Linux Scheduling

on

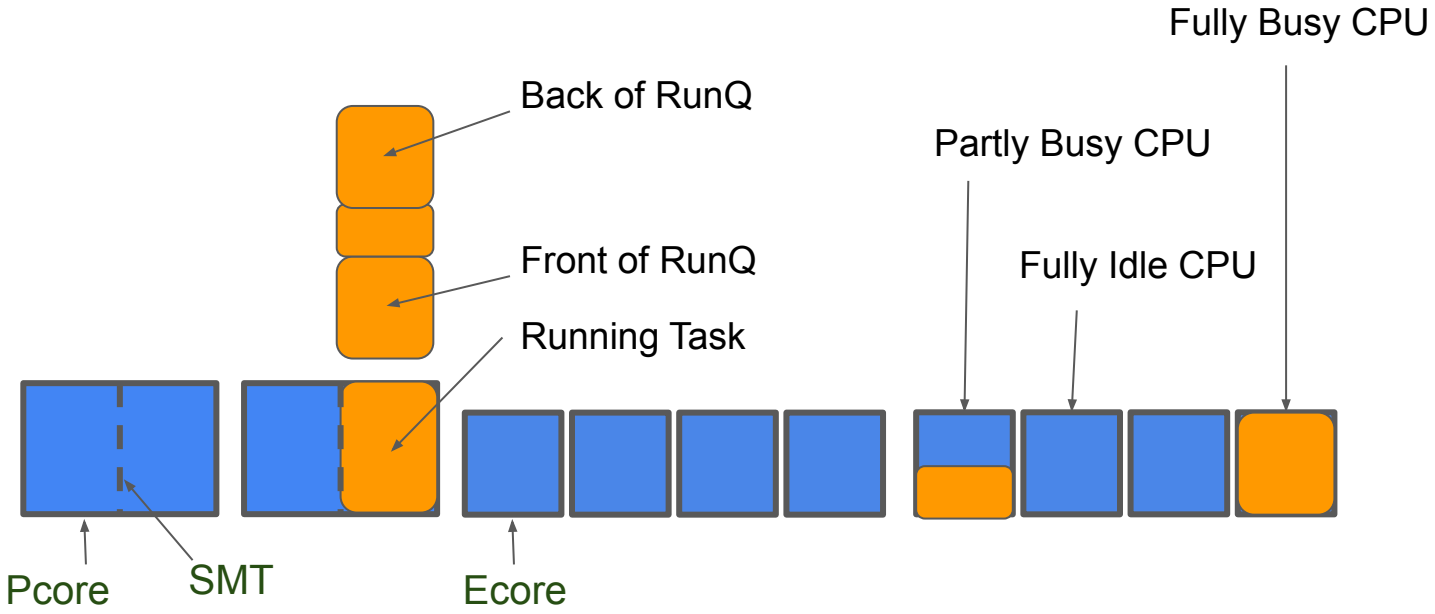
Intel Performance Hybrid Architecture

Len Brown, Ricardo Neri

Intel Open Source Technology Center

Linux Plumbers Conference - Dublin, Ireland Sept 2022

Linux tasks on Intel Hybrid "Pcores and Ecores"



Uniform Instruction Set on all CPUs

Linux v4.9 (before ITMT)

Task Placement:
1. Pcore equal to Ecore
2. Pcore HT sibling

Performance variability due to random placement

Linux v4.10 - v5.15 ITMT

Task Placement:
1. Pcore
2. Pcore HT sibling
3. Ecore

Scheduler erroneously prefers HT sibling over Ecore.

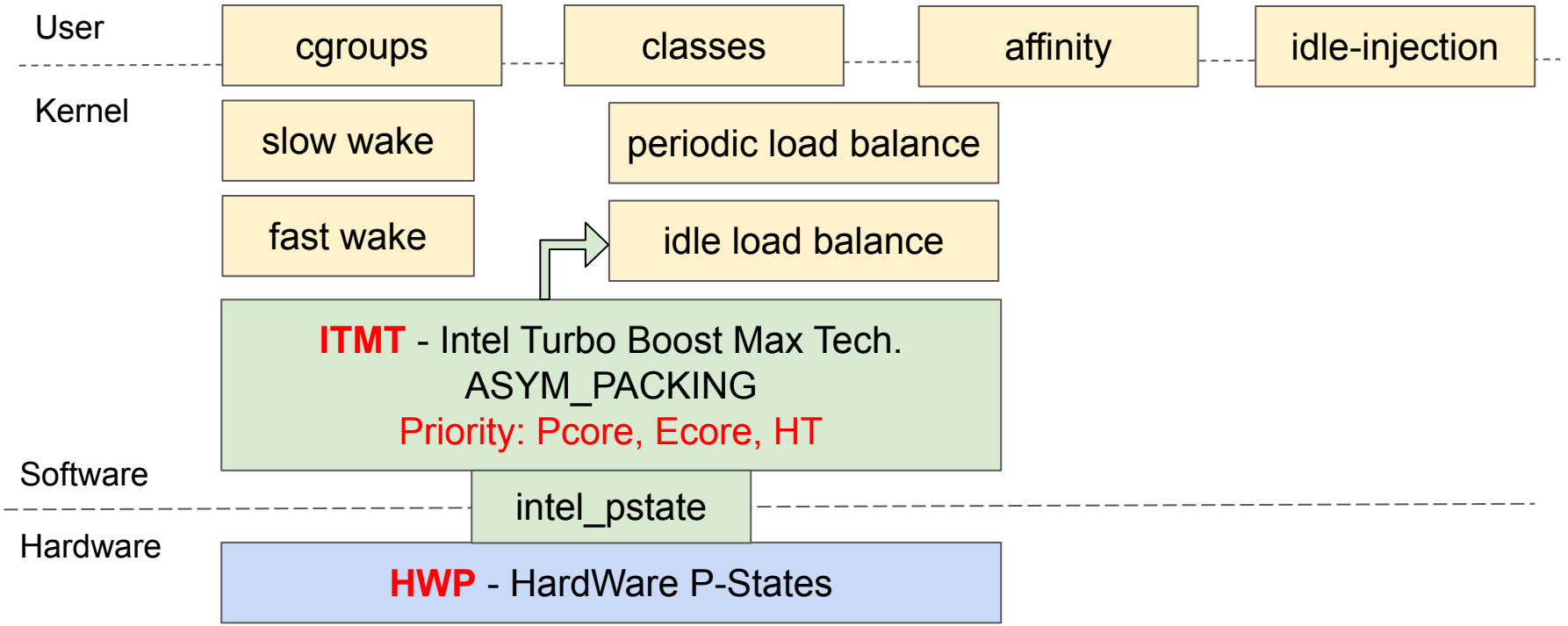
Disable ITMT: # echo 0 > /proc/sys/kernel/sched_itmt_enabled

Linux v5.16 ITMT

Task Placement:
1. Pcore
2. Ecore
3. Pcore HT sibling

Scheduler correctly spreads to Ecore before HT sibling.

ITMT Scheduler Architecture



Pcores are faster than Ecores[1]

At ISO Frequency, for a *nominal instruction mix*

Pcore/Ecore = 1.27

1. https://edc.intel.com/content/www/us/en/products/performance/benchmarks/hot-chips-2021/

Intel Hardware Feedback Interface (HFI)

Every CPU has Performance and Efficiency "Scores"

Index	CPU	Performance	Efficiency
0	0,1	56	92
1	2,3	56	92
2	4-7	30	100
3	8-11	30	100

56 = 1.27 * 4.4 Ghz

30 = 1.0 * 3.0 Ghz

Pcore Performance advantage depends on instructions

Performance ratio at ISO frequency:

Class	ISA Example	Pcore/Ecore Performance
0	SSE	1.27
1	AVX2	1.5
2	VNNI	2.0
3	PAUSE	1.0

Pcore/Ecore performance depends on Instruction mix (ISA class)

Intel Thread Director (ITD)

ITD adds (4) ISA classes to HFI

Index	CPU
0	0,1
1	2,3
2	4-7
3	8-11

P0	P1	P2	P3
56	66	88	44
56	66	88	44
30	30	30	30
30	30	30	30

E0	E1	E2	E3
92	92	92	92
92	92	92	92
100	100	100	100
100	100	100	100

Benefit of running on Pcore depends on type of instructions (ISA class)

ITD ISA Classification

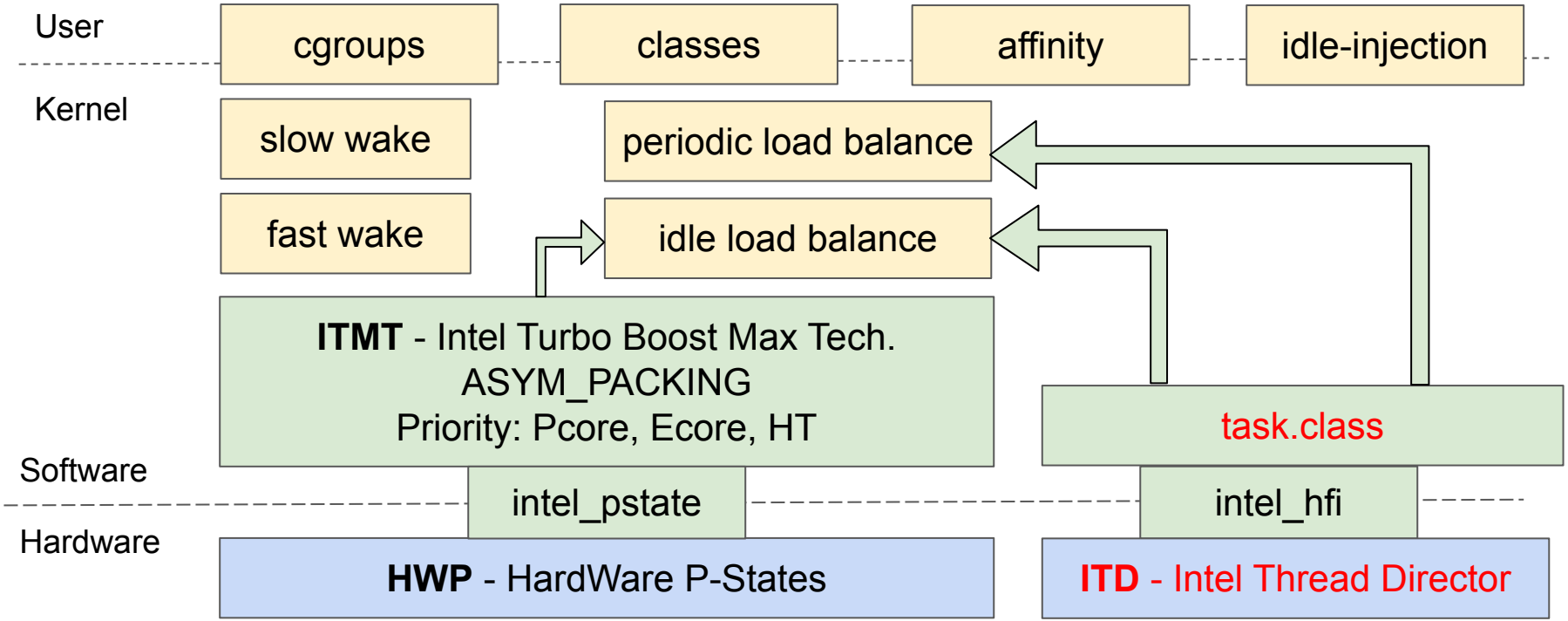
Linux user clock tick:

```
curr->class = MSR_IA32_HW_FEEDBACK_CHAR
```

Linux context switch:

```
HRESET
```

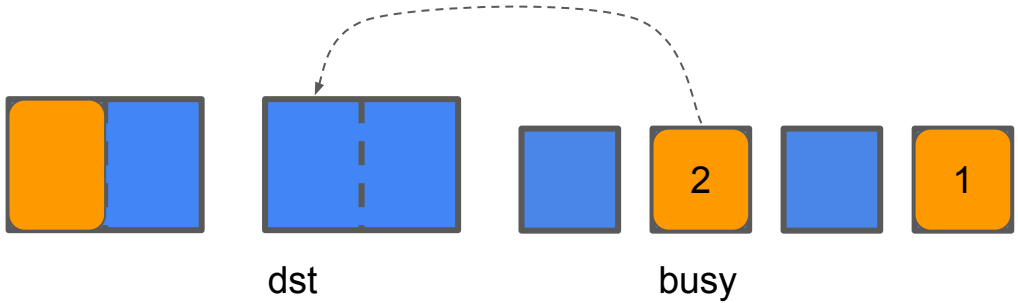
ITMT+ITD Scheduler Architecture



Idle Load Balance (Ecore -> Pcore)

When: partially idle: #PCORES <= #task <= #CPU

What: CPU "dst" enters idle, searches for "busy" to offload



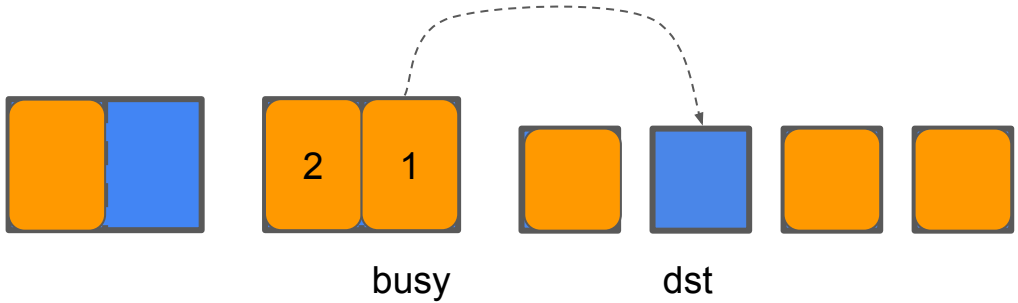
ITMT: Pcore pull from Ecore, Ecore pull from HT, always

ITD: *Opportunity* to break tie on which busy is "busiest"

Idle Load Balance (HT -> Ecore)

When: partially idle: #PCORES < #task <= #CPU

What: CPU "dst" enters idle, searches for busiest to offload



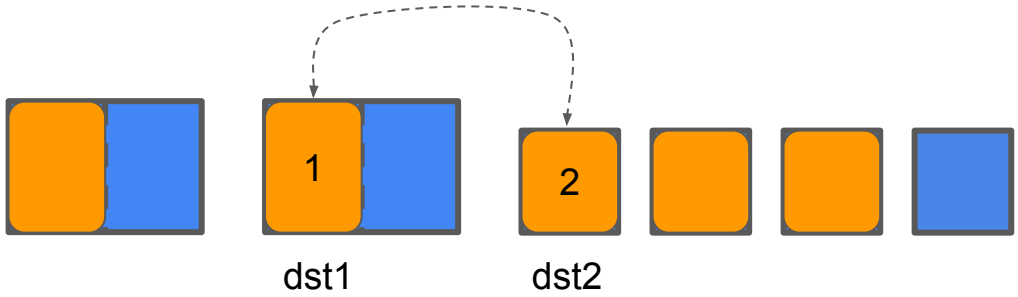
ITMT: Pcore pull from Ecore, Ecore pull from HT, always

ITD: *Opportunity* to break tie on which busy is "busiest"

Periodic Load Balance - Live Exchange

When: partially idle or fully utilized or overutilized

What: CPU "dst1" enters periodic load balancer, searches dst2



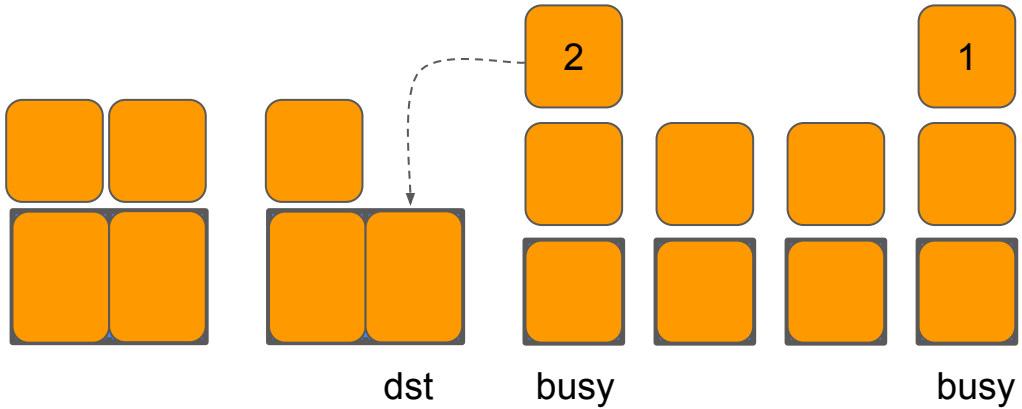
Today: NUMA load balance live exchange (on page fault)

Opportunity: ITD live exchange can increase throughput

Periodic Load Balance - Pull from Queue

When: overloaded/unbalanced: #CPU < #task

What: CPU "dst" enters periodic load balancer, searches busiest to help



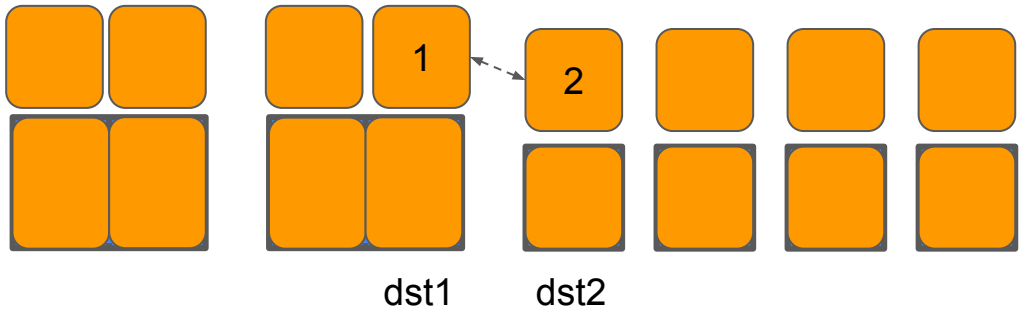
Today: dst selects busy, and pulls until balance reached

Opportunity: ITD can break tie on which sg is busiest

Periodic Load Balance - Queue Exchange

When: overloaded/balanced: #CPU < #task

What: CPU "dst1" enters periodic load balancer, searches dst2

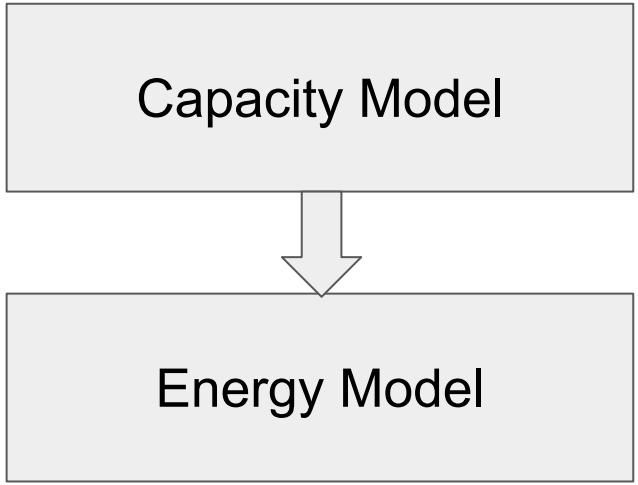


Today: "balanced" queues are untouched

Opportunity: ITD can increase throughput

Linux EAS

Task Placement:



Energy Model used as tie breaker when multiple possible Capacity Scenarios

Why EAS is not a good match for ADL

Capacity Model Challenges

1. SMT
2. Turbo
3. HWP

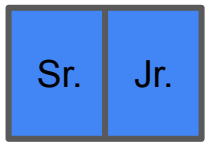
Energy Model Challenges

1. SMT
2. Static ITD Energy table
3. Pcore often more efficient than Ecore

Discussion

ITMT SMT migration Improvement (post Linux 6.0)

1. Remove superfluous Jr-sibling -> Sr-sibling migrations
2. Remove restriction on Ecore pulling from only Jr-sibling
3. Remove artificial software assignment of different priority for Jr and Sr



old

SMT siblings are independent CPUs with Jr/Sr priority

new

SMT Siblings are equal parts of one shared Core.