Algorithmic Skeletons for Parallelization of Embedded Real-Time Systems

Alexander Stegmeier, Martin Frieb, Ralf Jahr,
Theo Ungerer
University of Augsburg, Germany

HiRES 2015
(High-performance and Real-time Embedded Systems)

Motivation

- Multicore CPUs in embedded systems
- Challenging to develop parallel code
  - High implementation effort
  - Timing predictability

→ Need for facilitation in scope of HRT systems

Skeletons applicable for implementing parallelism

- Advantages
  - Abstraction simplifies programming
  - Timing predictability asured for each parallel execution

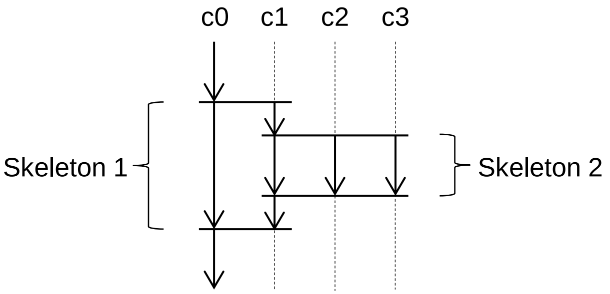
Library of Algorithmic Skeletons for Parallelization

Outline

- Motivation
- Supported Parallelism
- Implementation
- Application of Skeletons
- Evaluation
- Conclusion and Future Work

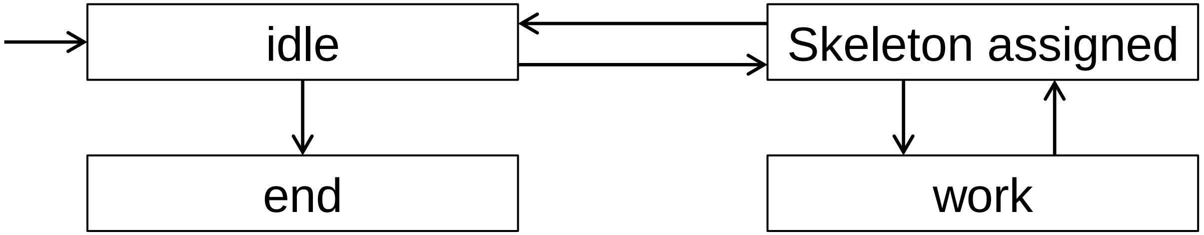
Supported Parallelism

- Structured parallelism
  - Commonly used patterns
  - facilitates time predictability
- Pattern based solutions
  - Task parallelism
  - Data parallelism
  - Pipeline parallelism
- Nested parallelism possible

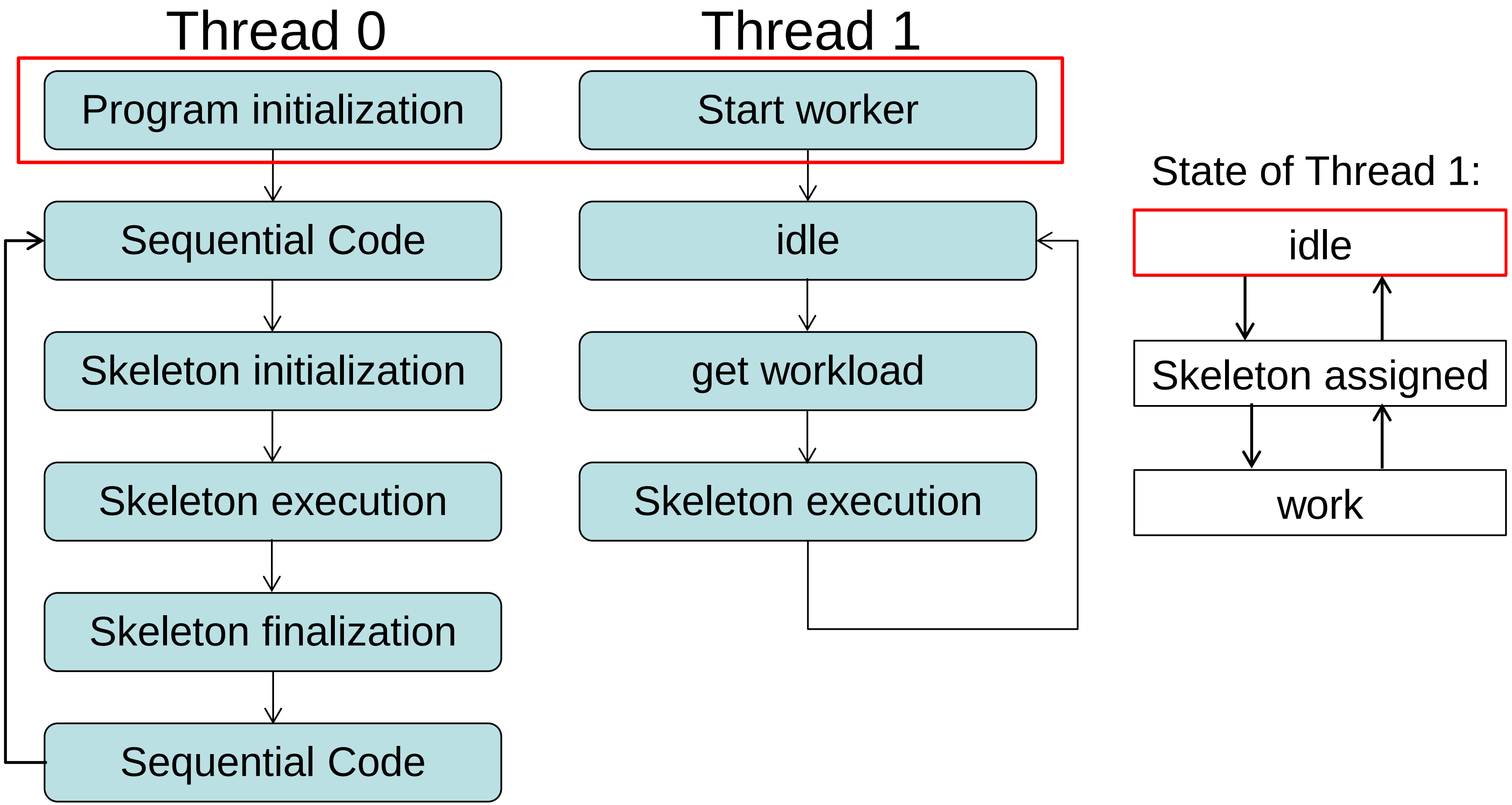


Underlying implementation layer

- **Starting phase:** initialize worker threads
- **Working phase:**
  - Main thread: execute user application
  - Worker threads:



Implementation



Universität Augsburg University

Task Parallelism

Global declarations:

```
//Functions to be called
tas_runnable_t tp_runnables[] = {
        (tas_runnable_t) runnable0,
        (tas_runnable_t) runnable1};

//Pointer to arguments for functions (NULL because not needed)
void * tp_args[4];

//Task Parallelism: list of ...
tas_taskparallel_t task_parallelism = {
        tp_runnables, //... functions to be called
        tp_args,      //... arguments of functions
        2};           //... and the number of functions
```

Task Parallelism

Invocation within function:

```
tas_taskparallel_init(&task_parallelism, THREAD_LIST);
tas_taskparallel_execute(&task_parallelism);
tas_taskparallel_finalize(&task_parallelism);
```

Universität Augsburg University

Data Parallelism

Global declarations:

```
//Definition of data type for argument array
typedef struct {
    int input_data[SIZE];
} dp_args_t;

//Array of arguments for workers
dp_args_t dp_args_array[NUM_WORKERS];

//Pointer to arguments for workers
void * dp_args[NUM_WORKERS];

//Data Parallelism: ...
tas_dataparallel_t data_parallel = {
    (tas_runnable_t) runnable, //...the function to be called
    dp_args,               //...list of arguments of functions
    NUM_WORKERS };         //...and the number of workers
```

Universität Augsburg University

Data Parallelism

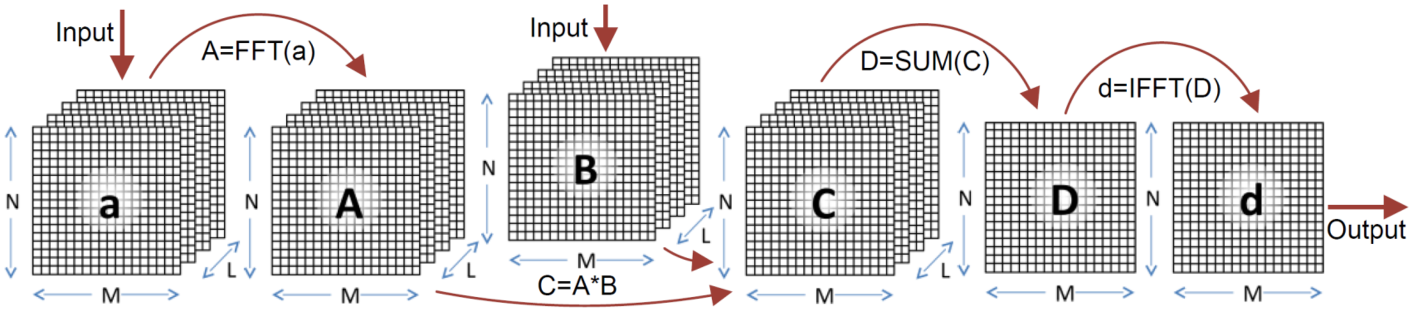
Invocation within function:

```
//Assign argument arrays to pointers
for(i = 0; i < NUM_WORKERS; i++) {
    dp_args[i] = &(dp_args_array[i]);
}

//Invocation
tas_dataparallel_init(&data_parallel,
THREAD_LIST);
tas_dataparallel_execute(&data_parallel);
tas_dataparallel_finalize(&data_parallel);
```

Benchmark application

- Streaming signal processing application:



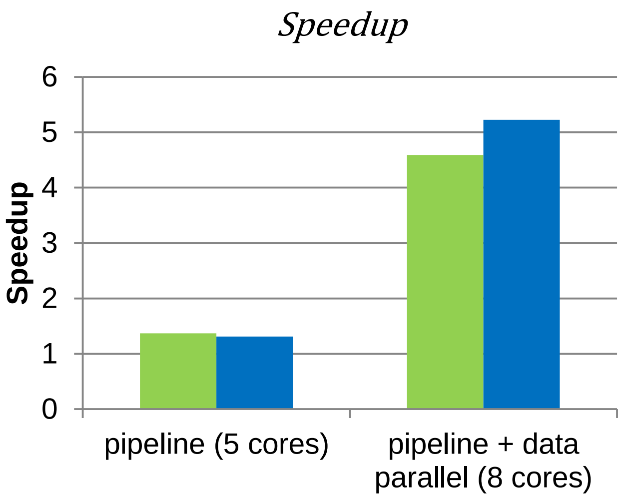
- 2 evaluated input sets:
  - small(16x16x4) / large(32x32x8) size for each matrix set
- 3 versions:
  - Sequential implementation
  - Pipeline parallelism (5 stages)
  - Pipeline parallelism (5 stages) + data parallelism (4 threads)

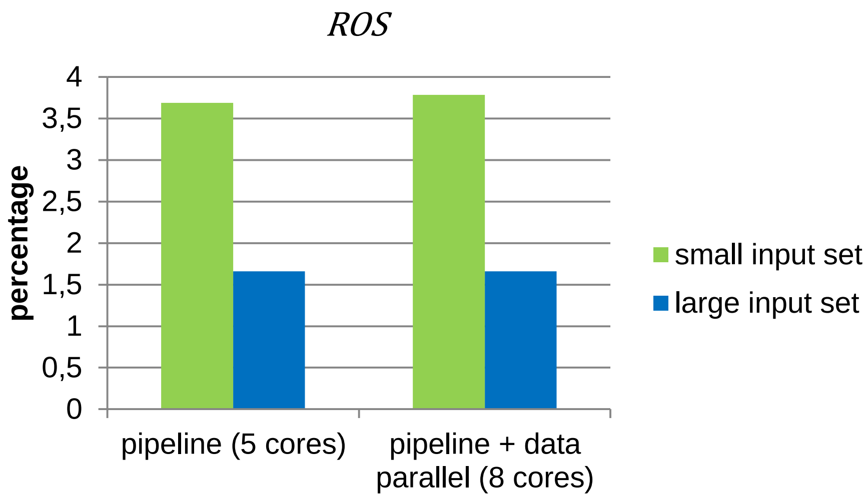
Simulator Setup

- parMERASA multicore with one cluster
- Applied up to 8 cores
- Shared memory (latency 40 cycles)
- Instruction scratchpad
- Data cache with LRU
- Details: see paper/parMERASA project

Results

- $$Speedup = \frac{Execution\ Time\ of\ Sequential\ Version}{Execution\ Time\ of\ Parallel\ Version}$$
- $$ROS^{*} = \frac{Execution\ Time\ of\ Overhead}{Execution\ Time\ of\ Sequential\ Version}$$





Relative execution time **Overhead based on **S**equential*

Conclusion and Future Work

Algorithmic Skeleton library developed:

- Applicable in hard real-time systems
- Facilitates implementation of structured parallelism
- Task Parallelism, Data Parallelism and Pipelinie Parallelism
- Moderate parallelization overhead

Future work:

- Timing-analysis with OTAWA
- Optimizations to tighten WCET estimation

Thank You!