Parallel Branch and Cut
for Dummies

Ted Ralphs
Department of Industrial and Manufacturing Systems Engineering
Lehigh University
Bethlehem, PA
Outline of Talk

- Introduction to Parallel Systems
- Introduction to Tree Search
- Parallelizing Branch and Bound
- Parallelizing Branch and Cut
- The SYMPHONY Library
- Computational Results
- Conclusions
Parallel Systems

- **Parallel System**: Parallel algorithm + parallel architecture [Kumar and Gupta ’94].
- **Scalability**: How well a parallel system takes advantage of increased computing resources.
- **Fixed problem size**: Efficiency decreases with more processors (Amdahl’s Law) [Amdahl ’67].
- **Fixed number of processors**: Efficiency increases with problem size.
- **Isoefficiency**: The rate problem size must be increased to maintain a fixed efficiency [Kumar and Rao ’87].

**Terms**

- Sequential runtime: $T_s$
- Sequential fraction: $s$
- Parallel runtime: $T_p$
- Parallel overhead: $T_o = NT_p - T_s$
- Speedup: $S = T_s/T_p$
- Efficiency: $E = S/N$
Tree Search

- Types of Tree Search Problems
  - Simple search
  - Complete search
  - Optimal search (DOPs)

- Application Areas
  - Discrete Optimization
  - Artificial Intelligence
  - Game Playing
  - Theorem Proving
  - Expert Systems

- Elements of Search Algorithms
  - Node splitting method (branching)
  - Search order
  - Pruning rules
Scalability Factors for Tree Search

• General algorithmic factors
  - Serial fraction
  - Degree of concurrency
  - Parallel overhead
  - Unnecessary/duplicate work
  - Bottlenecks

• Factors specific to search algorithms
  - The ratio of $U_{\text{calc}}$ to $U_{\text{comm}}$
  - Overall size of the search tree
  - Number of branches/children per node
  - Time to “ramp up”
  - Work Distribution Scheme
    – Single work pool
    – Multiple work pools
Parallel Branch and Bound

Finding a feasible solution quickly is important.

- **Diving** strategies
  - Lower memory and communication requirements.
  - Less node set-up time.
  - Find feasible solutions quickly.
  - Wasted computation

- **Best-first** strategies
  - High memory and communication requirements
  - Additional node set-up time
  - Inhibit finding feasible solutions
  - Minimize the size of the tree

- **Hybrid** strategies
  - Controlled Diving (Cost vs. Fractional)
  - Heuristics (initial and primal)
Parallel Branch, Cut, and Price

Global pools are an important additional factor.

- **Node pools**
  - Single pool
    - easy maintenance, load balancing, and termination detection
    - more accurate “global picture”
    - can become a bottleneck
  - Multiple pools
    - increase communication requirements
    - difficult to load balance
    - reduce “saturation”

- **Cut and Column Pools**
  - Must be scanned linearly
  - Single pool
    - slow to scan when large
    - (possibly) better global information
  - Multiple pools
    - smaller and faster to scan
    - contain more localized information
    - more overhead
Performance Measures for Parallel Search

- Measures of redundant/unnecessary work
  - number of nodes in the search tree
  - average time to process a node

- Measures of idle time
  - time spent waiting for work
  - time spent waiting for cuts/columns from pool
SYMPHONY

SYMPHONY is a generic framework for implementing parallel branch, cut, and price.

- It was designed specifically to run in parallel.
  - Can be built for serial, distributed, or shared-memory environments.
  - No knowledge of parallelism needed.
  - Runs on multiple platforms—even mixed.

- User supplies:
  - separation subroutines,
  - the initial LP relaxation,
  - feasibility checker, and
  - other optional subroutines.

- SYMPHONY takes care of all other aspects of algorithm execution, including communication.

- Available at www.BranchAndCut.org.
The Processes of Parallel Branch and Cut

Master
+ store problem data
+ service requests for data
+ compute initial upper bound
+ store best solution
+ handle i/o

Cut Generator
+ generate cuts violated by a particular LP solution
+ process subproblems
+ perform branching
+ check feasibility
+ send cuts to cut pool

LP Solver
+ new cuts

Cut Pool
+ maintain a list of "effective" inequalities
+ return all cuts violated by a particular LP solution

GUI
+ display solutions
+ input user cuts

Tree Manager
+ maintain search tree
+ track upper bound
+ service requests for active node data

parameters
root node
node data
upper bound
cut list
request data
new cuts
LP solution
violated cuts
subtree is finished

Copy cuts
LP solution
feas sol’n
request data
send data
Branching

- Branching decisions are critical, especially near the top of the tree.

- We want to minimize “ramp up” time while not increasing overall execution time – difficult!!

- Can branch on cuts or variables.
  - Any number of children is allowed.
  - Branch on several left hand values for a constraint.

- Strong branching is indispensable.
  - Select several branching candidates (can be cuts, variables, or both).
  - “Presolve” each candidate.
  - Choose the “best” for branching.
  - More candidates are evaluated near the top of the tree.
Pool Management

• Node Storage
  - Candidate nodes stored in a single pool.
  - Try to minimize memory use while not increasing node communication time.
  - The complete state of the tree is stored.
  - Node descriptions are stored explicitly or w.r.t. parent.

• Cut Storage
  - Cuts are stored in a single pool or multiple pools in a compact, efficient form.
  - The cut pools are allocated dynamically and serve assigned subtrees.
  - Pools are purged when they become too large.

• Search Management
  - The search algorithm is a best-first search with controlled diving based on either cost or degree of fractionality.
  - Best-first allows selection of “best” node.
  - Diving avoids node set-up costs and allows feasible solutions to be found quickly.
Summary Results: Vehicle Routing Problem

- Tests were performed on a network of 3 workstations powered by 4 DEC Alpha processors each. The problems are easy- to medium-difficulty problems from VRPLIB and other sources.

- Idle time is the total time spent waiting for work.

- Cut generation was not parallelized and no initial upper bound was provided for these runs.

- Software was SYMPHONY 2.7b and CPLEX 6.5.

<table>
<thead>
<tr>
<th>Number of LP processes</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td># of nodes</td>
<td>6593</td>
<td>6691</td>
<td>6504</td>
<td>6423</td>
</tr>
<tr>
<td>WC time</td>
<td>2493</td>
<td>1281</td>
<td>666</td>
<td>404</td>
</tr>
<tr>
<td>WC/node</td>
<td>0.38</td>
<td>0.38</td>
<td>0.41</td>
<td>0.50</td>
</tr>
<tr>
<td>Idle time</td>
<td>12.68</td>
<td>66.97</td>
<td>208.34</td>
<td>497.45</td>
</tr>
<tr>
<td>Idle/node</td>
<td>0.00</td>
<td>0.01</td>
<td>0.03</td>
<td>0.08</td>
</tr>
</tbody>
</table>
Summary Results: Traveling Salesman Problem

- Tests were performed on a network of 3 workstations powered by 4 DEC Alpha processors each. The problems are easy- to medium-difficulty problems from TSPLIB.

- Idle time is the total time spent waiting for work.

- Cut generation was not parallelized and no initial upper bound was provided for these runs.

- Software was SYMPHONY 2.7b and CPLEX 6.5.

<table>
<thead>
<tr>
<th>Number of LP processes</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td># of nodes</td>
<td>3405</td>
<td>3041</td>
<td>2917</td>
<td>2901</td>
</tr>
<tr>
<td>WC time</td>
<td>18119</td>
<td>10057</td>
<td>7445</td>
<td>5199</td>
</tr>
<tr>
<td>WC/node</td>
<td>5.32</td>
<td>6.60</td>
<td>10.21</td>
<td>14.34</td>
</tr>
<tr>
<td>Idle time</td>
<td>24.34</td>
<td>935.42</td>
<td>3527.61</td>
<td>10767.89</td>
</tr>
<tr>
<td>Idle/node</td>
<td>0.01</td>
<td>0.31</td>
<td>1.21</td>
<td>3.71</td>
</tr>
</tbody>
</table>
Another Experiment

Comparison of full diving to controlled diving.

<table>
<thead>
<tr>
<th></th>
<th>Number of LP processes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>nodes</td>
<td>5197</td>
</tr>
<tr>
<td>WC</td>
<td>23695</td>
</tr>
<tr>
<td>WC/node</td>
<td>4.56</td>
</tr>
<tr>
<td></td>
<td>3405</td>
</tr>
<tr>
<td>nodes</td>
<td>18119</td>
</tr>
<tr>
<td>WC</td>
<td>5.32</td>
</tr>
<tr>
<td>WC/node</td>
<td></td>
</tr>
</tbody>
</table>
Further Experiments

• Diving strategy

<table>
<thead>
<tr>
<th></th>
<th>VRP</th>
<th>TSP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Time</td>
</tr>
<tr>
<td>w/ UB</td>
<td>3524</td>
<td>241</td>
</tr>
<tr>
<td>w/o UB</td>
<td>6504</td>
<td>666</td>
</tr>
</tbody>
</table>

• Parallel Overhead

<table>
<thead>
<tr>
<th></th>
<th>Nodes</th>
<th>Time</th>
<th>Time/Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel</td>
<td>7014</td>
<td>24558</td>
<td>3.50</td>
</tr>
<tr>
<td>Sequential</td>
<td>7014</td>
<td>25446</td>
<td>3.63</td>
</tr>
</tbody>
</table>

• Shared vs. Distributed Memory

<table>
<thead>
<tr>
<th></th>
<th>Nodes</th>
<th>Time</th>
<th>Time/Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared</td>
<td>8896</td>
<td>2975</td>
<td>1.34</td>
</tr>
<tr>
<td>Distributed</td>
<td>9701</td>
<td>3392</td>
<td>1.40</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- Architecture does not play a very big role.
- Ramp-up time is not a problem.
- Unnecessary/duplicate work is not an issue.
- The cut pools are not significant bottlenecks.
- The single biggest factor in loss of speedup is the bottleneck created by the tree manager.
- Multi-pool approaches are well-studied, but tough to implement (see [Eckstein ’94], [Gendron and Crainic ’94]).
- A single tree manager with several slave node pools might be a good compromise (ala [Eckstein ’94]).
- A more decentralized approach may also be undertaken.