SIEMENS

EuroPAR 2016 | ROME Workshop

# Exploring Task Parallelism for Heterogeneous Systems Using Multicore Task Management API

Suyang Zhu[1], Sunita Chandrasekaran[2], Peng Sun[1], Barbara Chapman[1], Marcus Winter[3], Tobias Schuele[4]

[1] Dept. of Computer Science, University of Houston
[2] Dept. of Computer and Information Sciences, University of Delaware
[3] Hypnolords Gbr
[4] Siemens Corporate Technology

# Introduction
## Current Trends in Embedded Systems

**Embedded systems are everywhere:**

- Industrial automation
- Energy production and distribution
- Healthcare / medical imaging
- Transportation and traffic control
- Consumer electronics
- …

**Requirements and key characteristics:**

- Real-time capability (progress guarantees, nonblocking operations)
- Resource awareness (no dynamic memory allocation during operation)
- Portability / platform independence
- Energy efficiency
- Fine-grained control over hardware
- Heterogeneous systems
- …

**Industry 4.0**

Source: Siemens

**Autonomous driving**

Source: Bosch

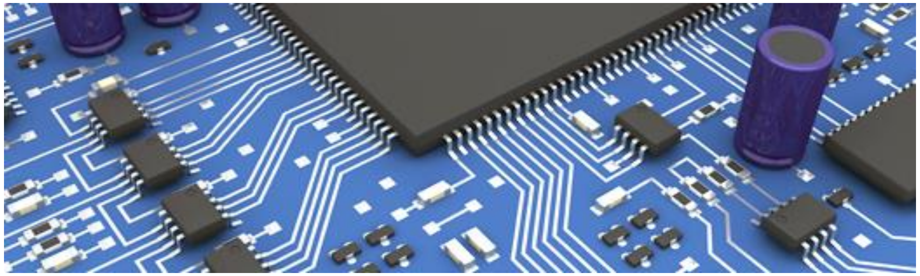
**In-field data analytics**

Source: Siemens

**Augmented / virtual reality**

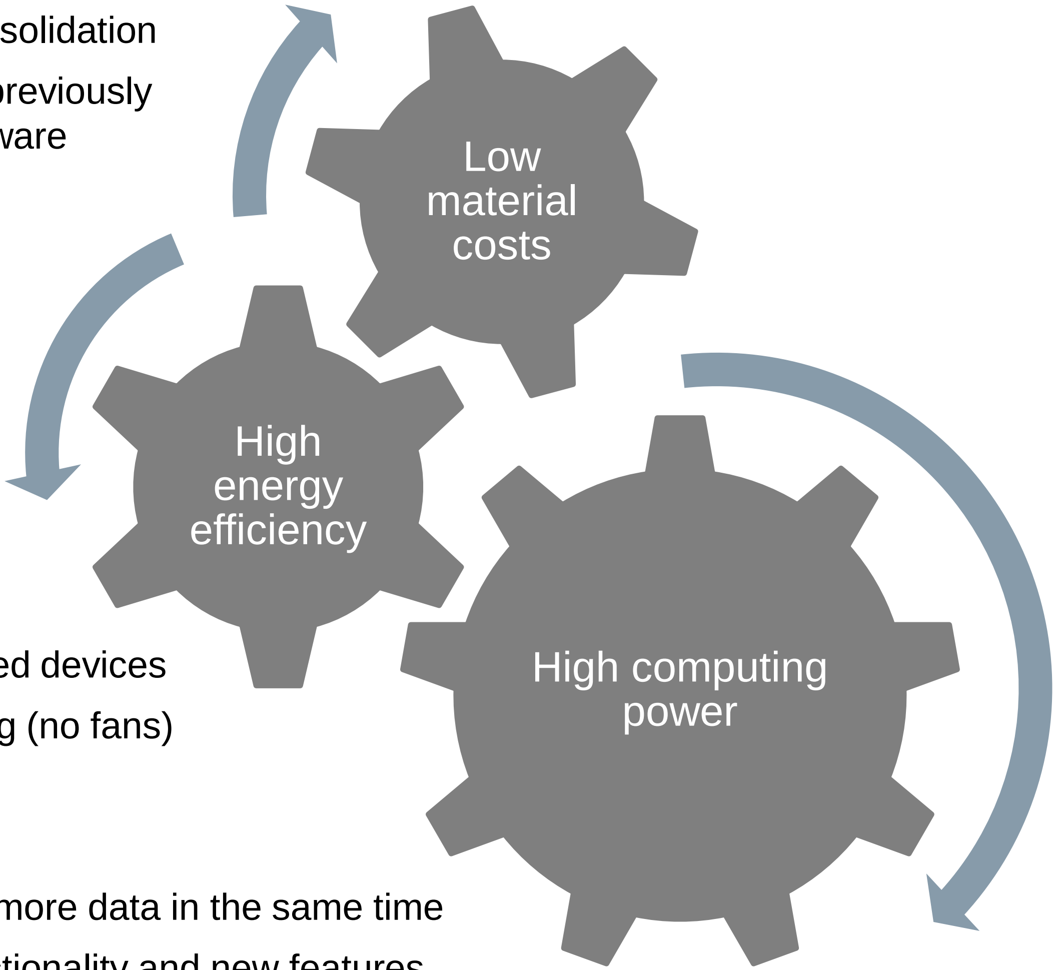
Source: Siemens

⋮ ⋮



# Benefits of Multi-/Manycore in Embedded Systems

- Functional consolidation
- Integration of previously separate hardware




- Battery-powered devices
- Passive cooling (no fans)

- Processing of more data in the same time
- Additional functionality and new features

Source: Siemens

# Introduction
## “In 2022, multicore will be everywhere.” (IEEE CS)

OpenMP

Open MPI

Microsoft
Parallel Patterns Library

intel
Threading Building Blocks


Most frameworks for parallel programming target **desktop / server / HPC applications**.

⇨ **Not suitable for embedded systems**

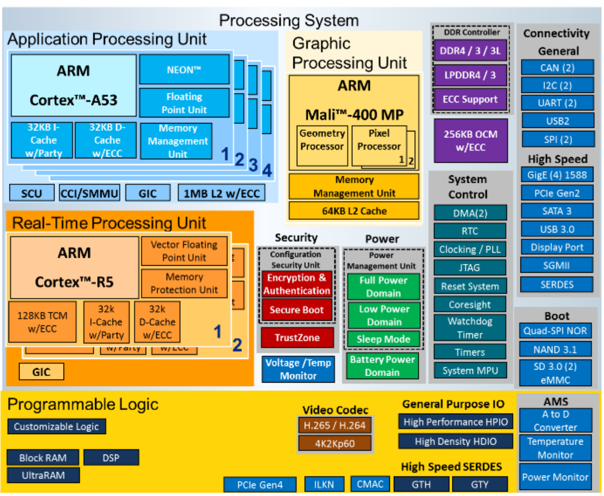
### Top challenges for multicore (IEEE CS 2022 Report)[1]

- Hard **real-time architectures** with local memory and their programming
- Low-power scalable **homogeneous** and **heterogeneous architectures**
- …

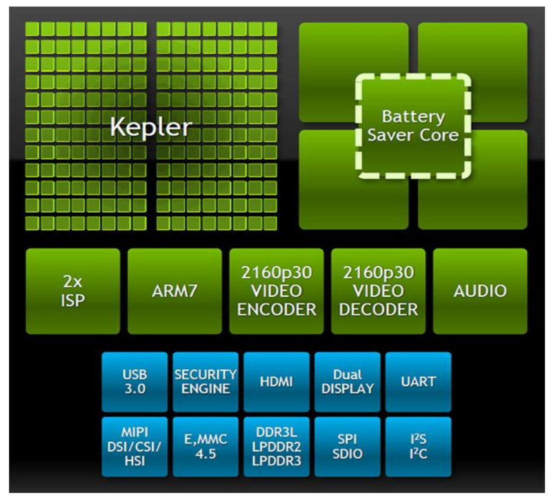
[1] H. Alkhatib, P. Faraboschi, E. Frachtenberg, H. Kasahara, D. Lange, P. Laplante, A. Merchant, D. Milojicic, and K. Schwan. *IEEE CS 2022 Report*. IEEE Computer Society, 2014. www.computer.org/cms/Computer.org/ComputingNow/2022Report.pdf

# Introduction
## Heterogeneous Systems

- Heterogeneous architectures provide **high performance at low power consumption** by incorporating **specialized processing units** to handle particular tasks.
- Processor manufacturers integrate **general purpose processors** together with **accelerators like GPUs and FPGAs** on the same chip.




Xilinx Zynq UltraScale MPSoC




Nvidia Tegra K1


Qualcomm Snapdragon 810

➪ **Increased complexity at silicon and system level**

➪ **Proprietary interfaces and tool-chains**

➪ **Long time-to-market, lack of portability**

# Programming Model
## Multicore Task Management API (MTAPI)

The Multicore Association develops and promotes open specifications for multicore product development.

### 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, …)

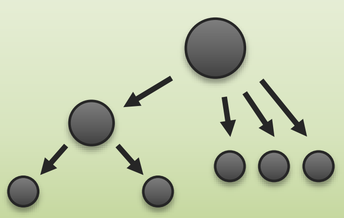
THE Multicore ASSOCIATION

Contributing members:

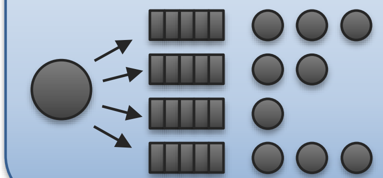
SIEMENS
Working group lead

UH, Texas Instruments, Qualcomm, ENEA, PolyCore Software, freescale semiconductor, LSI, Plurality, Wind River

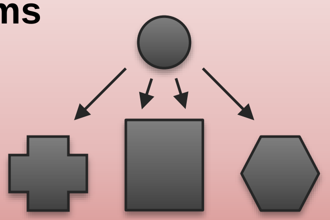
**Tasks**



**Queues**



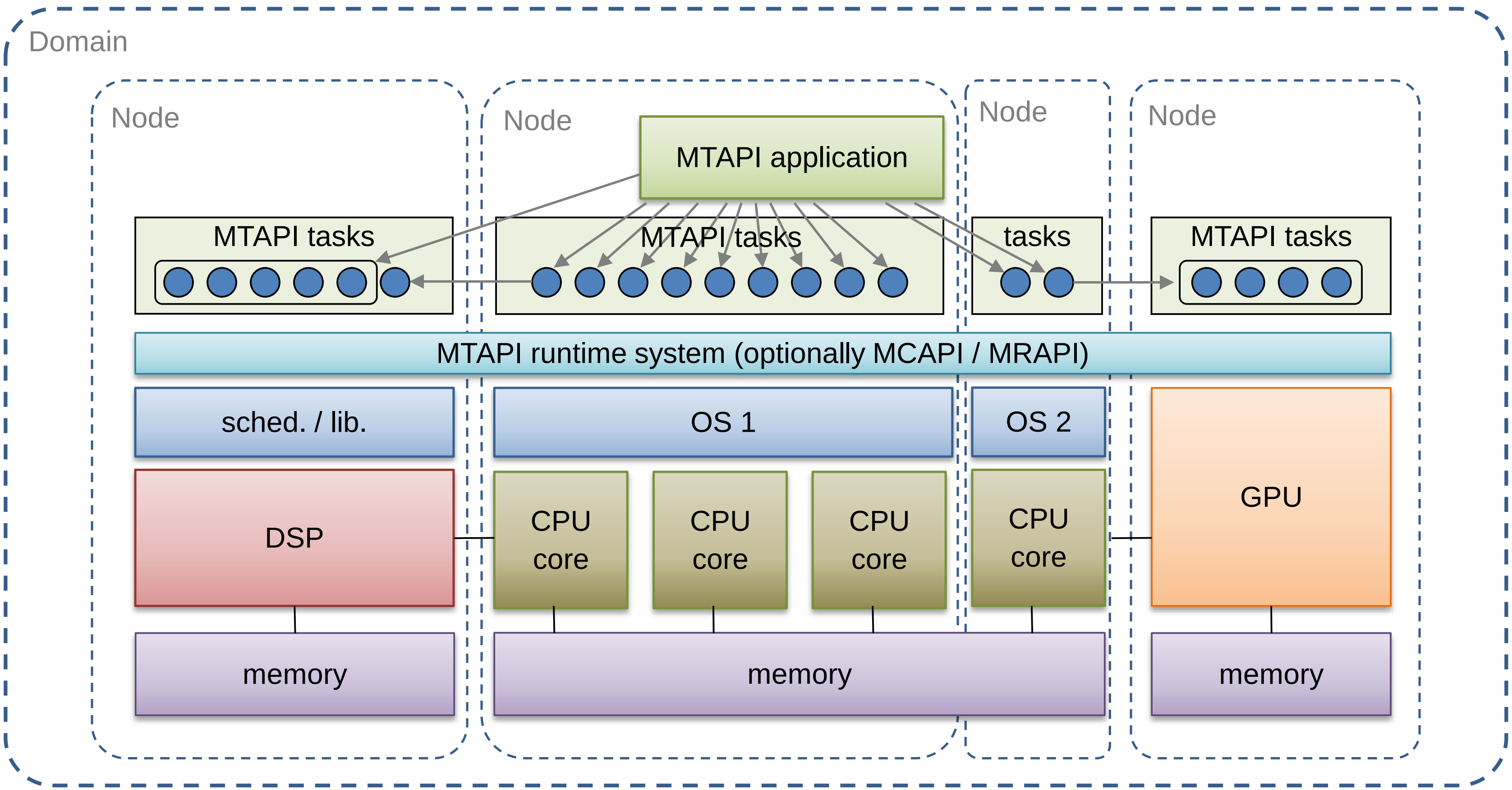
**Heterogeneous Systems**
- Shared memory
- Distributed memory
- Different instruction set architectures



# Programming Model
# MTAPI for Heterogeneous Systems

SIEMENS

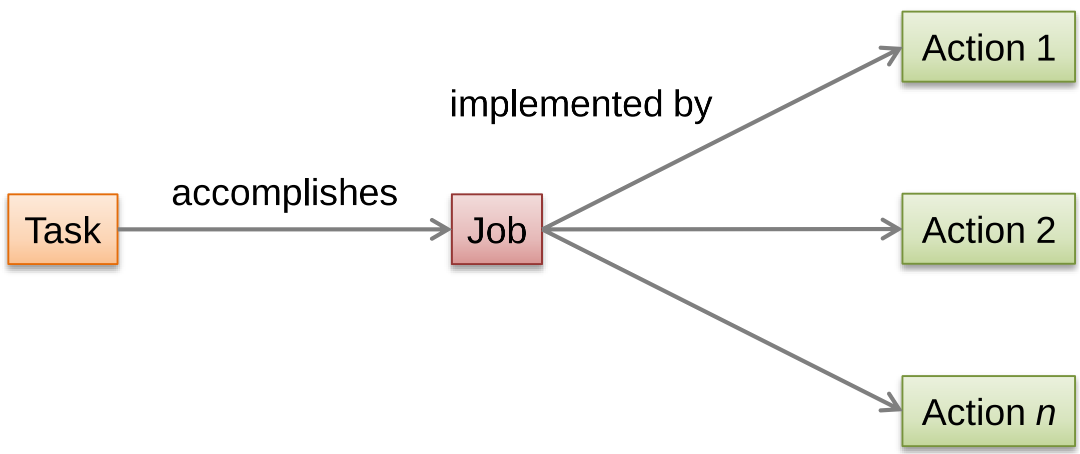
Heterogeneous systems are modelled using MTAPI nodes and domains.



# Programming Model
## MTAPI Terms in a Nut Shell

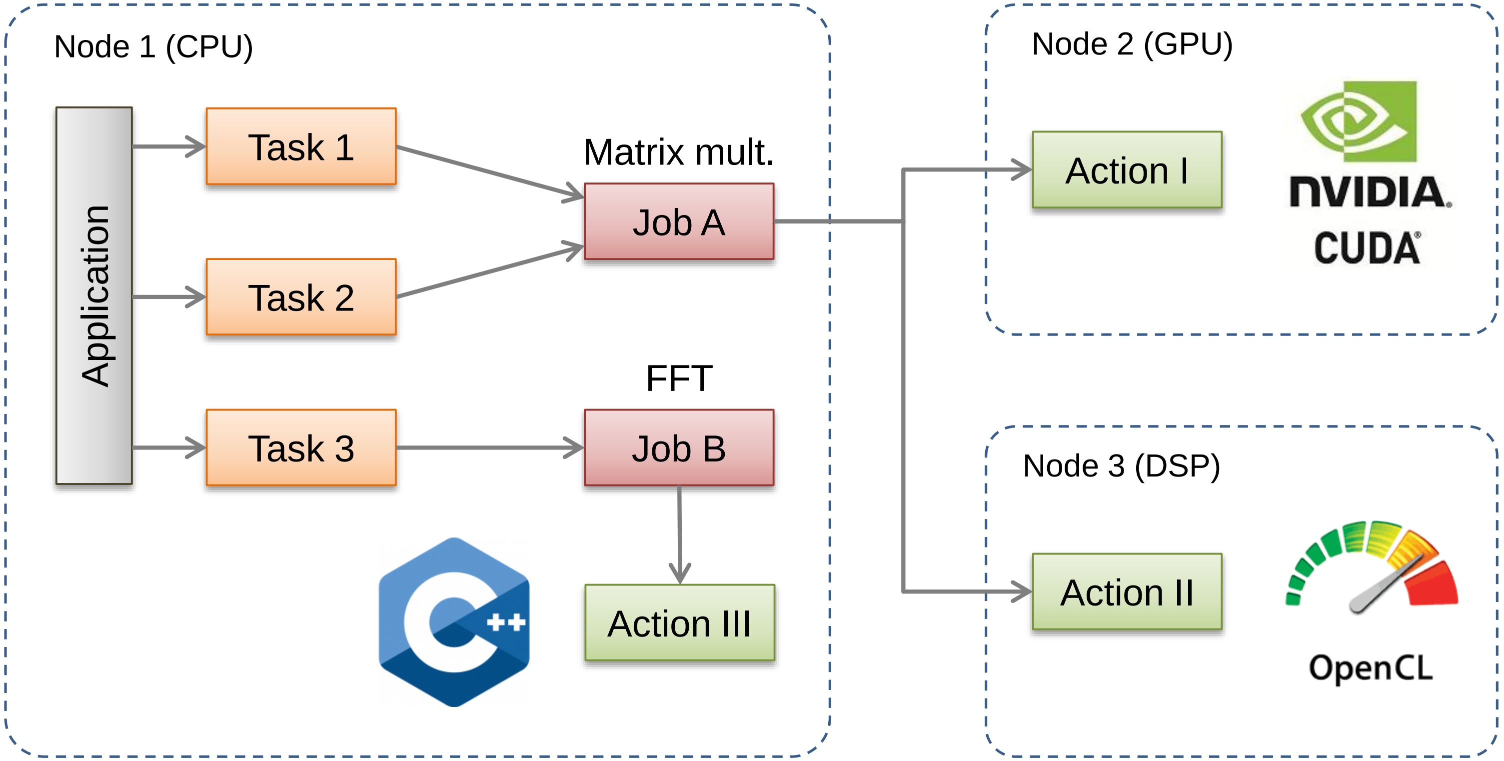
MTAPI distinguishes between jobs, actions, and tasks:

- **Job**: A piece of processing implemented by an action. Each job has a unique identifier.
- **Action**: Implementation of a job, may be hardware or software-defined.
- **Task**: Execution of a job resulting in the invocation of an action implementing the job associated with some data to be processed.




# Programming Model
## MTAPI for Heterogeneous Systems (cont.)

Example for the usage of MTAPI in heterogeneous systems:



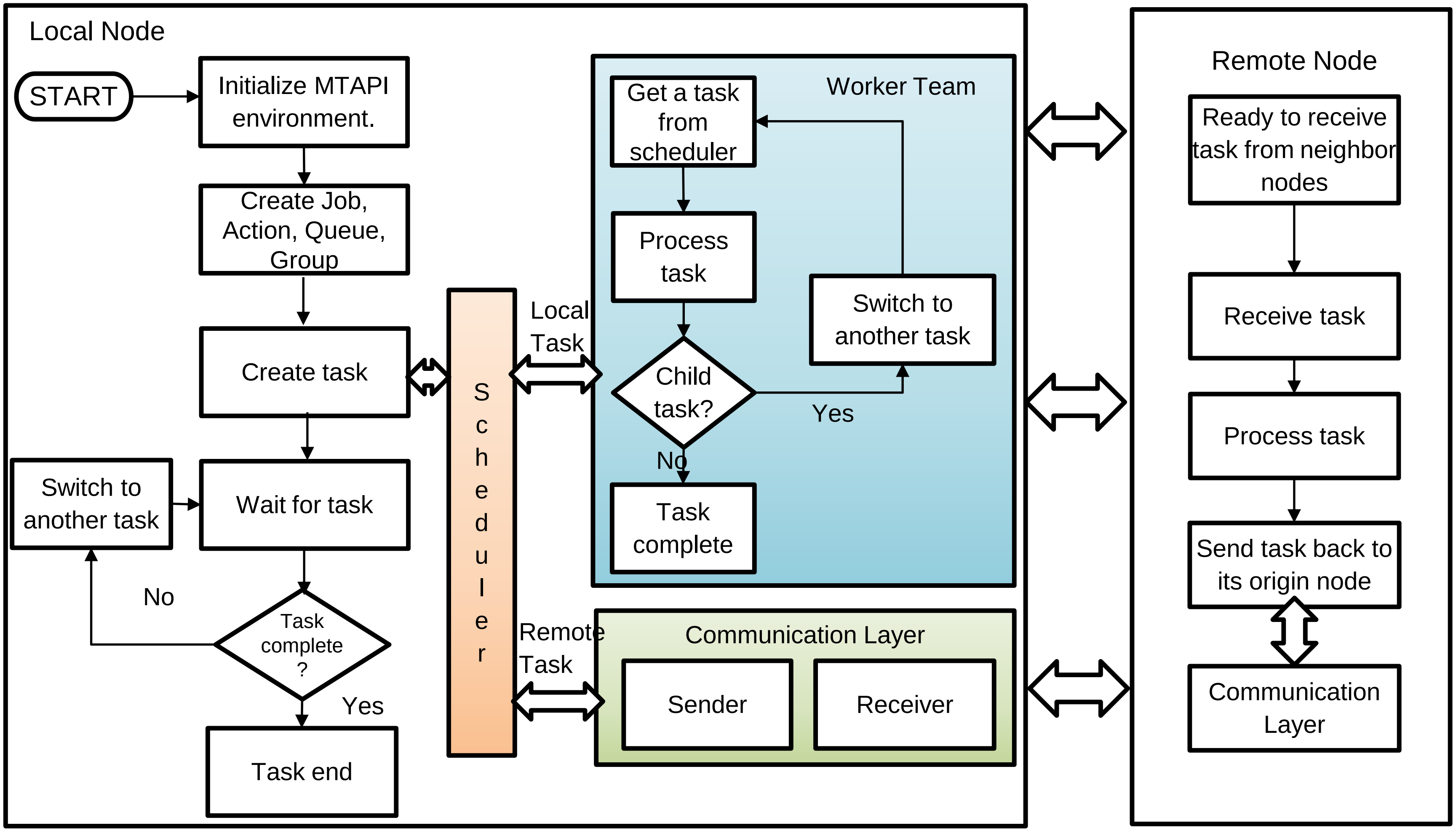
## Programming Model
# MTAPI for Heterogeneous Systems (cont.)

SIEMENS

### Example with three MTAPI jobs

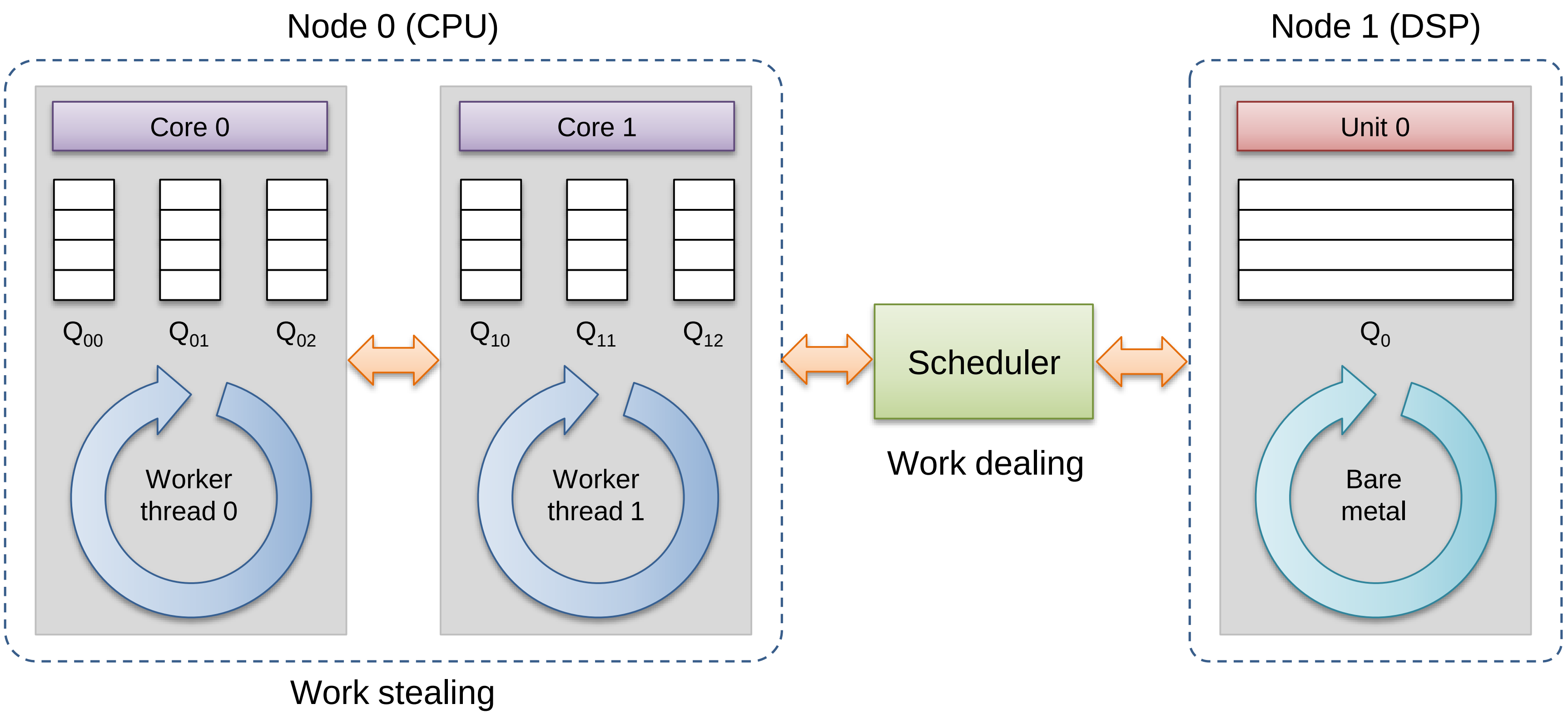
```
// Define actions
void Action_I(...) {CUDA_Kernel(arg->A, arg->B, arg->C, arg->n);}
void Action_II(...) {OpenCL_Kernel(arg->A, arg->B, arg->C, arg->n);}
void Action_III(...) {CPP_Kernel(arg->A, arg->B, arg->C, arg->n);}

// Create actions and associate them with jobs
mtapi_action_create(JOB_A, Action_I, ...);
mtapi_action_create(JOB_A, Action_II, ...);
mtapi_action_create(JOB_B, Action_III, ...);

// Start tasks
mtapi_task_hndl_t task[3];
task[0] = mtapi_task_start(0, JOB_A, args0, ...);
task[1] = mtapi_task_start(0, JOB_A, args1, ...);
task[2] = mtapi_task_start(0, JOB_B, args2, ...);

// Wait for task completion
mtapi_task_wait(task[0], MTAPI_INFINITE, ...);
mtapi_task_wait(task[1], MTAPI_INFINITE, ...);
mtapi_task_wait(task[2], MTAPI_INFINITE, ...);
```

# Implementation
## MTAPI Flow Chart



# Implementation
# MTAPI Scheduling

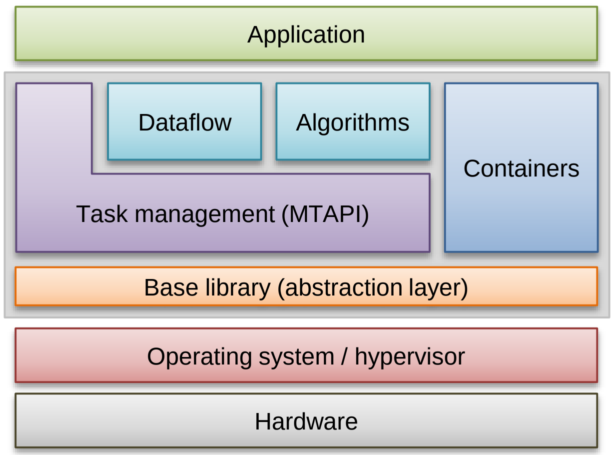
Example for scheduling MTAPI tasks in heterogeneous systems:



# Performance Evaluation
## MTAPI Implementations

### Embedded Multicore Building Blocks (EMB$^2$)[1]

- Open source library and runtime platform for embedded multicore systems
- Easy parallelization of existing code using high-level patterns
- Real-time capability, resource awareness
- Fine-grained control over core usage (task priorities, affinities)
- Lock-/wait-free implementation




### UH-MTAPI[2]

- MTAPI implementation developed at the Universities of Houston / Delaware
- Utilizes MCAPI for inter-node communication and MRAPI for resource management
- Has been used as runtime system for OpenMP programs

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

# Performance Evaluation
## Testbed and Benchmarks

**Reference platform:**

- NVIDIA Jetson TK1 development kit
- Tegra K1 SoC which contains
  - NVIDIA 4-Plus-1 Quad-Core ARM Cortex-A15 processor
  - Kepler GPU with 192 CUDA cores

Source: Nvidia

**Compiler:**

- GCC 4.8.4
- NVCC V6.5.30

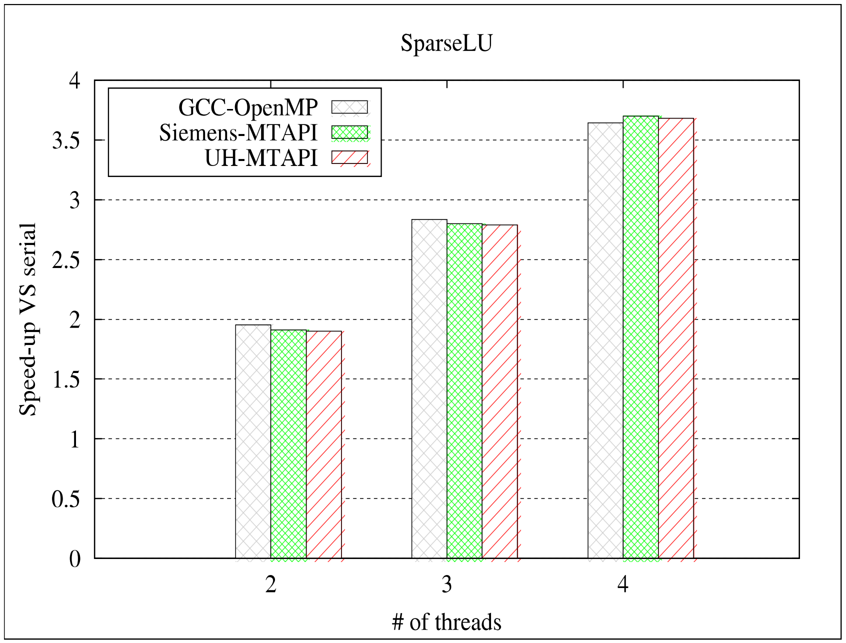
**Benchmarks:**

- Rodinia: Accelerating Compute-Intensive Applications with Accelerators[1]
- Barcelona OpenMP Task Suite (BOTS)[2]

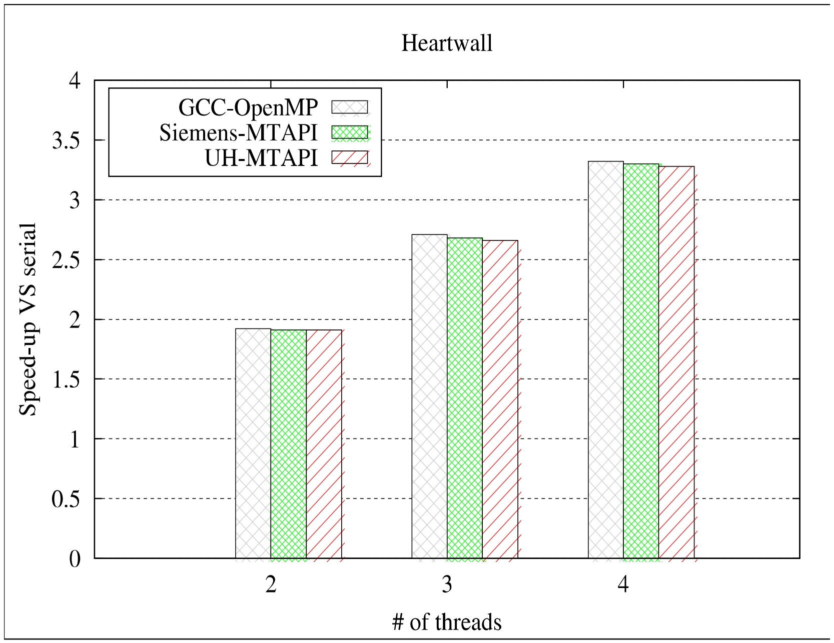
[1] https://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Rodinia:Accelerating_Compute-Intensive_Applications_with_Accelerators
[2] https://pm.bsc.es/projects/bots

# Performance Evaluation
## SparseLU and Heartwall

Performance relative to sequential implementation:




(a) SparseLU benchmark



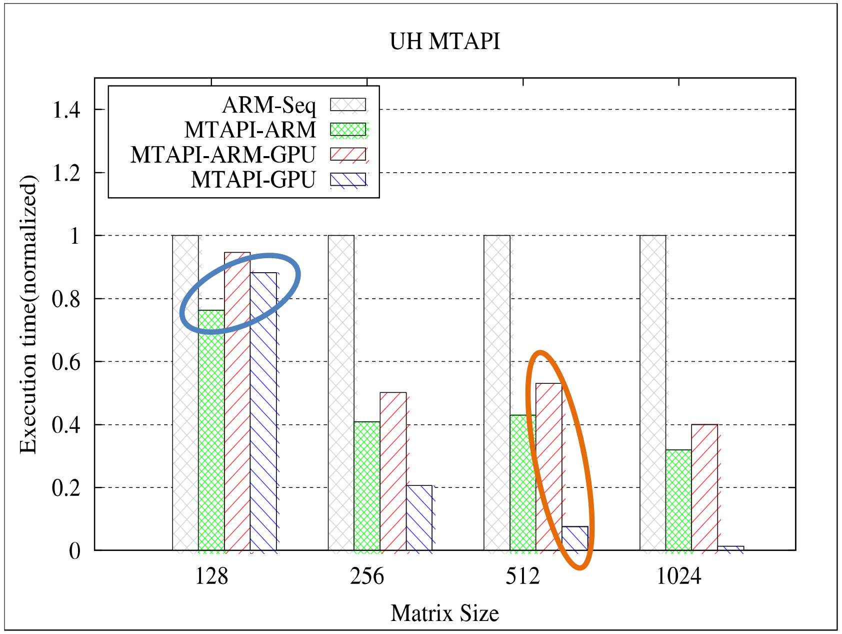

(b) Heartwall benchmark

- MTAPI implementations and OpenMP perform comparably well
- Heartwall benchmark does not scale linearly (memory bound)

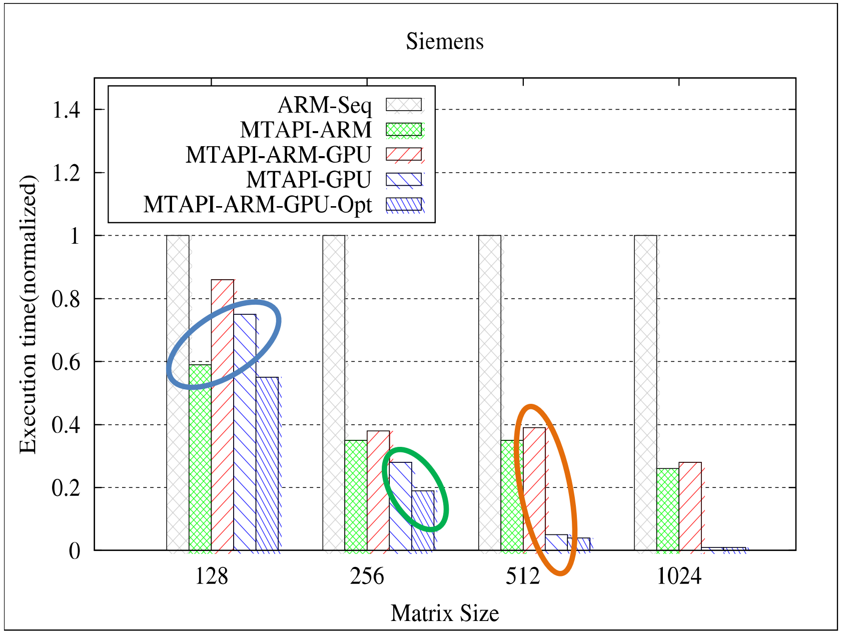
SIEMENS

# Performance Evaluation
## Matrix Multiplication

Normalized execution times for UH-MTAPI and Siemens MTAPI (EMB²):




(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

# Summary and Outlook

- **Existing frameworks** for parallel programming often **not suitable for embedded systems**
- SW development for **heterogeneous systems-on-a-chip** (SoCs) challenging due to **proprietary interfaces / tools**
- MTAPI provides **standard API** for leveraging **task parallelism** on embedded devices with multicore processors
  - designed for homogeneous and heterogeneous systems
  - support for shared and distributed memory
  - can even be used bare metal (w/o OS)
  - may serve as a basis for higher level programming models
- Experimental results show **competitive performance**

- **Improved scheduling algorithms** for heterogeneous and real-time systems
- **Support for further accelerators** such as DSPs and FPGAs