Leveraging Multiprocessor Embedded Systems using industry standards

Peng Sun, Sunita Chandrasekaran and Barbara Chapman

Department of Computer Science, University of Houston

psun5@cs.uh.edu

Abstract

Multicore embedded systems are rapidly emerging. Hardware designers are packing more and more features into their design. Introducing heterogeneity in these systems, i.e. adding cores of varying types does provide opportunities to solve problems in different aspects. However, this presents several challenges to embedded system programmers since software is still not mature enough to efficiently exploit the capabilities of the emerging hardware rich with cores of varying types.

Programmers still rely on understanding and using low-level hardware-specific API. This approach is not only very time-consuming but also tedious and error-prone. Moreover, the solutions developed are very closely tied to a particular hardware raising significant concerns with software portability.

What we need is an industry standard that will enable better programming practices for both current and future embedded systems. To that end, in our project, we have explored the possibility of using existing standards such as OpenMP that provides portable high-level programming constructs along with another industry-driven standard for multicore systems, MCA.
For our work, we have considered the GNU compiler since it is the compiler that mostly used in the embedded system domain facilitating open source development. We target a platform consisting of twelve PowerPC e6500 64-bit dual-threaded cores. We create a portable software solution by studying the GNU OpenMP runtime library and extending it to incorporate MCA libraries. The solution abstracts the low-level details of the target platform and the results show that the additional MCA layer does not incur any overhead. The results are competitive when compared with a proprietary toolchain.

OpenMP Introduction

OpenMP is a well-known portable and scalable programming model that facilitates programmers with simple, but versatile interface for developing parallel applications.
By inserting pragmas into an original serial program, the model makes it easy to leverage underlying hardware rapidly and effortlessly. OpenMP's primary model of parallel execution is fork - join.

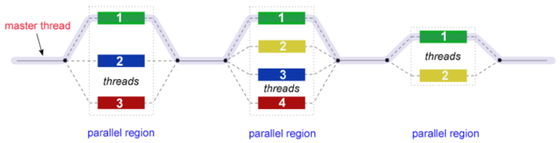

Figure1. OpenMP Fork-Join Execution Model

Target Platforms

T4240RDB Board

For our work, we have chosen T4240RDB platform from Freescale's QorIQ family.

The T4240RDB platform features twenty-four virtual threads from twelve PowerPC e6500 cores, running at 1.8GHz and providing the rich I/O capabilities.
The Freescale T4 processor family is commonly used in networking productions like routers, switches, gateways to fully utilize its combined control, data path support and application layer processing and also for general purpose embedded computing systems.

We configure the T4240RDB board to load the embedded kernel image from TFTP server while u-boot, and deploy an NFS root file system to be mounted.

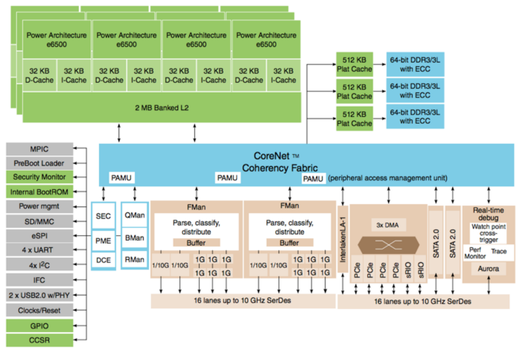

Figure1. Freescale T4240RDB Diagram

MCA Resource Management API

MRAPI seeks to handle resource management challenges of the most critical hardware resources on real products, including shared memory, remote memory, synchronization primitives, and metadata, for both SMP and AMP architectures. MRAPI can support virtually any number of cores, even each with a different instruction set, same or different OSes.

The MRAPI API contains the following fundamental concepts:
1) Domain and Nodes
2) Memory Primitives
3) Synchronization Primitives
4) System Resource Metadata

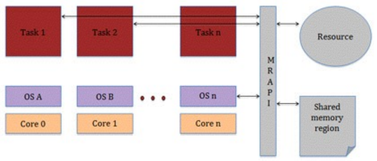

Figure2. Multicore Association Resource Management API

MRAPI Extension

1) MRAPI Node Management Extension:
We use MRAPI's node initialization process to create threads associated with MRAPI node IDs. MRAPI node initialization process creates new nodes associated with node IDs and registers the related node information in the global MRAPI database that is shared by all the nodes in one domain

2) MRAPI Memory Management Extension:
MRAPI shared memory constructs by default maps the memory allocation onto the system level shared memory, which is an Inter Process Communication (IPC) methodology. However, this is not a suitable methodology for OpenMP and the other thread-level parallel computation, even for embedded systems. To tackle this issue, we extend the MRAPI implementation to offer end-users more choices with memory allocation

Enhancing libGOMP with MRAPI

1) Node Management:
We map the MCA-libGOMP thread allocation to the MRAPI node management constructs, with thread creation, and exit handled internally by MRAPI. During runtime, when the OpenMP runtime library needs to fork a team of worker threads, the MRAPI node initialization is called by runtime.

2) Memory Mapping:
In the OpenMP program runtime, a set of global data structures needs to be maintained. For example, each team of nodes would need to keep a block of work share, to be assigned to each node later for computation. In this project, we extend the MRAPI shared memory constructs to be able to allocate thread-level shared data

3) Synchronization:
The synchronization primitives of MCA-libGOMP has been mapped to MRAPI Mutexes, preventing critical data races and managing accesses to the shared data. Specifically we use the MRAPI mutex create function to create the Mutex object as the initialization step, and map the Mutex lock and unlock functions to MRAPI Mutex routines as well.

4) Metadata Information:
MRAPI metadata constructs have also been utilized within the OpenMP runtime library. We mainly used the MRAPI metadata trees to retrieve the available number of processors online for node/thread management, to better serve the MRAPI nodes and threads allocation and management accordingly.

TABLE I: Relative overhead of MCA-libGOMP versus GNU OpenMP runtime

Directive	4	8	12	16	20	24
Parallel	0.98	1.04	0.73	0.98	0.98	1.03
For	1.00	1.10	1.10	1.01	1.31	1.49
Parallel_for	0.99	1.36	1.05	1.03	0.81	0.95
Barrier	0.90	0.93	1.13	0.90	1.48	1.32
Single	0.41	2.39	1.09	0.97	0.99	1.03
Critical	0.99	1.34	0.99	1.19	1.11	0.45
Reduction	0.98	0.97	1.00	0.94	1.07	1.01

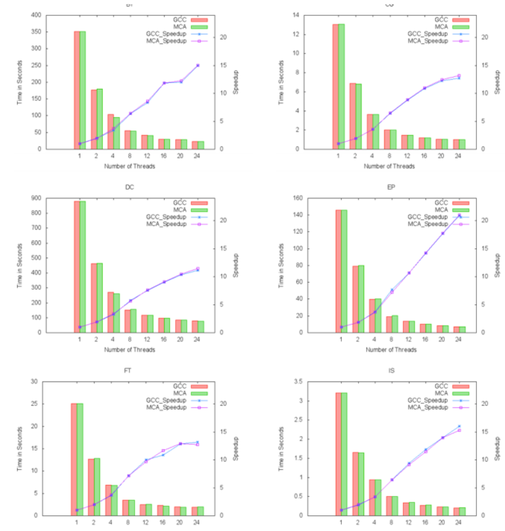

Figure3. Evaluating MCA-libGOMP runtime library using NAS benchmarks

Conclusions

Sophisticated portable toolchain is a dire necessity for embedded platforms. The gap between emerging hardware and lack of appropriate software is widening. To address this primary concern, we have explored the usage of a high-level programming model, OpenMP that uses MCA, another industry-based standard that promotes multicore technology.

In this poster, we present a commonly used embedded industry preferred OpenMP runtime with MRAPI to create a standardized application program interface for managing resources and providing source-code compatibility allowing applications to be ported to platforms.

Acknowledgments

This research has been supported by grants from Freescale Semiconductor Inc. in association with Semiconductor Research Corporation (SRC).

HPC TOOLS

UNIVERSITY of HOUSTON
DEPARTMENT OF COMPUTER SCIENCE