

PGAS in the Message-Driven Execution Model

Aaron Becker, Philip Miller, Laxmikant Kale
Parallel Programming Lab
Department of Computer Science
University of Illinois at Urbana-Champaign

Many Incomplete Languages¹

General-purpose languages are problematic

Simpler languages lose completeness

Compose them!

Gain safety, targeted optimization, productivity

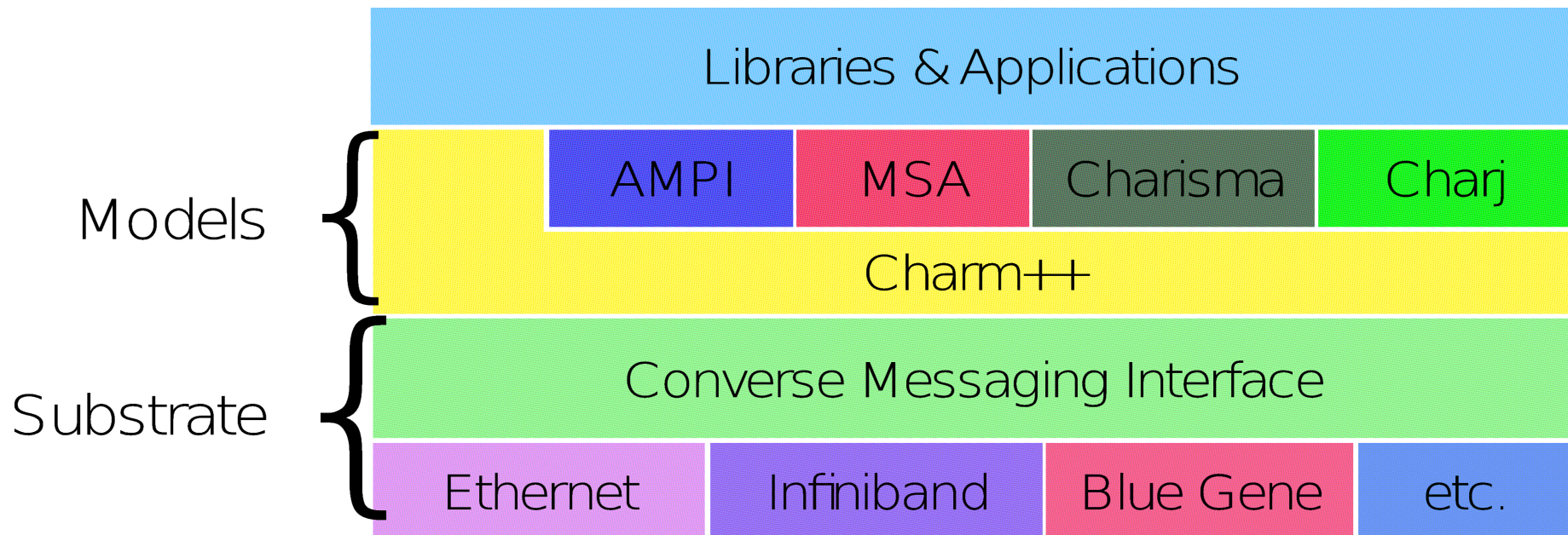
1: Chakravorty et al. LCPC '08

Message-Driven Execution

Solid common baseline

Inherently overlaps computation with communication

Charm++ ecosystem has languages, libraries, tools, and a record of scalable applications



Distributed Shared Arrays

Application usage of shared arrays have *phases*

Jacobi relaxation: old and new, switching;

Matrix multiply: operands, result;

etc.

How can we exploit this?

Phases in Programming Model²

Identify common access modes

Codify them

Derive safety properties and checks

Optimize with better knowledge

2: DeSouza & Kale, LCPC '04

Phases in Programming Model

Read-only: cache

Write-once: allocate fresh, and flush

Accumulation: reduction

Owner/Local: direct storage access

Read-Modify-Write: cache and flush

Safety

Avoid race conditions

Defined phases avoid race conditions

Synchronize at changeover

Enforcement

Static checks

- encode phase in types

- expose operations matching current phase

Dynamic checks:

- type matches current phase

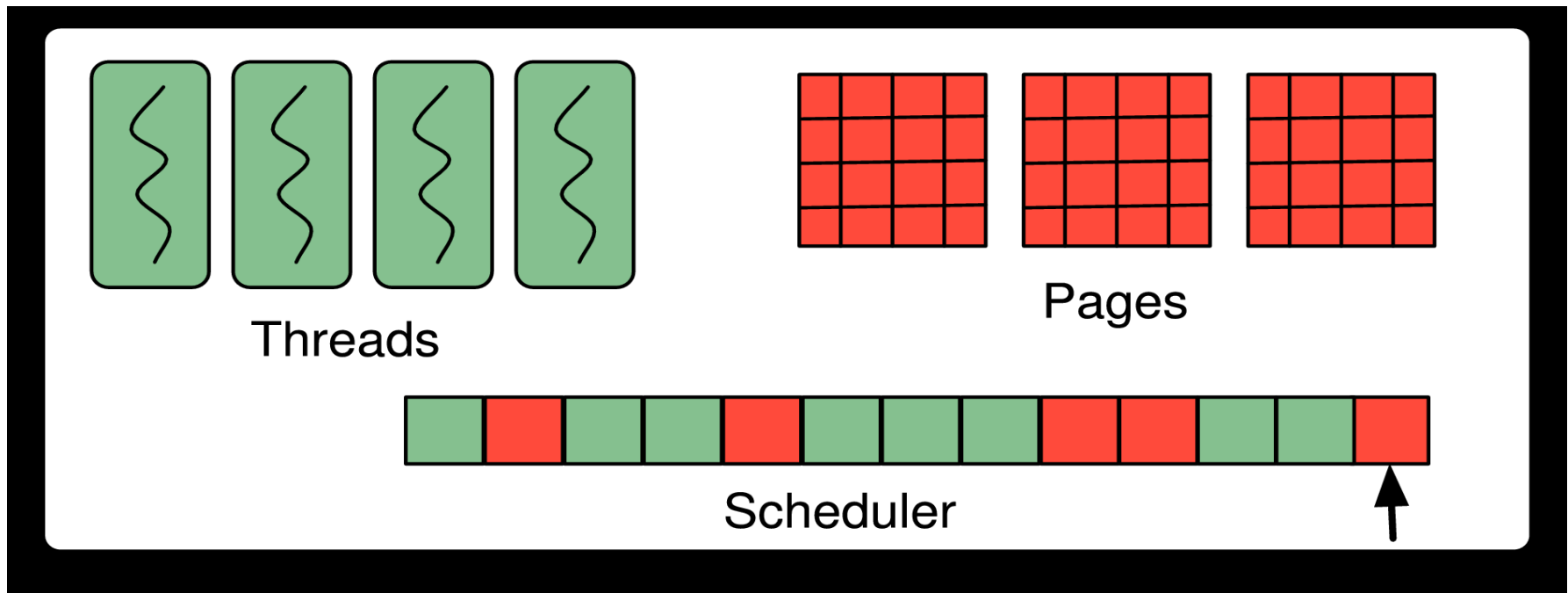
- exclusivity/uniqueness isn't violated

Asynchrony

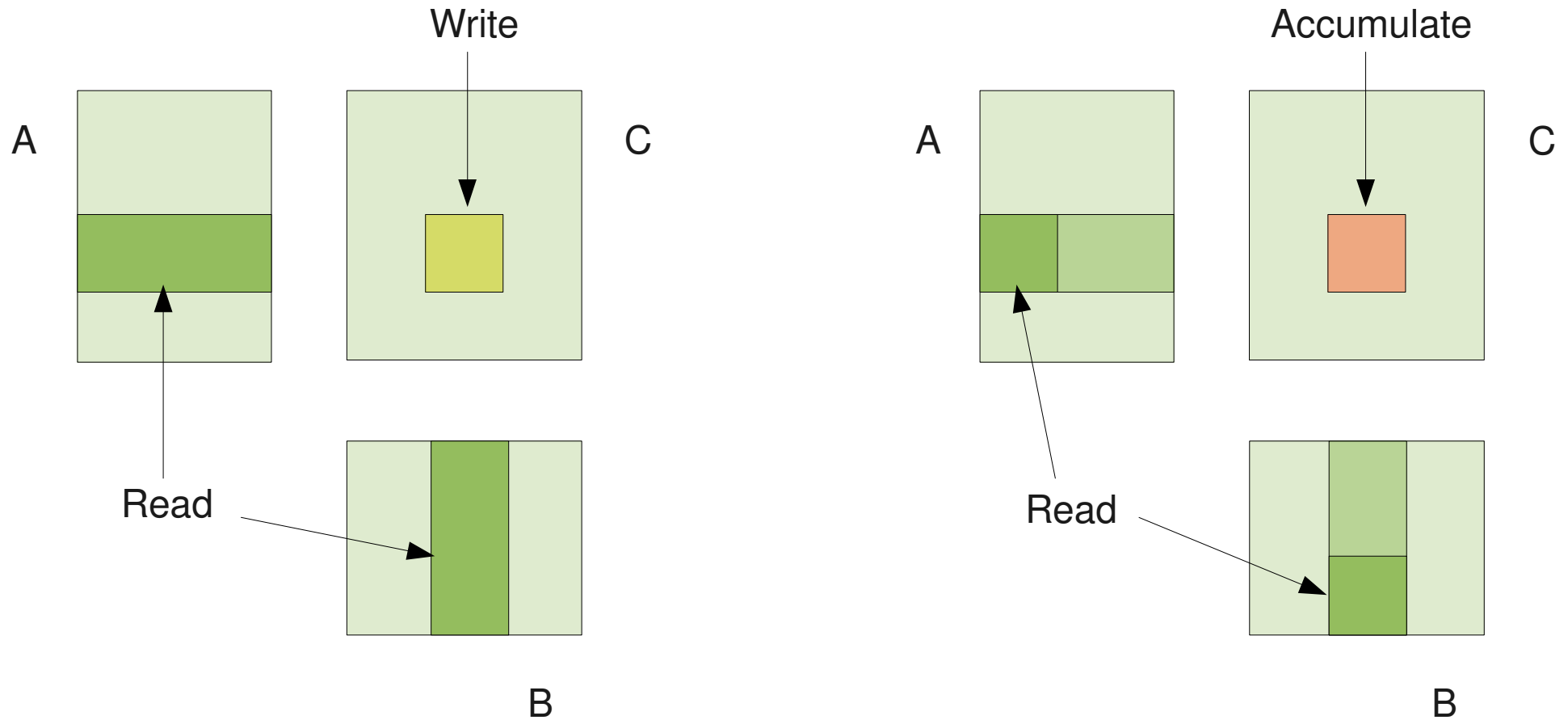
On remote access, fetch a suitable chunk into local cache

Let another thread work while waiting

Depends on overdecomposition



Matrix-Matrix Multiply Two Ways



Matrix-Matrix Multiply

In code

```
typedef MSA2D<double, MSA_NullA<double>, 5000,MSA_ROW_MAJOR> MSA2DRowMjr;  
typedef MSA2D<double, MSA_SumA<double>, 5000,MSA_COL_MAJOR> MSA2DCoIMjr;
```

```
MSA2DRowMjr arr1(ROWS1, COLS1, NUMWORKERS, cacheSize1); // row major  
MSA2DCoIMjr arr2(ROWS2, COLS2, NUMWORKERS, cacheSize2); // column major  
MSA2DRowMjr prod(ROWS1, COLS2, NUMWORKERS, cacheSize3); //product matrix
```

```
for(unsigned int c = 0; c < COLS2; c++) {  
    // Each thread computes a subset of rows of product matrix  
    for(unsigned int r = row_begin; r <= row_end; r++) {  
        double result = 0.0;  
        for(unsigned int k = 0; k < cols1; k++)  
            result += arr1[r][k] * arr2[k][c];  
        prod[r][c] = result;  
    }  
}
```

Thread-local accumulation

prod[r][c] = result; **Global memory access**

prod.sync(); **New phase for matrix prod**

```
void PlimptonMD(CoordMSA::Handle &hCoords,
                XyzMSA::Handle &hForces,
                AtomInfoMSA::Read &rAtominfo) {
    for (int timestep = 0; timestep < NUM_TIMESTEPS; timestep++) {
        // Force calculation for a section of the interaction matrix
        XyzMSA::Accum aForces = forces.syncToAccum(hForces);
        CoordMSA::Read rCoords = coords.syncToRead(hCoords);
        for (int i = i_start; i < i_end; ++i)
            for (int j = j_start; j < j_end; ++j) {
                XYZ force = calculateForce(rCoords(i), rAtominfo(i),
                                           rCoords(j), rAtominfo(j));
                aForces(i) += force;
                aForces(j) += force.negate(); }
        // Movement Integration for our subset of atoms
        XyzMSA::Read rForces = forces.syncToRead(aForces);
        CoordMSA::Owner oCoords = coords.syncToOwner(rCoords,
        Integrator(rAtominfo, rForces));
        hCoords = oCoords;
        hForces = rForces;
    } }
```

Optimization

Prefetching and cache control

Compile-time

- Access check elimination

- Strip-mining

Run-time

- Communication patterns

- Usage patterns

References

DeSouza and Kale. “MSA: Multiphase Specifically Shared Arrays”. LCPC 2004.

Chakravorty et al. “A Case Study in Tightly Coupled Multiparadigm Parallel Programming”. LCPC 2008

Scales, Gharachorloo & Thekkath. “Shasta: a low overhead, software-only approach for supporting fine-grain shared memory”. ASPLOS 1996

Carter. “Design of the Munin distributed shared memory system”. *Journal of Parallel and Distributed Computing*, 1995.

Keleher et al. “Treadmarks: Distributed shared memory on standard workstations and operating systems.” USENIX 1994.