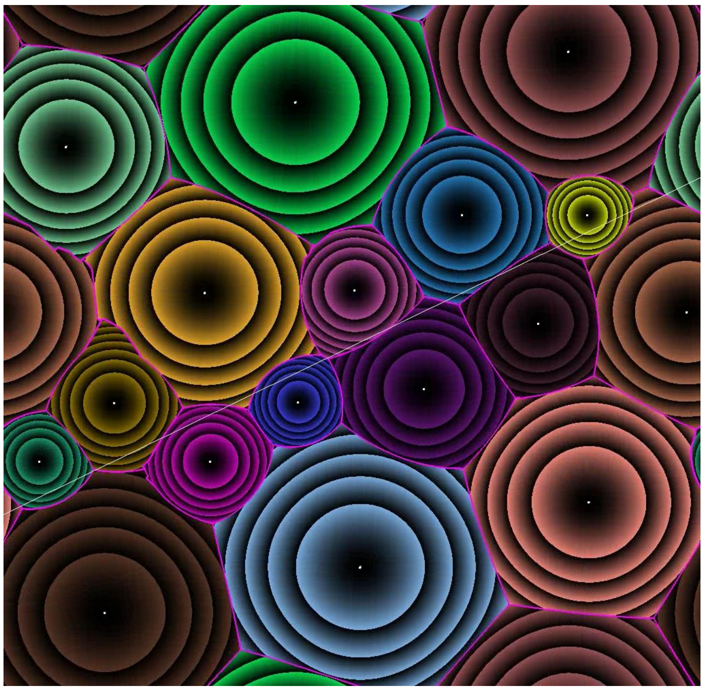
PISim
A parallelization framework for interaction simulations

Author(s):
Charypar, David

Publication Date:
2014-04

Permanent Link:
https://doi.org/10.3929/ethz-b-000082276 →



PISim: A Parallelization Framework for Interaction Simulations

David Charypar

Working paper 985

Institute for Transport Planning and Systems April 2014

Working paper Institute for Transport Planning and Systems 985

PISim: A Parallelization Framework for Interaction Simulations

David Charypar,
IVT
ETH Zürich

CH-8093 Zürich
phone: +41-44-633 35 62
fax: +41-44-633 10 57
dcharypar@gmail.com

April 2014

Abstract

This paper presents PISim, a **P**arallelization framework for **I**nteraction **Sim**ulations. PISim can be used to simulate models of short ranged object interactios in space on high-performance computers. By using PISim, users can avoid the difficulties and complexity usually involved in porting sequential sourcecode to parallel machines. The framework achieves this by combining severals fast and simple elements: First, it uses an efficient representation of space (a helical array) which exploits temporal continuity, second, it uses a straightforward way of communication between processors by means of message passing, third, it arranges simulation domains in cell clusters that are treated as individual objects, fourth, it monitors the computation times of each cluster continuously and uses model based control to grow or shrink the clusters as needed to balance the load between processors, and fifth, it encapsulates and hides all this complexity from the user by using programming templates to provide a slim interface for end users. This small interface makes it possible to adopt the framework easily to new problems. It quickens development cyles and therby enables users not familiar with parallel programming techniques to adapt rapid prototyping paradigms.

Keywords

Parallelization, Software Framework, Simulation

Preferred citation style

Charypar, D. (2014) **PISim**: A **P**arallelization Framework for **I**nteraction **Sim**ulations, *Working paper Institute for Transport Planning and Systems*, **985**, Institute for Transport Planning and Systems (IVT), ETH Zurich, Zurich.

1 Introduction

The framework presented in this paper was developed in the context of transport simulation problems. In this field there is a noticeable trend towards more sophisticated models, on the one hand, and higher resolutions, on the other. Mesoscopic and microscopic agent-based models represent typical examples of this development. The simulation framework presented here was developed with the goal to simplify the development of exactly this type of simulations.

One important requirement for the development towards higher resolutions is that the performance of the computing hardware increases simultaneously, as more resolution and higher complexity naturally require more computational power. In the years from about 2004 (see Wikipedia, 2014b) to 2014, computing performance has increased to a large amount due to multi-core architectures and ever larger computer clusters. TOP500 (2014)

One of the many challenges when trying to use multi-core CPUs and large scale computer clusters is the development of software that is able to fully exploit the available computing power. It is not only about writing efficient code and smart data structures but also about finding mechanisms to distribute the load between processors and designing the mode of collaboration among them.

It is needles to say that parallel programs (i.e. programs written to run in parallel on multiple CPUs) can be very complex, error prone to write and maintain, and even difficult to understand in the first place. Doing all this is a non-trivial task for modelers, engineers and computer scientists alike.

Transport planning — like many other research topics — is an interdisciplinary field with scientist from psychology and geography to civil engineering and physics working together to develop new models and methods. There is a vast variety of problems in transport planning and attached fields that can be approached using computationally intense methods and are suitable for parallel execution: Microscopic traffic flow simulations, activity planning, urban and land-use simulations, or activity planning are just a few examples. In many of these area, computational methods have become a very useful tool albeit the difficulties to develop parallel programs. However, one cannot expect each researcher from all the various of disciplines to be proficient in developing specialized parallel simulation code.

When one looks at examples of computationally intense simulations in transport planning (and probably other field as well), there are some common problems which often arise when it comes to parallelizing these methods. Often the life cycle of a piece of simulation software is as follows: First, the simulation program is developed using a small test scenario on a desktop computer or a laptop. During this phase finding suitable and consistent input data and developing good models for optimal output accuracy are the main focus. Then, in a second step, the software is applied to larger problems or more data. It is usually at this point when problems with computation complexity become apparent, and the running times of the code might very well increase beyond usability. Here, the idea of using more computing power to solve the problem (the brute force approach) comes up. Consequently, the developer of the existing software tries to adapt the software by adding a few parallel concepts to it. Unfortunately, it turns out that the resulting code does not make very good use of the computing power available.

The analysis of the situation shows quite often that the problem with parallelization efficiency lies in the layout of the source code used. The data structures might be optimized for convenience of use instead of being well suited for quick parallel execution. One way around this problem is to rewrite the code from scratch with the assumed data-flow in mind. This data-flow can then be translated into a set of methods used for data exchange between individual processors. This approach is known as explicit parallelization (see Wikipedia, 2014a) and often involves some sort of message passing between processors. While this approach can be fast it is at the same time a very complex and error prone approach to parallelization requiring a lot of expertise.

Fortunately, it turns out that for parallelizing certain categories of simulation similar structures can be used. As described later in this paper there are some simple restrictions that need to be obeyed to be able to use many CPUs efficiently. The idea of the described framework is to construct those structures once, centrally and then provide an interface to the framework that the user (the application programmer) can use to tell the framework what the simulated entities are and on how many processors these should be executed.

For the application programmer this framework has a range of benefits:

- He/she does not have to write the same complex parallelization code him-/herself over and over again.
- The same is true for the communication routines, like selecting and serializing data and sending messages.
- Separating the parallelization part of the software from the behavioral part of the modules improves program structure
- The resulting program is cleaner and easier to follow.
- An important side effect of this is better maintainability and ease of documentation.
- The code can make better use of computing power available.
- Consequently, results are generated more quickly and/or larger/more complex problems can be addressed.

2 Podcast chapters

The remainder of this paper has been created as a podcast available through youtube:

- PISim's sort data structure is explained in `https://www.youtube.com/watch?v=3ZdqYmkDnZk`
- The parallelization procedure is depicted in `https://www.youtube.com/watch?v=6cxhSMpDA_g`
- How the load is balanced between processors is described in `https://www.youtube.com/watch?v=ti7wOzbGq94`

3 References

TOP500 (2014) Development over time, webpage, April 2014, `http://www.top500.org/statistics/overtime`.

Wikipedia (2014a) Automatic parallelization, webpage, April 2014, `http://en.wikipedia.org/w/index.php?title=Parallel_computing&oldid=600172964#Automatic_parallelization`.

Wikipedia (2014b) Power5 microprocessor, webpage, April 2014, `http://en.wikipedia.org/w/index.php?title=POWER5&oldid=600650542`.