Parallelization Techniques for Semantic Web Reasoning Applications

Alexey Cheptsov, Matthias Assel

[1] HLRS - High Performance Computing Center Stuttgart, University of Stuttgart, Nobelstrasse 19, 70569 Stuttgart, Germany
{cheptsov, assel}@hlrs.de

Abstract. Performance is the most critical aspect towards achieving high scalability of Semantic Web reasoning applications, and considerably limits the application areas of them. There is still a deep mismatch between the requirements for reasoning on a Web scale and performance of the existing reasoning engines. The performance limitation can be considerably reduced by utilizing such large-scale e-Infrastructures as LarKC - the Large Knowledge Collider - an experimental platform for massive distributed incomplete reasoning, which offers several innovative approaches removing the scalability barriers, in particularly, by enabling transparent access to HPC systems. Efficient utilization of such resources is facilitated by means of parallelization being the major element for accomplishing performance and scalability of semantic applications. Here we discuss application of some emerging parallelization strategies and show the benefits obtained by using such systems as LarKC.

Keywords: Semantic Web Reasoning, LarKC, parallelization, multi-threading, message-passing

1 Introduction

Current Semantic Web reasoning systems do not scale to the requirements of the rapidly increasing amount of data, such as those coming from millions of sensors and mobile devices or the terabytes of scientific data produced by automated experimentation.

The latest attempts to overcome the above-mentioned limitations resulted in infrastructures for large-scale semantic reasoning, such as one set up by LarKC (the Large Knowledge Collider [1]) which focuses on reasoning over billions of structured data in heterogeneous data sets. Along with a number of original solutions for obtaining Web scale by semantic applications, LarKC offers services for transparently accessing diverse computing architectures, including multi-core (many-core) multi-processor, and cluster-based computer architectures as well as dedicated high-performance computers.

Parallelization enables simultaneous execution of independent computational operations and thus resolves the conflicts occurring between the concurrent operations while performing computation. Given the large problem sizes that are addressed by LarKC, and considering the benefits of parallelization, it seems natural to explore use of the main parallelization strategies for semantic applications, too. Here we discuss some major parallelization techniques for providing parallelism on task-, instruction-, and data-level, applied for LarKC's pilot applications. However, the investigated approaches and techniques are quite generic and can be potentially applied for any other Semantic Web engine.

2 Parallelization Patterns

There are several parallelization techniques, which have proven their usability for a wide range of optimization tasks and might be beneficial for semantic applications. They can be roughly classified according to the level at which the parallelism takes place (Fig. 1):

1) between loosely-coupled components (workflow level) – implementation of parallelism by running multiple instances of the same plug-in simultaneously
    - task-level parallelism
        - *workflow branching*
2) within a separate component ("plug-in" level) – implementation of parallelism in the concurrent regions of the component's algorithms
    - instruction-level parallelism
        - shared-memory systems: *multi-threading*
        - distributed-memory systems: *message-passing*
    - data and instruction-level parallelism
        - *MapReduce data processing*

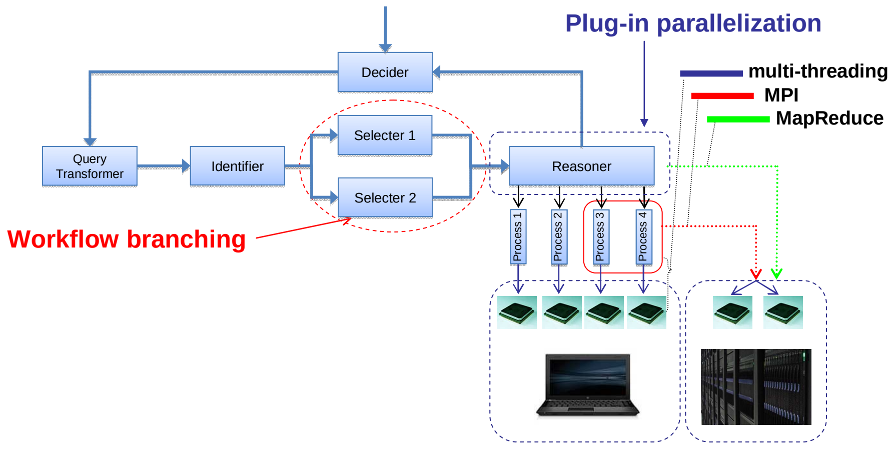

Fig. 1. For the semantic applications, which are described through complex workflows, parallelization can be applied on different levels: workflows (task- and data-level parallelism), or single components/plug-ins (data- and instruction-level parallelism).

3 Main Instruction-Level Parallelization Techniques

The techniques presented in Section 2 differ by complexity of their implementation and obtained performance impact. In this section we discuss only the approaches, which allow obtaining considerable performance impact with a minimum of implementation efforts for sequential code. In particular, we consider multi-threading and message-passing. The achieved performance impact is discussed as well.

3.1 Multi-threading

Most of today's CPUs are equipped with multiple cores. Unfortunately, many applications are still using only one of them for their processing (i.e., applications are still sequentially programmed) instead of distributing particular tasks to different processor cores concurrently. In order to make use of the capabilities provided by modern CPU architectures, applications must align their tasks according to the number of available cores.

Implementation of multi-threading for a sequential code is to large extent trivial and does not require much development efforts. For evaluation purposes, we implemented multi-threading support for the Urban Computing application of LarKC [2]. Realization of multi-threading for the most time consuming component of the investigated workflow allowed us to obtain a considerable performance speed-up (Table 1).

Table 1. Performance characteristics after applying multi-threading

Tested realization	Intel @ 1.8 GHz, 2 cores		Xeon @ 2.8 GHz, 8 cores	
	Time, ms	% of total execution	Time, ms	% of total execution
Single thread	4400	80	3400	74
Multiple threads	1100	37	900	36
Speed-up, times	**3.8**		**3.7**	

3.2 Message-Passing

The Message-Passing Interface (MPI) is the most widely used parallel programming paradigm for highly-scalable parallel applications. MPI enables sharing the application workload over various nodes of a parallel system (both shared and distributed memory architectures are supported). The synchronization between the nodes is achieved by means of the messages passed among the involved processes through the network interconnect. Implementations of MPI in Java (such as MPIJava or MPJ-Express) have enabled use of MPI also for Java applications. MPI is highly beneficial for computing-intensive applications, whereby scalability within a shared-memory space is not sufficient for obtaining the necessary performance.

For evaluation purposes, we implemented message-passing for the “Airhead” library from the S-Space package[1]. The parallelization technique was evaluated for the Linked Life Data subset used by University of Sheffield within the LarKC project. The obtained performance characteristics, collected in Table 2, prove great benefit of distributed-memory parallelisation not only for the investigated application, but also for similar ones coming from other areas of the Semantic Web.

Table 2. Performance characteristics after applying message-passing

Number of computing nodes	Intel @ 1.8 GHz, 2 cores		Xeon @ 2.8 GHz, 8 cores	
	Time, s.	Speed-up (to 1 CPU case)	Time, s.	Speed-up (to 1 CPU case)
1	750	1	57	1
2	-	-	20	**2.85**
4	-	-	10	**5.7**
8	-	-	5	**11.4**

4 Conclusions

In our tests we investigated the impact of the main instruction-level parallelization strategies, namely multi-threading and message-passing, on performance of two typical Semantic Web use cases. The first application was taken from the urban computing use case, where parallelization facilitates meeting real-time requirements. Whereas message-passing was not very useful for this application due to real-time performance requirements, applying multi-threading allowed the application to greatly benefit from the multi-core CPU architecture. The second application - random indexing - was much more complex as the first one, and made great benefit of message-passing that leveraged a cluster of shared-memory nodes for the application.
Our future investigations will concentrate on further approaches presented here (such as MapReduce [3]) as well as hybrid algorithms combining them (e.g. multi-threading inside a shared-memory node combined with message-passing among nodes).

References

1. Fensel, D., van Harmelen, F., Andersson, B., Brennan, P., Cunningham, H., Della Valle, E., et al. Towards LarKC: A Platform for Web-Scale Reasoning, In: Proceedings of the 2008 IEEE international Conference on Semantic Computing ICSC, pp. 524--529, IEEE Computer Society (2008)
2. Della Valle, E., Celino, I., Dell'Aglio, D. The Experience of Realizing a Semantic Web Urban Computing Application, Transactions in GIS 14,2 (2010)
3. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F. Scalable Distributed Reasoning Using MapReduce. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) The Semantic Web - ISWC 2009, LNCS, vol. 5823, pp. 634--649, Springer (2009)

[1] http://code.google.com/p/airhead-research/