RTEMS SMP Features

Joel Sherrill, Ph.D.

Joel.Sherrill@oarcorp.com
OAR Corporation
Huntsville Alabama USA

December 2014

Background

- SMP activities and features not covered
- Thread affinity
  - Classic and POSIX API Additions
- Currently Available SMP Thread Schedulers
- Clustered Scheduler
  - Classic API Additions
- SMP Deployment Examples
- Multiprocessor Resource Sharing Protocol (MRSP)
- Conclusion

Summary of SMP Activities

- High SMP activity level over past 18 months
- Combination of ESA, commercially sponsored, and Google Summer of Code activities
- SMP on SPARC, PowerPC, ARM, and x86
- Emphasis on performance and sound theoretical foundation

Miscellaneous SMP Related Features

- C11 and C++11 atomic operations support
- Locking in support libraries made SMP safe
- OS internal lock profiling added
- Per task variables disable in SMP configuration
- POSIX keys reworked and optimized
- New SMP tests added
- Many patches upstream to GCC and newlib

Thread Affinity

- Affinity is the binding of a thread to a subset of the available cores in an SMP system
- Scheduler must be aware of and honor affinity
- Can result in thread migrations
- Requires APIs to
  - Build and manipulate CPU sets
  - Dynamically alter thread affinity

Must be careful to avoid priority inversions and ensure schedulability.

<sys/cpuset.h> APIs

- Linux compatible with *BSD extensions
- Methods to build and manipulate both fixed and variable sized *cpu_set_t* instances
- Fixed is currently up to 32 cores
- Many (27) operations named `CPU_op` for fixed size and `CPU_op_S` for variable sized (when applicable)
  - *op* is one of ZERO, FILL, SET, CLR, ISSET, AND, OR, XOR, NAND, COUNT, EQUAL, ALLOC, FREE, CMP, EMPTY, COPY

Classic API Affinity APIs

- Use <sys/cpuset.h> APIs to build CPU set
- Only two methods

```
rtems_status_code rtems_task_get_affinity(
  rtems_id                  id,
  size_t                    cpusetsize,
  cpu_set_t                *cpuset
);

rtems_status_code rtems_task_set_affinity(
  rtems_id            id,
  size_t              cpusetsize,
  const cpu_set_t *cpuset
);
```

- If the thread is executing, it may be migrated to another CPU or be unable to execute

POSIX API Affinity APIs

- Compatible with Linux, similar to *BSD
- More calls in API due to POSIX thread attributes

```
int pthread_attr_setaffinity_np(
  pthread_attr_t  *attr,
  size_t           cpusetsize,
  const cpu_set_t *cpuset
);

int pthread_attr_getaffinity_np(
  const pthread_attr_t *attr,
  size_t                cpusetsize,
  cpu_set_t            *cpuset
```

```
int   pthread_setaffinity_np(
  pthread_t        id,
  size_t           cpusetsize,
  const cpu_set_t *cpuset
);

int   pthread_getaffinity_np(
  const pthread_t id,
  size_t          cpusetsize,
  cpu_set_t      *cpuset
);
```

NOTE: POSIX requires non-portable methods to end with the suffix _*np*.

Obtain Current Attributes for a Thread

- Also added a method found on Linux and *BSD to obtain a thread's current attributes

```
int  pthread_getattr_np(
  pthread_t       id,
  pthread_attr_t *attr
);
```

Without this method, a thread has no way to obtain its current attributes.

RTEMS

Affinity Code Example

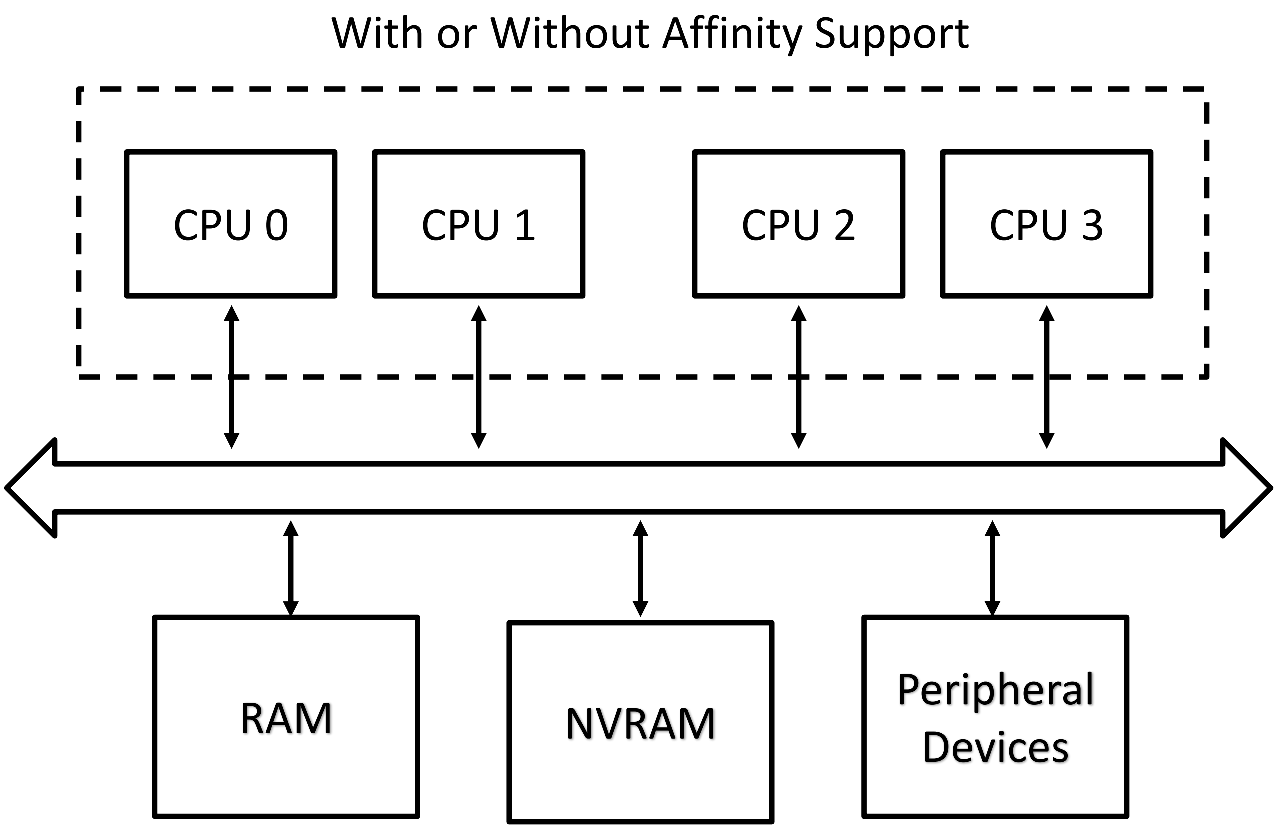
- Create a set of tasks where each is pinned to a specific CPU

```
cpu_set_t      cpuset;
uint32_t       i;
int            sc;
rtems_id       id[ NUM_CPUS ];

for (i=0; i<NUM_CPUS; i++){
  sc = rtems_task_create(
    rtems_build_name( 'T', 'A', '0', ('1' + i) ),
    10 + i,              /* vary the priority but locked to a cpu */
    RTEMS_MINIMUM_STACK_SIZE,
    RTEMS_DEFAULT_MODES,
    RTEMS_DEFAULT_ATTRIBUTES,
    &id[i]
  );

  /* set affinity before start to ensure it ONLY runs on proper CPU */
  CPU_ZERO(&cpuset);     /* Clear the cpuset to none set */
  CPU_SET(i, &cpuset);   /* Set only CPU i */

  sc = rtems_task_set_affinity(id[i], sizeof(cpu_set_t), &cpuset);
  sc = rtems_task_start(id[i], Task_1, i+1);
}
```

Available SMP Schedulers

- Priority SMP Scheduler
  - Deterministic Priority SMP Scheduler
- Priority Affinity SMP Scheduler
  - Deterministic Priority SMP Affinity Scheduler
- Simple SMP Scheduler
  - Simple SMP Priority Scheduler

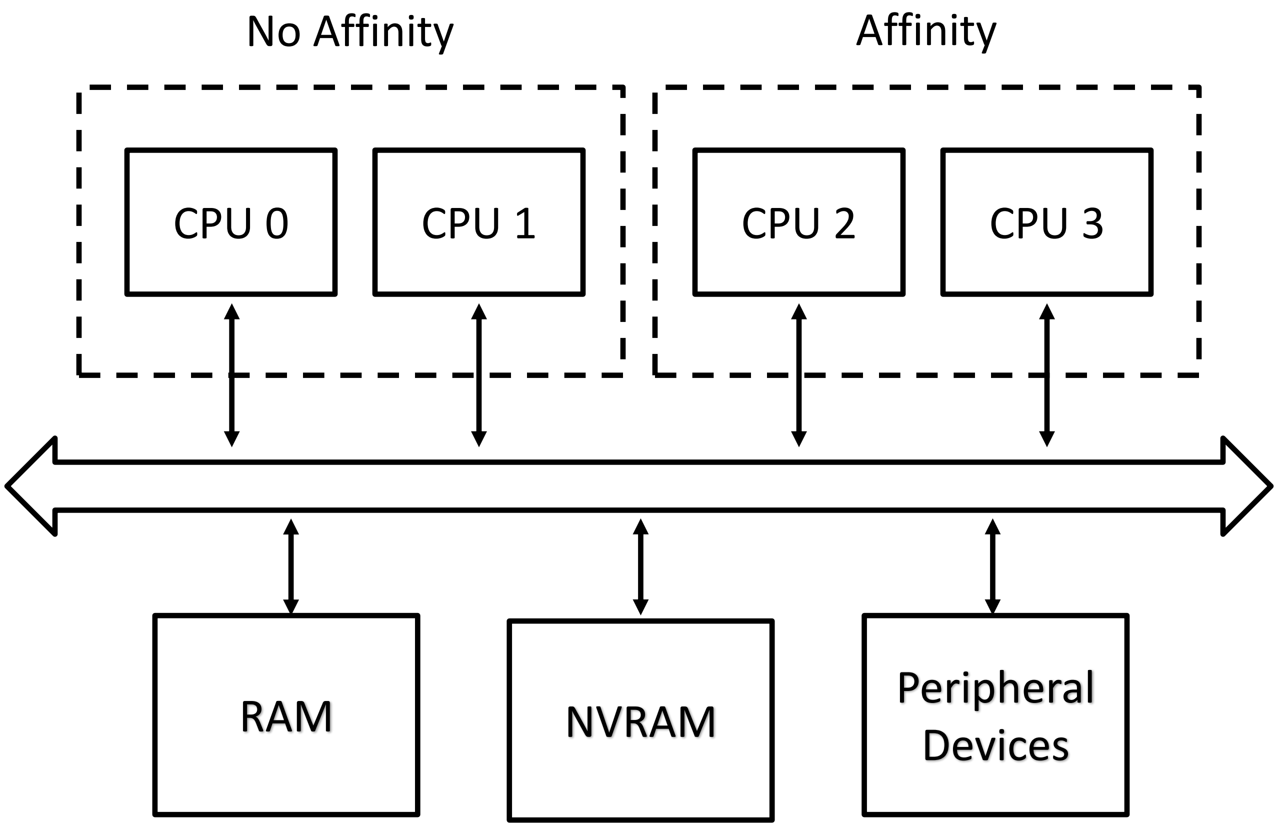
Single Scheduler Instance



Clustered Scheduling

- Single address space, single application but multiple thread schedulers for system
- Associate subset of cores with each instance
- Why would you do this?
  - Group sets of related threads
  - Lessen cache contention effects
  - Reduce complexity of schedulability analysis
  - Ease other resource conflicts
  - Leave core(s) idle for future growth
- Combined with affinity, you can have very fine-grained control over thread placement

Clustered Scheduling Example



RTEMS

Clustered Scheduling APIs

- Obtain name of scheduler instance

```
rtems_status_code rtems_scheduler_ident(
  rtems_name  name,
  rtems_id    *id
);
```

- Obtain and change scheduler instance associated with thread

```
rtems_status_code rtems_task_get_scheduler(
  rtems_id  task_id,
  rtems_id *scheduler_id
);

rtems_status_code rtems_task_set_scheduler(
  rtems_id task_id,
  rtems_id scheduler_id
);
```

- Obtain processor set associated with scheduler instance

```
rtems_status_code rtems_scheduler_get_processor_set(
  rtems_id   scheduler_id,
  size_t     cpusetsize,
  cpu_set_t *cpuset
);
```

Clustered Scheduling Configuration

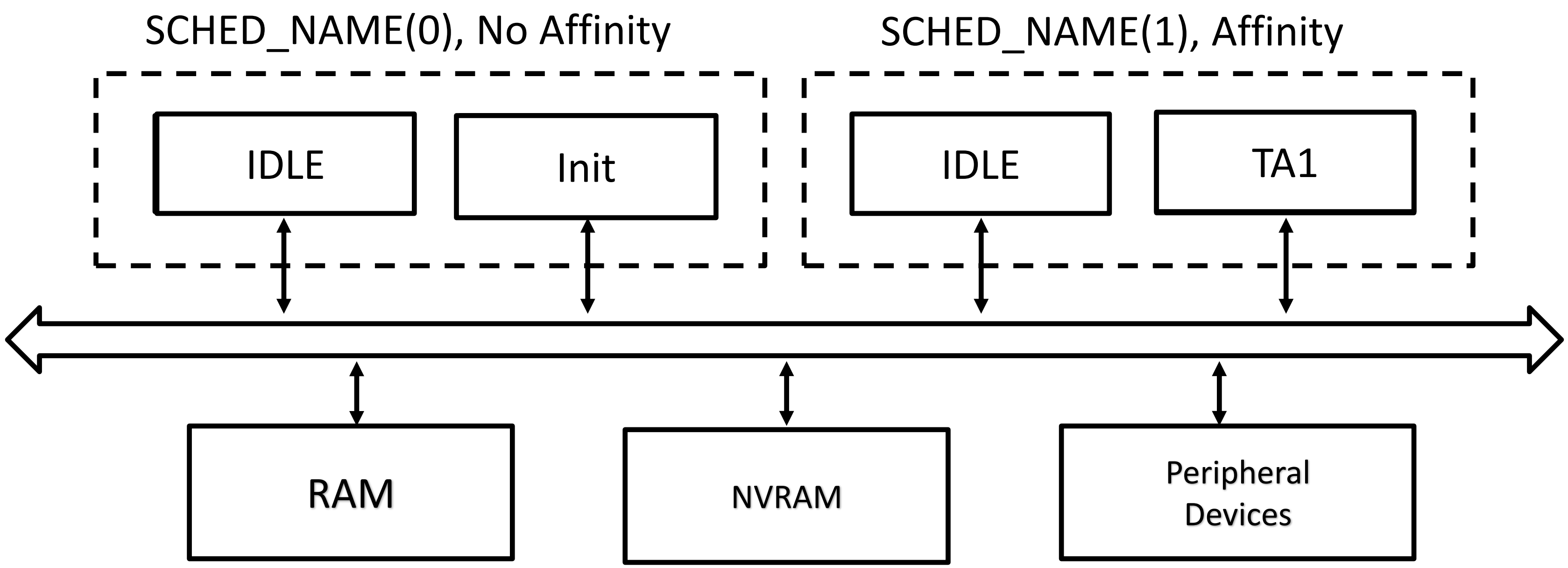
```
#define CONFIGURE_SMP_APPLICATION
#define CONFIGURE_SMP_MAXIMUM_PROCESSORS 4
#define CONFIGURE_SCHEDULER_PRIORITY_SMP
#define CONFIGURE_SCHEDULER_PRIORITY_AFFINITY_SMP

#include <rtems/scheduler.h>

shell_scheduler_name shell_scheduler_list[] = {
  "Priority SMP Scheduler",
  "Priority Affinity SMP Scheduler",
  ""
};

RTEMS_SCHEDULER_CONTEXT_PRIORITY_SMP( \
  a, CONFIGURE_MAXIMUM_PRIORITY + 1);
RTEMS_SCHEDULER_CONTEXT_PRIORITY_AFFINITY_SMP(
  b,
  CONFIGURE_MAXIMUM_PRIORITY + 1
);
```

```
#define CONFIGURE_SCHEDULER_CONTROLS \
 RTEMS_SCHEDULER_CONTROL_PRIORITY_SMP(a, \
   SCHED_NAME(0)), \
 RTEMS_SCHEDULER_CONTROL_PRIORITY_AFFINITY_SMP(b,\
   SCHED_NAME(1))

#define CONFIGURE_SMP_SCHEDULER_ASSIGNMENTS \
  RTEMS_SCHEDULER_ASSIGN(0, \
   RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY), \
  RTEMS_SCHEDULER_ASSIGN(0, \
    RTEMS_SCHEDULER_ASSIGN_PROCESSOR_OPTIONAL), \
  RTEMS_SCHEDULER_ASSIGN(1, \
    RTEMS_SCHEDULER_ASSIGN_PROCESSOR_OPTIONAL), \
  RTEMS_SCHEDULER_ASSIGN(1, \
    RTEMS_SCHEDULER_ASSIGN_PROCESSOR_OPTIONAL)
```

Moving a Thread to Another Scheduler

Scheduler instances and associated CPUs are configured by user
During RTEMS initialization, scheduler instances are initialized initialized and IDLE threads created
During RTEMS initialization, user initialization task created and started

```
rtems_task_create(TA1, …, &task_id);
rtems_scheduler_ident(SCHED_NAME(1), &scheduler_id);
rtems_task_set_scheduler(task_id, scheduler_id);
rtems_task_start(task_id, test_task, 1);
```



RTEMS

LEON3 Clustered Scheduling Example #2

- Architecture
  - N single core blocks
  - L1 cache per block
  - Cache coherence across cores
- Scheduler config
  - CPU0: IRQs and IO threads
  - CPU1: Critical thread set
  - CPU2-3: Worker threads
- Focus on schedulability and predictability

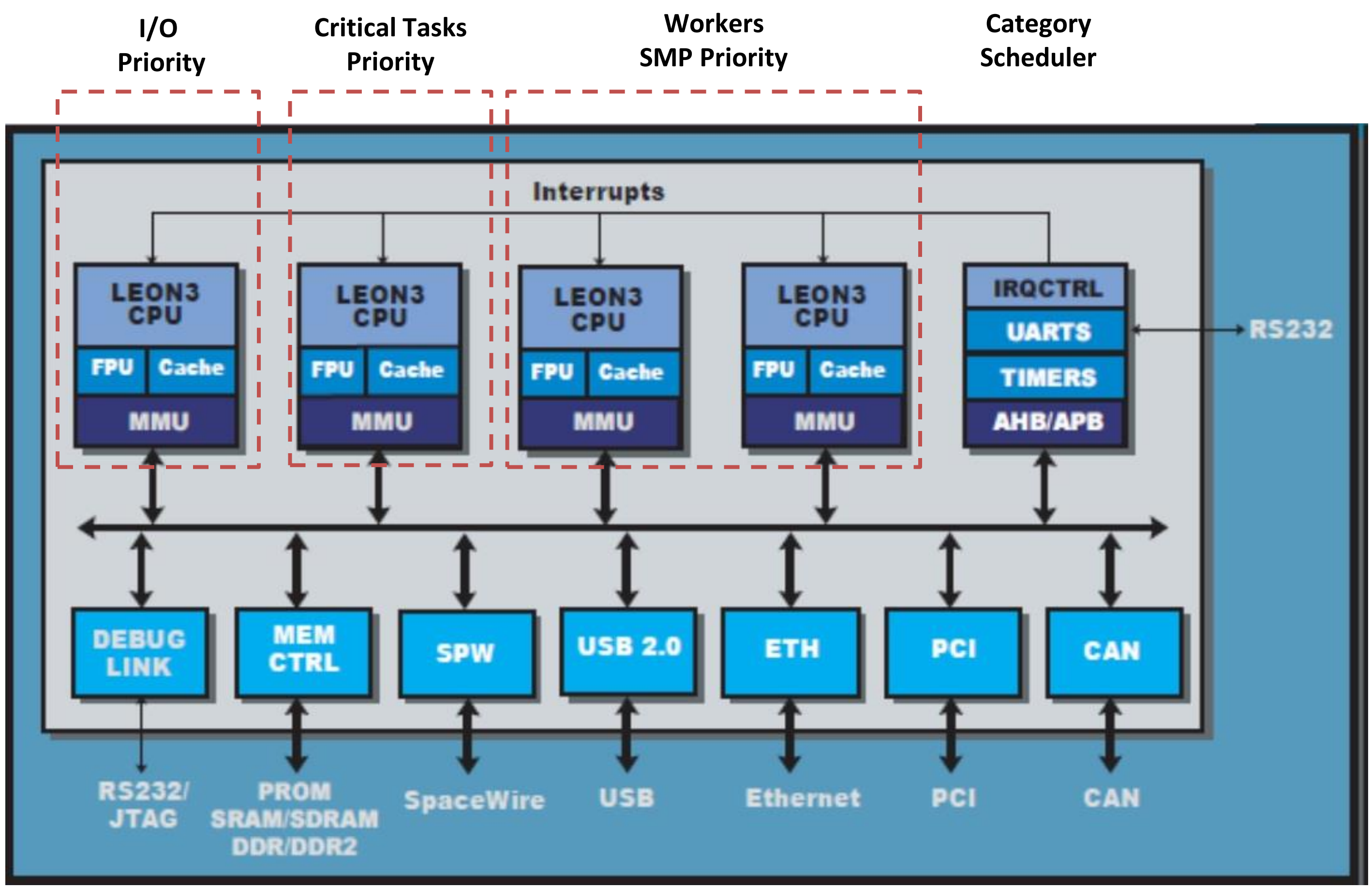


Figure: Aeroflex/Gaisler

AMD Bulldozer (FX) Example #1

- Architecture
  - N dual core blocks
  - L2 cache per block
  - Shared L3 cache
  - 1 FPU per block
- Scheduler config
  - One SMP Priority Affinity scheduler instance
  - Affinity to have FP threads on a single core within block
- Eliminates FPU contention
- Does not address cache effects and locality

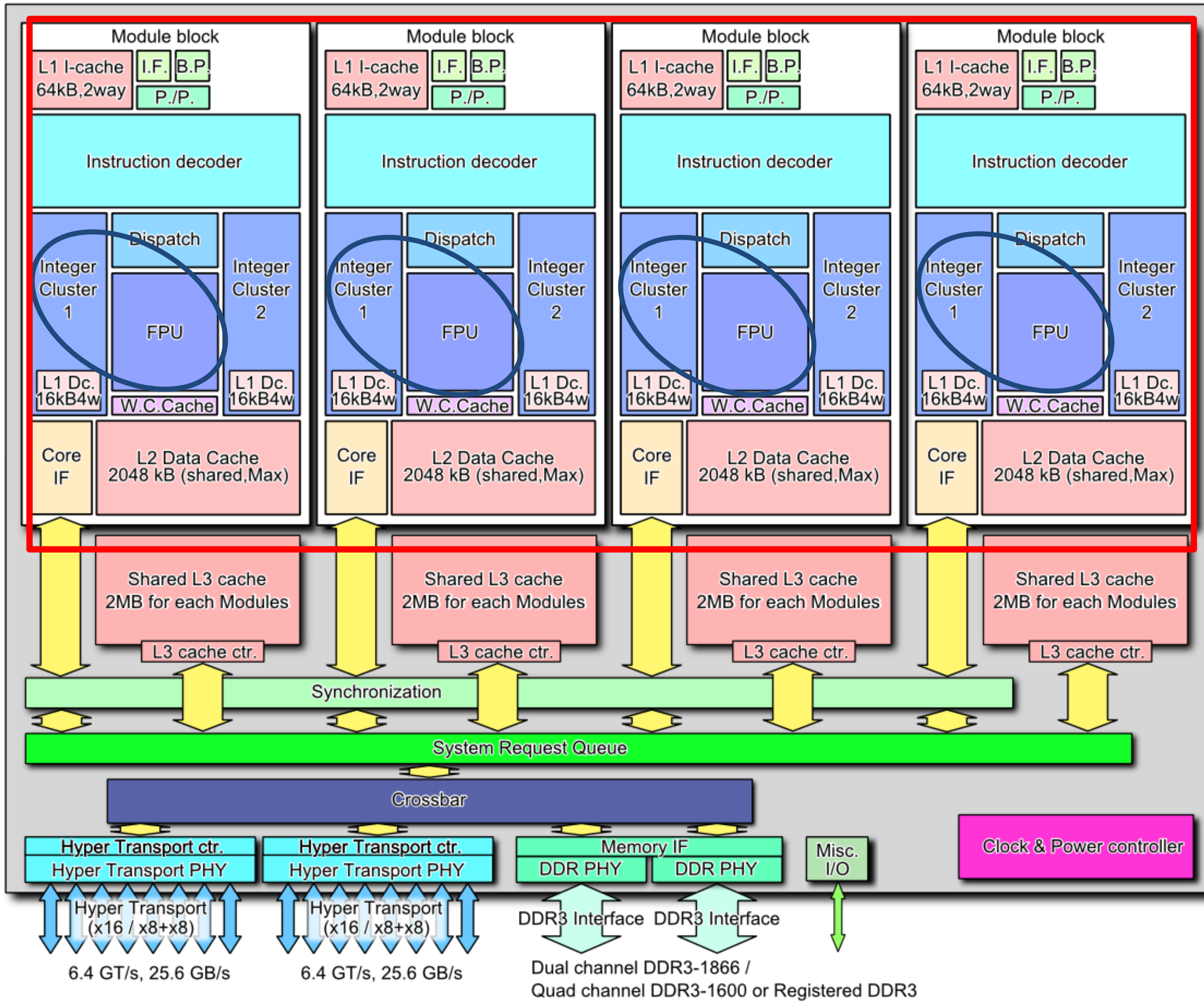


Figure: Wikipedia.org

AMD Bulldozer (FX) Example #2

- Architecture
  - N dual core blocks
  - L2 cache per block
  - Shared L3 cache
  - 1 FPU per block
- Scheduler config
  - Cluster of 4 scheduler instances with one SMP Priority Affinity scheduler instance per block
  - Affinity to have FP threads on a single core within block
- Avoids contention on cache and FPUs

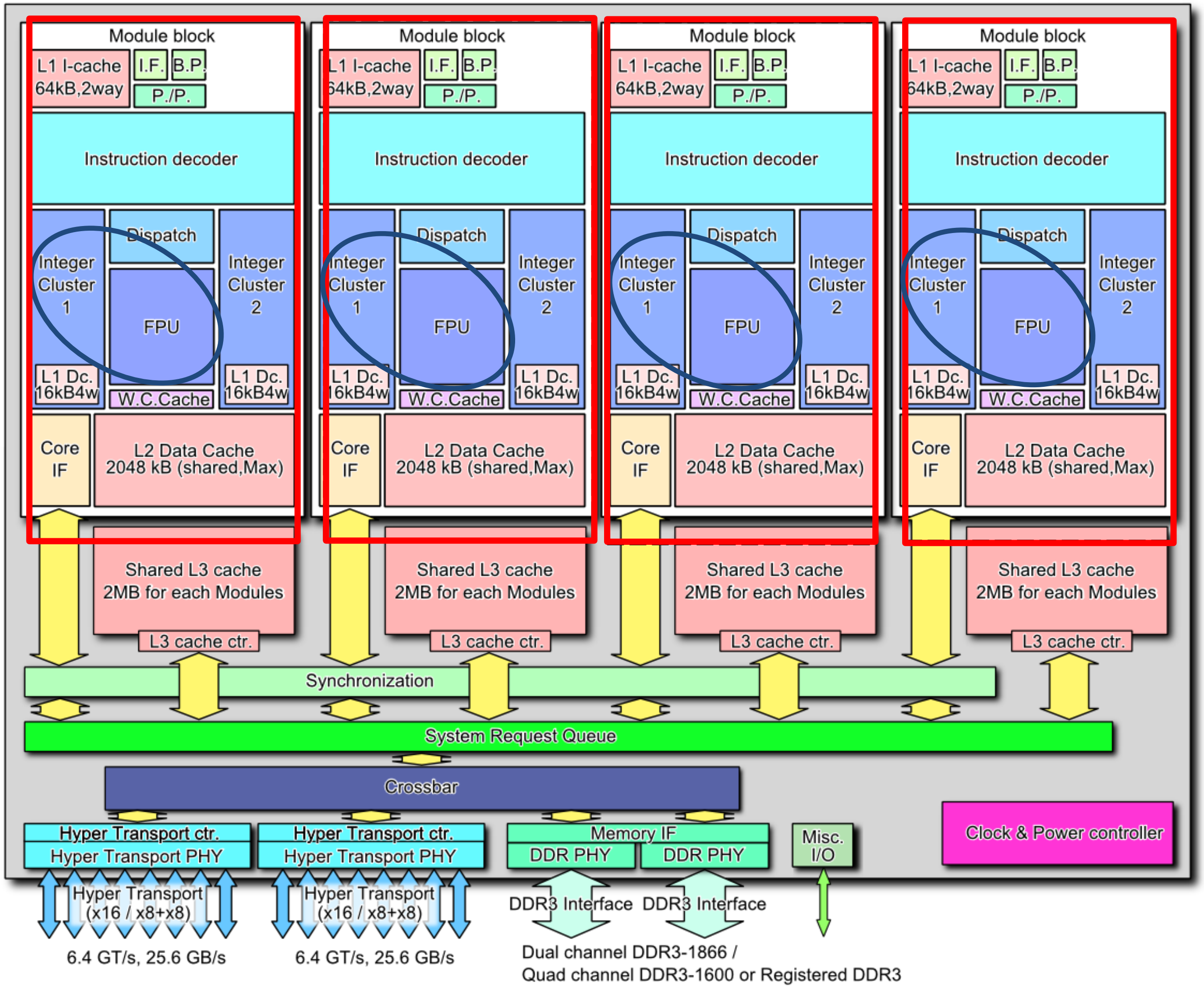


Figure: Wikipedia.org

RTEMS

Multiprocessor Resource Sharing Protocol (MRSP)

- Generalization of the Priority Ceiling Protocol
- Each MrsP semaphore uses a ceiling priority per scheduler instance
- Ceiling priorities specified with *rtems_semaphore_set_priority()*
- Task holding MrsP semaphore executes at the ceiling priority
- Tasks waiting for a MrsP semaphore will not relinquish the processor voluntarily
- If the owner of a MrsP semaphore gets preempted it can ask all tasks waiting for this semaphore to help out and temporarily borrow the right to execute on one of their assigned processors

- Reference:
  - Burns, A., and A. J. Wellings. "A Schedulability Compatible Multiprocessor Resource Sharing Protocol -MrsP." *Proceedings of the 25th Euromicro Conference on Real-Time Systems (ECRTS 2013)* (2013)

Conclusion

- RTEMS SMP is ready for production use
- SMP applications bring new challenges
- Features give users flexibility
- Designed to scale for many core systems
- Evolution will be driven by users

RTEMS

Contacts and Acknowledgements

Joel Sherrill, Ph.D.

OAR Corporation

Huntsville Alabama USA

Joel.Sherrill@oarcorp.com