How STF allows trivial overlap of I/O and computations on Forward RTM

Pedro Henrique Boniatti Colle, Lucas Mello Schnorr

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{phbcolle,schnorr}@inf.ufrgs.br

Abstract. *With the evermore complex ecosystem of High Performance Computing (HPC), the Sequential Task Flow (STF) model arrives to provide a convenient interface to compute in heterogeneous architectures. This paper presents* ***Star-Fletcher****, a Reverse Time Migration (RTM) program implemented using STF alongside time tiling which resulted in the side effect of complete I/O to disk masking.*

1. Introduction

It's the end days of homogeneous architectures. With the ever-increasing computing demands of High Performance Computing (HPC) applications, nodes in data centers are required to incorporate increasingly different hardware accelerators, even more after the AI boom. This increases the complexity of writing programs that uses all the resource in a node.

One of the answers for this problem are Sequential Task Flow (STF) runtime systems. In those, work is segmented in tasks with explicit dependencies. Those tasks are submitted sequentially and execute asynchronously with respect to their dependencies. With this model it is easy to distribute task between different architectures. One could implement a task to run in the CPU and also provide an implementation to run in the GPU. The runtime handles the load balancing and data transfers. Some popular runtimes are PaRSEC [Bosilca et al. 2013] and StarPU [Augonnet et al. 2009], the latter was used in this paper. Both of these systems have a data management side, which allows for easier tracking of dependencies and task submission. StarPU's automatic memory management was essential in enabling many mechanisms for this paper's program optimizations. Many applications adopted this paradigm for a plethora of contexts, like dense linear algebra [Abdulah et al. 2022], Computational Fluid Dynamics (CFD) [Couteyen Carpaye et al. 2018, Leandro Nesi et al. 2020] and Reverse Time Migration (RTM) applications [Boillot et al. 2014, AlOnazi et al. 2019].

RTM is a tool of seismic analysis for detecting subsurface deposits of materials. It propagates a wavefield forwards in time, records it and then integrates the recording back in time to generate a partial image. This process is repeated many times, each called a *shot*, to generate the final visualization. There are no dependencies between *shots*, making RTM applications embarrassingly parallel. The Fletcher algorithm [Fletcher et al. 2009] can deal with Tilted Transverse Isotropic (TTI) medium and is the chosen implementation of this paper's program.

One important characteristic of RTM is that it's recorded data between the forwards and backwards propagation can be massive. A normal volume of $1000 \times 1000 \times 500$

saved in FP64 results in $4\ GB$ of data, which is saved many times during execution. That makes this kind of application I/O bound. Modern research focuses in overlapping computation with I/O operations [AlOnazi et al. 2019, Künas et al. 2025]. AlOnazi's work calls back to the STF paradigm, in which the StarPU runtime system was used to manage the I/O and data transfer operations while the computation was done in GPUs. This utilization was limited to homogenous computation on GPUs and two-level parallelism, where the whole medium was placed in the GPUs shared memory. This resulted in a two-tier parallel system, where computations used traditional parallel methods while the I/O used the STF model.

In order to fill this design space, we propose **Star-Fletcher**, a first-order task parallelism with space-segmentation implementation of Fletcher using the StarPU library. It represents only the forwards wave propagation, with the snapshots being saved to disk in order to always simulate a medium too large to fit in main memory. The decision of space segmentation allows for first-order task parallelism while enabling spaces too large to fit in memory to be processed efficiently. We utilize the works of [Leandro Nesi et al. 2020, Qu et al. 2023] to implement time tiling in order to incur speed-ups with respect to cache reuse. By reformulating **Star-Fletcher** to fit this paradigm, all I/O operations became masked behind computation. This side effect was possible because STF, and especially StarPU, are extremely well-equipped to implement the time tiling paradigm.

In the section 2 it is discussed the implementation detail of **Star-Fletcher**, in section 3 it is present partial results of overlap of I/O and computation and in section 4 it is elaborated future directions for the project and which experimental results are being aimed for.

2. Methods and Materials

In Fletcher's basic implementation, two buffers are used, the current and the previous. The current is used to compute the values that will be added to the previous, akin to $prev+= F(curr)$. Then the buffers are swapped, and the process repeats until all iterations are executed. This format stops time tiling because it requires that an iteration completely finishes before starting the next one. To enable time tiling, the swap was removed, resulting in the computation $A_t := F(A_{t-1}) + A_{t-2}$. This relies on StarPU's automatic memory management[1]. With A_t as an exclusively write dependency, StarPU can allocate it when necessary and free it when no other task requires it. This mechanism allows for time tiling to comprise as many iterations as the system's memory allows. Another consequence is that writing $prev$ to disk, or in this case A_t, does not block the computation of the next iteration anymore, allowing for asynchronous I/O.

We have implemented three tasks in StarPU to implement time tiling : `wave_propagation`, `wave_insertion` and `write_block`. The space segmentation required disjointed memory, which required additional bounds checks when accessing neighboring blocks in the `wave_propagation` task. Using meta programming, the number of checks were reduced to three in the cross product and two in the second derivative. The disjointed space also affected `write_block` task. For it, each thread used its own file and wrote the received blocks in them with some metadata that enables later re-

[1]Chapter 27.2 of StarPU's manual `https://files.inria.fr/starpu/doc/html/AdvancedDataManagement.html`

construction of the full volume. Currently, on program termination, a script congeals all these files into one, which will be integrated in the final application.

3. Preliminary Results

The program was tested in one of the nodes of the 'cei' partition of INF/UFRGS's PCAD platform (2x Intel(R) Xeon(R) Silver 4116 CPUs 2.1 GHz, 48 threads, 24 cores). A volume of $600 \times 600 \times 600$ was iterated 10 times, snapshooting at various interval configurations. To illustrate the I/O masking, Figure 1 represents the execution where the space was divided in 64 blocks and, at each step, the full volume was saved to disk. The y axis represent interations and x axis time. The blocks are represented as vertical lines in each iteration, but they are represented so close together that it appears as a bar.

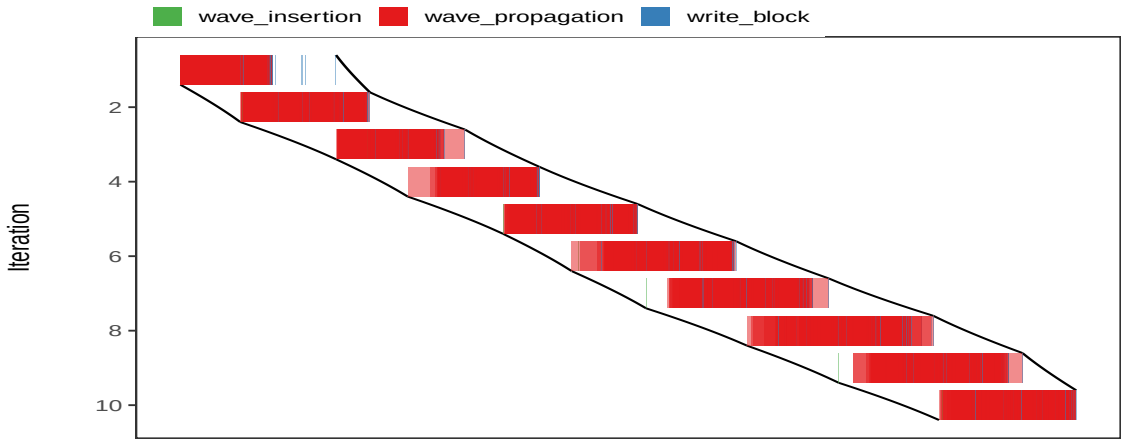

Figure 1. Trace showing the parallelization of tasks per iteration.

Barely any `write_block` is visible in Figure 1. This shows that it is being interleaved with the computation and is not part of the execution's critical path at any point. We can analyze the data in Table 1 to see that each I/O task does not comprise a big individual expense. Scattering them throughout computation results in an effective performance gain.

Table 1. Amount of Tasks executed with their Mean, Min and Max execution time in milliseconds.

Task	Amount	Mean	Min	Max
`wave_insertion`	50	2.9	1.83	6.33
`wave_propagation`	3200	99100	55100	451000
`write_block`	704	39.5	13.9	628

4. Conclusion and Future Work

The ease of implementing time tiling in **Star-Fletcher** was only possible due to the STF model. StarPU's features allow for an easy implementation and as a bonus it breaks dependencies that stop I/O masking. Nonetheless, this program is a work in progress and more comparisons with conventional parallel versions of fletcher are necessary to validate **Star-Fletcher** performance. It's still necessary to explore if the time tiling design really results in performance gains and in cache reuse.

References

Abdulah, S., Cao, Q., Pei, Y., Bosilca, G., Dongarra, J., Genton, M. G., Keyes, D. E., Ltaief, H., and Sun, Y. (2022). Accelerating Geostatistical Modeling and Prediction With Mixed-Precision Computations: A High-Productivity Approach With PaRSEC. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):964–976.

AlOnazi, A., Ltaief, H., Keyes, D., Said, I., and Thibault, S. (2019). Asynchronous Task-Based Execution of the Reverse Time Migration for the Oil and Gas Industry. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11. ISSN: 2168-9253.

Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P.-A. (2009). Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. In Sips, H., Epema, D., and Lin, H.-X., editors, *Euro-Par 2009 Parallel Processing*, pages 863–874, Berlin, Heidelberg. Springer Berlin Heidelberg.

Boillot, L., Bosilca, G., Agullo, E., and Calandra, H. (2014). Task-Based Programming for Seismic Imaging: Preliminary Results. In *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, pages 1259–1266.

Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., and Dongarra, J. J. (2013). PaRSEC: Exploiting Heterogeneity to Enhance Scalability. *Computing in Science & Engineering*, 15(6):36–45.

Couteyen Carpaye, J. M., Roman, J., and Brenner, P. (2018). Design and analysis of a task-based parallelization over a runtime system of an explicit finite-volume CFD code with adaptive time stepping. *Journal of Computational Science*, 28:439–454.

Fletcher, R. P., Du, X., and Fowler, P. J. (2009). Reverse time migration in tilted transversely isotropic (TTI) media. *GEOPHYSICS*, 74(6):WCA179–WCA187.

Künas, C. A., Freytag, G., and Navaux, P. O. A. (2025). Enhancing Reverse Time Migration Simulations in HPC Systems Through I/O and Computation Overlapping. In Guerrero, G., San Martín, J., Meneses, E., Barrios Hernández, C. J., Osthoff, C., and Monsalve Diaz, J. M., editors, *High Performance Computing*, pages 68–82, Cham. Springer Nature Switzerland.

Leandro Nesi, L., da Silva Serpa, M., Mello Schnorr, L., and Navaux, P. O. A. (2020). Task-based parallel strategies for computational fluid dynamic application in heterogeneous CPU/GPU resources. *Concurrency and Computation: Practice and Experience*, 32(20):e5772. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5772.

Qu, L., Abdelkhalak, R., Ltaief, H., Said, I., and Keyes, D. (2023). Exploiting temporal data reuse and asynchrony in the reverse time migration. *The International Journal of High Performance Computing Applications*, 37(2):132–150.