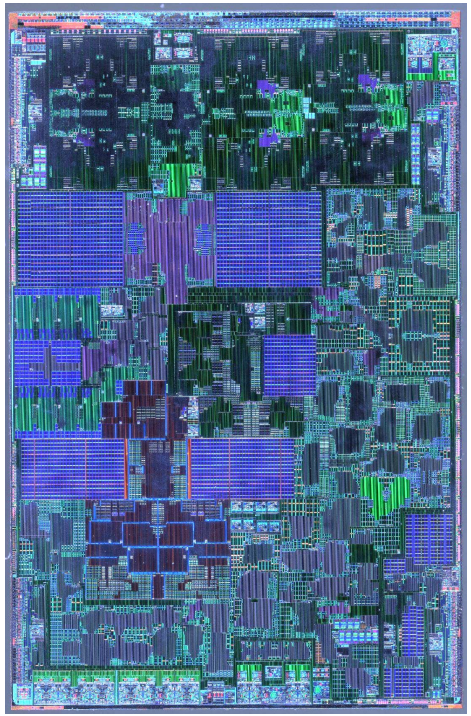

# Asymmetric Multiprocessing (ASMP) & OS Scheduling

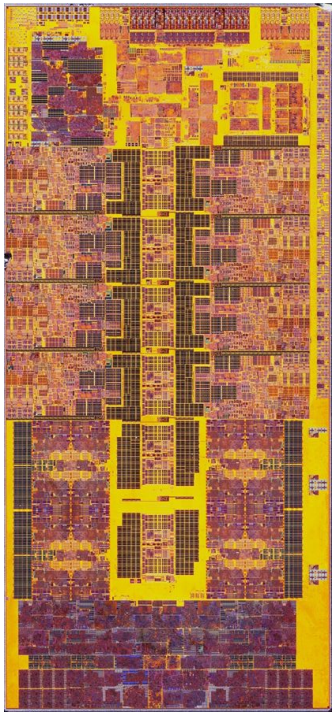
**Qucheng Jiang** **001569593**

**Shihan Zhao** **002772845**

# Outline



Apple A17 Pro



Intel i9-13900K

1.Introduction

2.big.Little Architecture Context (fr. ARM)

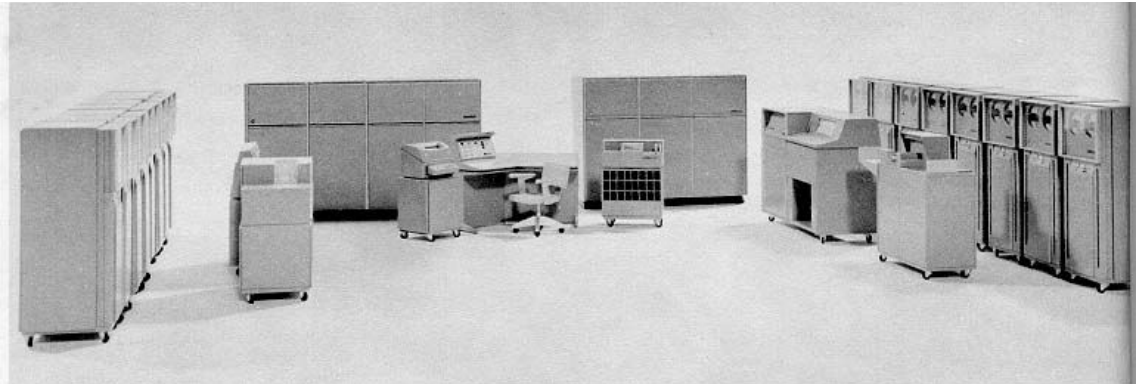
3.Completely Fair Scheduler (CFS)

4.Energy Aware Scheduling (EAS)

# History

Before the advent of Symmetric Multiprocessing (SMP), concept of Asymmetric Multiprocessing (ASMP) emerged in the 1960s and 1970s as a cost-effective method to enhance computing power by adding more CPUs without significant costs.

In recent years, ASMP architecture like big.Little has emerged mainly to solve the contradiction between power consumption and performance.



Burroughs B5000

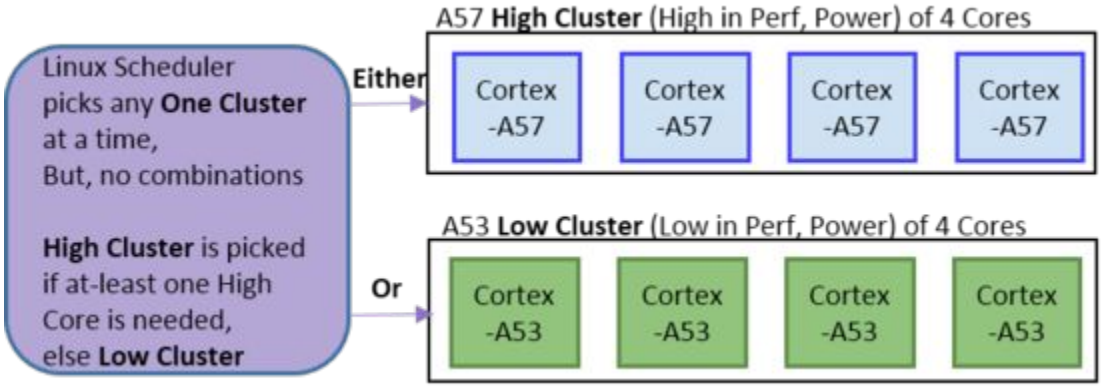
CDC 6500

https://livingcomputers.org/Computer-Collection/Vintage-Computers/Mainframes/CDC-6500.aspx

# Introduction

What is big.LITTLE?

- Complex multicore CPU architecture combining...
  - Several high performance “big” cores
  - Several lower power “small” cores



# Introduction

Why big.LITTLE?

• Targeting optimal power saving/performance balance

– Real life CPU load is bursty

• big.LITTLE allows for running power hungry cores only when bursts are coming

– Peak performance only when it's needed

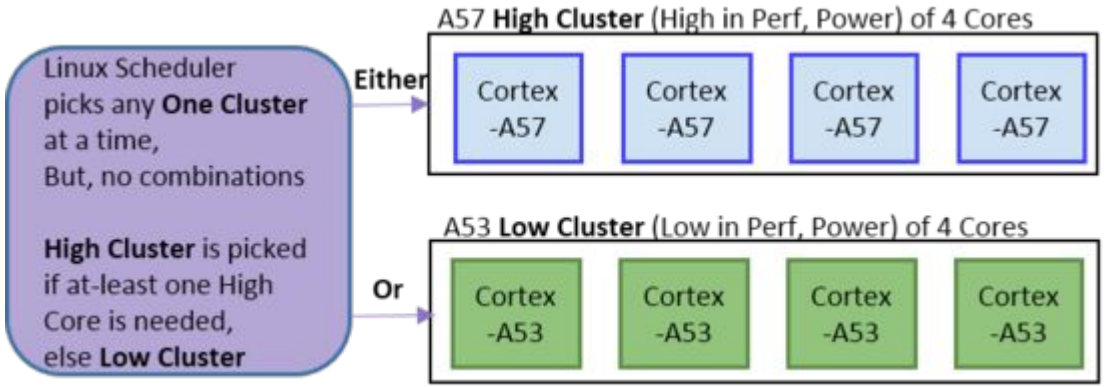
– Power optimized cores run most of the time

• More options for fine tuning compared to standard SMP

# Context

Big / LITTLE cores: how to combine

- Clustered switching
  - A cluster of big cores and a cluster of little ones
  - The OS can only use one cluster at a time
  - Standard SMP scheduling within the cluster

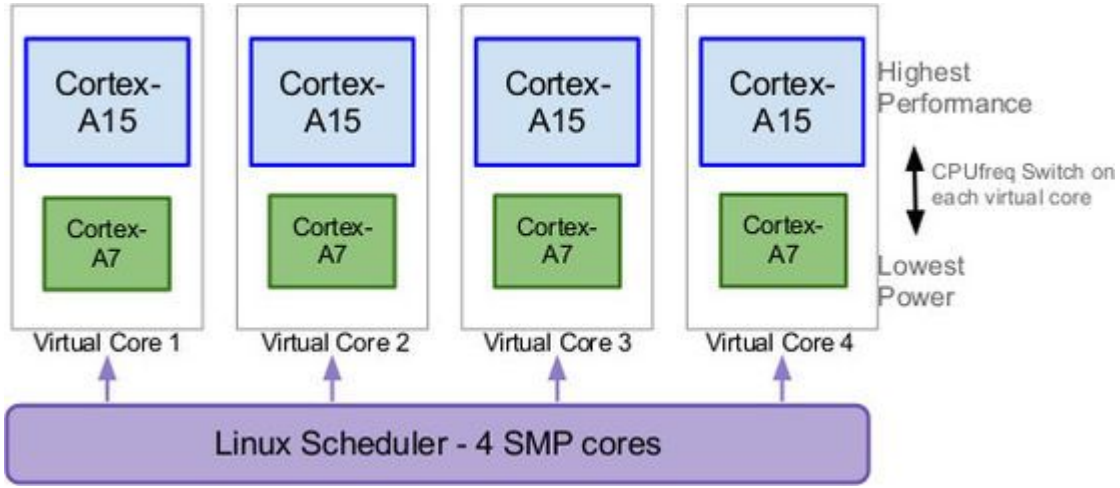


# Context

Big / LITTLE cores: how to combine

- In-kernel switching (CPU migration)

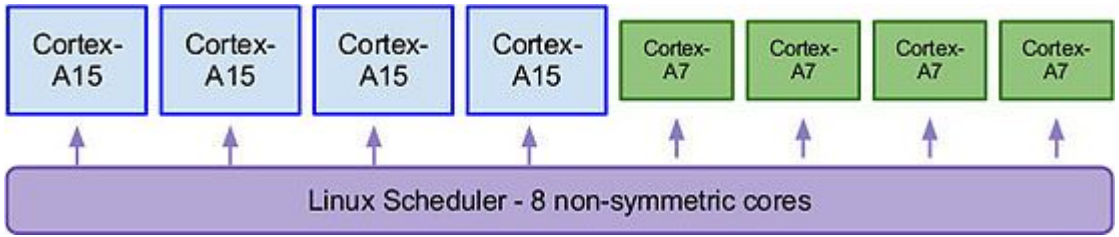
– Little and big cores are split into pairs



# Context

Big / LITTLE cores: how to combine

- Heterogeneous switching (HMP)
  - All cores can be used simultaneously



# CFS

Mainline Linux scheduler (“fair”)

- Goals of the completely fair scheduler (CFS)
  – Even distribution of task load across cores
  – The task ready to run should quickly find core to run on
- Implementation
  – Sorting tasks in ascending order by CPU bandwidth received
- Red-black trees are used to streamline the process
- The leftmost task off the tree is picked up next
  – It has the least spent execution time
- Limitations
  – Implies that the cores are the same (e. g. SMP)

# CFS-obstacles

Big (and LITTLE) obstacles

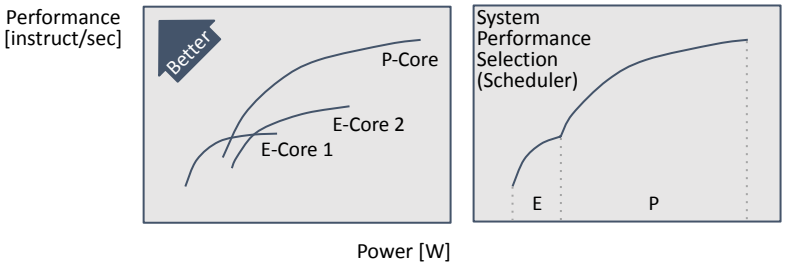
- Mainline CFS is not really applicable to b.L
  – Global symmetry principle doesn't work in asymmetrical system
- Big cores require careful treatment
  – Should only be run when it's really needed
- Power consumption and heating issues
  – Detection of such situation is the problem to solve
- Task packing problem

# Energy Aware Scheduling

Energy-aware scheduling (EAS) gives the scheduler the reference to predict the impact of its decisions on the energy consumed by CPUs. EAS relies on an Energy Model (EM) of the CPUs to select an energy-efficient CPU for each task, with a minimal impact on throughput.

🎯 Goal:

Minimize energy cost while still getting the job done without significantly impacting system throughput and reaching “good enough” performance.




🔺 Maximize:

Performance [inst/s] per Power [W]

= 🔻 Minimizing:

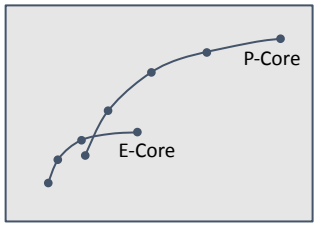
Energy [J] per Instruction

https://docs.kernel.org/next/scheduler/sched-energy.html
https://www.linaro.org/blog/energy-aware-scheduling-eas-progress-update

# Energy Aware Scheduling

Energy-aware scheduling (EAS) alters the assignment of Completely Fair Scheduler (CFS) tasks to CPUs by utilizing the Energy Model (EM) to make better scheduling decisions.

Energy Model (EM) enables the scheduler to assess the consequences of its decisions by using a much simpler design to minimize the impact on scheduler latency. This model aids in choosing the most energy-efficient CPU for task execution without compromising system performance, using knowledge of CPU capacities and energy costs.




- Operating Performance Point (OPP)

Energy Model (EM)

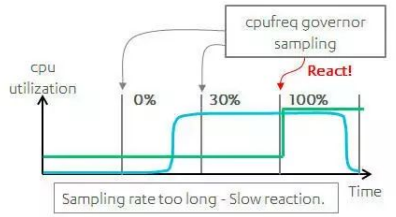
| E-Core | | P-Core | |
|---|---|---|---|
| Cap | Pwr | Cap | Pwr |
| 170 | 50 | 512 | 400 |
| 341 | 150 | 768 | 800 |
| 512 | 300 | 1024 | 1700 |

https://docs.kernel.org/next/scheduler/sched-energy.html
https://www.linaro.org/blog/energy-aware-scheduling-eas-progress-update

# Energy Aware Scheduling

During task wake-up, EAS leverages the EM along with Per-Entity Load Tracking (PELT) signals to select an optimal CPU. The function select_task_rq_fair() invokes find_energy_efficient_cpu(), which identifies the CPU with the highest available capacity in each performance domain. This choice helps maintain lower operational frequencies, therefore saving energy. The decision process evaluates if moving the task to a new CPU will be more energy-efficient compared to its previous CPU.

The compute_energy() function estimates the energy impact of migrating the waking task, simulating the new task distribution across CPUs. The em_pd_energy() API from the EM framework calculates the expected energy consumption for each performance domain based on this simulated distribution.

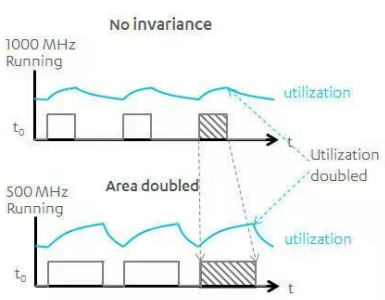



https://lkml.org/lkml/2015/8/14/296

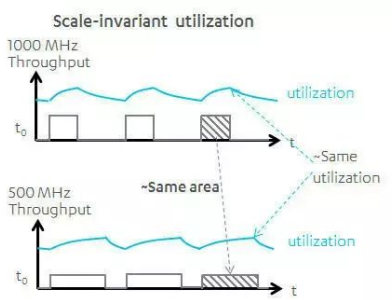
DVFS (cpufreq)

Disadvantage

* Only aware of the overall CPU loading, not aware of task migration

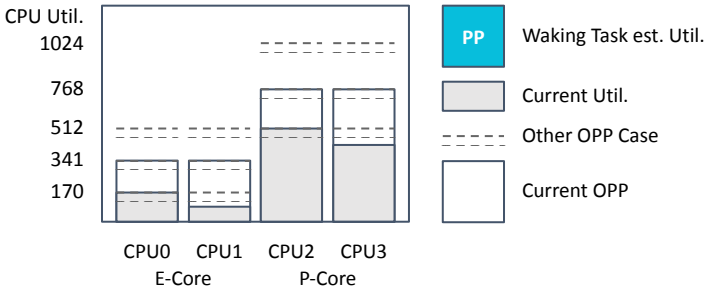


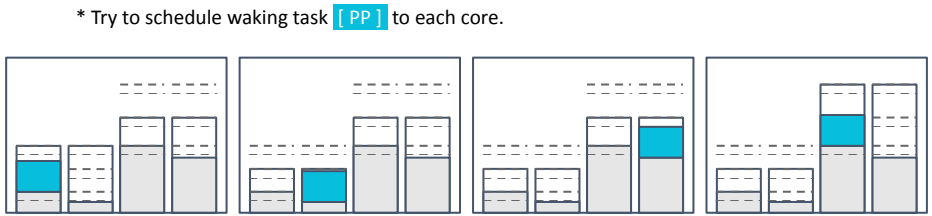

PELT



# EAS Example

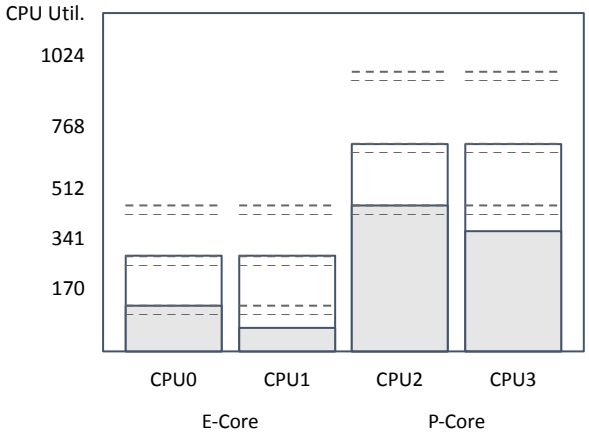
- A (fake) platform with 2 independent performance domains composed of two CPUs each. CPU0 and CPU1 are little CPUs; CPU2 and CPU3 are big.
- The scheduler must decide where to place a task P whose util_avg = 200 and prev_cpu = 0.
- The current utilization landscape of the CPUs is depicted on the graph below. CPUs 0-3 have a util_avg of 400, 100, 600 and 500 respectively. Each performance domain has three Operating Performance Points (OPPs). The CPU capacity and power cost associated with each OPP is listed in the Energy Model table. The util_avg of P is marked as ‘PP’.



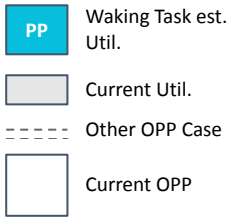

https://www.linaro.org/blog/energy-aware-scheduling-eas-progress-update/

# EAS Example



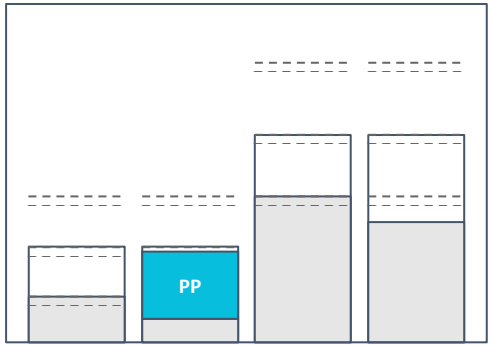

Energy Model (EM) All OPP Cases

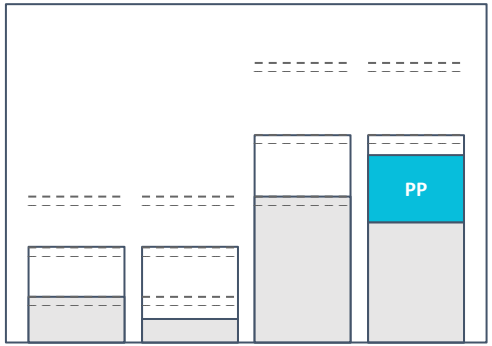
| E-Core | | P-Core | |
|---|---|---|---|
| Cap | Pwr | Cap | Pwr |
| 170 | – | 512 | – |
| 341 | – | 768 | – |
| 512 | – | 1024 | – |



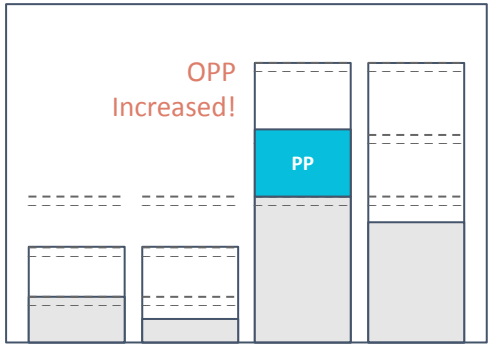

Schedule to CPU0


Schedule to CPU1




Schedule to CPU3




Schedule to CPU2

https://www.linaro.org/blog/energy-aware-scheduling-eas-progress-update/

# EAS Example

Total_energy = SUM(Energy(CPUn))
Energy(CPU) = Util. / Cap. * Pwr. * k

** k - allocation perference parameter*

Energy(CPU0) = 400 / 512 * 300 = 234
Energy(CPU1) = 100 / 512 * 300 = 58
Energy(CPU2) = 600 / 768 * 800 = 625
Energy(CPU3) = 500 / 768 * 800 = 520

**Total_energy = 1437**

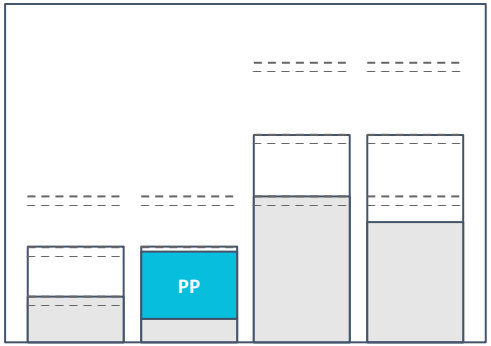
Energy(CPU0) = 200 / 341 * 150 = 88
Energy(CPU1) = 100 / 341 * 150 = 43
Energy(CPU2) = 600 / 768 * 800 = 625
Energy(CPU3) = 700 / 768 * 800 = 729

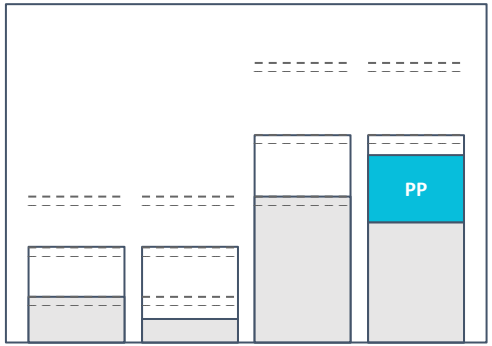
**Total_energy = 1485**

total_energy = 1437

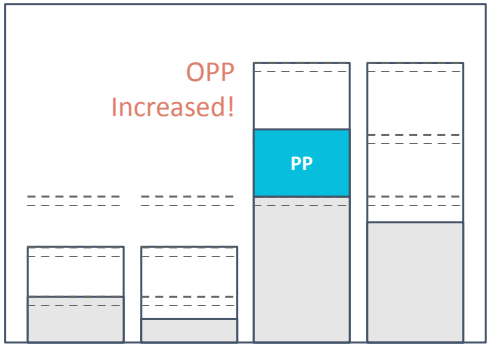

Winner \( ˚ヲ˚)/

total_energy = 1364




total_energy = 1485




total_energy = 2129

https://www.linaro.org/blog/energy-aware-scheduling-eas-progress-update/

# Other Scheduling Reference

**x86-64 & IA32:**

Intel Thread Director (ITD) + Hardware Feedback Interface (HFI) (Windows)

**64-bit ARM:**

Cluster-Aware Scheduling (Linux 5.16+ for Kunpeng SoCs)

https://kernel.org/doc/html/v5.18/x86/intel-hfi.html
https://www.phoronix.com/review/alderlake-p-e

# Thank you!