OpenMP for Embedded Systems

Sunita Chandrasekaran
Asst. Professor
Dept. of Computer & Information Sciences
University of Delaware
schandra@udel.edu

ACK: Peng Sun, Suyang Zhu, Cheng Wang, Barbara Chapman, Tobias Schuele, Marcus Winter

Heterogeneous Embedded Systems

- Incorporates specialized processing capabilities to handle specific tasks
- Example
    - CPU + GPU
    - ARM + GPU
    - ARM + DSP
    - CPU + FPGA

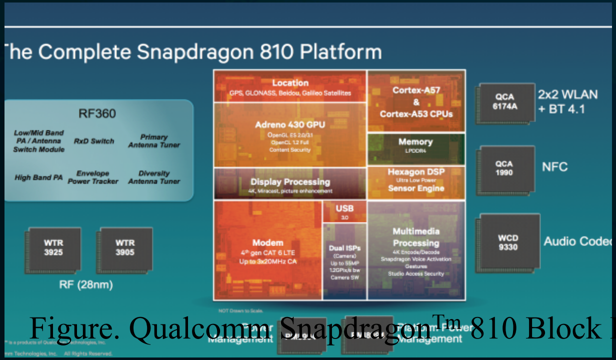


Figure. Qualcomm Snapdragon™ 810 Block Diagram

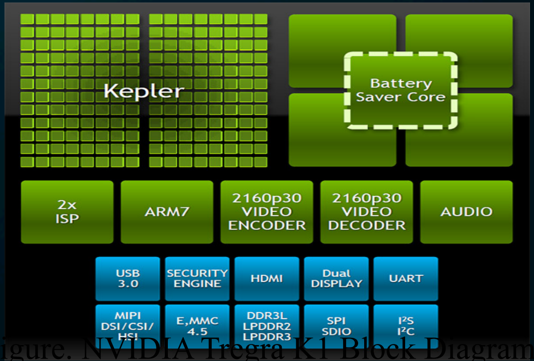


Figure. NVIDIA Tegra K1 Block Diagram

Programming Multicore Embedded Systems – A Real Challenge

- Heterogeneous systems present complexity at both silicon and system level
- Standards and tool-chain in embedded industry are proprietary
- Portability and scalability issues
- High time-to-market (TTM) solutions
- We need industry standards
  - To offer portable and scalable software solutions and target more than one platform

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

  - Not portable across more than one type of platform except for OpenCL
  - Most models are heavy-weight for embedded processors of limited resources
  - Most models require support from OS and compilers
    - Sometimes embedded systems are bare-metal
  - Some of the solutions are restricted to just the homogeneous environment

So what do we really need?

- Something that's not too low-level
- Something light-weight
- Something that can target heterogeneous embedded platforms (beyond CPUs-GPUs)
- Something that can help speed time-to-market for products
- Last but not the least – we need industry standards

Using industry standards

- Two of them used for this work
  - OpenMP
    - (high-level, directive-based)
  - Multicore Association (MCA) APIs
    - (low-level, light-weight catered to embedded systems)

Briefly, on OpenMP Implementations

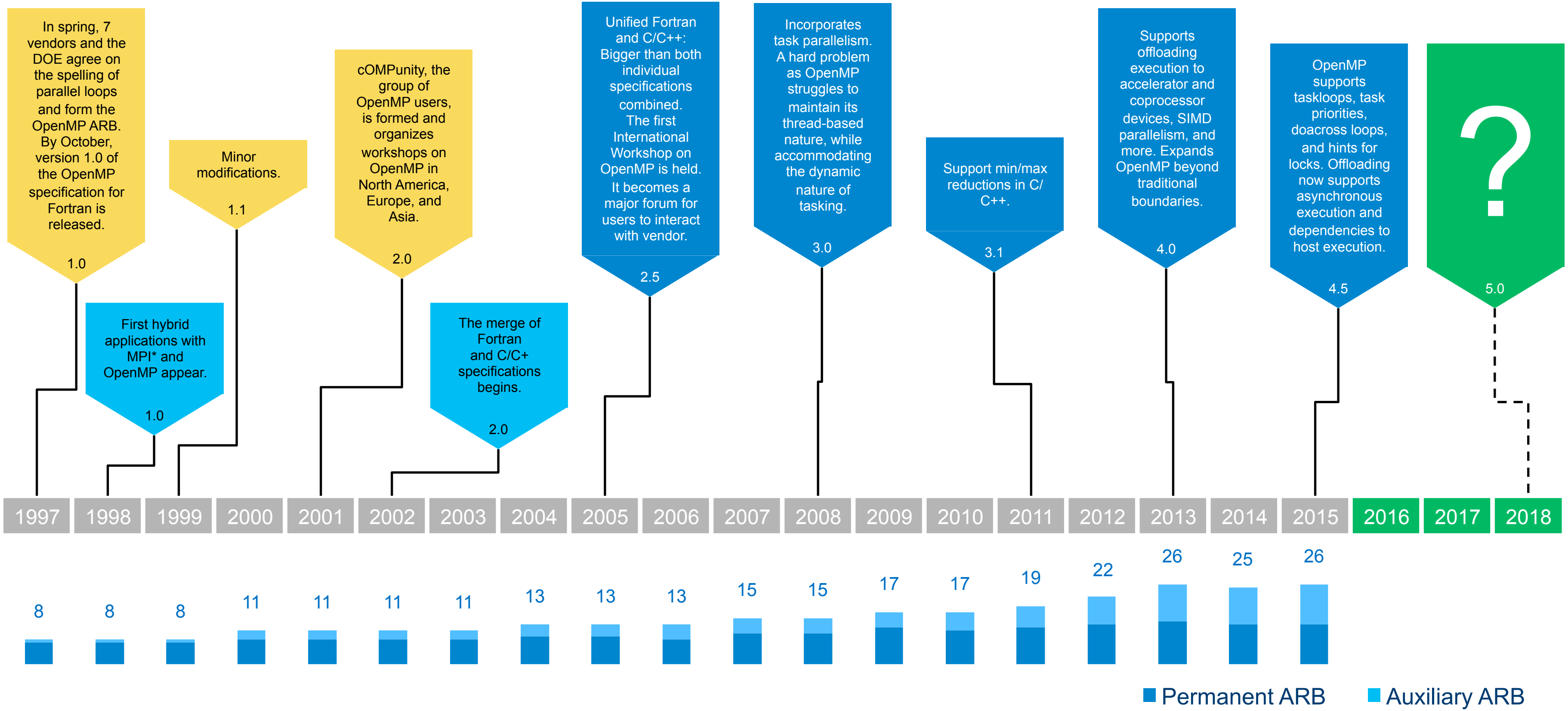
- Directives implemented via code modification and insertion of runtime library calls
  - Typical approach is outlining of code in parallel region
  - Or generation of micro tasks
- Runtime library responsible for managing threads
  - Scheduling loops
  - Scheduling tasks
  - Implementing synchronization
- 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.

History of OpenMP*



Multicore Association APIs (MCA)

- Develops standards to reduce complexity involved in writing software for multicore chips
- Capturing the basic elements and abstract hardware and system resources
- Cohesive set of foundation APIs

  Standardize communication (MCAPI)

  Resource Sharing (MRAPI)

  Task Management (MTAPI)

SIEMENS

TEXAS INSTRUMENTS

QUALCOMM

ENEA

PolyCore Software

freescale semiconductor

LSI

PLURALITY

WIND RIVER

Multicore Task Management API (MTAPI)

MTAPI

- **Standardized API** for task-parallel programming on a wide range of hardware architectures
- Developed and driven by practitioners of **market-leading companies**
- Part of Multicore-Association's **ecosystem** (MRAPI, MCAPI, SHIM, OpenAMP, …)

THE Multicore™ ASSOCIATION

Contributing members:

SIEMENS
Working group lead

UH

TEXAS INSTRUMENTS

QUALCOMM

ENEA

PolyCore Software

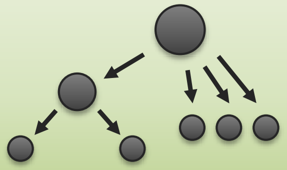
freescale semiconductor

LSI

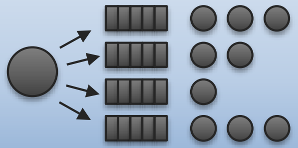
PLURALITY

WIND RIVER

Tasks

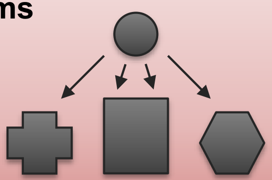


Queues



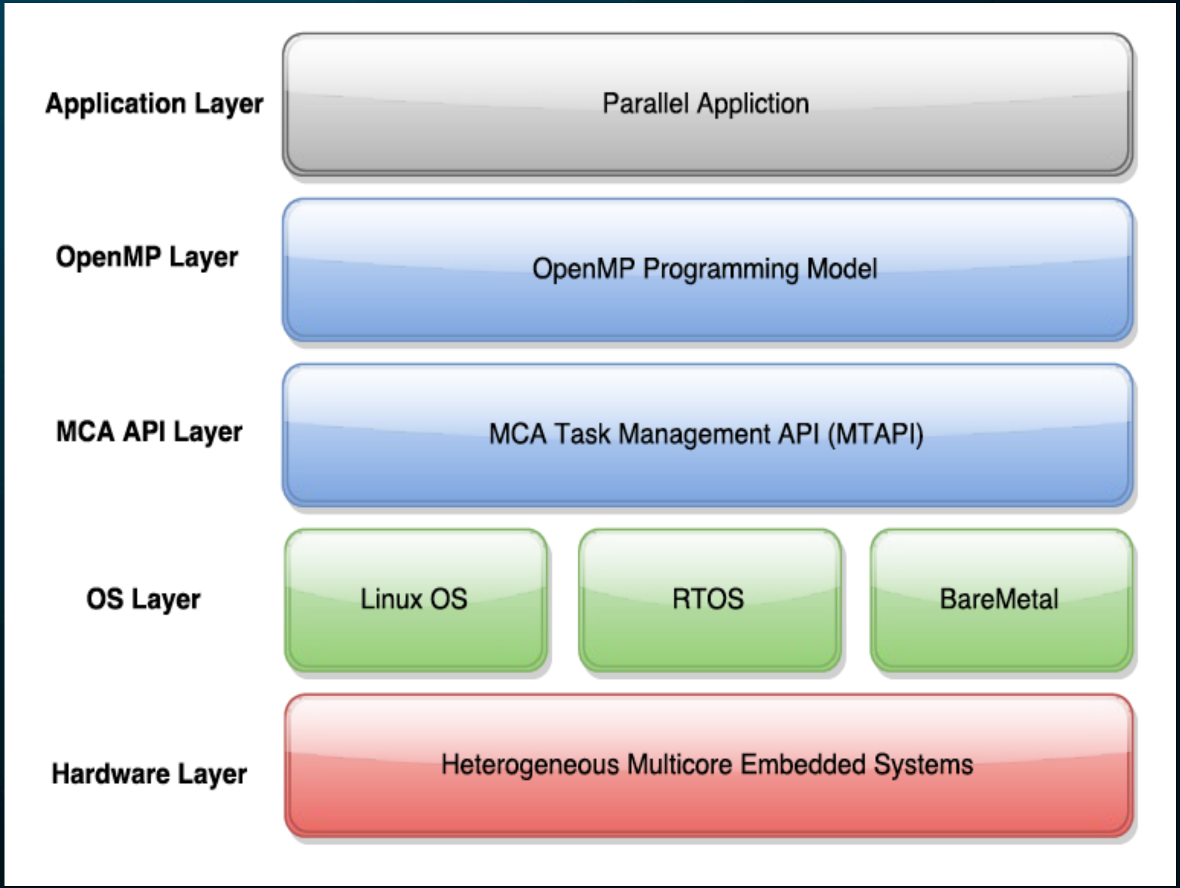
Heterogeneous Systems

- Shared memory
- Distributed memory
- Different instruction set architectures



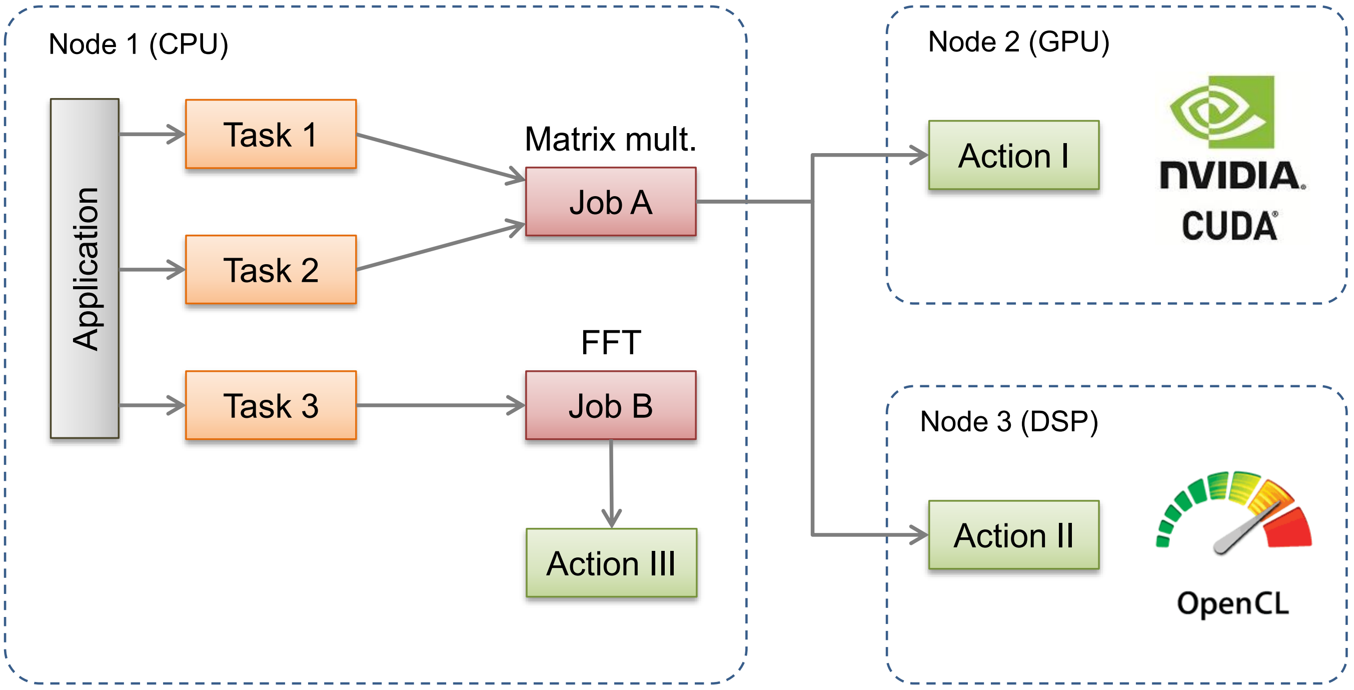
Ack: Siemens (Tobias Schuele, Urs Gleim)

OpenMP and MCA software stack



MTAPI Jobs, Tasks & Action

Example for the usage of MTAPI in heterogeneous systems:

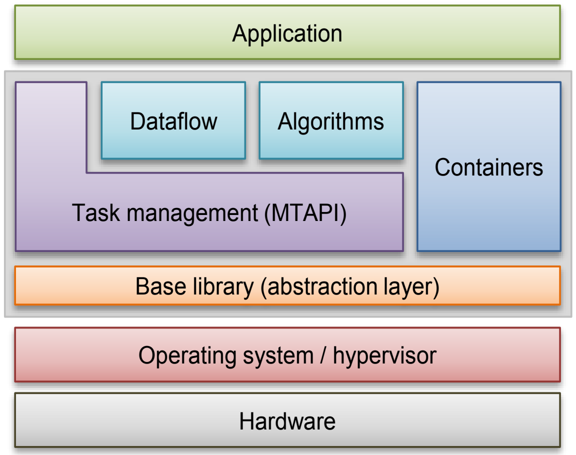


Ack: Siemens (Tobias Schuele, Urs Gleim)

MTAPI implementations

Embedded Multicore Building Blocks (EMB2)[1]

- Open source library and runtime platform for embedded multicore systems
- Real-time capability, resource awareness
- Fine-grained control over core usage (task priorities, affinities)

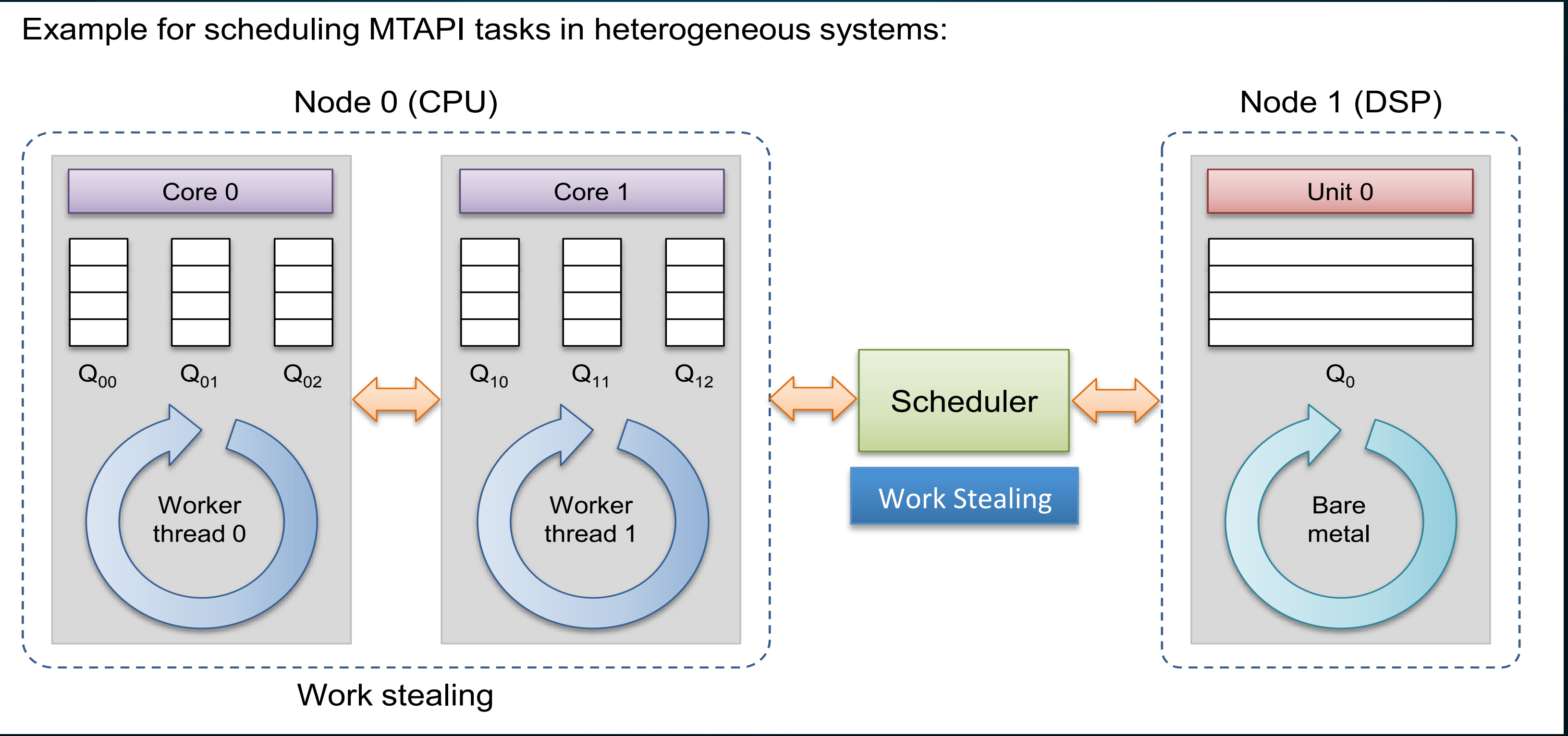


MTAPI implementation developed at the Universities of Houston / Delaware[2]

- Utilizes MCAPI for inter-node communication and MRAPI for resource management
- Used as runtime system for OpenMP programs

[1] https://github.com/siemens/embb
[2] https://github.com/MCAPro2015/OpenMP_MCA_Project

MTAPI Scheduling



Ack: Siemens (Tobias Schuele, Urs Gleim)

Testbed, Compiler and Benchmark

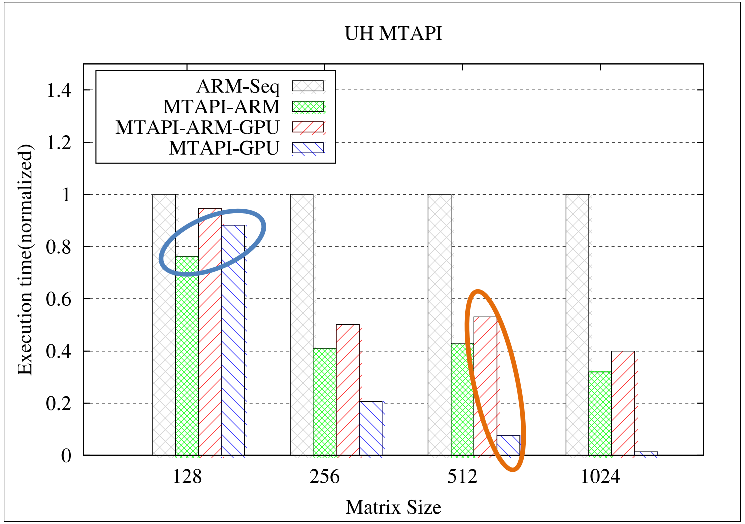
- Test beds:

  NVIDIA Jetson TK1 embedded development board with a Tegra K1 Soc (NVIDIA 4-Plus-1 Quad-Core ARM Cortex-A15 processor and a Kepler GPU with 192 CUDA cores).
- Compiler: Jetson (GCC 4.8.4, NVCC V6.5.30)
- Power Architecture from Freescale
  - Consisting of Pattern Matching Engine as specialized accelerator
- Benchmarks: [1]Rodinia and [2]BOTS.
- Reference Group: [3]Siemens MTAPI, GCC OpenMP

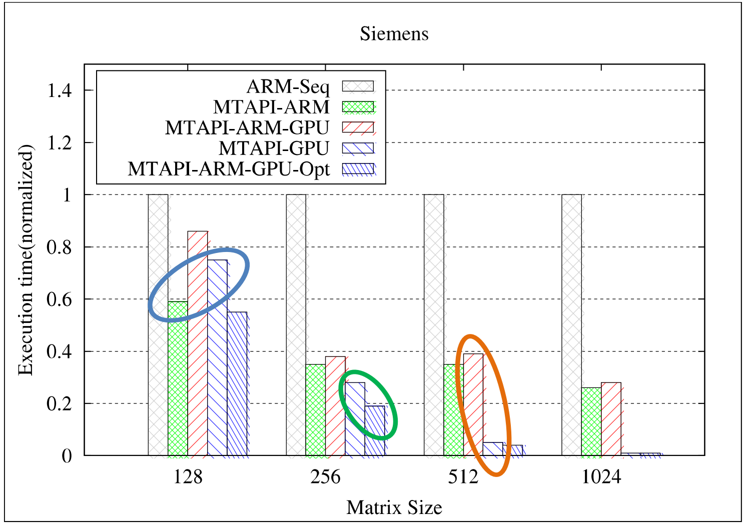
[1]Rodinia: https://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Rodinia:Accelerating_Compute-Intensive_Applications_with_Accelerators
[2]BOTS: https://pm.bsc.es/projects/bots
[3]Siemens-MTAPI: https://github.com/siemens/embb

Normalized execution times for UH-MTAPI and Siemens MTAPI (EMB^2) for MM

Normalized execution times for UH-MTAPI and Siemens MTAPI (EMB^2):



(a) UH-MTAPI

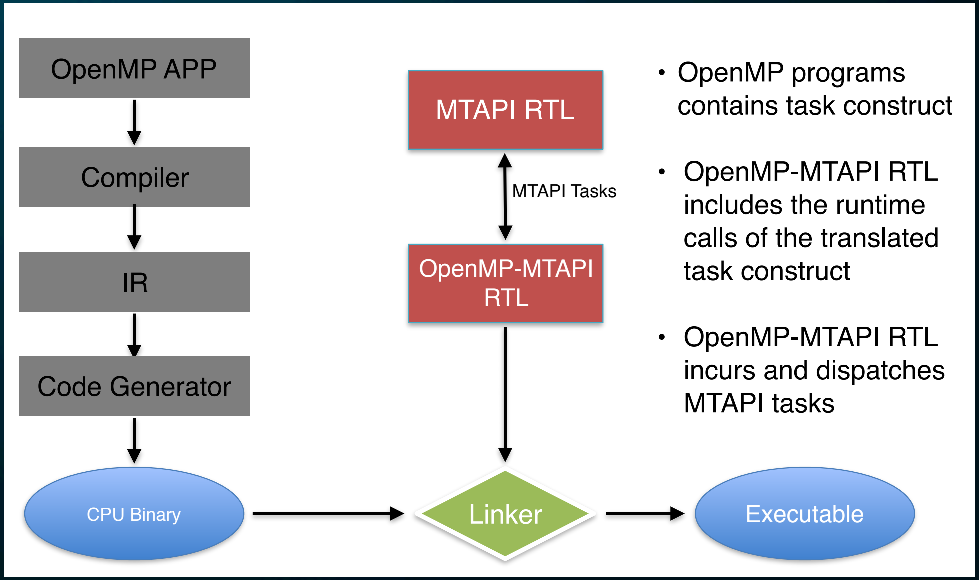


(b) Siemens MTAPI

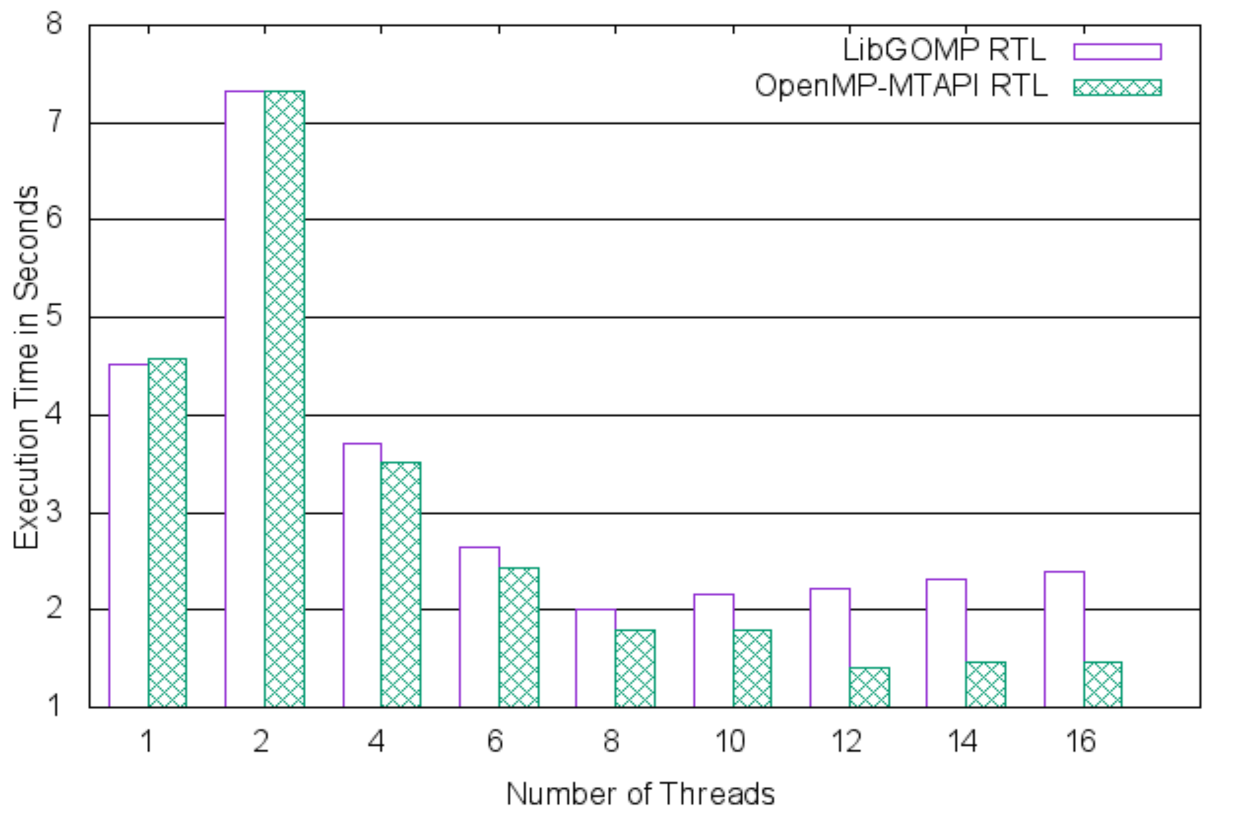
- MTAPI-ARM faster than MTAPI-GPU for small matrices due to overhead for data copying
- MTAPI-GPU faster than MTAPI-ARM-GPU for larger matrices due to load imbalance
- MTAPI-ARM-GPU-Opt always fastest due to asynchronous transfers and variable block sizes

OpenMP RTL translation to MTAPI

- Compiler front end translates OpenMP constructs to MTAPI-RTL functions
- RTL comprises of MTAPI function calls and we convert OpenMP tasks to MTAPI objects
- Embedded resources will rely on MTAPI for management of resources



OpenMP-> MTAPI Implementation SparseLU



Takeaways and Summary

- Industry standards are the way to go !
- OpenMP-MCA incurred little to no overhead
  - Targeting heterogeneous platforms
- Less learning curve
- Ability to maintain single code base