254** and **Normal** to **0.442** in the highest buckets. This demonstrates that memory compaction was effective in consolidating memory, though some residual fragmentation remained, especially at higher memory usage. The **DMA zone** showed minimal fragmentation throughout, and compaction had little impact there. Overall, compaction was successful in mitigating fragmentation but could not fully eliminate it in heavily fragmented areas.
When referring to the differences between huge pages and 4Kb pages, huge pages offer performance benefits by reducing TLB misses but are more sensitive to fragmentation because they need contiguous

memory blocks, while 4 KB pages are less prone to fragmentation but do not provide the same performance gains.