Analysis of Parallelized Memory Algorithms in High Performance Computing

Group 09: Prathyusha M R, Shreyas Udaya

I. Introduction

Remote sensing technologies have increased geospatial data collection and resolution, which requires efficient computational algorithms to process big geographic information systems (GIS) data. This data is crucial for hydrological, topographic analysis, environmental modeling, and earth surface simulation. These models help understand complex environmental interactions, facilitating informed decision-making and policy formulation. Several algorithms are developed to support computational tasks in environmental modeling. However, with the increase in data size, calculating parameters on a single computer is not practical using serial algorithms [1], [2].

Many researchers in geospatial data processing explored parallel algorithms to improve computational efficiency. Parallel programming utilizes modern parallel hardware in high-performance applications, including shared-memory architectures (like multicores) and distributed-memory architectures (like clusters). The complexity of this hardware is increasing due to out-of-order memory hierarchies, smart network interface cards and computing capabilities used as co-processors in various networking-related tasks [3].

Parallel algorithms are used to improve computational efficiency by breaking down complex problems into manageable tasks that can be executed simultaneously using multiple processors. Techniques include OpenMP, MPI, GPGPU, and asynchronous many-tasks. OpenMP is an established framework for parallel programming [2].

OpenMP is the application programming interface (API) standard for parallel computing using shared memory. It provides directives for shared-memory parallelism, enabling developers to create efficient and scalable parallel algorithms. In OpenMP, the program is shared among several threads, where each thread executes a portion of the code concurrently. These threads work together effectively due to the coordinated access to shared memory. This framework is advantageous in providing the parallel capabilities of contemporary multicore processors. It improves the efficiency of algorithms and applications in various fields [4].

In environment modeling, the flow accumulation algorithm is a crucial tool in hydrology and GIS for understanding surface water movement. With flow accumulation calculations in a rasterized topographical model, the algorithm determines each cell's overall upstream contributing area in a digital elevation model. This method helps identify primary flow paths within a watershed and is essential for flood prediction, watershed management, and terrain analysis. Its sophisticated variations and adaptations continually refine our understanding of surface water behavior in diverse terrains [1], [2], [5].

However, parallelization of flow accumulation tasks remains challenging due to spatial dependency and global computation. There is a need to reduce memory requirements for processing large datasets on a single computer.

II. Related Works

This section provides an overview of the existing research [6]–[12] in different parallel algorithms for flow accumulation calculation. It examines various approaches, evaluation methodologies, results, and the challenges they encountered in their respective studies.

Kotyra *et al.* [6] developed faster ways to calculate flow accumulation, resulting in shorter execution times. They suggested two approaches to parallelize flow accumulation algorithms: the bottom-up approach and the top-down approach. The study used 118 distinct data sets to compare six flow accumulation methods in sequential, parallel, and task-based implementations. The result inferred that the top-down algorithm was the fastest, with an average execution time of less than 30 seconds. For flow accumulation calculations, the parallel top-down implementation is observed to be the most suitable algorithm. The average processing time of task-based top-down implementation is 21.1% longer for subcatchments and 32.7% longer for rectangular frames, making it less efficient. The linear time complexity of the algorithm was measured in various settings, including frame 58 and frame 17, with 240 threads per core. Compared to the sequential version, the results showed a high correlation between the number of cores employed and the speedup. The increase in number of cores up to 60 led to reduced average computation time.

Jong *et al.* [7] developed flow accumulation algorithms to determine how the material flows downstream. For parallel and concurrent computations, they employed the asynchronous many-task (AMT) approach. AMT helps to prevent synchronization points and increase the composability of modeling operations. The AMT-based algorithms were evaluated for composability, performance and scalability. It is observed that they function well when paired with other ooperationsand utilize additional hardware efficiently. However, further research is required ito optimizethe algorithms for particular hardware architectures and to assess their performance on larger datasets. The limitation is that the performance of the algorithms was assessed on a limited set of datasets. Also, further research is essential to examine the effects of different hardware architectures, flow direction algorithms, scheduling strategies and programming languages.

Kotyra *et al.* [2] designed seven fast raster-based algorithms to determine the longest flow paths in flow direction grids using a linear time complexity approach. Eight large datasets were used to evaluate the algorithm, which was generated using a hydrological model. The authors compared the algorithm using existing GIS software. Depending on the dataset and algorithm, the algorithms obtained significant speedups up to 30 times quicker on Windows and 17 times faster on Ubuntu. The suggested algorithm achieved fast and accurate results in determining the longest flow pathways in flow direction grids. However, their approach might not be applicable to unsteady flow conditions since it is based on raster data and a steady-state flow assumption. Future research should explore algorithms based on more complex models. Also, it should support the scalability and portability of their algorithms to other platforms and architectures.

Cho *et al.* [8] proposed the longest flow path algorithm, which computes few rasters to enhance efficiency and decrease computation time. They developed an algorithm based on depth-first search and breadth-first search. The approach uses Hack's law-derived equations to estimate the longest flow length. In order to speed up traversal and eliminate inferior neighbor cells, the algorithm additionally uses a branching technique. The suggested method is evaluated through benchmark experiments conducted in Georgia and Texas, comparing the algorithm's performance with the Arc hydro longest flow path tool for ArcGIS Pro. The results showed that the algorithm's performance is affected by disk type and memory size, with solid-state drives and larger memory sizes resulting in faster computation times. The authors conclude that the proposed algorithm is valuable to environmental modeling and software. One limitation is that the experiments were conducted on a limited data set from two states in the United States. Furthermore, it is possible that the results may not be generalized to alternative geographical areas or datasets.

Stojanovic *et al.* [9] suggested accelerating the flow distribution phase using MPI on a cluster. The author suggested the parallelization of the flow distribution computation phase of the watershed analysis algorithm using MPI. Two different MPI implementations were discussed, along with the analysis of the advantages and challenges of both parallel implementations. The experimental evaluation is conducted on several large DEM datasets and varying numbers of computers in the cluster. They observed the approach that overlaps process computing and communication achieves the best results. The proposed MPI solutions effectively accelerate the flow accumulation step of watershed analysis. The speedup using MPI is significant compared to sequential execution. While these are effective, other methods are not considered, and neither are other libraries, such as OpenMP or CUDA.

Lal *et al.* [10] presented a quantitative analysis on the caches for memory divergent workloads simulated by gpgpu-sim. Increasing the size of the L1 data cache improved the spatial locality while increasing L2 improved temporal locally. The authors analyzed the impact of parameters like block size and thread count on the algorithm's performance and optimized it on different hardware configurations. The evaluation is based on benchmarks run on an NVIDIA GPU. For memory-divergent tasks, the study focused on data locality in GPU caches. Higher inter-warp hits (46%) at the L1 cache for memory-divergent workloads compared to the state-of-the-art. However, data over-fetch wastes around 50% of cache capacity and other limited resources. The limitations include its focus on NVIDIA GPU architectures, its limited application to different types of workloads, and its inability to consider other potential performance bottlenecks.

Kotyra *et al.* [11] proposed a fast watershed delineation algorithm for GPU that uses OpenMP and CUDA. The algorithm iteratively processes each cell in the flow direction raster, identifying its downstream neighbor and checking if it belongs to the same catchment area. It includes optimizations to reduce memory usage and improve performance. The algorithm outperformed traditional GIS software packages in terms of speed and efficiency. The algorithm's main loop calls the GPU kernel repeatedly, accounting for an average of 28.8% of the overall execution time. Data transfers account for 34.5% of the total time on average. The choice of hardware and software platforms affects the algorithm's performance, and its implementation may require specialized knowledge in parallel programming and GPU computing.

Huang *et al.* [12] discussed a comprehensive study of in-memory computing. They discussed portability, robustness, usability, and performance of software. The evolution of in-memory computing is explained. The authors suggested they commit history for two in-memory libraries and observed most of the commits were towards performance maintenance, suggesting it has a significant role in computation. The in-memory computing has better performance than traditional post-processing. The portability, robustness, usability, and implementation of software are all considered in the evaluation of performing in-memory computing. The results suggest that in-memory computing provides significantly better performance and scalability than traditional post-processing.

References

[1] H. Cho, "Memory-efficient flow accumulation using a look-around approach and its openmp parallelization," *Environmental Modelling & Software*, vol. 167, p. 105771, 2023.

[2] B. Kotyra and Łukasz Chabudziński, "Fast parallel algorithms for finding the longest flow paths in flow direction grids," *Environmental Modelling & Software*, vol. 167, p. 105728, 2023.

[3] J. Löff, D. Griebler, G. Mencagli, G. Araujo, M. Torquati, M. Danelutto, and L. G. Fernandes, "The nas parallel benchmarks for evaluating c++ parallel programming frameworks on shared-memory architectures," *Future Generation Computer Systems*, vol. 125, pp. 743–757, 2021.

[4] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: portable shared memory parallel programming*. MIT press, 2007.

[5] G. Zhou, H. Wei, and S. Fu, "A fast and simple algorithm for calculating flow accumulation matrices from raster digital elevation," *Frontiers of Earth Science*, vol. 13, pp. 317–326, 2019.

[6] B. Kotyra, Łukasz Chabudziński, and P. Stpiczyński, "High-performance parallel implementations of flow accumulation algorithms for multicore architectures," *Computers & Geosciences*, vol. 151, p. 104741, 2021.

[7] K. de Jong, D. Panja, D. Karssenberg, and M. van Kreveld, "Scalability and composability of flow accumulation algorithms based on asynchronous many-tasks," *Computers & Geosciences*, vol. 162, p. 105083, 2022.

[8] H. Cho, "A recursive algorithm for calculating the longest flow path and its iterative implementation," *Environmental Modelling & Software*, vol. 131, p. 104774, 2020.

[9] N. Stojanovic and D. Stojanovic, "Accelerating multiple flow accumulation algorithm using mpi on a cluster of computers," *Studies in Informatics and Control*, vol. 29, no. 3, pp. 307–316, 2020.

[10] S. Lal and B. Juurlink, "A quantitative study of locality in gpu caches," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 20th International Conference, SAMOS 2020, Samos, Greece, July 5–9, 2020, Proceedings*, (Berlin, Heidelberg), p. 228–242, Springer-Verlag, 2020.

[11] B. Kotyra, "High-performance watershed delineation algorithm for gpu using cuda and openmp," *Environmental Modelling & Software*, vol. 160, p. 105613, 2023.

[12] D. Huang, Z. Qin, Q. Liu, N. Podhorszki, and S. Klasky, "Identifying challenges and opportunities of in-memory computing on large hpc systems," *Journal of Parallel and Distributed Computing*, vol. 164, pp. 106–122, 2022.