StressPP: A Synthetic Workload Generator based on “Stress”

Politecnico di Milano, Advanced Computer Architectures Course

Andrea Guglielmini,
Matr. 893795

Why do we need a workload generator?

A workload generator is an useful tool to test the behaviour of an application while simulating different real-world hardware usages;

We are used to test programs while they are running alone on our computer but this environment is much different from the production one in which a number of other processes, with real workloads, may be running together with our application contending for resources.

This is why it’s useful to test a target program in order to reason about its behaviour and its performance while integrated with possible other noisy neighbour processes.

StressPP

StressPP

StressPP is a synthetic workload generator based on Stress.

It’s main advantages are the usage of *threads* instead of processes and the capability to precisely stress a CPU core.

It’s a flexible and ready to be applicable to real world scenarios.

It can be simply started via *command-line* or by writing your own script using it’s straightforward APIs.

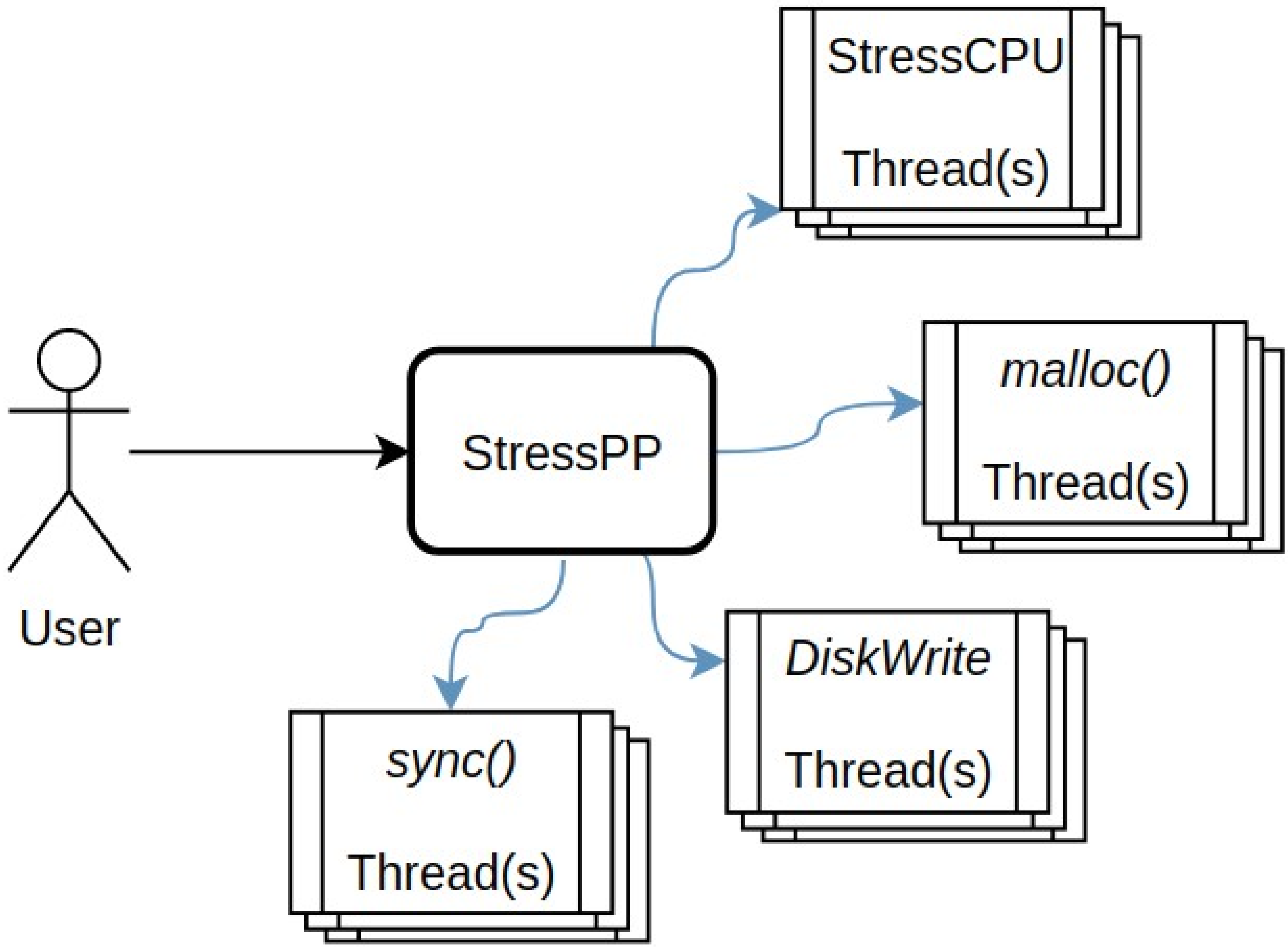
StressPP

It can compose three different types of workloads:

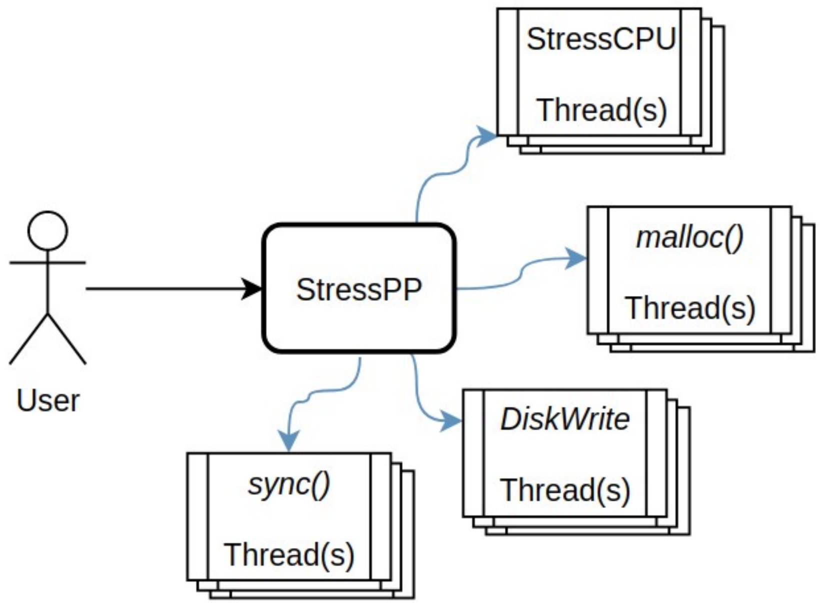
- **CPU Intensive:** issuing floating-point operations to designated CPU Cores.
- **Memory Intensive:** By allocating and random accessing virtual memory chunks.
- **Disk Intensive:** By writing and deleting fictitious files on local drive.
- **I/O Intensive:** By issuing several sync() to the underlying hardware.

The three modules can be executed independently or in chorus.

StressPP: Architecture



StressPP: Architecture



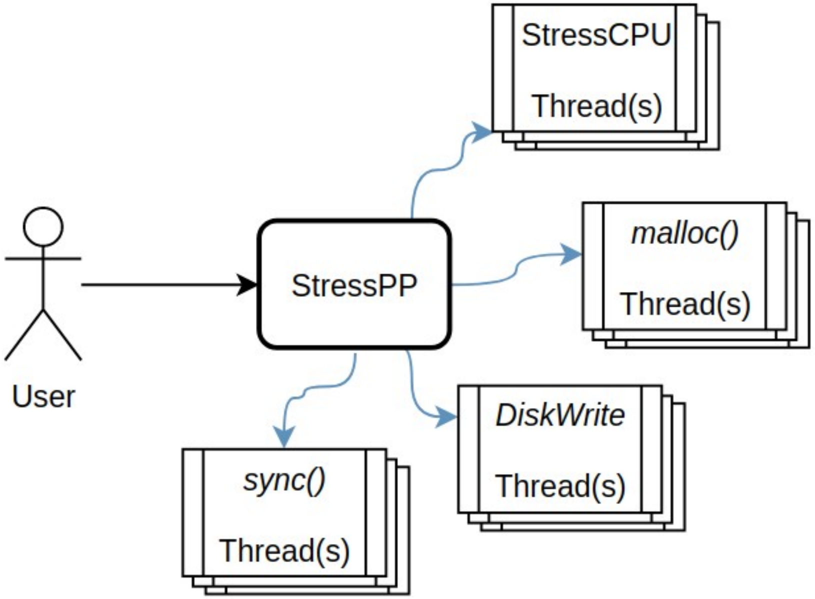
StressPP works by running different threads:

StressCPUThread is a wrapper for a plain *pthread,* it's used in order to easily *start, stop and manage the CPU affinity* of a single system thread.

An instance of *StressCpuThread* is instantiated for any core which the user requires to *stress* with floating point operations.

Optionally, the user may specify a CPU affinity matrix.

StressPP: Architecture



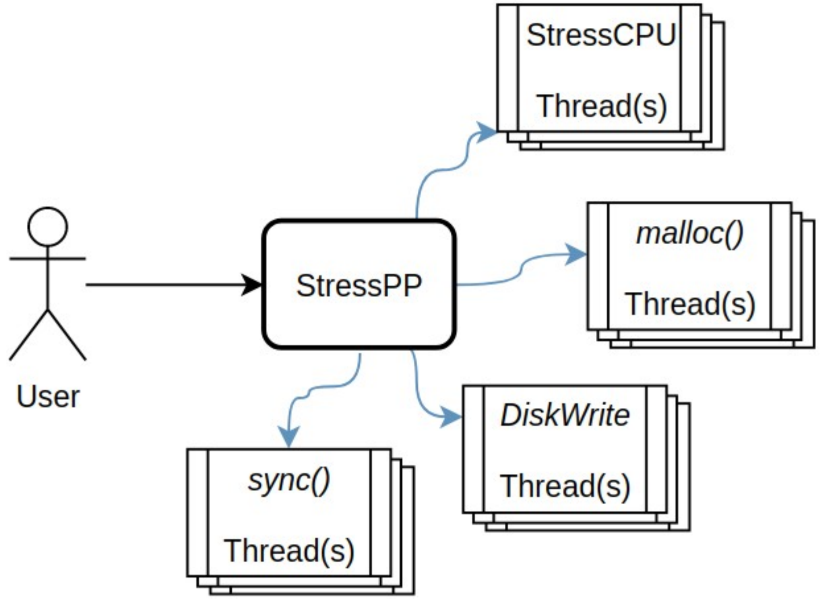
StressPP works by running different threads:

Malloc Threads are used in order to stress the virtual memory of the system.

The user may specify:

- How many threads to launch.
- How many *chunks* try to allocate.
- The size of a single chunk.

StressPP: Architecture



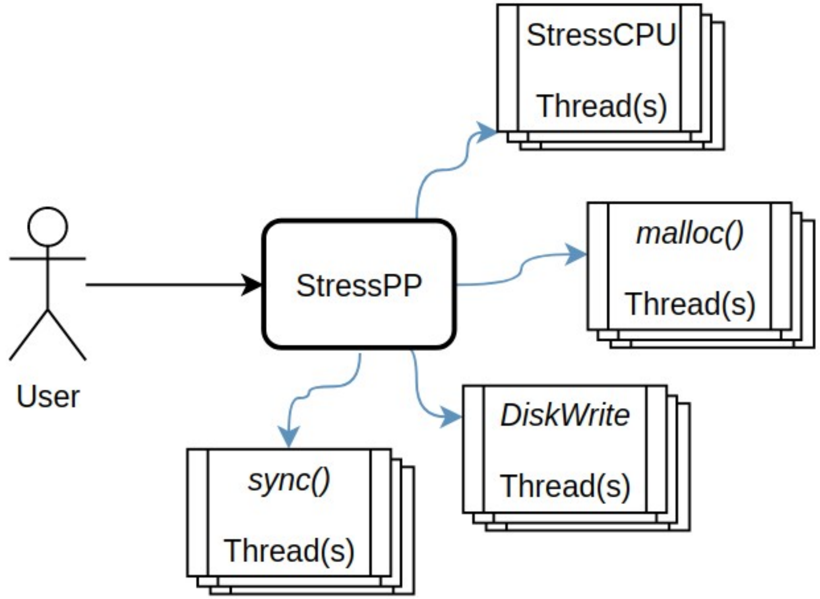
StressPP works by running different threads:

Disk Write Threads are used in order to stress the physical drive of the system.

The user may specify:

- How many threads to launch.
- How many *files* try to allocate.
- The size of a single file.
- If the allocated files needs to be deleted after completion.

StressPP: Architecture



StressPP works by running different threads:

Sync Threads are used in order to stress the I/O subsystem.

The user may just specify how many threads to launch.

StressPP: Validation

Metrics gathered while system in idle

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0,45    0,00    0,35    0,00    0,00   99,20

Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
nvme0n1           1,00         0,00        20,00         0,00          0        100          0
dm-0              5,20         0,00        20,80         0,00          0        104          0
dm-1              0,00         0,00         0,00         0,00          0          0          0
```

```
  1  [|                                                                                     0.7%]
  2  [                                                                                      0.0%]
  3  [||                                                                                    1.3%]
  4  [||                                                                                    2.0%]
  Mem[||||||||||||||||||||||||||||||||||||                                          2.65G/15.4G]
  Swp[                                                                                0K/7.72G]
```

```
 Performance counter stats for 'system wide':

     2.033.097.949      cycles                    #    0,017 GHz
        55.126.405      cache-misses
       820.373.624      instructions              #    0,40  insn per cycle
        120.313,74 msec cpu-clock                 #    4,000 CPUs utilized
            22.942      page-faults               #    0,191 K/sec
           148.889      r01C7                     #    0,001 M/sec

      30,078463556 seconds time elapsed
```

Note: **r01C7** is the number of scalar double precision floating point operations

StressPP: Validation

Metrics gathered while system in idle

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0,45    0,00    0,35    0,00    0,00   99,20

Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
nvme0n1           1,00         0,00        20,00         0,00          0        100          0
dm-0              5,20         0,00        20,80         0,00          0        104          0
dm-1              0,00         0,00         0,00         0,00          0          0          0
```

```
  1  [|                                                                  0.7%]
  2  [                                                                   0.0%]
  3  [||                                                                 1.3%]
  4  [||                                                                 2.0%]
  Mem[||||||||||||||||||||||||||||||||||||                         2.65G/15.4G]
  Swp[                                                                0K/7.72G]
```

```
 Performance counter stats for 'system wide':

     2.033.097.949      cycles                    #    0,017 GHz
        55.126.405      cache-misses
       820.373.624      instructions              #    0,40  insn per cycle
        120.313,74 msec cpu-clock                 #    4,000 CPUs utilized
            22.942      page-faults               #    0,191 K/sec
           148.889      r01C7                     #    0,001 M/sec

      30,078463556 seconds time elapsed
```

Note: No program is actively writing on HDD.

Note: No program is actively using CPU.

Note: No program is actively issuing FLOPs.

Note: There are few page faults.

StressPP: Validation

Stressing CPU: What we expect to see.

We'll run *StressPP* generating a CPU workload, without specifying a CPU affinity matrix.

What we expect?

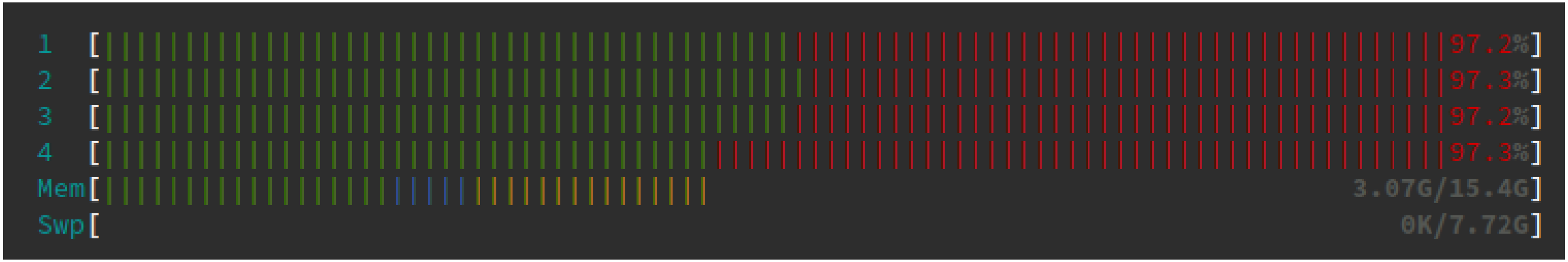
- Every core should get equally loaded.
- Few page faults.
- Increase in executed floating point instructions measured via *perf.*

Which StressPP options are used?

```
./stresspp –c 4
```

StressPP: Validation

Stressing CPU: Validation:




```
Performance counter stats for './stresspp -c 4':

  356.901.771.231      cycles                    #    3,404 GHz
        2.217.526      cache-misses
  134.963.606.884      instructions              #    0,38  insn per cycle
      104.851,34 msec cpu-clock                  #    3,495 CPUs utilized
              156      page-faults               #    0,001 K/sec
      268.543.578      r01C7                     #    2,561 M/sec

     30,002099367 seconds time elapsed
```

VALID: As we expected CPUs are equally loaded, there are few page faults and there's a strong increase in *FLOPs wrt idle case.*

StressPP: Validation

Stressing CPU: Validation **with affinity set on CPUs 0 and 3**

What we expect?

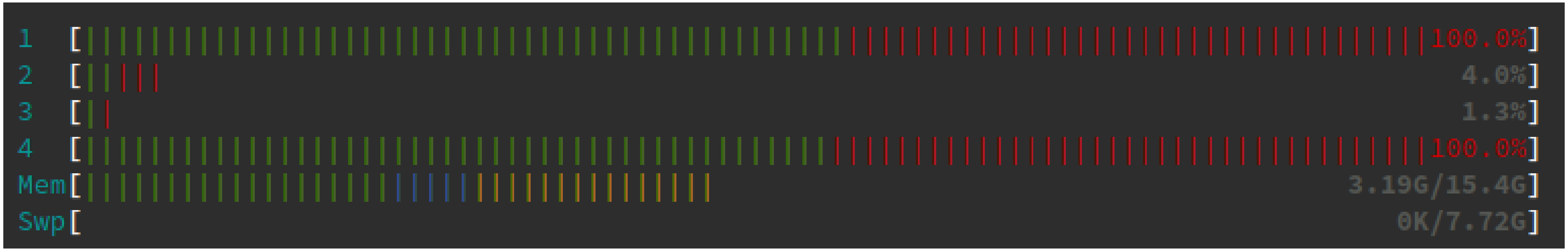
Same results as before but just core 0 and 3 should be loaded.

Which StressPP options are used?

```
./stresspp –cpu-affinity 1000,0001
```

StressPP: Validation

Stressing CPU: Validation **with affinity set on CPUs 0 and 3**




```
Performance counter stats for './stresspp --cpu-affinity 1000,0001':

  209.396.165.980      cycles                    #    3,491 GHz
        1.639.193      cache-misses
   95.476.187.540      instructions              #    0,46  insn per cycle
        59.983,67 msec cpu-clock                 #    1,999 CPUs utilized
              155      page-faults               #    0,003 K/sec
       58.341.963      r01C7                     #    0,973 M/sec

     30,002091388 seconds time elapsed
```

VALID: As we expected just CPUs **0 and 3** are stressed, there are few page faults and there’s a strong increase in *FLOPs wrt idle case.*

StressPP: Validation

Stressing Virtual Memory: What we expect to see.

We'll run *StressPP* generating a Virtual Memory workload by allocating and randomly accessing 24x500Mb chunks via malloc() using two threads.

What we expect?

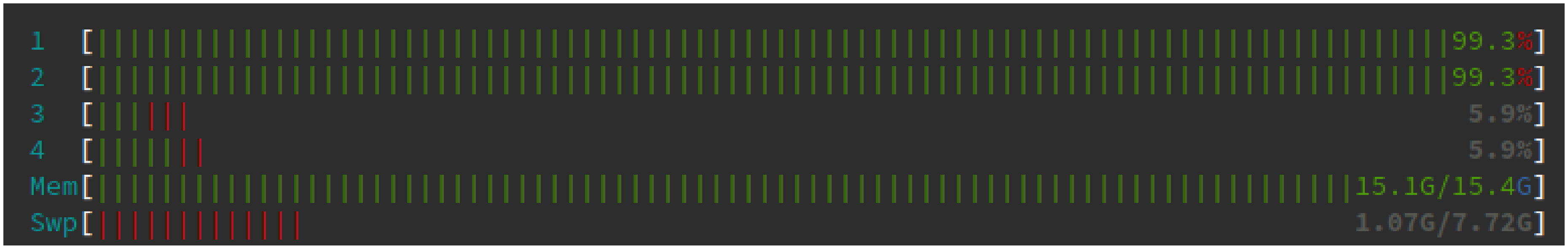
- Two cores should be busy performing memory operations.
- Page faults due to swap.
- High memory allocation.
- High cache misses.

Which StressPP options are used?

```
./stresspp -m 2 --vm-chunks 12 --vm-bytes 536870912
```

StressPP: Validation

Stressing Virtual Memory: Validation.




```
Performance counter stats for './stresspp -m 2 --vm-chunks 12 --vm-bytes 536870912':

  208.969.861.415      cycles                    #    3,490 GHz
    5.765.460.988      cache-misses
   81.158.257.288      instructions              #    0,39  insn per cycle
       59.883,06 msec cpu-clock                  #    1,965 CPUs utilized
        3.146.112      page-faults               #    0,053 M/sec
                0      r01C7                     #    0,000 K/sec

     30,471185401 seconds time elapsed
```

VALID: As we expected 2 CPUs are busy, *page-faults* as well as *cache-misses* are increased.

StressPP: Validation

Stressing Virtual Memory: Random Access Pattern.

In order to prove that the random access pattern is quite effective, the same experiment is performed but using a linear access pattern to read the allocated chunks

```
Performance counter stats for './stresspp -m 2 --vm-chunks 12 --vm-bytes 536870912':

  198.546.072.035      cycles                    #    3,473 GHz
      825.979.136      cache-misses
  199.168.523.747      instructions              #    1,00  insn per cycle
        57.166,55 msec cpu-clock                 #    1,878 CPUs utilized
        3.146.110      page-faults               #    0,055 M/sec
                0      r01C7                     #    0,000 K/sec

     30,442290875 seconds time elapsed
```

VALID: The *cache-misses* counter using a linear access patter is about 7x less than the one achieved with a *random* access.

StressPP: Validation

Stressing Disk: What we expect to see.

We'll run *StressPP* continuously writing and deleting 1Gig file on hardisk using a single thread.

What we expect?

- A single core should be busy with memory operations.
- High disk usage.

Which StressPP options are used?

```
./stresspp -d 1
```

StressPP: Validation

Stressing Disk: Validation.

```
1   [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
2   [|||||                                                                                 4.7%]
3   [||||||||                                                                              8.4%]
4   [||||                                                                                  3.4%]
Mem[|||||||||||||||||||||||||||                                                      3.18G/15.4G]
Swp[|||||||||||                                                                      1.03G/7.72G]
```

```
  104.528.446.847      cycles                    #    3,488 GHz
    1.419.306.550      cache-misses
  143.560.850.735      instructions              #    1,37  insn per cycle
        29.966,59 msec cpu-clock                 #    0,996 CPUs utilized
              411      page-faults               #    0,014 K/sec
                0      r01C7                     #    0,000 K/sec

     30,087785071 seconds time elapsed
```

```
Total DISK READ :       0.00 B/s | Total DISK WRITE :    1731.79 M/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:       0.00 B/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
12464 be/4 root        0.00 B/s  732.48 M/s  0.00 %  0.00 % ./stresspp -d 1
    1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % init
    2 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
```

VALID: As we expected there’s a single core busy in writing while the disk is written at *700Mb/s*

StressPP: Validation

Stressing I/O: What we expect to see.

We’ll run *StressPP* continuously asking to sync() using a single thread.

What we expect?

- High I/O requests.
- Low CPU usage since there are no operations in charge of the processor.

Which StressPP options are used?

```
./stresspp -i 1
```

StressPP: Validation

Stressing I/O: What we expect to see.

```
1  [||||||||||||||                                      14.2%]
2  [||||||||||||                                        12.1%]
3  [||||||||||||||||                                    16.3%]
4  [||||||||||||||||                                    16.5%]
Mem[||||||||||||||||||||||||||||                   3.49G/15.4G]
Swp[|||||||||||                                     999M/7.72G]
```

```
Performance counter stats for './stresspp -i 1':

   19.283.335.563      cycles                    #    2,956 GHz
       16.026.653      cache-misses
   22.322.442.337      instructions              #    1,16  insn per cycle
         6.524,10 msec cpu-clock                 #    0,217 CPUs utilized
              154      page-faults               #    0,024 K/sec
                0      r01C7                     #    0,000 K/sec

     30,002050080 seconds time elapsed
```

```
Total DISK READ :       0.00 B/s | Total DISK WRITE :      15.66 K/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:      31.32 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
16032 be/4 root        0.00 B/s    0.00 B/s  0.00 % 65.10 % ./stresspp -i 1
  312 be/3 root        0.00 B/s    3.91 K/s  0.00 %  0.46 % [jbd2/dm-0-8]
```

VALID: As we expected there’s no busy core and *stress*pp is actually the most I/O hungry program.

Thank you.

https://github.com/Guglio95/StressPP