A Team-Based Methodology of Memory Hierarchy-Aware Runtime Support in Coarray Fortran

Dounia Khaldi, Deepak Eachempati, Shiyao Ge, Pierre Jouvelot, Barbara Chapman

▶ **To cite this version:**

Dounia Khaldi, Deepak Eachempati, Shiyao Ge, Pierre Jouvelot, Barbara Chapman. A Team-Based Methodology of Memory Hierarchy-Aware Runtime Support in Coarray Fortran. IEEE Cluster 2015, Sep 2015, Chicago, United States. p.448-451, 10.1109/CLUSTER.2015.67 . hal-01251185

HAL Id: hal-01251185

https://minesparis-psl.hal.science/hal-01251185

Submitted on 5 Jan 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Team-Based Methodology of Memory Hierarchy-Aware Runtime Support in Coarray Fortran

Dounia Khaldi*, Deepak Eachempati*, Shiyao Ge*, Pierre Jouvelot† and Barbara Chapman*
*Department of Computer Science
University of Houston, Houston, Texas
{dkhaldi,dreachem,sge2,bchapman}@uh.edu
†MINES ParisTech, PSL Research University, France
pierre.jouvelot@mines-paristech.fr

***Abstract*—We describe how 2-level memory hierarchies can be exploited to optimize the implementation of teams in the parallel facet of the upcoming Fortran 2015 standard. We focus on reducing the cost associated with moving data within a computing node and between nodes, finding that this distinction is of key importance when looking at performance issues. We introduce a new hardware-aware approach for PGAS, to be used within a runtime system, to optimize the communications in the virtual topologies and clusters that are binding different teams together. We have applied, and implemented into the CAF OpenUH compiler, this methodology to three important collective operations, namely barrier, all-to-all reduction and one-to-all broadcast; this is the first Fortran compiler that both provides teams and handles such a memory hierarchy methodology within teams**

***Keywords*-Coarray Fortran; teams; PGAS; memory hierarchy; intra- and inter-node runtime; collective operations**

I. Introduction

The emergence of many-core computing nodes in large-scale distributed systems requires programming model implementers to consider more carefully memory hierarchy when looking at performance issues. Most parallel applications are programmed using the Message Passing Interface (MPI) [1], where multiple processes execute in a coordinated manner, communicating by performing *send* and *receive* operations. More recently, several languages and libraries have added support for explicit or implicit remote memory access (RMA) using so-called "one-sided communication", including languages following the Partitioned Global Address Space (PGAS) paradigm as well as MPI (MPI-2 added RMA to the interface and MPI-3 made significant refinements to better support it). Of special note is the Fortran 2008 addition for supporting *coarrays*, a language mechanism that enables RMA as a natural extension to Fortran's array syntax, informally named CAF[1]. In this paradigm, an *image* is an executing process in a Single Program Multiple Data (SPMD) program with its own copy of data.

The purpose of this paper is to provide new optimization strategies for coarrays. We suggest, first, to decompose applications into subproblems that may be worked upon concurrently, and organize this work among subsets of so-called image *teams*. These groups of images make it possible to divide applications into loosely-coupled subproblems that are handled by different subsets of images. Teams, already present for instance in MPI, are expected to be adopted in Fortran 2015. We, then, address the ensuing challenge of reducing the costs associated with moving data within a computing node and between nodes. Our approach is thus to combine a hierarchical decomposition of applications across two dimensions: (1) a logical partitioning in teams of the work, based on the application, and (2) a processor layout hierarchy, based on the underlying memory and communication hardware.

We applied this methodology to three common operations which may be executed collectively by a team of images in CAF: barrier, all-to-all reduction and broadcast (for these last two, see [2]). The classical algorithms for all three of these operations entail a fixed communication pattern among the images, making their total communication cost sensitive to the placement of images in parallel systems. Since a significant part of the execution time of an application is consumed while waiting for the completion of these operations, ensuring that their implementations are efficient, irrespective of image placement, is paramount. In this paper, we exploit the knowledge of the architecture that a run-time system can have, and describe, for the barrier collective operation, how we applied inside every team a new, two-level algorithm called *Team Dissemination Linear Barrier* (TDLB). Experimental evaluations using our implementation in the OpenUH compiler indicate that our proposed runtime awareness approach of memory hierarchy yields an up to 26-time execution time improvement over the basic dissemination algorithm for barriers [3].

The contributions of this paper are thus:

- the support of teams in Coarray Fortran within the OpenUH compiler;

[1]This acronym describes the Co-Array Fortran extension proposed by Cray Computer several years before it was adopted into the standard. We refer to the implementation of CAF in the OpenUH compiler as UHCAF. CAF 2.0 is an alternative Fortran language extension for supporting coarrays proposed by Rice University.

- the design of a two-step methodology for achieving better performance of collective operations, using runtime awareness of the memory hierarchy;
- the application of this methodology to the implementation of barriers, via the new TDLB algorithm, reductions and broadcasts; the novelty is to adapt existing techniques such as the dissemination algorithm to the PGAS model using one-sided communications;
- an evaluation of this methodology on two benchmarks: (1) our newly developed Coarray Fortran, CAF 2.0 and MPI (communicator concept) Teams Microbenchmark[2] suite [4] and (2) our porting to Coarray Fortran of the High Performance Linpack (HPL) benchmark [5], which uses teams.

We describe Coarray Fortran in Section II. The design and implementation of teams in the OpenUH compiler are introduced in Section III. In Section IV, we define our memory hierarchy awareness methodology and apply it to barrier, reduction and broadcast operations. Experimental results using our microbenchmarks and HPL are discussed in Section V. We survey other approaches for the implementation of teams in Section VI. We discuss future work and conclude in Section VII.

II. Coarray Fortran

Coarray Fortran is an explicitly-parallel extension of the Fortran 2008 standard that adheres to the PGAS programming model. Coarray Fortran programs follow an SPMD execution model, where all execution units, called images, are launched at the beginning of the program; each image executes the same code and the number of images remains unchanged during execution. Coarrays are shared data entities that are declared with the codimension attribute specifier and allocated collectively across all images. Subscripts of coarrays are specified with square brackets and provide a clear and straightforward representation of access to data on other images using 1-sided communication semantics. For example, the statement `A(:)[k] = B(:)` writes the elements of Coarray `A` on Image `k` with those of `B`.

Teams, added to Coarray Fortran, induce a hierarchical SPMD model. The initial team contains all the images. Subsets of images in a team may collectively form a new team, which are referenced using a handle of type `team_type`, using the `form team` statement. Every team has a unique identifier and a unique parent. Teams can be used to partition an application into different tasks executed by subteams. For instance, one can divide a logical grid into arbitrary subgrids. This could be used to group subsets of images performing computations on dense matrices into row- and/or column-oriented teams.

[2]Since teams are a relatively new concept for Coarray Fortran, there is no reference test suite for them; Teams Microbenchmark suite has been made publicly available for other implementers to get a baseline to compare themselves to.

Regarding performance, using teams, many collective operations can be overlapped; these collectives will work on just a subset of images; no global synchronizations among all the images are thus needed. Regarding memory, using Coarray Fortran teams, one can declare and allocate coarrays within a `change team` block. This allows a coarray to be allocated only in the images operating on it, thus utilizing more efficiently the available memory on each image.

III. Team Support in OpenUH

OpenUH [6], a branch of the open-source Open64 compiler suite that has been developed at the University of Houston, provides a solid base infrastructure for exploring implementation strategies for Coarray Fortran. The Fortran 95 front-end, originating from Cray, recognizes coarrays and parses the cosubscript syntactic extension.

We extended OpenUH to parse the `form team`, `change team`, `end team` and `sync team` constructs. We added the new type `team_type` to the type system of OpenUH and support for `get_team` and `team_id` intrinsics. We also adapted the CAF intrinsics `this_image`, `num_images` and `image_index` for teams.

During the back-end compilation process in OpenUH, team-related constructs are lowered to subroutine calls. In the runtime, we added a `team_type` data structure for storing image-specific information, such as the mapping from a new index to the process identifier in the lower communication layer. Also we provided support for team-related intrinsics, for example `get_team` and `team_id`.

Finally, we adapted atomic operations (`atomic_add`, `atomic_and`, etc.), synchronization operations (`sync images` and `sync all`), broadcast (`co_broadcast`) and reduction operations (`co_sum, co_max, co_min`) to work when executed by non-initial teams. Each subroutine works by using the global pointer to the current team, quickly obtaining the mapping of the image identifiers to the process identifiers in the `team_type` structure's image index mapping array.

IV. Memory Hierarchy and Teams

To make applications more scalable when running on nodes with many cores, the runtime should have some knowledge about the mapping of images on nodes and/or cores. If teams create subsets of images, there may be no simple relationship between the image structure and the actual underlying physical structure of the parallel system. Therefore, as a research methodology towards an efficient implementation of teams, we propose to introduce a memory hierarchy-aware runtime for PGAS, in order to optimize communications within teams via the distinction between local and remote memory accesses.

A. Methodology

To illustrate our methodology, we apply it to barriers, since the classic dissemination barrier algorithm is well-suited for distributed memory systems but not as efficient for the shared memory case ([2] also addresses reduction

and broadcast operations). In the dissemination algorithm, for n images, there are $n \log n$ synchronization notifications. On a shared memory system, in the worst case, all those notifications would have to be serialized. Contrast this with a centralized linear algorithm. There, only $2(n-1)$ notifications are needed, in two steps: first, notifications are sent from $n-1$ images to a dedicated, leader image; then, notifications proceed from the leader image to the $n-1$ slave images. Even if all those notifications are serialized, it is not as expensive as the dissemination algorithm.

If we consider instead a distributed system, where each of the n images is on its own node, then dissemination becomes faster. There are $n \log n$ total notifications, with n notifications performed in parallel in $\log n$ steps. For a centralized linear algorithm, everything would have to be serialized through a single node, so yielding $2(n-1)$ steps. These results are confirmed by Mellor-Crummey *et al* [7]. Our methodology will thus rely on detecting the images within a team that run locally on the same node (intranode set), assigning a leader for them and handling them with an intra-node strategy. After that, the leaders, which are on different nodes, are handled in a remote manner.

B. The TDLB Synchronization Algorithm

Our new memory hierarchy-aware barrier is a three-stepped, two-level algorithm: (1) a designated leader on each node waits for the remaining images on the same node to arrive at the barrier; (2) all leader images, one from each node with at least one image in the team, synchronize using a dissemination algorithm; and (3) each node leader notifies the remaining images on the same node that they may leave the barrier. Our Team Dissemination Linear Barrier (TDLB) algorithm is specified in Algorithm 1 (see [2] for details). Nodes' leaders are synchronized via the dissemination algorithm (fit for message passing), while synchronization within a node uses a linear barrier (well adapted to shared memory systems). Note that `cocounter` is a variable used in counter-based algorithms for barriers.

ALGORITHM 1: Team Dissemination Linear Barrier, run by each image in Team `team`

```
procedure TDLB(team)
 me = this_image(team)
 cocounter = team.cocounter
 ι_leader = get_leader(team, me)
 //Step 1: slaves synchronize with the leader
 linear_counter_1(team,me,ι_leader,cocounter);
 if (ι_leader == me) then
  pgased_dissemination(team, ι_leader);
  //Step 2: leaders notify their intranode set
  linear_counter_2(team,me,ι_leader,cocounter);
 end
```

V. Experimental Results

We ran the Teams Microbenchmark suite [4] and a porting of HPL on a cluster of 44 nodes connected via a 4xDDR InfiniBand (IB) switch, with dual quad-core AMD Opteron processors running at 2.2GHz on 16GB of main memory per node. We compared the implementation of these benchmarks in OpenUH 3.0.40 with two other implementations: (1) the source-to-source Rice CAF 2.0 compiler version 1.14.0, which uses ROSE [8] and GFortran 4.4.7 as backends, and (2) an MPI version, which we ran using both Open MPI 1.8.3 and MVAPICH 2.0beta. Both OpenUH and Rice CAF 2.0 implementations rely on GASNet's Infiniband verbs runtime implementation. We used GASNet 1.22.2.

A. Teams Microbenchmarks

We applied the methodology of Section IV-A to barrier operations (see [2] for all-to-all reduction and one-to-all broadcast operations). The performance of our implementation of teams for these collectives is assessed via our microbenchmarks. Contrarily to Algorithm 9 in [7], which relies on two synchronization arrays for its implementation of a barrier operation, and the one described in [3], which is using two waits, our dissemination algorithm is based on a `sync_flags` carry, thus taking advantage of the features of a PGAS model with only one wait.

We compared TDLB with (1) the GASNet RDMA dissemination algorithm, which uses put operations to implement the dissemination barrier described in [7], (2) the GASNet IB dissemination algorithm, which directly uses Infiniband verbs for communication to implement the same algorithm, (3) CAF 2.0, which uses also the dissemination barrier described in [7], (4) MPI using `MPI_Barrier` of MVAPICH, default Open MPI and Open MPI with the hierarchy-awareness options (`hierarch` and `sm` modules) and (5) the current version of UHCAF, which uses the pure dissemination algorithm described in [7].

Even though TDLB is portable and can be used with any communication layer or conduit, our experiments show that (1), with one image per node, it performs as well as a pure dissemination algorithm in the case of a flat hierarchy and (2), with 8 images per node, it is well optimized to handle the memory hierarchy and is only marginally more expensive than the low-level dissemination algorithm implemented directly over the IB verbs that GASNet provides.

B. HPL

We implemented a Coarray Fortran version of HPL [5], which solves systems of linear equations, thus testing temporal and spatial run-time localities. We based our version of HPL on its CAF 2.0 port [9]. HPL makes use of row teams and column teams for performing updates of the matrix data.

Figure 1 compares the performance results using the two-level approach in UHCAF, the one-level approach in UHCAF, CAF 2.0 using GFortran as backend, CAF 2.0 using OpenUH as backend compiler and Open MPI using GCC compiler. These preliminary results suggest that the two-level approach in UHCAF provides up to 32% improvement over a typical one-level approach. Overall, we obtained 95 GFLOP/s on 256 cores, as compared to 29.48 (GFortran

backend) and 80 (OpenUH backend) with CAF 2.0.

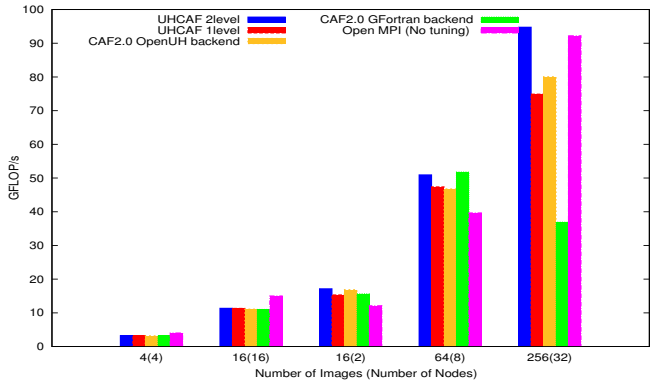

Figure 1: Performance results for HPL (`-O3` option)

VI. Related Work

CAF 2.0 [10] included teams as first-class objects since its inception (see [2] for more related work details). OpenSHMEM [11] proposes the concepts of teams and spaces in order to allow allocation of memory only across subteams. None of these two works provided any memory hierarchy information for teams. We combine team-based image grouping with an awareness of the memory hierarchy.

The notion of grouping and hierarchy is present in other programming paradigms. For instance, the MPI-2 specification allows for RMA within a group of MPI processes represented by a communicator through the mechanism of window creation [12]. Techniques to support the scalability of communicators and groups in MPI are presented in [13]. Our work parallels this approach, within the PGAS framework. We believe that the team-based, one-sided communication model of Coarray Fortran is easier to handle by programmers than the grouping semantics of MPI (and its need for windows). Our work shows that better programmability does not come at a cost, since our implementation of Coarray Fortran is competitive with MPI.

VII. Conclusions and Future Work

We proposed a PGAS-based design methodology for supporting efficient communication between teams for Coarray Fortran that takes into account the memory hierarchy of clusters to efficiently implement collective algorithms. We showed how the memory hierarchy can be exploited at run time to optimize team implementations via the distinction between local and remote memory accesses.

To evaluate our memory-hierarchy approach, we extended the implementation of Coarray Fortran within the OpenUH compiler with teams and applied our methodology to barrier, reduction, and broadcast operations, getting up to, respectively, 26-, 74- and 3-fold performance improvements over the default approach. We also got better performance results on HPL compared to the one-level approach and original CAF 2.0 version we based our's on.

Future work will look at how our methodology can support multi-level hierarchies to represent different network topologies or on-node locality domains such as NUMA memory nodes, shared caches, processor sockets and cores.

References

[1] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI-The Complete Reference, Volume 1: The MPI Core (2nd Edition)*. Cambridge, MA, USA: MIT Press, 1998.

[2] D. Khaldi, S. Ge, D. Eachempati, P. Jouvelot, and B. Chapman, "A Team-based Design Methodology for Memory Hierarchy-Aware Runtime Support in Coarray Fortran," MINES ParisTech, Technical Report, Tech. Rep. E/376/CRI, Jul. 2015.

[3] D. Hensgen, R. Finkel, and U. Manber, "Two Algorithms for Barrier Synchronization," *Int. J. Parallel Program.*, Feb. 1988.

[4] "HPCTools Teams Microbenchmarks," https://github.com/dkhaldi/teams_microbenchmarks.

[5] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. http://www.netlib.org/benchmark/hpl/.

[6] B. Chapman, D. Eachempati, and O. Hernandez, "Experiences Developing the OpenUH Compiler and Runtime Infrastructure," *Int. J. Parallel Program.*, vol. 41, no. 6, pp. 825–854, Dec. 2013.

[7] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for Scalable Synchronization on Shared-memory Multiprocessors," *ACM Trans. Comput. Syst.*, vol. 9, no. 1, Feb. 1991.

[8] D. Quinlan, "ROSE: Compiler Support For Object-Oriented Frameworks," *Parallel Processing Letters*, vol. 10, 2000.

[9] J. Guohua, J. Mellor-Crummey, L. Adhianto, W. Scherer, and C. Yang, "Implementation and Performance Evaluation of the HPC Challenge Benchmarks in Coarray Fortran 2.0," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, May 2011, pp. 1089–1100.

[10] J. Mellor-Crummey, L. Adhianto, W. N. Scherer, and G. Jin, "A New Vision for Coarray Fortran," in *Proceedings of the Third Conference on Partitioned Global Address Space Programing Models*, ser. PGAS '09. New York, NY, USA: ACM, 2009, pp. 5:1–5:9.

[11] A. Welch, S. Pophale, P. Shamis, O. Hernandez, S. Poole, and B. Chapman, "Extending the OpenSHMEM Memory Model to Support User-Defined Spaces," *PGAS 2014*, oct 2014.

[12] A. Moody, D. Ahn, and B. Supinski, "Exascale Algorithms for Generalized MPI_Comm_split," in *Recent Advances in the Message Passing Interface*, ser. Lecture Notes in Computer Science, Y. Cotronis, A. Danalis, D. Nikolopoulos, and J. Dongarra, Eds. Springer Berlin Heidelberg, 2011.

[13] H. Kamal, S. M. Mirtaheri, and A. Wagner, "Scalability of Communicators and Groups in MPI," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 264–275.