Jan Jürjens, Kurt Schneider (Hrsg.): Software Engineering 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 83

Increasing the Throughput of Pipe-and-Filter Architectures by Integrating the Task Farm Parallelization Pattern

Christian Wulf[1] und Wilhelm Hasselbring[2]

Abstract: The Pipe-and-Filter style represents a well-known family of component-based architectures. By executing each filter on a dedicated processing unit, it is also possible to leverage contemporary distributed systems and multi-core systems for a high throughput. However, this simple parallelization approach is not very effective when (1) the workload is uneven distributed over all filters and when (2) the number of available processing units exceeds the number of filters.

In this paper, we explain how we utilize the task farm parallelization pattern in order to increase the throughput of Pipe-and-Filter architectures. Furthermore, we describe an associated modular self-adaptive mechanism which enables the automatic resource-efficient reaction on unevenly distributed workload. Finally, we refer to an extensive experimental evaluation of our self-adaptive task farm performed by us. The results show that our task farm (1) increases the overall throughput and (2) scales well according to the current workload.

Keywords: Pipe-and-Filter, TeeTime, parallelization, task farm pattern, self-adaptation

With the use and adoption of big data, the Pipe-and-Filter (P&F) architectural style gained an increased popularity both in industry and in research. Recent research [Al13, GTA06, Su10] addresses the problem of how to leverage and to optimize contemporary multi-core systems for a high throughput.

Our approach [WWH16b] provides a solution for this problem by automatically duplicating a given stage[3], preferably the slowest one, and executing their instances in parallel. A *distributer* stage ensures that incoming data elements are evenly distributed among the instances. Analogously, a *merger* stage collects the outgoing data elements from the instances and passes them on to the next stage. For the sake of reusability, we provide a composite stage which implements this behavior for a given stage in a transparent way. We call this stage the *Task Farm Stage* (TFS) since it effectively implements the Task Farm parallelization pattern [Al13].

Moreover, we provide an associated self-adaptation manager (SAM) which automatically adapts the number of stage instances at runtime based on the current throughput of the TFS. A `Monitoring` component monitors the throughput of the pipes which connect the distributor and the stage instances with each other. An `Analysis` component analyzes the measurements of the `Monitoring` component and calculates how much the throughput of the TFS has changed since the last few measurements. A `Reconfiguration` component

[1] Kiel University, Software Engineering Group, 24098 Kiel, chw@informatik.uni-kiel.de
[2] Kiel University, Software Engineering Group, 24098 Kiel, wha@informatik.uni-kiel.de
[3] We use the term *stage* as generalization for data sources, filters, and data sinks, as categorized by [Bu96].

takes the result of the `Analysis` component and decides whether the TFS should add or remove a stage instance. The cycle is then completed and starts again with the `Monitoring` component after a user-defined delay (our default is 50 ms).

We performed an extensive experimental evaluation of our TFS and the associated SAM on four different multi-core systems. We employed a CPU-intensive, an I/O-intensive, and a hybrid scenario in order to show (1) that our TFS is able to increase the throughput of various P&F architectures and (2) that our SAM scales well. We achieved speedups ranging from 1.5 to 7.3 for our scenarios. For more details, we refer to the full paper [WWH16b] and our replication package [WWH16a].

For our evaluation, we use the P&F framework TeeTime [Th, WEH14]. It allows to model and to execute arbitrary P&F architectures. For example, it supports feedback loops, multiple input/output ports per stage, and the composition of several stages to a single one. Stages modeled with TeeTime can be stateless or stateful and can reuse other stages. Ports are typed and allow to interconnect stages with synchronized and unsynchronized pipes. However, TeeTime's major strengths are its support for a concurrent execution and its ability to provide a high throughput. It utilizes contemporary multi-core systems by executing stages of a P&F architecture in parallel. In particular, threads can be assigned to stages in an arbitrary manner. Hence, we chose this framework for our evaluation.

References

[Al13] Aldinucci, Marco; Danelutto, Marco; Kilpatrick, Peter; Torquati, Massimo: FastFlow: high-level and efficient streaming on multi-core. In: Programming Multi-core and Many-core Computing Systems, Parallel and Distributed Computing, chapter 13. Wiley, 2013.

[Bu96] Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter; Stal, Michael: Pattern-oriented Software Architecture: A System of Patterns. Wiley, 1996.

[GTA06] Gordon, Michael I.; Thies, William; Amarasinghe, Saman: Exploiting Coarse-grained Task, Data, and Pipeline Parallelism in Stream Programs. In: Proc. of the 12th International Conference on ASPLOS. 2006.

[Su10] Suleman, M. Aater; Qureshi, Moinuddin K.; Khubaib; Patt, Yale N.: Feedback-directed Pipeline Parallelism. In: Proc. of the Int. Conf. on PACT. 2010.

[Th] The TeeTime project. `https://teetime-framework.github.io`.

[WEH14] Wulf, Christian; Ehmke, Nils Christian; Hasselbring, Wilhelm: Toward a Generic and Concurrency-Aware Pipes & Filters Framework. In: Symposium on Software Performance: Joint Descartes/Kieker/Palladio Days. November 2014.

[WWH16a] Wulf, Christian; Wiechmann, Christian Claus; Hasselbring, Wilhelm: Data for: Increasing the Throughput of Pipe-and-Filter Architectures by Integrating the Task Farm Parallelization Pattern. doi: 10.5281/zenodo.46776, March 2016.

[WWH16b] Wulf, Christian; Wiechmann, Christian Claus; Hasselbring, Wilhelm: Increasing the Throughput of Pipe-and-Filter Architectures by Integrating the Task Farm Parallelization Pattern. In: Proceedings of the 19th International Symposium on CBSE. 2016.