Towards an efficient parallel skeleton for generic iterative stencil computations in distributed GPUs

Manuel de Castro
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
manuel.castro@alumnos.uva.es

Inmaculada Santamaría-Valenzuela
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
msantamaria@infor.uva.es

Sergio Miguel-López
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
sergio.miguel.lopez@alumnos.uva.es

Yuri Torres
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
yuri.torres@infor.uva.es

Arturo Gonzalez-Escribano
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
arturo@infor.uva.es

I. Introduction

Iterative stencil computations, or Iterative Stencil Loops (ISLs), are a class of computations that update the elements of an array using the values of the previous iteration for some neighbor elements chosen with a specific pattern. The stencil pattern determines both the exact neighbors used and the weights applied to each one. These programs are used to compute the numerical solution of partial differential equations in discretized spaces, simulate the evolution of physical systems, compute convolutions, apply image filters, etc.

This important class of computations presents a high degree of parallelism. On a given iteration, the update of each array position can be computed independently. They are appropriate for many-core computational models and platforms, such as modern GPUs. Thus, there are many implementations using programming models such as CUDA [1] or OpenCL [2].

The memory capabilities of GPUs are not enough to store the huge arrays desired for many real case situations. Thus, approaches for using multi-GPUs in one or more distributed nodes are needed. However, programming distributed multi-GPUs with CUDA or OpenCL implies introducing many complexities in the code to deal with data partition, the management of different devices, and repetitive data transfers between the host and the devices. It also implies introducing a distributed programming model, such as MPI, to manually manage the communications across nodes.

These data transfers and communications are repeated across iterations, and can be automated. Thus, specific frameworks and parallel skeletons for stencil computations in shared and distributed memory exist (e.g. [3]). Multi-GPU approaches and frameworks have been also proposed and tested (e.g. [4]–[7]). However, some are restricted to specific types of stencils, some are limited to specific types of devices or use non-generalizable optimization techniques, and some do not completely hide the programming complexities to the user.

A more generic approach is to use modern programming abstractions that can be used to simplify part of these tasks and to increase portability. For example, Celerity [8] is a distributed heterogeneous programming model built on top of SYCL [9]. Celerity provides some general communication patterns, including one for stencil or similar computations based on neighbor synchronization patterns. Nevertheless, the current implementations of these tools are not yet completely efficient in the management of memory or asynchronous data transfers, and some limitations are still found in their interfaces (e.g. non-compact, irregular and asymmetrical stencils).

In this work we present a high-productivity parallel programming skeleton for iterative stencil computations on distributed multi-GPUs. It supports any type of n-dimensional geometric stencils (compact, non-compact, symmetric or asymmetric) of any order. It is implemented as a C99 library function usable in any C/C++ environment. It uses an abstract specification of the stencil pattern (neighbors and weights) to internally derive the data partition, synchronizations and communications, splitting the computation to better overlap it with communications. The skeleton includes a generic GPU kernel that can be used for any stencil. It also includes a tool to generate more optimized kernels for each stencil pattern, or even the possibility of providing a user optimized kernel. To enable fast synchronization and device management it is programmed with the Controller heterogeneous programming model [10], [11], that provides similar portability as SYCL approaches. The distributed communications are internally calculated and managed with the Hitmap library [12], built on top of MPI. A preliminary experimental study using NVIDIA GPU clusters shows that this skeleton can achieve very good strong and weak scalability for several types of stencils with up to 16 GPUs in 4 nodes. The results also show that our proposal outperforms a specific implementation of an example stencil using Celerity in a smallheterogeneous cluster with 4 GPUs in 2 nodes.

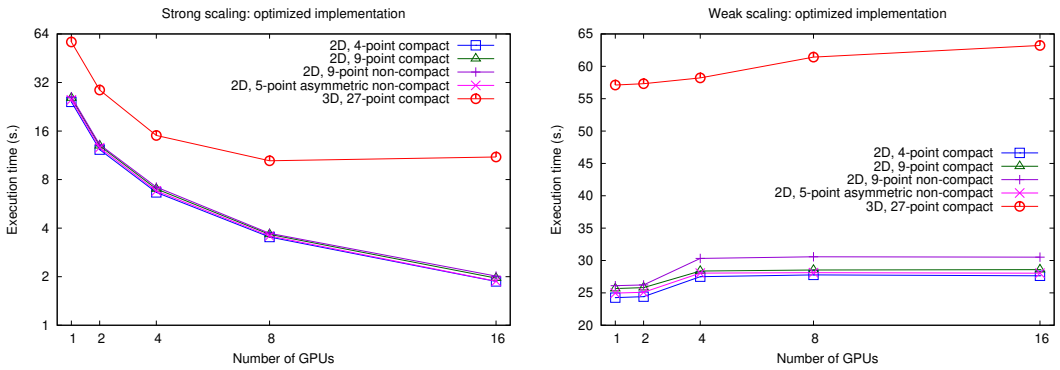

Fig. 1: Scalability results for the stencil skeleton using automatically generated kernels.

II. The stencil computation skeleton

The proposed skeleton is presented as a reentrant higher-order function implemented in C99, usable with any modern C/C++ compiler. It computes a given number of iterations for a stencil computation, applying the operator to each element of an n-dimensional array. The stencil operator can be expressed as an array of weights for each relative neighbor, and a division factor of the weighted sum. The skeleton provides a generic kernel that works for any stencil, but the user can provide a further optimized kernel code. Optimized kernels can also be automatically generated with a provided tool. Other parameters include pointers to functions to initialize the distributed array and to output the results.

The skeleton analyzes the stencil pattern, determining borders and halo sizes on each direction. It computes the array partition and communication pattern, allocating the array buffers. On each iteration it executes the following steps:

1) Execute the stencil operator only on the border elements.
2) Initiate asynchronous operations to transfer the computed borders to the host, to communicate them to other processes, and to transfer the received halos to the device.
3) Meanwhile, the stencil operation is computed for all the remaining array elements in the device, overlapping this computation with the communication.

III. Experimentation

An experimental study is conducted to test the scalability properties of the proposed skeleton. Strong and weak scalability has been tested for optimized implementations of the following stencil operations:

- 2D 4-points compact stencil (Jacobi).
- 2D 9-points compact stencil.
- 2D 9-points non-compact, second order stencil.
- 2D 5-points non-compact, asymmetric stencil.
- 3D 27-points compact stencil.

The main experimentation platform is the CTE-POWER cluster of BSC [13], comprising nodes with four NVIDIA Tesla V100 GPUs each. Each experiment executes 1000 iterations of each stencil. The array size for strong-scale experiments is 44000×44000, that is near the maximum size that fits in a 16GB V100 GPU using the Controller model. For weak-scaling analysis, the size of the array grows proportionally to the number of GPUs. Figure 1 shows that the proposed skeleton achieves good strong and weak scalability, except for the 3D stencil with enough GPUs. In this case, communications have much higher volume than in 2D, and their latencies cannot be completely ovelapped with the computations.

GPUs	Celerity avg. time (s)	Proposal avg. time (s)
1 Black	20.94	15.83
2 Black	11.99	9.12
2 Black + 2 V100	5.92	4.77

TABLE I: Performance comparison with a Celerity implementation of the 2D 4-point Jacobi stencil.

We also present a preliminary comparison of strong-scaling between the skeleton and a Celerity implementation of the 2D 4-point Jacobi stencil adapted to the last Celerity version from [14]. The experiments are conducted with facilities of the University of Valladolid, including one node with two NVIDIA TITAN Black GPUs, and one node with two NVIDIA V100 GPUs. The Controller model allows to use array sizes near to 28000×28000 in the TITAN Black GPUs. Nevertheless, for a fair comparison the experiments are performed with arrays of 18000×18000, that is near the maximum supported by Celerity. Each experiment is repeated 5 times. The results presented in table I show that our proposed skeleton achieves a higher performance than the Celerity implementation.

IV. Conclusion

We present a high-productivity parallel skeleton for iterative stencil computations in distributed multi-GPUs, built on top of a portable heterogeneous programming model. The experimental results show a good strong and weak scalability in a NVIDIA GPUs cluster of up to 16 GPUs in 4 nodes. The preliminary results also indicate that our proposal outperforms an example stencil implementation using Celerity. On-going and future work include more complete experimentation, exploiting temporal locality to improve performance and achieve scalability in the 3D case, and further improvements of the skeleton for more heterogenous clusters. The software, case study programs and experimental data can be downloaded at http://trasgo.infor.uva.es/controller/.

ACKNOWLEDGEMENT

This research has been partially funded by the Spanish Ministerio de Ciencia e Innovación with the ERDF program of the European Union, project PCAS (TIN2017-88614-R); "Nos impulsa" Junta de Castilla y León - FEDER Grants, projects PROPHET and PROPHET-2 (VA082P17, VA226P20); the Spanish Ministerio de Educación, Cultura y Deporte with a Beca de Colaboración Universitaria 2020/2021 (20CO1/013177); and by the program Salvador de Madariaga/Fulbright Scholar Grant (PRX17/00674).

REFERENCES

[1] NVIDIA, "NVIDIA CUDA Compute Unified Device Architecture Programming Guide Version 11.4.1," 2021. http://docs.nvidia.com/cuda/, Last visit: August 2021.

[2] K. O. W. Group *et al.*, "The OpenCL Specification, version 1.0. 29, 8 December 2008."

[3] M. Aldinucci, M. Danelutto, M. Drocco, P. Kilpatrick, C. Misale, G. Peretti Pezzi, and M. Torquati, "A parallel pattern for iterative stencil + reduce," *J. Supercomput.*, vol. 74, p. 5690–5705, Nov. 2018.

[4] S. Breuer, M. Steuwer, and S. Gorlatch, "Extending the skelcl skeleton library for stencil computations on multi-gpu systems," in *HiStencils 2014*, 2014.

[5] T. Shimokawabe, T. Aoki, and N. Onodera, "High-productivity framework for large-scale gpu/cpu stencil applications," *Procedia Computer Science*, vol. 80, pp. 1646–1657, 2016. International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.

[6] H. Tanaka, Y. Ishihara, R. Sakamoto, T. Nakamura, Y. Kimura, K. Nitadori, M. Tsubouchi, and J. Makino, "Automatic generation of high-order finite-difference code with temporal blocking for extreme-scale many-core systems," in *2018 IEEE/ACM 4th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*, pp. 29–36, 2018.

[7] B. Joó, T. Kurth, M. A. Clark, J. Kim, C. R. Trott, D. Ibanez, D. Sunderland, and J. Deslippe, "Performance portability of a wilson dslash stencil operator mini-app using kokkos and sycl," in *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pp. 14–25, 2019.

[8] P. Thoman, P. Salzmann, B. Cosenza, and T. Fahringer, "Celerity: High-level c++ for accelerator clusters," in *Euro-Par*, 2019.

[9] R. Keryell, M. Rovatsou, and L. Howes, "SYCL Specification - Generic heterogeneous computing for modern C++," 2020.

[10] A. Moreton-Fernandez, H. Ortega-Arranz, and A. Gonzalez-Escribano, "Controllers: An abstraction to ease the use of hardware accelerators," *The International Journal of High Performance Computing Applications*, vol. 32, no. 6, pp. 838–853, 2018.

[11] G. Rodriguez-Canal, Y. Torres, F. J. Andújar, and A. Gonzalez-Escribano, "Efficient heterogeneous programming with fpgas using the controller model," *The Journal of Supercomputing*, 2021.

[12] A. Gonzalez-Escribano, Y. Torres, J. Fresno, and D. R. Llanos, "An Extensible System for Multilevel Automatic Data Partition and Mapping," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1145–1154, 2014.

[13] BSC, "CTE-POWER User's Guide," 2021. https://www.bsc.es/user-support/power.php. Last visit: August 2021.

[14] B. Cosenza, "Test benchmarks for celerity," 2020. https://github.com/bcosenza/celerity-bench/blob/master/polybench/stencils/jacobi_2d.cc, Last visit: August 2021.