



Designing an Industry Standard API to Manage Multicore System Resources

August 11, 2009

Presented by: Jim Holt, Chairman of MRAPI Working Group

▶ Jim Holt

- Leading the Processor Core Architecture and Modeling Team for Freescale's Networking and Multimedia group.
- MCA Board member representative on behalf of Freescale
- Chairman of MRAPI working group
- Major contributor on MCAPI specification



Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © Freescale Semiconductor, Inc. 2009.

- ▶ Overview of Multicore Association working groups
- ▶ Why a resource management API ?
- ▶ Existing approaches and gaps
- ▶ MRAPI: the Multicore Resource Management API
 - Philosophy
 - Alignment with Multicore Association Activities
 - Current Feature Set
- ▶ Some Resource Management Use Cases
- ▶ Conclusions

► Currently Active

- Multicore Resource Management API (MRAPI)
- Multicore Programming Practices (MPP)
- Multicore Virtualization Working Group (MVWG)

► Completed

- Multicore Communications API (MCAPI)
- Downloaded now by almost 500 people
- Working its way into commercial products

Why a Resource Management API ?

- ▶ Embedded multicore programming resembles other forms of parallel & distributed programming
- ▶ Programmers need implementation mechanisms to coordinate critical sections and resource sharing in their applications
 - Mutexes
 - Semaphores
 - Shared/Remote Memory
 - Metadata
- ▶ Many existing standards
 - Are too heavyweight (e.g., those designed for distributed systems, such as MPI or CORBA)
 - Or they rely on the presence of an SMP operating system (such as Pthreads)
 - Or they won't work without physical shared memory
- ▶ Needs of embedded systems are not encompassed in existing standards
 - Explicit support for allocating on-chip fast SRAM versus off-chip DDR
 - Selecting the mechanism for transferring data in systems with distributed memories



Existing Approaches and Gaps

The MRAPI working group has reviewed:

- POSIX mutexes and semaphores
- POSIX shared memory
- Proprietary solutions from MCA member companies
- IMEC's vision for resource managers
- The Linux approach to providing metadata via device trees
- The DaCS API
- The PAPI API



Gaps are outlined in working group internal docs

- Biggest gap is **dependence on SMP operating systems** for these features
- Goal: provide features in a multicore system with multiple OS, private memory, heterogeneous cores, virtualization, or even bare metal apps

We also need to distinguish between **resource management primitives** and **full-featured resource managers**

- Want to define primitives which allow others to build portable apps and services
- A full-featured, system-level resource manager could be built on top of the MCA APIs, but a full fledged **standard resource manager** would be a big undertaking!

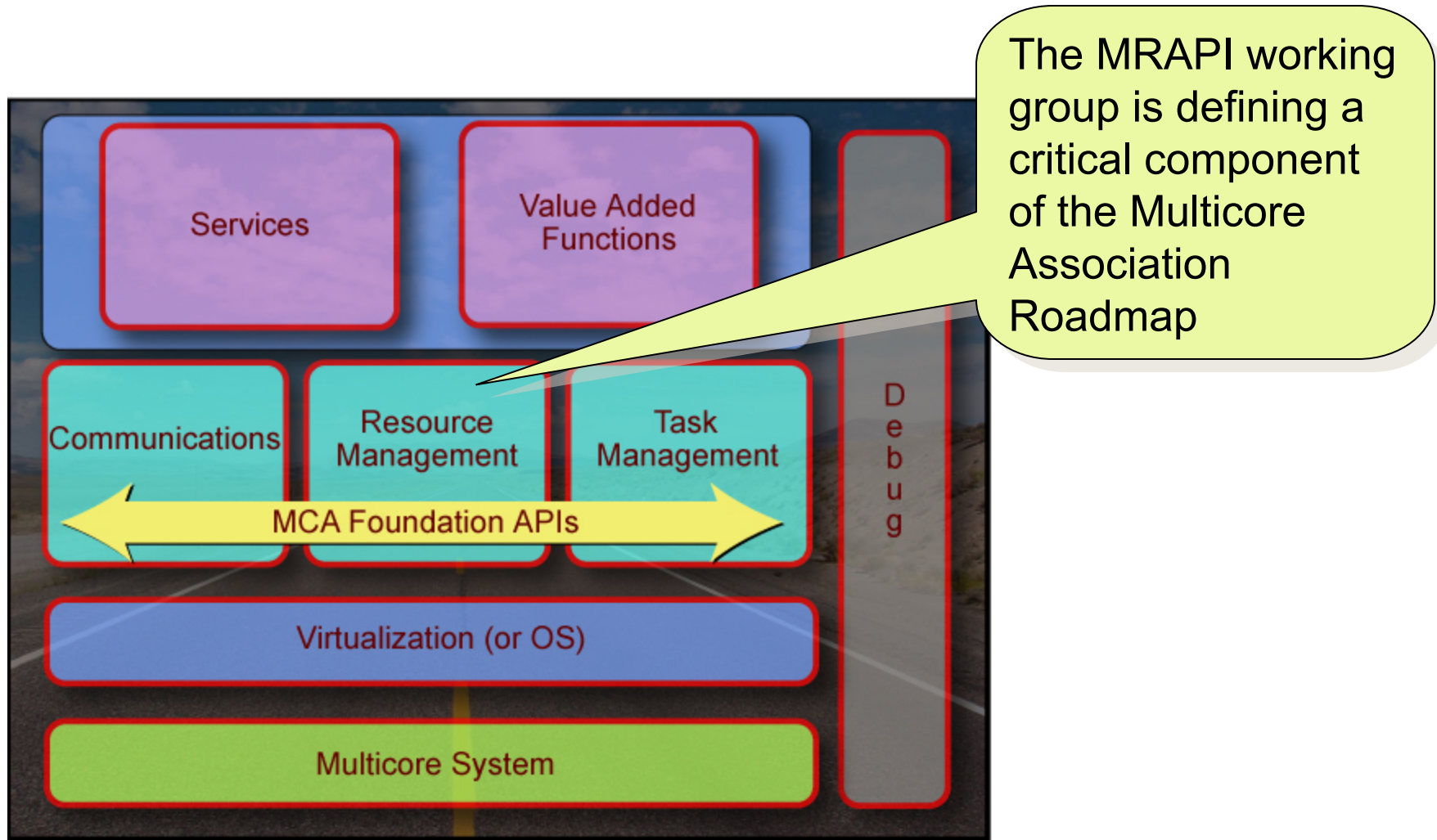
MRAPI: the Multicore Resource Management API

Philosophy



- ▶ Small application layer API, suitable for 'closely distributed' cores
- ▶ Easy to learn and use, incorporates essential feature set
- ▶ Supports lightweight and performant implementations
- ▶ Does not prevent/preclude use of complementary approaches
- ▶ Allows silicon providers to optimize their hardware and take advantage of hardware features
- ▶ Allows implementers to differentiate their offerings
- ▶ Can run on top of an OS, hypervisor, or bare metal
- ▶ Can co-exist with hardware acceleration
- ▶ Supports hardware implementations of the API
- ▶ Does not require homogeneous cores, operating system, or memory architecture on chip

Alignment with Multicore Association Activities



Alignment with Multicore Association Activities

The MRAPI working group is following in the footsteps of the MCAPI working group

- Adopted similar philosophies
- Adopted the same style for API, datatypes, etc.
- Ensuring that MRAPI functionality is orthogonal to MCAPI functionality while making sure they are interoperable (for example discussions around shared memory for MRAPI and zero copy messaging for MCAPI)

The MRAPI WG is monitoring the work of MPP group and MVWG, also sharing early drafts of the MRAPI requirements and specification with those working groups.

Synchronization Primitives

Mutexes – binary primitives

- *could be provided by shared memory, a distributed runtime, or other means*

Semaphores – counting primitives

- *support creation of reader/writer locks*
- *same implementation goals as mutexes*

Memory Primitives

Shared Memory – allocate and manage application shared memory regions where there is **physical shared memory to support**

- *Support for requesting memory with specific attributes*
- *Support for allocation based on a set of sharing entities*

Remote Memory – allow application to manage buffers that are **shared but not implemented on top of physical shared memory**

- *Allows distribution via chip-specific methods such as DMA transfers, SRIO, software cache*
- *Supports random access, scatter/gather, and has hooks for software managed coherency*

Metadata Primitives

Limited to **hardware information** rather than being a facility for an application to create and manage its own metadata

- *this additional functionality could be a layered service*

Application Level Sharing vs. Resource Locking

- ▶ MRAPI API does not preclude an application-level implementation - it cannot enforce 'ownership' of a resource in the same way an operating system can
- ▶ Therefore, MRAPI will not directly provide an API to 'lock' a resource
- ▶ The semantics we wish to support are cooperative resource sharing
 - mutexes and semaphores provide sufficient functionality to provide this
- ▶ This higher level capability can be provided as a service using a combination of MRAPI and Hypervisor features, for example

MRAPI: the Multicore Resource Management API

Metadata Primitives (features under investigation)

- ▶ Extensible support for queries regarding static hardware resources
 - Determine the number of CPUs, caches, etc.
 - Determine the number and types of memory regions available
 - Hardware accelerators
 - Etc.
- ▶ Support for querying attributes of these resources
 - **Static attributes** such as size, latency, etc.
 - **Dynamic attributes** such as utilization, etc., with counter rollover callbacks
- ▶ Support for system-level event notification such as power savings states, device failures, hypervisor repartitioning
 - Requery of Metadata could return a potentially different answers following such events
- ▶ No direct support for application level metadata
 - This functionality could be layered on top of the Multicore Association APIs

(1) use case: remote allocate and push

Description: *nodeB** wants to access local (*non-physically shared*) memory on *nodeA**, fill it with data via DMA, and then notify *nodeA** that the buffer is ready for reading

(2) use case: random access remote memory

Description: *nodeA** and *nodeB** want to share a buffer of non-shared memory and access it randomly

- ▶ Must allow underlying implementation to be a 'software cache', DMA, RapidIO, etc.
- ▶ Currently, the application can indicate which underlying method it desires via attributes assigned to the remote buffer.

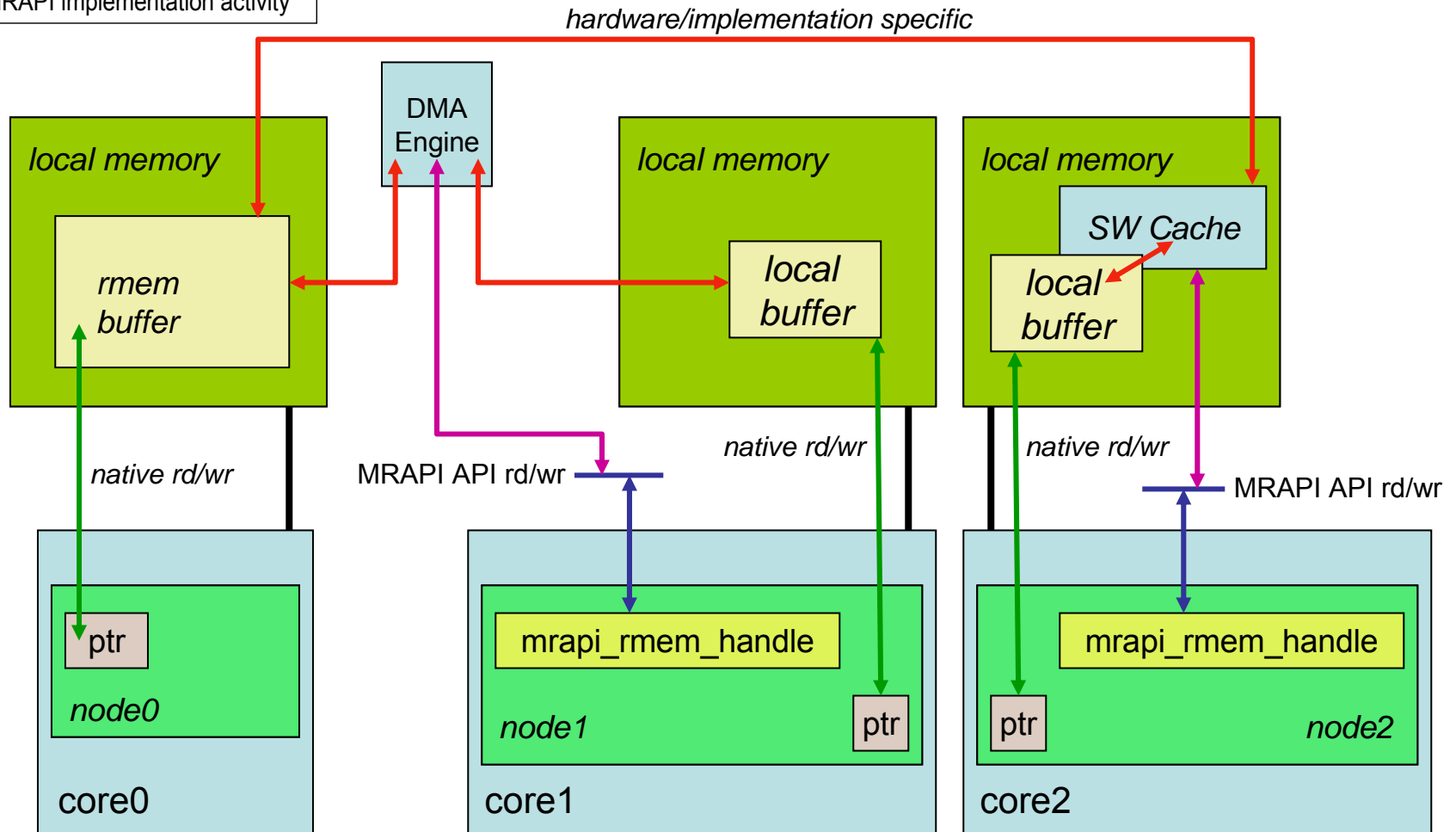
Other use cases are under discussion within the working group

* In MCAPI/MRAPI terminology a 'node' may be a CPU, a process, or a thread

LEGEND

- program data accesses
- data movement
- MRAPI API calls
- MRAPI implementation activity

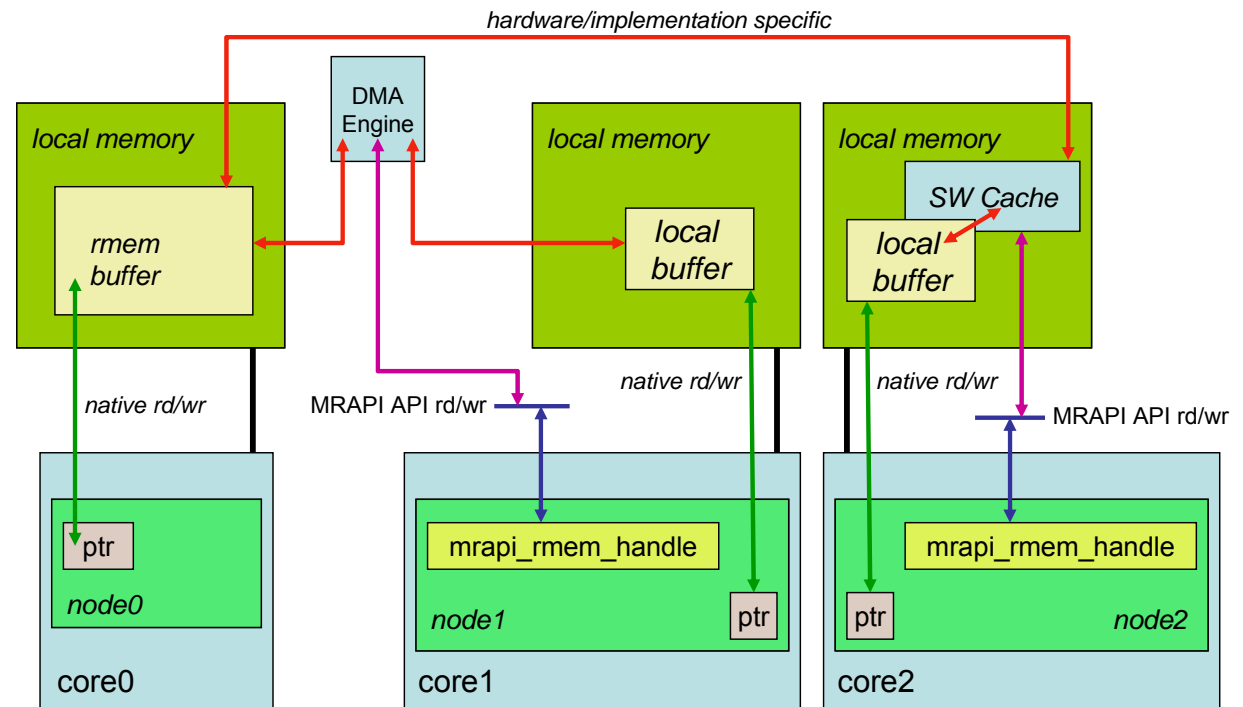
Remote Memory Concepts



LEGEND

- program data accesses
- data movement
- MRAPI API calls
- MRAPI implementation activity

Remote Memory Concepts



Local pointer based read/write always OK, but coherency issues are managed by application using flush and synch calls

Remote access (read or write) always results in a copy & must use API

Implementation can define multiple ways to access, but must provide MRAPI_ACCESTYPE_DEFAULT, which is guaranteed to work

Case A: Nominal Power-Up state

1. MRAPI is queried as to available resources. Qualifiers could be applied to limit the required information to CPUs, Memory subsystem, Hardware accelerators, etc.
2. MRAPI is queried as to the attributes of resources needed at this point in time
3. Example application scenarios:
 - a) Packet Processing - based on the attributes returned for network HW accelerators, and CPUs the application may choose to perform packet processing utilizing hardware acceleration and optimized multi-threaded software on all of the cores
 - b) Video Processing (H.264) – based on attributes returned for the system (nature of heterogeneous cores in the system), partition the task of decoding the video streams between the mix of heterogeneous cores to minimize latency.

Case B: Reduced Power state

1. On a state transition from “**Normal power**” state to a “**Low Power**” state, MRAPI informs the application via a callback function that minimally notifies the app that some change has occurred.
2. MRAPI is re-queried as in step (1) in Case A
3. Based on less resources being available (and/or a low power state with reduced MIPS), the application could employ a different execution strategy (repeat Case A step 3 with different outcome).

- ▶ The MRAPI Working group is defining an API for cooperative application-level resource sharing
 - This API is targeted to support both homogeneous and heterogeneous multicore systems in the embedded domain, where existing standards do not scale well or do not encompass all of the requirements of the domain
- ▶ It is envisioned that higher levels of Resource Management (such as a dynamic resource manager that provides guarantees regarding security and quality of service to client applications) could be built using a combination of features from the Multicore Association roadmap
- ▶ The draft specification is nearing completion and will move to external review in the near future.

Multicore Association Membership

▶ Executive Board Membership

- Freescale Semiconductor, Huawei Technologies, Intel, Mentor Graphics, Nokia Siemens Networks, Plurality, PolyCore Software, Samsung Electronics, Texas Instruments, Tiler, Wind River

▶ Working Group Membership

- CAPS entreprise, Codeplay, CriticalBlue, Enea, eSOL, IMEC, LG Electronics, LSI, MIPS Technologies, National Instruments, nCore Design, Open Kernel Labs, Virtutech, VMware

▶ University Membership

- Carnegie Mellon University, Delft University of Technology, University of Utah

**Become a Member of The Multicore Association Today -
Don't Let the Multicore Revolution Leave You Behind.**

www.multicore-association.org

Questions and Answers