Embedded Linux Conference 2013

Task Scheduling for Multicore Embedded Devices

2013. 02. 22.

Gap-Joo Na
(funkygap@etri.re.kr)

ETRI
한국전자통신연구원
Electronics and Telecommunications Research Institute

Contents

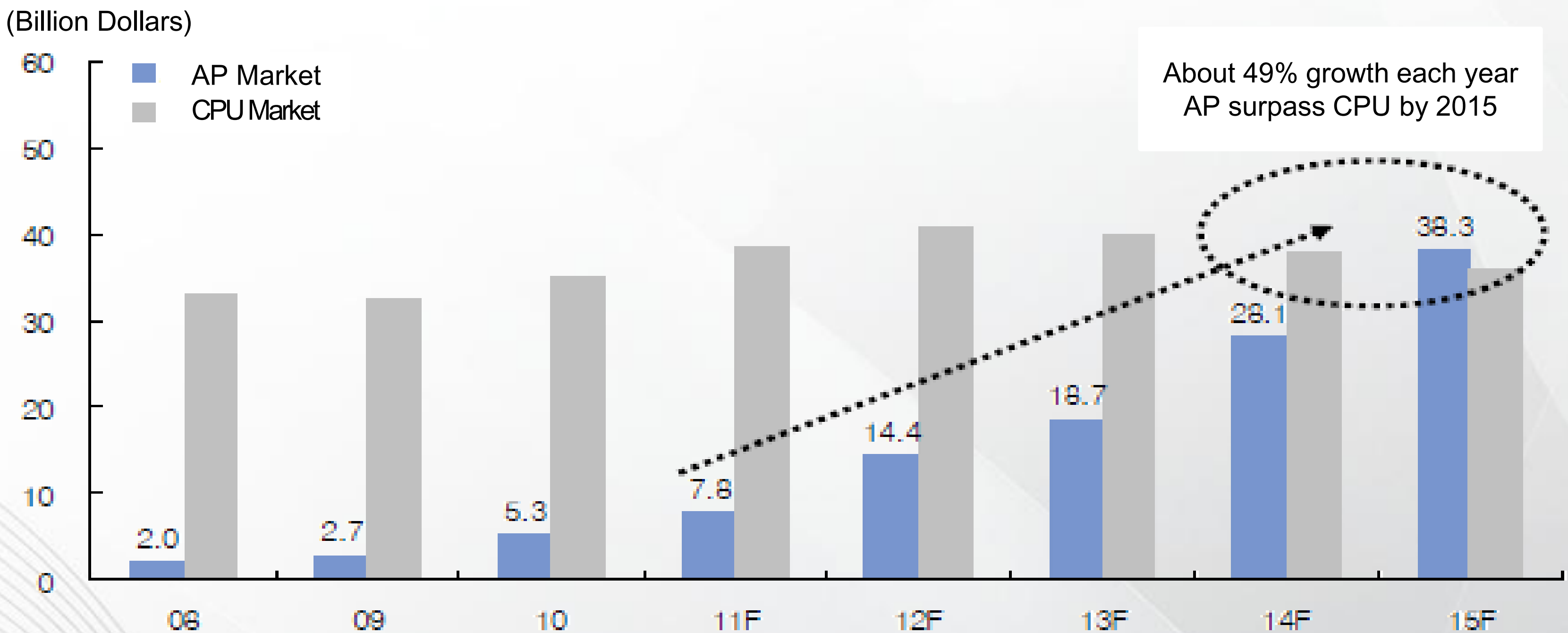
Background

What is multicore??

1. Multicore trends
2. New Architectures
3. Software Support

Multicore Trends on Embedded Devices

❖ Multicore based Application Processor market is expanding

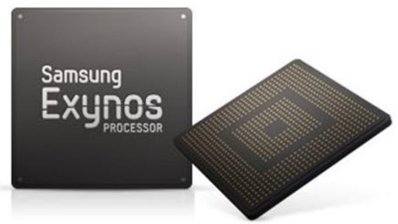


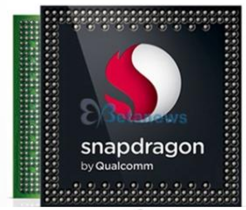
Global Server/Desktop CPU and Application Processor Market Forecast

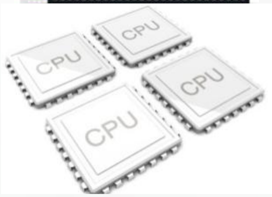
Multicore based Embedded Devices

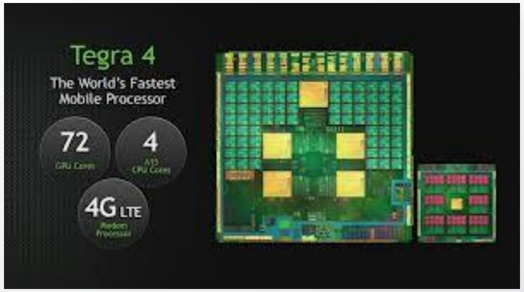
Multicore based embedded products have been come rapidly

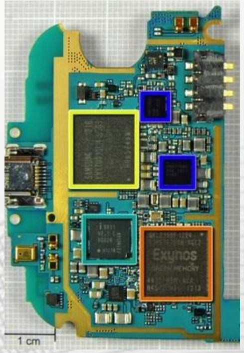
- Ex) Quad-core CPU:Exynos 4412, Snapdragon S4, Tegra 4, etc









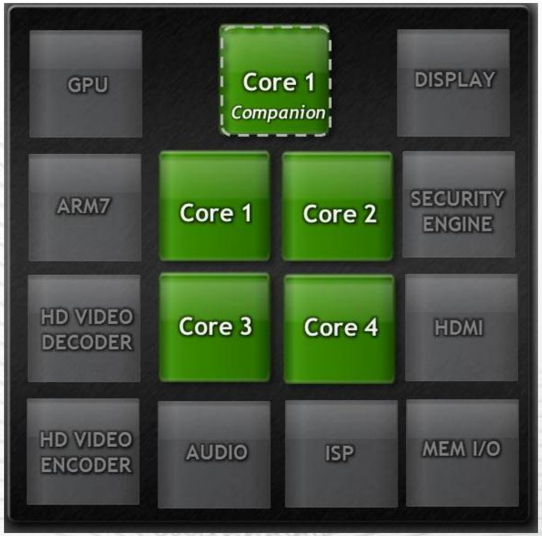


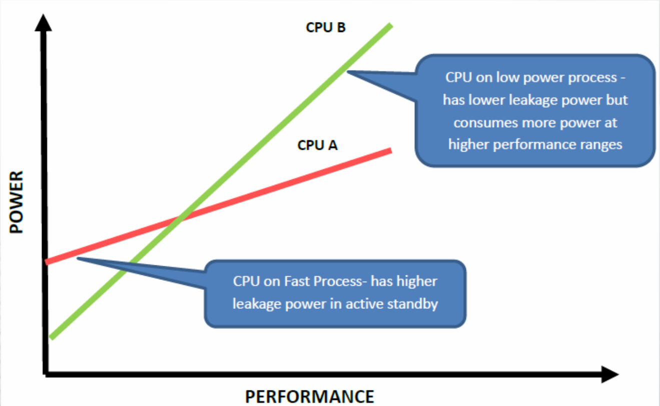
New Mobile Processor Archtecture [1]

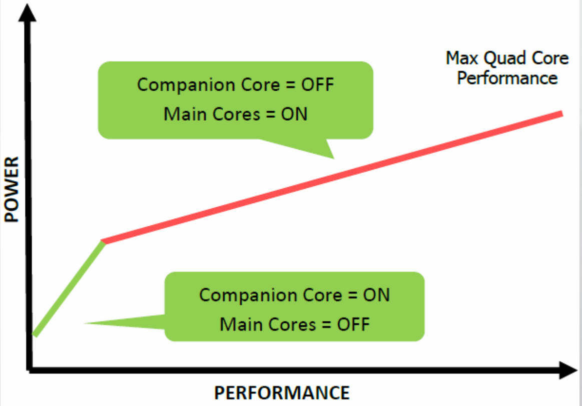
nVidia Tegra3, 'Variable SMP' :

- A Multi-Core CPU Architecture for Low Power and High Performance

	Power optimized Companion CPU Core	Performance optimized main CPU Cores
Architecture	Cortex A9	Cortex A9
Process Technology	Low Power (LP)	General/Fast (G).
Operating Frequency Range	0 MHz to 500 MHz	0 MHz to Max GHz







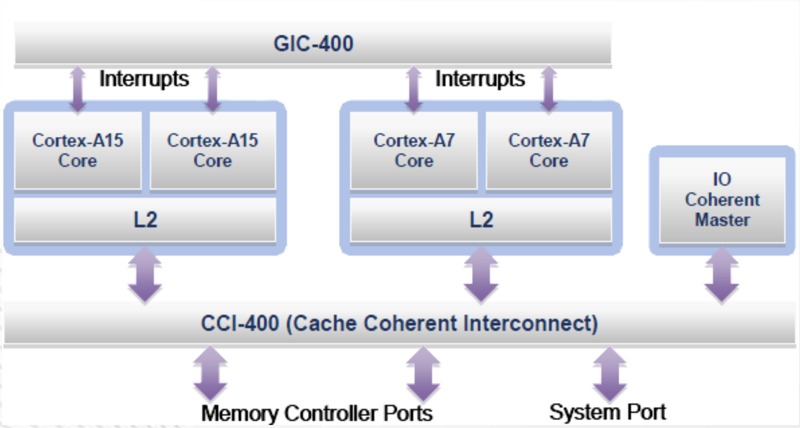
Tegra3 Architecture

Power-Performance gain curve of vSMP technology

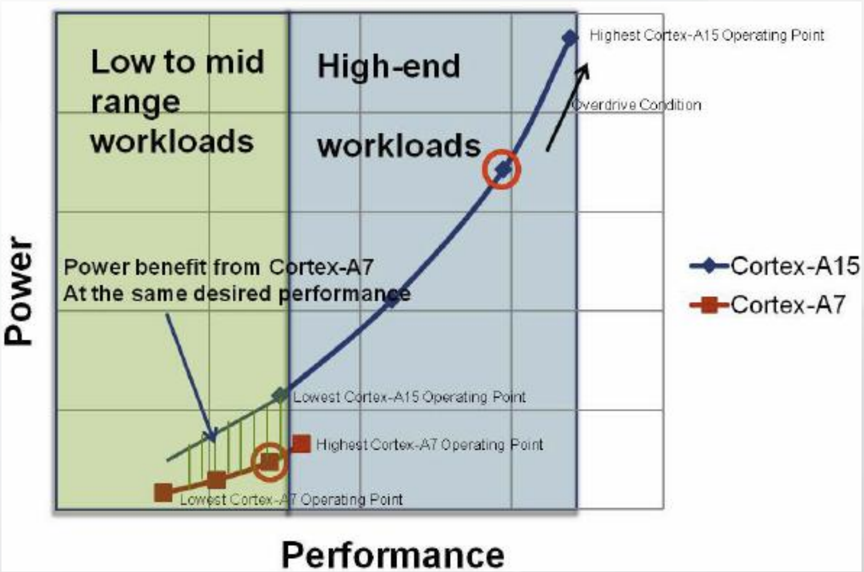
New Mobile Processor Archtecture [2]

arm big.LITTLE solution :

- Cortex-A15 and Cortex-A7 are ISA identical
- Cortex-A15 achieves high performance
- Cortex-A7 is highly energy efficient
- Samsung, Exynos 5410 : 4 big cores(Cotex-A15) + 4 small cores(Cotex-A7)



big.LITTLE Architecture



Cortex-A15-Cortex-A7 DVFS Curves

Software Issues in Multicore

Insufficient adaptation for multicore due to high complexity of multicore software development

- Useful programing libraries and models are needed for multicore
  - Ex) OpenMP, OpenCL, …
- More software development tools are needed for multicore
- Continuously, enhanced OS features are needed for multicore
  - Load balancing issue, Cache affinity,…

New Software Issues in Multicore

- Traditional kernel technique
- Energy efficient SW technique
- Heterogeneous SW technique
- Virtual SW technique

History of Linux Scheduler

1. Before v2.6
2. After v2.6
3. Current Scheduler

Scheduler Before Kernel V2.6

Version 1.2

- User circular queue & Minimal design
- Round-Robin scheduling policy
  - Ring type runqueue for runnable task

Version 2.2

- Scheduling class supporting
  - Real-Time, Non Real-Time Task Class
- Including SMP(Symmetric Multiprocessing) support

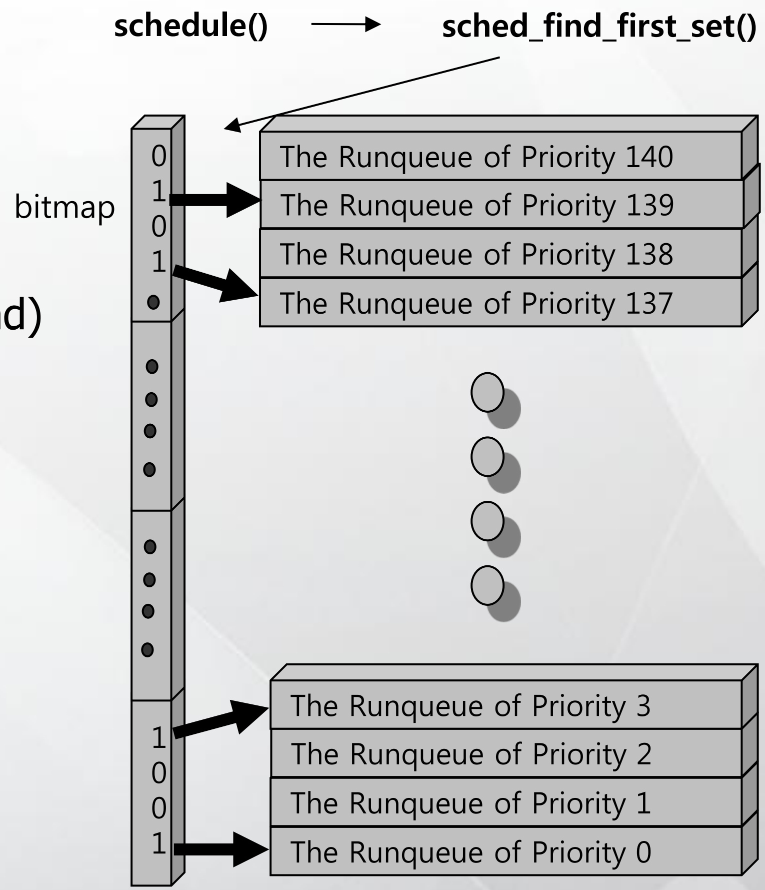
Version 2.4

- Lack of scalability
- Weak for real-time systems
- Single runqueue supporting
  - Throughput oriented design
  - O(N) complexity: the time it takes to schedule a task is a function of the number of tasks in the system

The early 2.6 Scheduler

O(1) Scheduler

- O(1) complexity supporting : using bitmap operation
- Dual runqueues
  - Active run queue
  - Expired run queue
- Much more scalable
- Incorporated interactivity metrics
  - Numerous heuristics (I/O , processor bound)



The early 2.6 scheduler data structure

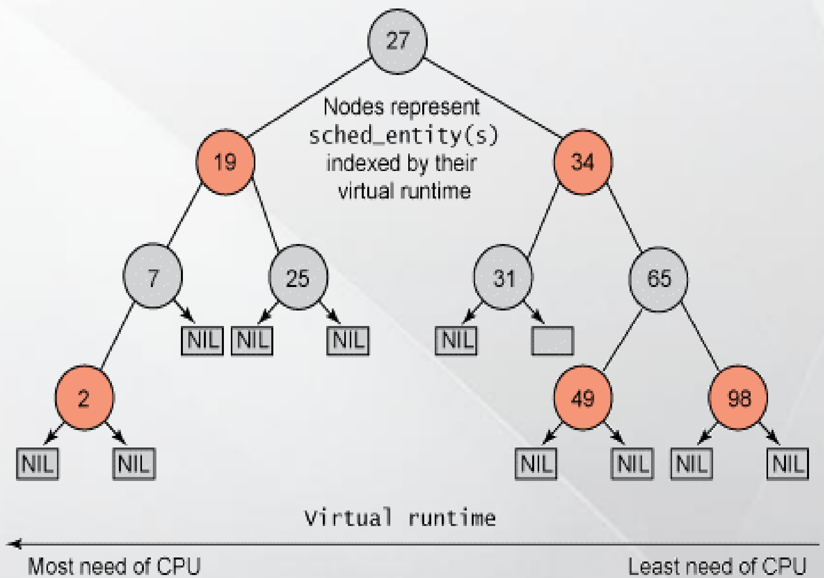
Problems of O(1)

- Slow response time
  - Frequent time slice allocation
- Throughput fall
  - Excessive switching overhead
- None fair condition
  - Nice 0 (100ms), Nice 1(95ms) => 5%
  - Nice 18(10ms), Nice 19(5ms) => 50%

Current Scheduler

Completely Fair Scheduler

- The main idea is to maintain balance(fairness) in providing processor time to tasks
- To determine the balance, the CFS maintains the amount of time provided to a given task in what's called the 'virtual time'
- The CFS maintains a time-ordered red-black tree
  - Self balancing
  - O(log n) time complexity
- SMP affinity
- Basic load balancing
- Priorities and CFS
- CFS group scheduling (after 2.6.24)



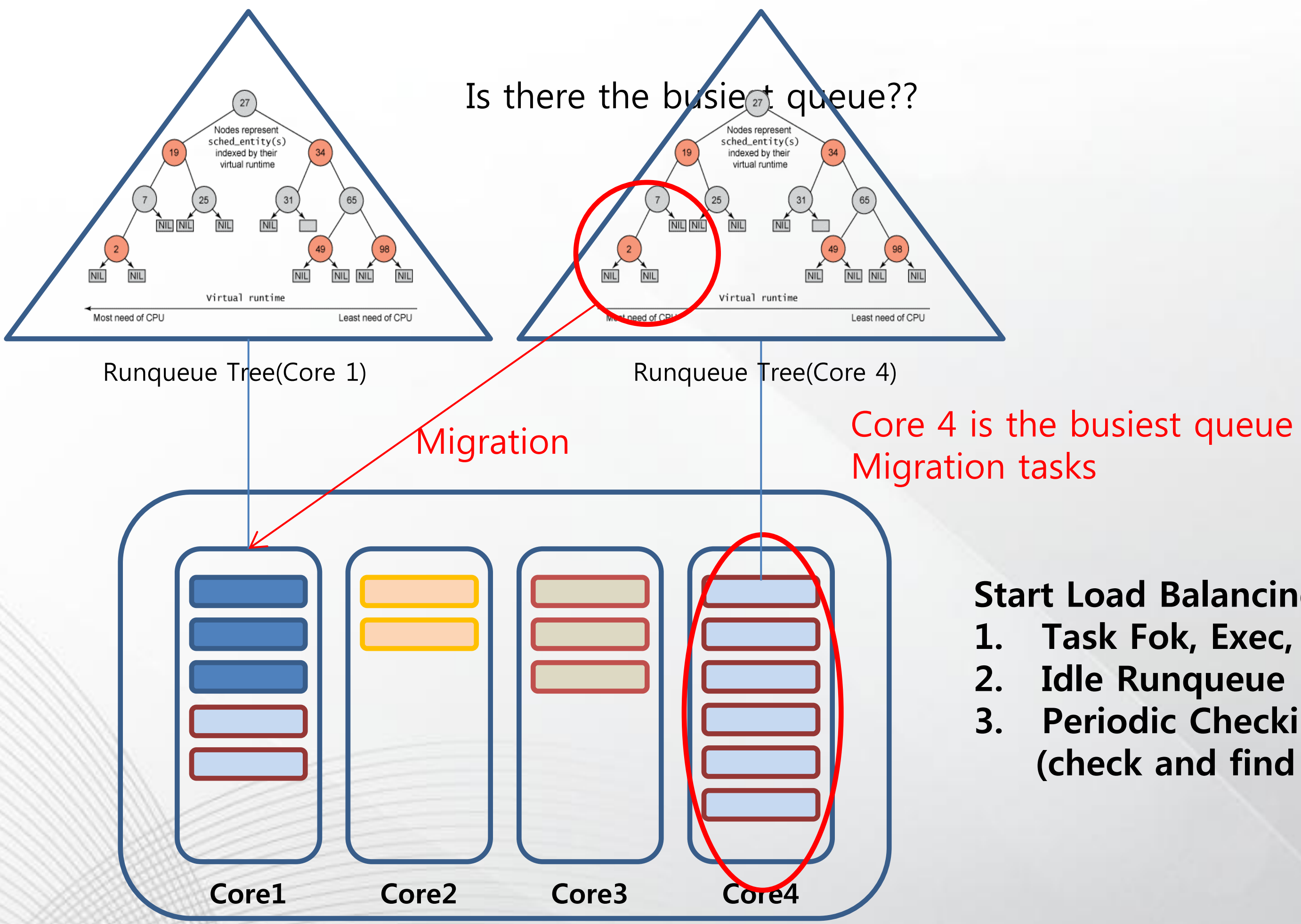
Example of a red-black tree

CFS(Completely Fair Scheduler)

1. Load Balancing
2. Limitations
3. Requirement of multicore embedded devices

Load Balancing of CFS (1)

Supports Basic Load Balancing Feature



Start Load Balancing in CFS

1. **Task Fok, Exec, Wakeup**
2. **Idle Runqueue**
3. **Periodic Checking Algorithm (check and find busiest runqueue)**

Load Balancing of CFS (2)

Completely Fair Scheduler

- Load of runqueue :

$$L_k = \sum_{\tau_i \in S_k} W(\tau_i)$$

- Amount of load to be moved :

$$L_{imbal} = min(min(L_{busiest}, L_{avg}), L_{avg} - L_k)$$

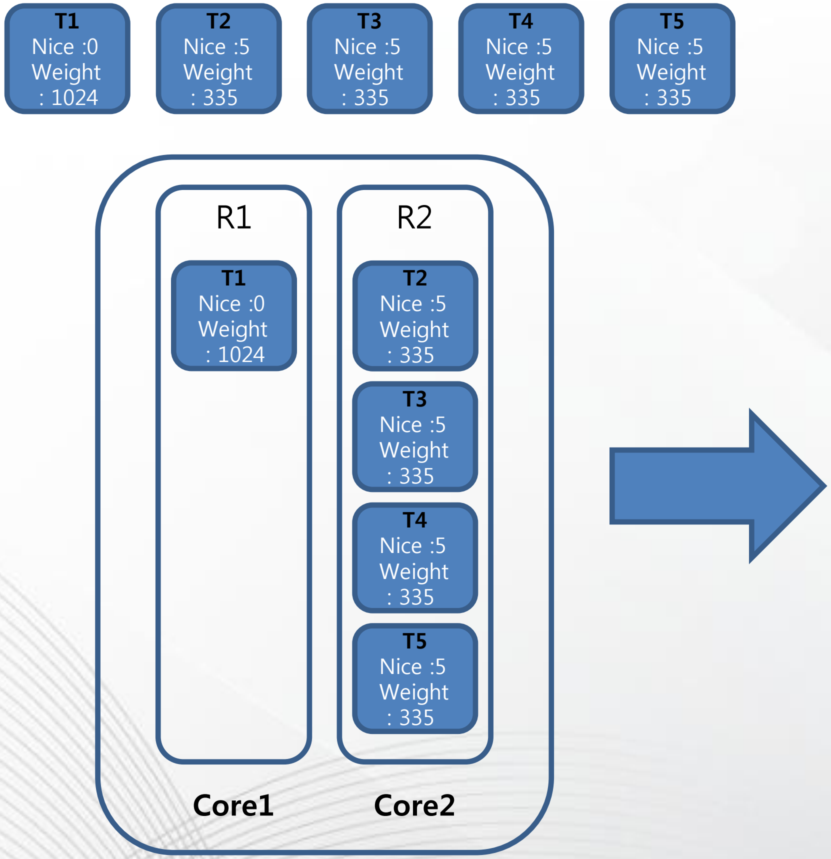
- CFS does not move any task if the following condition holds :

$$L_{imbal} < \min_{\tau_i \in S_{busiest}} (W(\tau_i)) / 2$$

Limitations of CFS (1)

Weight-based Algorithm

- Fail to achieve fairness in multicore



Weight of R1 (runqueue of core1) : 1024
Weight of R2 (runqueue of core2) : 335 * 4 = 1340
Average of runque Load : 1182

$$L_k = \sum_{\tau_i \in S_k} W(\tau_i)$$

$$L_{imbal} = min(min(L_{busiest}, L_{avg}), L_{avg} - L_k)$$

L_{imbal} = min(min(1340, 1182), 1182 – 1024)
= 158

But, 158 < 335/2 $L_{imbal} < \min_{\tau_i \in S_{busiest}} (W(\tau_i)) / 2$

Load Balancing will not be performed

T1 weight = (T2~T5) weight X 3

Run Time of T1 = Run Time of (T2~T5) X 4

=> Fairness will be broken.

Multicore and CFS

Multicore Scheduler

- Load Balancing
  - The most effective distribution is to have equal amounts of each core
  - Global fairness is most important

- Caches of Processors
  - CPU-affinity should be considered
  - Cache effectiveness vs. Global fairness

Embedded Devices

- I/O intensive processing
- Small number of tasks
- Foreground vs. Background Task
- Interactive task (touch screen GUI)
- Energy efficient
- Web-based application

It's time to rethink the previous task scheduler for multicore embedded devices

Case Study (DWRR)

1. Introduction
2. Basic Concept
3. Operation
4. Weak Points

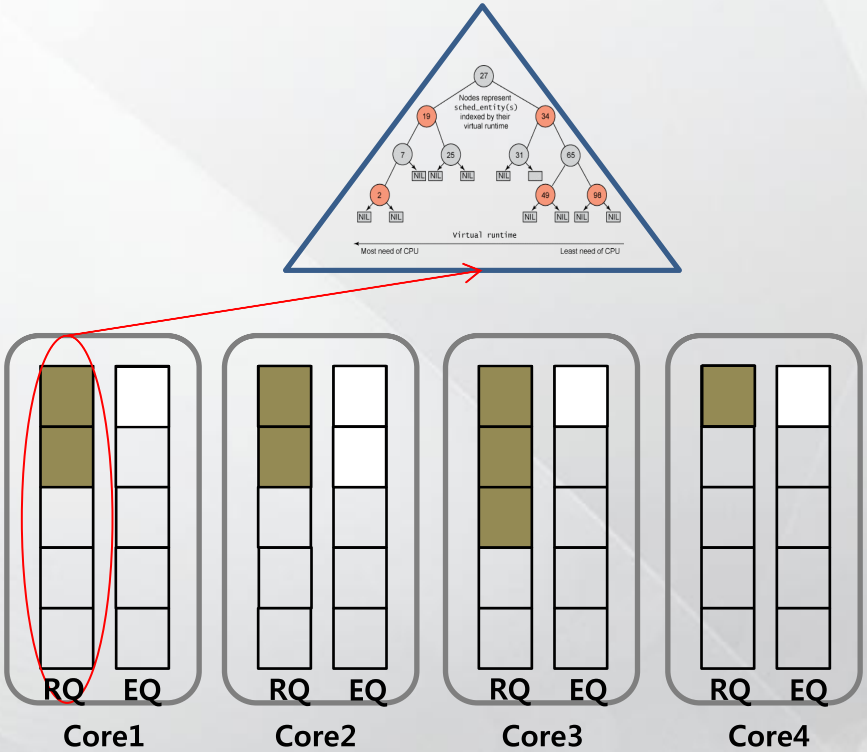
DWRR(Distributed Weighted Round-Robun)

Main Goal : Enhances Global Task Fairness based on Multicore

- Li, T., Baumberger, D., and Hahn, S.: "Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin", ACM SIGPLAN Notices, 2009, 44, (4), pp. 65-74

Key Idea :

- Manages task fairness every round



RQ : runqueue
EQ : Expired RQ
All queue : red-black Tree

Basic Concept

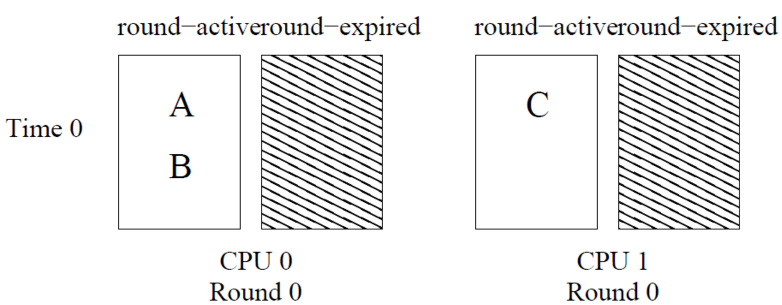
Local Fairness

- Round : the shortest time period during which every thread in the system completes at least one of its round slice
- Round Slice : w*B (w : thread’ s weight, B : system-wide constant)

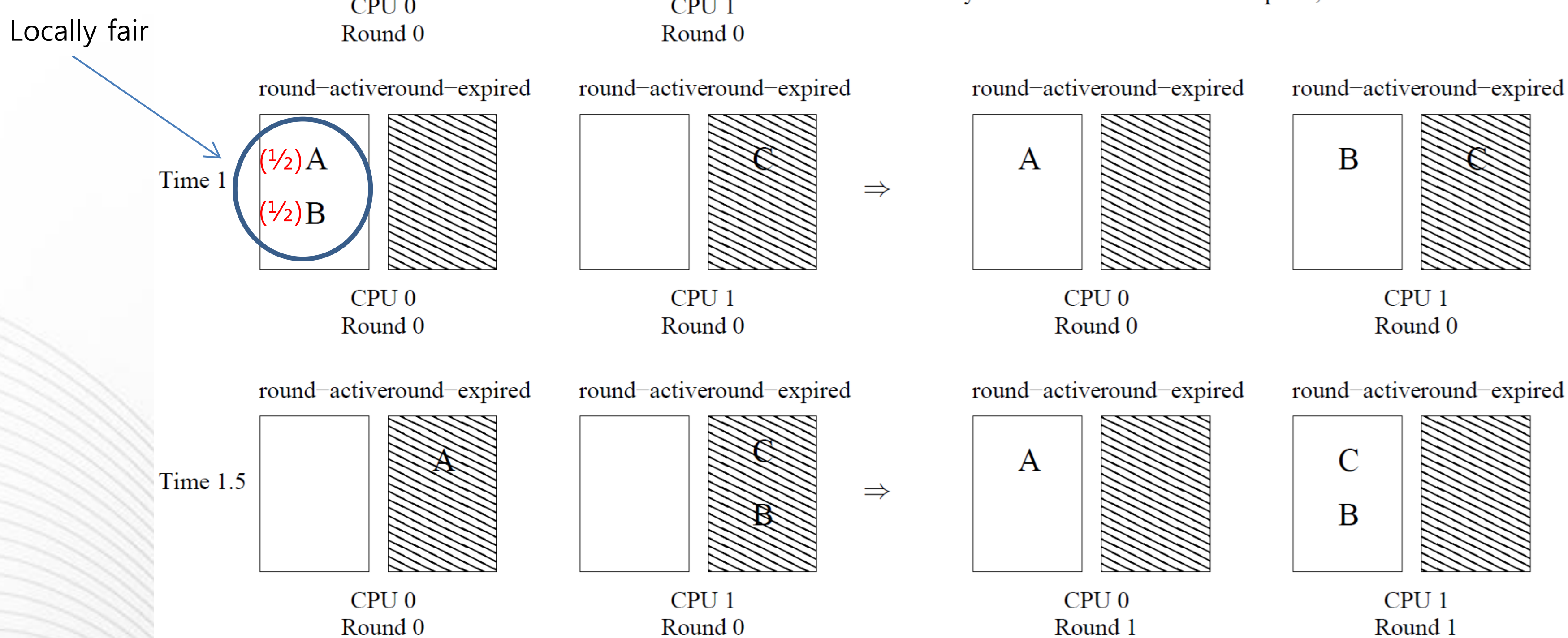
Global Fairness

- Round Balancing
  - It allows threads to go through the same number of rounds in any time interval.
  - Whenever a CPU finishes round balancing to move over threads from other CPUs before advancing to the next round

Operation of DWRR



Assume A, B, and C have weight one and round slice of one time unit.
Time 0: A and B start on CPU 0, C on CPU 1.
Time 1 (left): A and B each finish half a round and remain in round−active.
C finishes one round and moves to round−expired.
Time 1 (right): CPU 1 performs round balancing and moves B over.
Time 1.5 (left): A and B both finish one round and move to round−expired.
Time 1.5 (right): Both CPU 0 and 1 have nothing to do for round balancing.
So they switch round−active and −expired, and advance to next round.



Source : Li, T., Baumberger, D., and Hahn, S.: "Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin", ACM SIGPLAN Notices, 2009, 44, (4), pp. 65-74

DWRR Weak Points

Possibility or Riskiness

- DWRR can always guarantee higher fairness among tasks
- But, DWRR may suffer from poor interactivity due to the existence of two runqueues originated from O(1) scheduler
- Frequent task migration may cause migration overhead
- DWRR has several practical implementation issues

Experiments (CFS vs. DWRR)

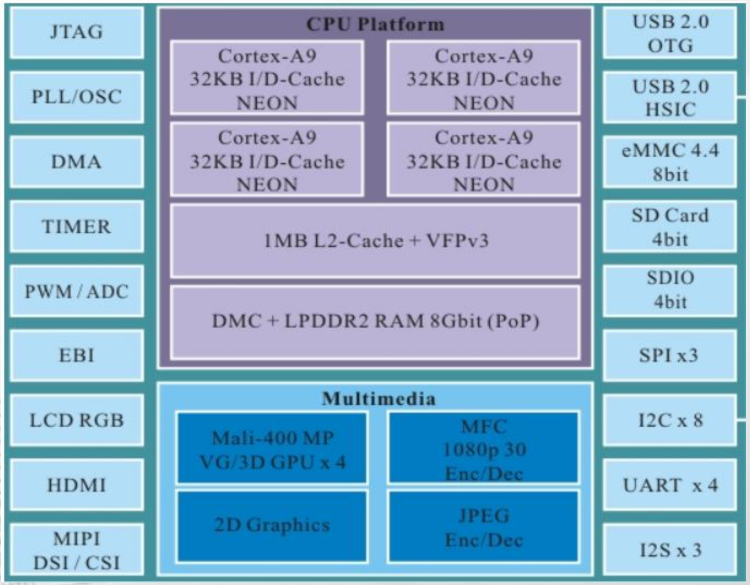
1. Test Environment
2. Fairness Test
3. Scheduler Benchmark
4. CPU Intensive Workload
5. Database Workload
6. JavaScript Benchmark

Test Environment

H/W and S/W

- Target Board : OdroidQ (hardkernel)
  - Exynos 4412 ARM Cotex-A9 Quad Core
  - Linaro Ubuntu 12.04
  - Kernel version 3.0.41
  - CFS (sched_min_granularity = 0.75ms, sched_latency = 6ms, sched_nr_latency = 8)
  - DWRR (round slice = 25msec)

Architecture



Arm Quad Core Architecture

Target System

Fairness Test

Global Fairness :

- **Test Method**
  - **Creates and runs 5 threads on 4 multicores**
  - **Measures average utilization of each cores and calculates standard deviation**

CFS (3.0.15)

```
top - 19:05:25 up 8 min,  2 users,  load average: 5.01, 3.56, 1.68
Tasks: 149 total,   6 running, 143 sleeping,   0 stopped,   0 zombie
Cpu(s): 99.3%us,  0.5%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:    920668k total,   456572k used,   464096k free,    40976k buffers
Swap:        0k total,        0k used,        0k free,   193944k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5318 root      20   0  1168  232  184 R   99  0.0   4:59.27 test_while
 5317 root      20   0  1168  232  184 R   96  0.0   4:37.08 test_while
 5319 root      20   0  1168  232  184 R   76  0.0   3:57.92 test_while
 5321 root      20   0  1168  232  184 R   74  0.0   3:55.35 test_while
 5320 root      20   0  1168  232  184 R   52  0.0   4:17.00 test_while
 4160 root     -99   0     0    0    0 S    1  0.0   0:04.35 dhd_dpc
 5322 root      20   0  2160  984  700 R    1  0.1   0:01.95 top
 3962 root      20   0  5676 2388 1788 S    0  0.3   0:00.14 modem-manager
 4159 root     -98   0     0    0    0 S    0  0.0   0:01.02 dhd_watchdog
 4691 linaro    20   0  126m  12m 8848 S    0  1.4   0:01.65 gnome-settings-
 5555 root      20   0     0    0    0 S    0  0.0   0:00.31 kworker/0:0
    1 root      20   0  3244 1644  980 S    0  0.2   0:03.52 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S    0  0.0   0:00.00 ksoftirqd/0
    6 root      RT   0     0    0    0 S    0  0.0   0:00.00 migration/0
    7 root      RT   0     0    0    0 S    0  0.0   0:00.00 migration/1
    8 root      20   0     0    0    0 S    0  0.0   0:00.00 kworker/1:0
```

CFS with DWRR (3.0.15)

```
top - 18:52:16 up  1:49,  3 users,  load average: 4.78, 3.60, 1.71
Tasks: 150 total,   6 running, 144 sleeping,   0 stopped,   0 zombie
Cpu(s): 99.0%us,  0.3%sy,  0.0%ni,  0.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    920668k total,   534972k used,   385696k free,    31296k buffers
Swap:        0k total,        0k used,        0k free,   261960k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
14212 linaro    20   0  1168  232  184 R   80  0.0   4:52.91 test_while
14215 linaro    20   0  1168  232  184 R   79  0.0   4:56.06 test_while
14213 linaro    20   0  1168  232  184 R   79  0.0   4:53.91 test_while
14211 linaro    20   0  1168  232  184 R   79  0.0   4:59.32 test_while
14214 linaro    20   0  1168  232  184 R   79  0.0   4:55.01 test_while
 5148 root      20   0  2960 1184  312 S    1  0.1   0:08.86 udevd
 4720 colord    20   0 50148 9244 6940 S    0  1.0   0:01.17 colord
 4740 linaro    20   0  216m  50m  25m S    0  5.6   0:08.59 unity-2d-shell
14718 root      20   0  2160  980  700 R    0  0.1   0:00.31 top
    1 root      20   0  3112 1624  980 S    0  0.2   0:06.27 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S    0  0.0   0:00.05 ksoftirqd/0
    6 root      RT   0     0    0    0 S    0  0.0   0:00.00 migration/0
    7 root      RT   0     0    0    0 S    0  0.0   0:00.00 migration/1
    8 root      20   0     0    0    0 S    0  0.0   0:00.00 kworker/1:0
    9 root      20   0     0    0    0 S    0  0.0   0:00.06 ksoftirqd/1
   10 root      RT   0     0    0    0 S    0  0.0   0:00.00 migration/2
```

Scheduler Performance

Scheduling Latency Benchmark : Sysbench (test : threads)

- **When a scheduler has a large number of threads competing for some set of mutexes**

- **Commnad :**
  - ***sysbench –num-threads=32 –test=threads –thead-yields=100 0–thread-locks= 8 run***

scheduler	CFS (sched_granularity = 0.75)	CFS (sched_granularity = 0.5)	CFS (sched_granularity = 0.25)	DWRR (round_slice unit = 0.25)
Total time	12.8319s	13.0980s	21.3573s	7.4515s
Total Number of Events	10000	10000	10000	10000
Total time taken by event execution	410.0162	418.2351	682.6435	237.6006
Threads Fairness	Events (avg/stddev) : 312.5000/9.67 Execution time (avg/stddev) : 12.8130/0.01	Events (avg/stddev) : 312.5000/10.60 Execution time (avg/stddev) : 13.0698/0.01	Events (avg/stddev) : 312.5000/6.98 Execution time (avg/stddev) : 21.3326/0.01	Events (avg/stddev) : 312.5000/41.63 Execution time (avg/stddev) : 7.4250/0.01

CPU Intensive Test

Experiments

Video Codec Processing

- **Mplayer –benchmark –nosound –ao null –vo null robot_720p.mp4**
  - **Running time : 150s**

scheduler	CFS (sched_granularity = 0.75)	CFS (sched_granularity = 0.5)	CFS (sched_granularity = 0.25)	DWRR (round_slice unit = 0.25)
BenchmarkS	Video Codec : 38.011s Video Out : 0.016s Audio : 0s Sys : 1.138s Total : 39.165s	Video Codec : 39.562s Video Out : 0.017s Audio : 0s Sys : 2.381s Total : 41.960s	Video Codec : 40.206s Video Out : 0.017s Audio : 0s Sys : 6.104s Total : 46.327s	Video Codec : 26.580s Video Out : 0.014s Audio : 0s Sys : 1.024s Total : 27.617s
Benchmark%	Video Codec : 97.0533% Video Out : 0.0409% Sys : 2.9057% Total : 100%	Video Codec : 94.2858% Video Out : 0.0403% Sys : 5.6739% Total : 100%	Video Codec : 86.7869% Video Out : 0.0362% Sys : 13.1769% Total : 100%	Video Codec : 96.2441% Video Out : 0.0489% Sys : 3.7070% Total : 100%

Database Workload

Experiments

Real database workload (Online Transaction Process)

- **Benchmark : Sysbench, Database : Mysql**
  - sysbench --test=oltp --mysql-user=sbtest --mysql-password=sbtest --mysql-table-engine=myisam --oltp-table-size=1000000 --mysql-socket=/var/run/mysqld/mysqld.sock prepare
  - sysbench --test=oltp --mysql-user=sbtest --mysql-password=sbtest --oltp-table-size=1000000 --mysql-socket=/var/run/mysqld/mysqld.sock --max-requests=100000 --oltp-read-only --num-threads=16 run

scheduler	CFS (sched_granularity = 0.75)	CFS (sched_granularity = 0.5)	CFS (sched_granularity = 0.25)	DWRR (round_slice unit = 0.25)
Query Performed	Read : 1400644 Write : 0 Other : 200092 Total : 1600736	Read : 1400560 Write : 0 Other : 200080 Total : 1600640	Read : 1400616 Write : 0 Other : 200088 Total : 1600704	Read : 1400896 Write : 0 Other : 200128 Total : 1601024
Transactions	100046 (273.63 per sec.)	100040 (322.05 per sec.)	100044 (389.61 per sec.)	100064 (353.96 per sec.)

JavaScript Benchmark

V Experiments

SunSpider Java script Benchmark (http://www.webkit.org/perf/sunspider/sunspider.html)

scheduler	CFS (sched_granularity = 0.75)	CFS (sched_granularity = 0.5)	CFS (sched_granularity = 0.25)	DWRR (round_slice unit = 0.25)
Total	1533.6ms +/- 0.8%	2289.1ms +/- 0.6%	2284.4ms +/- 0.6%	1533.2ms +/- 1.3%
3d	265.7ms +/- 1.5%	374.0ms +/- 1.5%	371.6ms +/- 1.8%	273.4ms +/- 2.9%
access	191.0ms +/- 2.8%	299.7ms +/- 2.8%	301.9ms +/- 4.0%	194.3ms +/- 4.8%
bitops	102.5ms +/- 2.3%	178.1ms +/- 3.3%	176.1ms +/- 2.5%	105.6ms +/- 7.0%
controlflow	14.5ms +/- 4.2%	23.1ms +/- 6.6%	22.2ms +/- 3.0%	14.4ms +/- 5.3%
cryoto	148.1ms +/- 6.8%	190.5ms +/- 1.6%	192.7ms +/- 2.4%	145.2ms +/- 2.7%
date	200.3ms +/- 6.7%	283.9ms +/- 2.7%	288.6ms +/- 2.0%	199.7ms +/- 4.6%
math	205.0ms +/- 1.3%	207.3ms +/- 2.2%	205.0ms +/- 1.3%	124.2ms +/- 4.6%
regexp	123.1ms +/- 1.1%	99.3ms +/- 2.0%	99.4s +/- 3.3%	65.2ms +/- 5.1%
string	417.3ms +/- 2.2%	633.2ms +/- 1.1%	626.9ms +/- 0.7%	411.2ms +/- 1.7%

Conclusion

Conclusion and future work

Conclusion

- Multicore processors are becoming an integral part of embedded devices
- In Linux, CFS is the best scheduler until now
  - CFS performs load balancing depending on task's weight
  - The weight-based algorithms fails to achieve global fairness in practical
- DWRR can be new trial to improve the multicore in terms of fairness
- Rethink the scheduler for multicore embedded devices.

Future Work

- Optimal load balancing algorithm
- Enhanced runqueue structure
- Per core scheduler policy

Q & A

Thank You

ETRI
한국전자통신연구원
Electronics and Telecommunications Research Institute