Parallel Programming Paradigms

The Author

1 Parallel Programming Paradigms and the PGAS Model

Parallel computing has evolved through a rich set of paradigms, each reflecting assumptions about memory visibility, communication, and coordination. Among these, the Partitioned Global Address Space (PGAS) model occupies a unique position, offering a compelling blend of abstraction and control.

1.1 The PGAS Model: Bridging Shared and Distributed Memory

The Partitioned Global Address Space (PGAS) model assumes a globally addressable memory space that is logically shared but physically partitioned. Each processing element (or image) can access all memory, but local memory accesses are faster than remote ones. This model promotes locality-aware programming while avoiding the full complexity of message-passing.

Characteristic	**PGAS Position**
Abstraction level	Medium to High
Memory model	Shared (logical) / Distributed (physical)
Communication model	One-sided, asynchronous preferred
Locality control	Explicit
Ease of use	Higher than MPI, lower than shared memory
Scalability	High

Table 1: Core attributes of PGAS

1.2 Examples of PGAS Languages and Systems

- **Coarray Fortran** (CAF): Integrated into modern Fortran standards
- **UPC / UPC++**: Unified Parallel C variants
- **Chapel**: Cray's open-source, high-productivity language
- **X10**: Developed by IBM for async programming
- **Legion / Regent**: Uses logical regions and task graphs
- **OpenSHMEM**: Portable interface for PGAS-style programming

1.3 Comparison with Other Paradigms

Paradigm	**Memory Model**	**Comm. Style**	**Typical Use**	**Examples**
Shared Memory	Global, shared	Implicit	SMP systems	OpenMP, Cilk
Message Passing	Distributed	Explicit (send/recv)	HPC clusters	MPI
PGAS	Global (logical)	One-sided, explicit	Hybrid systems	Coarrays, UPC++
Task-Based	Mixed	Task-graph-based	Irregular, dynamic apps	Legion, PaRSEC
Data Parallel	Global (flat)	Compiler-managed	GPUs, vector ops	CUDA, OpenCL
Actor/Functional	Distributed	Message passing	Reactive, fault-tolerant apps	Erlang, Akka

Table 2: Comparison of major parallel programming paradigms

1.4 PGAS in Fortran: Compiler-Native Parallelism

Modern Fortran has embraced PGAS through **coarrays**, a language-native mechanism introduced in Fortran 2008 and enhanced in Fortran 2018 and 2023. Coarrays allow variables to be indexed across images using square brackets, e.g., `A[2]` to access a remote copy of `A`.

Unlike MPI, which requires separate libraries and explicit send/receive calls, coarrays are integrated into the language and leverage compiler-managed communication. They support both SPMD (single program, multiple data) and collective synchronization constructs.

1.5 The Evolutionary Path: From Fortran to Coarrays to Teams

- **Fortran 77 and earlier**: Lacked native parallel constructs. Parallelism handled through external tools like OpenMP or MPI.
- **Fortran 90/95**: Introduced modules, array syntax, and better structure for scientific computing.
- **Fortran 2008**: Introduced **coarrays**, making Fortran a true PGAS language.
- **Fortran 2018/2023**: Enhanced coarray capabilities, including **events**, **teams**, and improved collective operations.

Coarrays represent Fortran's shift from being a vehicle for numerical methods to a language capable of expressing parallelism directly. Teams and collectives in Fortran 2018+ bring the power of structured communication patterns into scientific codes.

1.6 Academic References for Legion

Legion is a PGAS-flavored task-based model. Notable references:

- M. Bauer, S. Treichler, E. Slaughter, A. Aiken. *Legion: Expressing Locality and Independence with Logical Regions.* SC'12. DOI:10.1109/SC.2012.71
- S. Treichler, M. Bauer, A. Aiken. *A Case for Task-Based Programming Models.* HotPar'14.
- The Legion programming site: `https://legion.stanford.edu/`

Legion enables programmers to express independence and locality explicitly, making it well-suited to exascale-class problems. It underlies **Legate**, a NumPy-like interface designed to run on clusters with GPUs or CPUs.

1.7 Conclusion: PGAS as a Compass in the Parallel Programming Landscape

PGAS is not a halfway point — it is a paradigm that enables clean reasoning about data locality, memory layout, and one-sided communication, while retaining scalable performance. Its adoption in Fortran (through coarrays) represents a natural evolution that aligns with the needs of modern simulation codes.

As HPC evolves toward exascale and heterogeneous architectures, PGAS and task-based models like Legion offer compelling paths forward — marrying productivity with performance.