Scheduling in MyThOS

Whitepaper

Randolf Rotta, `rottaran@b-tu.de`
Vladimir Nikolov, `vladimir.nikolov@uni-ulm.de`
Lutz Schubert, `lutz.schubert@uni-ulm.de`

19th April 2016

Contents

1 Overview

Scheduling in MyThOS comprises of two components: Executing tasks on each processing unit and management of available processing units. The first component is responsible for executing tasks assigned to a particular hardware thread. The latter component tracks the occupation of hardware threads and assigns them to applications upon request.

2 Local Scheduling

Tasks, that are assigned to a hardware thread for execution, are placed into differently prioritized queues. They are subsequently processed by the hardware thread according to their priorities. Tasks of the same priority are processed in a FIFO manner. After the execution of a task is started, it is run to completion without preemption by another task. This allows MyThOS to employ a thin scheduling layer, that only causes low overhead. Also the overhead induced by task switching in classical preemptive systems, e.g. by cache and TLB invalidations, does not occur in MyThOS. However, the simplicity and efficiency of this scheduling architecture has drawbacks as well. When a task is placed in a queue, it may spend an arbitrary time in the queue, waiting for tasks with earlier arrival or higher priority to finish. Therefore no guarantee for the time this task finishes its computations or even, when it is first executed, can be given.

For some applications it may be necessary to provide a possibility to set a maximum response time for each task and ensure the completion of the task before this deadline and therefore provide a responsive system. This can be achieved by integrating a notion of runtime and deadlines into MyThOS, utilizing methods known from real-time systems. To provide the highest resource usage possible, while delivering low response times, additionally suitable partitioning and scheduling algorithms need to be employed. Partitioning is required to distribute tasks among the available processing units, while a resource-aware scheduling algorithm is responsible for providing the required performance for each task on a single processing unit. To provide each task with a requested performance, it may be necessary to inject waiting times between the execution of two tasks to avoid the processing unit being blocked for an important task. However, it has to be ensured, that the additional overhead caused by a more sophisticated scheduling algorithm does not impact the responsiveness of the system.

3 HWTM Hardware-Thread Manager

The *Hardware-Thread Management* (HWTM) component is responsible for keeping track of the current occupation status of the available processing units within a system. Since in our current evaluation platform the smallest addressable computational unit is a hardware thread (HWT), the HWTM provides a resource management facility on that level.

Traditionally, existing processes are stored and managed by an operating system within a single central data structure, e.g. a list of process control blocks (PCBs). These processes are then distributed over the set of available computation units according to the decisions of a scheduler. Regarding a HPC scenario where several independent processes residing on different computation units may issue requests for new child processes or tasks in parallel, a central management structure easily becomes a bottleneck.

The idea behind the HWTM is to distribute the responsibility of finding free units, creating processes, allocating them on these units and keeping track of their status. This is achieved by a replication strategy of several HWTM instances, each running on a different core and responsible for a different set of available hardware threads. On this way, the creation and management overhead of threads is parallelized.

For each new process one of these HWTM instances is automatically assigned as its private manager and stored within a core local variable. From application point of view, a process issues a request for new threads to its private HWTM. It does not have to care about their allocation or occupation, i.e. this is the job of the HWTM. When the request is processed, the requested number of threads, if possible, is handed over to the application. Each of these processes resides on a different computation unit, i.e. on a different hardware thread. The application then is responsible to issue computations or tasks to these newly created and runnable threads.

An interesting situation occurs when a request can not be served (fully) by a responsible HWTM, because its private set of processing units is already depleted. In that case the remainder of a partially served request is delegated automatically to another HWTM unit. For this, HWTMs have to synchronize their current occupation status on each operation.

For the state of the art, a partial sub-request is always delegated to the instance with the most available processing units, but however, other strategies like “best fit” are also possible. The HWTM instances automatically keep track of their delegation status. When threads terminate or are explicitly released by the application, the according resources are also automatically released on the HWTM instance level.

In order to establish an optimal configuration of HWTM instances for a particular platform a simulation model was provided, which takes into account several factors like communication costs between HWTMs, local queuing of requests, processing costs of each request etc. Given some particular values for the factors which are known from preliminary application executions the best distribution and amount of HWTMs can be determined. With this decentralized resource management architecture the response time of a thread request can be shortened enormously during runtime.

Bibliography