CHAPTER 40

Other Experiences with Computational Performance Improvements

Kai Nagel

MATSim has always had the simulation of large regions as its goal, and as such was always interested in high computational performance. The team had, when it started with the Java-based MATSim (cf. 46.2.1.4), considerable experience in parallel computing (Nagel and Schleicher, 1994; Rickert and Nagel, 2001; Nagel and Rickert, 2001; Cetin et al., 2003) as well as with more general message-based approaches (Gloor and Nagel, 2005) that resemble today's Protocol Buffers (Google Developers, 2015). However, the move to Java (see Section 46.2.1.4), a decision for faster conceptual progress and reduced maintenance effort, also had the consequence that the MPI (Message Passing Interface) approach to parallel computing could no longer be used and was thus given up. See Section 46.2.1.4 for details.

The behavioral modules of MATSim, such as route (Section 4.5.1.2) or destination (Chapter 27) innovation, are conceptually straightforward to parallelize by multi-threading, and that was implemented in MATSim from early on (Balmer et al., 2009b, see Section 4.2.3 how to use this). The remaining challenge then is to parallelize the mobsim, in which the parallel threads need to interact closely. For example, assume that we compute 24 hours of traffic in 120 seconds of computing time (cf. Table 40.1). With the 1 second time steps used in the QSim this means 720 update rounds per second, and thus 720 inter-thread interactions per second.

An attempt to use the CUDA (Compute Unified Device Architecture, a parallel computing platform and API by NVIDIA) for the C language (Strippgen and Nagel, 2009b,a; Strippgen, 2009) ran into the same problems as the earlier parallel DEQSim also written in C/C++ (Charypar et al., 2007a): The time necessary to transmit the necessary information back and forth between the Java-based MATSim and the C/C++-based external package used up all the performance gains. In consequence, the DEQSim was re-implemented as the so-called JDEQSim in Java (Waraich et al., 2015, also see Section 4.3.2). Before parallelizing the JDEQSim, however, it was decided to first accelerate the processing of the events since that was identified as the main bottleneck. Section 4.2.3 describes

How to cite this book chapter:

Nagel, K. 2016. Other Experiences with Computational Performance Improvements. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 267–268. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.40. License: CC-BY 4.0

how to use parallel events handling. The parallel version of the JDEQSim (Waraich et al., 2015) never made it into the MATSim main repository.

At the same time, the standard QSim was improved by other people, for example by keeping track of active links and not doing any computation on links without activity. Ch. Dobler made the QSim multi-threaded. He reported (Dobler, 2013, Chapter 5) close-to-linear speed-ups with large scenarios, but only small—if any—performance gains with small scenarios. That is, multi-threading helped greatly with overall computing times for large scenarios on large shared-memory computers, but little with with quick turn-around during experimentation. More recent hardware seems to have improved the situation also for small scenarios (Table 40.1) so that it was eventually decided to remove the single-threaded variant of the QSim and concentrate development on the multi-threaded variant only.

Lämmel et al. (2016) experiments with using Protocol Buffers (Protocol Buffers web page, accessed 2015) in order to couple two different mobsims.

The PSim (Chapter 39) addresses the problem from a different angle: Rather than accelerating the QSim itself, it attempts to make use of the fact that (1) adding or removing a small number of synthetic travelers does not change congestion patterns very much and thus alternative plans can be evaluated in parallel, and (2) the congestion patterns generated by the mobsim do not vary that much from one iteration to the next so that the mobsim does not have to be re-run every time after some synthetic travelers have moved to different alternatives.

Märki et al. (2014) and Dobler (2013) point out that the number of iterations to reach equilibrium can be reduced when the synthetic travelers perform within-day re-routing – this points into the same direction as Lu et al. (2015) who claim that equilibrium iterations will not be necessary at all with well-calibrated behavioral models and a realistic starting point.

MATSim needs, at least for large scenarios, a large amount of RAM. One could say that within the usual space-time tradeoff in computation,[1] in most situations MATSim rather consumes more memory in order to reduce the computation time. Memory-saving compressed routes are available as an option in the `<plans>` section of the config file. MATSim can be seen as an object-oriented database in RAM; attempts to provide a backing by a relational database were not successful when they were tried (Raney and Nagel, 2004, 2006, ; also see Section 46.2.1.3).

To summarize: (1) The behavioral parts of MATSim parallize easily; the main challenge is the mobsim. (2) The main challenge with parallelizing the mobsim is not so much the pure performance improvement, but to achieve this in a way that it remains integrated with the MATSim main development, and at little or no additional maintenance effort.

Computer	population size	1 thread	4 threads	6 threads	8 threads
laptop 2010	1% = 23 500	432 sec	(X)	(X)	(X)
laptop 2014	1% = 23 500	110 sec		57 sec	55 sec
laptop 2014	10% = 235 000			200 sec	

"(X)" means that the laptop was no longer useful for secondary tasks.

Table 40.1: Computing times of the mobsim for the Gauteng scenario (see Chapter 69) with 523 000 links for different computers, different population sizes, and different numbers of threads. "laptop 2010" refers to a high end Mac Pro laptop from 2010, "laptop 2014" refers to a high end Mac Pro laptop from 2014. We can see a speed increase close to a factor of four from 2010 to 2014, and then in 2014 an additional factor of two with multi-threading. These results were shown at several seminars, but never published elsewhere.

[1] See `https://en.wikipedia.org/wiki/Space-time_tradeoff`.