Andrew Andersen
Dr. Munehiro Fukuda
CSS499 Winter 2016
March 18, 2016

Term Paper: Parallelization of UrbanSim with the MASS Library

Introduction

For my project in the Distributed Systems Laboratory this quarter, I attempted to apply the MASS (Multi-Agent Spatial Simulation) library to parallelize a computationally intensive urban simulation software program across computing clusters. My advisor, University of Washington-Bothell Computer and Software Systems Chair Dr. Munehiro Fukuda, and I ultimately decided to suspend the project, but I learned a great deal from from the process. I write this term paper in hopes of not only reflecting on my journey, but to possibly provide an entry point for other students who may venture down the same path, whether with this particular piece of software or with a similar one. With that in mind, I will provide an overview of the software system, explain the challenges that led us to the decision to suspend the project, and provide some insight into what I learned from the experience.

UrbanSim Overview

UrbanSim was described by its creator, Paul Waddell as an urban simulation model, designed to address modern pressures faced by urban planners. (Waddell 2002) Waddell and others initially built UrbanSim in the Java programming language with emphasis on the interaction between transportation, environment, and urban development. (Noth 2003) The modern iteration of UrbanSim, written in Python, is comprised of a series of related models, which interact with large data sets to provide predictive analysis about potential growth trends for a particular region. Urban planners can change scenario inputs (e.g. zoning, development incentives, etc.) in order to see the impact each scenario might have on how the urban area will develop. At its essence, UrbanSim is a highly complex rendition of the formerly popular SimCity video game.

UrbanSim has components of both an independent software program and a library. It includes a proprietary model controller library called Ocra, which is always called an runtime to orchestrate model interactions. Orca reads data and configuration files and stores them in tables. Software developers then write models based on their specific needs, typically with each model instantiating a class and calling methods from UrbanSim model packages. The Urban Data Science Toolkit (UDST) team, who currently maintains UrbanSim as an open-source project on GitHub, lists seven models as standard for a robust implementation of the software. (UDST) Those models are broken into three subcategories as follows:

Residential

- **Pricing Model:** Hedonic regression is used to determine the prices of residential units.
- **Location Choice Model:** Some households relocate to different units.
- **Transition Model:** New households are generated and added to the model.

Non-Residential

- **Pricing Model:** Hedonic regression is used to determine the prices of residential units.
- **Location Choice Model:** Some households relocate to different units.
- **Transition Model:** New households are generated and added to the model.

Developer

- **Development Choice:** Typically a implemented as a pro-forma (projected income vs. cost) model.

The similarities between the residential and non-residential models are not coincidental. In the example provided by UDST in their repository, each respective pair of residential and non-residential models instantiate the same objects and make calls to the same methods from the same classes from the UrbanSim library.

Identifying Candidate Methods for Parallelization

Dr. Fukuda asked me to identify candidate methods or functions for parallelization. Candidate methods should have the following characteristics:

1. Computationally intensive
2. Conventional data structures providing clear input and output types
3. Relatively large amounts of data being processed

We also wanted to ensure any methods we chose came directly from the UrbanSim codebase, rather than from the code base for the provided example so that other developers could take advantage of the parallelized software.

Andrew Andersen
Dr. Munehiro Fukuda
CSS499 Winter 2016
March 18, 2016

Based on prior input from the Pacific Northwest National Laboratories and Puget Sound Regional Council, UrbanSim seemed like it would provide ample opportunities to identify computationally intensive functions. As recently as 2011, researchers in Austin, Texas, reported runtimes of 30 minutes per iteration (one year) of UrbanSim on a 2.66 GHz dual core processor with 3.5 GB of RAM. (Kakaraparthi 2011) However, a single iteration using the sample provided by UDST took less than 10 seconds on my personal computer: a quad core Intel i7 at 2.00 GHz with 8 GB of RAM.

I was initially discouraged by the relatively quick processing times of the program, as the perceived opportunity to make gains from parallelization seemed limited. However, the provided sample only covered San Francisco proper: encompassing approximately 350,000 households, 150,000 buildings, and 225,000 jobs. Dr. Fukuda was in contact with members of the Pacific Northwest National Laboratory (PNNL), who had previously claimed simulations had taken them several hours to complete. I hoped that perhaps as the input data increased, runtimes would increase at greater than linear pace.

UrbanSim's console output provides processing time for each of the included models, so I initially used this information to investigate the models with relatively longer processing times. As seen in Table 1, three models stood out as computationally intensive: the Residential and Non-Residential Location Choice Models and the Feasibility Model, which is part of the overall Developer Model.

Though the Feasibility Model had a longer runtime, I chose to investigate the two Location Choice Models instead, since they shared a common base class from the UrbanSim repository.

Model Name	Avg Runtime (sec)
Residential Pricing	0.7
Non-Residential Rent	0.5
Household Relocation	0.1
Household Location Choice	1.5
Households Transition	0.1
Jobs Relocation	0.1
Jobs Location Choice	1.5
Jobs Transition	0.1
Feasibility	1.9
Residential Development	0.2
Non-Residential Development	0.3
Total	**7.0**

Table 1: Output was steady across several iterations, with three models standing out as computationally intensive

Challenges

The UrbanSim codebase presented a number of challenges. First, I found Python to be a challenging language to read. The language's dynamic typing makes leads to relatively quick development cycles, but at the detriment of those who might read the code later, especially for the express purpose of identifying data flows to and from methods. Second, UrbanSim implements several Python libraries, including Pandas and NumPy, both of which are centered around complex data structures, and neither of which provide robust documentation

on internals. Third, UrbanSim's current documentation is patchwork, and much of the previous literature is now irrelevant. In the words of the UDST team, the project is undergoing “constant reengineering” (UDST). Additionally, the domain language for UrbanSim (economics of urban development) is highly esoteric, compounding the confusion of the lack of strict type definitions. Finally, the MASS library does not include a Python implementation (it is available in C++ and Java), so upon identifying one or more target methods, I would be required to use an external library like Boost or a hybrid language like Cython to make calls to methods in the MASS library.

Despite these challenges, I managed over the course of the quarter to gain a solid understanding of the system. I tried to apply techniques from software development courses I've taken this year to create rough diagrams as I read through the system. I also digested several of Dr. Waddell's academic papers on the early versions of UrbanSim. Though the system has changed in some ways, the underlying architecture has persevered. Many of the original components have been consumed by the Orcas model control module. Orcas handles a great deal of the interaction between secondary and primary memory and keeps track of data structures at runtime. However, few of its interactions with the system proved to be computationally intensive.

I used an open-source tool called Line Profiler (Kern) to measure the relative computational complexity of each statement as I investigated methods. I had hoped that the Line Profiler would help me to identify one or two obvious candidates that called for the lion's share of computational time. However, in trying to fulfill the second part of Dr. Fukuda's requirements to identify simple data structures, I had to dig deep into the architecture. At each level, subprocesses iteratively accounted for large portions of the system's complexity, and I could not identify any obvious candidates. Many of those I identified ran in well under half a second.

Amdahl’s Law

What we came to accept with respect to attempting to parallelize UrbanSim across computer clusters was that Amdahl's Law was likely to render our efforts fruitless. The Seattle metropolitan statistical area is approximately four times larger (Census) in terms of households than the San Francisco example (1.35 million to 350,000). I did not identify any methods that would presumably scale exponentially, so if we consider a linear increase, we might expect 30 seconds per iteration (7 seconds times 4) for the Seattle area. A 20-year cycle would take only 10 minutes.

If one particular process dominated the computational complexity of the UrbanSim system, parallelization would be useful. But the bulk of modern UrbanSim's computational complexity is rooted in methods that were unfit for parallelization. Even if we could parallelize the three most computationally intensive models in the SanFrancisco example, we would only be parallelizing about 55% of the total complexity.

According to Amdahl's Law, the limit on how much improvement in speed can be accomplished for a system is dependent upon the portion that can be parallelized. If we use the 55% mark, we can apply Amdahl's Law as (1 / (1 – p)), where p is the percentage of the program that can be improved via parallel processing. Thus, even if we could fully parallelize all three of those models, we would see at most a 2.22 times improvement in processing time. As such, Dr. Fukuda recommended that I shift my focus to a

different project in the spring.

Learnings and Takeaways

While it was frustrating to be unable to apply the MASS library to UrbanSim, I learned a great deal from participating in this project. This marked the first time that I attempted to gain intimate knowledge of a codebase of this size (10^4 magnitude lines of code). Initially I found the scope of the codebase daunting, and I had difficulty determining the best entry points to begin navigating through it. However, I noticed a tangible difference in the speed at which I was reading the code from the beginning of the quarter to the end. Beyond the direct benefits of engaging with my project, I learned a great deal from my team members in the Distributed Systems Laboratory through their presentations and daily discussions. I believe this experience will prove useful for me in both my professional and academic careers.

Andrew Andersen
Dr. Munehiro Fukuda
CSS499 Winter 2016
March 18, 2016

Works Cited and Referenced

Kakaraparthi, Siva Karthik, Kockelman, Kara M. “An Application of Urbansim to the Austin, Texas Region: Integrated-Model Forecasts for the Year 2030.” *Journal of Urban Planning and Development*. 137.3 (2011): 238-247. Electronic. Pre-print.

Kern, Robert. line_profiler GitHub repository. https://github.com/rkern/line_profiler. Electronic. Accessed January through March 2016.

Noth, Michael, Borning, Alan, Waddell, Paul. “An extensible, modular architecture for simulating urban development, transportation, and environmental impacts.” Computers, Environment and Urban Systems. 27 (2003): 181-203. Electronic.

United States Census Bureau. “American FactFinder.” U.S. Department of Commerce. Electronic. Accessed 15 May 2016.

Urban Data Science Toolkit. GitHub repository. https://github.com/UDST. Electronic Accessed January through March 2016.

Waddell, Paul, et. al. “Microsimulation of Urban Development and Location Choices: Design and Implementation of UrbanSim.” *Networks and Spacial Economics*. 3.1 (2003): 43-67. Electronic.

Wadell, Paul. “Modeling Urban Development for Land Use, Transportation, and Environmental Planning.” *APA Journal*. 68.3 (2002): 297-314. Electronic.