Babeș-Bolyai University

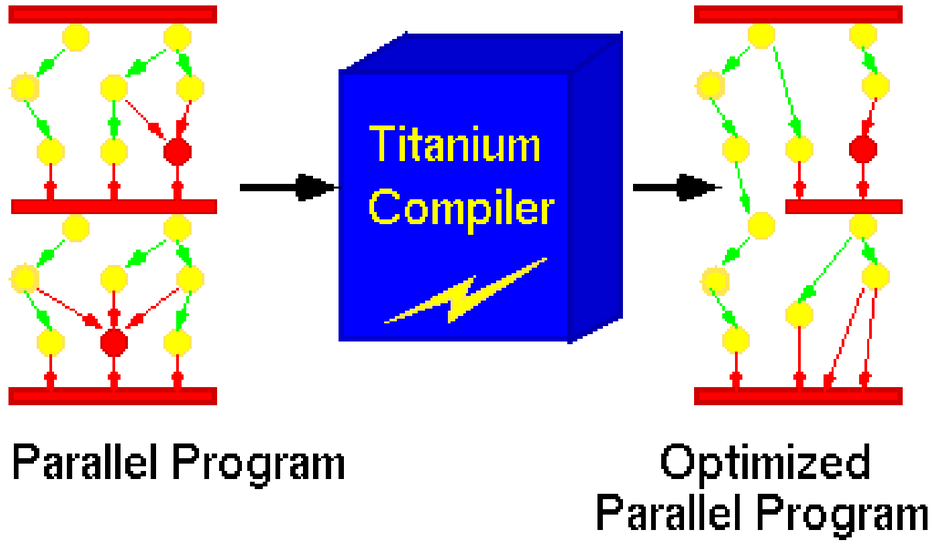
TITANIUM A JAVA-BASED PGAS PARALLEL PROGRAMMING MODEL

Author: Turcu Ciprian-Stelian

WHAT IS TITANIUM?

What Are Structural Holes?

- Java-based parallel programming language
- Focus on scientific computing applications
- Combines high-level abstractions with low-level control
- Developed at UC Berkeley (1995)



Picture from https://titanium.cs.berkeley.edu/

THE PROBLEM TITANIUM SOLVED

- Shift from custom supercomputers to commodity clusters
- Need for portable parallel programming
- Java lacked HPC features
- Balance between productivity and performance

CORE ARCHITECTURE - SPMD MODEL

- All ranks run identical code, operate on distinct data slices
- Example: 4 ranks processing different segments of a large array
- Barrier-based synchronization ensures collective progress
- Simple scaling: increase ranks, assign data partitions, insert barriers

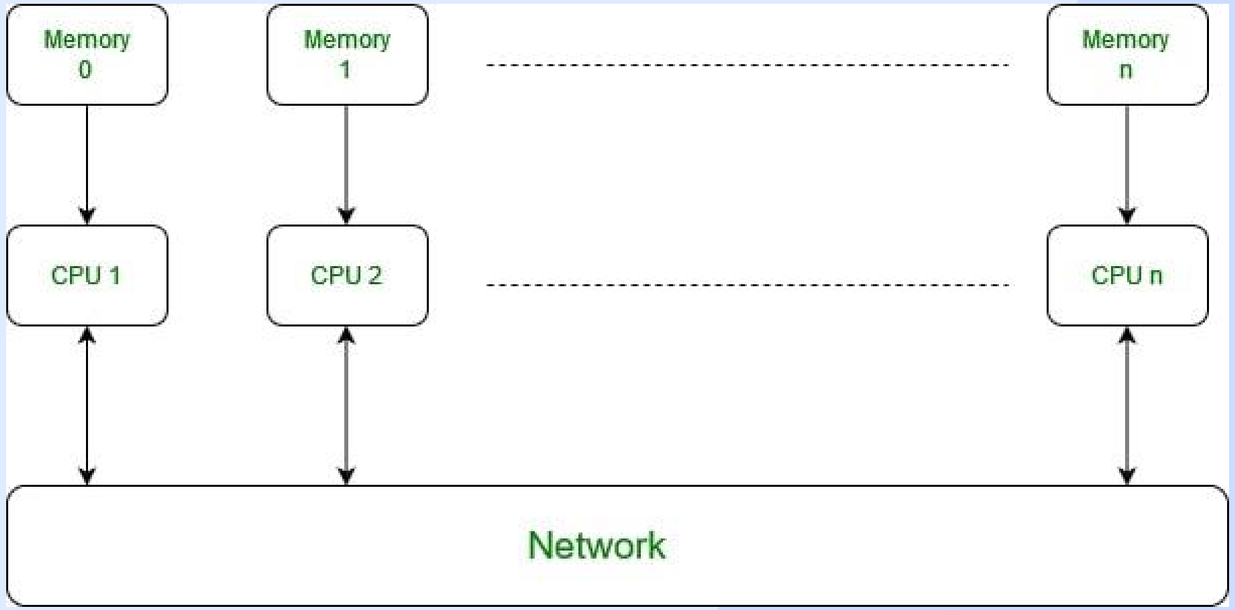


Image from: https://www.geeksforgeeks.org/single-program-multiple-data-spmd-model/

CORE ARCHITECTURE - PGAS MEMORY MODEL

- Partitioned Global Address Space
- Logically shared, physically distributed
- Local access = fast, remote access = network transfer
- Transparent remote memory access

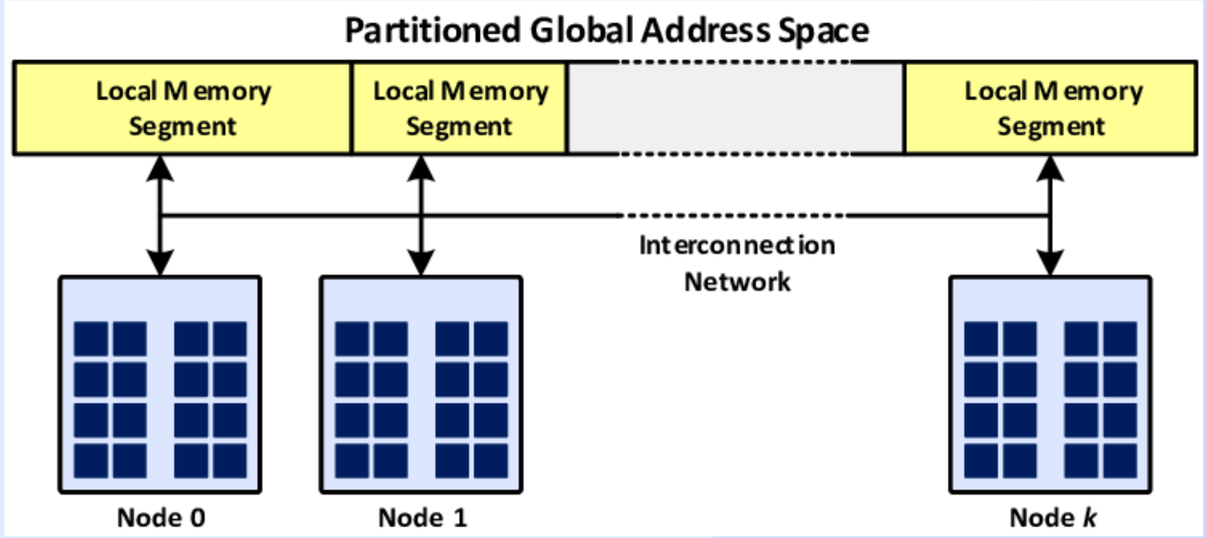


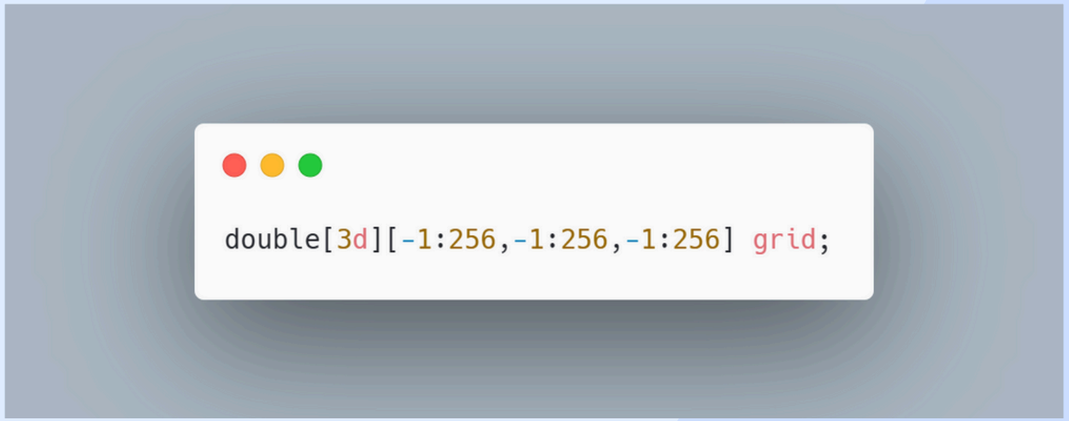
Image from https://www.researchgate.net/figure/Segmented-partitioned-global-address-space-in-GASPI_fig1_355238912:

HELLO WORLD EXAMPLE

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello from proc " + Ti.thisProc() + " out of " + Ti.numProcs());
        Ti.barrier();
    }
}
```

LANGUAGE FEATURES - MULTI-DIMENSIONAL ARRAYS

- True multi-dimensional arrays
- Flexible index bounds (including negative)
- Domain operations (shrink, expand)
- Natural scientific computing support



SYNCHRONIZATION AND SAFETY

- **single** qualifier for synchronized variables
- Barrier alignment requirement
- Prevents common parallel programming errors
- Compile-time and runtime checks

LOCAL VS GLOBAL POINTERS

- Global pointers (default) - can point anywhere
- Local pointers - guaranteed local data
- Compiler optimizations for local access
- Performance transparency

COMPARISON WITH OTHER MODELS

1

Unified Parallel C (UPC):

- PGAS in C; shared arrays with block distribution, upc_forall loops
- Titanium: Java OO, per-process local blocks + global pointers, requires explicit Exchanges

2

Chapel:

- High-level global-view arrays with distribution specs; integrated parallel loops, tasks
- Titanium: Lower-level SPMD + PGAS; programmer-centric data placement & explicit synchronization

3

MPI:

- Explicit send/receive, no global address space; ultimate control but heavy boilerplate
- Titanium: Implicit one-sided PGAS accesses, collectives, safety checks, Java syntax

ADVANTAGES & CHALLANGES OF TITANIUM

Advantages:

- High productivity: concise code, rich abstractions (domains, arrays, collectives)
- Java-like safety: type safety, memory safety, barrier alignment checks
- Performance: Near-MPI performance via GASNet optimizations
- Portability: Run unmodified on SMP or clusters
- Support for irregular data structures via global pointers

Challenges:

- Steep learning curve: barrier alignment, “single” qualifiers, region-based memory
- Explicit local/global bookkeeping: programmer-managed exchanges & pointers
- Limited ecosystem: fewer libraries, IDE support, community compared to C/Fortran
- Debugging/profiling: fewer mature tools vs. MPI profiler suites

```
public class ParallelSum {
    public static void main(String[] args) {
        // Get process info
        int numProcs = Ti.numProcs();
        int myRank = Ti.thisProc();

        // Create local array - each process gets different portion
        int localSize = 1000;
        double[1d] localArray = new double[1d][0:localSize-1];

        // Initialize with different values per process
        foreach (Point<1> p in localArray.domain()) {
            localArray[p] = myRank * 100 + p[0];  // Unique values
        }

        // Compute local sum
        double localSum = 0.0;
        foreach (Point<1> p in localArray.domain()) {
            localSum += localArray[p];
        }

        // Global reduction to get total sum
        double globalSum = Ti.reduce(Ti.ADD, localSum);

        // Master process prints result
        if (myRank == 0) {
            System.out.println("Total sum across " + numProcs + " processes: " + globalSum);
        }

        Ti.barrier();  // Synchronize before exit
    }
}
```

CONCLUSION

- Successful combination of productivity and performance
- Pioneered many PGAS concepts still used today
- Demonstrated viability of high-level HPC languages
- Continues to influence modern parallel programming

Babeș-Bolyai University

THANK YOU

Turcu Ciprian-Stelian | 2025