Dynamic Load Balancing in Charm++

Abhinav S Bhatle
Parallel Programming Lab, UIUC
Outline

• Dynamic Load Balancing framework in Charm++
• Measurement Based Load Balancing
• Examples:
  – Hybrid Load Balancers
  – Topology- aware Load Balancers
• User Control and Flexibility
• Future Work
Dynamic Load-Balancing

• Task of load balancing (LB)
  – Given a collection of migratable objects and a set of processors
  – Find a mapping of objects to processors
    • Almost same amount of computation on each processor
  – Additional constraints
    • Ensure communication between processors is minimum
    • Take topology of the machine into consideration

• Dynamic mapping of charges to processors
  – Load on processors keeps changing during the actual execution
Load-Balancing Approaches

• A rich set of strategies in Charm++
• Two main ideas
  – No correlation between successive iterations
    • Fully dynamic
    • Seed load balancers
  – Load varies slightly over iterations
    • CSE, Molecular Dynamics simulations
    • Measurement-based load balancers
Principle of Persistence

- Object communication patterns and computational loads tend to persist over time
  - In spite of dynamic behavior
    - Abrupt and large, but infrequent changes (e.g. AMR)
    - Slow and small changes (e.g. particle migration)
- Parallel analog of principle of locality
  - Heuristics, that hold for most CSE applications
Measurement Based Load Balancing

• Based on principle of persistence
• Runtime instrumentation (LB Database)
  – communication volume and computation time
• Measurement based load balancers
  – Use the database periodically to make new decisions
  – Many alternative strategies can use the database
    • Centralized vs. distributed
    • Greedy improvements vs. complete reassignment
    • Topology-aware
## Load Balancer Strategies

<table>
<thead>
<tr>
<th>Centralized</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Object load data are sent to processor 0</td>
<td></td>
</tr>
<tr>
<td>- Integrate to a complete object graph</td>
<td></td>
</tr>
<tr>
<td>- Migration decision is broadcasted from processor 0</td>
<td></td>
</tr>
<tr>
<td>- Global barrier</td>
<td></td>
</tr>
<tr>
<td>- Load balancing among neighboring processors</td>
<td></td>
</tr>
<tr>
<td>- Build partial object graph</td>
<td></td>
</tr>
<tr>
<td>- Migration decision is sent to its neighbors</td>
<td></td>
</tr>
<tr>
<td>- No global barrier</td>
<td></td>
</tr>
</tbody>
</table>
Load Balancing on Large Machines

• Existing load balancing strategies don’t scale on extremely large machines
• Limitations of centralized strategies:
  – Central node: memory/communication bottleneck
  – Decision-making algorithms tend to be very slow
• Limitations of distributed strategies:
  – Difficult to achieve well-informed load balancing decisions
Simulation Study - Memory Overhead

Simulation performed with the performance simulator BigSim

Memory usage (MB)

Number of objects

lb_test benchmark is a parameterized program that creates a specified number of communicating objects in 2D-mesh.
Load Balancing Execution Time

Execution time of load balancing algorithms on a 64K processor simulation
Hierarchical Load Balancers

- Hierarchical distributed load balancers
  - Divide into processor groups
  - Apply different strategies at each level
  - Scalable to a large number of processors
Hierarchical Tree (an example)

64K processor hierarchical tree

Apply different strategies at each level
An Example: Hybrid LB

- 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
- A particular hybrid strategy that works well

Our HybridLB Scheme

Refinement-based Load balancing

Greedy-based Load balancing

Load Data (OCG)

token

object
Memory Overhead

Memory usage (MB)

Number of Objects

Simulation of lb_test (for 64k processors)
Total Load Balancing Time

Simulation of lb_test for 64K processors

<table>
<thead>
<tr>
<th>Number of Objects</th>
<th>GreedyCommLB</th>
<th>HybridLB (GreedyCommLB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>22.63MB</td>
<td>22.57MB</td>
</tr>
<tr>
<td>8192</td>
<td>6.8MB</td>
<td>22.57MB</td>
</tr>
<tr>
<td>16384</td>
<td>6.8MB</td>
<td>22.63MB</td>
</tr>
</tbody>
</table>

lb_test benchmark’s actual run on BG/L at IBM (512K objects)
Load Balancing Quality

Simulation of lb_test for 64K processors

Maximum predicted load (seconds)

Number of Objects

- GreedyCommLB
- HybridLB
Topology-aware mapping of tasks

• Problem
  – Map tasks to processors connected in a topology, such that:
    • Compute load on processors is balanced
    • Communicating chares (objects) are placed on nearby processors.
Mapping Model

• Task Graph :
  - $G_t = (V_t, E_t)$
  - Weighted graph, undirected edges
  - Nodes $\leftrightarrow$ shares, $w(v_a) \leftrightarrow$ computation
  - Edges $\leftrightarrow$ communication, $c_{ab} \leftrightarrow$ bytes between $v_a$ and $v_b$

• Topology- graph :
  - $G_p = (V_p, E_p)$
  - Nodes $\leftrightarrow$ processors
  - Edges $\leftrightarrow$ Direct Network Links
  - Ex: 3D- Torus, 2D- Mesh, Hypercube
Model (Contd.)

• Task Mapping
  - Assigns tasks to processors
  - $P : V_t \rightarrow V_p$

• Hop-Bytes
  - Hop-Bytes $\Leftrightarrow$ Communication cost
  - The cost imposed on the network is more if more links are used
  - Weigh inter-processor communication by distance on the network
Load Balancing Framework in Charm++

• Issues of mapping and decomposition separated
• User had full control over mapping
• Many choices
  – Initial static mapping
  – Mapping at run-time as newer objects created
  – Write a new load balancing strategy: inherit from BaseLB
Future Work

• Hybrid Model-based Load Balancers
  – User gives a model to the LB
  – Combine it with measurement based load balancer

• Multicast aware Load Balancers
  – Try and place targets of multicast on the same processor
Conclusions

• Measurement based LBs are good for most cases
• Need scalable LBs in the future due to large machines like BG/L
  - Hybrid Load Balancers
  - Communication sensitive LBs
  - Topology aware LBs