Hardware Performance Counters

Janice McMahon

September 2022

Motivation for Hardware Performance Counters

- The Lucata workflow allows for many opportunities to check functional correctness via x86 execution and simulation
- However, understanding system performance may require more detailed insight into the hardware performance counters that can be used.
- There are 5 types of possible counters
  - Node Counters
  - MSP Counters
  - GC Cluster Counters
  - GC Counters
  - SRIO Counters

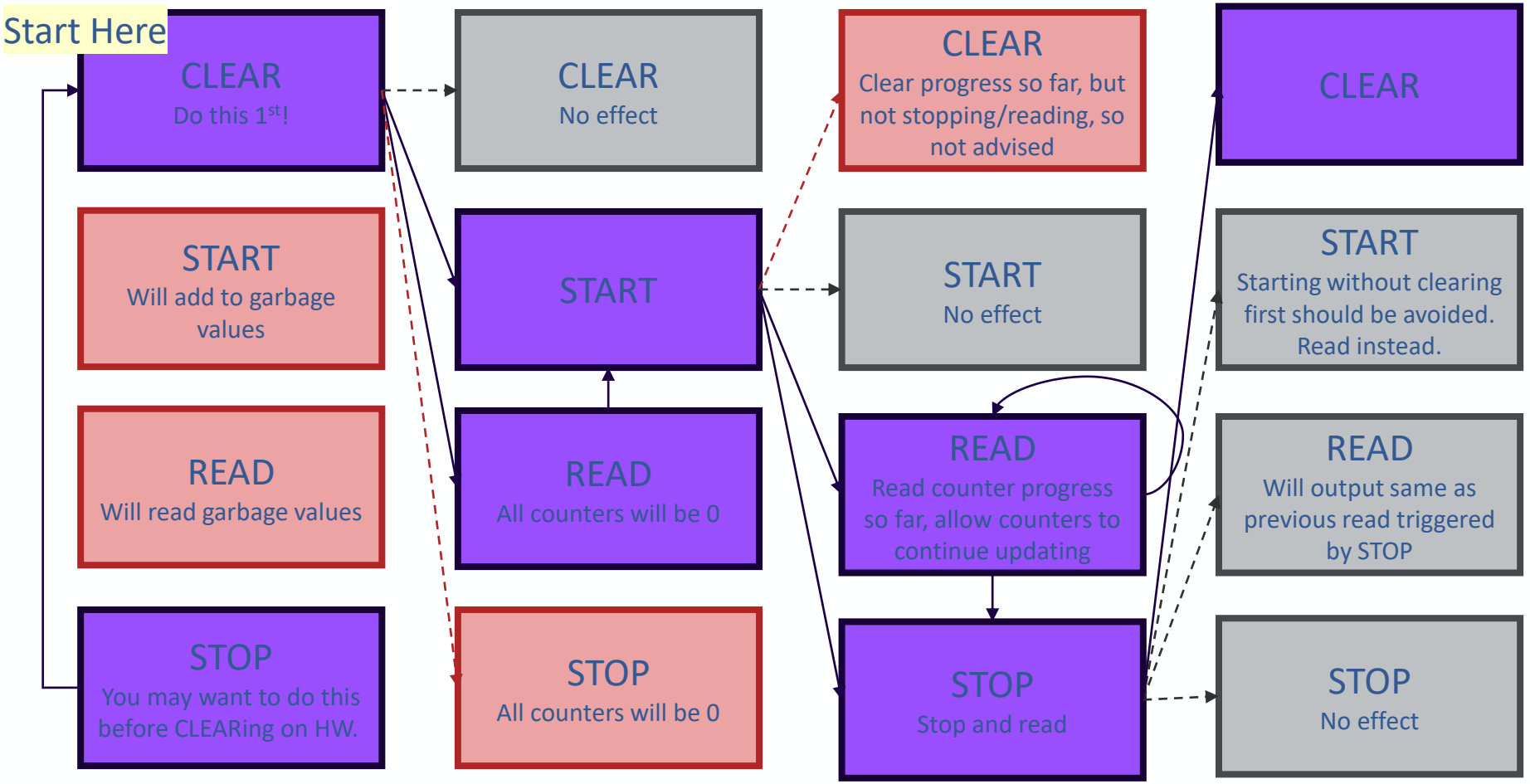
Function for Hardware Performance Counters

- Single library function controls counters

```
void lu_profile_perfcntr(perfcntr_e op, char *str);
```

- "Op" argument determines operation to be performed
  - PFC_CLEAR: initialize/clear counters
  - PFC_START: start the counters without reading
  - PFC_READ: output current counter values
  - PFC_STOP: stop and read counters
- "Str" argument determines string for output file
  - Used as key for JSON object
  - Should be unique for all calls to profiler
- Use only during serial portion of code
- Must be used according to rules for well-defined behavior

Performance Counter Usage Flowchart



Performance Counters: example

```
starttiming();
lu_profile_perfcntr(PFC_CLEAR, "SIMPLE ARRAY SUM CLEARING COUNTERS");
lu_profile_perfcntr(PFC_READ, "SIMPLE ARRAY SUM READING COUNTERS AFTER CLEARING");
lu_profile_perfcntr(PFC_START, "SIMPLE ARRAY SUM STARTING COUNTERS");5

// Spawn child thread to sum values in A
sumA = cilk_spawn sum(A, epn);

// Call function to sum values of B (in current/parent thread)
sumB = sum(B, epn);

// Wait for spawned thread to complete
cilk_sync;

printf("A = %ld, B = %ld\n", sumA, sumB);
lu_profile_perfcntr(PFC_STOP, "SIMPLE ARRAY SUM STOPPING COUNTERS AT END");
return 0;
```

Initialize and start counters in beginning, serial portion of program

Stop and read counters at end of program

Performance Counter Output

```
{
 "SIMPLE ARRAY SUM CLEARING COUNTERS": "N/A",
 "SIMPLE ARRAY SUM READING COUNTERS AFTER CLEARING": {
    "System_Parameters": {
       "Total_Nodes":2,
       "MSPs_Per_Node":8,
       "GC_Clusters_Per_Node":3,
       "GCs_Per_GC_Cluster":8
    },
    "MSP_Counters": {
       "MSP_memory_reads": {
          "Node0": {
             "MSP0":0,
             "MSP1":0,
             "MSP2":0,
             "MSP3":0,
             "MSP4":0,
             "MSP5":0,
             "MSP6":0,
             "MSP7":0
          },
          "Node1": {
             "MSP0":0,
             "MSP1":0,
             "MSP2":0,
             "MSP3":0,
             "MSP4":0,
             "MSP5":0,
             "MSP6":0,
             "MSP7":0
          }
       },
       ...
```

```
"SIMPLE ARRAY SUM STARTING COUNTERS": "N/A",
"SIMPLE ARRAY SUM STOPPING COUNTERS AT END": {
 "System_Parameters": {
    "Total_Nodes":2,
    "MSPs_Per_Node":8,
    "GC_Clusters_Per_Node":3,
    "GCs_Per_GC_Cluster":8
 },
 "MSP_Counters": {
    "MSP_memory_reads": {
       "Node0": {
          "MSP0":49,
          "MSP1":23,
          "MSP2":144,
          "MSP3":153,
          "MSP4":7,
          "MSP5":70,
          "MSP6":61,
          "MSP7":109
       },
       "Node1": {
          "MSP0":35,
          "MSP1":5,
          "MSP2":2,
          "MSP3":0,
          "MSP4":0,
          "MSP5":0,
          "MSP6":0,
          "MSP7":0
       }
...
```

Output in JSON format

.hpc file created automatically by simulator

Hardware Execution

- Node managing software creates log files for each node in JSON format
- Counter output .hpc file is created by Python script that aggregates hardware log files

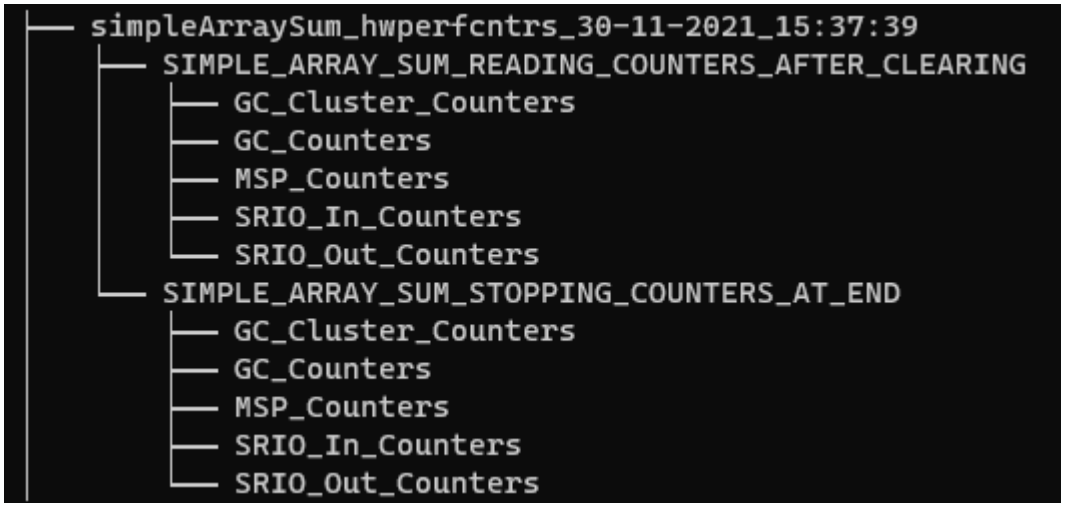
```
1 usage: emu_pc_concat.py [-h] -f PC_FILE [-d OUTPUT_DIR]
2
3 optional arguments:
4 -h, --help show this help message and exit
5 -f PC_FILE, --pc_file PC_FILE
6             mn_exec_pc.PID.log file name
7 -d OUTPUT_DIR, --output_dir OUTPUT_DIR
8             path to directory to save .hpc file if you don't want
9             to save it to default location (path to
10              mn_exec_pc.PID.log file)
```

Performance Counter Graphs

- Graph images are created by Python script that operates on .hpc output file

```
1 usage: make_hpc_plots.py [-h] -f HPC_FILE [-l LOG_FILE] [-n] [-s]
2
3 optional arguments:
4   -h, --help show this help message and exit
5   -f HPC_FILE, --hpc_file HPC_FILE
6               .hpc file name
7   -l LOG_FILE, --log_file LOG_FILE
8               .out file name
9   -n, --no_graphs only make csvs, no graphs
10  -s, --stop_reads stop parsing JSON after a specific read
```

Graph Directory Structure



Performance Counter Comparison

- Counter reads from different versions of program compared by Python script that operates on .hpc output files

```
1 usage: compare_hwperfcntr_jsons.py [-h] -f FILES [FILES ...] -d SAVE_DIR [-c]
2
3 optional arguments:
4    -h, --help show this help message and exit
5    -f FILES [FILES ...], --files FILES [FILES ...]
6                list of .hpc files to compare
7    -d SAVE_DIR, --save_dir SAVE_DIR
8                where to save the comparison graphs
9    -c, --compare_all Use this flag to compare all sets instead of choosing
10               which sets of counters
```

Usage:

```
./compare_hwperfcntr_jsons.py -f simpleArraySum_hwperfcntrs_diffele.hpc
simpleArraySum_hwperfcntrs.hpc -d example_dir
```

Performance Counter Polling

- Frequent sampling of performance counters on hardware to gather data on system changes over time
- Activated via flag to hardware execution command
- Raw data .csv file and graphs created by Python script executing on hardware log file with polling data

Graph Generation from Polling Data

```
usage: emu_analyze_pr.py [-h] -f PR_FILE [-d OUTPUT_DIR] [-p PREFIX] [-n]
                         [-c NUM_INTERVALS] [-v] [-l] [-s] [-b] [-i INPUT_CSV]

optional arguments:
  -h, --help show this help message and exit
  -f PR_FILE, --pr_file PR_FILE
                        mn_exec_pr.PID.log file name
  -d OUTPUT_DIR, --output_dir OUTPUT_DIR
                        path to directory to save output files to if you don't
                        want to save it to default location (path to
                        mn_exec_pr.PID.log file)
  -p PREFIX, --prefix PREFIX
                        Add prefix to graph directory name and graph names
  -n, --no_graphs only make csvs, no graphs
  -c NUM_INTERVALS, --choose_intervals NUM_INTERVALS
                        choose which intervals to graph
  -v, --no_csv do not create csv
  -l, --no_max_line do not add max threads line to threads graphs
  -s, --stacked_gc_plot
                        graph GC CONTEXTS as stacked area plot
  -b, --gc_subplots graph GC CONTEXTS as one subplot per GC
  -i INPUT_CSV, --input_csv INPUT_CSV
                        path to existing csv, don't have to re-create. CAN
                        SAVE A LOT OF TIME
```

Example: Array Sum

```
…
int main()
{
  allocate_arrays(); // Allocate arrays using malloc2d
  initialize_arrays(); // Initialize arrays

  starttiming();
  lu_profile_perfcntr(PFC_CLEAR, "CLEAR COUNTERS");
  lu_profile_perfcntr(PFC_READ, "READ COUNTERS AFTER CLEAR");
  lu_profile_perfcntr(PFC_START, "START COUNTERS");

  sumA = cilk_spawn sum(A, epn); // Spawn child thread for A
  sumB = sum(B, epn); // Sum values of B in parent thread
  cilk_sync; // Wait for spawned thread to complete
  printf("A = %ld, B = %ld\n", sumA, sumB);
  lu_profile_perfcntr(PFC_STOP, "STOP COUNTERS AT END");
}
```

Clear, read, then start counters

Read after clear; values are zero

Stop counters at end; implicit read of values

Sample Program Execution: Array_Sum.c

```
>>>>>>>> emu-cc ArraySum_hpc.c -o ArraySum_hpc.mwx -l memoryweb
>>>>>>>> emusim.x ArraySum_hpc.mwx

        SystemC 2.3.3-Accellera --- Sep  7 2022 09:15:59
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
Start untimed simulation with local date and time= Wed Sep 14 12:49:27 2022

[WARN]: In /home/tdysart/Toolchains/22.R2-Sept7/llvm-
cilk/mwsim/src/timsim.cpp:sc_main:590 Calling READCNTRS when not profiling!
Timed simulation starting...
A = 32, B = 64
End untimed simulation with local date and time= Wed Sep 14 12:49:31 2022


Info: /OSCI/SystemC: Simulation stopped by user.
>>>>>>>> ls ArraySum_hpc.*
ArraySum_hpc.c    ArraySum_hpc.hpc  ArraySum_hpc.mwx
ArraySum_hpc.cdc  ArraySum_hpc.mps
>>>>>>>> make_hpc_plots.py -f ArraySum_hpc.hpc
Find all graphs in: ./ArraySum_hpc_14-09-2022_12:49:55
The last hpc call to analyze will be 1
Program called lu_profile_perfcntr with message: SIMPLE ARRAY SUM READING
COUNTERS AFTER CLEARING
Generating Graphs for [SIMPLE ARRAY SUM READING COUNTERS AFTER CLEARING]...
/home/tdysart/Toolchains/22.R2-Sept7/llvm-cilk/lucata-
22.R2.Sept7/bin/make_hpc_plots.py:177: UserWarning: Tight layout not
applied. The bottom and top margins cannot be made large enough to
accommodate all axes decorations.
  plt.tight_layout()
Program called lu_profile_perfcntr with message: SIMPLE ARRAY SUM STOPPING
COUNTERS AT END
Generating Graphs for [SIMPLE ARRAY SUM STOPPING COUNTERS AT END]...
Stopping here after read 1
hpc_file_name_base: ArraySum_hpc.hpc
>>>>>>>> ls ArraySum_hpc_14-09-2022_12\:49\:55/*
ArraySum_hpc_14-09-2022_12:49:55/ArraySum_hpc.hpc.csv

'ArraySum_hpc_14-09-
2022_12:49:55/SIMPLE_ARRAY_SUM_READING_COUNTERS_AFTER_CLEARING':
GC_Cluster_Counters  MSP_Counters

'ArraySum_hpc_14-09-
2022_12:49:55/SIMPLE_ARRAY_SUM_STOPPING_COUNTERS_AT_END':
GC_Cluster_Counters  GC_Counters  MSP_Counters
```

Counter output file created

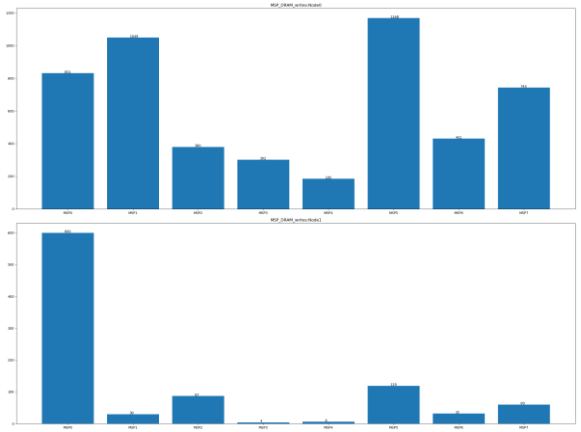
Graphs created in subdirectory

- Simulator creates .hpc file automatically
- Profile images generated after simulation by script
- Subdirectory created for each read of counters in program

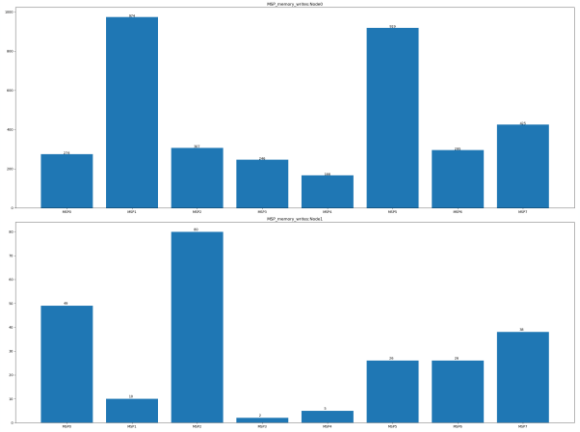
Hybrid BFS: MSP Counters I

2 Nodes, Graph Scale 5

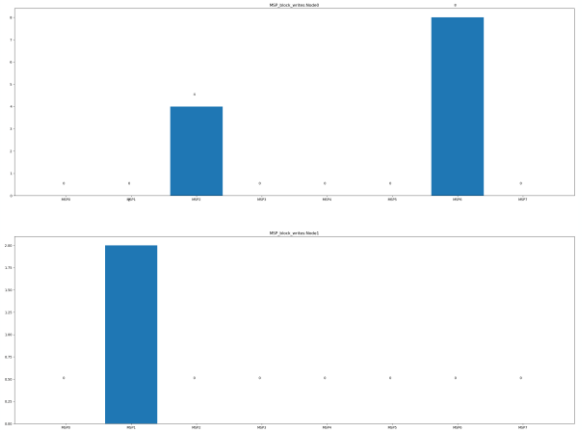
DRAM Writes



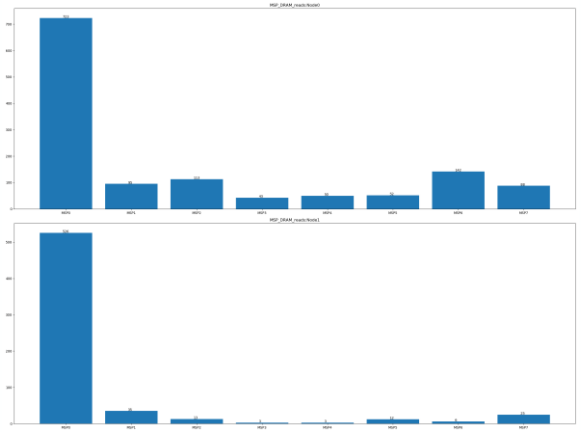
Memory Writes



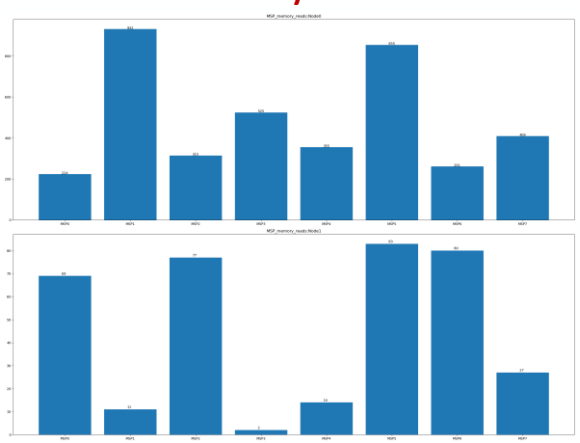
Block Writes



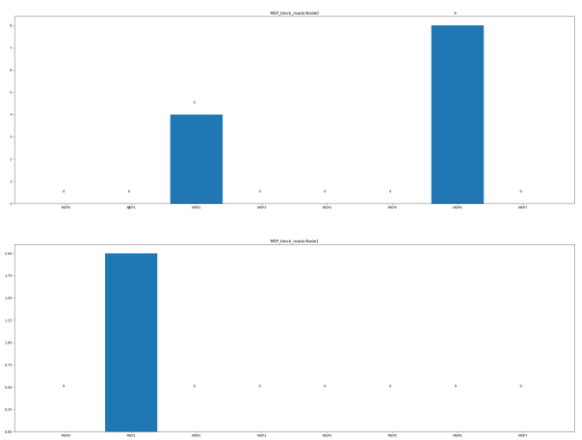
DRAM Reads



Memory Reads

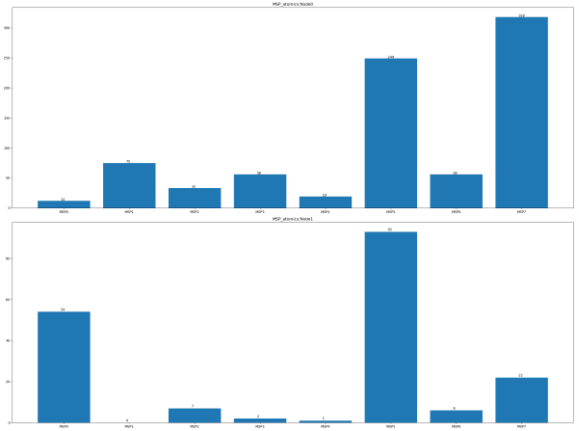


Block Reads

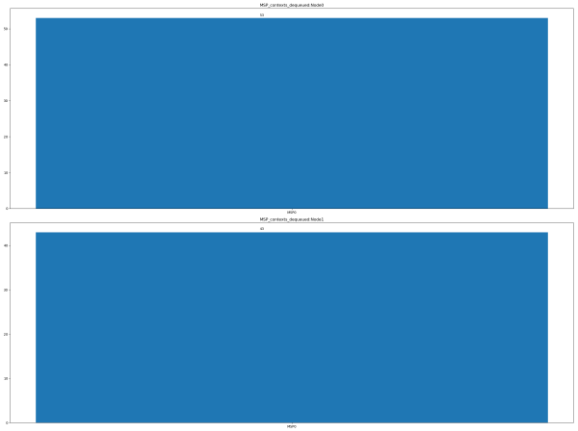


Hybrid BFS: MSP Counters II

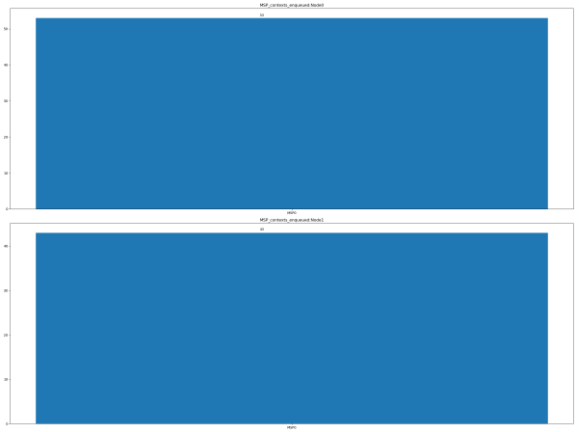
Atomics



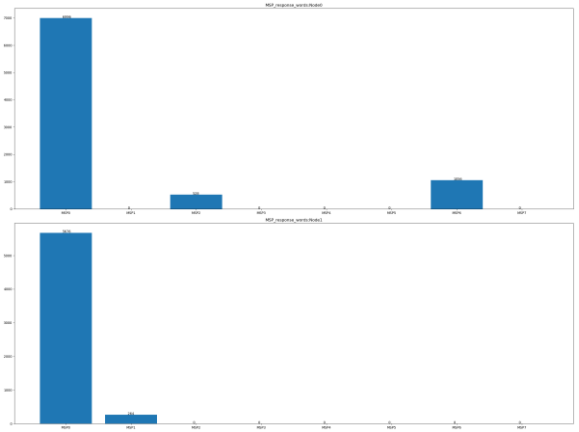
Contexts Dequeued



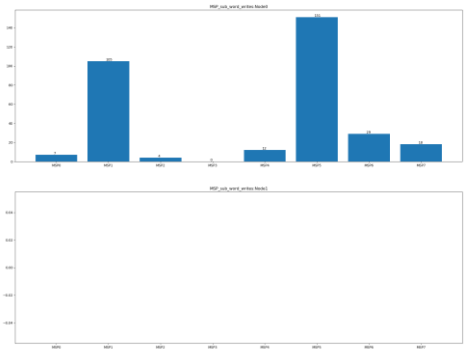
Contexts Enqueued



Response Words



Sub-Word Writes

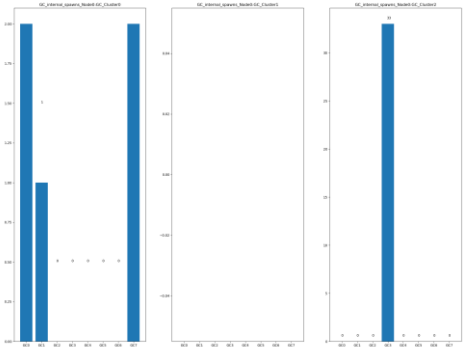


2 Nodes
Graph Scale 5

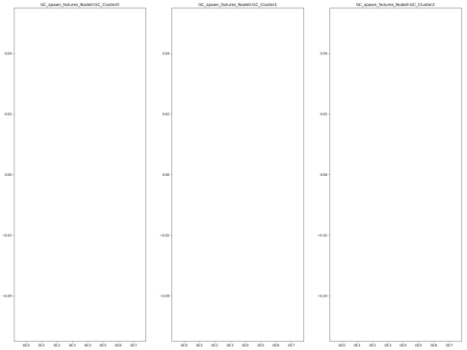
Hybrid BFS: GC Counters I

Node 0

Internal Spawns

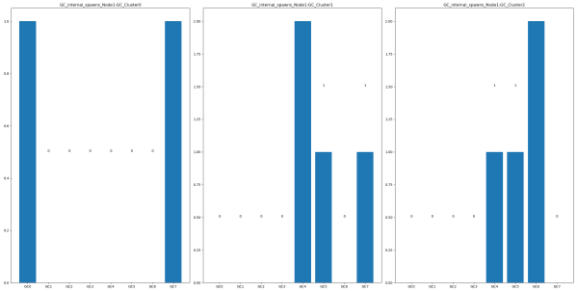


Spawn Failures

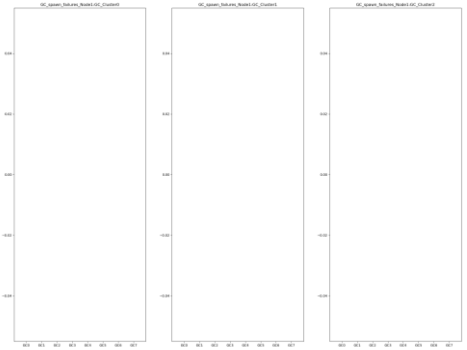


Node 1

Internal Spawns



Spawn Failures

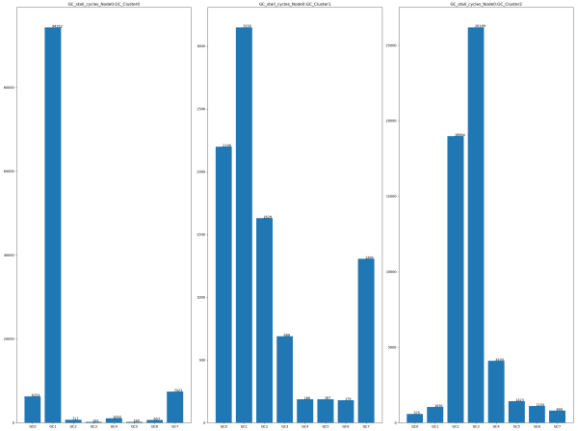


2 Nodes
Graph Scale 5

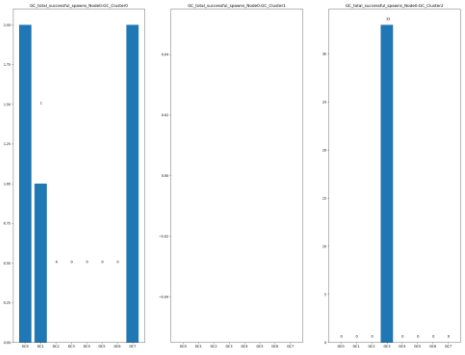
Hybrid BFS: GC Counters II

Node 0

Stall Cycles

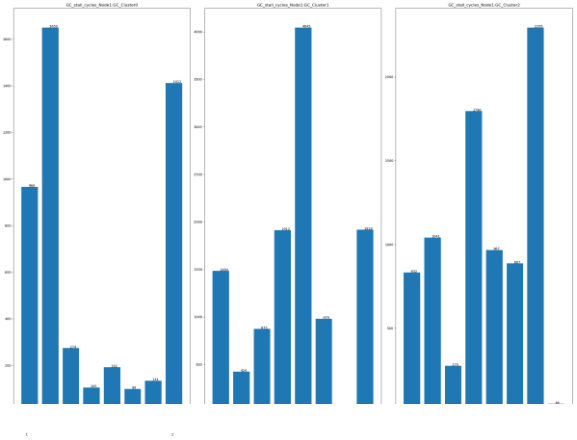


Total Successful Spawns

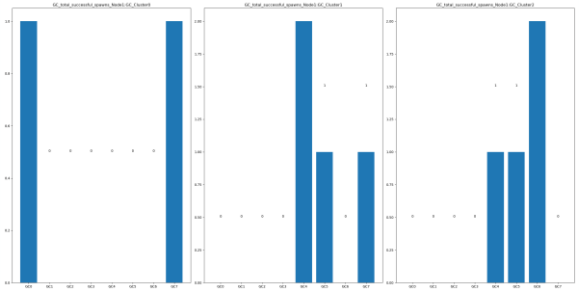


Node 1

Stall Cycles



Total Successful Spawns

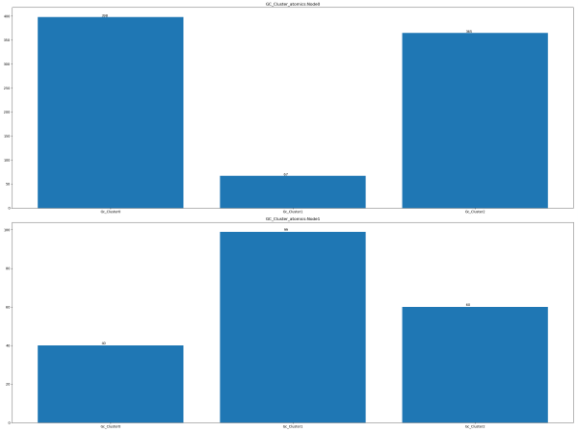


2 Nodes
Graph Scale 5

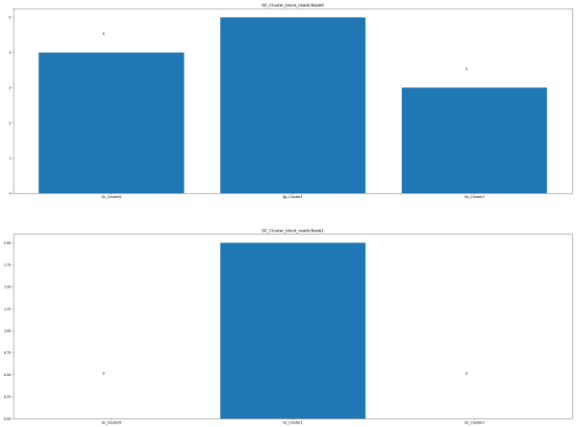
Hybrid BFS: GC Cluster Counters I

2 Nodes, Graph Scale 5

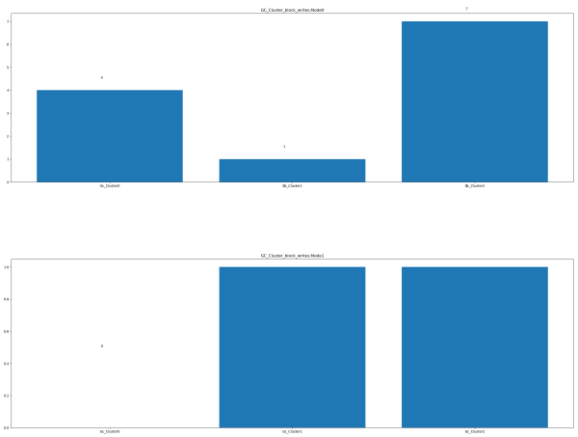
Atomics



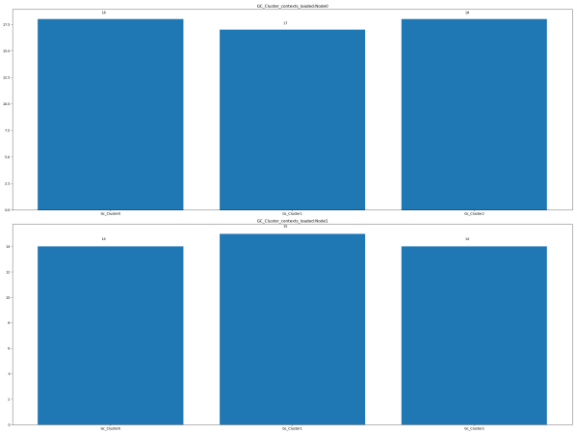
Block Reads



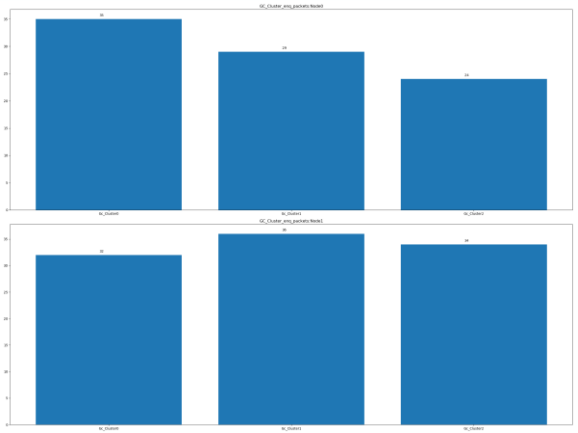
Block Writes



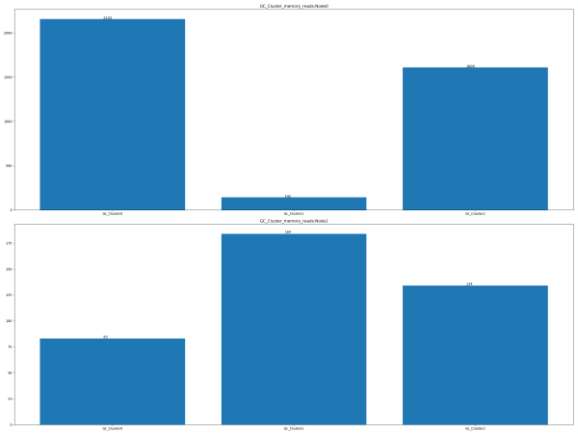
Contexts Loaded



Enq Packets

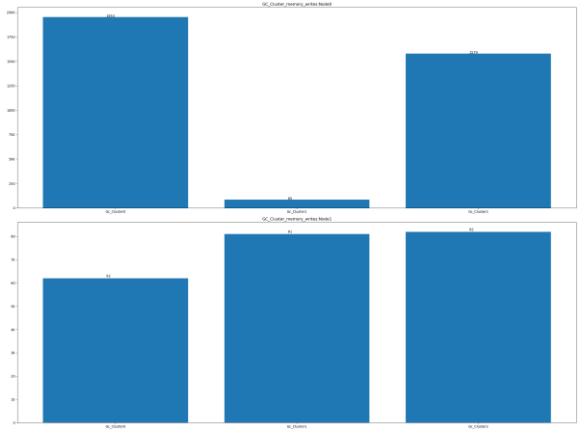


Memory Reads

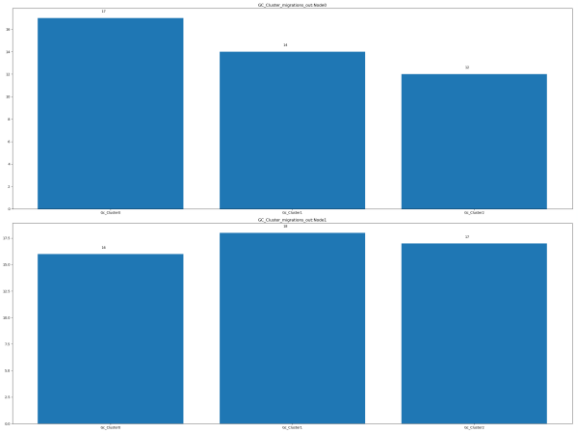


GC Cluster Counters II

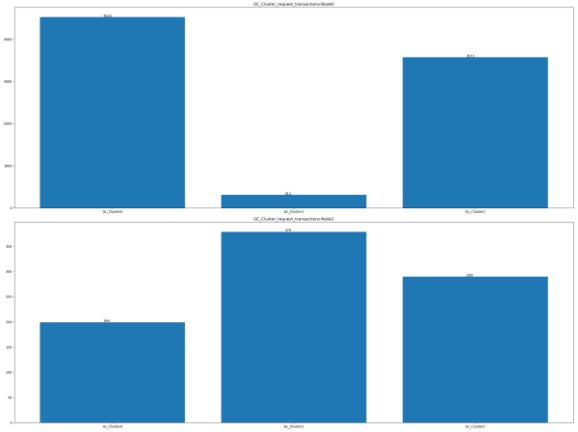
Memory Writes



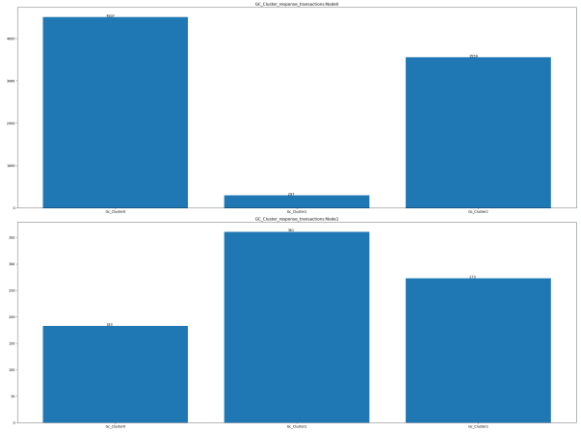
Migrations Out



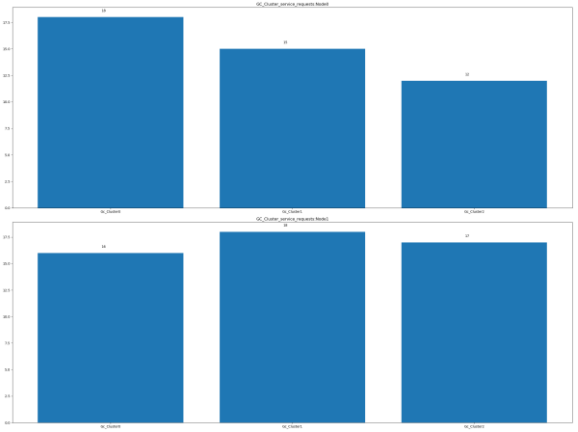
Request Transactions



Response Transactions



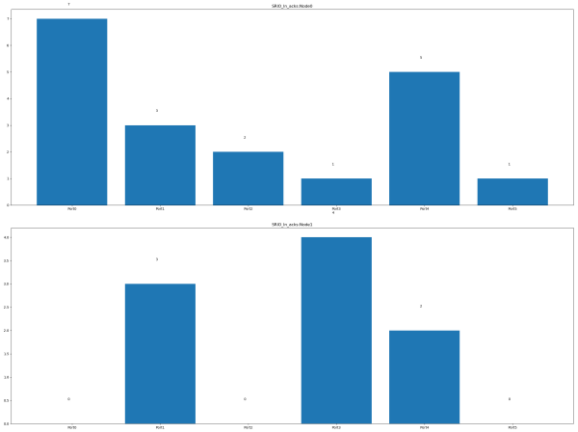
Service Requests



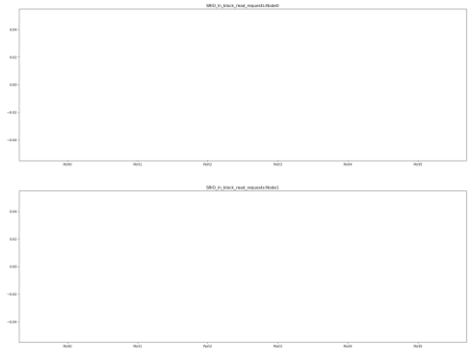
2 Nodes
Graph Scale 5

BFS: SRIO In I

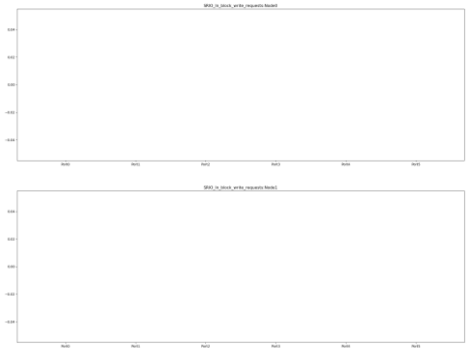
Acks



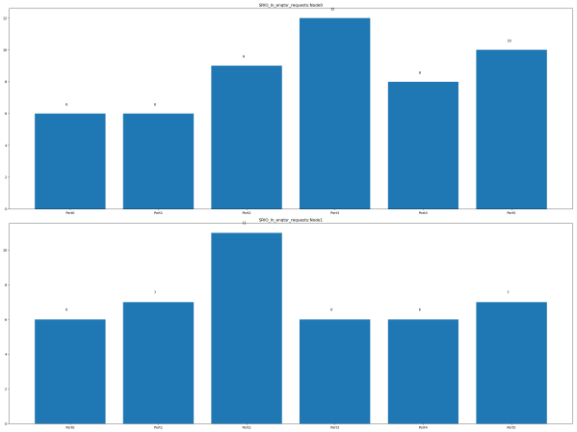
Block Read Requests



Block Write Requests



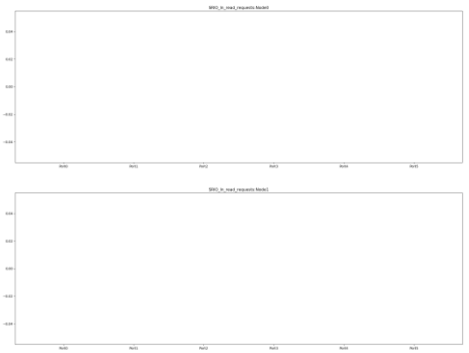
EnqTSR Requests



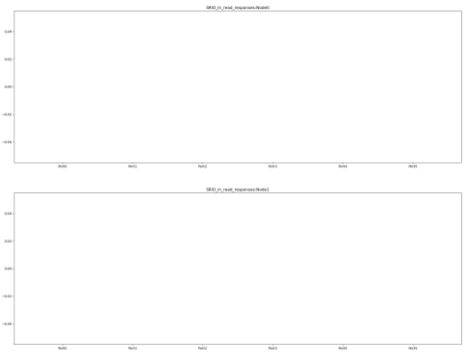
2 Nodes
Graph Scale 5

BFS: SRIO In II

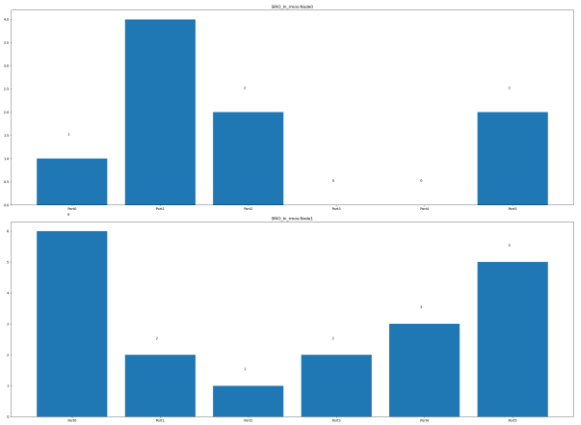
Read Requests



Read Responses



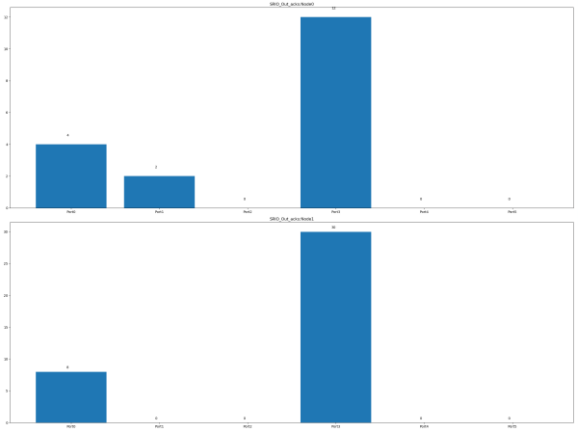
Remotes



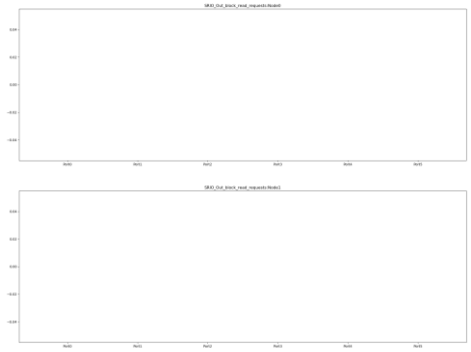
2 Nodes
Graph Scale 5

BFS: SRIO Out I

Acks

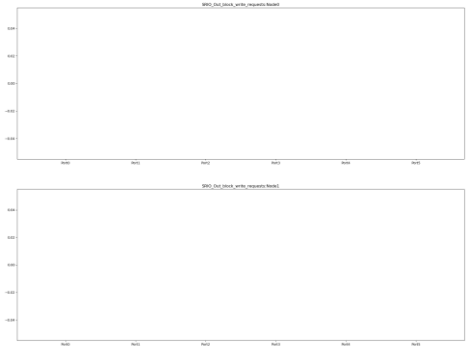


Block Read Requests

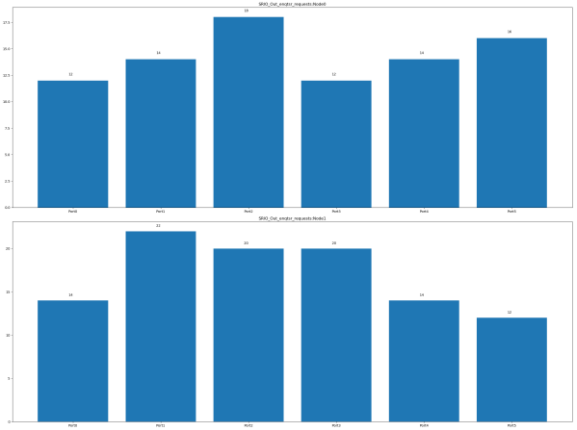


2 Nodes
Graph Scale 5

Block Write Requests

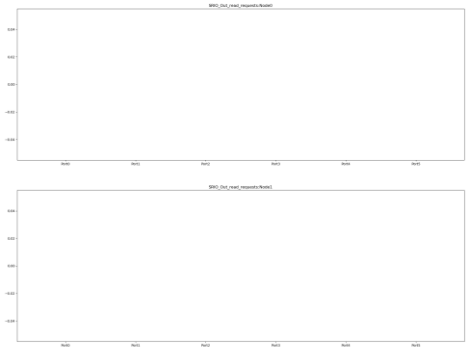


EnqTSR Requests

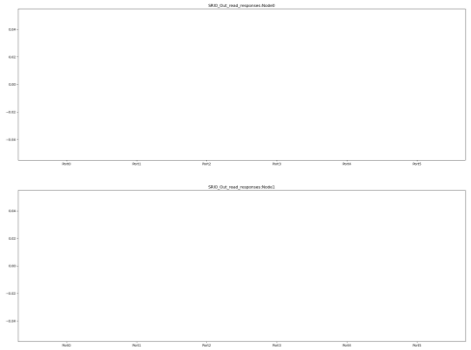


BFS: SRIO Out II

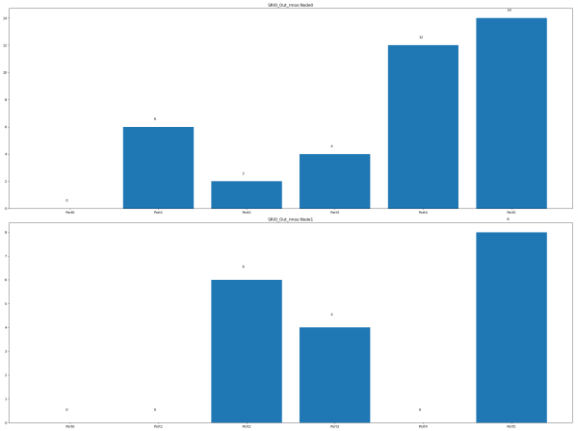
Read Requests



Read Responses



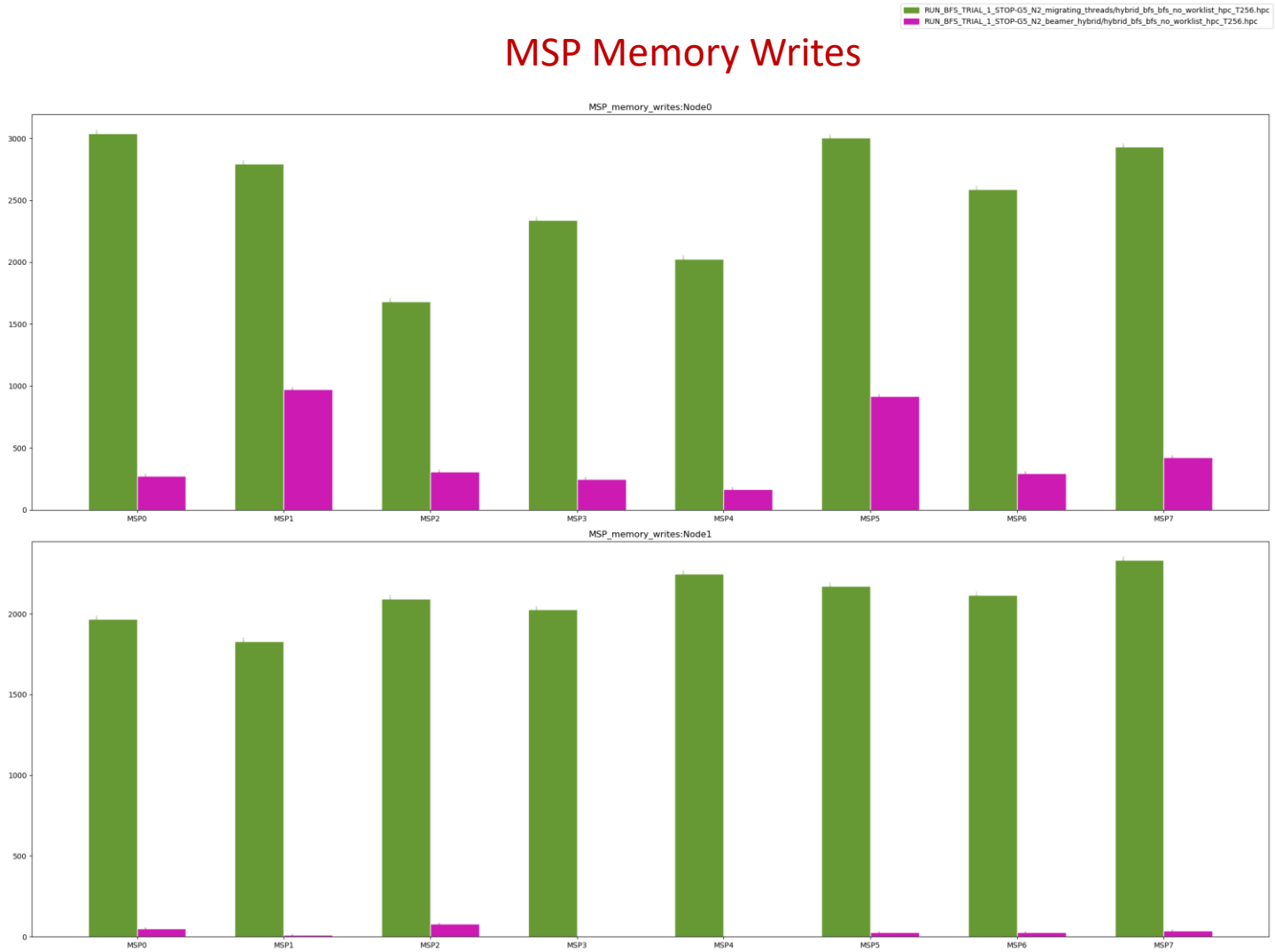
Remotes



2 Nodes
Graph Scale 5

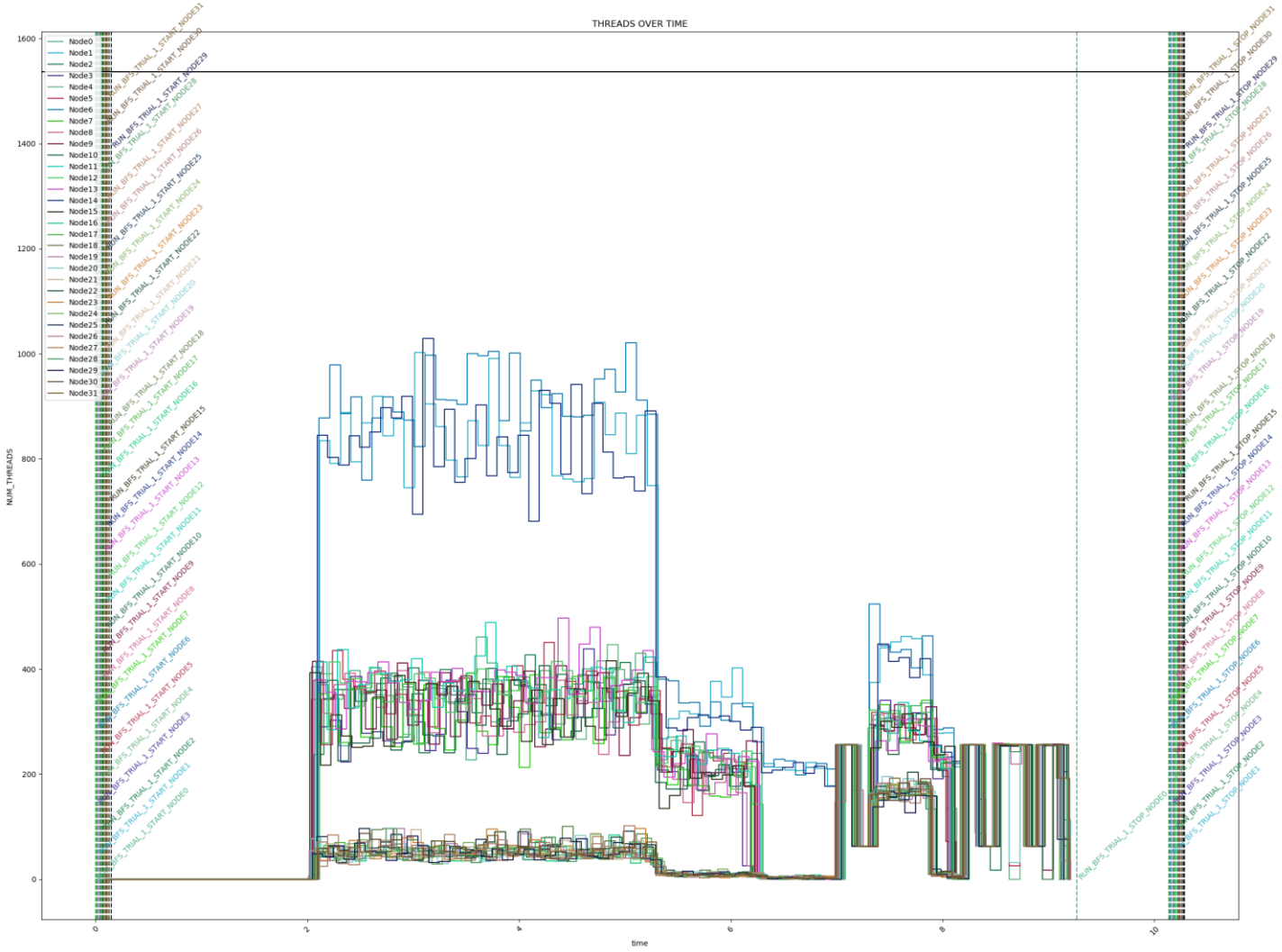
BFS Comparison: Beamer vs. Migrating Threads

MSP Memory Writes



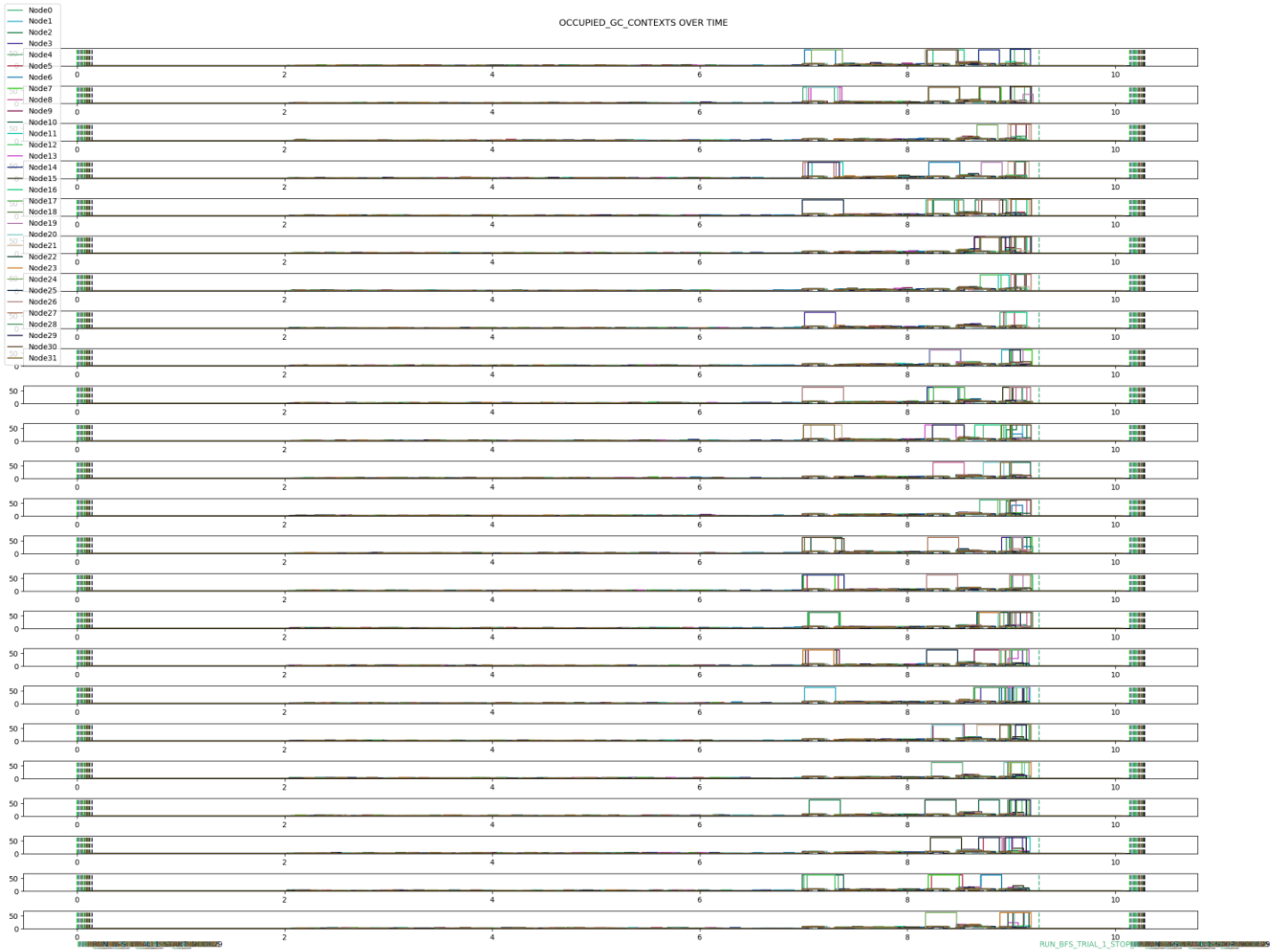
2 Nodes
Graph Scale 5

BFS Polling: Number of Threads over Time



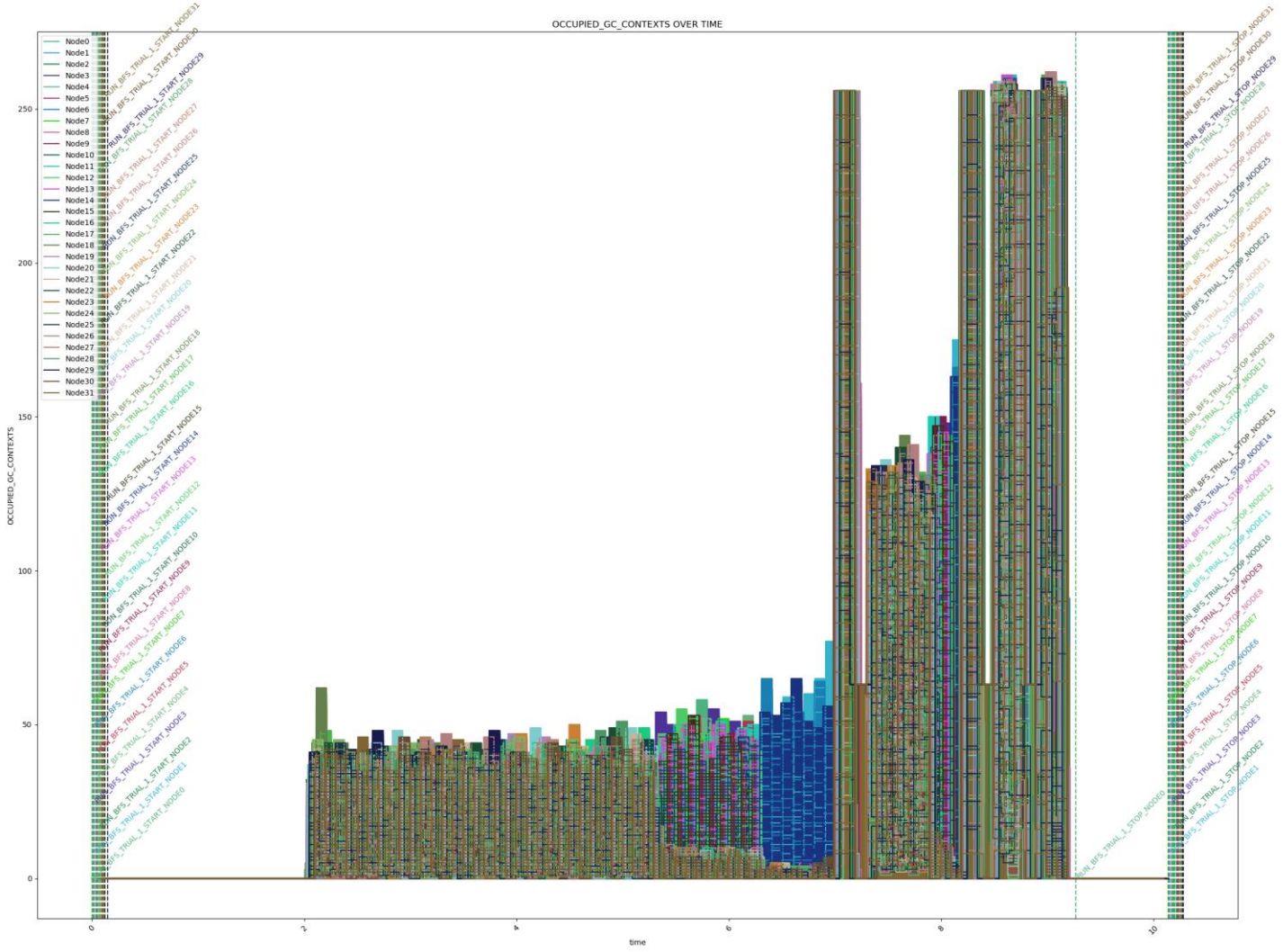
32 Nodes
100 ms
Graph
Scale 28

BFS Polling: GC Contexts over Time



32 Nodes
100 ms
Graph
Scale 28

BFS Polling: GC Contexts Stacked



32 Nodes
100 ms
Graph
Scale 28

Unit Summary: Performance Counters

- Library call inserted in code
- Graphs generated by script
- Example graphs shown

Exercises:
Try reading counters with more complex codes (extra spawns to create failures, extra memory operations)
Try inserting multiple read commands in code
Try comparing versions and running on hardware with polling