Communication-Aware Diffusion Load Balancing for Persistently Interacting Objects

Maya Taylor, Kavitha Chandrasekar, Laxmikant Kale

Department of Computer Science

University of Illinois at Urbana-Champaign

Dynamic Load Balancing

- Parallel applications with irregular and time- varying workloads often suffer from load imbalance.
- A potential solution: dynamic load balancing
  - the migration of work across processors at runtime, typically in an iterative context
- Various approaches to dynamic load balancing exist:
  - application developers manually balance load
  - runtime systems automatically handle load balancing

The Charm++ Runtime System

- Charm++ is a programming model and a parallel runtime system that supports dynamic load balancing
- It is one of the foundational programming models for the ‘migratable objects’ framework
- A Charm++ program consists of collections of user-defined C++ objects that interact with each other via asynchronous method invocation
- The Charm++ runtime handles:
  - mapping of objects to processors
  - scheduling of method invocations on each processor
  - load balancing

Problem Definition

Given:

- A set of interacting objects, each with:
  - a current processor assignment
  - associated computational load
- A sparse graph of weighted communication edges representing interactions between objects.

The objective: compute a new mapping accounting for…

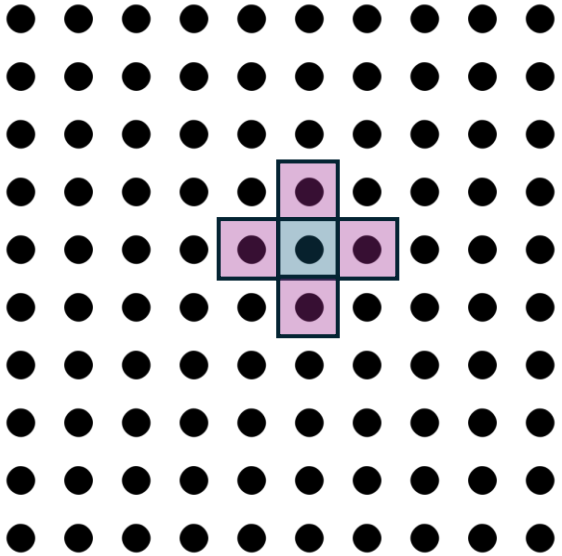
- load imbalance (ratio of maximum to average processor load)
- communication cost (ratio of inter-node communication to intra-node communication)
- number of migrations
- the cost of computing the mapping itself.

Our Contribution

A distributed load balancing algorithm that

- operates on a communication graph of objects and their processor mappings
- assumes initial object mappings prioritize intra-PE communication
- computes new object placing focused on
  - keeping migrations minimal
  - balancing load
  - and preserving communication locality

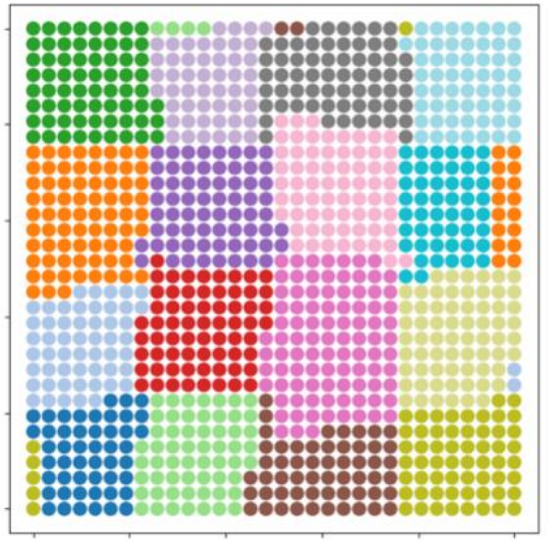
Intuition via a Stencil Application



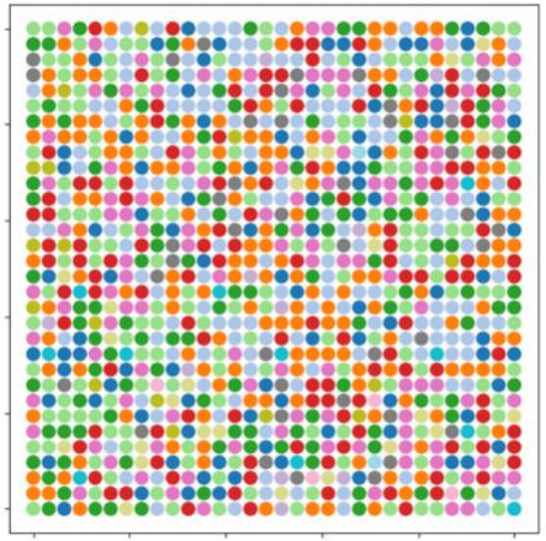
- 2D 5-point stencil application
- Each object updates its value based on its own state and the states of its four immediate neighbors (north, south, east, and west).

Intuition via a Stencil Application

- 2D grid of migratable objects
- Each object resides on a processor (denoted by color)
- Communication locality visualized by contiguous blocks of color



good locality



poor locality

The ‘Diffusion’ Approach

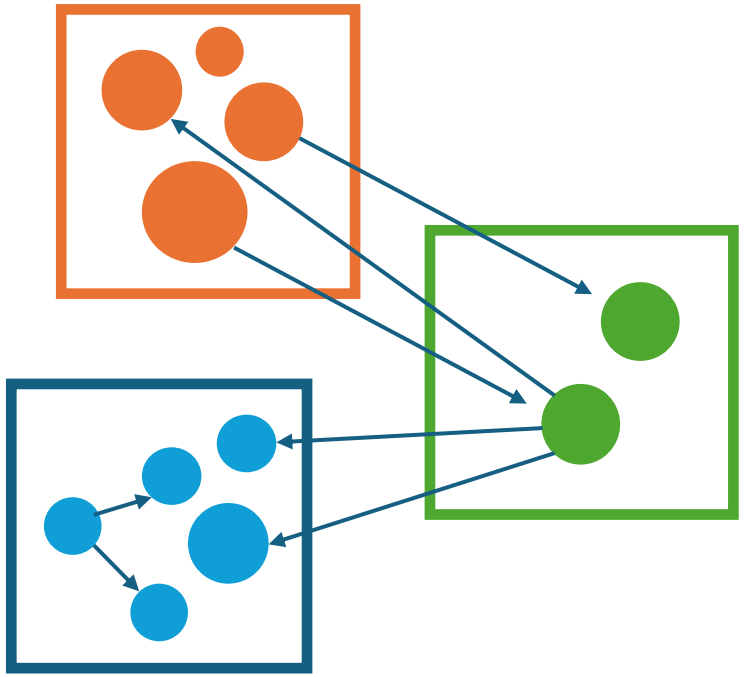
- Given a mapping with good locality but poor load balance…
- Don’t compute a new mapping from scratch!
- Instead, diffuse load across existing communication edges
  - minimize migrations
  - approach better load balance

The Three-Stage Diffusion Algorithm

1. Construct a processor neighbor graph based on an application's communication patterns
2. Compute the ideal diffusive load redistribution at the processor level, without considering object granularity
3. Select specific objects for migration

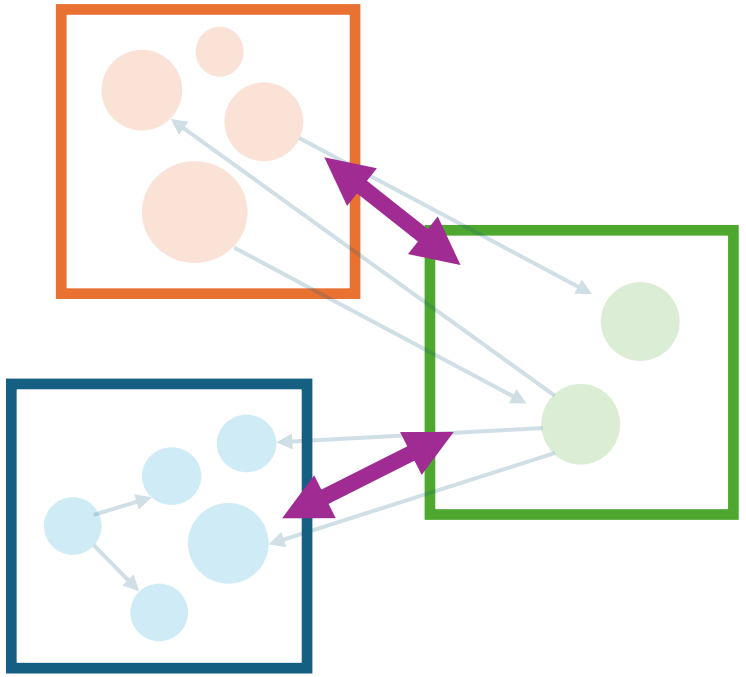
Stage 1: Neighbor Selection

- Given an initial mapping of objects to processors and their communication history
- Every processor selects some k neighbors with which they will exchange objects



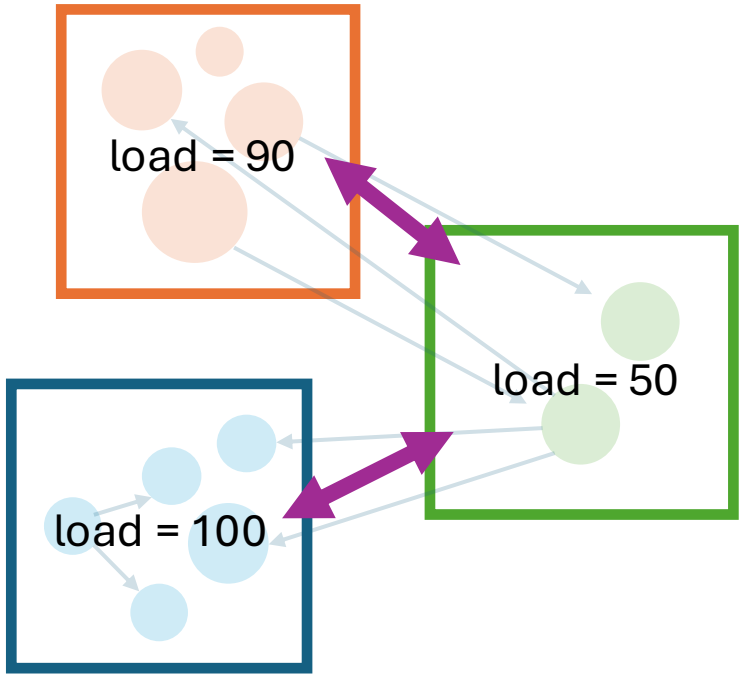
Stage 1: Neighbor Selection

- Given an initial mapping of objects to processors and their communication history
- Every processor selects some k neighbors with which they will exchange objects



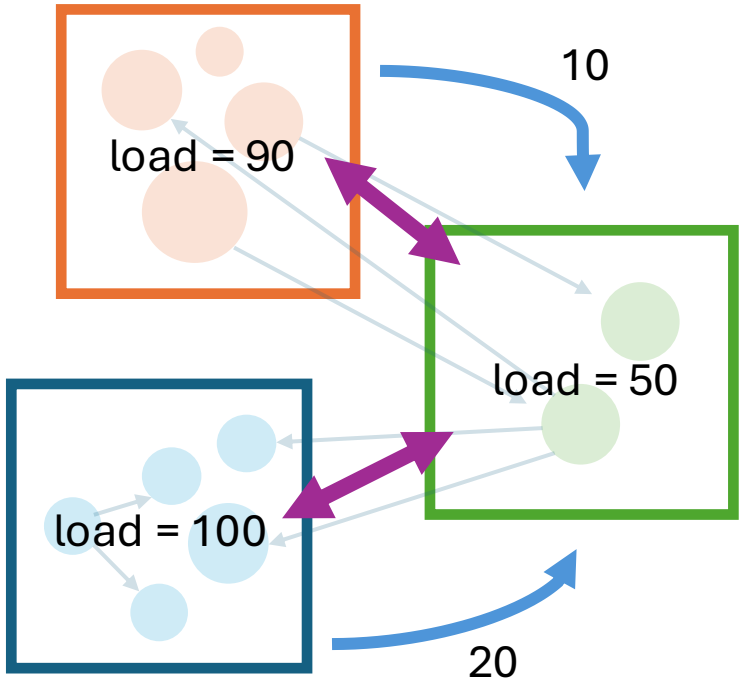
Stage 2: Virtual Load Balancing

- Processors collectively compute the ‘ideal’ redistribution of load across the neighbor graph
- This phase ignores object granularity



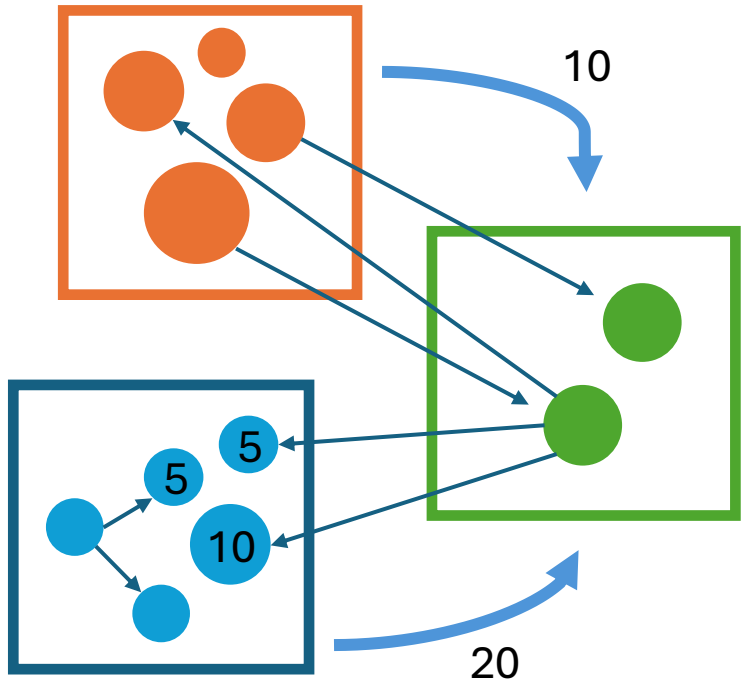
Stage 2: Virtual Load Balancing

- Processors collectively compute the ‘ideal’ redistribution of load across the neighbor graph
- This phase ignores object granularity



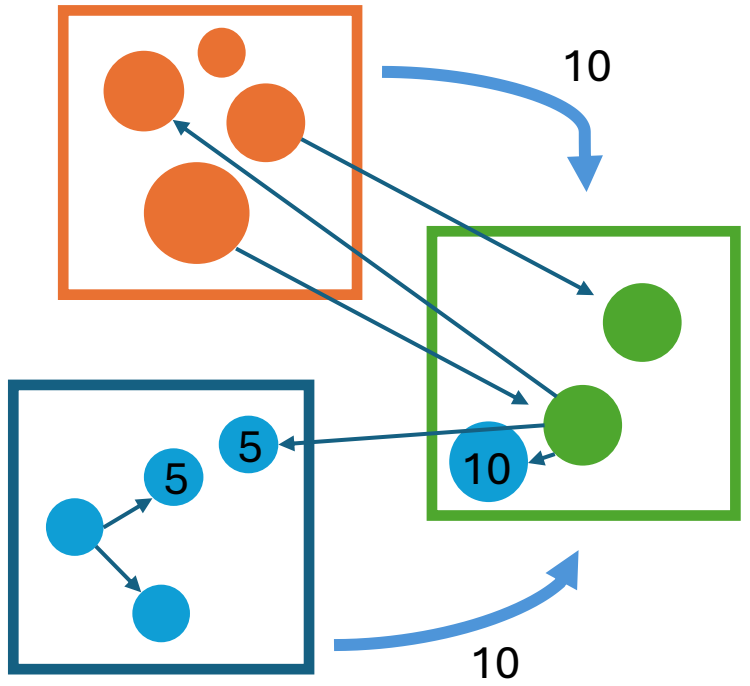
Stage 3: Object Selection

- Processors choose objects to migrate to approach the optimal load transfer found in stage 2
- Object selection is communication-aware: when possible, objects that already communicate with a given neighbor are migrated there



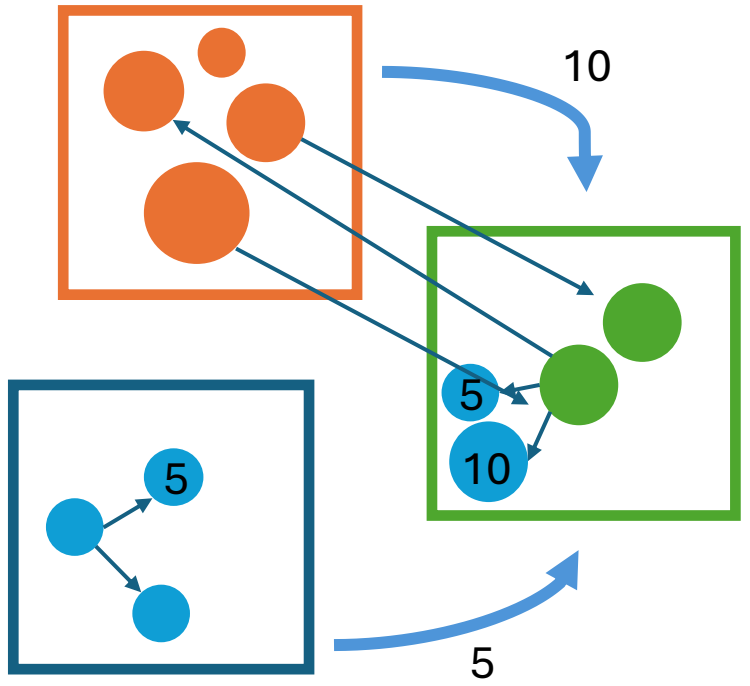
Stage 3: Object Selection

- Processors choose objects to migrate to approach the optimal load transfer found in stage 2
- Object selection is communication-aware: when possible, objects that already communicate with a given neighbor are migrated there



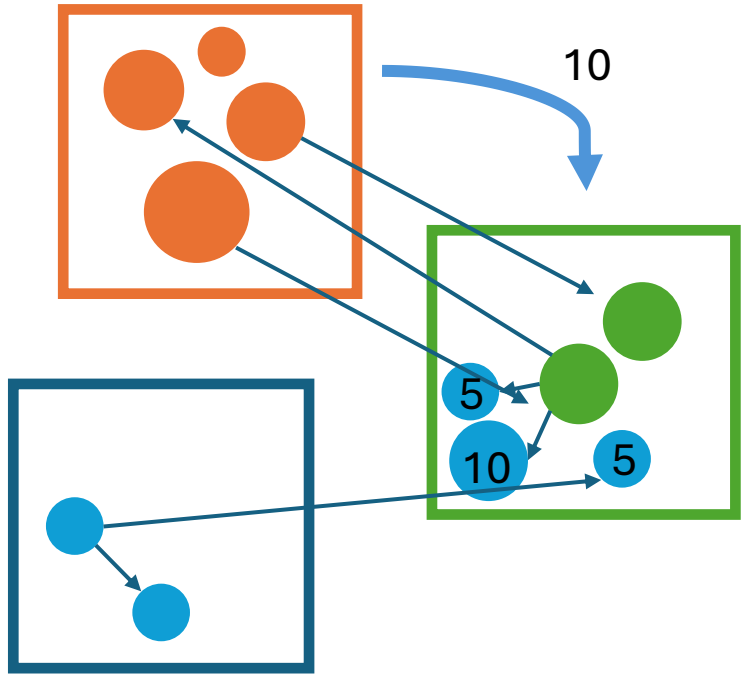
Stage 3: Object Selection

- Processors choose objects to migrate to approach the optimal load transfer found in stage 2
- Object selection is communication-aware: when possible, objects that already communicate with a given neighbor are migrated there



Stage 3: Object Selection

- Processors choose objects to migrate to approach the optimal load transfer found in stage 2
- Object selection is communication-aware: when possible, objects that already communicate with a given neighbor are migrated there



Metric Prioritization

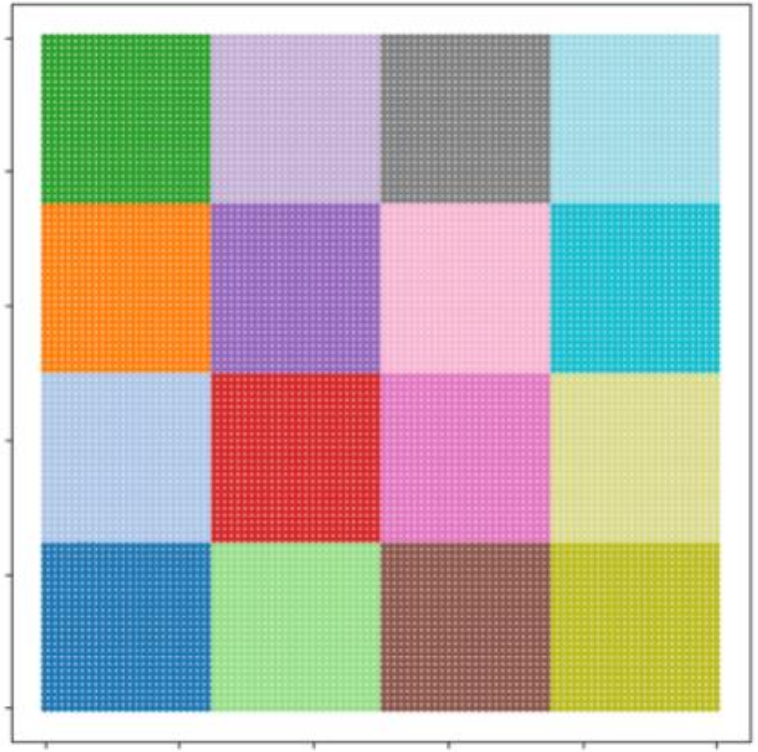
- Diffusion prioritizes load balance over other metrics
- Aims to align migrations with communication patterns as much as possible

Centroid-based Diffusion

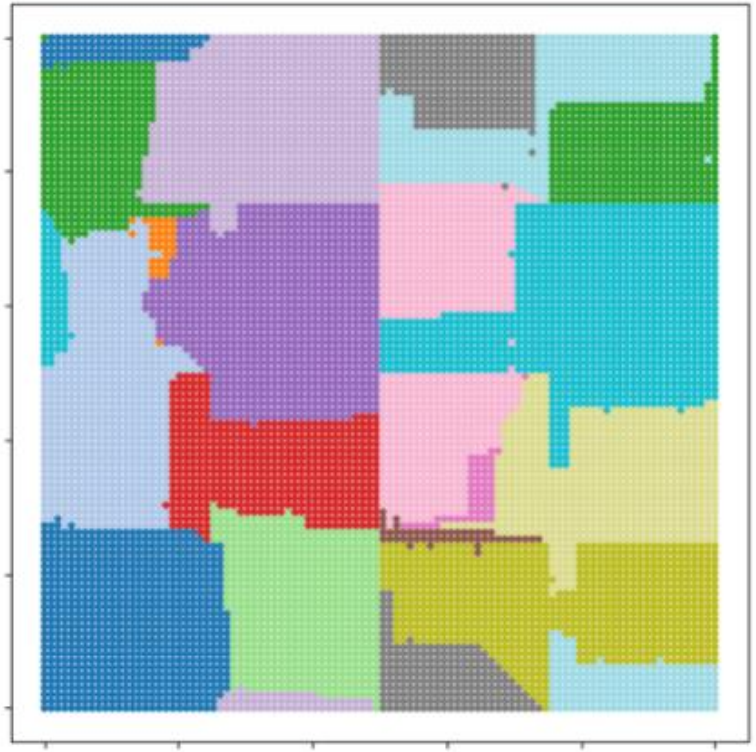
What if the communication patterns are unavailable?

- We propose an approximation to communication patterns based on object ‘coordinates’
- Coordinates are specified by the application developer such that distance between objects is inversely correlated with communication
- This is feasible particularly in applications with a physical representation corresponding to communication locality, such as particle simulations

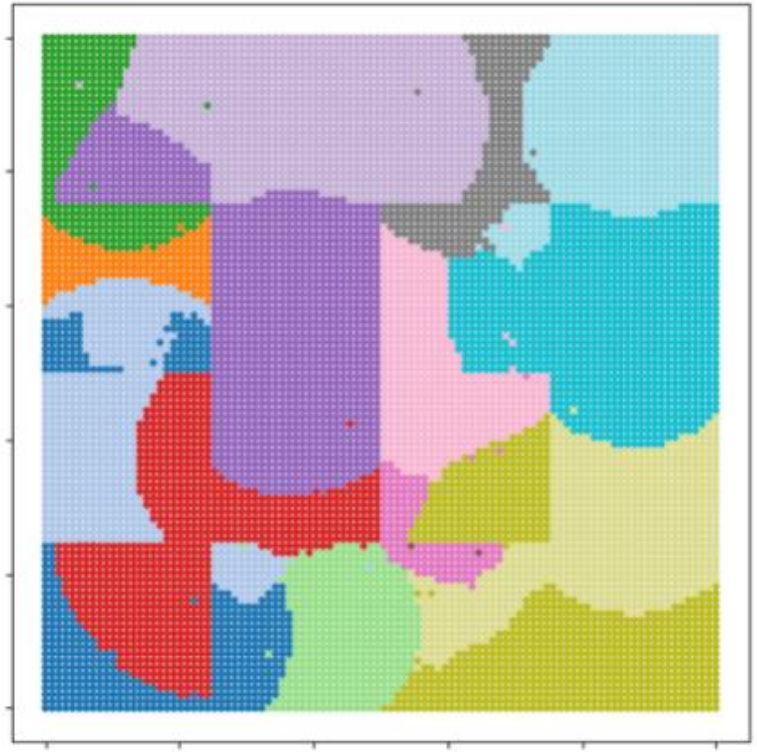
Stencil with Synthetic Load Imbalance



original mapping



communication-based diffusion



coordinate-based diffusion

Simulation Infrastructure

For initial evaluation, we implement a load balancing simulation infrastructure using Charm++

- Input: object loads, coordinates, and communication edges
- Supports injection of synthetic load imbalance injection
- Models various load balancing strategies independent of the application
- Simulates any number of processes without requiring true at- scale execution.

Evaluation in Simulation

Metric	Initial	GreedyRefine	METIS	ParMETIS	Diff-Comm	Diff-Coord
Benchmark 1: 8 PEs						
max/avg load	1.32	1.00	1.00	1.01	1.06	1.05
external/internal comm (MB)	.5	.887	.343	.637	.58	.648
% migrations	-	19.9%	87.1%	14.6%	18.9%	17%
Benchmark 2: 32 PEs						
max/avg load	1.37	1.00	1.00	1.04	1.02	1.02
external/internal comm (MB)	.143	.422	.158	.194	.173	.227
% migrations	-	18.7%	98.9%	6.6%	15.4%	17.6%
Benchmark 3: 128 PEs						
max/avg load	1.37	1.00	1.00	1.20	1.09	1.14
external/internal comm (MB)	.6	.999	.217	.602	.625	.616
% migrations	-	18.8%	99%	10.8%	18.2%	17.8%

TABLE II: Comparison of load balance metrics across 5 algorithms and synthetic benchmarks with a 3D stencil communication patterns. All benchmarks use the same synthetic load imbalance injection, wherein every 1st and 2nd PEs mod 7 is overloaded, and every 3rd mod 7 is underloaded.

Evaluation in Simulation

Metric	Initial	GreedyRefine	METIS	ParMETIS	Diff-Comm	Diff-Coord
Benchmark 1: 8 PEs						
max/avg load	1.32	1.00	1.00	1.01	1.06	1.05
external/internal comm (MB)	.5	.887	.343	.637		
% migrations	-	19.9%	87.1%	14.6%		
Benchmark 2: 32 PEs						
max/avg load	1.37	1.00	1.00	1.04		
external/internal comm (MB)	.143	.422	.158	.194		
% migrations	-	18.7%	98.9%	6.6%		
Benchmark 3: 128 PEs						
max/avg load	1.37	1.00	1.00	1.20		
external/internal comm (MB)	.6	.999	.217	.602		
% migrations	-	18.8%	99%	10.8%		

TABLE II: Comparison of load balance metrics across 5 algorithms and synthetic benchmarks with a 3D stencil communication patterns. All benchmarks use the same synthetic load imbalance injection, wherein every 1st and 2nd PEs mod 7 is overloaded, and every 3rd mod 7 is underloaded.

GreedyRefine:

- migrates greedily from overloaded to underloaded processors
- no awareness of communication
- but achieves near-optimal load balance
- keeps migrations minimal

Evaluation in Simulation

Metric	Initial	GreedyRefine	METIS	ParMETIS	Diff-Comm	Diff-Coord
Benchmark 1: 8 PEs						
max/avg load	1.32	1.00	1.00	1.01	1.06	1.05
external/internal comm (MB)	.5	.887	.343	.637	.58	.648
% migrations	-	19.9%	87.1%	14.6%		
Benchmark 2: 32 PEs						
max/avg load	1.37	1.00	1.00	1.04		
external/internal comm (MB)	.143	.422	.158	.194		
% migrations	-	18.7%	98.9%	6.6%		
Benchmark 3: 128 PEs						
max/avg load	1.37	1.00	1.00	1.20		
external/internal comm (MB)	.6	.999	.217	.602	.625	.616
% migrations	-	18.8%	99%	10.8%	18.2%	17.8%

TABLE II: Comparison of load balance metrics across 5 algorithms and synthetic benchmarks with a 3D stencil communication patterns. All benchmarks use the same synthetic load imbalance injection, wherein every 1st and 2nd PEs mod 7 is overloaded, and every 3rd mod 7 is underloaded.

METIS:

- performs full communication graph repartitioning
- high migration count
- optimizes for communication ratio

Evaluation in Simulation

Metric	Initial	GreedyRefine	METIS	ParMETIS	Diff-Comm	Diff-Coord
Benchmark 1: 8 PEs						
max/avg load	1.32	1.00	1.00	1.01	1.06	1.05
external/internal comm (MB)	.5	.887	.343	.637	.58	.648
% migrations	-	19.9%	87.1%	14.6%	18.9%	17%
Benchmark 2: 32 PEs						
max/avg load	1.37	1.00	1.00	1.04	1.02	1.02
external/internal comm (MB)	.143	.422	.158	.194	.173	.227
% migrations	-	18.7%	98.9%	6.6%	15.4%	17.6%
			1.00	1.20	1.09	1.14
			.217	.602	.625	.616
			99%	10.8%	18.2%	17.8%

TABLE II: Compa and synthetic benchmarks with a 3D stencil communication patterns. All bench ction, wherein every 1st and 2nd PEs mod 7 is overloaded, and every 3rd mod

ParMETIS:

- performs more adaptive graph partitioning
- considers initial partitioning
- worse communication ratios than METIS
- but fewer migrations

Evaluation in Simulation

Metric	Initial	GreedyRefine	METIS	ParMETIS	Diff-Comm	Diff-Coord
Benchmark 1: 8 PEs						
max/avg load	1.32	1.00	1.00	1.01	1.06	1.05
external/internal comm (MB)	.5	.887	.343	.637	.58	.648
% migrations	-	19.9%	87.1%	14.6%	18.9%	17%
Benchmark 2: 32 PEs						
max/avg load	1.37	1.00	1.00	1.04	1.02	1.02
		.422	.158	.194	.173	.227
		18.7%	98.9%	6.6%	15.4%	17.6%
		1.00	1.00	1.20	1.09	1.14
		.999	.217	.602	.625	.616
% migrations	-	18.8%	99%	10.8%	18.2%	17.8%

Diff-Comm
- Comparable to ParMETIS
- Requires less tuning

TABLE II: Comparison of load balance metrics across 5 algorithms and synthetic benchmarks with a 3D stencil communication patterns. All benchmarks use the same synthetic load imbalance injection, wherein every 1st and 2nd PEs mod 7 is overloaded, and every 3rd mod 7 is underloaded.

Evaluation in Simulation

Metric	Initial	GreedyRefine	METIS	ParMETIS	Diff-Comm	Diff-Coord
Benchmark 1: 8 PEs						
max/avg load	1.32	1.00	1.00	1.01	1.06	1.05
external/internal comm (MB)	.5	.887	.343	.637	.58	.648
% migrations	-	19.9%	87.1%	14.6%	18.9%	17%
Benchmark 2: 32 PEs						
max/avg load	1.37	1.00	1.00	1.04	1.02	1.02
		.422	.158	.194	.173	.227
		18.7%	98.9%	6.6%	15.4%	17.6%
		1.00	1.00	1.20	1.09	1.14
		.999	.217	.602	.625	.616
		18.8%	99%	10.8%	18.2%	17.8%

Diff-Coord
- slightly worse than Diff-Comm across the board
- reflects 'approximate' nature

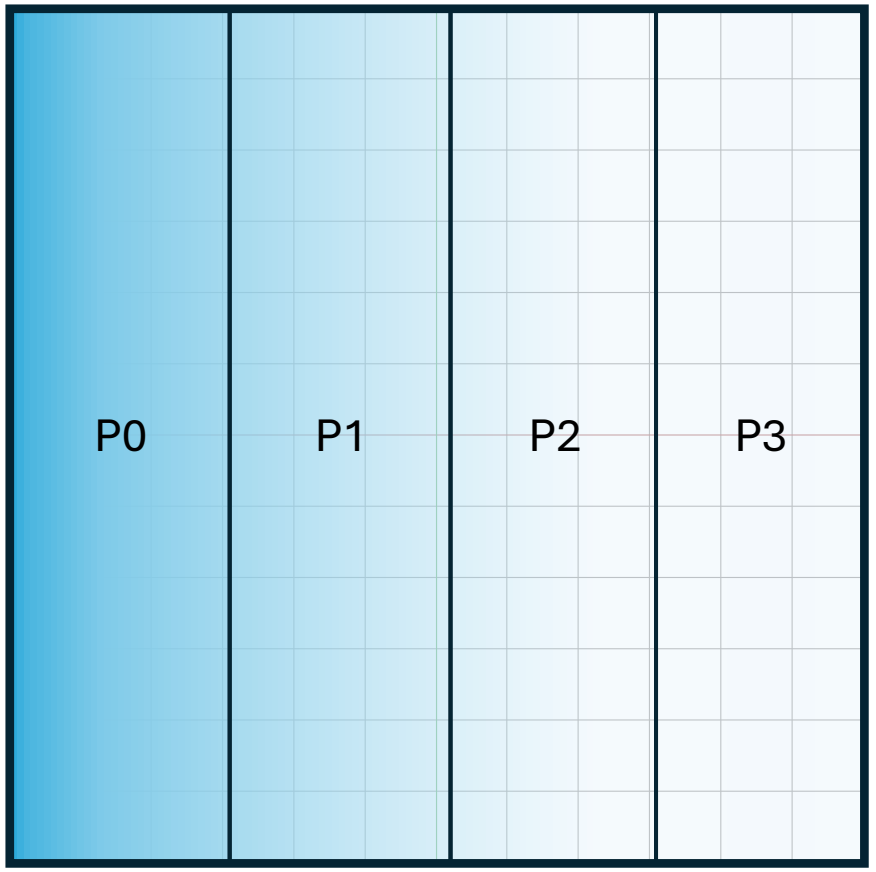
TABLE II: Comparison of load balance metrics across 5 algorithms and synthetic benchmarks with a 3D stencil communication patterns. All benchmarks use the same synthetic load imbalance injection, wherein every 1st and 2nd PEs mod 7 is overloaded, and every 3rd mod 7 is underloaded.

Mini-App Evaluation

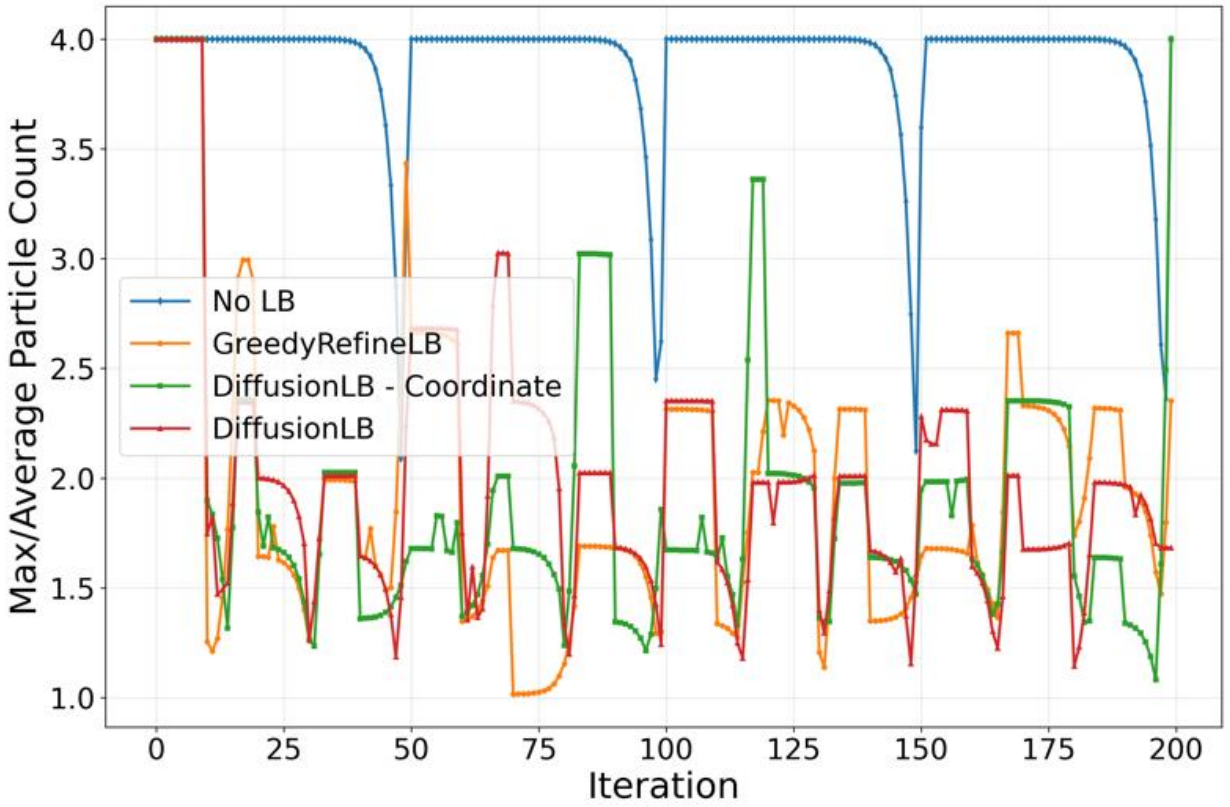
- Charm++ implementation of the Particle-in-Cell-inspired benchmark presented in the Parallel Research Kernels benchmark suite (PIC-PRK)
- 2D particle simulation designed to assess the load-balancing capabilities of parallel runtimes.
- Load imbalance specified by the initial particle distribution
- Load imbalance patterns evolve in a predictable fashion over time, as particles migrate horizontally at constant rate

PIC-PRK Load Imbalance

- Initial distribution:
  - exponential in the x-dimension
  - uniformly random in the y-dimension
- Initial partitioning
  - block columns distributed across 4 processors

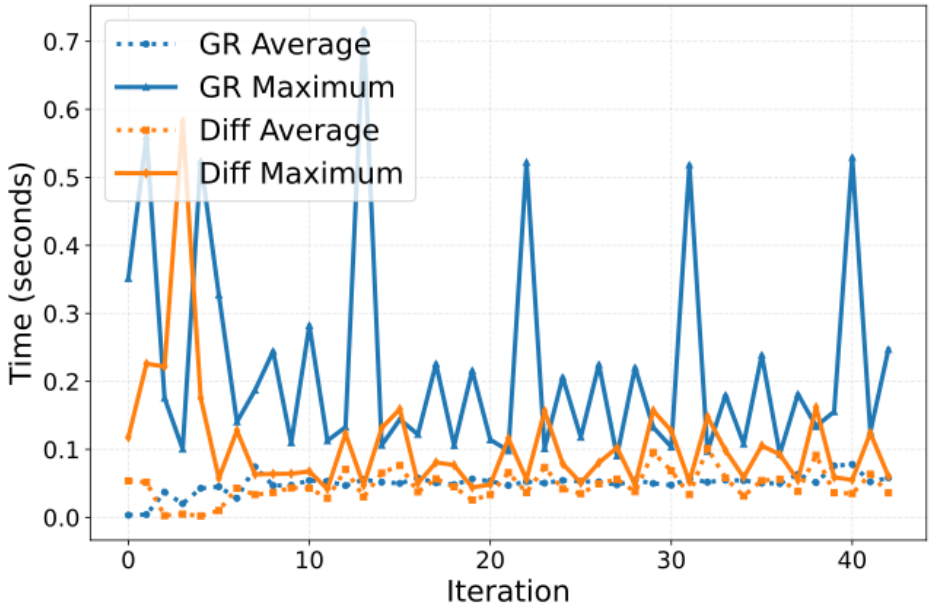


Particle distribution under load balancing



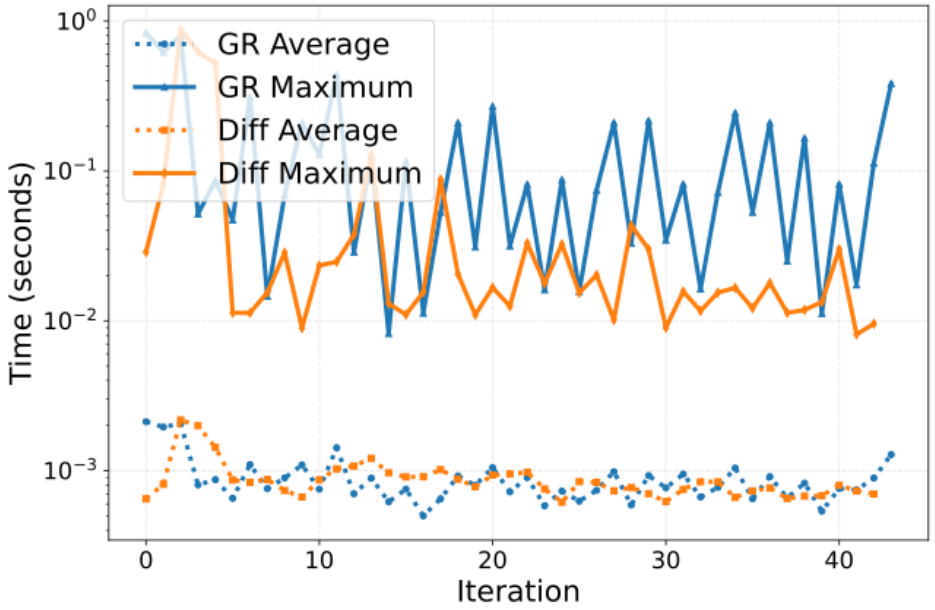
- Load balancing performed every 10 iterations
- Communication-based diffusion: 50% improvement over GreedyRefine
- Coordinate-based Diffusion: 48% improvement over GreedyRefine

PIC-PRC Evaluation



Communication time:

- Diffusion less reactive to particle movement



Computation time:

- Diffusion produces 2.5x speedup on average for max

Limitations

- PIC-PRK is representative of very restricted and synthetic particle movement
- ParMETIS usage not yet supported in Charm++, and a comparison outside of the simulation infrastructure wasn’t possible here
- The Coordinate approximation requires more algorithmic development to avoid fragmentation of processor regions and improve scalability
- Current evaluation is limited to regular 2D-grid regimes

Future Work

- Explore Diffusion performance in full- scale applications beyond the PIC PRK benchmark, eg. the Charm++ ChaNGa cosmological simulation
- Deeper comparison between Diffusion and adaptive ParMETIS, exploring the ParMETIS parameter space and potential Charm++-ParMETIS integration
- Ongoing modularity efforts - exploring effective models to share load balancing strategies across different runtime systems, eg. DARMA-vt

Thank you!

Maya Taylor, Kavitha Chandrasekar, Laxmikant Kale

Contact: mayat4@illinois.edu