HPX – An open source C++ Standard Library for Parallelism and Concurrency

Thomas Heller
Friedrich-Alexander University
Erlangen-Nuremberg
Erlangen, Bavaria, Germany

Patrick Diehl
Polytechnique Montreal
Montreal, Quebec, Canada

Zachary Byerly
Louisiana State University
Baton Rouge, Louisiana, USA

John Biddiscombe
Swiss Supercomputing Centre (CSCS)
Lugano, Switzerland

Hartmut Kaiser
Louisiana State University
Baton Rouge, Louisiana, USA

ABSTRACT

To achieve scalability with today's heterogeneous HPC resources, we need a dramatic shift in our thinking; MPI+X is not enough. Asynchronous Many Task (AMT) runtime systems break down the global barriers imposed by the Bulk Synchronous Programming model. HPX is an open-source, C++ Standards compliant AMT runtime system that is developed by a diverse international community of collaborators called The Ste||ar Group. HPX provides features which allow application developers to naturally use key design patterns, such as overlapping communication and computation, decentralizing of control flow, oversubscribing execution resources and sending work to data instead of data to work. The Ste||ar Group is comprised of physicists, engineers, and computer scientists; men and women from many different institutions and affiliations, and over a dozen different countries. We are committed to advancing the development of scalable parallel applications by providing a platform for collaborating and exchanging ideas. In this paper, we give a detailed description of the features HPX provides and how they help achieve scalability and programmability, a list of applications of HPX including two large NSF funded collaborations (STORM, for storm surge forecasting; and STAR (OctoTiger) an astrophysics project which runs at 96.8% parallel efficiency on 643,280 cores), and we end with a description of how HPX and the Ste||ar Group fit into the open source community.

CCS CONCEPTS

• **Computing methodologies** → *Parallel computing methodologies*;

KEYWORDS

parallelism, concurrency, standard library, C++

1 INTRODUCTION

The free lunch is over [19] - the end of Moore's Law means we have to use more cores instead of faster cores. The massive increase in on-node parallelism is also motivated by the need to keep power consumption in balance [18]. We have been using large numbers of cores in promising architectures for many years, like GPUs, FPGAs, and other many core systems; now we have Intel's Knights Landing with up to 72 cores. Efficiently using that amount of parallelism, especially with heterogeneous resources, requires a paradigm shift; we must develop new effective and efficient parallel programming techniques to allow the usage of all parts in the system. All in all, it can be expected that concurrency in future systems will be increased by an order of magnitude.

HPX is an open-source, C++ Standards compliant, Asynchronous Many Task (AMT) runtime system. Because HPX is a truly collaborative, international project with many contributors, we could not say that it was developed at one (or two, or three...) universities. The Ste||ar Group was created "to promote the development of scalable parallel applications by providing a community for ideas, a framework for collaboration, and a platform for communicating these concepts to the broader community." We congregate on our IRC channel (#ste||ar on freenode.net), have a website and blog (stellar-group.org), as well as many active projects in close relation to HPX. For the last 3 years we have been an organization in Google Summer of Code, mentoring 17 students. The Ste||ar Group is diverse and inclusive; we have members from at least a dozen different countries. Trying to create a library this size that is truly open-source and C++ Standards compliant is not easy, but it is something that we are committed to. We also believe it is critical to have an open exchange of ideas and to reach out to domain scientists (physicists, engineers, biologists, etc.) who would benefit from using HPX. This is why we have invested so much time and energy into our two scientific computing projects: STORM, a collaborative effort which aims to improve storm surge forecasting; and STAR (OctoTiger) an astrophysics project which has generated exciting

results running at 96.8% parallel efficiency on 643,280 cores. In addition leading the runtime support effort in the european FETHPC project AllScale to research further possibilities to leverage large scale AMT systems.

In the current landscape of scientific computing, the conventional thinking is that the currently prevalent programming model of MPI+X is suitable enough to tackle those challenges with minor adaptations [1]. That is, MPI is used as the tool for inter-node communication, and X is a placeholder to represent intra-node parallelization, such as OpenMP and/or CUDA. Other inter-node communication paradigms, such as partitioned global address space (PGAS), emerged as well, which focus on one sided messages together with explicit, mostly global synchronization. While the promise of keeping the current programming patterns and known tools sounds appealing, the disjoint approach results in the requirement to maintain various different interfaces and the question of interoperability is, as of yet, unclear [8].

Moving from Bulk Synchronous Programming (BSP) towards fine grained, constraint based synchronization in order to limit the effects of global barriers, can only be achieved by changing paradigms for parallel programming. This paradigm shift is enabled by the emerging Asynchronous Many Task (AMT) runtime systems, that carry properties to alleviate the aforementioned limitations. It is therefore not a coincidence that the C++ Programming Language, starting with the standard updated in 2011, gained support for concurrency by defining a memory model in a multi-threaded world as well as laying the foundation towards enabling task based parallelism by adapting the `future` [2, 9] concept. Later on, in 2017, based on those foundational layers, support for parallel algorithms was added, which coincidentally, covers the need for data parallel algorithms. The HPX parallel runtime system unifies an AMT tailored for HPC usage combined with strict adherence to the C++ standard. It therefore represents a combination of well-known ideas (such as data flow [6, 7], futures [2, 9], and Continuation Passing Style (CPS)) with new and unique overarching concepts. The combination of these ideas and their strict application forms underlying design principles that makes this model unique. HPX provides:

- A C++ Standards-conforming API enabling wait-free asynchronous execution.
- `futures`, `channels`, and other asynchronous primitives.
- An Active Global Address Space (AGAS) that supports load balancing through object migration.
- An active messaging network layer that ships functions to data.
- A work-stealing lightweight task scheduler that enables finer-grained parallelization and synchronization.
- The versatile and powerful in-situ performance observation, measurement, analysis, and adaptation framework APEX.
- Integration of GPUs with HPX.Compute [5] and HPXCL for providing a single source solution to heterogeneity

HPX's features and programming model allow application developers to naturally use key design patterns, such as overlapping communication and computation, decentralizing of control flow, oversubscribing execution resources and sending work to data instead of data to work. Using Futurization, developers can express complex data flow execution trees that generate millions of HPX

tasks that by definition execute in the proper order. This paper is structured as follows:

In Section 2 we summarize briefly the important features of HPX for parallelism and concurrency. The application of HPX in different high performance computing related topics are presented in Section 3. In Section 4 we review the aspect of open source software and community building. Finally, we conclude with Section 5.

2 HPX RUNTIME COMPONENTS

In this section we briefly give an overview of the HPX runtime system components with a focus on parallelism and concurrency. Figure 1 shows five runtime components of the runtime system which are briefly reviewed in the following. The core element is the thread mamager which provides the local thread management, see Section 2.1. The Active Global Address Space (AGAS) provides an abstraction over globally addressable C++ objects which support RPC (Section 2.2). In Section 2.3 the parcel handler which implements the one-side active massaging layer is reviewed. For runtime adaptivity HPX exposes several local and global performance counters, see Section 2.4. For more details we refer to [16].
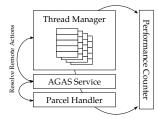

Figure 1: Overview of the HPX Runtime System components.

2.1 Thread manager

Within HPX a light-weight user level threading mechanism is used to offer the necessary capabilities to implement all higher level APIs. Predefiend scheduling policies are provided to determine the optimal task scheduling for a given application and/or algorithm. This enables the programmer to focus on algorithms instead of thinking in terms of CPU resources. The predefined scheduling policies may be interchanged at runtime, additionally, new scheduling policies may be provided by the user for more application specific control. The currently available scheduling policies are summarized by: **static:** the static scheduling policy maintains one queue per core while disabling task stealing alltogether; **thread local:** the thread local policy is currently the default policy. This policy maintains one queue per core, whenever a core runs out of work, tasks will be stolen from neighboring queues, high priority queues also exist to prioritize important tasks over others; **hierarchical:** the hierarchical policy maintains a tree of queues with the purpose of creating an hierarchical structure, while new tasks are always enqueued at the root, when one core fetches new work, the available tasks trickle down the hierarchy.

2.2 Active Global Address Space

After having the lightweight user level tasks available the next important module within HPX is the Active Global Address Space

(AGAS). It's main purpose is to serve as the foundation to support distributed computing and is related to other global address space attempts which are represented by the PGAS family. Their main purpose is to hide explicit message passing. By having each object living in AGAS being represented by a GID (Global ID), the location of said object does not need to be bound to a particular locality and leads to uniformity in terms of independence whether a object is located remotely or local. In addition, this independence allows for transparent migration between different process local address spaces with AGAS being responsible for proper address resolution.

2.3 Parcel

To support distributed computing, mechanisms for exchanging messages over a network interconnect are necessary. The current HPC landscape is dominated by two major programming paradigms. The first is represented by MPI which offers a wide variety of different communication primitives for passing messages between processes. It is best known for its two-sided communication primitives where the sending side issues a receive operation which contains the buffers, size of the buffers, a tag and the destination. The destination has to proactively post the receiving part of this operation. Those operations are extended by asynchronous versions. In the HPX programming model, neither of those previously discussed forms of communication seems to be appropriate. While having a versatile form of invoking light weight threads locally, it seems natural that this should be extended in such a way to allow for remote invocations of threads as well. This form of communication is often referred to as RPC or Active Messaging. An Active Message in HPX is called a Parcel. The parcel is the underlying concept that allows us to implement RPC calls to remote localities in a C++ fashion. That means that we require a arbitrary, yet type safe, number of arguments, as well as the means to return values from remote procedure invocations. This operation is one-sided, as the RPC destination does not need to actively poll on incoming messages, and follows the semantics of a regular C++ member function call. For more details we refer to [3, 14].

2.4 Performance counters

One important feature that helps improve performance portability is the means to be able to intrinsically profile certain aspects of a program. To assist in the decision making process, an HPX intrinsic performance counter framework has been devised, which is able to encompass HPX internal performance metrics as well as being extensible so that application specific metrics may be supplied, or platform dependant hardware performance counters (eg. using PAPI). The performance counters are readable via AGAS, by having each counter registered with a specific symbolic name, making them accessible throughout the system (i.e. from any node). For more details we refer to [10].

3 APPLICATIONS OF HPX IN HIGH PERFORMANCE COMPUTING

The concepts of parallelism and concurrency were successfully applied in following applications.

LibGeoDecomp N-Body Code. Scientific computing simulations have generally been developed by domain scientists (physicists, materials scientists, engineers, etc.), even those who either currently use or would greatly benefit from using HPC resources. LibGeoDecomp, a Library for Geometric Decomposition, reduces the effort required to develop scalable, efficient codes. Libgeodecomp handles the spatial and temporal loops, as well as the data storage, while the domain scientist supplies the code required to update one unit. Heterogeneous machines make this even more difficult. A 3D N-Body code written in LibGeoDecomp was used to compare the performance of the MPI backend and the HPX backend. HPX showed perfect scaling at the single node level (98% peak performance) and 89% peak performance using the Xeon Phi Knights Corner coprocessor [13]. Further improvements to the communication layer [15] have improved the performance of this application,outperforming the MPI implementation by a factor of 1.4 and achieving a parallel efficiency of 87% at 1024 compute nodes (16384 cores).

C++ Co-array Semantics. HPX has been used to design a high performance API, written in C++, that allows developers to use semantics similar to Co-array Fortran to write distributed code. The goal was to enable the creation a more easily understandable SPMD-style applications, while adhering to the C++ standard and taking advantage of cutting edge features of the language. Results from performance measurements show that scalability and good performance can be achieved, and with much better programmability and development efficiency, especially for domain scientists [20].

GPGPU support. There is currently no usable standards-conforming library solution for writing C++ code that is portable across heterogeneous architectures (GPGPUs, accelerators, SIMD vector units, general purpose cores). We have set out to provide a solution to this problem with HPX. [12] describes the design and implementation based on the C++17 Standard and provides extensions that ensure locality of work and data. The well-known STREAM benchmark was ported and compared to running the corresponding code natively. We show that using our single source, generic, and extensible abstraction results in no loss of performance. A SYCL backend for this API has also been developed, allowing for single-source programming of OpenCL devices in C++. A comparison using the STREAM benchmark showed only minimal performance penalties compared to SYCL for large arrays.

Linear Algebra Building Blocks. Codes that make use of different threading libraries (OpenMP, Kokkos, TBB, HPX, etc.) for on node parallelism cannot be easily mixed since they all wish to *own* the compute resources and composability of code is reduced when diffferent runtimes are mixed. We have therefore begun the process of creating linear algebra building blocks that can make use of vendor optimized libraries (such as MKL, cublas) on a node, but be integrated cleanly with HPX to make higher level BLAS type functions that are commonly used in HPC applications. Current benchmarks (Cholesky) show that the HPX versions perform as well as the leading libraries in this area.

Storm Surge Forecasting. Tropical cyclones pose a significant threat to both people and property. The most damaging can cause tens of billions of dollars in property damage, and the deadliest can

kill hundreds of thousands of people. Storm surge causes most of the fatalities associated with tropical cyclones, but using computational models it can be forecasted. ADCIRC [17] is an unstructured grid coastal circulation and storm surge prediction model, widely used by the coastal engineering and oceanographic communities. The unstructured triangular grid allows for seamless modeling of a large range of scales, allowing higher resolution near the coast and inland and lower resolution in the open ocean. Producing reliable forecasting results enables emergency managers and other officials to make decisions that will reduce danger to human lives and property. Fast, efficient calculation of results is critical in this role, and so the importance of high performance computing is critical. We have used a novel approach [4] to update the Fortran MPI implementation of a discontinuous Galerkin version of ADCIRC, DGSWEM, to use HPX and LibGeoDecomp as a parallel driver, while still retaining the original Fortran source code. LibGeoDecomp provides an HPX data flow based driver which greatly mitigates problems of load imbalance, exposes more parallel work, overlaps communication and computation - all leading to decreased execution time. With HPX, we are also able to run efficiently on Intel's Xeon Phi Knights Landing on TACC's Stampede 2.

Astrophysics Research. The astrophysics research group at Louisiana State University studies the mergers of double white dwarf binary star systems using hydrodynamic simulations. These mergers are believed to be the source of Type Ia Supernovae, which due to the physics of the explosion, always produce roughly the same amount of light. These "standard candles" were used by two independent groups in the discovery of the accelerating expansion of the universe, winning them the 2011 Nobel Prize in Physics. Due to the relatively short timescale on which these supernovae occur, the light from the binary star system leading up to the merger has been rarely observed. So we rely on computer simulations to give us insights into these mergers. Previously, the group at LSU used static uniform cylindrical mesh hydrodynamic code, written in Fortran and parallelized with MPI to simulate these mergers. In a collaborative effort funded by the NSF, the group has developed a highly efficient, Adaptive Mesh Refinement simulation based on HPX, using the parallel C++ programming model which is now being incorporated into the ISO C++ Standard. Futurization is used to transform sequential code into wait-free asynchronous tasks. They demonstrate a model run using 14 levels of refinement, which achieves a parallel efficiency of 96.8% on NERSC's Cori (643,280 Intel Knights Landing cores) using billions of lightweight threads.

4 HPX AND OPEN SOURCE

Open source software (OSS) in high performance computing is already prevalent, i. e. all super computers in Top 500 list released 2017 run on Linux/Unix. More recently collaborative projects, like OpenHPC[1], where vendors, educational institutions, and research organizations try to enable the complete HPC stack as OSS. One of their statements is they "will provide flexibility for multiple configurations and scalability to meet a wide variety of user needs". In the previous sections we argued that HPX can be used to address the scalability on multiple configurations. One example is the seamless

integration of acceleration cards, like GPUs [11] and Xeon Phi, in the asynchronous execution graph.

Responding to requests from users, HPX core developers have a implemented several important features, including flexible but standards conforming parallel algorithms, IBVerbs and libfabrics parcelports, and thread priorities. As demonstrated by the ratio of closed tickets to total tickets generated by non-core developers, the team of HPX core developers is responsive to the requests of its users. HPX has a geographically decentralized team of core developers, each supported by seperate funding, and a much larger team of other developers contributing to not only to HPX but to other projects using HPX. Because of the exciting, cutting edge work being done, HPX attracts top C++ talent, many of which are students.

This section focus on the statistics with respect to open source software, see Table 1. The complete source code of HPX is hosted on github[2] (702 stars so far) and contains 598093 lines of code where 465869 are C++ code[3]. HPX's stable version is 1.0 containing 15 releases. HPX is licensed under Boost Software License (Version 1.0) which conforms to the Free/Libre Open Source Software (FOSS) concept. Continuous integration is done for Linux with CircleCI and for Windows with AppVeyor. Another aspect of open source software

Attribute	
License	Boost Software License, Version 1.0
Version	1.0 with 15 releases
Commits	18431 from 78 contributors
Lines of Code	465869 (pure C++) and 598093 (all files)
Continuous Integration	CircleCI and AppVeyor

Table 1: Statistics of HPX's source code and community

is its community; HPX is developed under the umbrella of Ste||ar group which includes 78 fellows from all over the world who contributed 17136 commits. Figure 2(a) shows that in the last two years approximately 10 people contributed to HPX each month. Since 2014 the Stellar group was accepted as an organization for Google Summer of Code (GSoC). In the last three summers 17 students contributed valuable features to HPX or application using HPX. Figure 2(b) shows the commits per month since 2008 when HPX was hosted on github. Since 2014 the commits per month increase during the summer due to contributions of the GSoC students.
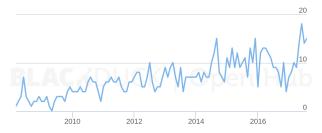
5 CONCLUSION AND OUTLOOK

On the latest and future super computers effective and efficient parallel programming techniques are necessary to address the heterogeneity of the system. In addition, the relevance of an open source solution for a HPC stack, like the OpenHPC project, is developing. With HPX as an open source C++ standard library for parallelism and concurrency both of these features are addressed. First, we provide a parallel, AMT runtime system tailored to, but not limited to, HPC usage. One focus is the strict adherence to the
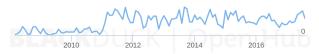
---

[1]http://www.openhpc.community/

[2]https://github.com/STEllAR-GROUP/hpx
[3]https://www.openhub.net/p/stellar-hpx/

(a) Contributors per month. In the last to years we had around 10 different contributors per month.

(b) Commit per month from 78 contributors. Note, that some of this contributors are GSoC students which contribute for three months during the summer. Therefore, in the summer the commits per month increase.

Figure 2: Commits and contributors from the beginning of 2008 when HPX was hosted on github. The picures were taken from openhub.

C++ standard, where well-known concepts, i. e. futures, are implemented, as well as contributing new innovative ideas like Continuation Passing Style programming and dataflow to the broader C++ Community. Second, HPX can be used to write applications for heterogeneous many core systems, without changes to the code – with HPX.Compute [5] we provide a unified model for heterogeneous programming using CUDA and SYCL to provide a single source solution to heterogeneity. With HPXCL we provide a means for application developers to write GPU kernels (CUDA and OpenCL) which can be compiled and run asynchronously on arbitrary devices in a heterogeneous system.

We have shown the successful application of HPX in geometric decomposition, storm surge forecasting, linear algebra, astrophysics, crack and fracture mechanics, and computational fluid dynamic. As one example, HPX showed perfect scaling for the three dimensional N body benchmark within LibGeoComp at the single node level (98%) and (89%) peak performance on the Xeon Phi Knights Corner coprocessor and could outperform MPI implementation by a factor of 1.4. For the astrophysics study of merging double white dwarf binary star systems, a parallel efficiency of 96.8% on NERSC's Cori using 643,280 Intel Knights Landing cores was demonstrated using billions of lightweight threads.

REFERENCES

[1] David A. Bader. 2016. Evolving MPI+X Toward Exascale. *Computer* 49, 8 (2016), 10. https://doi.org/doi.ieeecomputersociety.org/10.1109/MC.2016.232

[2] Henry C. Baker and Carl Hewitt. 1977. The incremental garbage collection of processes. In *SIGART Bull*. ACM, New York, NY, USA, 55–59. Issue 64. https://doi.org/10.1145/872736.806932

[3] John Biddiscombe, Thomas Heller, Anton Bikineev, and Hartmut Kaiser. 2017. Zero Copy Serialization using RMA in the Distributed Task-Based HPX runtime. In *14th International Conference on Applied Computing*. IADIS.

[4] Z. D. Byerly, H. Kaiser, S. Brus, and A. Schäfer. 2015. A Non-intrusive Technique for Interfacing Legacy Fortran Codes with Modern C++ Runtime Systems. In *2015 Third International Symposium on Computing and Networking (CANDAR).*

[5] Marcin Copik and Hartmut Kaiser. 2017. Using SYCL As an Implementation Framework for HPX.Compute. In *Proceedings of the 5th International Workshop on OpenCL (IWOCL 2017)*. ACM, New York, NY, USA, Article 30, 7 pages. https://doi.org/10.1145/3078155.3078187

[6] J. B. Dennis. 1980. Data Flow Supercomputers. *Computer* 13, 11 (1980), 48–56. https://doi.org/10.1109/MC.1980.1653418

[7] Jack B. Dennis and David Misunas. 1998. A Primlinary Architecture for a Basic Data-Flow Processor. In *25 Years ISCA: Retrospectives and Reprints*. 125–131.

[8] James Dinan, Pavan Balaji, David Goodell, Douglas Miller, Marc Snir, and Rajeev Thakur. 2013. Enabling MPI interoperability through flexible communication endpoints. In *Proceedings of the 20th European MPI Users' Group Meeting*. ACM, 13–18.

[9] Daniel P. Friedman and David S. Wise. 1976. CONS Should Not Evaluate its Arguments. In *ICALP*. 257–284.

[10] Patricia Grubel, Hartmut Kaiser, Kevin Huck, and Jeanine Cook. 2016. Using Intrinsic Performance Counters to Assess Efficiency in Task-based Parallel Applications. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE, 1692–1701.

[11] Thomas Heller, Hartmut Kaiser, Patrick Diehl, Dietmar Fey, and Marc Alexander Schweitzer. 2016. *Closing the Performance Gap with Modern C++*. Springer International Publishing, Cham, 18–31. https://doi.org/10.1007/978-3-319-46079-6_2

[12] Thomas Heller, Hartmut Kaiser, Juhan Frank, Dominic Marcello, Adrian Serio, Bryce Adelstein Lelbach, Alice Koniges, Kevin A. Huck, Patricia Grubel, Matthias Kretz, and John Biddiscombe. 2017. *Harnessing Billions of Tasks for a Scalable Portable Hydrodynamics Simulation of the Merger of Two Stars*. Technical Report.

[13] Thomas Heller, Hartmut Kaiser, Andreas Schäfer, and Dietmar Fey. 2013. Using hpx and libgeodecomp for scaling hpc applications on heterogeneous supercomputers. In *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. ACM, 1.

[14] Hartmut Kaiser, Maciej Brodowicz, and Thomas Sterling. 2009. ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications. In *Parallel Processing Workshops*. IEEE Computer Society, Los Alamitos, CA, USA, 394–401. https://doi.org/10.1109/ICPPW.2009.14

[15] Hartmut Kaiser, Thomas Heller, Bryce Adelstein-Lelbach, Adrian Serio, and Dietmar Fey. 2014. 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)*. ACM, New York, NY, USA, Article 6, 11 pages. https://doi.org/10.1145/2676870.2676883

[16] Hartmut Kaiser, Thomas Heller, Daniel Bourgeois, and Dietmar Fey. 2015. Higher-level Parallelization for Local and Distributed Asynchronous Task-based Programming. In *Proceedings of the First International Workshop on Extreme Scale Programming Models and Middleware (ESPM '15)*. ACM, New York, NY, USA, 29–37. https://doi.org/10.1145/2832241.2832244

[17] Richard A Luettich Jr, Johannes J Westerink, and Norman W Scheffner. 1992. *AD-CIRC: An Advanced Three-Dimensional Circulation Model for Shelves, Coasts, and Estuaries. Report 1. Theory and Methodology of ADCIRC-2DDI and ADCIRC-3DL*. Technical Report. COASTAL ENGINEERING RESEARCH CENTER VICKSBURG MS.

[18] P. E. Ross. 2008. Why CPU Frequency Stalled. *IEEE Spectrum* 45, 4 (4 2008), 72–72. https://doi.org/10.1109/MSPEC.2008.4476447

[19] Herb Sutter. 2005. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's journal* 30, 3 (2005), 202–210.

[20] Antoine Tran Tan and Hartmut Kaiser. 2016. Extending C++ with Co-array Semantics. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY 2016)*. ACM, New York, NY, USA, 63–68. https://doi.org/10.1145/2935323.2935332

503–507. https://doi.org/10.1109/CANDAR.2015.71