Welcome back!

ADVANCED TOPICS: MULTI CPU SCHEDULING

Shivaram Venkataraman

CS 537, Fall 2024

ADMINISTRIVIA

Project 6 – last project!

- Deadline end of ~~next~~ this week (friday)

Midterm 3

- December 19th, 10:05am
- Details on Piazza soon

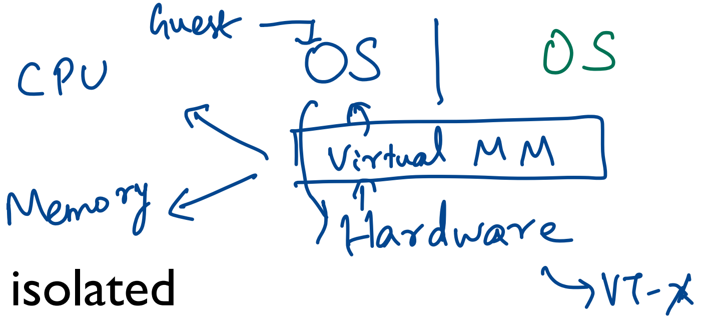
→ rooms

→ syllabus, old exams etc.

AGENDA / LEARNING OUTCOMES

How to perform CPU scheduling on multiple processors?

VMM RECAP



Virtual machine: Complete compute environment, isolated

Virtual machine monitor / Hypervisor: control resources (direct or part of OS)

Trap-and-emulate to handle system calls

Software TLB handler: maintain Physical → Machine page tables

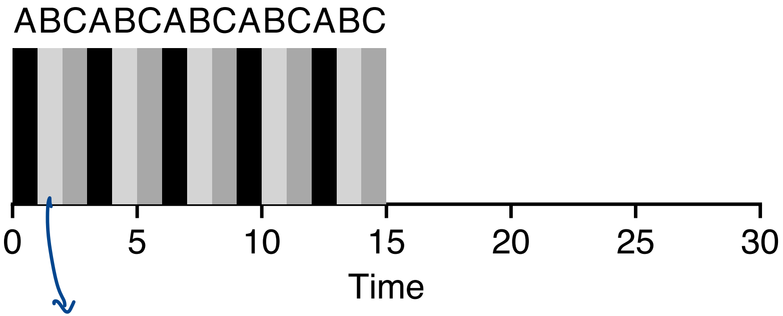
"real" memory

Paravirtualization - modify guest OS for efficiency

Intel VT-X extensions – new hardware primitives to support virtualization

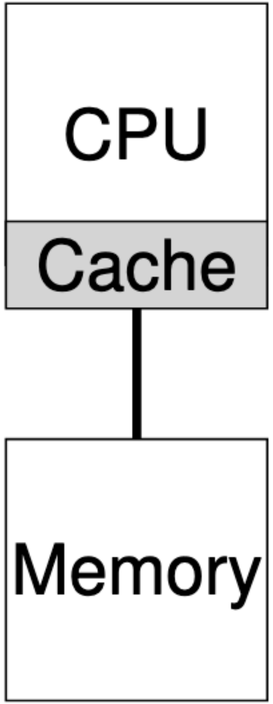
PREVIOUSLY ON SCHEDULING

Policies : choosing next process to run



time slice

↳ how long a process run



schedule(

list process).

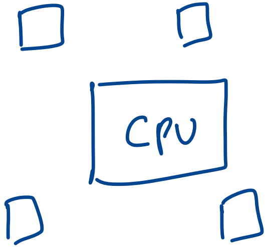
selected process to run

MULTI PROCESSORS

P0, P1, P2, P3

2 CPUs here

Two key goals

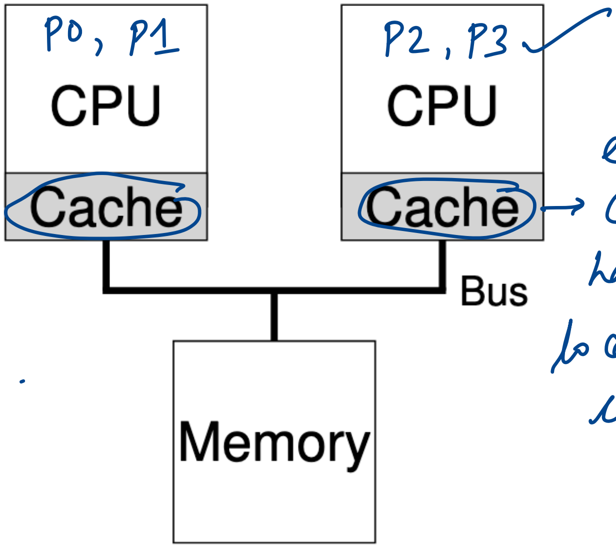


Cache Affinity

↳ if a process runs on a CPU, next time slice run it on same CPU.

Load Balancing

↳ You want processes to be evenly spread out across CPUs



each CPU has some local cache

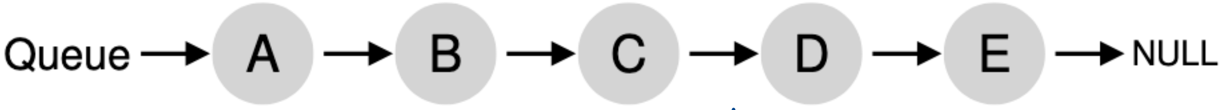
SINGLE QUEUE SCHEDULING

schedule (list Process)

4 or 8 CPUs in system

↳ process to run on next free CPU

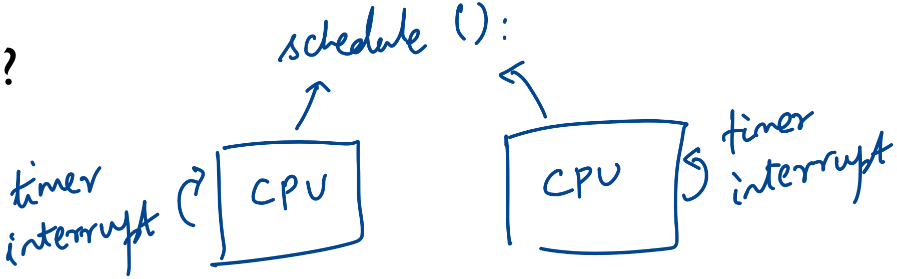
Maintain a single queue of all runnable jobs



↑ shared — .

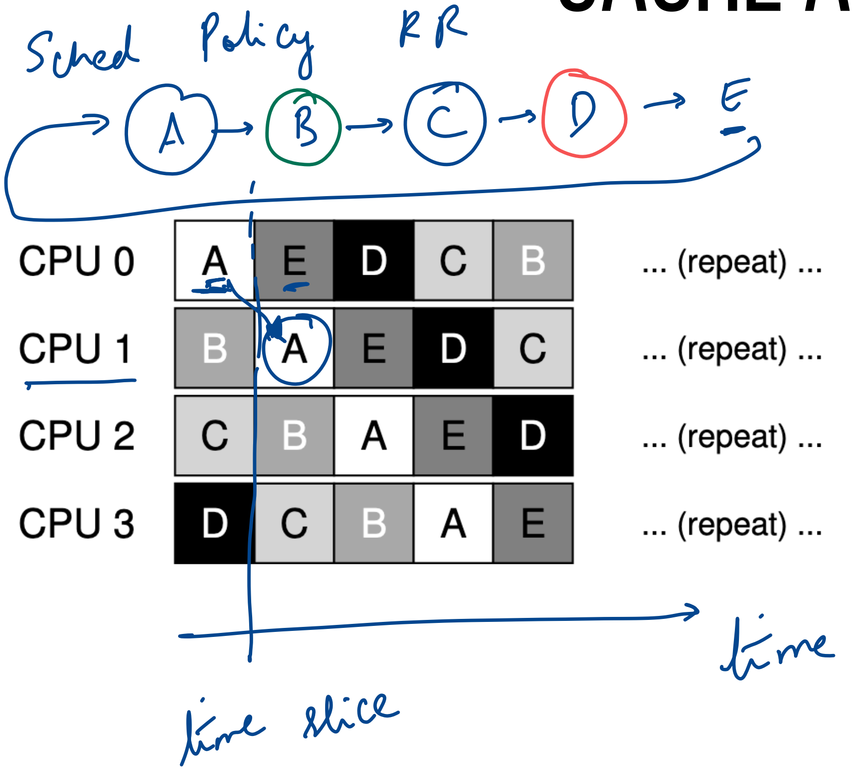
Scalability challenge

Locking overhead?



↳ as you increase num CPUs, lock contention !!

CACHE AFFINITY



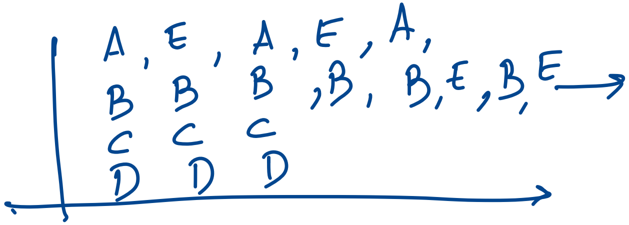
CPU 0	A	E	D	C	B	... (repeat) ...
CPU 1	B	A	E	D	C	... (repeat) ...
CPU 2	C	B	A	E	D	... (repeat) ...
CPU 3	D	C	B	A	E	... (repeat) ...

→ is affected when using single queue algorithms

Associate affinity with each job

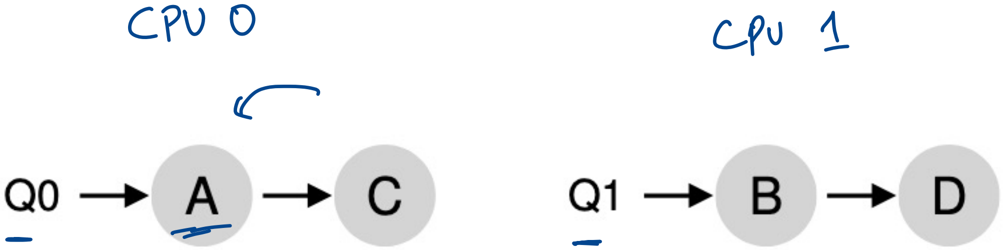
→ add affinity field to each job

Move jobs to ensure fairness / load balance



MULTI QUEUE SCHEDULING

2 CPUs system ⇒



Jobs are only on one queue!

Maintain a queue per CPU
Within each queue, use existing scheduling algorithms

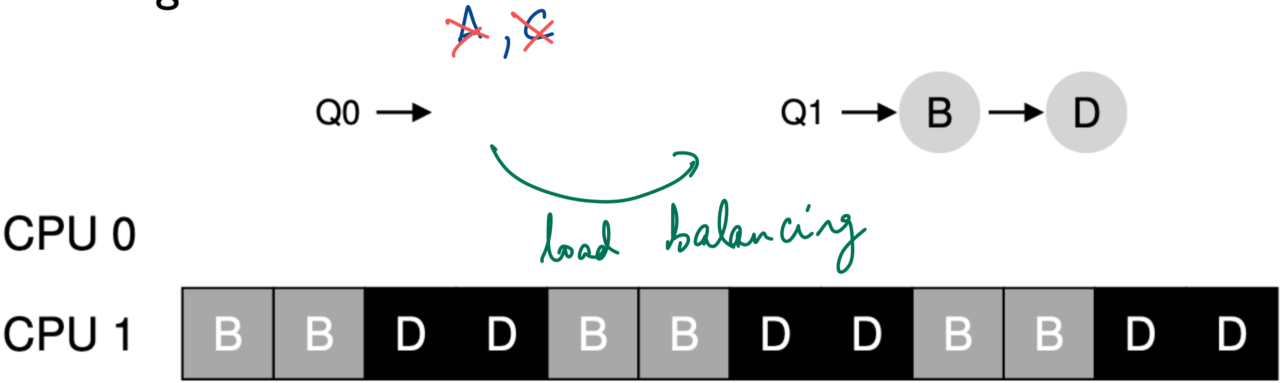
Scalability? No need for lock → scales much better

Cache affinity? Processes run on the same CPU again & again

MULTI QUEUE CHALLENGES → CFS

How to place new jobs in queues?

Load balancing



LINUX: COMPLETELY FAIR SCHEDULER (CFS)

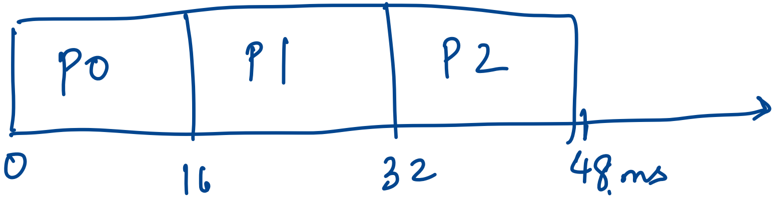
Similar approach to stride scheduler (remember P4!?)

Goal: Divide a CPU using the concept of virtual runtime (vruntime) → progress that a process has so far

Approach: Pick the process which has the lowest vruntime

When to switch processes:

48ms ← sched_latency – how long before switch ("fairness window") → 3 active jobs



24 active jobs → 2 ms ??

min_granularity → 6 ms

PRIORITY IN CFS

-20 → highest priority

0 → default

Niceness: Parameter can be set anywhere from -20 to +19

Positive nice values lower priority, negative values higher priority

nice(1) - Linux man page

Name

nice - run a program with modified scheduling priority

→ Across scheduler

nice

Map niceness value to “weight”

Higher priority gets larger time slice

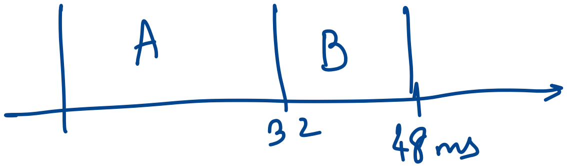
vruntime increment is scaled inversely to weight

↳ higher priority

vruntime grows slower

A : -10

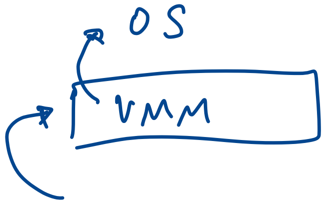
B : 0



COURSE FEEDBACK SURVEY, QUIZ

https://heliocampusac.wisc.edu/

QUIZ 21



VPN → ~~PFN~~

Machine Frame Number

What is the difference between a Type 1 and Type 2 Hypervisor?

↳ Type 1 : directly bare metal . Xen

Type 2 : runs as part of Host O S . Linux KVM

Which of the following is not part of the Processor Status Word? ⟶ Goldberg , Popek

↳ General Purpose Register

True or False: Executing a system call in a VM is no more expensive than a normal system call.

False . Trap & emulate → additional steps

When a virtual machine experiences a page fault, which of the following is NOT true?

→ VPN → PFN mapping without VMM's help

CFS: HANDLING THREAD JOINS

New thread or thread wakes up from sleep → long running I/O

How to set vruntime?

New thread: vruntime equal to the maximum vruntime of runnable threads

↳ "back of the queue"

I/O wakeup: Set to minimum of all runnable jobs right now

↳ "front of the queue" → 1 chance to go right away

CFS ON MULTI PROCESSORS

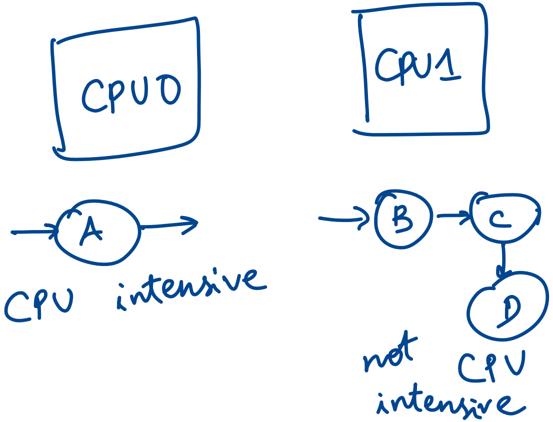
Load Balancing: Goal is to balance out work (or load) across all cores

Example: 1 CPU-intensive thread vs. 10 threads that mostly sleep

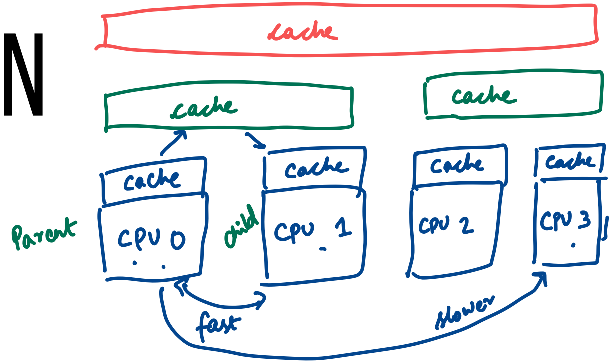
"work"

Load of a thread: average CPU utilization of a thread

Effective goal: Balance sum of load across cores



CFS: THREAD CREATION



Decide which cores are suitable to host the thread

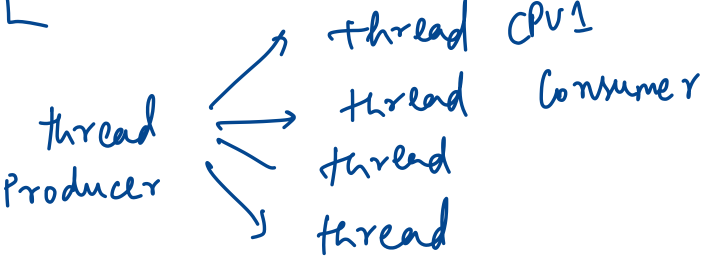
Approach: heuristics to decide suitable cores.

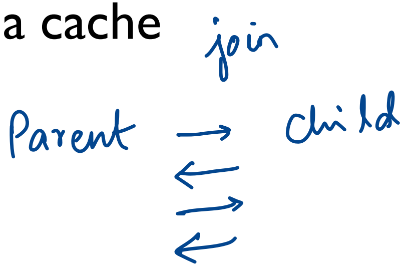
Pick core among those with lowest load

load balancing across cores in the system

1-to-many producer-consumer: Spread out consumers across cores

1-to-1 communication: Restricts to cores sharing a cache

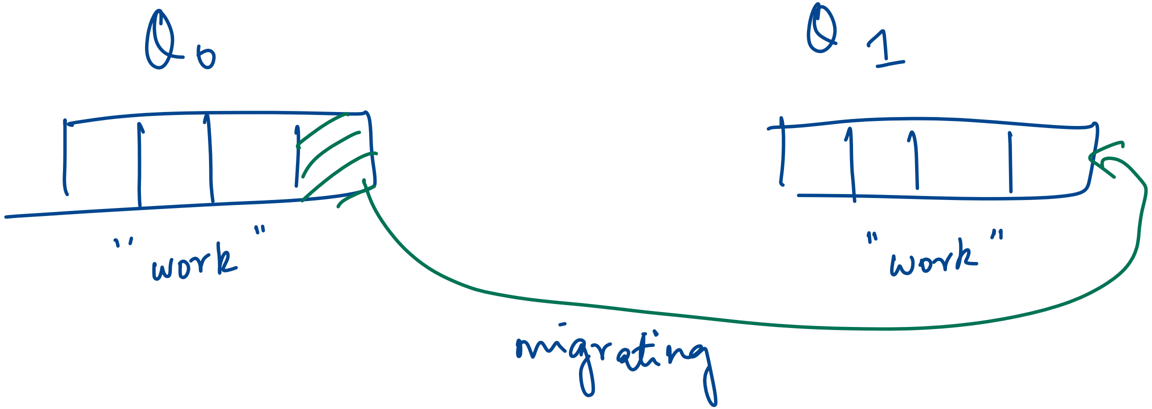




CFS: LOAD BALANCING

Periodically (e.g., 4ms) steal work from other cores

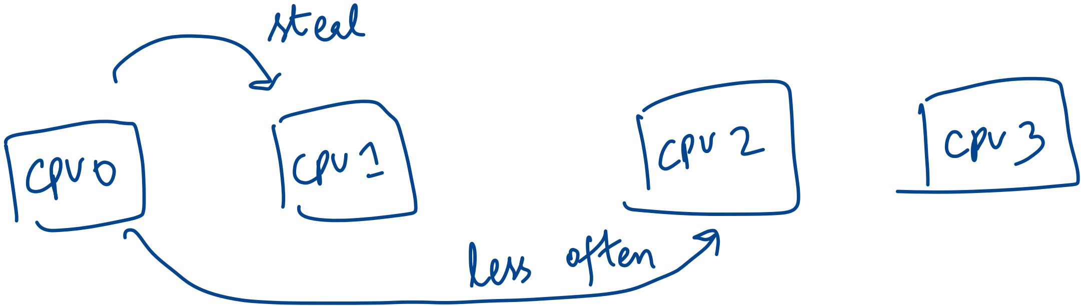
When stealing work, even out the load between the two cores



CFS: LOAD BALANCING

Topology awareness while work stealing

Try to steal work more frequently from cores that are "close" vs. cores that are "remote" (e.g., on a remote NUMA node)



Load difference is small (less than 25% in practice), no load balancing all of threads

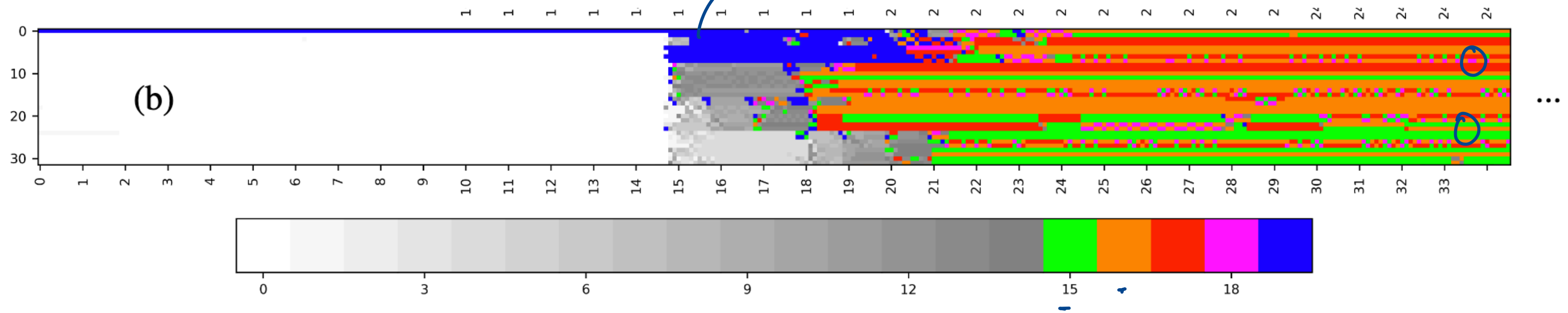


Figure 6: Number of threads per core over time on (a) ULE and (b) CFS. Each line represents a core (32 in total), time passes on the x-axis (in seconds), and colors represent the numbers of threads on the core. Thread counts below 15 are represented in shades of grey. Threads are pinned on core 0 for the first 14.5 seconds of the execution.

Start 512 spinning threads on core 0. Let load balancer work

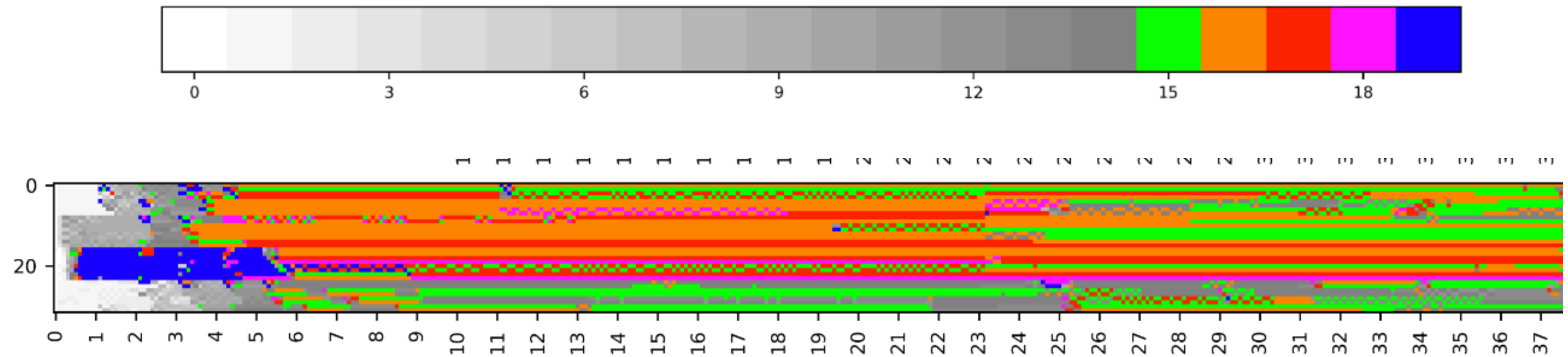


Figure 7: Number of threads per core over time on c-ray on (a) ULE and (b) CFS. start pinned on core 0.

Create 512 threads.
Threads are not pinned at creation time
scheduler chooses a core for each thread
All threads wait on a barrier before computation

ULE (BSD SCHEDULER)

Aims to even out the number of threads per core (not load)

Choosing a core for a newly created thread: affinity heuristic

Periodic load balancing only by core 0.

a thread from the most loaded core, the (donor)

to the less loaded core, the (receiver)

Next class: Distributed Systems

Last week!