Profiling and Bottlenecks Analysis of an Agent-based Dengue Fever Simulation Model

Pablo A.S. Hugen[1], **Guilherme Galante**[2], **Rogério L. Rizzi**[1], **Eduardo A.A. Cunha**[1]

[1] Colegiado de Ciência da Computacao
Universidade Estadual do Oeste do Paraná (UNIOESTE)
85819-110 – Cascavel – PR – Brazil

`{pablo.hugen,guilherme.galante,rogerio.rizzi,eduardo.argou}@unioeste.br`

Abstract. *Agent-Based Model simulations plays a critical role in computational epidemiology, usually employing GPU aceleration to account for large scale scenarios. In this work, we profile and analyze an agent based model for dengue fever disease spreading simulation as part of a bachelor thesis work, to find bottlenecks and performance issues. We have discovered a bottleneck in two of the model kernels, and inefficient host-to-device memory transfers. The use of a unified memory architecture as also running more kernels in parallel using ISO C++17 Parallelism are proposed as future solutions to these challenges.*

1. Introduction

The spread of infectious diseases, notably exemplified by the *SARS-Cov-2* and *H1N1* outbreaks, has been significantly accelerated by global interconnectivity, highlighting the critical role of computational technologies in public health management. Thus, mathematical models and simulations, as highlighted by [Rachah and Silva 2024], have proven indispensable for formulating effective health strategies. Moreover, the necessity for simulating complex epidemiological scenarios and advancements in High Performance Computing have propelled the use of Agent-Based Models (*ABMs*) in computational epidemiology simulations, as discussed by [Elsheikh 2024]. These models offer insights into epidemic dynamics by simulating disease progression on an individual level, emphasizing the spatial and temporal interaction between agents and their environment [Cunha et al. 2022]. However, the implementation of *ABMs* is not without challenges: They require significant computational effort, both in terms of memory storage and processing time, **which becomes a limiting factor as the number of individuals or spatial resolution increases** [Rosenstrom et al. 2024].

As a result, Graphics Processing Units (*GPUs*) acceleration have been used to deal with the agent scale problem, as shown by [Kitson et al. 2024, Thomopoulos and Tsichlas 2024], usually employng the *CUDA* programming model. Despite this, the model proposed by [Cunha et al. 2022], even though implemented using *CUDA*, still suffers from the scale problem, leading to impractical simulation times in real-world scenarios. Furthermore, recent studies have demonstrated the effectiveness of ISO C++ parallel algorithms in enhancing computational performance across a range of scenarios, using *GPUs* and other accelerators to offer a standardized, cross-platform approach to parallelism [Brown et al. 2019]. In view of this, to address these challenges, the authors are currently working on implementing a model based on the work of [Cunha et al. 2022] utilizing the *ISO C++ parallel algorithms* for potential performance improvements. Therefore, the primary objective of the present work is to **profile the Cunha model to identify and analyze bottlenecks**, aiming to aid in significantly reducing simulation times for real world scenarios.

2. Methodology

In the simulator, each individual agent is characterized by attributes including age, location, and health status, within a spatial environment modeled as a mobility graph, as shown by Figure 1. Agents interact within the surroundings through a series of time steps or cycles, enabling movement within the environment, contact with other agents, being prone to environmental forces and other control measures, and also transitions in health status. These status changes adhere to the *SEIRS* model (as illustrated in Figure 2), cycling agents through predetermined health states: Susceptible, Exposed, Infected, and Recovered; thus dynamically simulating the spread and impact of diseases over time. This conceptual disease spreading model is based on the work by [Cunha et al. 2022], incorporating the Monte Carlo method to account for the stochastic nature of disease spreading, running multiple simulations with random parameters within a predetermined range. Specifically, the model is adapted for *Dengue* fever, incorporating both human and mosquito agents in the simulation. The [Cunha et al. 2022] original model implementation is available online at Github.

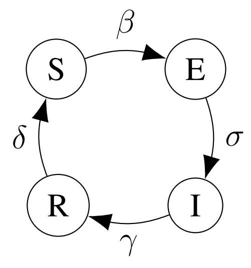


Figure 1. SEIRS Model

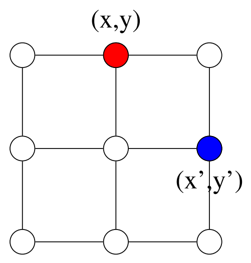


Figure 2. Mobility Graph

The new implementation leverages the *ISO C++17 Parallel Algorithms* specification, facilitating portable parallel programming with the Standard Template Library (STL). This is achieved through the *Nvidia HPC SDK*, which supports full C++17 on CPUs and allows offloading of parallel algorithms to Nvidia GPUs, thereby enabling GPU programming without the need for directives, pragmas, or annotations. Such an approach ensures that programs utilizing C++17 parallel algorithms are easily portable across most platforms. The full implementation can be found at this repository.

For the execution and experimental setup of the simulation, a computer equipped with an Intel Core i7-11390H processor, featuring 4 cores and 16 threads, was used. This system have 16GB of RAM and an NVIDIA GeForce MX450 graphics card with 2GB GDDR6 memory. The simulation scenario consisted of 365 cycles, designed to emulate the dynamics within a city block of Cascavel/PR, providing a comprehensive view of the environmental interactions over an annual period. To analyze the performance of CUDA kernels within the simulation, the Nvidia Visual Profiler (nvvp) tool was employed, offering detailed insights into the execution efficiency and potential bottlenecks within the GPU-accelerated processes.

3. Results

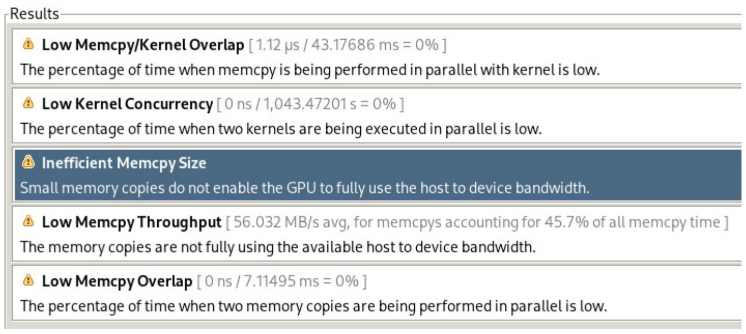
First, the utilization of the Nvidia Visual Profiler tool analysed the simulation kernels performance. The profiling data, as summarized in Table 1, revealed the execution time, frequency of invocations, and the relative importance of each kernel with an overal effect greater than 1% in the context of the entire simulation. The total time of the simulation was roughly *1139.54s*, with *811.19ms* of profiling overhead.

Table 1. Execution time and Invocations of kernels with Importance $>$ 1%

Kernel	Total Execution Time (s)	Invocations	Importance (%)
MosquitosMovement	419.7169	4380	40.2
MosquitosContact	323.8985	4380	31
HumansContact	122.0778	4380	11.7
HumansMovement	27.2122	1095	2.6
Generation	20.16748	361	1.9
MosquitosControl	14.4168	365	1.4

Notably, the *MosquitosMovement* and *MosquitosContact* kernels dominated the computational effort, accounting for over 71% of the total importance, highlighting the implementation complexity of the mosquitoes dynamics, and displaying them as the primary bottlenecks within the simulation. Next, the *HumansContact* and *HumansMovement* kernels, while less computationally intensive, still played a significant role in the simulation time. Finally, the *Generation* and *MosquitosControl* kernels, though having lower importance percentages, can also be considered as optimization targets.

Following, the memory transfers patterns between the CPU and GPU were examined. It was identified host-to-device transfers at 7.11ms for 55 transfers, with 38.63MB at a throughput of 5.429 GB/s, and device-to-host transfers at 35.97ms for 14,747 invocations, transferring 102.402MB at 2.847 GB/s. With those data, the profiler was able to identify some key points in the memory performance of the simulator, as shown in the Figure 3.


Results

Low Memcpy/Kernel Overlap [1.12 μs / 43.17686 ms = 0%]
The percentage of time when memcpy is being performed in parallel with kernel is low.

Low Kernel Concurrency [0 ns / 1,043.47201 s = 0%]
The percentage of time when two kernels are being executed in parallel is low.

Inefficient Memcpy Size
Small memory copies do not enable the GPU to fully use the host to device bandwidth.

Low Memcpy Throughput [56.032 MB/s avg, for memcpys accounting for 45.7% of all memcpy time]
The memory copies are not fully using the available host to device bandwidth.

Low Memcpy Overlap [0 ns / 7.11495 ms = 0%]
The percentage of time when two memory copies are being performed in parallel is low.

Figure 3. Data movement profiler analysis.

Particularly, as presented in the "*Inefficient Memcpy size*" field, the profiling tool revealed a significant use of small and inefficient device-to-host memory copies, primarily due to frequent state transfers for I/O operations. To mitigate this, strategies such as deferring simulation state presentation until completion or employing CUDA Unified Memory technology are suggested. Additionally, it can be seen in the field "*Low Memcpy Throughput*" that the analysis identified issues with low memory throughput, detecting transfers at only *56.032 MB/s* in average. Also, "*Low Memcpy Overlap*" at 0% display a lack of parallel transfers, which could be addressed by implementing asynchronous memory transfers between the host and GPU. Also, other critical observations are the absence of concurrent kernel executions ("*Low Kernel Concurrency*" value at 0%) and no execution/transfer overlapping, highlighted on the section "*Low Memcpy/Kernel Overlap*". Those informations presents a complex challenge that necessitates a redesign of the simulator architecture, an aspect currently under development.

4. Conclusion and future work

In conclusion, we found that the main bottlenecks of the [Cunha et al. 2022] model were the *MosquitosMovement* and *MosquitosContact* kernels, which accounted for over 71% of the total importance. The memory transfers patterns between the CPU and GPU revealed a lot inefficient device-to-host memory copies, primarily due to frequent state transfers for I/O operations. Finally, the absence of concurrent kernel execution was identified as a critical observation, posing a complex challenge that implies the necessity of a redesign of the simulator architecture to enhance overall performance.

Thus, future work is being directed towards a fundamental redesign of the simulator architecture to optimize data access patterns and reduce complexity in *MosquitosMovement* and *MosquitosContact* operations. This redesign aims to enable concurrent kernel execution through the implementation using *ISO C++17 Parallel STL algorithms*. Concurrently, efforts are underway to refactor state I/O operations, adopting unified memory architecture as a first class approach of the simulator.

References

Brown, G., Reyes, R., and Wong, M. (2019). Towards heterogeneous and distributed computing in c++. In *Proceedings of the International Workshop on OpenCL*, pages 1–5.

Cunha, E. A. A. et al. (2022). Aperfeiçoamento e ajuste paramétrico de modelo baseado em agentes para simulação da transmissão da dengue. Programa de pós-graduação em ciência da computaç ão, Universidade Estadual do Oeste do Paraná, Cascavel-PR.

Elsheikh, A. (2024). Promising and worth-to-try future directions for advancing state-of-the-art surrogates methods of agent-based models in social and health computational sciences. *arXiv preprint arXiv:2403.04417*.

Kitson, J., Costello, I., Chen, J., Jiménez, D., Hoops, S., Mortveit, H., Meneses, E., Yeom, J.-S., Marathe, M. V., and Bhatele, A. (2024). A large-scale epidemic simulation framework for realistic social contact networks. *arXiv preprint arXiv:2401.08124*.

Rachah, A. and Silva, T. L. (2024). An agent-based model for controlling pandemic infectious diseases transmission dynamics with the use of face masks. In *AIP Conference Proceedings*, volume 3034. AIP Publishing.

Rosenstrom, E. T., Ivy, J. S., Mayorga, M. E., and Swann, J. L. (2024). Covsim: A stochastic agent-based covid-19 simulation model for north carolina. *Epidemics*, page 100752.

Thomopoulos, V. and Tsichlas, K. (2024). An agent-based model for disease epidemics in greece. *Information*, 15(3):150.