Extending Chapel to Support Fabric Attached Memory

Amitha C
HPC Business Group
Hewlett Packard Enterprise

Brad Chamberlain
HPC Business Group
Hewlett Packard Enterprise

Sharad Singhal
Hewlett Packard Labs
Hewlett Packard Enterprise

Clarete Crasta
HPC Business Group
Hewlett Packard Enterprise

Keywords: fabric-attached memory, Chapel, FAM distributed arrays

Abstract

This is a paper submitted to CUG-2022[12] and we intend to present the same content to CHIUW to seek feedback from the Chapel users group.

Fabric Attached Memory (FAM) is of increasing interest in HPC clusters because it enables fast access to large datasets required in High Performance Data Analytics (HPDA) and Exploratory Data Analytics (EDA) [1]. Most approaches to handling FAM force programmers either to use low-level APIs, which are difficult to program, or to rely upon abstractions from file systems or key-value stores, which make accessing FAM less attractive than other levels in the memory model due to the overhead they bring. The Chapel language is designed to allow HPC programmers to use high-level programming constructs that are easy to use, while delegating the task of managing data and compute partitioning across the cluster to the Chapel compiler and runtime. This abstract summarizes the approach to integrate FAM access within the Chapel language described in the paper [12].

Introduction & Problem Solved

HPC programmers generally write code that is aware of low-level details of the underlying hardware and cluster topology in order to achieve the highest possible performance. However, this makes programs hard to write, understand, and maintain over time. Programming models take a "library-based" approach, where much of the complexity is contained within libraries (e.g., MPI or SHMEM) and abstracted away from the application writer. The Chapel language [2], [3] pioneered at Cray Inc. and led by HPE, takes a different approach—it is a language designed for programmer productivity targeted at high performance computing; the programmer simply defines large data sets, and uses compiler constructs to indicate parallelism in the language. Like most other languages, Chapel was designed at a time when memory resources were intimately tied to the CPU. While it handles distribution of data and compute to compute nodes in a cluster, it does not currently support abstractions for disaggregated memory. HPE is developing a large-scale prototype for fabric-attached memory as part of a US government funded project, and multiple efforts are under way [4], [5] within HPE to understand how to best provide software to program FAM.

Our solution adds support for accessing FAM-resident data to the Chapel runtime and compiler with minimal language changes while ensuring the same level of abstraction and parallelism that currently exists in the language for data that is resident in compute node memory.

Implementation Architecture

A guiding philosophy in our solution is to minimize changes within Chapel, while ensuring existing Chapel programs are not impacted. While there are a number of ways FAM can be exposed within Chapel, our solution currently focuses on enabling distributed arrays that are resident in FAM.

Distributed arrays form an important aspect of large-scale programming on HPC clusters. While the programmer treats the array as a single large sequence of elements, Chapel actually distributes the elements of the array across nodes. The programmer can specify array distributions using Chapel language statements [3], [6]. Array distributions provide a "global view" of the array that allows programmers to operate on the array as if it was a local array, even though its data is distributed across locales internally. Since array indices are partitioned and distributed among locales, the operations on the distributed array are broken down into multiple tasks based on array partitioning, and

are assigned to the respective locales. Our solution takes advantage of abstraction and flexibility within Chapel to define a new distribution policy [7] under Chapel external modules, for array data resident in FAM.

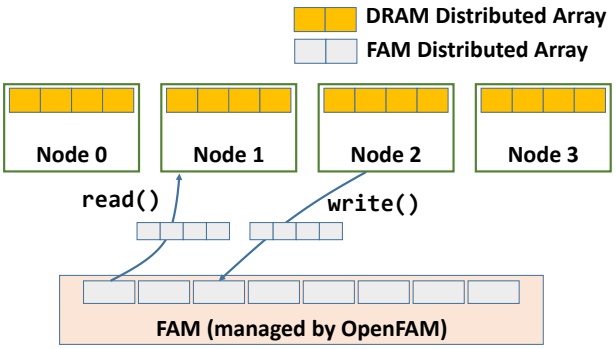

Figure 1: DRAM- and FAM-resident distributed arrays

Figure 1 shows a FAM-resident array with the target nodes participating in reads/writes to FAM in parallel. Unlike other distributed arrays in Chapel which reside in DRAM, FAM arrays can outlive the application. To support this, the FAM distribution module requires FAM arrays to be named during array creation. The FAM distribution module converts the high-level array operations into FAM-specific accesses underneath. When a FAM array is created, the complete array of the desired size is allocated on FAM by the current locale. Each locale is then assigned a partition upon which to operate. Parallel array operations such as forall, reduce or scan are internally divided into multiple tasks based on the partitioning, and executed in parallel by the target nodes. For FAM accesses, the distribution module internally calls the OpenFAM library [8] interfaces through the Chapel OpenFAM module. The FAM distribution uses the same domain partitioning policy as the Block distribution [6]. We envision that our solution would ultimately enable Chapel programs to operate on FAM arrays like any other distributed array in Chapel.

Evaluation

We have prototyped an initial proof-of concept code, which provides the following operations.

- Random access, Serial and parallel iterations including zippered loops on the FAM array.
- Bulk transfer – array to array assignment operations including both FAM and any distributed/vanilla array.
- Reduce, scan, slicing and re-indexing operations on the FAM array.

We have enabled multi-dimensional FAM distributed arrays and we now plan to evaluate the benefits of FAM distributed arrays through performance benchmarks.

Related Work

Other competitive approaches to disaggregation represent data in FAM using file-system [9] or key-value store [10] abstractions. Academic research literature also has published efforts supporting transparent paging to fabric-attached memory [11]. While attractive from a programming perspective, these approaches often require kernel modules and introduce large paging overheads. FAM access to Chapel applications cluster-wide can also be enabled at the application level through external libraries such as OpenFAM [8], or DAOS [9], but application developers are required to understand the APIs provided by those libraries, manage FAM and data distribution in FAM, and handle errors explicitly.

All of these approaches involve programming overhead, which contrasts with Chapel's philosophy of programmer productivity. Our solution leverages the OpenFAM library, and can be modified to leverage other libraries that provide access to FAM, while making FAM operations transparent to the programmer.

Conclusion and future work

We have put together a prototype, and are working through the different cases necessary in preparation for identifying all compiler and runtime changes in Chapel. Our PoCs use alternate approaches to enable early adopters to experiment with FAM using Chapel. Due to the alternate approach used in the implementation and dependencies from the compiler, there is a change in the semantics when performing certain operations on FAM arrays.

As next steps, we want to explore FAM usage in Arkouda [13], [14] and test the performance benefits to Arkouda or other Chapel application that can take advantage of FAM. We also plan to explore enabling FAM object classes; and other possible ways of exposing FAM to Chapel users. We anticipate that our experience will identify the requisite modifications needed in the language, compiler, and runtime, enabling us to make those changes open-source as a part of the Chapel language itself.

Acknowledgements

We would like to thank Sanish Suresh, Greg Titus, Elliot Ronaghan, Michael Ferguson, Shome Porno, Chinmay Ghosh and Dave Emberson for their contributions to this project.

References

[1] Peng, R. Pearce, and M. Gokhale, "On the Memory Underutilization: Exploring Disaggregated Memory on HPC Systems," in 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Sep. 2020, pp. 183–190. doi: 10.1109/SBAC-PAD49847.2020.00034

[2] Chapel project home page: https://chapel-lang.org/

[3] 'Chapel', Bradford L. Chamberlain, Programming Models for Parallel Computing, edited by Pavan Balaji, published by MIT Press, November 2015.

[4] B.K. Gautham et al., "Supporting Fabric-Attached Memory in OpenSHMEM", Submitted to Tech Con 2022, Phoenix, AZ, USA, Jun. 2022.

[5] J. Byrne et al., "FAM for Python Programmers," Submitted to Tech Con 2022, Phoenix, AZ, USA, Jun. 2022.

[6] Chapel: Standard Layouts and Distributions: https://chapel-lang.org/docs/modules/layoutdist.html

[7] B. L. Chamberlain, S. J. Deitz, D. Iten, S-E Choi, "User-Defined Distributions and Layouts in Chapel: Philosophy and Framework" in 2nd USENIX Workshop on Hot Topics in Parallelism (HotPAR'10), June 2010, Berkeley, CA, https://www.usenix.org/legacy/events/hotpar10/tech/full_papers/Chamberlain.pdf

[8] K. Keeton, S. Singhal, and M. Raymond, "The OpenFAM API: A Programming Model for Disaggregated Persistent Memory," in OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Extreme Heterogeneity, Cham, 2019, pp. 70–89. doi: 10.1007/978-3-030-04918-8_5.

[9] "DAOS and Intel® OptaneTM Technology for High-Performance Storage," Intel. https://www.intel.com/content/www/us/en/high-performance-computing/daos-high-performance-storage-brief.html (accessed Aug. 27, 2020).

[10] Y. Shan, S.-Y. Tsai, and Y. Zhang, "Distributed shared persistent memory," in Proceedings of the 2017 Symposium on Cloud Computing, Santa Clara, California, Sep. 2017, pp. 323–337. doi: 10.1145/3127479.3128610.

[11] Calciu et al., "Rethinking software runtimes for disaggregated memory," in Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, New York, NY, USA: Association for Computing Machinery, 2021, pp. 79–92. Accessed: May 14, 2021. [Online]. Available: https://doi.org/10.1145/3445814.3446713

[12] C, Amitha et al., "Extending Chapel to Support Fabric Attached Memory", Submitted to CUG 2022, Monterey, California, May. 2022. https://chapel-lang.org/tmp/CUG_2022_Chapel_v1.1.pdf

[13] Arkouda (αρκούδα): NumPy-like arrays at massive scale backed by Chapel. Bears-R-Us, 2021. Accessed: Aug. 29, 2021. [Online]. Available: https://github.com/Bears-R-Us/arkouda

[14] M. Merrill, W. Reus, and T. Neumann, "Arkouda: interactive data exploration backed by Chapel," in Proceedings of the ACM SIGPLAN 6th on Chapel Implementers and Users Workshop, New York, NY, USA, Jun. 2019, p. 28. doi: 10.1145/3329722.3330148.