OpenMP for Heterogeneous Multicore Embedded Systems using MCA API standard interface

Sunita Chandrasekaran (sunita@cs.uh.edu)  
Peng Sun, Suyang zhu, Barbara Chapman  
HPCTools Group, University of Houston, USA

in collaboration with freescale semiconductor (FSL)  
and Semiconductor Research Corporation (SRC)

OpenMP Booth @ SC14
Real-world applications using embedded systems

FPGAS used to receive a number of acoustic pings from an image

DSP, a device to interact with a user through the embedded processor within

A self-piloted car powered by the NVIDIA Jetson TK1
Embedded programmers’ requirements

• Write once, reuse anywhere
  o Avoid rewriting from scratch
  o Provide incremental migration path essential for application codes
  o Exploit multiple levels of parallelism
• ...with familiar and/or commodity programming models
  o None are perfect, but industry adoption is critical
OpenMP – widely adopted standard (www.openmp.org)

- Industry standard for shared memory parallel programming
  - Supported by large # of vendors (TI, AMD, IBM, Intel, Cray, NVIDIA, HP.....)
  - OpenMP 4.0 – provides support for compute devices such as Xeon Phis, GPUs and others

- High level directive-based solution
  - Compiler directives, library routines and envt. variables
  - Programming Model - Fork-Join parallelism
  - Threads communicate by sharing variables
  - Synchronizations to control race conditions
  - Structured programming to reduce likelihood of bugs

Programming made easy!

```c
void main()
{
    double Res[1000];
    #pragma omp parallel for
    for(int i=0;i<1000;i++) {
        do_huge_comp(Res[i]);
    }
}
```
OpenMP Solution Stack

- OpenMP Application
- Directives, Compiler
- OpenMP library
- Environment variables
- Runtime library
- OS/system support for shared memory
- Core 1
- Core 2
- Core n
OpenMP for Embedded Systems

- Programming embedded systems a challenge
- Need high-level standards such as OpenMP
  - Runtime relies on lower level components
    - OS, threading/hardware libraries, memory allocation, synchronization e.g. Linux, Pthreads
      - But embedded systems typically lack some of these features
  - OpenMP has shared-memory cache-coherent memory model
    - But embedded platforms feature distributed, non-uniform memory, with no cache-coherency
    - Vocabulary for heterogeneity is required in the embedded space

OpenMP 4.0 is there!!
Multicore Association Industry standard API (MCA)

MCA Foundation APIs

**Communications (MCAPI)**
- Lightweight messaging

**Resource Management (MRAPI)**
- Basic synchronization
- Shared/Distributed Memory
- System Metadata

**Task Management (MTAPI)**
- Task lifecycle
- Task placement
- Task priority

**SW/HW Interface for Multicore/Manycore (SHIM)**
- XML HW description from SW perspective
Tasks in Heterogeneous Systems

- Tasks execute a job, implemented by an action function, on a local or remote node
- Task can be started individually or via queues (to influence the scheduling behavior)
- Arguments are copied to the remote node
- Results are copied back to the node that started the task
Tasks, Jobs, Actions
Runtime Design and Optimizations

- Optimized Thread Creation, Waiting and Awakening
  - All threads in a team cannot be identical
  - Uses MRAPI meta data primitives
  - Avoid over-subscription
  - Distributed spin-waiting

- Synchronization Construct

- Memory Model
  - Uses MRAPI shared/remote memory primitives
Freescale’s Communication processor with data path

QorIQ P4080 processor
- 4-8 Power architecture e500mc cores
- Accelerators
  - Encryption (SEC)
  - Pattern Matching Engine (PME)
- Target applications:
  - Aerospace and Defense
  - Ethernet Switch, Router
  - Pre-crash detection
  - Forward Collision Warning

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=P4080&tid=redP4040
Portable OpenMP Implementation

- Translated OpenMP for MPSoCs
- Used Multicore Association (MCA) APIs as target for our OpenMP translation
- Developed MCA-based runtime:
  - Portable across MPSoCs
  - Light-weight
  - Supports non-cache-coherent systems
  - Performance comparable to customized vendor-specific implementations
Compilation Process

- **OpenUH as our frontend**
  - open source, optimizing compiler suite for C, C++, and Fortran, based on [Open64](#).
  - Translates C+OpenMP source into C with OpenMP runtime function calls

- **PowerPC-GCC as our backend to generate the object file and libraries**

- **Final executable file is generated by linking the object file, our OpenMP runtime library and the MCA runtime library.**

Dual-core power processor from Freescale Semiconductor
Enhanced OpenMP runtime Vs proprietary runtime

**DIJKSTRA**

**JACOBI**

**FFT**

**LU Decomposition**
OpenMP + MCA for heterogeneous systems

QorlQ P4080 processor
- 4-8 Power architecture e500mc cores
- Accelerators
  - Encryption (SEC)
  - Pattern Matching Engine (PME)
- Target applications:
  - Aerospace and Defense
  - Ethernet Switch, Router
  - Pre-crash detection
  - Forward Collision Warning

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=P4080&tid=redP4040
We explored and analyzed DPAA and its component PME in depth for our work. We intend to abstract the low level details of PME by exploiting the capabilities of MCAPI. This API has the vocabulary to establish communication on a bare metal implementation (no OS at all), that was one of the motivational aspects to choose PME as our candidate accelerator that is also a bare metal hardware accelerator. This hardware accelerator does not offer shared memory support with the host processors, the power processor cores (e500mc cores) for P4080. As a result, the data movement between the host and the accelerators need to be handled via a DMA channel explicitly. Thus we will be deploying a standard communication API such as MCAPI to handle the data transportation and messaging.

Implement MCAPI on PME

The main challenge before deploying MCAPI for PME was to understand the libraries and the technology of the platform that are at a relatively low level. We spent quite an amount of time to gather all the hardware and software functionalities of PME and its usage. There are a few MCAPI implementations available that we explored to gain a deeper understanding about the APIs. One of them is the open source implementation called OpenMCAPI created by Mentor Graphics. There is yet another implementation called the XMCAPI from the University of Tsukuba, which extends the prototype implementation of MCAPI available on the website and can be used for inter-chip communication via Ethernet. But unfortunately, these implementations only target the shared memory architectures.

For our work, we began by studying PME and its available resources. We realized that it supports both pre-built binary files to run the PME from the command line and a library based API in high-level C. We used the 'pmrec' tool to create an MCA wrapper for communication between the power processor and PME.
MCAPI Connectionless technique

MCAPI Connectionless Message on FSL P4080DS Board

MCAPI Node 1 (e500mc)
endpoint<1,1>

PMCI message send to <2,1>

PMCI message received from <2,1>

MCAPI Node 2 (DPAA PME)
endpoint<2,1>

Messages - Connectionless
Flexible, less configuration
Portable across different message send/receive endpoint
Message prioritized
MxAPI for accelerators – lessons learned

• PME does not share memory with the main processor
• Data movement via a DMA channel
• Required thorough knowledge of low-level API very tightly tied to the hardware
  o Time consuming
  o Requires constant support from the vendor to understand the low-level API
• Created an MCA wrapper to abstract all low-level details
  o However the wrapper can be used for devices that relies on that same low-level API.
MTAPI Design and Implementation - Current Status

- On-going work: Implementing MTAPI features
- Writing small test cases to validate the MTAPI implementation
- Need to evaluate the MTAPI implementation on a heterogeneous multicore platform
- Preliminary results demonstrated overhead while communicating with a remote node through MCAPI
OpenMP and MCA API

Application Layer
- OpenMP Applications
  - Directives
  - Runtime Library Routines
  - Environment Variables

OpenMP Programming Layer
- OpenMP Runtime Library

MCA APIs Layer
- MRAPI
- MCAPI
- MTAPI

System Layer
- No OS??

Hardware Layer
- Multicore Embedded Systems

No OS??
Summary

- Extend OpenMP runtime library with MxAPI as the translation layer to target heterogeneous multicore SoCs
  - MTAPI prototype implementation – on-going @ UH
    - SIEMENS : http://www.techdesignforums.com/blog/2014/10/31/siemens-produces-open-source-code-multicore-acceleration/
  - Targeted a specialized accelerators
Accelerators are more than just GPUs

Future is Heterogeneous!

Accelerators are more than just GPUs.
MCAPI – Communication API

Connection-oriented Channels

MCAPI Domain 1

MCAPI Node 1
- endpoint <1,1,1>
  - attributes
  - port 1
- endpoint <1,1,2>
  - attributes
  - port 2

MCAPI Domain 2

MCAPI Node 2
- endpoint <1,2,1>
  - attributes
  - port 1

Connectionless Message

MCAPI Domain 1

MCAPI Node 1
- endpoint <1,1,1>
  - attributes
  - port 1
- endpoint <1,1,2>
  - attributes
  - port 2

MCAPI Domain 2

MCAPI Node 2
- endpoint <1,2,1>
  - attributes
  - port 1

Messages: Connectionless
- More flexible, less configuration
- Blocking and non-blocking
- Prioritized messages

Connectionless Has More Flexibility

Packets and Scalars: Connected
- More efficient, more configuration
- Blocking and non-blocking packets
- Blocking scalars

Connected Is More Efficient
MRAPI Memory Concept

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

**hardware/implementation specific**

```
local memory

rmem buffer

native rd/wr

ptr

node0

core0

DMA Engine

MRAPI API rd/wr

local memory

local buffer

native rd/wr

ptr

mrafi_rmem_handle

node1

core1

local memory

local buffer

native rd/wr

node2

core2

SW Cache

MRAPI API rd/wr

mrafi_rmem_handle

ptr
```