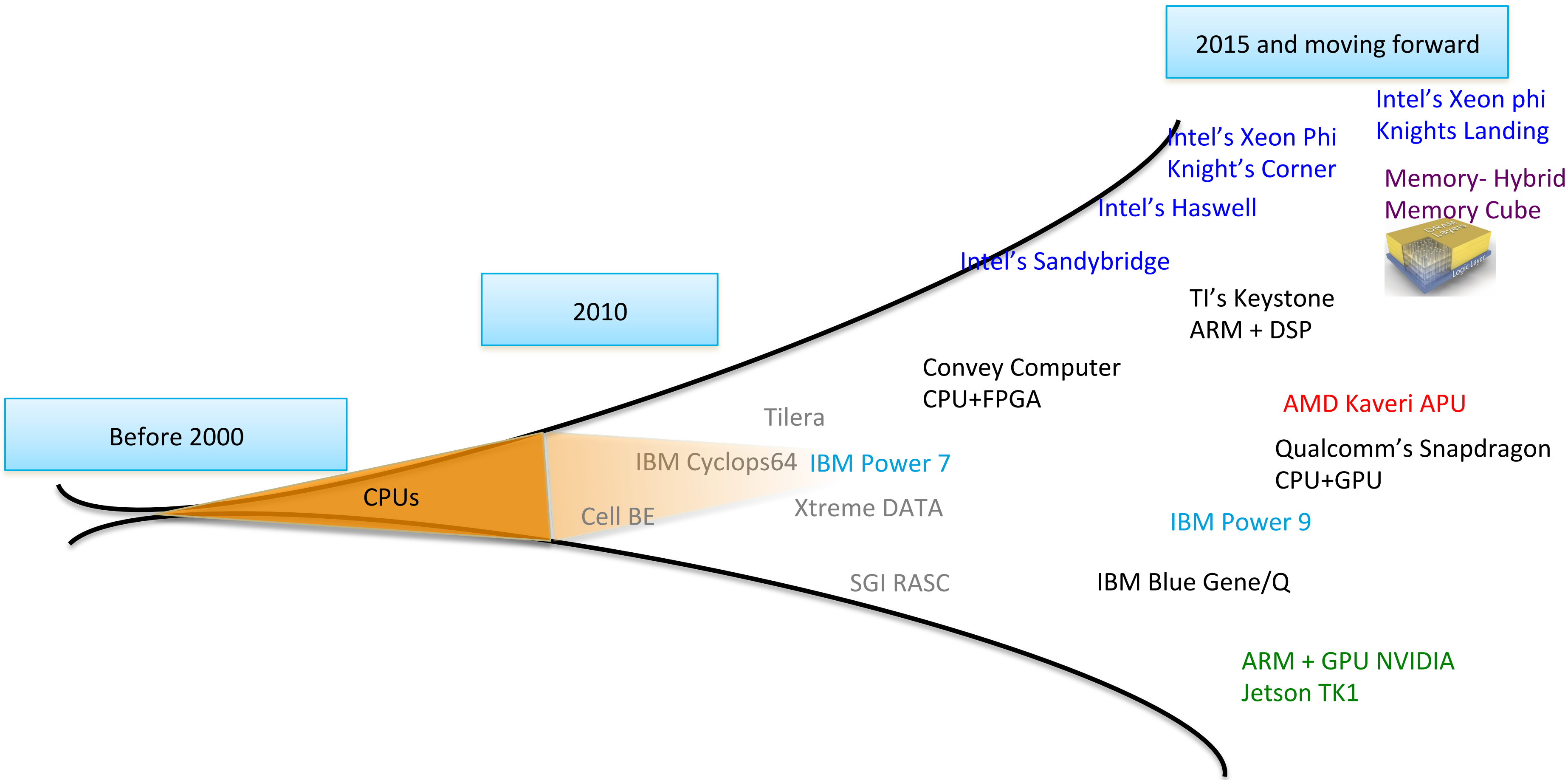
OpenMP Booth @ SC15
#2036

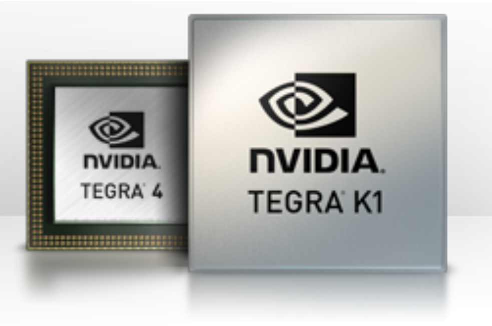
OpenMP for Embedded Systems - A Task-based Programming Model for Multicore Embedded Systems using Industry Standards, OpenMP and MCA

Sunita Chandrasekaran
University of Delaware
Contact: schandra@udel.edu

Hardware trend



Today's Embedded Systems

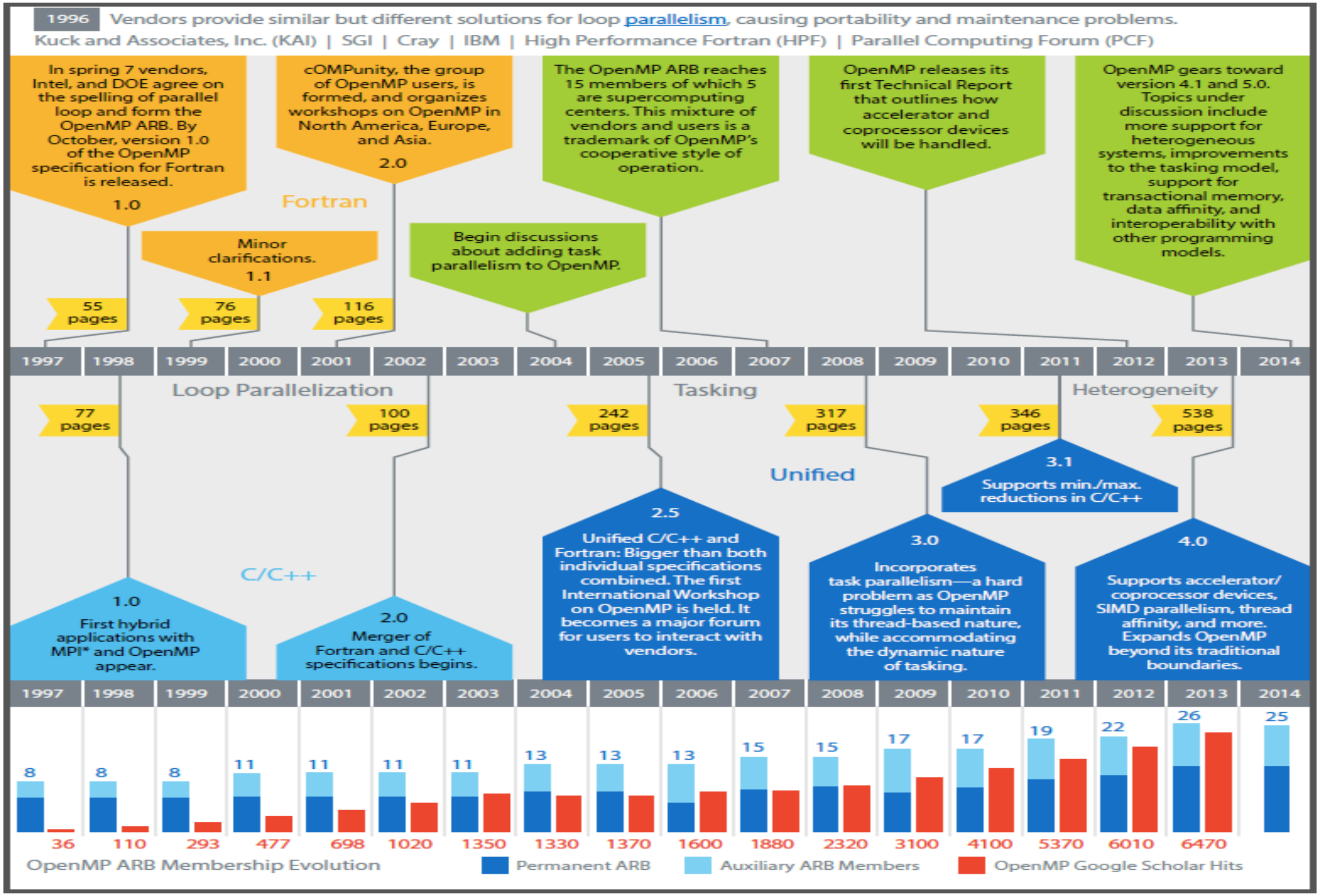


A Self-Piloted Car powered on NVidia Tegra TK1 Chip (ARM + GPU)

Programming Multicore Embedded Systems – A Real Challenge

- Heterogeneous systems present complexity at both silicon and system level.
- Software tool-chain most of the times are proprietary
  - Too tied to hardware
  - Programmers spend too much time on dealing with low-level details Proprietary
  - Write once, never reuse
  - Portability major concern
- High time-to-market (TTM) solutions
- We need industry standards that can offer portable software solutions and target more than one platform

OpenMP Timeline



At the OpenMP BoF Session during SC14, James Cownie from Intel showed an OpenMP Timeline

Briefly, on OpenMP Implementations

- Directives implemented via code modification and insertion of runtime library calls
  - Typical approach is outlining of code in parallel region
  - Or generation of micro tasks
- Runtime library responsible for managing threads
  - Scheduling loops
  - Scheduling tasks
  - Implementing synchronization
  - Collector API provides interface to give external tools state information
- Implementation effort is reasonable

OpenMP Code	Translation

int main(void)
{
int a,b,c;
#pragma omp parallel \ private(c)
do_sth(a,b,c);
return 0;
}
	_INT32 main()
	{
	int a,b,c;
	/* microtask */
	void __ompregion_main1()
	{
	_INT32 __mplocal_c;
	/*shared variables are kept intact, substitute accesses to private variable*/
	do_sth(a, b, __mplocal_c);
	}
	…
	/*OpenMP runtime calls */
	__ompc_fork(&__ompregion_main1);
	…
	}

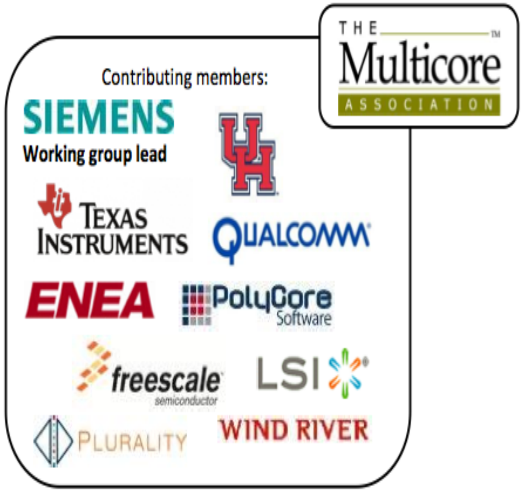
Each compiler has custom run-time support. Quality of the runtime system has major impact on performance.

How suitable are the state-of-the-art models for heterogeneous embedded systems?

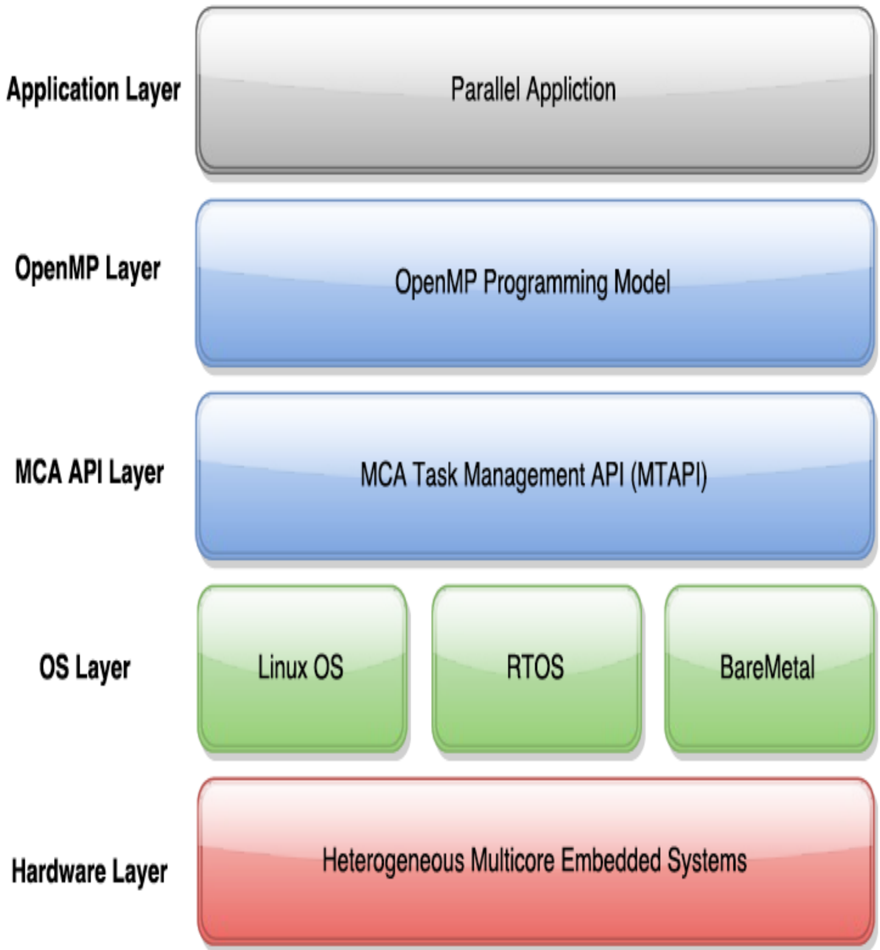
- Most of the models are heavy-weight, such as OpenMP, MPI for embedded systems
- Require particular support from operating systems and compilers
  - Embedded platforms are sometimes systems are even bare-metal
- State-of-the-art solutions showcase solutions for particular platforms
  - Portability is a major concern
- Complexity in programming and debugging(e. g. TBB)

Multicore Association (MCA) Industry Standards

- To overcome the programming challenges of heterogeneous multicore embedded systems and enable multicore product development, Multicore Association (MCA) http://www.multicore-association.org/index.php, an industry association defines and promotes open specification for:
  - Managing resources (cores/memory) using Multicore Resource Management API (MRAPI),
  - Communicating across cores/nodes using Multicore Communication API (MCAPI) and
  - Leverages task parallelism for symmetric and asymmetric multicore processors using Multicore Task Management API (MTAPI).



OpenMP + MCA:
A Solution to Abstraction, Portability,



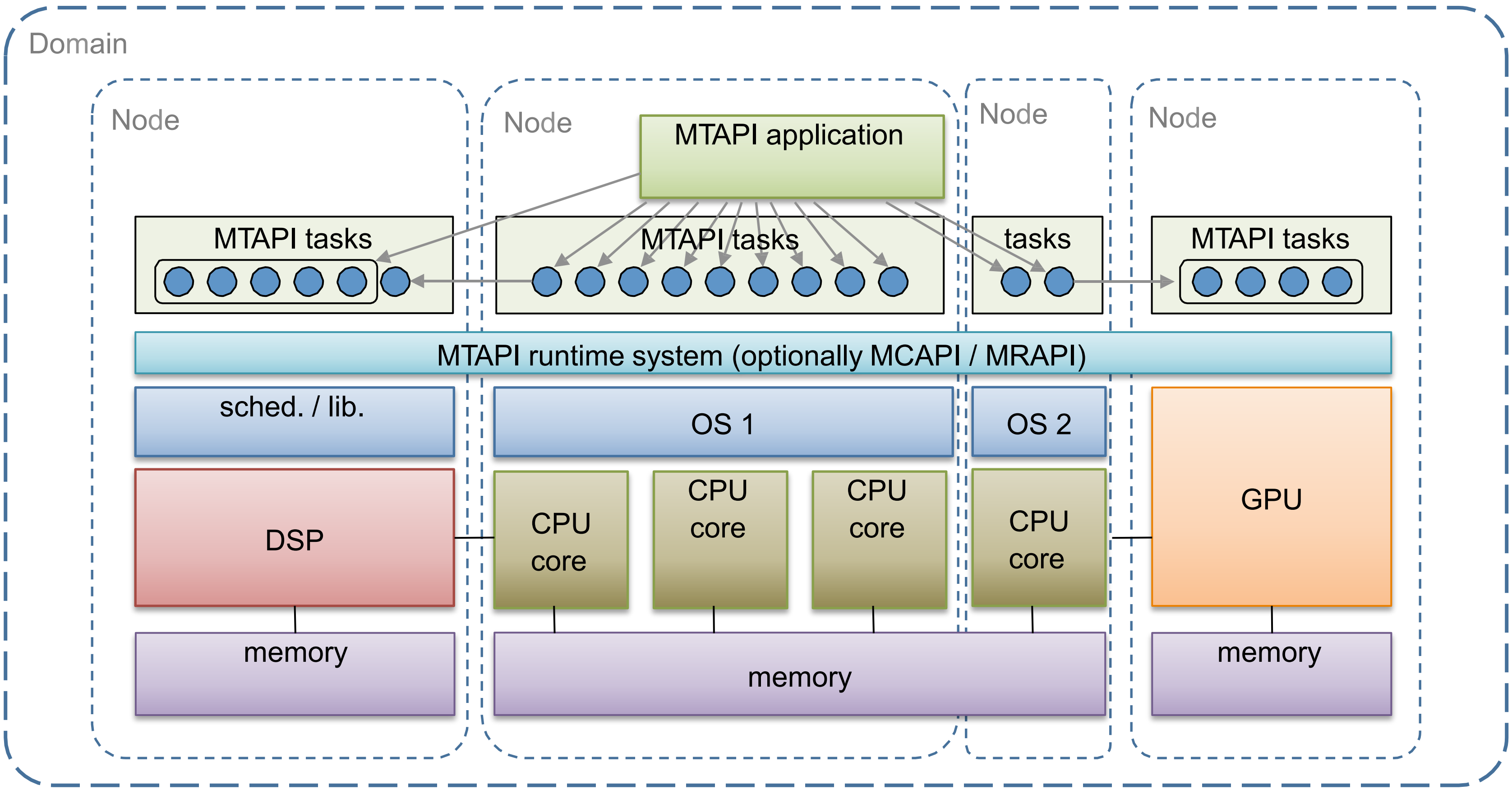
OpenMP
- High-level programming interface
- Increases programmer productivity
- Incremental development model
- Vocabularies for Heterogeneity

Multicore Association libraries
- Provide the low-level portability enablement
- MRAPI – Resource Management
- MCAPI – Communication Management
- MTAPI – Task Management

Prior Work

- Translation of OpenMP to MCA
- Extending MRAPI to support different memory systems
  - On PowerPC processors (IPDPSW 2015, TECHCON 2014, LCTES 2013)
- Communicating between PowerPC and a specialized accelerator (Pattern Matching Engine)
  - TECHCON 2014

MTAPI Framework

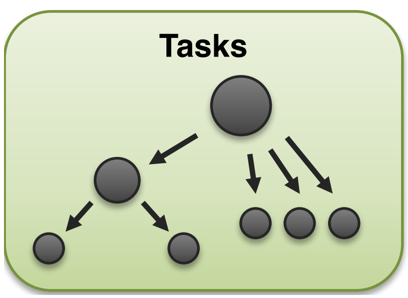


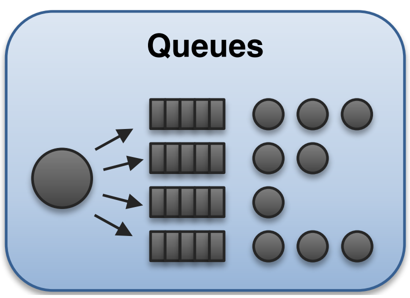
Overview of MTAPI

- Industry standardized API for task-parallel programming for embedded cores
- Contributed by leading industry companies and academia
- Has the potential to support heterogeneous systems, with different memory models or different ISAs
- Scalable
- MTAPI specification is designed for the minimal implementations in plain C that can be built on top of a wide range of OS or even bare metal environment.
- Existing implementations:
  - UH-MTAPI open-source implementation (on-going)
  - Siemens developed an open-source implementation of the MTAPI, part of a larger project called Embedded Multicore Building Blocks (EMBB)

Overview of MTAPI

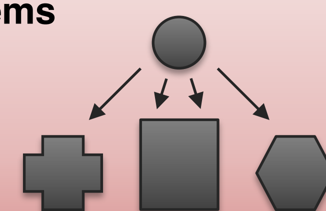
- **Job**: An MTAPI job describes the work to be done. It is an abstraction of the processing implemented in hardware or software by actions. Multiple actions can implemented the same job based on different architectures.
- **Action:** An MTAPI action is the hardware or software implementation of a job. A software action consists of the implementation of the action function on the target processors.
- **Task:** An MTAPI task represents the computation associated with its data environment. A task is a lightweight operation with fine granularity.





Heterogeneous Systems

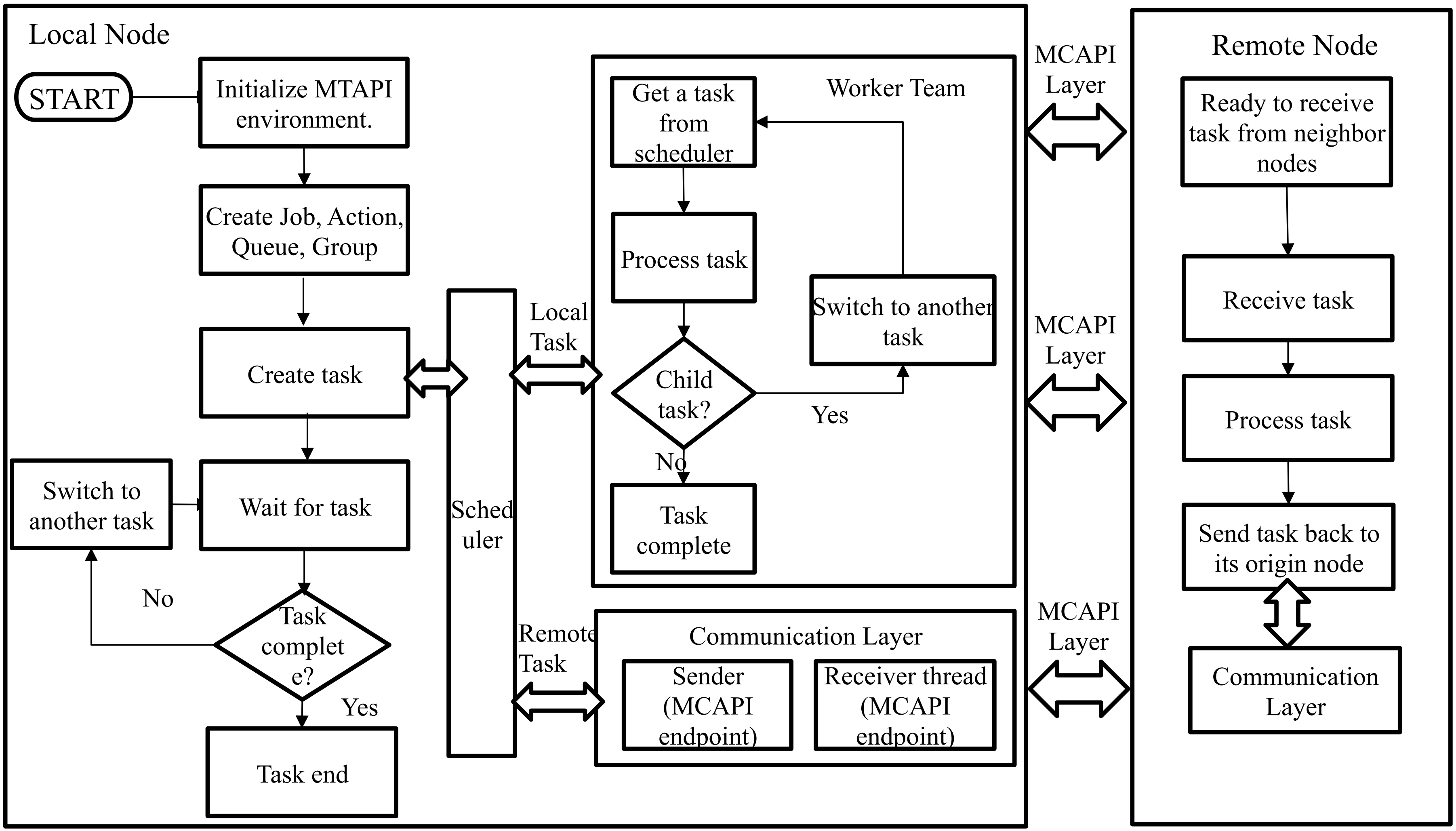
- Shared memory
- Distributed memory
- Different instruction set architectures



2 Approaches to evaluate

- Evaluated stand-alone MTAPI implementation
  - Code written using MTAPI
- Evaluated OpenMP's translation to MTAPI
  - Code written using OpenMP

Stand-alone UH-MTAPI Implementation
MTAPI Work Flow



Testbed, Compiler and Benchmark

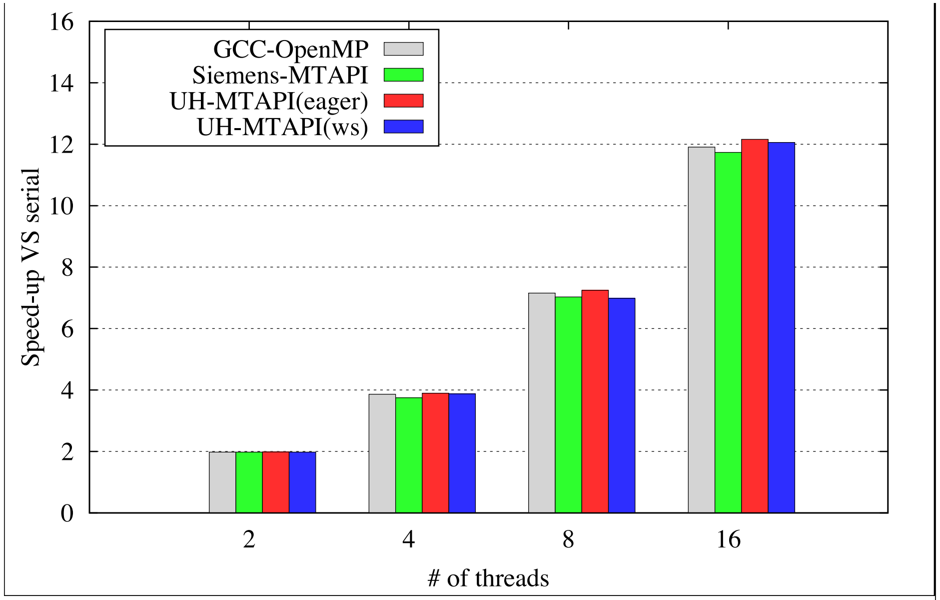
- Test beds:
  - 8 x86-64 Intel Xeon E5-2640 (15M Cache, 2.5 GHz) cores
  - NVIDIA Jetson TK1 embedded development board - 4-Plus-1 Quad-Core ARM Cortex-A15 processor and a Kepler GPU with 192 CUDA cores.
- Compiler:
  - x86-: GCC 4.7.2, NVCC V6.5.12
  - Jetson: GCC 4.8.4, NVCC V6.5.30
- Benchmarks: [1]Rodinia and [2]BOTS.
- Reference implementations:
  - [3]Siemens MTAPI, GNU's OpenMP task implementation

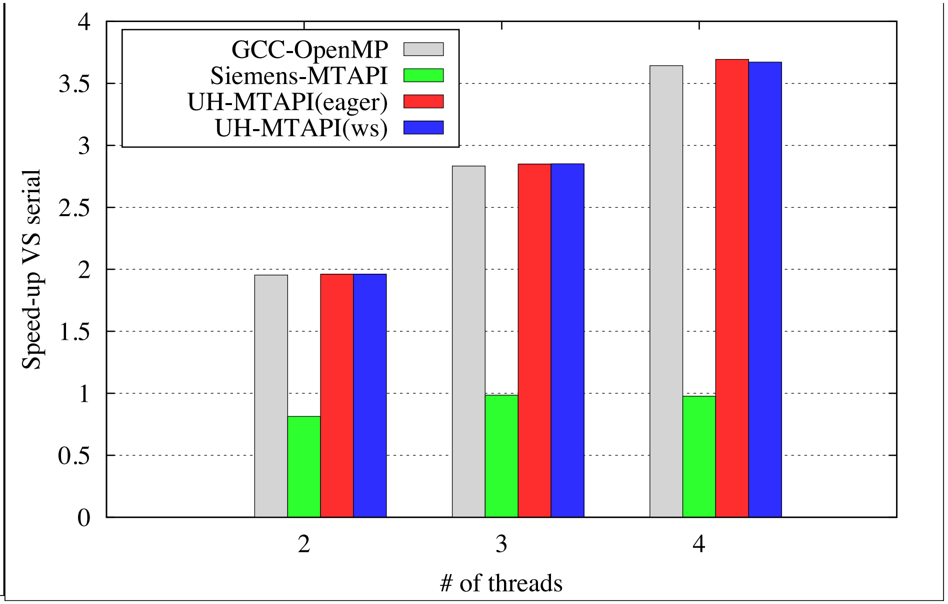
[1]Rodinia: https://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Rodinia:Accelerating_Compute-Intensive_Applications_with_Accelerators

[2]BOTS: https://pm.bsc.es/projects/bots

[3]Siemens-MTAPI: https://github.com/siemens/embb

Stand-alone UH-MTAPI Implementation – Evaluation, Benchmark: Sparse LU

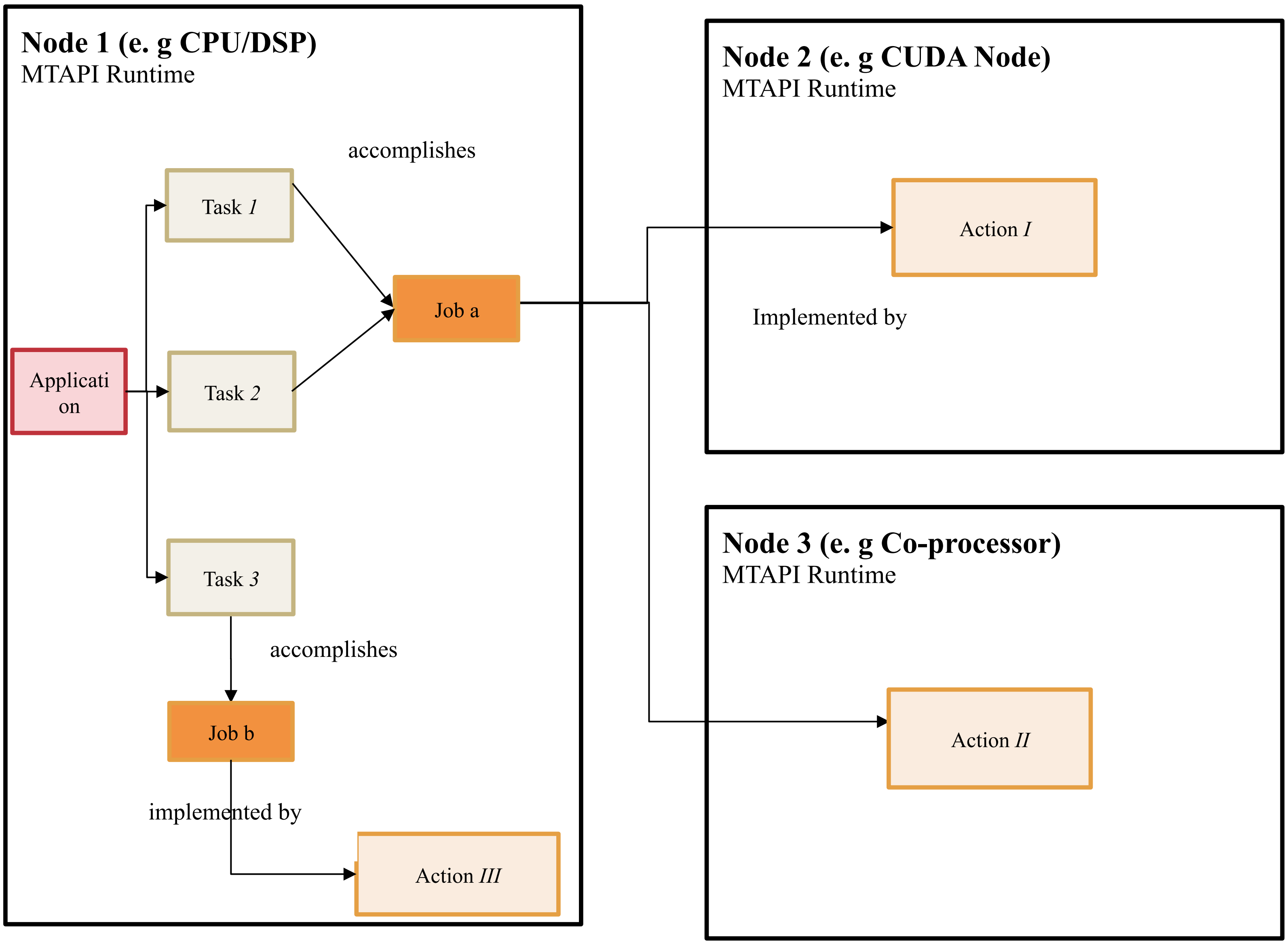




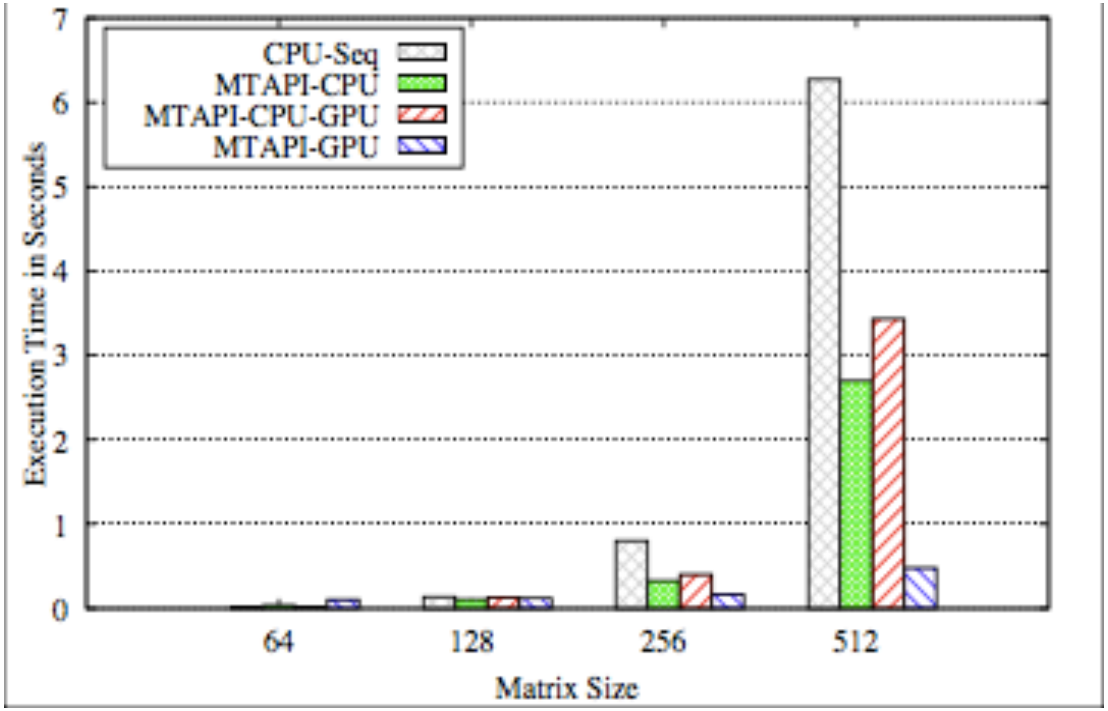
Intel 8 cores
Priority scheduler and work-stealing scheduler.

NVIDIA Jetson TK1 4 ARM cores
Priority scheduler and work-stealing scheduler.

An MTAPI job can have more than 1 action



Stand-alone UH-MTAPI Implementation – Matrix-Matrix Multiplication



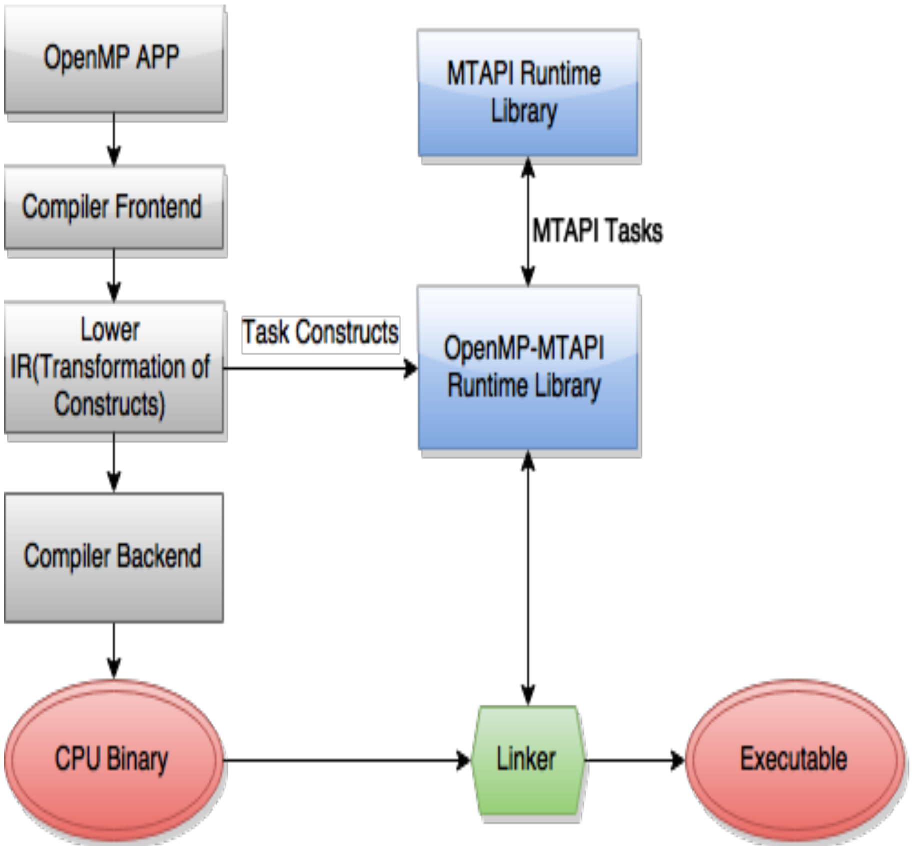
NVIDIA Jetson TK1 embedded development board ARM and GPU
Priority scheduler and work-stealing scheduler.

OpenMP RTL translation to MTAPI

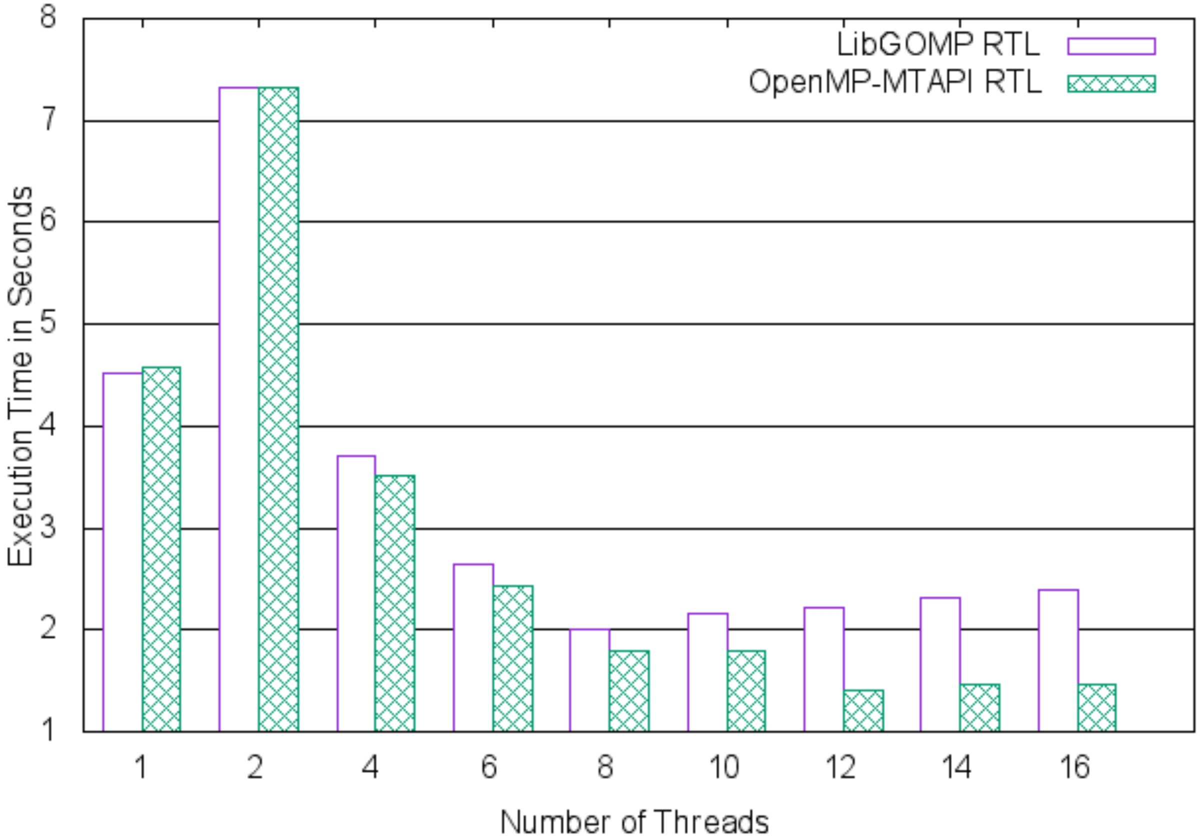
- Parallel Construct incurs MTAPI resource allocation
  - Rely MTAPI to handle thread pool
- Task Construct
  - Task initialization and execution
  - Enter the MTAPI Task Queue
- Taskwait Construct
  - Wait for the descendant task for completion
- Taskgroup Construct
  - Simplified task synchronization mechanism
  - Each MTAPI task will associated with a group ID

OpenMP RTL translation to MTAPI

- By leveraging MTAPI, the enhanced OpenMP RTL could be seamlessly mapped to various possible architectures.
- Compiler frontend translates OpenMP constructs to OpenMP – MTAPI RTL function calls.
- In the RTL, we implement the MTAPI function calls and convert OpenMP tasks to MTAPI objects
- Our RTL will be linked to the OpenMP applications during the run time.
- Thread pool and other computation resources will rely on MTAPI for management



OpenMP-> MTAPI Implementation – Sparse LU



Contributions

- Created a task-based implementation for embedded systems using industry standards - MTAPI
- Translated OpenMP to MTAPI
  - Further abstraction
  - Easier to program, Less tedious
  - Faster software development time
  - Faster time to market solutions (TTM)
  - Code once, multiple reuses
- Targeted more than one platform
  - 8 Intel x86 cores
  - 4 ARM cores from NVIDIA Jetson TK1 + 1 Kepler GPU with 192 cores
- Comparable performance, lesser to no overhead

Peng Sun, Sunita Chandrasekaran, Barbara Chapman, "Deploying OpenMP Task Parallelism on Multicore Embedded Systems with MCA Task APIs", In Proc. Of IEEE HPCC (to appear)

Acknowledgment

- Peng Sun, Ph.D. student
- Suyang Zhu, M.S. student
- Cheng Wang, Ph.D. Student
- Freescale Semiconductor