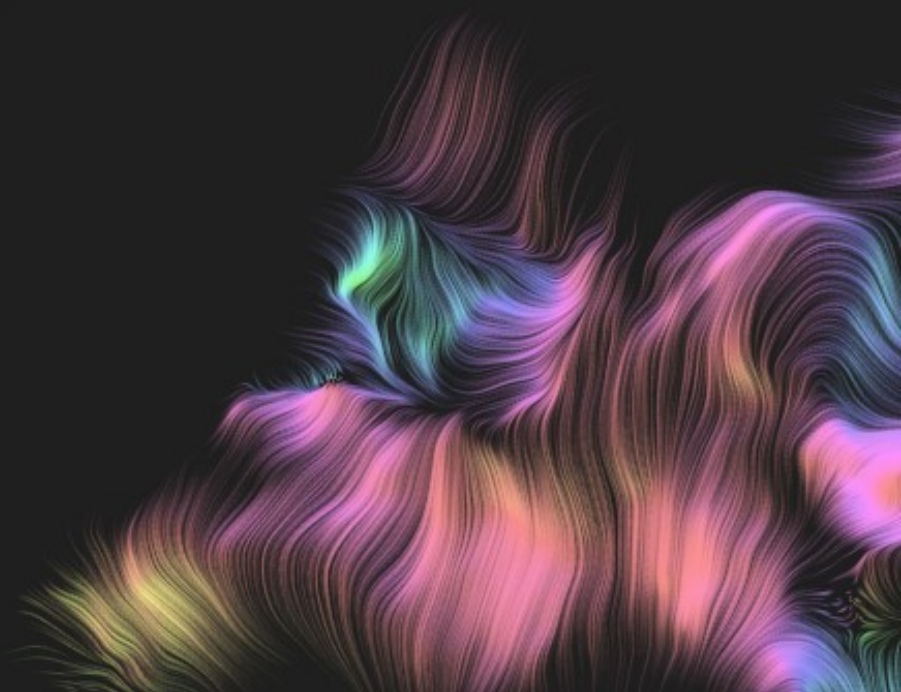



# CPU Emulation for Wrekavoc

Tomasz Buchert, June 2010



# Validation of distributed algorithms

- Formal analysis too complex
  - Fallback to experimental validation
  - Validation in heterogeneous environments
  - Scalability of large experiments
  - No realistic simulation/emulation of a processor
- 

# To simulate or not to simulate?

- Simulation may not be enough
- Models are unrealistic
- Application is not easily modeled
- Use « emulation » instead – reuse existing processors
- Emulate many processors using just one
- Bend multi-core processor to your will!



# Goal: the full emulation

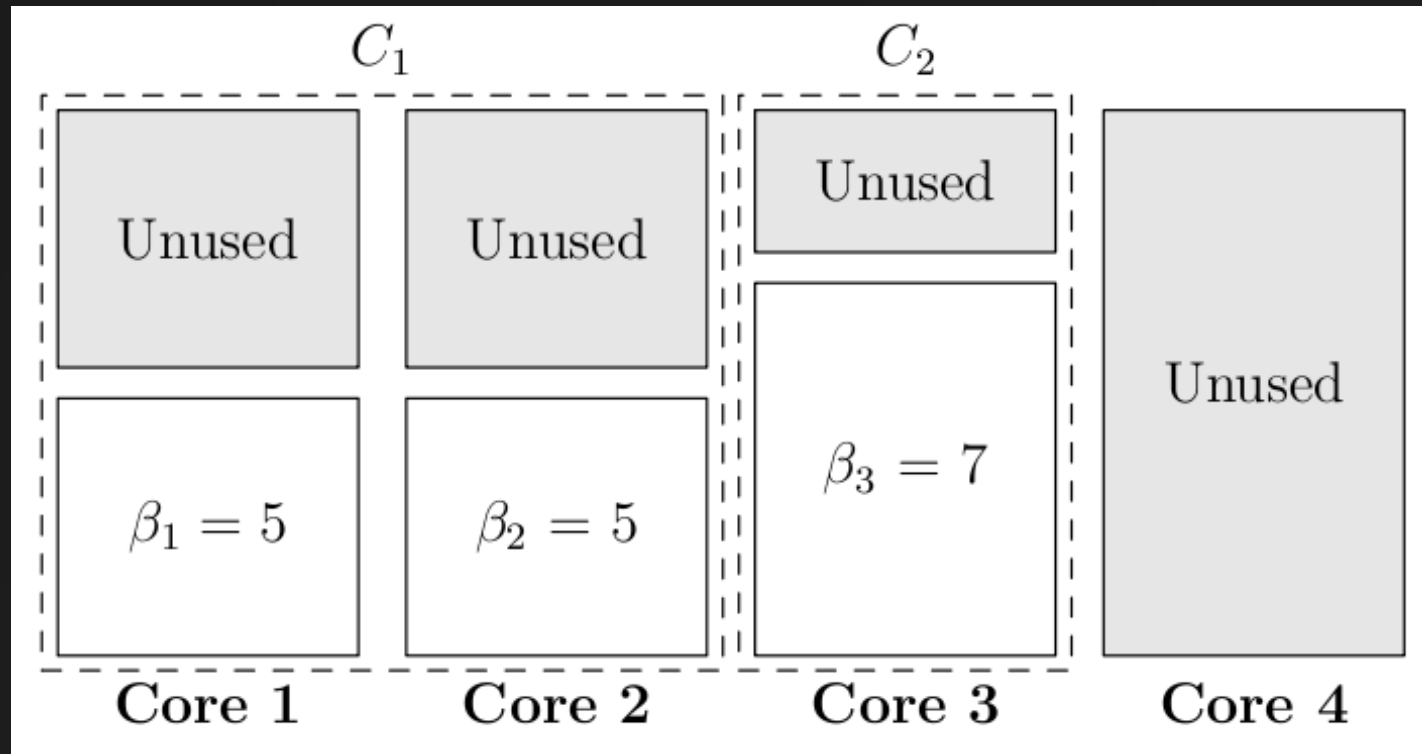
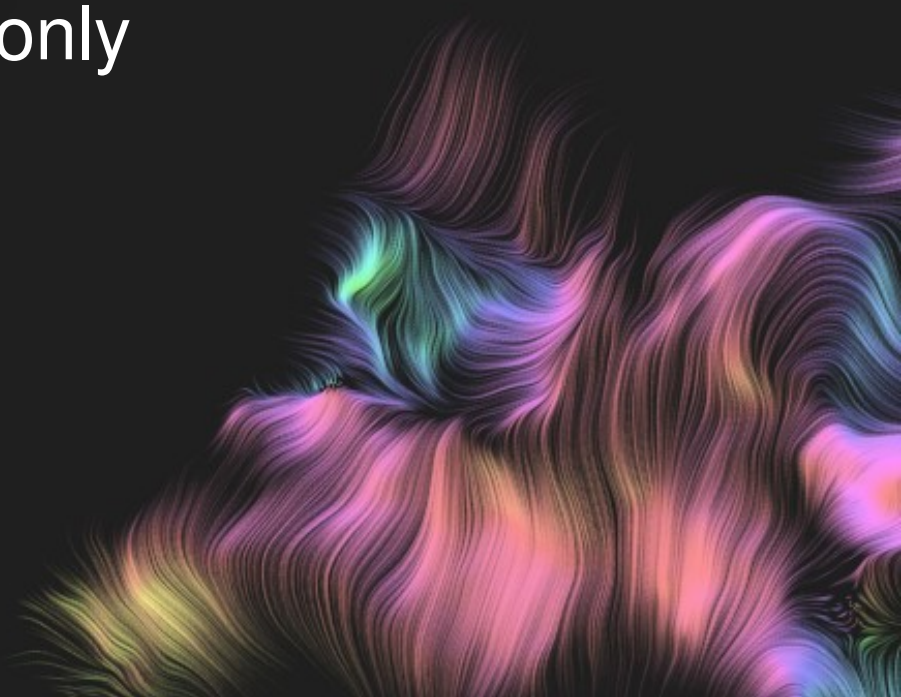


Fig. 1: An example of a CPU emulation problem. Here:  $N = 4$ ,  $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 10$ ,  $M = 2$ ,  $C_1 = \{1, 2\}$ ,  $C_2 = \{3\}$ ,  $\beta_1 = \beta_2 = 5$ ,  $\beta_3 = 7$ ,  $\beta_4 = 0$ .

# The full emulation?

- What about:
  - Processor cache?
  - Memory speed?
  - Simultaneous multithreading?
- OK, let's focus on CPU speed only



# Approaches

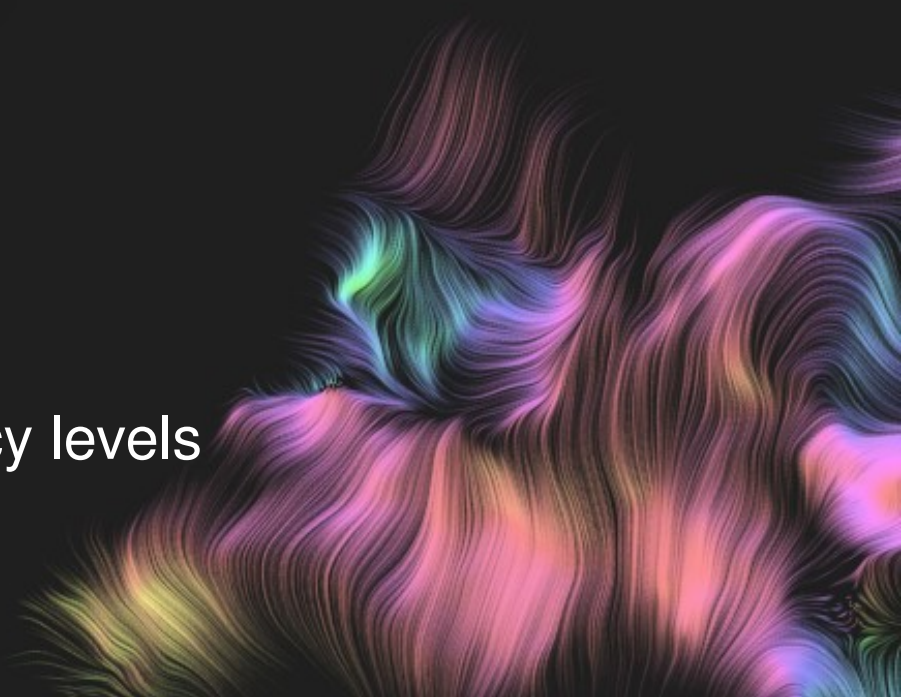
- Tools:
  - Linux
  - Cpusets (on top of Cgroups)
- Methods:
  - Dynamic frequency scaling (abbrev. CPU-Freq)
  - CPU-Lim
  - Fracas



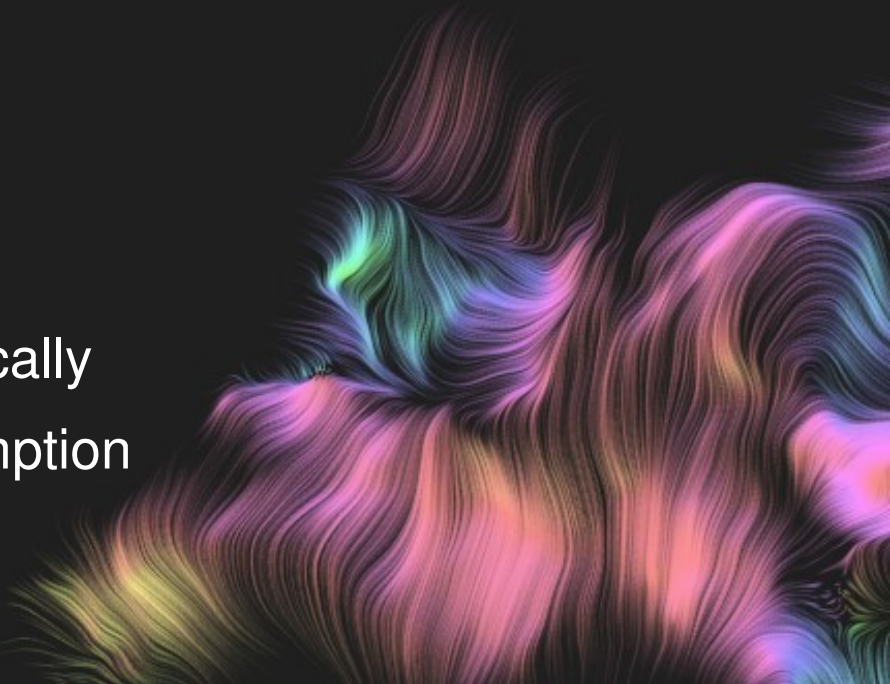


# Dynamic frequency scaling

- AKA Intel Enhanced SpeedStep or AMD Cool'n'Quiet
- Hardware solution to reduce:
  - Heat
  - Noise
  - Power usage
- Pros:
  - No overhead of emulation
  - Completely unintrusive
- Cons:
  - Only a finite set of different frequency levels



# CPU-Lim

- Method available in Wrekavoc tool
  - The algorithm:
    - If CPU usage  $\geq$  threshold  $\mapsto$  send SIGSTOP to the process
    - If CPU usage  $<$  threshold  $\mapsto$  send SIGCONT to the process
  - CPU usage: CPU time of the process / process lifetime
  - Pros:
    - Easy and almost POSIX-compliant
  - Cons:
    - Intrusive and unscalable
    - Decision to stop the process is made locally
    - Sleeping is indistinguishable from preemption
- 

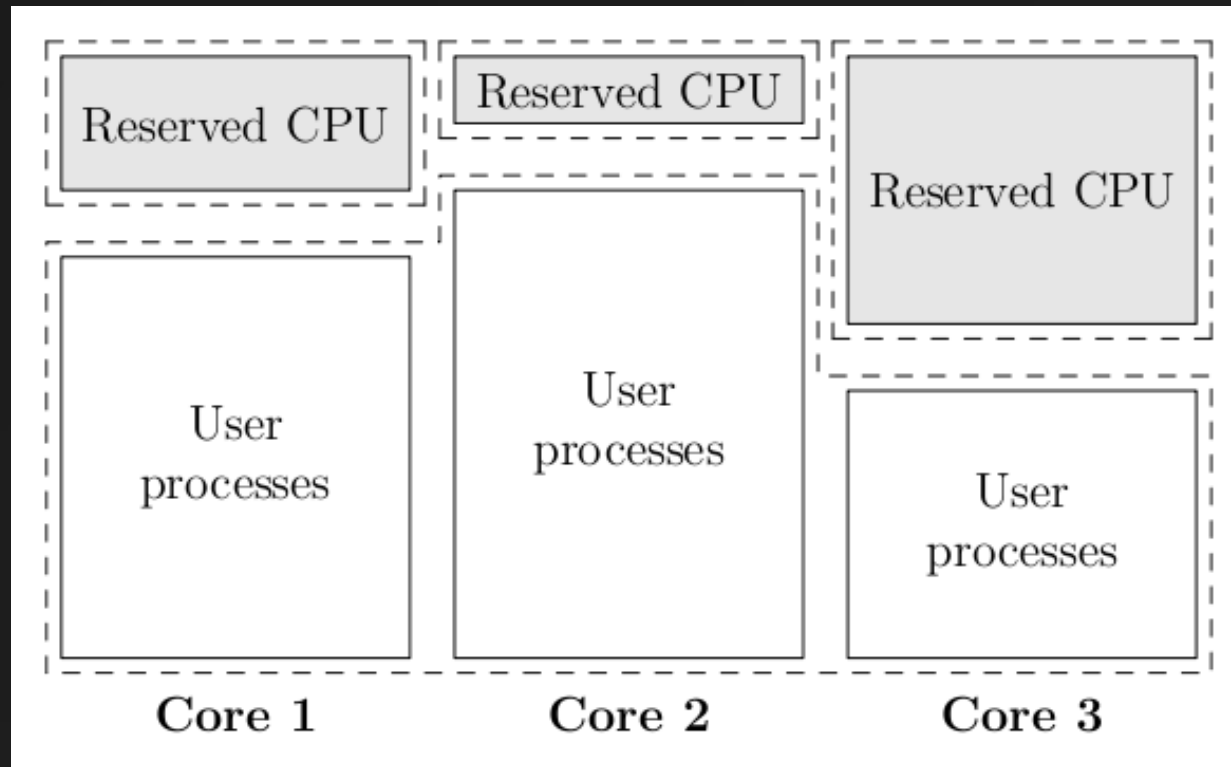


# Fracas

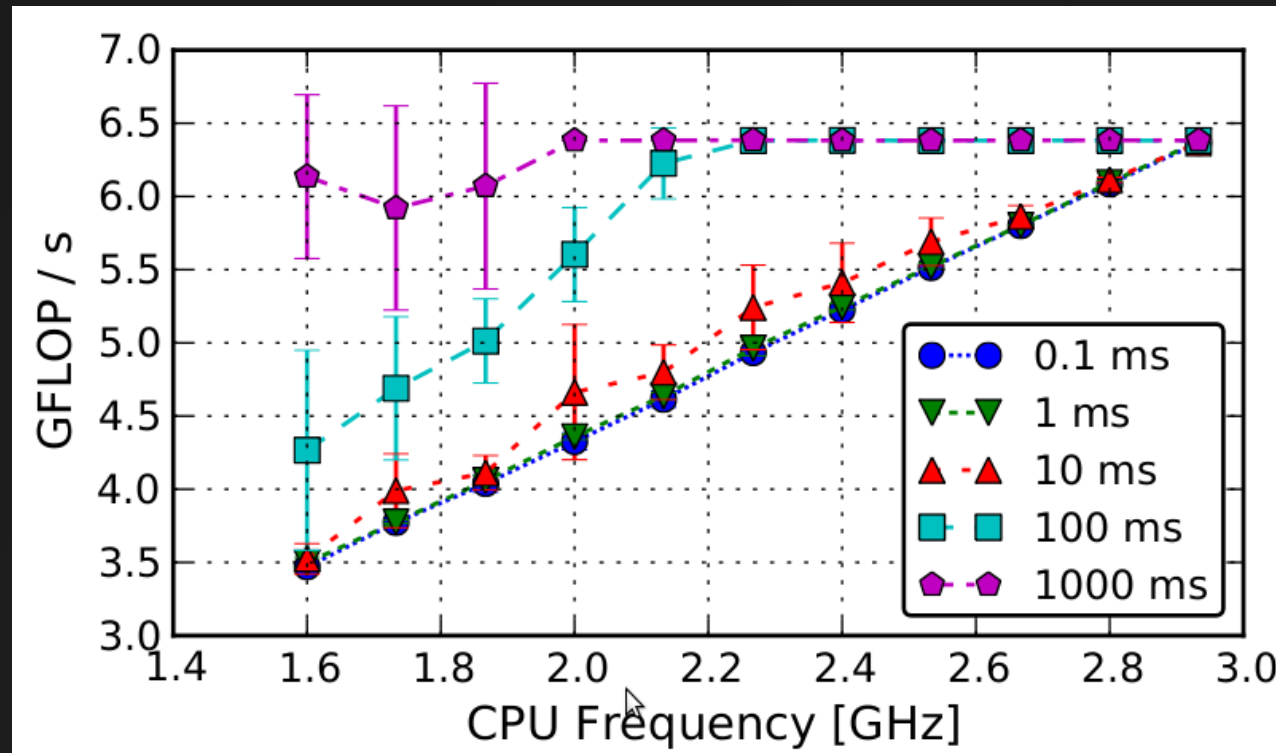
- Based on KRASH tool
- Uses Linux Cgroups
- A predefined portion of the CPU is given to tasks burning CPU
- All other processes are given the rest of the CPU time
- Pros:
  - Unintrusive
  - Scalable
- Cons:
  - Sensitive to the configuration of the scheduler
  - Unportable to different OSes



# Fracas (cont.)



# Fracas & latency of the scheduler



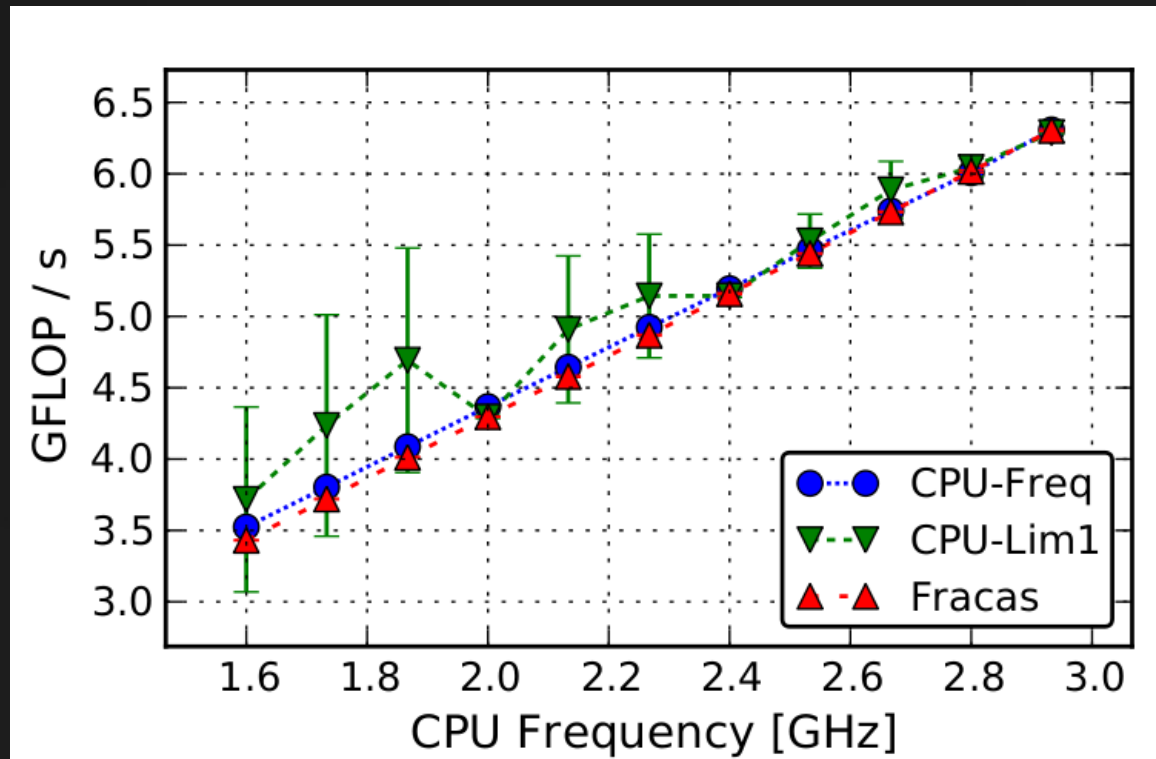
- Expected result: a straight line
- The lower frequency  $\mapsto$  better results

# Evaluation

- Based on different types of work:
  - CPU-intensive
  - IO-bound
  - Multitasking
- Tests only for CPU speeds provided by freq. scaling
- Each test repeated 10 times



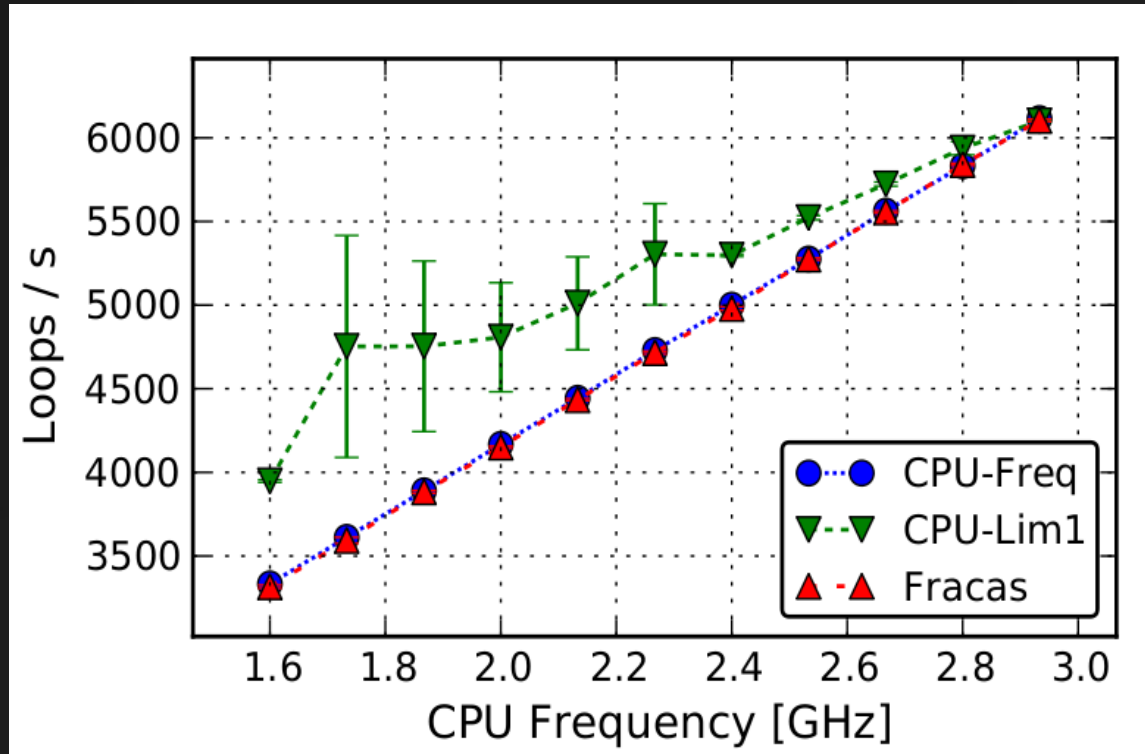
# CPU-bound work



- Fracas & CPU-Freq are doing fine
- CPU-Lim gives unstable results

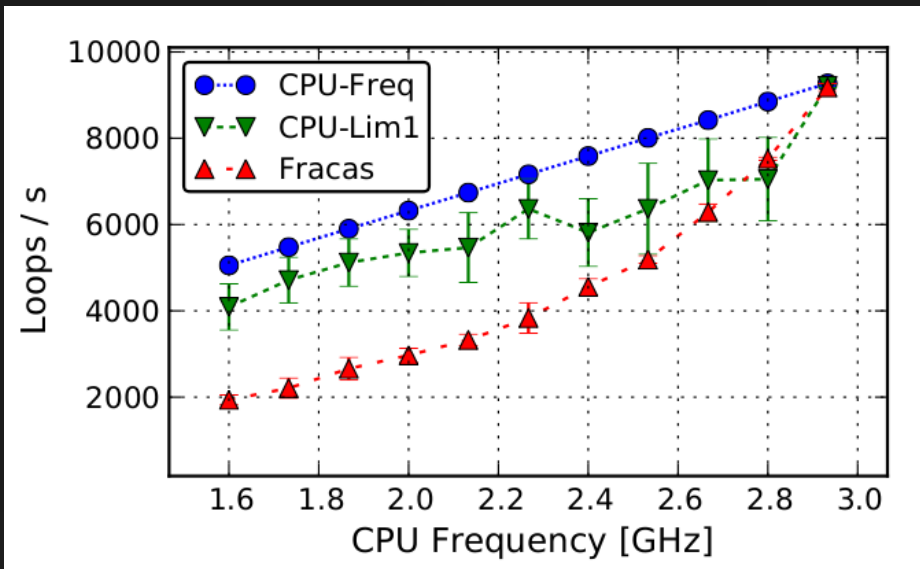


# IO-bound work



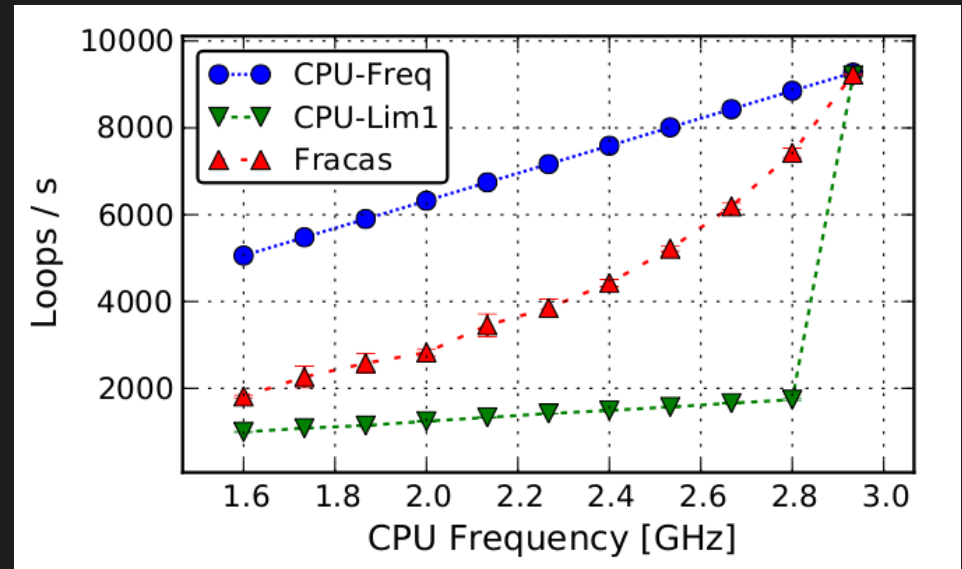
- Fracas & CPU-Freq are doing just fine
- CPU-Lim can't cope with a sleeping process

# Multitasking



## Multiprocessing:

- CPU-Freq shows the best behavior
- CPU-Lim introduces visible overhead
- Fracas is stable, yet gives unexpected results

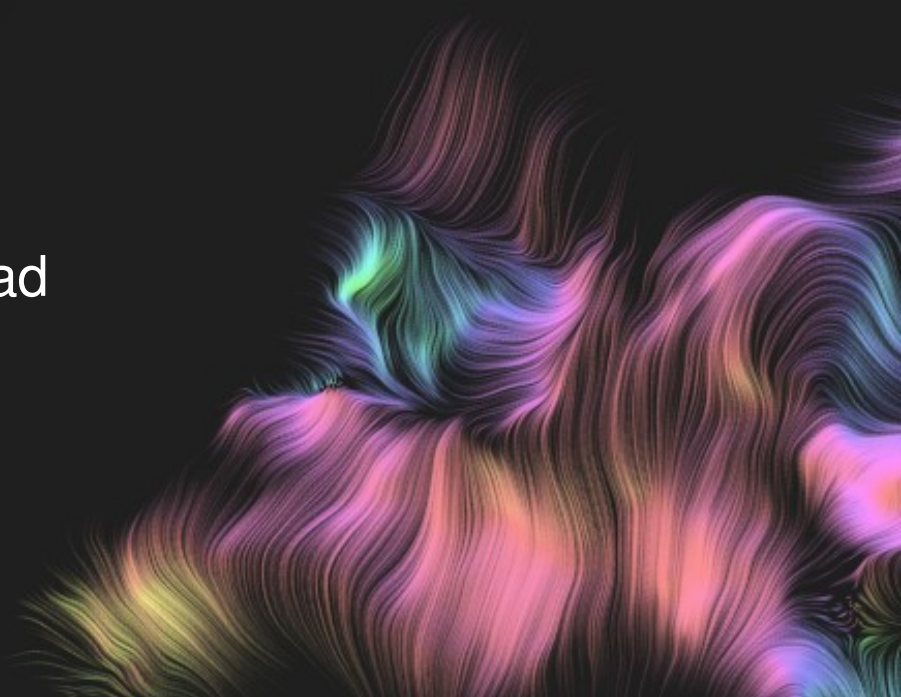


## Multithreading:

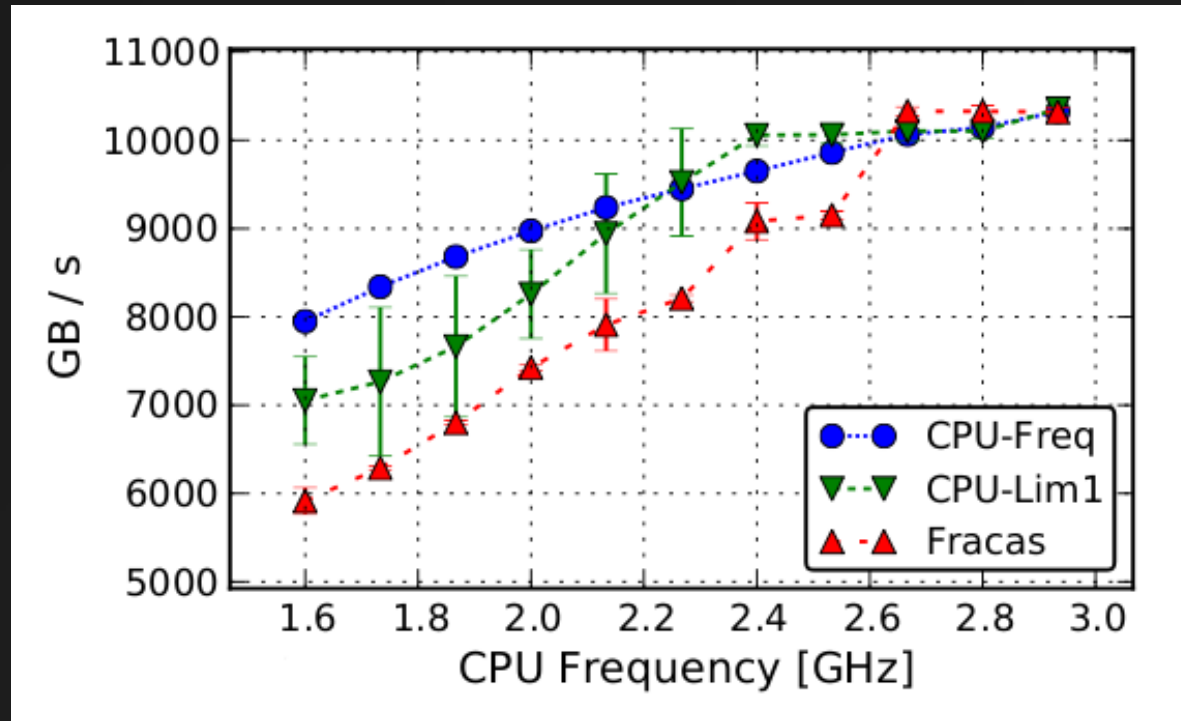
- CPU-Freq shows the best behavior (again)
- CPU-Lim can't control multithreaded work
- Fracas is stable, yet gives unexpected results (again)

# Summary

- CPU-Freq:
  - Very good results
  - Coarse granularity
- CPU-Lim:
  - Flawed
  - Intrusive
  - Hardly scalable
- Fracas:
  - Good behavior for a single-task workload
  - Scalable
  - Bad behavior for multitask workload



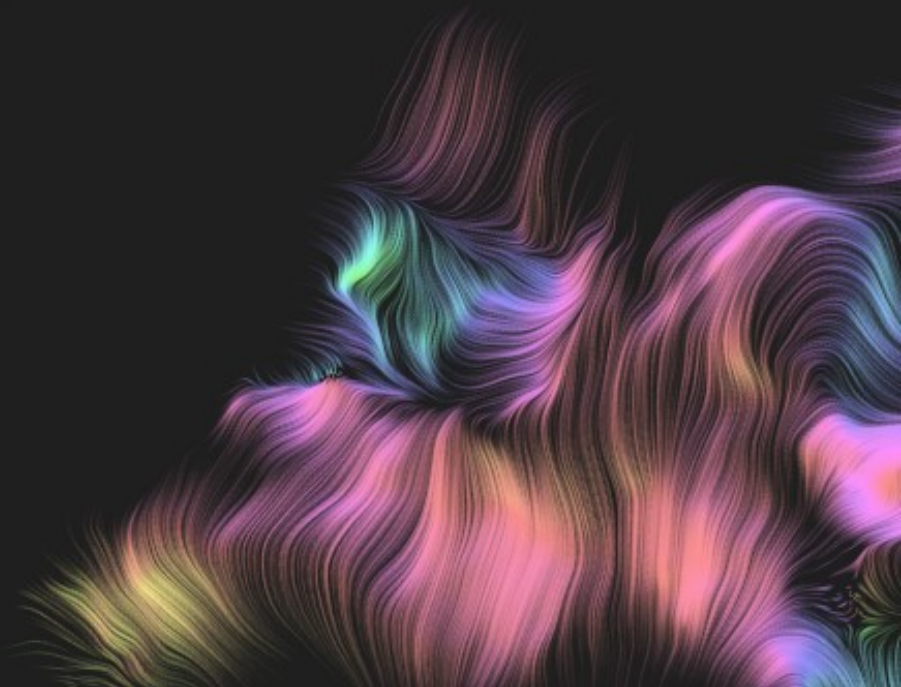
# STREAM benchmark



- All methods change the perceived memory speed ...
- ... and each method in its own, peculiar way

# Future Work

- Improve Fracas method to cover multitask work
- Merge Fracas method with Wrekavoc
- Devise a method to emulate memory speed
- Devise methods to emulate other aspects of CPU
- Take over the world :)





**Thank you  
for your attention.**

