

Charm++ Load Balancing Framework

Gengbin Zheng
Parallel Programming Laboratory
UIUC

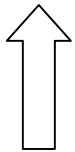
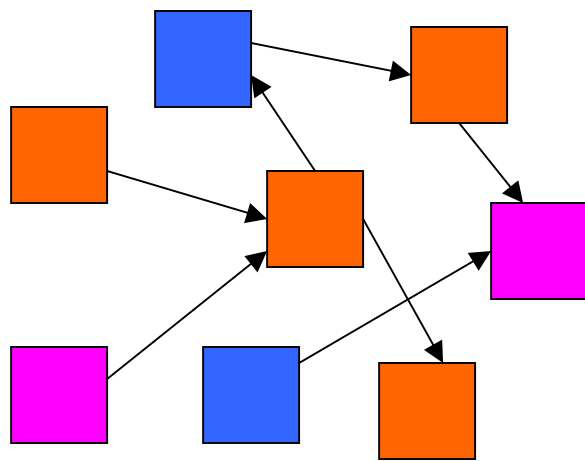
Overview

- Dynamic, adaptive load balancing
 - Migratable objects
- Application independent strategy
 - Object-communication graph
- Pluggable
 - A rich set of load balancing strategies

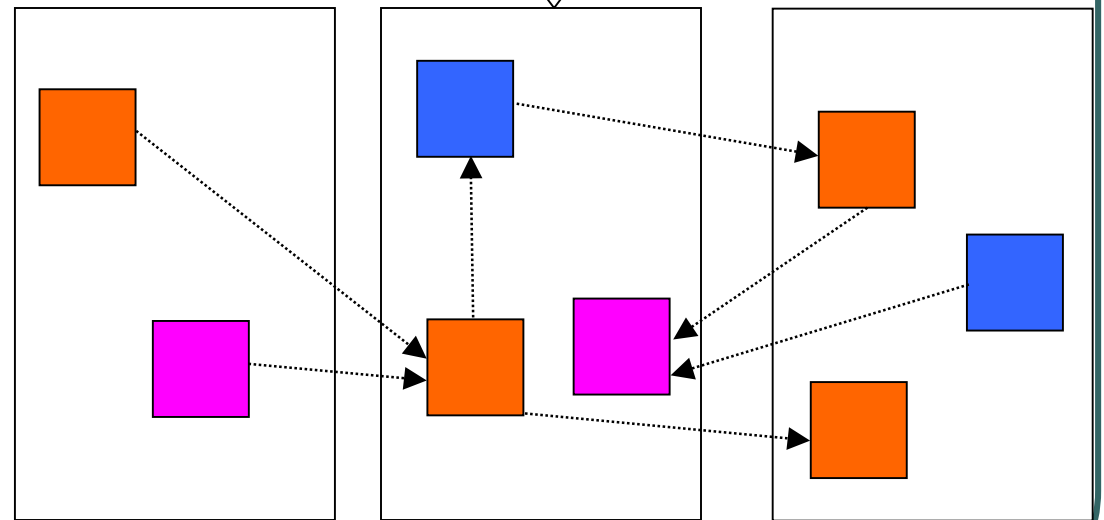
Load Balancing – graph partitioning

Weighted object graph in view of Load Balancer

mapping of objects



LB View



Charm++ PE

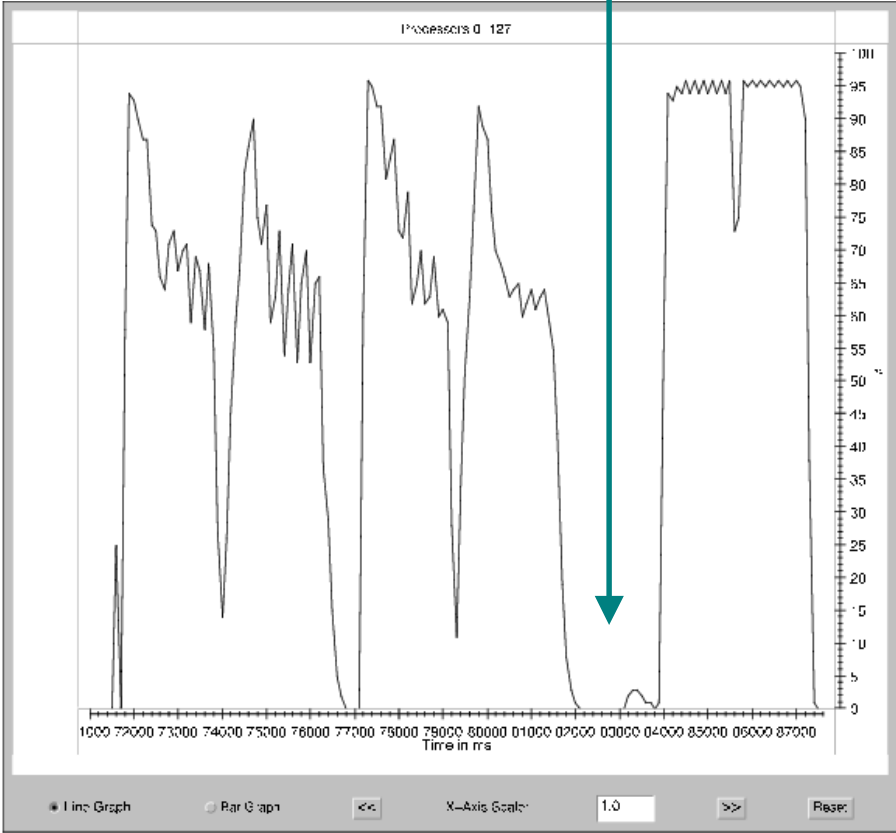
Measurement Based Load Balancing

- Based on Principle of persistence
 - Computation/communication pattern tend to persist
- Components:
 - Load collection module
 - Runtime instrumentation module to measure communication volume and computation time
 - Migration decision-making module
 - Use the load database periodically to make new decisions
 - Centralized vs distributed
 - Greedy improvements vs refinement adjustments
 - Taking communication into account
 - Taking dependences into account (More complex)
 - Topology-aware

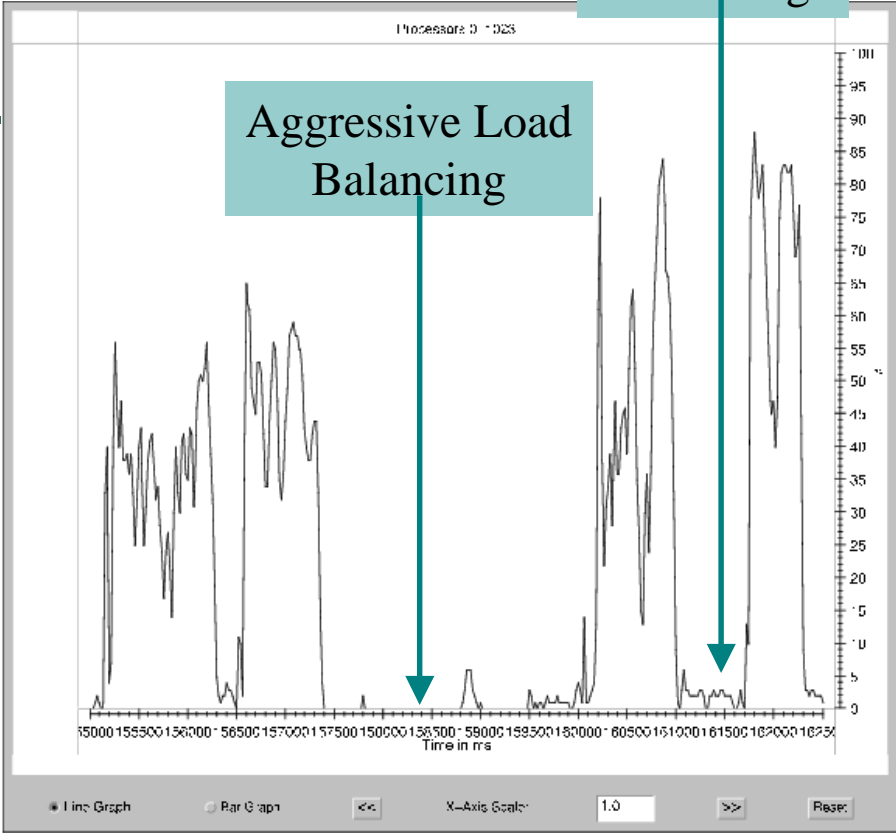
Applications

- NAMD, LeanMD
- CSE applications – crack propagation
 - AMPI thread migration
- Cosmology
- Fault tolerance

Load Balancing



Refinement Load Balancing



Aggressive Load Balancing

Processor Utilization against Time on (a) 128 (b) 1024 processors
On 128 processor, a single load balancing step suffices, but
On 1024 processors, we need a “refinement” step.

Load Balancing For Petascale Machines - Challenges

- Existing load balancing strategies don't scale on extremely large machines
 - Memory footprint; decision making time
 - Consider an application with 1M objects on 64K processors

● Centralized

- Object load data are sent to processor 0
- Integrate to a complete object graph
- Migration decision is broadcast from processor 0
- Global barrier

● Distributed

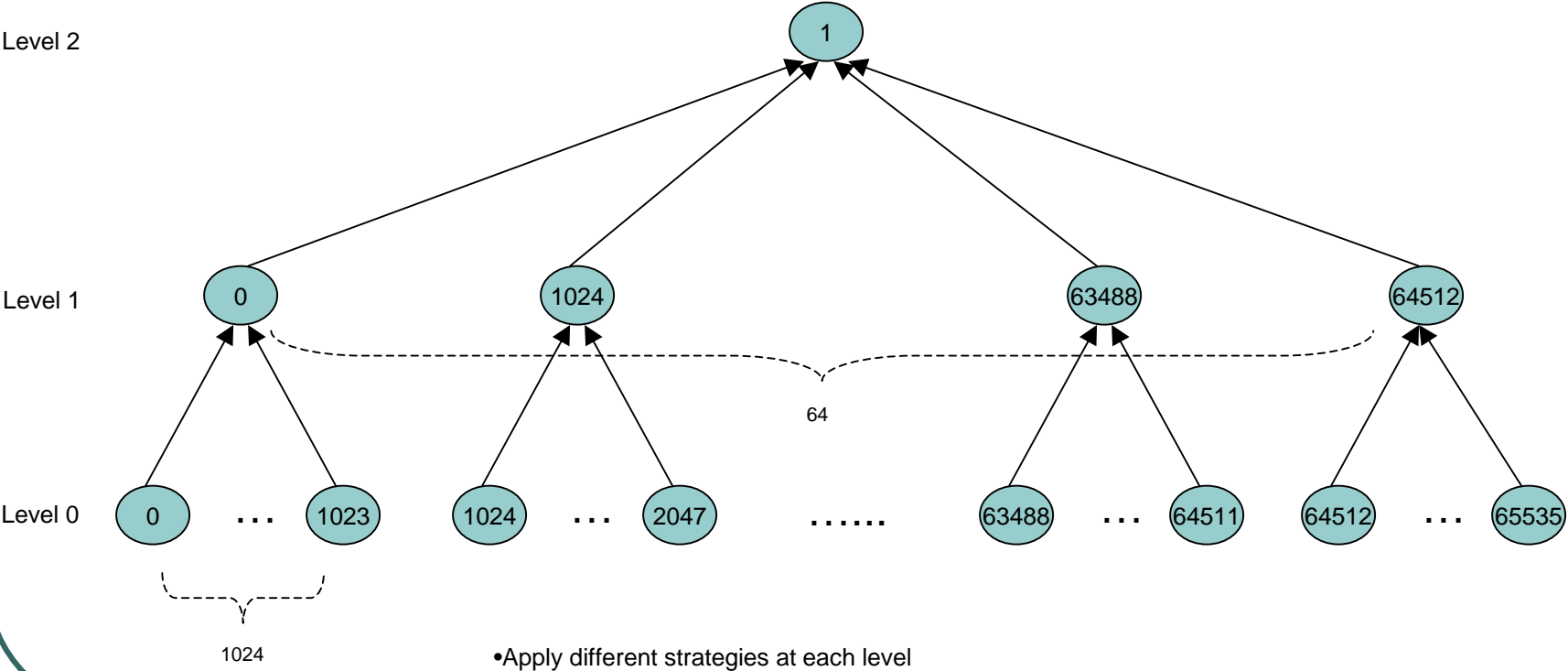
- Load balancing among neighboring processors
- Build partial object graph
- Migration decision is sent to its neighbors
- No global barrier

A Hybrid Load Balancing Strategy

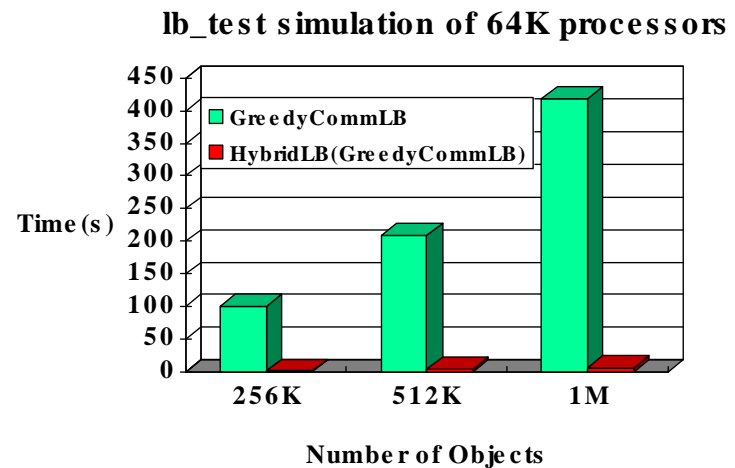
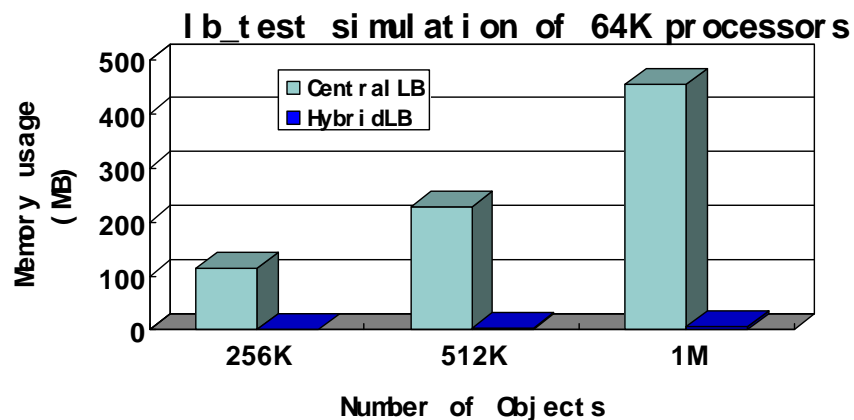
- Dividing processors into independent sets of groups, and groups are organized in hierarchies (decentralized)
- Each group has a leader (the central node) which performs centralized load balancing

Hierarchical Tree (an example)

64K processor hierarchical tree



Simulation Study - Memory Usage



Simulation of lb_test benchmark with the performance simulator

N procs	4096	8192	16384
Memory	6.8MB	22.57MB	22.63MB

lb_test benchmark actual run on BG/L at IBM (512K objects)

Topology-aware mapping of tasks

- Problem
 - Assign K objects to P processors, such that:
 - Compute load on processors is balanced
 - Communicating objects are placed on nearby processors.
- Two Phase Solution
 - Coalesce objects into P computationally-balanced tasks
 - Map the tasks to the correct processors
 - So as to minimize “hops-per-byte” metric

$$\begin{aligned} \text{Hops per Byte} &= \frac{\text{Hop Bytes}}{\sum_{e_{ab} \in E_t} c_{ab}} \\ &= \frac{\sum_{e_{ab} \in E_t} c_{ab} \times d(P(v_a), P(v_b))}{\sum_{e_{ab} \in E_t} c_{ab}} \end{aligned}$$