PGAS for (In)coherent Manycore Systems

Robert Kuban

Abstract As the architectural walls drive the number of parallel cores in multicore systems up, it becomes harder to maintain cache coherence across all of the physical memory and all cores. On the other hand, even given a performant cache-coherent system, the unavoidable non-uniform memory (NUMA) and non-uniform cache (NUCA) architectures make programming for it difficult. A potential solution to this problem is to interpret massive multicore machines as a distributed system with remote memory access, and therefore, use existing distributed programming models. A natural fit for such an approach is the PGAS model, which provides a global address space divided into partitions that can be either local or remote. Unfortunately, completely discarding the notion of sharing ignores the efficient hardware mechanisms available in multicore machines with shared memory. This survey examines PGAS frameworks and communication libraries with an focus on the PGAS model to enable PGAS applications to exploit shared memory in massive multicore machines without sacrificing the benefits of the PGAS programming model.

1 Introduction

As processor manufacturers face the power wall [13], further performance can mostly be gained through an increase in parallelisms. An extreme of this can be seen in manycore processors and accelerators that will happily trade core-complexity for a larger number of cores, sacrificing single core performance, but improving overall throughput. Eventually, architectures using full-featured cores started to adapt the interconnect designs employed in manycore architectures, increasing the number of cores per socket dramatically.

On the other hand, parallel architectures are notoriously hard to program for. As serial performance of a single core does not increase any more, processores require highly parallel applications, runtimes, and operating systems to exploit the large number of cores. This poses the question how applications for manycore systems should be designed, and what programming model they should use. Models based on a global address

This work was supported by the German Research Foundation (DFG) under grant no. NO 625/7-2.
DOI: 10.26127/BTUOpen-5618

space become more prevalent even in distributed systems, as the low-level interface between tightly-coupled system components has converged towards memory access: RMA has become more widely supported in networks, PCIe basically routes memory accesses between devices, and high-performance cache-coherent interfaces integrate accelerators tightly into the physical address space of the host machine.

Programming with a global address space can be misleading if the underlying memory is not uniform, hence, has not the same properties regardless of the address. The Partitioned Global Address Space (PGAS) model addresses this problem by dividing the global address space into partitions with different properties. In common PGAS languages, each partition is local memory corresponding to a core, socket or cluster node. By making the locality of the memory visible, the PGAS model enables programmers to reason about the locality of their program. However, by discarding the notion of shared memory, PGAS frameworks can not fully exploit hardware mechanisms for replication available in multicore machines with real shared memory and shared caches.

This survey examines PGAS frameworks and communication libraries with an focus on the PGAS model to enables PGAS applications to exploit shared memory in massive multicore machines without sacrificing the benefits of the PGAS programming model. Section 2 examines a selection of manycore and massive multicore architectures. Section 3 provides a short overview over the PGAS ecosystem. Section 4.1 looks into different PGAS frameworks. Section 4.2 examines communication libraries used by PGAS frameworks. Section 5 surveys the memory models typically provided by PGAS frameworks and communication libraries. Section 6 looks into how shared data and replication is handled in PGAS frameworks and what kinds of optimisations are applied by PGAS languages. Section 7 summarizes the lessons learned from PGAS frameworks and formulates goals for extending the PGAS programming model for better shared memory support.

2 Manycore and Massive Multicore Systems

As single core performance is limited by the architectural walls, the number of cores in a system must increase in order to increase the overall performance. This section looks into different options how an increasing number of cores is integrated into manycore and massive multicore systems.

Chip-level Early single-chip manycore processors including the Tilera [10] and the Single Chip Cloud Computer (SCC) [72] use a mesh interconnect topology. Whereas the Tilera uses a distributed cache architecture, the SCC is not cache coherent.

Later commercially available manycore processors tend to provide coherent caches. The XeonPhi Knight Corner processor uses two bidirectional rings as an interconnect [75], but the XeonPhi family switched to a mesh topology with the next generation (Knights Landing), using bidirectional half-rings along each dimension [134]. Later the mesh-based interconnects found its way into processor architectures with fully-featured cores. A good example for this is the Intel Skylake-SK processor, which adapted a mesh

interconnect in contrast to its ring-based predecessor [137]. Based on the hop count between different processors, cache directories and memory controllers, NUMA-like effects can appear even on a single die. Intel's Sub-NUMA Clustering [134] clusters cores with memories controllers and their assigned cache directories in order to expose this locality to the programmer.

Package-level To increase the number of cores per package without scaling up the number of cores per die, one can disintegrate a processor into chiplets by fitting multiple dies into a package. Due to the lower die area per chiplet, this increases yield in production compared to larger single chip processors [80].

The AMD Zen Architecture uses chiplets called "Zeppelin" [9] as the basic building block, which integrates 8 cores on a single die. Up to 4 of these chiplet are integrated into into one package, enabling up to 32 cores in a 2 socket system. Package-level integration can also be used to integrate local die-stacked memory into the package[80], potentially lessening the effect of the memory wall.

System-level Classical NUMA configurations connect up to 8 different processors with each other throught a concurrent point-to-point interconnects, like UPI in Intel processors [109] or AMDs Infinity Fabrik Global Memory (IF GMI).

The NUMAlink technology used in HP Hyperdom Servers [74] enables to scale NUMA to more processors than originally supported by the processor vendor. NUMAlink does this by replacing one processor in the supported configurations with a router that manages concurrency and address translation between boards. However, it is hard to see where to draw a line to classical shared memory supercomputers at this level.

Currently there are a number of compiting standards and proporitary interfaces that enables coherent memory between hosts and accellerators, such as CLX [37], OpenCapi [115], NVLINK [114], CCIX [27], and RapidIO [121].

Conclusion The system architecture increased the amount of cores on each level: more cores on each die, multible dies in one package, large-scale NUMA systems, and support for manycore accelerators. However it has become clear that, despite its costs in large scale systems, consistent shared memory is not given up upon. If anything, standardizations efforts on cache coherent interfaces enable accelerators to participate in the cache coherence protocol and benefit from fast access to cached data.

3 PGAS

The PGAS model is described fairly good by it name. It is a *global address space* that is explicitly *partitioned* into *places* to represent locality. *Places* consist of execution units as well as memory. From the view of a place, the memory can either be *local* (at the same place), or *remote* (at any other place). Each place may also have *private* memory that is only accessable locally, hence, from the same place. In general, all execution units in a places have a relatively uniformly view on the resources of the system [28]: For example,

layer	role	example
application	problem solving	page rank, fluid dynamics
domain-specific library	high-level operations	linear algebra, graph representation, stencil operations
PGAS framework	programming model	global address space, locality and concurrency control
communication lbraries	hardware abstractions	(active) messages, remote memory access

Table 1: Potential layers in a PGAS application.

in a cluster of NUMA machines, memory access latencies can differ depending on the NUMA domain. However, these latencies may are relatively small compared to remote memory access through the network.

As a model, PGAS is a reasonable compromise between implicit locality, and exposing all available information to the programmer. For example, the hardware locality library hwloc [19] provides a very detailed picture of the memory hierarchy and locations of execution units of a system. Unfortunately, it is hard to derive meaningful decisions about data distributions on application level without condensing this information into a more simple model. On the other hand, models that do not provide an abstraction for locations are unable to allow the programmer to control or reason about data distribution. The trade-off between programability and performance is typically called productivity, and has been studied as an argument for the adoption of PGAS languages [24, 51, 149, 21].

The PGAS programming model can be implemented on language- as well as on library-level [155]. In the following, the term *PGAS framework* will be used to include both PGAS languages and library-based PGAS implementations. Naturally, PGAS frameworks do not exist in a vacuum, but are used by an application. Table 1 lists different layers potentially found in a PGAS application. PGAS frameworks are typically implemented on top of a communication library, that abstract from the concrete hardware, operating system details, or vendor-specific network interfaces. The PGAS framework further abstracts from the concrete machine by implementing the global address space, locality and concurrency control. On top the PGAS framework, a well-structured application may consist of domain-specific libraries, that use the PGAS framework to implement primitives that are used by a concrete application. This may include linear algebra operations, graph processing frameworks, or stencil operations. In reality, unfortunately, this layer might be convoluted with the actual application. Finally, there is an application layer that formulates a concrete problems in terms of the high-level operations. For example, the application may load and process a graph representing website links in order to calculate a page rank.

De Wael et al. [48] have classified PGAS languages considering their model for parallelism, the topology, the data distribution and the access to remote data. For par-

allelism, they differentiate between implicit parallelism, single program multiple data (SPMD), and the asynchronous partitioned global address model (APGAS) [122]. For the topology, they differentiate between a flat topology (with numbered partitions), an user-defined mesh (flat topology with an mapping from the mesh nodes), and a hierarchical topology (partitions are partitioned further). Data can be distributed implicitly by the runtime or explicitly by the user. The data itself can be regular, such as arrays divided up into same sized blocks, or irregular. Finally, access to remote data can either be implicit, or must be made explicit by the programmer.

In the following, partitioned global address space implies that the global partitions are visible and directly accessable to the programmer. This excludes *array languages* such as ZPL [98] and HPF [101], as well as *distributed shared memory* (DSM) systems and *distributed object frameworks* with implicit object distribution.

DSM Software-based distributed shared memory (SW-DSM) started with the page-based DSM Ivy [97]. In current high-performance computing runtimes, SW-DSM is often an ingredient of distributed application runtimes and facilitates the data movements. Legion [8] has a task oriented programming model and is implemented with on top of Realm [140]. Realm manages data in *physical regions*, which can have multiple instantiates with the same memory layout. Legion enforces its memory model via task ordering. The StarPU [4] runtime is supported by a distributed shared memory. It supports multiple concurrent readers and access to partitions of data using filters and enforces consistency with a Modified/Shared/Invalid protocol. HPX [78] uses an Active Global Adress Space (AGAS), in which active messages can be send to migrateable logical partitions. Charm++ [79] also provide logical partition, which can be made are migratable by user-provided serialisation functions [1]. It also supports distributed objects.

Distributed Objects The overlap between distributed object frameworks and PGAS frameworks is obvious, for example, in the RMI middleware TACO [113], which have an explicitly partitioned object space and enables RMA over a global array pointer. However, most PGAS frameworks tend to focus on regular data structures, data parallelism and single-sided remote access (RMA), whereas distributed object frameworks tend to focus on (irregular) objects, remote method invocations and their semantics.

4 Data Access

In distributed systems, data can in general accessed in two ways: Data-shipping copies the data to the location that want to access it, whereas function-hipping transfers the control flow to the location of the data.

The two-sided model for communication generally provides two operations: `send` and `receive` (`send`/`recv`). `Send` expects a user-provided buffer as an argument, which is transmitted to the receiver. Analogously, the receiver uses the `receive` function to determine a buffer where the received message should be copied into.

In systems with local shared memory, memory can be mapped (`map`) into the address space of a process. Afterwards, the process can access the memory like its private memory, hence, directly through the processors instruction set.

`Put` and `get` are the operations associated with the one-side model for distributed computing. `Put` writes to a global address, whereas `get` loads from a global address. In distributed systems, these operations can be implemented using hardware supported remote memory access (RMA).

An active message consists of data to be transmitted and a handler to be executed at the receiver. One can distinguish between different kinds of messages [105]: Short messages only consisting of arguments for the handler. Medium messages can name a buffer from which data is copied, the destination buffer is chosen by the communication library and is only valid during the execution of the handler. For long messages, the sender determines both the source and destination buffer.

Atomic operations (atomics) are operations that appear as if they are executed instantaneously, hence, they can not be observed in a transitional state. Often these operations have (optional) memory ordering semantics, which allows to to use them for synchronization.

4.1 PGAS Frameworks

This section aims to give an overview over several PGAS framework, and will look at the most popular one in more detail. De Wael et al. [48] categorize the PGAS languages historically as *retrospective PGAS languages* such as HPF, ZPL, and GA; *original PGAS languages* such as CAF, Titanium, and UPC; *HPCS PGAS languages* such as Chapel, X10, and Fortress; and *recent PGAS languages* such as XCalableMP. Here, we will focus on a subset of languages that is still maintained and used outside of example code. However, this choice is somewhat subjective and exclusion does not imply a language is unmaintained or without real-world application.

When aiming to backtrack the evolution of the PGAS frameworks, it is helpful to follow the contributions of the UC Berkely, which had a great influence on their development. Most notably are: active messages [53, 42, 105], the early parallel C extension Split-C [41], the PGAS languages Titanium [150] and Unified Parallel C (UPC) [25], the communication middleware GASNet [14], and the PGAS library UPC++ [155].

Unified Parallel C (UPC) is one of the original PGAS languages, provides a SPMD programming model with explicit distribution and implicit remote access [48]. There is a huge diversity of UPC compilers and runtime environments. *Berkeley UPC*[11] is basically the reference implementation of UPC and implements UPC 1.3 [143]. Its runtime can be used with different compilers and source-to source translators, for example Gnu UPC [64] and the Clang UPC Toolkit[36]. However, other runtime implementations are available: the GCC UPC Portal4 runtime [56], the MuPC UPC run time system [124, 153], and ScaleUPC [154], which targets only shared memory multiprocessors.

HP UPC has been the first commercial available UPC compiler, but is now discontinued. Its final version is 3.3 [120] and implements UPC 1.2 [142]. Further vendor

	send/recv.	map	put/get	active msg.	atomics
PGAS frameworks					
UPC	-	-	x	-	> 1.3
UPC++	-	-	x	> 2.0	x
X10	-	-	-	x	-
Chapel	-	-	x	x	x
XcalableMP (XMP)	-	-	x	-	x
communication libraries					
MPI	x	> 3.0	> 2.0	-	> 3.0[1,2]
OpenSHEM	-	-	x	-	x[1]
GASNet(EX)	-	-	x	x	$>$ EX
ARMCI	-	-	x	-	x[2]
UCCS/UCX	x	x	x	x	x
GASPI	x[3]	-	x	-	x[2]
shared memory					
POSIX SHM	-	x	-	-	x[1]
XPMEM	-	x	-	-	x[1]
LiMIC	x	-	-	-	-
KNEM	x	-	x	-	-
CMA	-	-	x	-	-

Table 2: Data access in different middleware

[1] via instructions on mapped memory
[2] via API call
[3] for passive communication

implementations of UPC include the UPC Cray Compiler [40], HP-X UPC[73] by Mellanox and XL UPC for the IBM PERCS architecture [138].

UPC++ [155] is a library-based C++ PGAS framework from the University of Berkeley. It uses futures [7] to allow the programmer to overlap computation and communication. Habanero UPC++ [91] is a extension that integrates UPC++ with the HabaneroC++ local workstealing scheduling framework.

X10 [34] is a PGAS language originating from IBM's entry to the DARPA High Productivity Computing Systems (HPCS) project. It coined the term APGAS [122], that is an execution model where asynchronous task-based parallelism is combined with the PGAS model. X10 has become adopted by the research community with a relatively high number of publications that build on the language, for example with aggregating synchronization primitives [130] or global work balancing [152]. There also is a wide variety of applications written in X10.

The APGAS runtime [139] inherits a fault-tolerant programming model developed for Resilent X10 [43], whereas the Habenero Java language [26] is a less closely related hybrid between X10 and Java.

Chapel originates from Crays entry [22] to the HPCS project. Interesting about Chapels design is especially the clear separation between locality and concurrency, as well as the concept of *multiresolution programming* [30]: The language provides reasonable defaults for implicit distribution and concurrency, but allows the programmer to control both explicitly if necessary. Since the end of the HPCS program, Chapel has significantly matured [29], improving the performance as well as productivity features.

XcalableMP (XMP) [95, 141] is a pragma-based framework, which supports an OpenMP-like [44] data-parallel global view on data structures, the OpenMP dependency-driven task interface [63, 141], but also a partitioned global address space for explicit location control. Unlike other PGAS frameworks, global data structure are always accessed locally. To maintain consistency, the `reflect` annotation exchanges replicated *shadow* areas of global data structures. XcalableMP is a successor of OpenMPD [96], which has a similar concept for global data structures, but uses MPI as its location-aware communication model.

Trends In general, a rise of PGAS libraries can be observed, for example the APGAS library for Java [139]. For C++, there is UPC++ [155], DASH [57, 55] and other PGAS libraries [52]. Interestingly, the more recent PGAS frameworks X10 and UPC++ shift their focus from actual data distribution to providing a object-based toolkit for data access and synchronization. The PyGAS [50] Python PGAS extension provides transparent object proxies and generally resembles a distributed object framework. Another trend is the support of memory regions that are not in a 1-to-1 relation to places [132, 6, 84]. These regions allow the PGAS framework to model, for example, accelerator memory.

4.2 Communication Libraries

Communication libraries are a wide topic which spans a spectrum between thin software layers interfacing with the hardware and frameworks with a programmability comparable to a complete PGAS language. Here, we will examine three different levels of communication libraries: High-level libraries that provides complete environment that facilitates programming without additional productivity layers. Intermediate-level libraries often abstract from specific hardware and vendors and are targeted at the implementer of higher-level frameworks. Low-level communication libraries targets a specific hardware or hardware of a specific vendor. For low-level libraries, we will look into low-level API for shared memory in more detail.

High-level libraries MPI is the standard for distributed computing focusing mainly on two-sided communication. With the latest versions, MPI has added support for one-sided communication. Whereas first iterations where not well-suited for the implementation of PGAS frameworks [17], support for one-sided communication has improved considerably [71, 49]. Although MPI is often used in a BSP-style [145] programming model, it is common to see other programming models, e.g. PGAS, or other middleware implementated on top of MPI [17, 153, 45, 156] In the last years, MPI has adapted to the changing environment towards multicore clusters and remote memory access: MPI windows allow access to remote memory [108] via `put` and `get` operations and provide a mean to access allocate memory shared between multible ranks of one cluster node [70, 108]. There is a number of MPI implementations, but most notable are MPICH2 and OpenMPI, and vendor-specific derivatives.

The SHEM libraries stems from an environment of more tightly integrated supercomputers. In consequence, SHMEM enables one-sides communication and features a symmetric heap model. OpenSHMEM [33, 116] is a standardization effort for SHMEM, its implementation is based on the lower-level library UCX. Similar to MPI, SHMEM is widely supported by different vendors. GPSHMEM is a SHMEM implementation on top of ARMCI [118]. There are also attempts to use SHMEM as the programming model for manycore environments (TSHMEM [92, 93]).

Intermediate-level libraries More on the lower end of the spectrum, there are libraries like GASNet [16], its successor GASNetEX [18, 59], ARMCI [110] and GASPI [132].

GASNet(EX) is build around the idea of active messages, transferring data but also carrying out an operation after the data is transferred. The extended API of GASNet also provides `put` and `get` functions that perform RMA without the need to execute a message handler.

GASPI [132] emerged from an effort to standardize GPI [67]. GPI has evolved from the Fraunhofer Virtual Machine (FVM) [103]. In contrast to other communication libraries, communication functions are designed with timeouts in order to enable robustness to node failures. Remote access is carried out relative to segments in the global address space without imposing an specific memory model. Its implementation GPI-2 specifically targets Infinibands interconnects as well as accelerators. Although GASPI can

replace MPI, it can also be used as a complemement to accelerate time-critical compute kernels [133].

ARMCI [111, 110] is designed as a one-sided memory copy engine to be used as a supplement to MPI. It is based on generalized I/O vectors, but also provides mutexes for synchronization, memory allocation functions, and atomic accumulate operations. As the communication middleware of global arrays [112], ARMCI has been replaced by ComEx [45], which provides a similar API, but supports a wider set of platforms by providing a MPI backend.

UCCS [127] and its successor UCX [126] aim to unify communication libraries for different programming models by providing a toolbox consisting of active messages, RDMA primitives, atomic operations, collective operations, and wrappers for bootstrap and runtime environments (RTE).

Vendor- and hardware-specific libraries libfabric [66] is a core component of the OpenFabrics Interfaces (OFI) and aims to be used by higher level frameworks like MPI or SHMEM. LAPI [125] for PowerPC, PAMI [90] for IBM BlueGene/Q supercomputers and Crays DMAPP [20] are both vendor-spezific middleware, which can be used to implement higher level or vendor-independed middleware. There are communication libraries like Mellanox MXM, Cray GNI, Intel PSM. These libraries often provide a additional services above hardware features, such as tag-matching, and abstract from different hardware of a single vendor.

Shared memory libraries Infrastructure for communication libraries also includes APIs for intranode shared memory communication. The POSIX Shared Memory API and the XPMEM [147] kernel module both can be used to establish shared memory regions between processes. XEMEM [83] is a extension to XPMEM that provide a name service and allow communication between different virtual machines or containers on a shared memory machine. LiMIC [77] enables tag-matching two-sided single-copy memory transfer implemented as a Linux kernel module. KNEM [65] is a kernel module that provides safe single-copy one-sided RDMA-emulation between processes, including collective memory transfers [102]. CMA [146] is part of the Linux kernel since version 3.2 and allows one-sided access to memory of other processes.

5 Memory Models in PGAS

A memory model is a contract between the programmer and a language concerning the order and visibility of memory accesses. For context, this section first gives an introduction into memory models. The rest of this section discusses the memory models found in PGAS frameworks and communication libraries.

Sequential Consistency (SC) [94] guaranties that a global total ordering of every memory access exists and that order is an interleaving of the programm orders of the processes involved. The programm order is the order of memory accesses as written down in the program code. Therefore, algorithm written in SC are relatively easy to reason about.

Unfortunately, establishing a global order of all accesses between all processors is expensive. Consequently, Processor Consistency or Pipelined RAM (PRAM) [99] allows reordering read accesses before independent write accesses, which allows processors to buffer writes in a local queue. Additionally, not every order given in program order is meaningful, hence, only a portion of programm order constraints are nessecary to enforce the correct program semantics. Hybrid memory models [107, 61] label memory accesses to synchronizing variables, which are in turn used to establish order between non-sychronizing accesses. Release consistency [61] is a family of memory models that prominently uses `aquire` and `release` labels. For distributed shared memory, the further relaxed *lazy release consistency* [82] is more popular, as it does not require eager distribution of data. Location consistency [58, 100] is a hybrid memory model designed for systems with incoherent caches.

Finally, a important property of memory models for programming languages is *sequential consistent for data race free* (SC4DRF). It means that if a programm does not have any data races, it behaves as if all accesses are sequentially concistent. Modern programming languages (C++ [12], Java [106]) have converged towards a memory model with this property as a compromise between performance on current hardware and programability. The concept of programs being "synchronized enough" has been first introduced with the data-race-free-0 (DRF0) [2] memory model.

5.1 Frameworks

The memory model of PGAS frameworks often starts out underspecifed und improves as implementors and user explore the edge cases. A good example for that is the UPC memory model: Some early work towards a clear definition can be found in the master thesis of Kuchera [87], followed by tech reports about a programmer-friendly specification based on abstract state machines [89] and illustrative test cases for the model [88]. After a tech report [148] and further discussion [86], the current formal definition can be found in the UPC specification [143, §B]: Access to shared variables in UPC can be either *strict* or *relaxed*. All strict accesses are totally ordered, hence, a program consisting only of strict memory accesses is sequential consistent. Two accesses are ordered if they are issued by the same thread and at-least one of them is strict. Finally, dependent accesses are ordered within each thread. In result, the UPC memory model guaranties *sequential consistency for data-race-free* executions. The semantics of library calls is described in terms of strict memory accesses to a synchronization variable. Non-collective memory operations have semantics similar to weak memory accesses, whereas collective operations may have fence semantics depending on the arguments. Atomic operation can have either weak or strong semantics.

The UPC++ memory model [5] extends the C++11 memory model by specifying the happens-before and sequenced-before relations for the global effects of UPC++ library functions, and therefore, remote memory accesses. UPC++ supports atomic `get`, `put` and `fetch-and-add` operations for 32 and 64 bit integer types. These operations support C++ memory order descriptions.

There is very few documented work on memory models for X10. Most importantly, no

model is given in the specification [123]. Similarly, no primitive atomic operations are specified. A proposed X10 memory model [157] addresses local memory ordering, and assumes that remote accesses will be ordered by activity creation and synchronization.

The Chapel memory model [32] is motivated by multi-resolution programming. In order to guarantee that the correctness of algorithms does not change regardless of the data distribution, the memory model can not differentiate between remote and local accesses. It is inspired by XC [135], and the C++11 memory model [32, §29.2]. Consequently, the atomic operations also mirrow the C++11 atomics. Internally, this high level model model is mapped to model based on fences [54], reminiscent of release consistency [61]. An `acquire` fence ensures that no stale data from the cache is returned. The `release` fence conceptually writes back dirty entries in the cache, hence, making writes observable to the outside.

5.2 Communication Libraries

Poole et al. [119] have surveyed remote and local completion and ordering primitives with a focus on SHMEM implementations. SHMEM provide a `fence` operation that guaranteed that, for every remote process, every store issued before the `fence` is completed before any store issued after the fence. The `quiet` operation is stronger, as it guaranties that any store issued before `quiet` is completed before any store issued after it. Some SHMEM implementations additionally guarantee remote completion for `fence` and `quiet`. A `barrier` has the usual semantics for synchronization, however, some SHMEM implementation guarantee remote completion for any outstanding store.

In contrast to the SHMEM semantics, modern PGAS framework are often oriented on the (a)synchronous GASNet semantics, which did not seperate *ordering* from *remote completion* [28]. Hence, the only way to order outstanding stores is to wait for remote completion. As of version 1.8.1 [15], GASNet does not support atomic operations. However, GASNetEX [6, 18] adds support for atomic domains as introduced by [143]. Atomic domains select a implementation for atomic operations based on a datatype, requested operations, and hardware support.

For GASPI [132], there is an accepted proposal [85] demanding sequential consistency between ordinary access to a local memory segment and GASPI functions executed by the same thread. However, this does not imply any global memory ordering, which must by archived by using queues and synchronization via notifications. All operation in a queue are completed locally after *waiting* on a queue. This waiting does not guarantee remote completion [60]. Furthermore, all operation in a local queue with the same remote are completed in order. This includes *notifications*, which can therefore be used to wait for completion of a data transfer on the remote end. GASPI also provides extended communication calls that combine one or more memory operation (`read`/`write`) and a notification. These calls do not enforce ordering with other operations in the same queue [131]. GASPI supports the atomic operations `fetch-and-add` as well as `compare-and-swap`. These operations are independant from queues and have no ordering semantics.

ARMCI [110] can order memory access by waiting for local completion (`wait`) or

	ordering primitives	ordering by atomics	inspiration
UPC	strict accesses, fence	strict atomics	SC4DRF
UPC++	waiting	acq&rel	C++11
Chapel	waiting	acq&rel	C++11, XC
X10	waiting	locally	Java, C++11
OpenSHMEM	fence, quiet	none	-
GASNet-EX	active messages, waiting	optional acq&rel	-
GASPI	queues, notifications	none	-
ARMCI	fence	none	LC
coreRMA	flush	-	RMA, PGAS
Calin et al. [21]	channels	-	PGAS

Table 3: Memory models in PGAS

global completion (`fence`). The memory model for remote access is described [110] as similar to location consistency [58]. Additionally, ARMCI provides totally ordered atomic accumulation operations.

5.3 Abstract Models

Finally, there has been some effort to construct more abstract models that can be applied to multiple PGAS frameworks: Calin et al. [21] examines the robustness of PGAS programs. Robustness is a less strict correctness criterion than data-race freedom and was originally proposed by Shasha and Snir [128] for shared memory program. coreRMA [46] is a memory model for RMA programming. It is used to discover contradictions in the documentation of libraries for one-sided memory access and predict the behavior of programs.

5.4 Conclusion

This section has given an overview over the memory models used in the PGAS ecosystem. As a trend, younger frameworks tend to have underspecified or less documented memory models. Table 3 illustrates quite clearly that communication libraries typically have relaxed memory models that require synchronization with low-level primitive such as fences, whereas PGAS Framworks tend to provide SC4DRF guaranties. However, they often additionally provide relaxed semantics for data transfer. In conclusion, PGAS applications seem to benefit from hybrid memory models in which they give almost no guaranties about non-synchronizing accesses, but the ordering semantics of synchronizing accesses are quite strong.

6 Shared Data and Replication in PGAS

As remote accesses are costly, replication of read-only data can greatly reduce the global communication and improve the performance [68]. For example, distributed graph algorithms generate a lot of small remote memory accesses, and benefit from communication coalescing and caching read-only data [39, 38]. Accesses to read-mostly data should also benefit from replication, but can harm the programmability, as it must be kept consistent.

The PGAS programming model itself offers no primitives for replication of data. However, for performance reasons, is is often desirable to reuse date retrieved from a remote place or to localize data for fast access. Data can be replicated explicitly in the application or transparently by the underlying implementation if the memory model allows for that. In this section, shared data pattern in PGAS application are reviewed before discussing replication-based optimizations found in PGAS frameworks.

6.1 Patterns in PGAS Applications

This section looks into patterns in PGAS application. It does not aspire to give a complete list of existing PGAS application, but give a idea what patterns and problems might emerge in PGAS applications.

Load Balancing The Scalable Parallel Numerical CSP Solver [76] is a branch-and-prune algorithm, and uses a domain specific work balancing scheme with search space approximation. In a later iteration, it used the *Global Load Balancing Library* [152], for X10, which employes a workstealing scheme. Another example for using GLB employes workstealing on a dense matrix multiplication, and argues in favor of using dynamic load balancing for heterogeneous architectures [129]. Load balancing is also used for Barnes-Hut algorithms [151] or in the more abstract unbalanced tree search problem [104]. In respect to productivity of PGAS applications, global load balancing poses a problem: data used by tasks must be moved, replicated or accessed remotely. Remote access naturally degrade performance, however, manual data movement or replication may harm the programmability of the PGAS languages.

Global Containers Another interesting pattern is the use of global containers. For example, GridPACK [117] is a power grid simulation framework which uses a distributed hash table in order to distribute data across all places. ScaleGraph [47] uses distributed arrays as global container, and collective operations in teams.

6.2 Optimizations

Optimizing data access can be potentially be carried out by the programmer, statically at compile time, or dynamically at runtime.

Compile-time optimizations For the retrospective PGAS language ZPL, Sung-Eun Choi and Snyder [136] discusses three types of optimization that can be applied by a compiler, but also by a dedicated application programmer: *Redundant communication removal* keeps a copy of remote data in a local variable instead of fetching it repeatedly. *Communication combination* coalesces communication to a single host to a single message. This reduces the number of messages send over the network. Combination of communication is a pattern often present in PGAS applications, whether it is by optimizing by hand [62, 38], or by using static analysis [3]. *Communication pipelining* sends data before the receiver initiated a receive. This optimization does only make sense in a two-sided communication model. However, it is closely related to asychronous communication in a one-sided model, as the data transfer can be initiated before the data is actually needed, which helps to hide communication latency.

Even if data is local, access through shared pointers can significantly increase the access latency [62, 35, 23, 68], and potentially prevent compiler optimization. *Localization* makes data accessible through local / native pointers. In UPC(++), there is the upc_cast() [144, §7.7.2.1] function, respectively the local()[5, §3.2] method, which enables casting to a shared/global pointer to a local pointer. For Chapel, a *local* keyword [68] has been proposed, which marks all accesses in a code block as local.

The most notable feature of Clang UPC[36] is the representation of global pointers as a special address space, which allows to use LLVM-based optimisation on remote accesses. This seems similar to the work of Hayashi et al. [69].

Runtime optimizations Other then compile-time optimizations, most runtime optimizations can only made by online algorithms, hence using incomplete data. *Caching* [153, 54] keeps an copy of the data after first accessing it. Caching can be used when the same address, hence also the same remote, is accessed repeatedly. This reduces wait times for recurring accesses and is conceptually related to *redundant communication removal*. *Prefetching* [62, 81] fetches data before it is needed. As opposed to caching, it reduces wait times for first accesses. Conceptually, prefetching is losely related to *communication pipeling* as it can overlap the data transfer with preceeding computation. *Coalacing* bundles multiple accesses into one communication. Communication coalacing bundles multiple accesses to different addresses on the same remote. This is a very similar concept to *communication combination*. I/O vectors like in ARMCI enable explicit accumulation of requests, however, runtimes may additionally accumulate accesses [110].

7 Conclusion

In this paper, we have surveyed the PGAS ecosystems with a focus on applicability to manycore and extreme multicore systems. This allowed us to examine the memory models used for PGAS languages, interfaces for data access, and the use of replication in the PGAS frameworks.

PGAS is a reasonable programming model for future manycore systems. However, it can not fully exploit shared memory in these kind of machines. The survey showed

that application generally benefit from the locality-awareness of the PGAS model and the descriptiveness of the data access interface, but struggle to offer an easy to use interface that supports replication. Extending the PGAS model with a shared memory abstraction is a way out of this dilemma. Applying some the lessons learned in the previous sections, we can name some desirable properties for a possible shared memory extention.

Modern PGAS frameworks provide efficient mechanisms for synchronization, remote access and data distribution. In order to benefit from these mechanisms, a shared memory extension must integrate into the host framework. A major concern is a integration into the memory model of the PGAS framework: Ordering primitives must also interact with the ordering primitives in the PGAS framework.

One factor in success of PGAS frameworks is that they allow to overlap computation with communications, for example through asynchronous communication or two-phase barriers. To enable performance comparable to pure PGAS implementations, a shared memory extension should expose asynchronous interfaces for operations that potentially cause global communication.

The memory semantics in PGAS frameworks is oriented on the relatively weak memory semantics of R(D)MA memory access, especially for large data transfers. Analogously, the shared memory API should allow to implement protocols that exploit the native memory model of a shared memory machine with zero overhead.

PGAS programming models provide a number of synchronization mechanisms. This may includes barriers for BSP-style programming model, or fine-granular mechanisms like synchronisation variables [32, §25.3]. Analogously, a lot of parallel runtimes implement collective operations, as they are an expressive tool for formulating algorithms and can be highly optimized for distributed and shared-memory systems [31]. Trying to replace them with shared memory primitives, for example atomic operations, may be futile.

References

[1] Bilge Acun et al. “Parallel Programming with Migratable Objects: Charm++ in Practice”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’14. New Orleans, Louisana: IEEE Press, 2014, pp. 647–658. DOI: 10.1109/SC.2014.58.

[2] Sarita V. Adve and Mark D. Hill. “Weak Ordering – a New Definition”. In: *SIGARCH Comput. Archit. News* 18.2SI (May 1990), pp. 2–14. DOI: 10.1145/325096.325100.

[3] Michail Alvanos et al. “Improving Communication in PGAS Environments: Static and Dynamic Coalescing in UPC”. In: *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*. ICS ’13. Eugene, Oregon, USA: ACM, 2013, pp. 129–138. DOI: 10.1145/2464996.2465006.

[4] Cédric Augonnet, Samuel Thibault, and Raymond Namyst. *StarPU: a Runtime System for Scheduling Tasks over Accelerator-Based Multicore Machines*. Research Report RR-7240. INRIA, Mar. 2010, p. 33. URL: https://hal.inria.fr/inria-00467677.

[5] J. Bachan et al. *UPC++ Specification v1.0, Draft 4*. Tech. rep. LBNL-2001066. Lawrence Berkeley National Laboratory, Sept. 2017. URL: http://escholarship.org/uc/item/2nm9n3jm.

[6] Scott B. Baden and Paul Hargrove. *UPC++ and GASNet: PGAS Support for Exascale Apps and Runtimes*. Poster at Exascale Computing Project (ECP) Annual Meeting. 2018.

[7] Henry C. Baker Jr. and Carl Hewitt. "The Incremental Garbage Collection of Processes". In: *Proceedings of the 1977 Symposium on Artificial Intelligence and Programming Languages*. ACM, 1977, pp. 55–59. DOI: 10.1145/800228.806932.

[8] M. Bauer et al. "Legion: Expressing locality and independence with logical regions". In: *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. Nov. 2012, pp. 1–11. DOI: 10.1109/SC.2012.71.

[9] N. Beck et al. "'Zeppelin': An SoC for multichip architectures". In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. 2018, pp. 40–42.

[10] S. Bell et al. "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect". In: *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*. Feb. 2008, pp. 88–598. DOI: 10.1109/ISSCC.2008.4523070.

[11] *Berkeley UPC - Unified Parallel C*. URL: https://upc.lbl.gov/.

[12] Hans-J. Boehm and Sarita V. Adve. "Foundations of the C++ Concurrency Memory Model". In: *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '08. Tucson, AZ, USA: ACM, 2008, pp. 68–78. DOI: 10.1145/1375581.1375591.

[13] M. Bohr. "A 30 Year Retrospective on Dennard's MOSFET Scaling Paper". In: *IEEE Solid-State Circuits Society Newsletter* 12.1 (2007), pp. 11–13.

[14] D. Bonachea. *GASNet Specification, v1.1*. aznet. UC Berkeley, 2002. URL: http://upc.lbl.gov/publications/CSD-02-1207.pdf.

[15] D. Bonachea and P. Hargrove. *GASNet Specification, v1.8.1*. Tech. rep. LBNL-2001064. Lawrence Berkeley National Laboratory, Aug. 2017. URL: http://escholarship.org/uc/item/03b5g0q4.

[16] D. Bonachea and J. Jeong. "GASNet: A Portable High-Performance Communication Layer for Global Address-Space Languages". In: *CS258 Parallel Computer Architecture Project,* (2002).

[17] Dan Bonachea and Jason Duell. "Problems with Using MPI 1.1 and 2.0 As Compilation Targets for Parallel Language Implementations". In: *Int. J. High Perform. Comput. Netw.* 1.1-3 (Aug. 2004), pp. 91–99. DOI: 10.1504/IJHPCN.2004.007569.

[18] Dan Bonachea and Paul H. Hargrove. *GASNet-EX: A High-Performance, Portable Communication Library for Exascale*. Tech. rep. Lawrence Berkeley National Laboratory, 2018. DOI: 10.25344/S4QP4W.

[19] F. Broquedis et al. "hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications". In: *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*. Feb. 2010, pp. 180–186. DOI: 10.1109/PDP.2010.67.

[20] Monika ten Bruggencate and Duncan Roweth. "DMAPP - An API for One-sided Program Models on Baker Systems". In: *Cray User Group 2010 Proceedings*. 2010. URL: https://cug.org/5-publications/proceedings_attendee_lists/CUG10CD/pages/1-program/final_program/CUG10_Proceedings/pages/authors/01-5Monday/03B-tenBruggencate-Paper-2.pdf.

[21] Georgel Calin et al. "A Theory of Partitioned Global Address Spaces". In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013)*. Vol. 24. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 127–139. DOI: 10.4230/LIPIcs.FSTTCS.2013.127.

[22] D. Callahan, B. L. Chamberlain, and H. P. Zima. "The cascade high productivity language". In: *Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments, 2004. Proceedings.* Apr. 2004, pp. 52–60. DOI: 10.1109/HIPS.2004.1299190.

[23] F. Cantonnet et al. "Fast Address Translation Techniques for Distributed Shared Memory Compilers". In: *19th IEEE International Parallel and Distributed Processing Symposium*. Apr. 2005, 52b–52b. DOI: 10.1109/IPDPS.2005.219.

[24] F. Cantonnet et al. "Productivity analysis of the UPC language". In: *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* Apr. 2004, pp. 254–. DOI: 10.1109/IPDPS.2004.1303318.

[25] William W. Carlson et al. *Introduction to UPC and Language Specification*. Tech. rep. CCS-TR-99-157. Second Printing. IDA Center for Computing Sciences, May 1999.

[26] Vincent Cavé et al. "Habanero-Java: The New Adventures of Old X10". In: *Proceedings of the 9th International Conference on Principles and Practice of Programming in Java*. PPPJ '11. Kongens Lyngby, Denmark: ACM, 2011, pp. 51–61. DOI: 10.1145/2093157.2093165.

[27] *CCIX Consortium*. URL: https://www.ccixconsortium.com/.

[28] B.L. Chamberlain, D. Callahan, and H.P. Zima. "Parallel Programmability and the Chapel Language". In: *International Journal of High Performance Computing Applications* 21.3 (2007), pp. 291–312. DOI: 10.1177/1094342007078442. eprint: http://hpc.sagepub.com/content/21/3/291.full.pdf+html.

[29] Brad Chamberlain et al. "Chapel Comes of Age: Productive Parallelism at Scale". In: *CUG 2018*. 2018.

[30] Bradford L. Chamberlain and Cray Inc. *Multiresolution Languages for Portable yet Efficient Parallel Programming*. 2007.

[31] Ernie Chan et al. "Collective communication: theory, practice, and experience". In: *Concurrency and Computation: Practice and Experience* 19.13 (Sept. 2007), pp. 1749–1783. DOI: 10.1002/cpe.1206.

[32] *Chapel Language Specification*. Version 0.984. Cray Inc. Oct. 2017.

[33] Barbara Chapman et al. "Introducing OpenSHMEM: SHMEM for the PGAS Community". In: *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*. PGAS '10. New York, New York, USA: ACM, 2010, 2:1–2:3. DOI: 10.1145/2020373.2020375.

[34] Philippe Charles et al. "X10: An Object-oriented Approach to Non-uniform Cluster Computing". In: *SIGPLAN Not.* 40.10 (Oct. 2005), pp. 519–538. DOI: 10.1145/1103845.1094852.

[35] Wei-Yu Chen et al. "A Performance Analysis of the Berkeley UPC Compiler". In: *Proceedings of the 17th Annual International Conference on Supercomputing*. ICS '03. San Francisco, CA, USA: ACM, 2003, pp. 63–73. DOI: 10.1145/782814.782825.

[36] *Clang UPC Tool Set*. URL: https://clangupc.github.io/.

[37] *Compute Express Link consortium*. URL: https://www.computeexpresslink.org/.

[38] Guojing Cong, George Almasi, and Vijay Saraswat. "Fast PGAS Implementation of Distributed Graph Algorithms". In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '10. IEEE Computer Society, 2010, pp. 1–11. DOI: 10.1109/SC.2010.26.

[39] Guojing Cong, Gheorghe Almasi, and Vijay Saraswat. "Fast PGAS Connected Components Algorithms". In: *Proceedings of the Third Conference on Partitioned Global Address Space Programing Models*. PGAS '09. Ashburn, Virginia, USA: ACM, 2009, 13:1–13:6. DOI: 10.1145/1809961.1809979.

[40] *Cray C and C++ Reference Manual (S-2179) 8.6*. June 2017.

[41] D. E. Culler et al. "Parallel Programming in Split-C". In: *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*. Supercomputing '93. Portland, Oregon, USA: ACM, 1993, pp. 262–273. DOI: 10.1145/169627.169724.

[42] David Culler et al. “Generic active message interface specification”. 1994.

[43] David Cunningham et al. “Resilient X10: Efficient Failure-aware Programming”. In: *SIGPLAN Not.* 49.8 (Feb. 2014), pp. 67–80. DOI: 10.1145/2692916.2555248.

[44] L. Dagum and R. Menon. “OpenMP: an industry standard API for shared-memory programming”. In: *IEEE Computational Science and Engineering* 5.1 (Jan. 1998), pp. 46–55. DOI: 10.1109/99.660313.

[45] J. Daily et al. “On the suitability of MPI as a PGAS runtime”. In: *2014 21st International Conference on High Performance Computing (HiPC)*. Dec. 2014, pp. 1–10. DOI: 10.1109/HiPC.2014.7116712.

[46] Andrei Marian Dan et al. “Modeling and Analysis of Remote Memory Access Programming”. In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. OOPSLA 2016. Amsterdam, Netherlands: ACM, 2016, pp. 129–144. DOI: 10.1145/2983990.2984033.

[47] Miyuru Dayarathna, Charuwat Houngkaew, and Toyotaro Suzumura. “Introducing ScaleGraph: An X10 Library for Billion Scale Graph Analytics”. In: *Proceedings of the 2012 ACM SIGPLAN X10 Workshop*. X10 ’12. Beijing, China: ACM, 2012, 6:1–6:9. DOI: 10.1145/2246056.2246062.

[48] Mattias De Wael et al. “Partitioned Global Address Space Languages”. In: *ACM Comput. Surv.* 47.4 (May 2015), 62:1–62:27. DOI: 10.1145/2716320.

[49] James Dinan et al. “An implementation and evaluation of the MPI 3.0 one-sided communication interface”. In: *Concurrency and Computation: Practice and Experience* 28.17 (2016). cpe.3758, pp. 4385–4404. DOI: 10.1002/cpe.3758.

[50] M. Driscoll et al. *PyGAS: A Partitioned Global Address Space Extension for Python*. Poster in the PGAS Conference. 2012. URL: https://people.eecs.berkeley.edu/~driscoll/pdfs/pgas2012.pdf.

[51] Kemal Ebcioglu et al. “An experiment in measuring the productivity of three parallel programming languages”. In: *In Proceedings of the Workshop on Productivity and Performance in High-End Computing (P-PHEC’06)*. IEEE, Los Alamitos, CA, 2006.

[52] Nick Edmonds, Douglas Gregor, and Andrew Lumsdaine. “Extensible PGAS Semantics for C++”. In: *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*. PGAS ’10. New York, New York, USA: ACM, 2010, 12:1–12:10. DOI: 10.1145/2020373.2020385.

[53] Thorsten von Eicken et al. “Active Messages: A Mechanism for Integrated Communication and Computation”. In: *SIGARCH Comput. Archit. News* 20.2 (Apr. 1992), pp. 256–266. DOI: 10.1145/146628.140382.

[54] M. P. Ferguson and D. Buettner. “Caching Puts and Gets in a PGAS Language Runtime”. In: *2015 9th International Conference on Partitioned Global Address Space Programming Models*. Sept. 2015, pp. 13–24. DOI: 10.1109/PGAS.2015.10.

[55] K. Fuerlinger, T. Fuchs, and R. Kowalewski. "DASH: A C++ PGAS Library for Distributed Data Structures and Parallel Algorithms". In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. Dec. 2016, pp. 983–990. DOI: 10.1109/HPCC-SmartCity-DSS.2016.0140.

[56] Gary Funck and Nenad Vukicevic. *UPC Runtime Design UtilizingPortals-4*. Revision: 1.2. Oct. 2012. URL: http://gccupc.org/documents/portals4/portals4-upc-runtime-design.pdf.

[57] Karl Fürlinger et al. "DASH: Data Structures and Algorithms with Support for Hierarchical Locality". In: *Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part II*. Springer International Publishing, 2014, pp. 542–552. DOI: 10.1007/978-3-319-14313-2_46.

[58] G.R. Gao and V. Sarkar. "Location consistency - a new memory model and cache consistency protocol". In: *Computers, IEEE Transactions on* 49.8 (Aug. 2000), pp. 798–813. DOI: 10.1109/12.868026.

[59] *GASNet-EX API Description*. accessed 20.05.2019. URL: https://gasnet.lbl.gov/docs/GASNet-EX.txt.

[60] *GASPI: Global Address Space Programming Interface – Speciation of a PGAS API for communication*. Tech. rep. Version 17.1. 2017. URL: https://raw.githubusercontent.com/GASPI-Forum/GASPI-Forum.github.io/master/standards/GASPI-17.1.pdf.

[61] Kourosh Gharachorloo et al. "Memory Consistency and Event Ordering in Scalable Shared-memory Multiprocessors". In: *Proceedings of the 17th Annual International Symposium on Computer Architecture*. ISCA '90. Seattle, Washington, USA: ACM, 1990, pp. 15–26. DOI: 10.1145/325164.325102.

[62] T. El-Ghazawi and S. Chauvin. "UPC benchmarking issues". In: *International Conference on Parallel Processing, 2001.* Sept. 2001, pp. 365–372. DOI: 10.1109/ICPP.2001.952082.

[63] P. Ghosh, Y. Yan, and B. Chapman. "Support for Dependency Driven Executions among OpenMP Tasks". In: *2012 Data-Flow Execution Models for Extreme Scale Computing*. Sept. 2012, pp. 48–54. DOI: 10.1109/DFM.2012.16.

[64] *GNU Unified Parallel C (GUPC)*. URL: https://gcc.gnu.org/projects/gupc.html.

[65] Brice Goglin and Stéphanie Moreaud. "KNEM: A generic and scalable kernel-assisted intra-node MPI communication framework". In: *Journal of Parallel and Distributed Computing* 73.2 (2013), pp. 176–188. DOI: 10.1016/j.jpdc.2012.09.016.

[66] P. Grun et al. “A Brief Introduction to the OpenFabrics Interfaces - A New Network API for Maximizing High Performance Application Efficiency”. In: *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. Aug. 2015, pp. 34–39. DOI: 10.1109/HOTI.2015.19.

[67] Daniel Grünewald and Christian Simmendinger. “The GASPI API specification and its implementation GPI 2.0”. In: *Proceedings of the 7th International Conference on PGAS Programming Models*. 2014, pp. 243–248. URL: http://www.pgas2013.org.uk/programme.

[68] R. Haque and D. Richards. “Optimizing PGAS Overhead in a Multi-locale Chapel Implementation of CoMD”. In: *2016 PGAS Applications Workshop (PAW)*. Nov. 2016, pp. 25–32. DOI: 10.1109/PAW.2016.009.

[69] Akihiro Hayashi et al. “LLVM-based Communication Optimizations for PGAS Programs”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM ’15. Austin, Texas: ACM, 2015, 1:1–1:11. DOI: 10.1145/2833157.2833164.

[70] Torsten Hoefler et al. “MPI + MPI: a new hybrid approach to parallel programming with MPI plus shared memory”. In: *Computing* (2013). DOI: 10.1007/s00607-013-0324-2.

[71] Torsten Hoefler et al. “Remote Memory Access Programming in MPI-3”. In: *ACM Trans. Parallel Comput.* 2.2 (June 2015), 9:1–9:26. DOI: 10.1145/2780584.

[72] J. Howard et al. “A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS”. In: *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*. 2010, pp. 108–109.

[73] *HPC-X UPC*. URL: https://www.mellanox.com/products/software/hpcx-upc.

[74] *HPE’s Superdome Gets An SGI NUMAlink Makeover*. URL: https://www.nextplatform.com/2017/11/06/hpes-superdome-gets-sgi-numalink-makeover/.

[75] *Intel® Xeon Phi™ Coprocessor System Software Developers Guide*. SKU# 328207-003EN. Intel. 2014.

[76] Daisuke Ishii, Kazuki Yoshizoe, and Toyotaro Suzumura. “Scalable Parallel Numerical CSP Solver”. In: *Principles and Practice of Constraint Programming: 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*. Springer International Publishing, 2014, pp. 398–406. DOI: 10.1007/978-3-319-10428-7_30.

[77] H. W. Jin et al. “LiMIC: support for high-performance MPI intra-node communication on Linux cluster”. In: *2005 International Conference on Parallel Processing (ICPP’05)*. June 2005, pp. 184–191. DOI: 10.1109/ICPP.2005.48.

[78] Hartmut Kaiser et al. “HPX: A Task Based Programming Model in a Global Address Space”. In: *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*. PGAS ’14. Eugene, OR, USA: ACM, 2014, 6:1–6:11. DOI: 10.1145/2676870.2676883.

[79] Laxmikant V. Kale and Sanjeev Krishnan. “CHARM++: A Portable Concurrent Object Oriented System Based on C++”. In: *Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications*. OOPSLA ’93. Washington, D.C., USA: ACM, 1993, pp. 91–108. DOI: 10.1145/165854.165874.

[80] A. Kannan, N. E. Jerger, and G. H. Loh. “Enabling interposer-based disintegration of multi-core processors”. In: *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2015, pp. 546–558.

[81] Engin Kayraklioglu, Michael P. Ferguson, and Tarek El-Ghazawi. “LAPPS: Locality-Aware Productive Prefetching Support for PGAS”. In: *ACM Trans. Archit. Code Optim.* 15.3 (Aug. 2018), 28:1–28:26. DOI: 10.1145/3233299.

[82] Pete Keleher, Alan L. Cox, and Willy Zwaenepoel. “Lazy Release Consistency for Software Distributed Shared Memory”. In: *Proceedings of the 19th Annual International Symposium on Computer Architecture*. ISCA ’92. Queensland, Australia: ACM, 1992, pp. 13–21. DOI: 10.1145/139669.139676.

[83] Brian Kocoloski and John Lange. “XEMEM: Efficient Shared Memory for Composed Applications on Multi-OS/R Exascale Systems”. In: *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC ’15. Portland, Oregon, USA: ACM, 2015, pp. 89–100. DOI: 10.1145/2749246.2749274.

[84] R. Kowalewski et al. “Utilizing Heterogeneous Memory Hierarchies in the PGAS Model”. In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. Mar. 2018, pp. 353–357. DOI: 10.1109/PDP2018.2018.00063.

[85] Olaf Krzikalla. *GASPI proposal: Memory Ordering*. accepted errata proposal. Nov. 2016. URL: http://www.gaspi.de/proposals/memory_model.pdf.

[86] W. Kuchera and C. Wallace. “The UPC memory model: problems and prospects”. In: *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* Apr. 2004, pp. 16–. DOI: 10.1109/IPDPS.2004.1302921.

[87] William Kuchera. “Illuminating the UPC Memory Model”. MA thesis. 2003.

[88] William Kuchera and Charles Wallace. *Illustrative test cases for the UPC memory modell*. Tech. rep. Michigan Technological University, 2003. URL: http://www.mtu.edu/cs/research/papers/pdfs/upc-mm-tests.pdf.

[89] William Kuchera and Charles Wallace. *Toward a programmer-friendly formal spezification of the UPC memory model*. Tech. rep. CS-TR-03-01. Michigan Technological University, 2003. URL: http://www.mtu.edu/cs/research/papers/pdfs/upc-memory-model.pdf.

[90] S. Kumar et al. “PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer”. In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. May 2012, pp. 763–773. DOI: 10.1109/IPDPS.2012.73.

[91] Vivek Kumar et al. “HabaneroUPC++: A Compiler-free PGAS Library”. In: *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*. PGAS ’14. Eugene, OR, USA: ACM, 2014, 5:1–5:10. DOI: 10.1145/2676870.2676879.

[92] Bryant C. Lam et al. “Benchmarking Parallel Performance on Many-Core Processors”. In: *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools: First Workshop, OpenSHMEM 2014, Annapolis, MD, USA, March 4-6, 2014. Proceedings*. Springer International Publishing, 2014, pp. 29–43. DOI: 10.1007/978-3-319-05215-1_3.

[93] Bryant C. Lam et al. “Low-level PGAS computing on many-core processors with TSHMEM”. In: *Concurrency and Computation: Practice and Experience* 27.17 (2015). cpe.3569, pp. 5288–5310. DOI: 10.1002/cpe.3569.

[94] L. Lamport. “How to make a correct multiprocess program execute correctly on a multiprocessor”. In: *Computers, IEEE Transactions on* 46.7 (July 1997), pp. 779–782. DOI: 10.1109/12.599898.

[95] J. Lee and M. Sato. “Implementation and Performance Evaluation of XcalableMP: A Parallel Programming Language for Distributed Memory Systems”. In: *2010 39th International Conference on Parallel Processing Workshops*. Sept. 2010, pp. 413–420. DOI: 10.1109/ICPPW.2010.62.

[96] J. Lee, M. Sato, and T. Boku. “OpenMPD: A Directive-Based Data Parallel Language Extension for Distributed Memory Systems”. In: *2008 International Conference on Parallel Processing - Workshops*. Sept. 2008, pp. 121–128. DOI: 10.1109/ICPP-W.2008.28.

[97] Kai Li. “IVY: A Shared Virtual Memory System for Parallel Computing.” In: *ICPP (2)* 88 (1988), p. 94.

[98] Calvin Lin and Lawrence Snyder. “ZPL: An array sublanguage”. In: *Languages and Compilers for Parallel Computing: 6th International Workshop Portland, Oregon, USA, August 12–14, 1993 Proceedings*. Springer Berlin Heidelberg, 1994, pp. 96–114. DOI: 10.1007/3-540-57659-2_6.

[99] Richard J. Lipton and Jonathan Sandberg. *PRAM: A Scalable Shared Memory*. Tech. rep. TR-180-88. Prineton University, Aug. 1988. URL: https://www.cs.princeton.edu/research/techreps/TR-180-88.

[100] G. Long, N. Yuan, and D. Fan. “Location Consistency Model Revisited: Problem, Solution and Prospects”. In: *2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*. Dec. 2008, pp. 91–98. DOI: 10.1109/PDCAT.2008.31.

[101] D. B. Loveman. "High performance Fortran". In: *IEEE Parallel Distributed Technology: Systems Applications* 1.1 (Feb. 1993), pp. 25–42. DOI: 10.1109/88.219857.

[102] T. Ma et al. "Kernel Assisted Collective Intra-node MPI Communication among Multi-Core and Many-Core CPUs". In: *2011 International Conference on Parallel Processing*. Sept. 2011, pp. 532–541. DOI: 10.1109/ICPP.2011.29.

[103] Rui Machado and Carsten Lojewski. "The Fraunhofer virtual machine: a communication library and runtime system based on the RDMA model". In: *Computer Science - Research and Development* 23.3 (June 2009), pp. 125–132. DOI: 10.1007/s00450-009-0088-2.

[104] Rui Machado et al. "Unbalanced tree search on a manycore system using the GPI programming model". In: *Computer Science - Research and Development* 26.3 (Apr. 2011), p. 229. DOI: 10.1007/s00450-011-0163-3.

[105] Alan Mainwaring and David Culler. *Active Message Applications Programming Interface and Communication Subsystem Organization*. Tech. rep. UC Berkely, 1996. URL: http://gasnet.lbl.gov/amudp/AM-2spec.pdf.

[106] Jeremy Manson, William Pugh, and Sarita V. Adve. "The Java memory model". In: *SIGPLAN Not.* 40.1 (Jan. 2005), pp. 378–391. DOI: 10.1145/1047659.1040336.

[107] David Mosberger. "Memory consistency models". In: *SIGOPS Oper. Syst. Rev.* 27.1 (Jan. 1993), pp. 18–26. DOI: 10.1145/160551.160553.

[108] *MPI: A Message-Passing Interface Standard – Version 3.1*. Tech. rep. Message Passing Interface Forum, 2015. URL: https://www.mpi-forum.org/docs/.

[109] David Mulnix. *Intel Xeon Processor Scalable Family Technical Overview*. last updated 06/10/2019. June 2017. URL: https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview.

[110] J. Nieplocha et al. "High Performance Remote Memory Access Communication: The Armci Approach". In: *The International Journal of High Performance Computing Applications* 20.2 (2006), pp. 233–253. DOI: 10.1177/1094342006064504. eprint: http://dx.doi.org/10.1177/1094342006064504.

[111] Jarek Nieplocha and Jialin Ju. *ARMCI: A Portable Aggregate Remote Memory Copy Interface*. Tech. rep. Pazific Nordwest National Laboratory, 2000. URL: http://hpc.pnl.gov/armci/documentation.htm.

[112] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield. "Global Arrays: A Portable "Shared-memory" Programming Model for Distributed Memory Computers". In: *Proceedings of the 1994 ACM/IEEE Conference on Supercomputing*. Supercomputing '94. Washington, D.C.: IEEE Computer Society Press, 1994, pp. 340–349. ISBN: 0-8186-6605-6. URL: http://dl.acm.org/citation.cfm?id=602770.602833.

[113] Jörg Nolte, Yutaka Ishikawa, and Mitsuhisa Sato. "TACO: Prototyping High-level Object-oriented Programming Constructs by Means of Template Based Programming Techniques". In: *SIGPLAN Not.* 36.12 (Dec. 2001), pp. 35–49. DOI: 10.1145/583960.583965.

[114] *NVLINK and NVSWITCH*. URL: https://www.nvidia.com/en-us/data-center/nvlink/.

[115] *OpenCAPI Consortium*. URL: https://opencapi.org/.

[116] *OpenSHMEM - Application Programming Interface Version 1.4*. Dec. 2017. URL: http://www.openshmem.org/site/Specification.

[117] B. Palmer. "Application of PGAS Programming to Power Grid Simulation". In: *2016 PGAS Applications Workshop (PAW)*. Nov. 2016, pp. 33–40. DOI: 10.1109/PAW.2016.010.

[118] Krzysztof Parzyszek. "Generalized Portable SHMEM Library for High Performance Computing". AAI3105098. PhD thesis. Iowa State University, 2003.

[119] Stephen W. Poole et al. "OpenSHMEM - Toward a Unified RMA Model". In: *Encyclopedia of Parallel Computing*. Springer US, 2011, pp. 1379–1391. DOI: 10.1007/978-0-387-09766-4_490.

[120] *QuickSpecs HP UPC Version 3.3*. July 2011. URL: https://www.hpe.com/h20195/v2/GetDocument.aspx?docname=c04154428.

[121] *RapidIO consortium*. URL: https://rapidio.org.

[122] Vijay Saraswat et al. "The asynchronous partitioned global address space model". In: *in 1st ACM SIGPLAN Workshop on Advances in Message Passing (AMP'10*. ACM Press, 2010.

[123] Vijay Saraswat et al. *X10 Language Specification*. Tech. rep. Version 2.6.1. June 2017.

[124] Jeevan Savant and Steve Seidel. *MuPC: A Run Time System For Unified Parallel C*. Tech. rep. Michigan Technological University, 2002. URL: http://www.mtu.edu/cs/research/papers/pdfs/jeevan-thesis.pdf.

[125] G. Shah et al. "Performance and experience with LAPI-a new high-performance communication library for the IBM RS/6000 SP". In: *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*. Mar. 1998, pp. 260–266. DOI: 10.1109/IPPS.1998.669923.

[126] P. Shamis et al. "UCX: An Open Source Framework for HPC Network APIs and Beyond". In: *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. Aug. 2015, pp. 40–43. DOI: 10.1109/HOTI.2015.13.

[127] Pavel Shamis et al. “Designing a High Performance OpenSHMEM Implementation Using Universal Common Communication Substrate as a Communication Middleware”. In: *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools*. Springer International Publishing, 2014, pp. 1–13. ISBN: 978-3-319-05215-1.

[128] Dennis Shasha and Marc Snir. “Efficient and Correct Execution of Parallel Programs That Share Memory”. In: *ACM Trans. Program. Lang. Syst.* 10.2 (Apr. 1988), pp. 282–312. DOI: 10.1145/42190.42277.

[129] Brendan Sheridan and Jeremy T. Fineman. “A Case for Distributed Work-stealing in Regular Applications”. In: *Proceedings of the 6th ACM SIGPLAN Workshop on X10*. X10 2016. Santa Barbara, CA, USA: ACM, 2016, pp. 32–33. DOI: 10.1145/2931028.2931035.

[130] J. Shirako et al. “Phaser accumulators: A new reduction construct for dynamic parallelism”. In: *2009 IEEE International Symposium on Parallel Distributed Processing*. May 2009, pp. 1–12. DOI: 10.1109/IPDPS.2009.5161071.

[131] Christian Simmendinger. *Errata proposal - Read/Write ordering*. Nov. 2016.

[132] Christian Simmendinger, Mirko Rahn, and Daniel Gruenewald. “The GASPI API: A Failure Tolerant PGAS API for Asynchronous Dataflow on Heterogeneous Architectures”. In: *Sustained Simulation Performance 2014*. Springer International Publishing, 2015, pp. 17–32. DOI: 10.1007/978-3-319-10626-7_2.

[133] Christian Simmendinger et al. “Interoperability strategies for GASPI and MPI in large-scale scientific applications”. In: *The International Journal of High Performance Computing Applications* 33.3 (2019), pp. 554–568. DOI: 10.1177/1094342018808359.

[134] A. Sodani et al. “Knights Landing: Second-Generation Intel Xeon Phi Product”. In: *IEEE Micro* 36.2 (Mar. 2016), pp. 34–46. DOI: 10.1109/MM.2016.25.

[135] Daniel J. Sorin, Mark D. Hill, and David A. Wood. *A Primer on Memory Consistency and Cache Coherence*. 1st. Morgan & Claypool Publishers, 2011. ISBN: 9781608455645.

[136] Sung-Eun Choi and L. Snyder. “Quantifying the effects of communication optimizations”. In: *Proceedings of the 1997 International Conference on Parallel Processing (Cat. No.97TB100162)*. Aug. 1997, pp. 218–222. DOI: 10.1109/ICPP.1997.622647.

[137] S. M. Tam et al. “SkyLake-SP: A 14nm 28-Core xeon® processor”. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. 2018, pp. 34–36.

[138] Gabriel Tanase et al. *Performance Analysis of the IBM XL UPC on the PERCS Architecture*. Tech. rep. RC25360. IBM, 2013.

[139] Olivier Tardieu. "The APGAS Library: Resilient Parallel and Distributed Programming in Java 8". In: *Proceedings of the ACM SIGPLAN Workshop on X10*. X10 2015. Portland, OR, USA: ACM, 2015, pp. 25–26. DOI: 10.1145/2771774.2771780.

[140] Sean Treichler, Michael Bauer, and Alex Aiken. "Realm: An Event-based Low-level Runtime for Distributed Memory Architectures". In: *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*. PACT '14. Edmonton, AB, Canada: ACM, 2014, pp. 263–276. DOI: 10.1145/2628071.2628084.

[141] Keisuke Tsugane et al. "Multi-tasking Execution in PGAS Language XcalableMP and Communication Optimization on Many-core Clusters". In: *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*. HPC Asia 2018. Chiyoda, Tokyo, Japan: ACM, 2018, pp. 75–85. DOI: 10.1145/3149457.3154482.

[142] UPC Consortium. *UPC Language Specifications V1.2*. Tech. rep. Lawrence Berkeley National Lab, 2005. URL: http://upc-lang.org.

[143] UPC Consortium. *UPC Language Specifications, Version 1.3*. Tech. rep. LBNL-6623E. Lawrence Berkeley National Lab, 2013. URL: http://upc-lang.org.

[144] UPC Consortium. *UPC Optional Library Specifications – Version 1.3*. Tech. rep. 2013.

[145] Leslie G. Valiant. "A Bridging Model for Parallel Computation". In: *Commun. ACM* 33.8 (Aug. 1990), pp. 103–111. DOI: 10.1145/79173.79181.

[146] Jerome Vienne. "Benefits of Cross Memory Attach for MPI Libraries on HPC Clusters". In: *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. XSEDE '14. Atlanta, GA, USA: ACM, 2014, 33:1–33:6. DOI: 10.1145/2616498.2616532.

[147] Michael Woodacre et al. *The SGI Altix 3000 global shared-memory architecture*. SGI White Paper. Silicon Graphics, Inc. 2003.

[148] K. Yelick, D. Bonachea, and C. Wallace. *A Proposal for a UPC Memory Consistency Model, v1.0*. Tech. rep. LBNL-54983. Lawrence Berkeley National Lab Tech, May 2004.

[149] Katherine Yelick et al. "Productivity and Performance Using Partitioned Global Address Space Languages". In: *Proceedings of the 2007 International Workshop on Parallel Symbolic Computation*. PASCO '07. London, Ontario, Canada: ACM, 2007, pp. 24–32. DOI: 10.1145/1278177.1278183.

[150] Kathy Yelick et al. "Titanium: a high-performance Java dialect". In: *Concurrency: Practice and Experience* 10.11-13 (1998), pp. 825–836. DOI: 10.1002/(SICI)1096-9128(199809/11)10:11/13<825::AID-CPE383>3.0.CO;2-H.

[151] J. Zhang, B. Behzad, and M. Snir. "Design of a Multithreaded Barnes-Hut Algorithm for Multicore Clusters". In: *IEEE Transactions on Parallel and Distributed Systems* 26.7 (July 2015), pp. 1861–1873. DOI: 10.1109/TPDS.2014.2331243.

[152] Wei Zhang et al. "GLB: Lifeline-based Global Load Balancing Library in x10". In: *Proceedings of the First Workshop on Parallel Programming for Analytics Applications*. PPAA '14. Orlando, Florida, USA: ACM, 2014, pp. 31–40. DOI: 10.1145/2567634.2567639.

[153] Zhang Zhang, J. Savant, and S. Seidel. "A UPC runtime system based on MPI and POSIX threads". In: *14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06)*. Feb. 2006. DOI: 10.1109/PDP.2006.16.

[154] Weiming Zhao and Zhenlin Wang. "ScaleUPC: A UPC Compiler for Multi-core Systems". In: *Proceedings of the Third Conference on Partitioned Global Address Space Programing Models*. PGAS '09. Ashburn, Virginia, USA: ACM, 2009, 11:1–11:8. DOI: 10.1145/1809961.1809976.

[155] Yili Zheng et al. "UPC++: A PGAS Extension for C++". In: *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. May 2014, pp. 1105–1114. DOI: 10.1109/IPDPS.2014.115.

[156] Huan Zhou, Kamran Idrees, and José Gracia. "Leveraging MPI-3 Shared-Memory Extensions for Efficient PGAS Runtime Systems". In: *Euro-Par 2015: Parallel Processing: 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, August 24-28, 2015, Proceedings*. Springer Berlin Heidelberg, 2015, pp. 373–384. DOI: 10.1007/978-3-662-48096-0_29.

[157] Andreas Zwinkau. "A Memory Model for X10". In: *Proceedings of the 6th ACM SIGPLAN Workshop on X10*. X10 2016. Santa Barbara, CA, USA: ACM, 2016, pp. 7–12. DOI: 10.1145/2931028.2931031.