Introduction

In hard real-time systems, it is mandatory to verify the temporal behavior of the application, e.g., the compliance to deadline constraints. In parallel task scheduling, inter- and intra-task parallelism has to be considered in the timing analysis, where inter-task parallelism refers to the co-scheduling of different tasks and intra-task parallelism refers to parallel execution of a single task. In the context of task models for parallel computing, fork/join models [1], synchronous parallel task models, and DAG (directed acyclic graph) based task models [2–7] have been proposed and analyzed with respect to real-time constraints.

In the gang task model, a set of threads is grouped together into a so called gang with the additional constraint that all threads of a gang must be co-scheduled at the same time on available processors. It has been demonstrated that gang-based parallel computing can improve the performance in many cases [8, 9]. Due to its practicability, the gang model is supported by many parallel computing standards, e.g., MPI, OpenMP, Open ACC or GPU computing.

In the stationary gang scheduling paradigm, each gang task is statically assigned to a set of processors, in which the cardinality of the set is equal to the gang size of the task. After this assignment is done, a gang task is only eligible to be executed on stationary processors assigned to it.

Thesis

1. In this thesis, the student should implement fixed-priority stationary GANG scheduling in LITMUS-RT1 using the C programming language.

2. Moreover the student should devise experiments to evaluate the scheduling overheads and run-time variations using the provided tracing tools in LITMUS-RT.

3. (optional) The student could assess if it is possible to integrate memory bandwidth controllers, e.g., memguard2 into stationary GANG scheduling to limit memory contention of co-scheduled tasks.

Ideally, this thesis should either demonstrate the benefits of the stationary constraint with respect to memory contention, scheduling overheads and run-time variations or hint to the performance problems of this approach. If you are interested, please do not hesitate to contact me for further information and literature.

Required Skill

- Basic knowledge with Linux and systems programming
- Comfortable with C and Python
- Interested in Real-Time scheduling

Acquired Skills after the work

- Knowledge of parallel real-time scheduling algorithms
- Knowledge of real-time operating systems
- Design, Analysis, and Implementation of system software for real-time systems

References


