GPU acceleration of SWIFT for heterogeneous Exascale supercomputers

Abouzied Nasar[1], Georgios Fourtakas[1], Scott[2] Kay, Benedict D. Rogers[1]
Departments: 1. Civil Engineering and Management. 2. Physics and Astronomy. The University of Manchester. Manchester, UK

Matthieu Schaller
Lorentz Institute for theoretical physics. Leiden University
Leiden, The Netherlands

I. Introduction

As part of ExCALIBUR[1], a UK-wide research programme aimed at achieving exascalable scientific software, we present recent development of the open-source SPH solver SWIFT[2] transforming the solver into a heterogeneous compute framework leveraging both CPUs and GPUs. Traditional Smoothed Particle Hydrodynamics (SPH) solvers rely on thread-parallelism (e.g. [1], [2], [3]) but SWIFT uniquely adopts task-parallelism to address scalability challenges due to SPH's Lagrangian nature ([4], [5]) which are exacerbated by space and time adaptivity [6] as required for feasible time-to-solution for simulations with $O(10^{11})$ particles [7]. GPU acceleration within such a complex framework is challenging and this paper presents novel solutions aimed at maximising GPU acceleration efficiency whilst maintaining the scalability benefits of task-parallelism.

Recently deployed and upcoming exascale supercomputers, capable of $O(10^{18})$ Floating Operations Per Second (FLOPS), mark a step-change in computational performance and our ability to resolve larger problems primarily through heterogeneous computing with GPUs [8], [9]. However, most existing High-Performance Computing (HPC) software has been largely optimised for CPU-based parallelisation. In addition to the UK exascale programme ExCALIBUR[1], there are concerted efforts globally[3,4] aimed at developing software capable of leveraging the new GPU-dominated massively parallel exascale architectures. New HPC superchips, such as Nvidia's Grace-Hopper and AMD's MI300x, are beginning to replace conventional GPUs. These superchips feature large CPU memory pools paired with relatively smaller GPU memory and particularly high-bandwidth bespoke CPU-GPU connections which aim to "unify" CPU and GPU memory. This facilitates the use of larger datasets by mitigating efficiency losses from slower CPU-GPU data transfers over conventional PCIe buses. For instance, the UK and EU's top supercomputers (Isambard-AI and Jupiter, respectively) are composed of thousands of Grace-Hopper superchips where the CPU-GPU connection is 450 GB/s allowing rapid data transfer from large (960 GB) CPU memory to smaller (144 GB) GPU memory, compared to 64 GB/s on state-of-the-art PCIe 5. Conventional GPU-accelerated SPH solvers (e.g. [1], [3], [10]) optimize performance by retaining data on GPUs for the entire simulation and as such exhibit massive speedups in comparison to purely CPU-based solvers. However, this approach underutilizes exascale systems designed to alleviate limitations imposed by much smaller memory availability on single GPUs.

By leveraging the larger CPU memory and faster interconnects of exascale machines, this paper proposes novel techniques to optimize task execution and memory usage, enabling the cosmology solver SWIFT to harness the full potential of exascale heterogeneous architectures which will enable the creation of higher fidelity digital twins of our universe (e.g. [7]) and better our understanding of the complex interplay of the multi-scale physics involved. As this research focuses on accelerating the SPH hydrodynamics solver in SWIFT it will extend its capabilities beyond astrophysics forming the foundation of a novel SPH solver for engineering applications which require extreme fidelity and adaptivity such as boiling, multi-phase, turbulent flows. The efficiency of our proposed approach and speedups from 2.2 to 11 (in comparison to CPU execution) are demonstrated on the Grace-Hopper superchips and on a similarly designed pre-cursor system with V100 GPUs.

II. GPU implementation summary

In SWIFT, SPH particle interactions are of two types: 1) self tasks which interact all particles in a cell with each other, 2) pair tasks which interact particles in one cell with particles in another cell (and vice-versa). Self and pair tasks are further refined into density, gradient and force sub-types which compute particle densities, dissipation and acceleration, respectively [11]. With the proposed approach, the aforementioned task sub-types are offloaded for GPU acceleration as they are the most compute intensive. Other task types of minimal compute intensity (e.g. time integration, MPI communications) are left for the CPU to execute which: (a) Allows concurrent CPU *and* GPU computations. (b) Minimises inefficient GPU operations where memory access overheads outweigh computational parallelisation benefits. (c) Allows re-use of the highly efficient CPU-based MPI mechanisms.

SWIFT's task scheduler [12] is designed to continuously enqueue tasks for each of the available CPU cores as and when they have been unlocked which occurs after other tasks they depend on have been completed.

[1] https://excalibur.ac.uk
[2] https://swift.strw.leidenuniv.nl
[3] https://exascale-projects.eu
[4] https://www.exascaleproject.org

As shown in Figure 1, all available threads fetch a task from their queues and after executing the task unlock its dependencies and fetch the next task in the queue. The scheduler feeds tasks into the queues until all tasks are complete, at which point the solver moves onto the next computational time step.

Whilst SWIFT's scheduling process is highly suited for CPU execution, SWIFT's tasks are too fine-grained for efficient execution on GPUs as they typically act on $O(100\text{-}1000)$ particles as-and-when they are fetched by a CPU thread. This would be inefficient on the GPU due to the overheads of launching GPU kernels (blocks of computations) and CPU-GPU data transfer dominating when the size of a computation is too small. As such, we developed an approach where particle data required for tasks of the same type acting on particles in different cells is packed into large "part_send" arrays of CUDA specific data structures (float4, int4, etc.) with 128-bit alignment restriction, designed to vectorise data access and computations on the GPU.

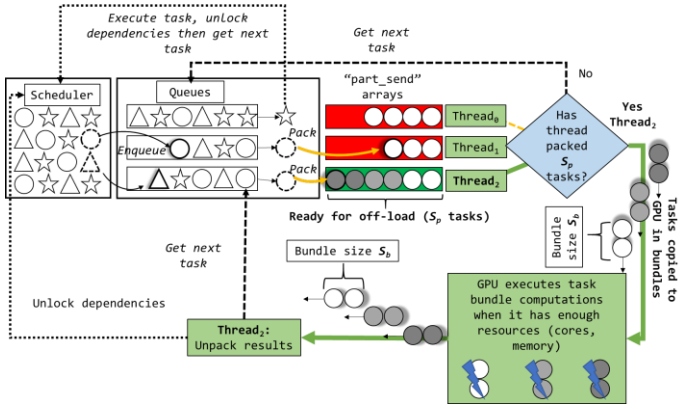

Figure 1. GPU-offload mechanism for SWIFT tasks. Circles in this image represent pack tasks where the data required for the computations is packed into arrays of part_send structs. Once a thread has packed the data for S_p tasks the offload process begins.

New pack tasks are implemented within SWIFT such that the packing process is also task-parallel. In pack tasks, the data for particles in a cell on which the task acts is packed as-and-when a CPU thread acquires the task. As shown in Figure 1, we design the GPU offload process such that when pack size (S_p) tasks have been packed we begin transferring the `part_send` array to the GPU and launching a set of GPU kernels to perform the computations required. To optimise the code further, the `part_send` array is split into smaller chunks each containing the data required for a bundle of S_b tasks where $S_b<S_p$. This allows us to leverage CUDA streams where the host issues instructions operating on different sub sets of the `part_send` arrays via different streams. This allows overlapping computations issued in one stream with CPU-GPU communications issued in other streams thereby hiding the CPU-GPU data transfer time underneath GPU computations which hides latency significantly improving total time for the completion of a pack of S_p tasks.

III. Results

The GPU offloading process discussed in Section II is used to execute the density, gradient, and force tasks in two sets of simulations on different systems. The first system consists of Nvidia V100 GPUs connected to POWER9 CPUs and the second consists of Nvidia GH200 superchip. On both systems, the CPU-GPU connections are proprietary NVLink connections which allow for extremely fast CPU-GPU communications (in comparison to PCIe). The computational time required for the GPUs to execute the SPH tasks is compared with the time required to execute the same tasks on the respective CPU of the system. The test case used is the 3-D Gresho-Chan vortex test [13] with $N_p=256^3$ particles contained in 64^3 cells to ensure cell width is equal to $2h$ where h is the smoothing length. The GPU time includes the time required to pack the task data and the time required to update the CPU particle data after the GPU has completed the computations, i.e. all CPU-GPU communication time is included. 32 CPU threads are used when offloading to the GPU to pack the task data and also compute the other required tasks (time integration, etc. [11]).

A. V100 GPU vs POWER9 CPU

Figure 2 shows the compute times for several pack sizes S_p and bundle sizes S_b. There is an optimal combination (S_p=2048 and S_b=512) for self tasks and (S_p=1024 and S_b=256) for pair tasks when executed on the V100 GPU. This is due to this combination allowing for the maximum overlap between GPU computations and CPU-GPU communications as shown in Figure 3 which shows that CPU-GPU communications occur whilst the GPU is busy computing tasks for other bundles.

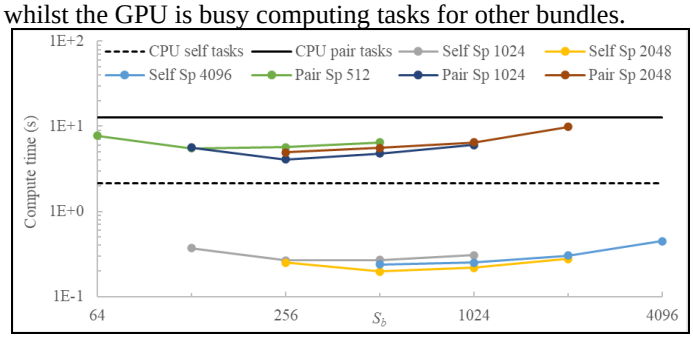

Figure 2. Variation of task compute time on V100 with S_p and S_b for $N_p = 256^3$ using 32 host threads compared to CPU times with 32 threads.

Comparing the CPU times in Figure 2 to the optimum configuration for the GPU the speedup when executing the tasks on the GPU is 3.1 for pair tasks and 10.8 for self tasks. The smaller speedup for pair tasks is due to the pair tasks requiring more data for fewer computations; since most particles in pair tasks are further than $2h$ from each other fewer particles within the data sets offloaded to the GPU interact therefore the ratio of communications to computations is greater when compared to self tasks where most particles in the same cell will interact with each other. Also, the CPU code uses extremely optimised pre-sorted particle interactions for pair tasks (pseudo-Verlet lists [14]) which are not implemented in the GPU code therefore the pair task comparison is not like-for-like.

The compute times for the optimum combination of S_p and S_b is dominated by the time required to pack and unpack the particle data after the GPU has sent the results back to the CPU, roughly 50% of total time. Whilst the packing/unpacking is parallelised over 32 threads, the complexity of the original CPU form of particle data (structs of arrays of structs of arrays) prevents vectorisation of the packing process. However, work is underway to amend the CPU data structures to resemble the GPU data such that the need for data structure conversion is eliminated. This is envisaged to improve the overall speedups of GPU offloading potentially doubling the achieved speedups.

Figure 3. Profile for a GPU offload cycle on V100 with N_p=256^3, S_p=2048 and S_b=512. (a) Pair tasks. (b) Self tasks. Green, lavender and dark blue blocks represent CPU to GPU data transfer, GPU to CPU data transfer and kernels executing self and pair tasks, respectively.

B. H100 GPU vs Grace CPU

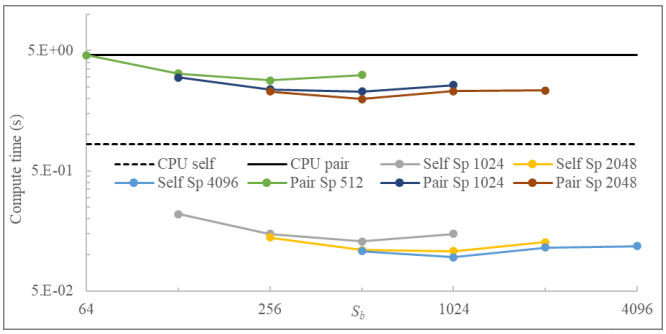

Figure 4. Variation of task compute time on GH200 with S_p and S_b for N_p = 256^3 using 32 host threads compared to CPU times with 32 threads.

Figure 4 shows the timings for the same computations in Section III. A., as executed on the Grace-Hopper superchip. Similar behaviour is observed with an optimum combination of S_b and S_p emerging as (S_p=4096 and S_b=1024) for the self tasks. The achieved speedups for the Hopper GPU are 2.3 and 8.8 for the self and pair tasks respectively in comparison to the *Grace CPU*. The packing/unpacking time in the H100 computations is 75% of total offload time (in comparison to 50% for the V100) which further highlights the need for the planned development of amending SWIFT's original CPU data structure.

IV. Conclusions and Current and Future Work

The developments presented in this abstract demonstrate the potential of our novel approach for GPU acceleration within task-parallel SPH with significant speedups of task execution on GPUs (2.2 to 11 depending on task type) vs CPU execution with 32 cores. However, this can be significantly improved by minimising or eliminating the wall clock time for packing and unpacking the data before offloading to GPUs which is currently up to 75% of the total GPU offload time.

In current work, the functionality and correctness of SWIFT's massively scalable CPU-based MPI communication strategy has been validated with GPU accelerated simulations executed on multiple nodes using multiple GPUs per node. Access to large systems with O(10-100) superchip nodes is currently being arranged for massive computational scalability testing which will be presented in SPHERIC 2025. Current activity is also focusing on developing further optimisation of the data structures, incorporating adaptivity and improving portability across different HPC systems.

Acknowledgment

The authors would like to acknowledge EPSRC for grant EP/W026775/1 which funded the research in this paper.

References

[1] Hérault, A., G. Bilotta, and R.A. Dalrymple. Journal of Hydraulic Research, 2010. 48(sup1): p. 74-79.
[2] Cercos-Pita, J.L. Computer Physics Communications, 2015. 192: p. 295-312.
[3] Domínguez, J.M., G. Fourtakas, C. Altomare, R.B. Canelas, A. Tafuni, O. García-Feal, I. Martínez-Estévez, A. Mokos, R. Vacondio, and A.J. Crespo. Computational Particle Mechanics, 2022. 9(5): p. 867-895.
[4] Zhu, G., J. Hughes, S. Zheng, and D. Greaves. Computer Physics Communications, 2023. 284: p. 108608.
[5] Guo, X., B.D. Rogers, S. Lind, and P.K. Stansby. Computer Physics Communications, 2018. 233: p. 16-28.
[6] Borrow, J., R.G. Bower, P.W. Draper, P. Gonnet, and M. Schaller. arXiv preprint arXiv:1807.01341, 2018.
[7] Schaye, J., R. Kugel, M. Schaller, J.C. Helly, J. Braspenning, W. Elbers, I.G. McCarthy, M.P. van Daalen, B. Vandenbroucke, and C.S. Frenk. Monthly Notices of the Royal Astronomical Society, 2023. 526(4): p. 4978-5020.
[8] Donahue, A.S., P.M. Caldwell, L. Bertagna, H. Beydoun, P.A. Bogenschutz, A. Bradley, T.C. Clevenger, J. Foucar, C. Golaz, and O. Guba. Journal of Advances in Modeling Earth Systems, 2024. 16(7): p. e2024MS004314.
[9] Sprague, M.A., S. Ananthan, G. Vijayakumar, and M. Robinson. in Journal of Physics: Conference Series. 2020. IOP Publishing.
[10] Marongiu, J.-C., J. Leduc, and M. Schaller. in 6th international SPHERIC workshop. Hamburg, Germany. 2011.
[11] Schaller, M., J. Borrow, P.W. Draper, M. Ivkovic, S. McAlpine, B. Vandenbroucke, Y. Bahé, E. Chaikin, A.B. Chalk, and T.K. Chan. Monthly Notices of the Royal Astronomical Society, 2024. 530(2): p. 2378-2419.
[12] Gonnet, P., A.B. Chalk, and M. Schaller. arXiv preprint arXiv:1601.05384, 2016.
[13] Borrow, J., M. Schaller, R.G. Bower, and J. Schaye. Monthly Notices of the Royal Astronomical Society, 2022. 511(2): p. 2367-2389.
[14] Willis, J.S., M. Schaller, P. Gonnet, R.G. Bower, and P.W. Draper, in Parallel Computing is Everywhere. 2018, IOS Press. p. 507-516.