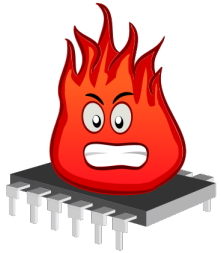


stress-ng

Finding kernel bugs through stress testing (a software hammer for kernels and hardware)

Kernel Recipes

Why do stress testing?

- Find breakage points (kernel panics, races, lock-ups...)
- Check for correct behaviour under stress
- Test modes of failure (e.g. what happens on low memory?)
- Test for stable behaviour outside of expected usage
- Exercise scaling/load (CPUs, memory, I/O) – does it scale well?
- Burn-in testing (e.g. detecting CPU / disk / memory errors)

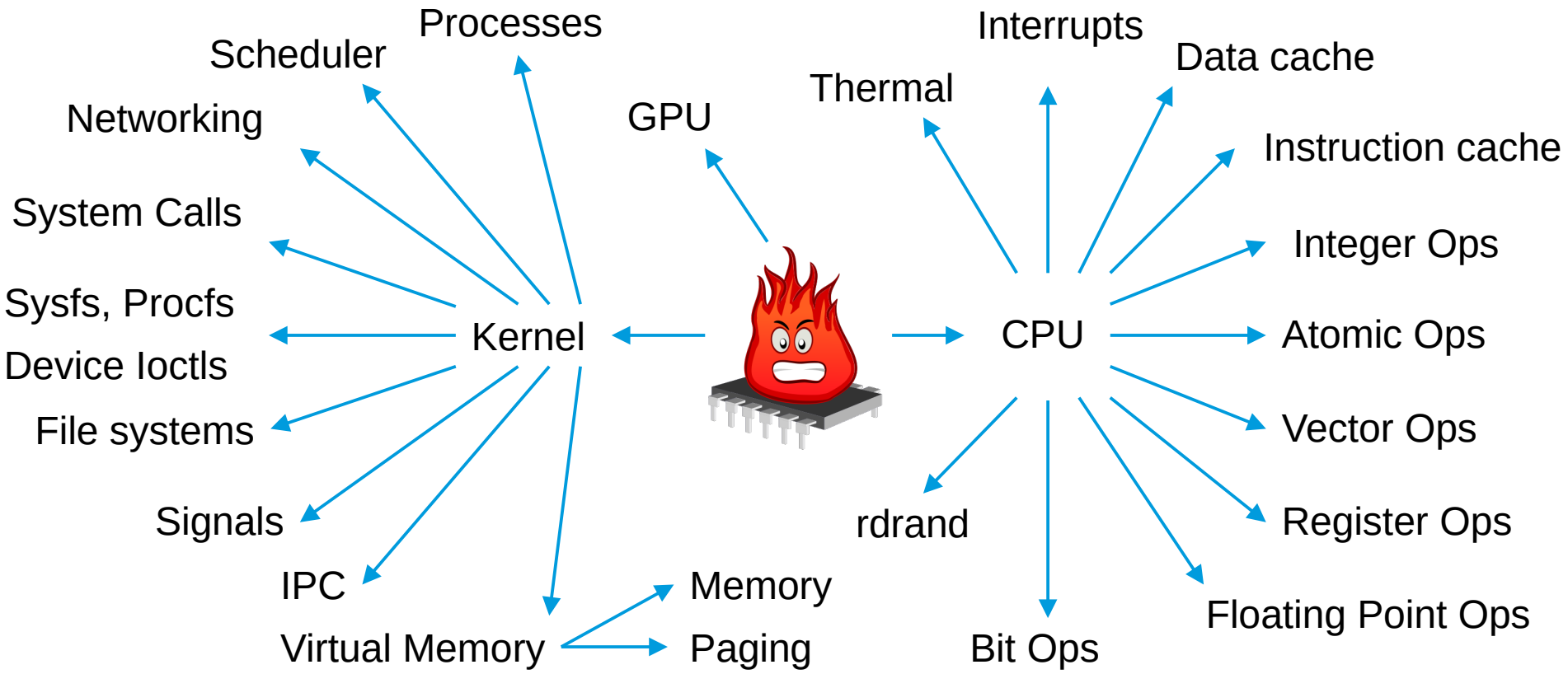
Why use Stress-ng?

- Already found 60+ kernel bugs
- ~20 kernel performance improvements
- Kernel 0-day performance testing
- Used by silicon vendors (new silicon + kernel bring-up)
- Used for kernel regression testing (e.g. Ubuntu kernel)
- Used in stress testing server and cloud environments
- Cited in 80+ academic research papers - synthetic stress testing
- LKP-tests (Linux kernel performance test tool)

Stress-ng, 10 years ago..

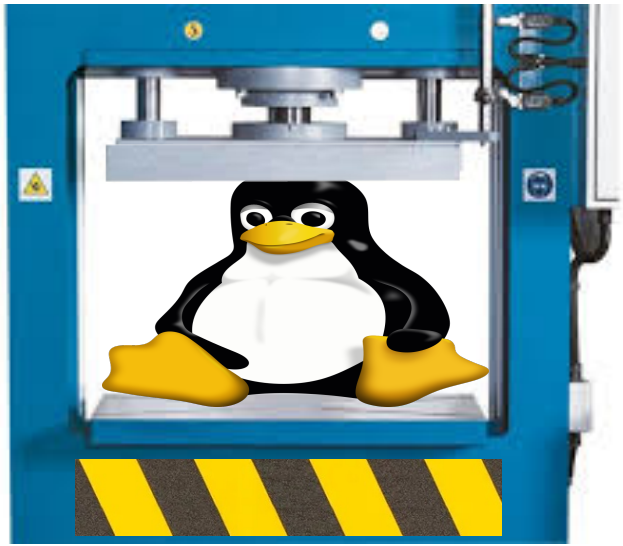
- Stress Laptops, Thermal Overrun
- Simple stress tests (stressors)
- Compatible with the ‘stress’ tool
- Exercised Intel thermal daemon
- Ubuntu Laptop enablement

Stress-ng in 2023, 300+ stressors



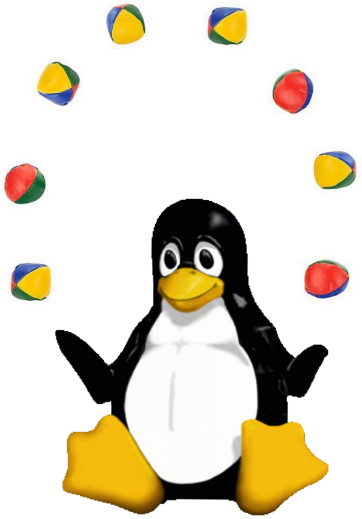
Stress-ng vs Kernels

Pressure stressors



Repeated hammering

Juggling resources

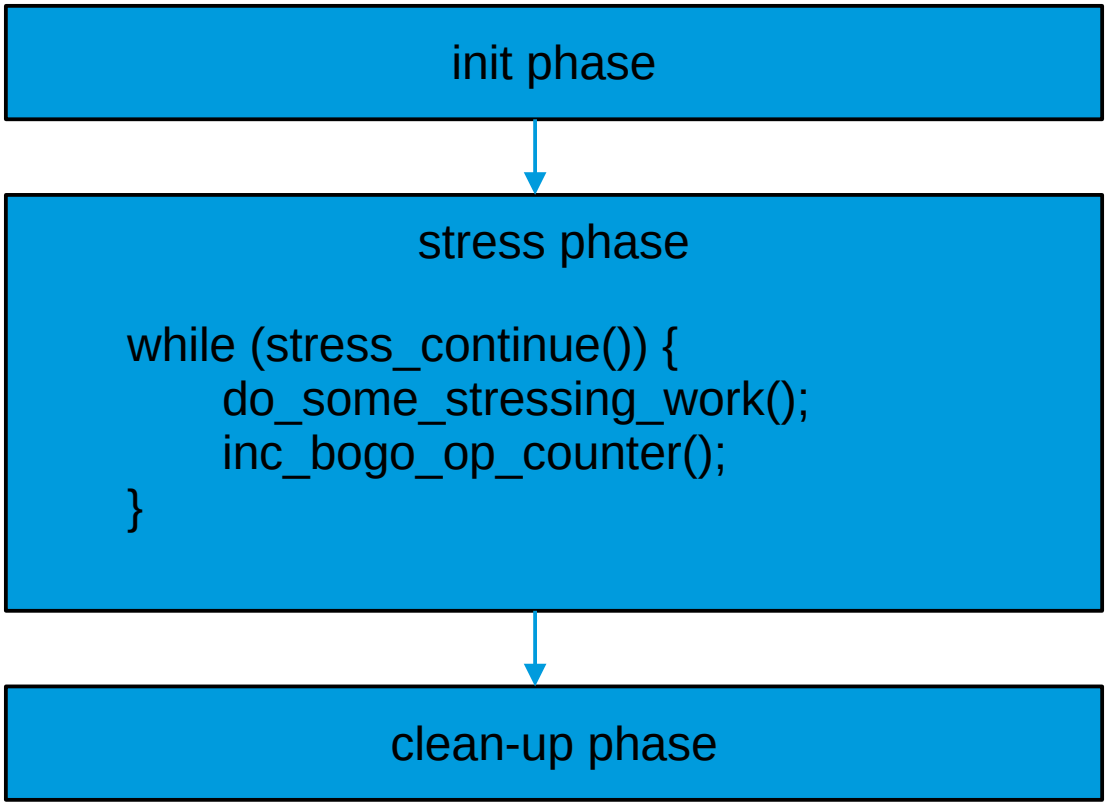


What is a Stressor?

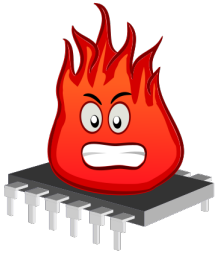
Normally a single process forked from stress-ng

Stressor may be one or more child process or one or more pthreads in more complex stress cases.

Stressor terminates on SIGALRM or reached maximum bogo-op count



Stress-ng options



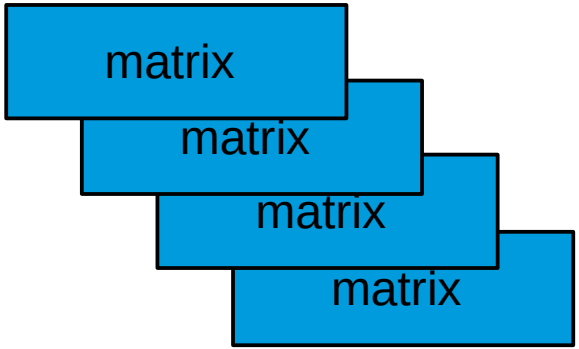
- Stressor options
  - Number of instances
  - Optional loop iterations (bogo-ops)
  - Optional per-stressor extra options
- Global options
  - Run duration (--timeout, -t)
  - Verify mode (--verify)
  - Performance Metrics (--metrics)
  - Logging (--log-file filename)
  - Perf Events (--perf)
  - ..and many more!

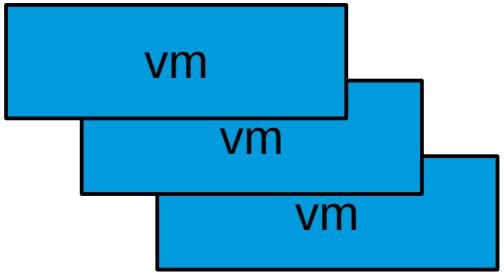
stress-ng --mmap 4 --mmap-ops 10000 --verify --metrics

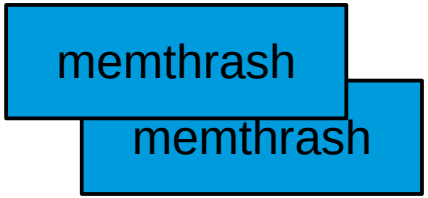
Running multiple stressors in parallel

stress-ng --matrix 4 --vm 3 --memthrash 2 --timeout 1m

4 instances of matrix stressor, 3 instances of vm stressor, 2 instances of memthrash stressor all running in parallel for 1 minute







Stressing CPUs

stress-ng --matrix 8 --timeout 5m --thermalstat 1

8 instances of matrix stressor, run for 5 minutes and print thermal statistics every second (good mix of cache + compute = toasty silicon)

stress-ng --vecmath 2 --fp 2 --cpu 4 -t 200 --tz

2 instances of vector math stressor, 2 instances of floating point stressor, 4 instances of CPU stressor, run for 200 seconds, print thermal zone information at the end

and also:

af-alg, atomic, branch, bsearch, cache, cacheline, context, cpu, crypt, dekker, eigen, far-branch, flush-cache, fp, goto, hash, heapsort...

Stressing Memory

stress-ng --vm 0 --verify --vmstat 60 -t 1h

vm stressor run on all online CPUs, verification enabled, show vmstat stats every minute, soak test for 1 hour

stress-ng --memrate 1 -t 1m

benchmark memory read/write rates with various sized read/writes for 1 minute

stress-ng --brk 0 --stack 0 --bigheap 0 --oom-pipe -t 15m

consume memory, force low memory OOM scenarios

Stressing Networking

stress-ng --udp 1 --udp-port 2000

udp stressor (client/server send/recv) on port 2000, 1 instance

stress-ng --sock 4 --sock-domain ipv6 --sock-if lo --sock-port 9000 --sock-protocol tcp --sock-type stream --sock-zerocopy -t 1h

tcp ipv6 stream test on loopback, try to use zerocopy on port 9000

and also:

dccp, netdev, netlink-proc, netlink-task, ping-sock, rawsock, rawpkt, rawudp, sctp, sockabuse, sockfd, sockmany, tun, udp-flood

Stressing File Systems

stress-ng --iomix 10 --smart --verify -t 1h --temp-path /mnt/test

10 instances of mixed I/O operations, enable S.M.A.R.T. checks with I/O test verification, 1 hour soak test on filesystem on /mnt/test

stress-ng --revio 1 --seek 1 --verify -t 1d

1 reverse I/O stressor (creates lots of extents) and 1 random seek stressor, enable verification, soak test for 1 day

and also:

access, aio, aiol, chattr, chdir, chmod, chown, copy-file, dentry, dir, dirdeep, dirmany, fallocate, fiemap, file-ioctl, filename, flock, fsize, fstat, getdent, hdd, ioprio, lease, ramfs, readahead, rename, seal, tmpfs...

Stressing Kernel Interfaces

sudo stress-ng --sysfs 4 --procfs 4 --dev 4

traverse and exercise sysfs and procfs, exercise device ioctls

stress-ng --enosys 0 --sysinval 0 --vdso 0 --x86syscall 0

exercise non-existent system call numbers, exercise invalid system call argument passing (syzkaller super-lite), exercise vdso system calls, x86 system call mechanism

-ETOOMUCH Stress

Deep breath….

Over 300 stressors!

I cannot cover all of them in a short presentation.

I cannot cover all the 900+ options.

Please refer to the manual before asking if there is a stressor for a specific test case :-)

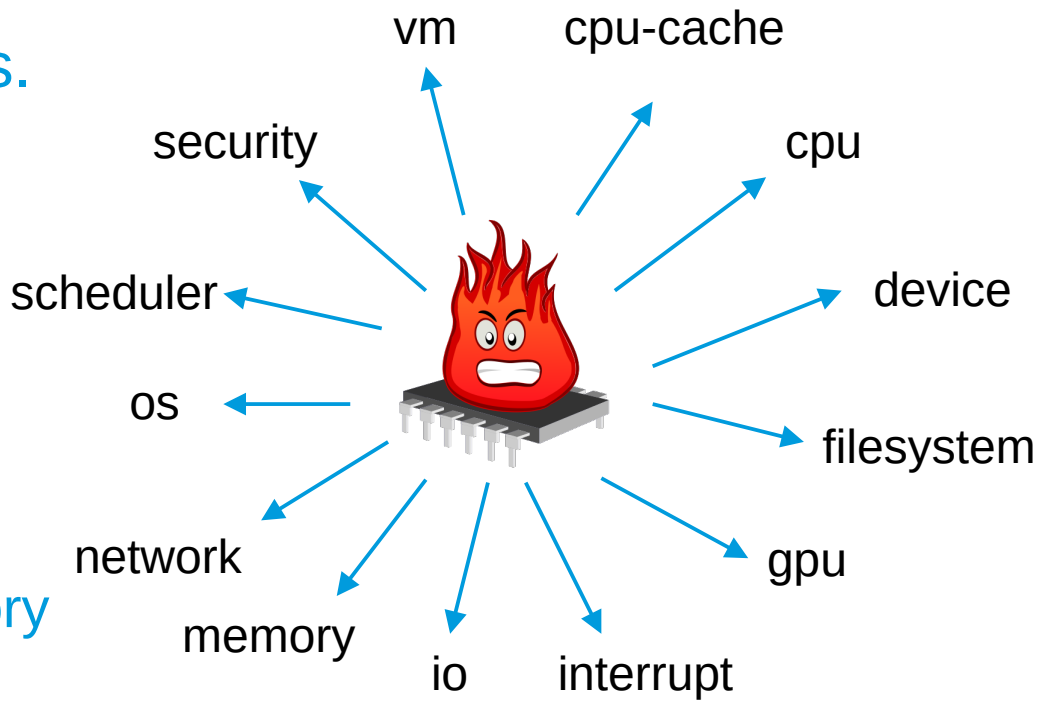
Stressor classes

Stressors are grouped into classes.

A class has one or more related stressors.
A stressor can be in one or more classes.

stress-ng --class vm -t 1m --seq 8

run all stressors in the virtual memory class one after each other for 1 minute with 8 instances per stressor.



Running multiple stressors sequentially

stress-ng --seq 2 --class network -t 1m

run all the network related stressors one after another for 1 minute each, each stressor is run with 2 instances running in parallel

stress-ng --seq 8 --with vm,cache,memthrash,mmap -t 1m

run each stressor one after another for 1 minute each, each stressor is run with 8 instances running in parallel

Running permutations of stressors

stress-ng --perm 1 --class scheduler -t 1m

run permutations of all the scheduler related stressors one after another for 1 minute each, one instance of each stressor.

stress-ng --perm 8 --with brk,bigheap,stack -t 2m

run permutations of stressors one after another for 2 minutes each, each stressor is run with 8 instances running in parallel. E.g. brk, brk + bigheap, bigheap, stack, brk + stack, bigheap + stack, brk + bigheap + stack.

Stressor Methods

stress-ng --vm 1 --vm-method flip --vm-bytes 90% --verify

execise 90% of available virtual memory using bit-flipping & verification

stress-ng --cpu 0 --cpu-method div64 --verify

exercise CPUs with 64 bit integer division operations

stress-ng --memthrash 1 --memthrash-method spinwrite

thrash memory with random spin-looped writes

by default, stressors with method options will run sequentially through all their stressing methods

Useful extra options

--verify	enable sanity checking (slows down stressors)
--oom-avoid	try to avoid out-of-memory kills
--klog-check	check for kernel crash messages
--no-rand-seed	use same random seed for test repeatability
--exclude list	exclude stressors (useful for --class options)
--ignite-cpu	try to make CPU extra toasty (need root privs)
--oomable	do not restart an OOM'd stressor
--taskset list	pin stressors to specific CPUs

Micro benchmarking

- Bogo-ops/sec and metrics can be useful for micro benchmarking specific use-cases. Use --metrics option.
- Performance regression testing. Use same version of stress-ng!

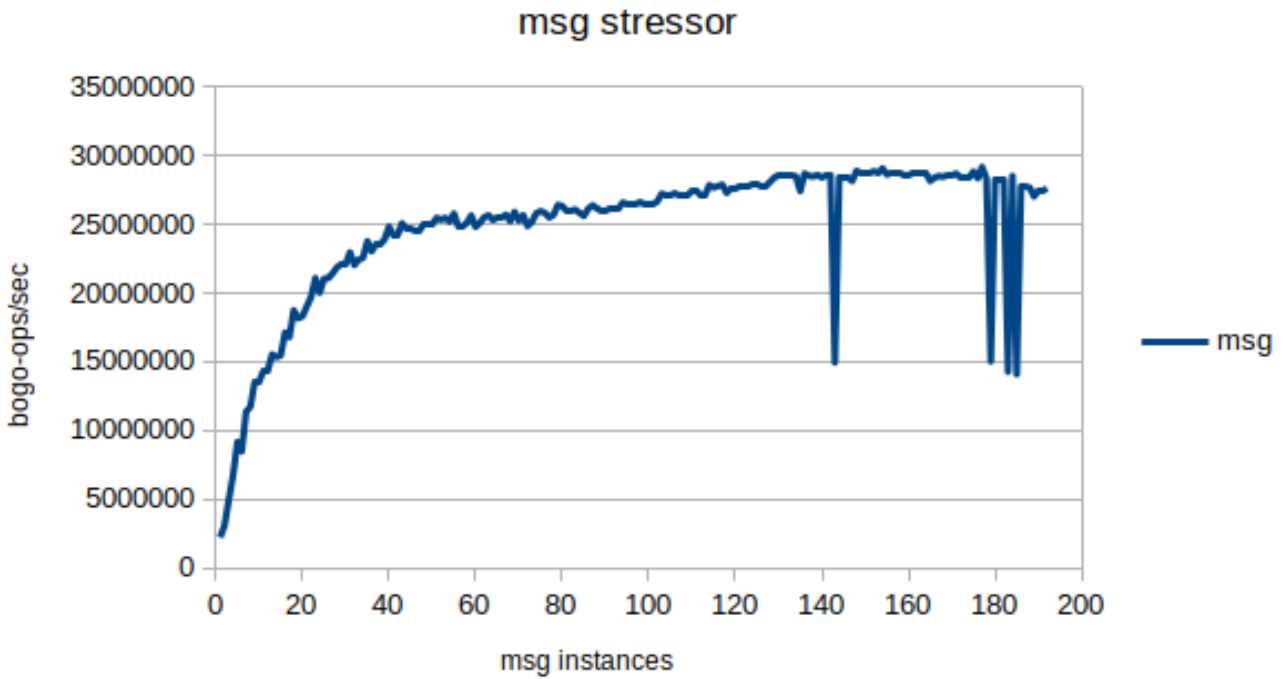
```
stress-ng: info:  [2701103] setting to a 10 secs run per stressor
stress-ng: info:  [2701103] dispatching hogs: 1 mlock
stress-ng: metrc: [2701103] stressor       bogo ops real time  usr time  sys time   bogo ops/s     bogo ops/s CPU used per
stress-ng: metrc: [2701103]                           (secs)    (secs)    (secs)   (real time) (usr+sys time) instance (%)
stress-ng: metrc: [2701103] mlock             50186     10.56      0.22     10.28      4751.66        4776.81        99.47
stress-ng: metrc: [2701103] miscellaneous metrics:
stress-ng: metrc: [2701103] mlock              3994.48 nanosecs per mlock call (geometric mean of 1 instances)
stress-ng: info:  [2701103] skipped: 0
stress-ng: info:  [2701103] passed: 1: mlock (1)
stress-ng: info:  [2701103] failed: 0
stress-ng: info:  [2701103] metrics untrustworthy: 0
stress-ng: info:  [2701103] successful run completed in 10.56 secs
```

Perf events

- Perf events can be useful for checking CPU and kernel utilization with the --perf option (use sudo to see more events)

```
cking@t480:~/repos/stress-ng$ stress-ng --cpu 1 --cpu-method fft --perf -t 10
stress-ng: info:  [3786978] setting to a 10 second run per stressor
stress-ng: info:  [3786978] dispatching hogs: 1 cpu
stress-ng: info:  [3786978] cpu:
stress-ng: info:  [3786978]            35,516,321,739 CPU Cycles                     3.551 B/sec
stress-ng: info:  [3786978]           104,677,138,708 Instructions                  10.466 B/sec (2.947 instr. per cycle)
stress-ng: info:  [3786978]            13,178,827,724 Branch Instructions            1.318 B/sec
stress-ng: info:  [3786978]                16,057,660 Branch Misses                  1.605 M/sec ( 0.122%)
stress-ng: info:  [3786978]               237,977,421 Bus Cycles                    23.793 M/sec
stress-ng: info:  [3786978]            18,801,302,438 Total Cycles                   1.880 B/sec
stress-ng: info:  [3786978]                21,622,769 Cache References               2.162 M/sec
stress-ng: info:  [3786978]                 4,594,099 Cache Misses                   0.459 M/sec (21.247%)
stress-ng: info:  [3786978]            24,607,608,140 Cache L1D Read                 2.460 B/sec
stress-ng: info:  [3786978]             1,775,541,544 Cache L1D Read Miss            0.178 B/sec ( 7.215%)
```

Does it scale?



Does stress performance scale with number of instances?

How to build

git clone https://github.com/ColinIanKing/stress-ng

… install any dependencies (see the README.md file)

cd stress-ng

make clean && make -j $(nproc)

make pdf

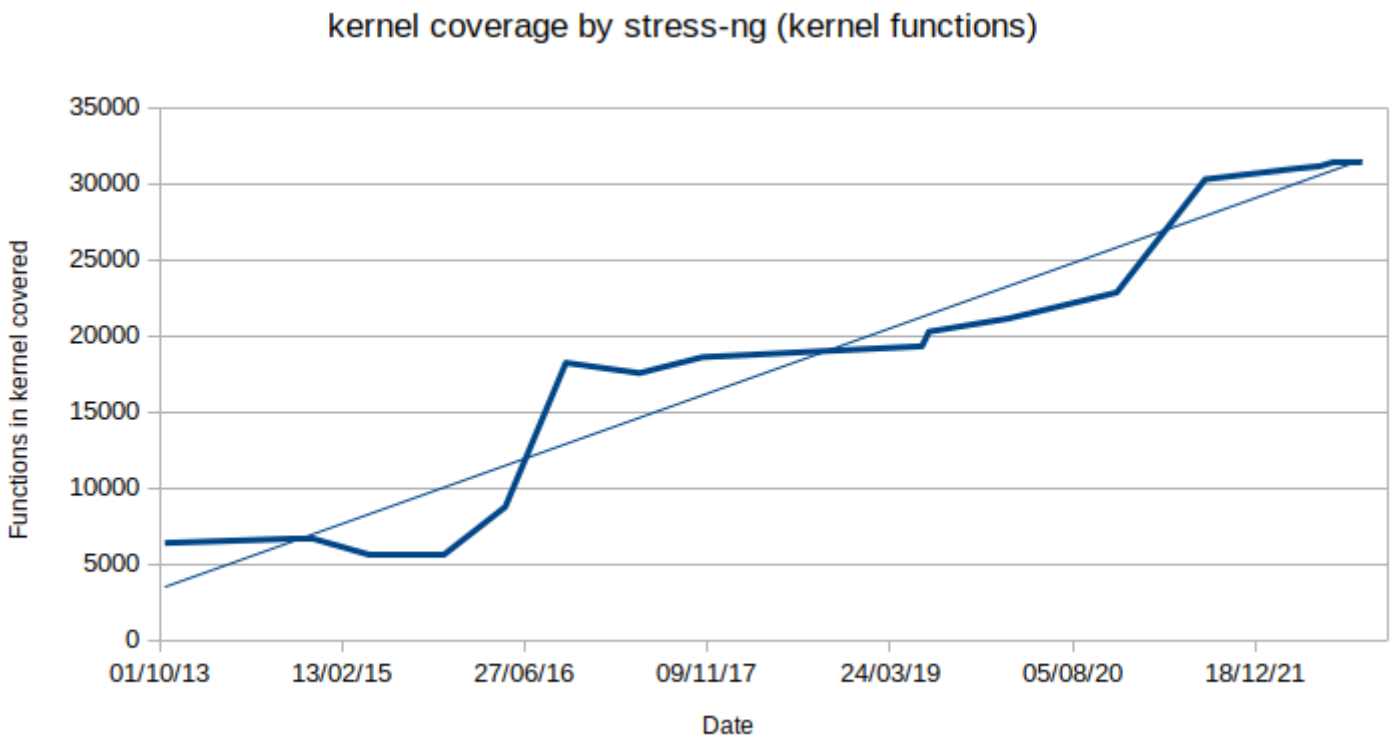
..or install using your favourite distro (maybe old or out of date)

..or use the docker image on the github project page

What drives stress-ng development?

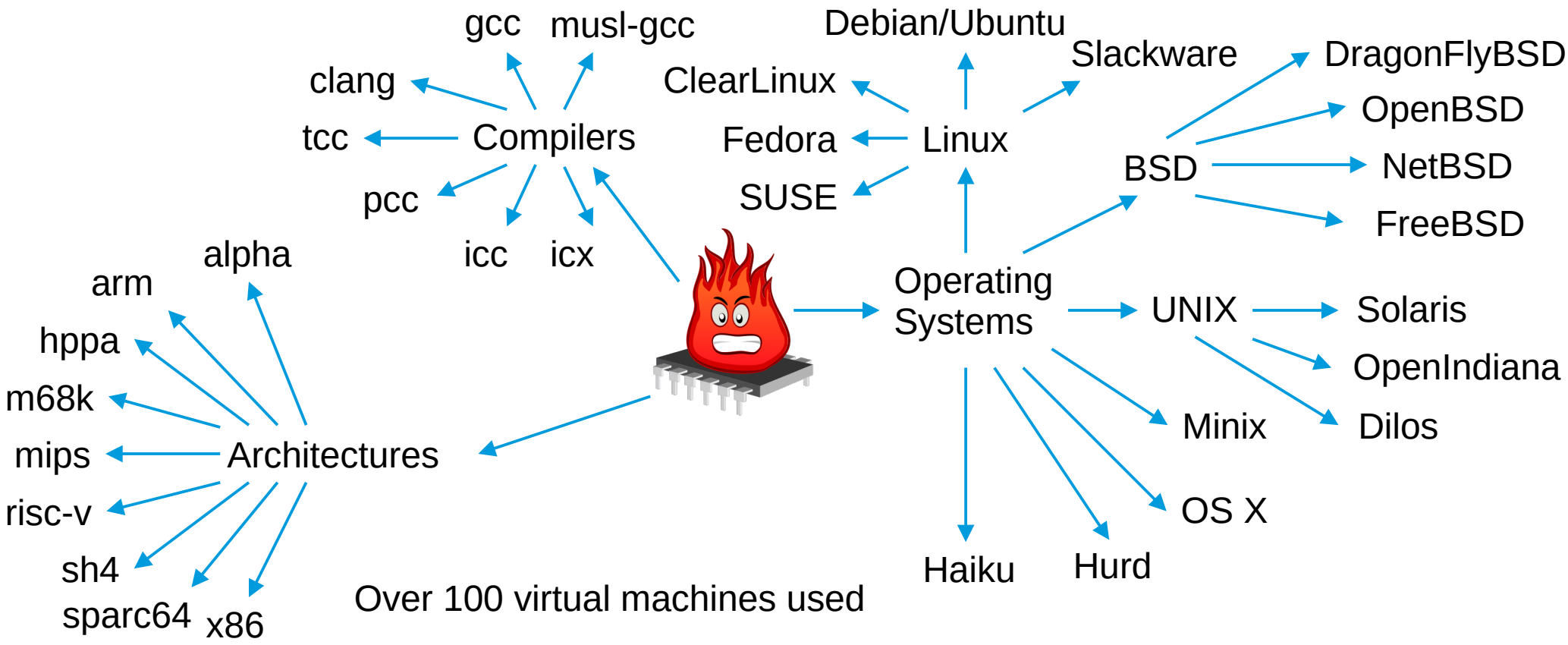
- New kernel features (system calls, ioctls, sysfs/procfs, devices)
- Kernel gcov coverage holes (checked on each new kernel)
  - Directed coverage testing, another never ending task!
- New processor features
- New architectures
- Kernel bugs (implement some reproducers)
- User requests or user provided stressors
  - Contributions always welcome!

Kernel Test Coverage



Dates not to scale

Portability – Release Testing



Find out more

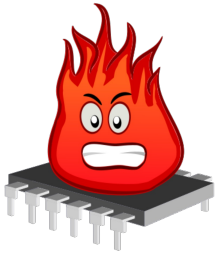
Read the manual (man page), ‘make pdf’ to make PDF version

- Plenty of per-stressor information
- About 90 pages – a lot of options!
- Future work: write a quick start man page

Quick start Reference Guide:

https://wiki.ubuntu.com/Kernel/Reference/stress-ng

Project Information + Questions



github.com/ColinIanKing/stress-ng

email: colin.i.king@gmail.com

Any Questions?