# Implementing Scalable Parallel Search Algorithms for Data-Intensive Applications

Ted Ralphs
Industrial and Systems Engineering
Lehigh University
http://www.lehigh.edu/~tkr2

Laszlo Ladanyi
IBM T.J. Watson Research Center

Matt Saltzman
Clemson University

International Conference on Computational Science 2002, Tuesday April 23, 2002

# Outline of Talk

- Overview of parallel tree search
  - Knowledge sharing
  - Data-intensive applications
- Overview of branch, cut, and price algorithms
- The Abstract Library for Parallel Search (ALPS)
  - Scalability
  - Data handling
- Computational results
- Conclusions

# Parallel Systems and Scalability

- Parallel System: Parallel algorithm + parallel architecture.
- Scalability: How well a parallel system takes advantage of increased computing resources.
- Terms

| | |
|---|---|
| Sequential runtime | $T_s$ |
| Parallel runtime | $T_p$ |
| Parallel overhead | $T_o = NT_p - T_s$ |
| Speedup | $S = T_s/T_p$ |
| Efficiency | $E = S/N$ |

# Tree Search Algorithms

- 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 rule
  - Bounding method (optimization only)

# Constructing Parallel Tree Search Algorithms

- Main contributors to parallel overhead
  - Communication Overhead
  - Idle Time (Ramp Up/Down Time)
  - Idle Time (Handshaking)
  - Performance of Non-critical Work
- Non-critical work is essentially work that would not have been performed in the sequential algorithm.
- The primary way in which tree search algorithms differ is the way in which *knowledge* is shared.
- Sharing knowledge helps eliminate the performance of non-critical work.
- If all processes have "perfect knowledge," then no non-critical work will be performed.
- However, knowledge sharing may also increase communication overhead and idle time.

# Knowledge Bases

- Knowledge is shared through *knowledge bases*.
- Methods for disseminating knowledge
  - Pull: Process requests information from the knowledge base (asynchronously or synchronously).
  - Push: Knowledge base broadcasts knowledge to processes.
  - An important parameter to consider is whether the current task is interrupted when knowledge is received or not.
- Basic examples of knowledge to be shared.
  - Bounds
  - Node Descriptions
- We will see more later.

# Data-intensive Tree Search

- In some applications, the amount of information needed to describe each search tree node is very large.
- This can make memory an issue and also increase communication overhead.
- Abstractly, we will think of each node as being described by a list of *objects*.
- For the applications we are interested in, the list of objects does not change much from parent to child.
- We can therefore store the description of an entire subtree very compactly using *differencing*.

# Example: Combinatorial Optimization

- A *combinatorial optimization problem* $CP = (E, \mathcal{F})$ consists of
  - A *ground set* E,
  - A set $\mathcal{F} \subseteq 2^E$ of *feasible solutions*, and
  - A *cost function* $c \in \mathbf{Z}^{\mathbf{E}}$ (optional).
- The *cost* of $S \in \mathcal{F}$ is $c(S) = \sum_{e \in S} c_e$.
- Problem: Find a least cost member of $\mathcal{F}$.
- Example: The Integer Knapsack Problem
  - We have a set of $n$ items with weights $w_i$ and profits $p_i$.
  - We want to choose the most profitable subset of the $n$ items such that the total weight is less than $W$.
  - We can branch by choosing an item and either forcing it in or out of the knapsack.

# Branch and Bound

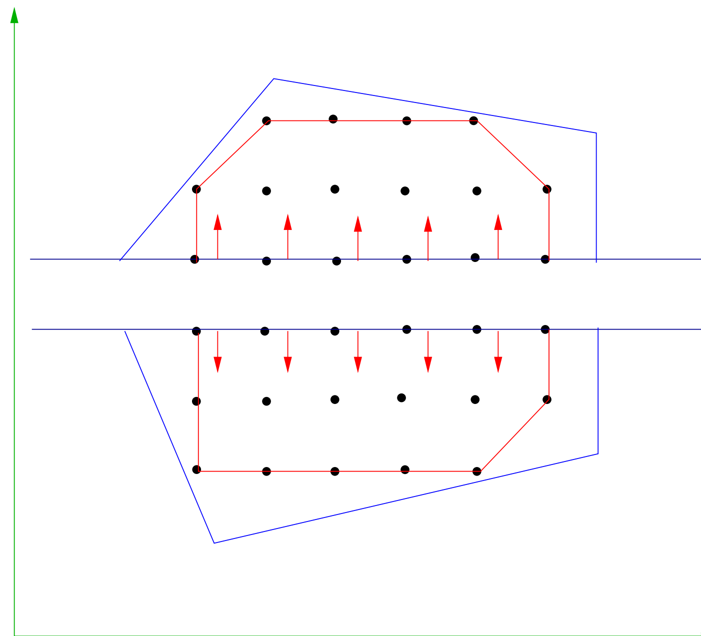
- *Branch and bound* is the most commonly-used algorithm for solving combinatorial optimization problems.
- It is a divide and conquer approach.
- Suppose $F$ is the set of feasible solutions and we wish to solve $\min_{s \in F} c^T x$.
- Consider a partition of $F$ into subsets $F_1, \ldots F_k$. Then

$$\min_{x \in F} c^T x = \min_{\{1 \leq i \leq k\}} \{\min_{x \in F_i} c^T x\}$$

- The branch and bound method is to solve each of the new subproblem recursively.
- If we cannot solve a given subproblem, then we try to either derive a bound that will allow us to *prune* it or prove it is *infeasible*.
- Failing that, we further subdivide the problem (*branch*).

# Branch, Cut and Price Algorithms

- Feasible solutions can be viewed as incidence vectors in $\mathbf{R}^n$.
- The convex hull of feasible incidence vectors is a polyhedron over which we can optimize.
- The description of this polyhedron consists of a list of linear inequalities and is not typically known at the outset.
- The BCP algorithm uses approximations of this polyhedron to obtain bounds and branches by dividing the polyhedron into smaller pieces.



# Knowledge Sharing in BCP

- In BCP, knowledge discovery consists of finding the linear inequalities that describe or approximate the polyhedron.
- The more inequalities we know, the better bounds we can obtain.
- Finding these inequalities can be time consuming, so we want to share them when they are found.
- Hence we have a new kind of knowledge that must be shared.
- Knowledge bases in BCP
  - Node Pools
  - Inequality Pools

# The ALPS Project

- In partnership with IBM and the COIN-OR project.
- Multi-layered C++ class library for implementing scalable, parallel tree search algorithms.
- Design is fully generic and portable.
  - Support for implementing general tree search algorithms.
  - Support for any bounding scheme.
  - No assumptions on problem/algorithm type.
  - No dependence on architecture/operating system.
  - No dependence on third-party software (communications, solvers).
- Emphasis on parallel scalability.
- Support for large-scale, data-intensive applications (such as BCP).
- Support for advanced methods not available in commercial codes.

# ALPS Design

Modular library design with minimal assumptions in each layer.

**ALPS** Abstract Library for Parallel Search

- manages the search tree.
- prioritizes based on quality.

**BiCePS** Branch, Constrain, and Price Software

- manages the data.
- adds notion of objects and differencing.

**BLIS** BiCePS Linear Integer Solver

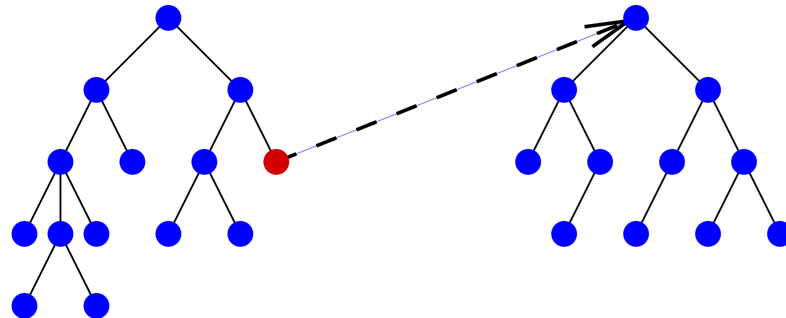
- Implementation of BCP algorithm.
- Objects are the inequalities.

# Scalability Issues for Parallel Search

- Unit of work (granularity)
- Number and location of knowledge bases
- Synchronous vs. asynchronous messaging
- Ramp-up/ramp-down time

# Scalability: Granularity

Work unit is a subtree.



Advantages:

- less communication.
- more compact storage via differencing.

Disadvantage:

- more possibility of non-critical work being done.

# Scalability: Master - Hubs - Workers Paradigm

## Master

- has global information (node quality and distribution).
- balances load between hubs.
- balances quantity *and* quality.

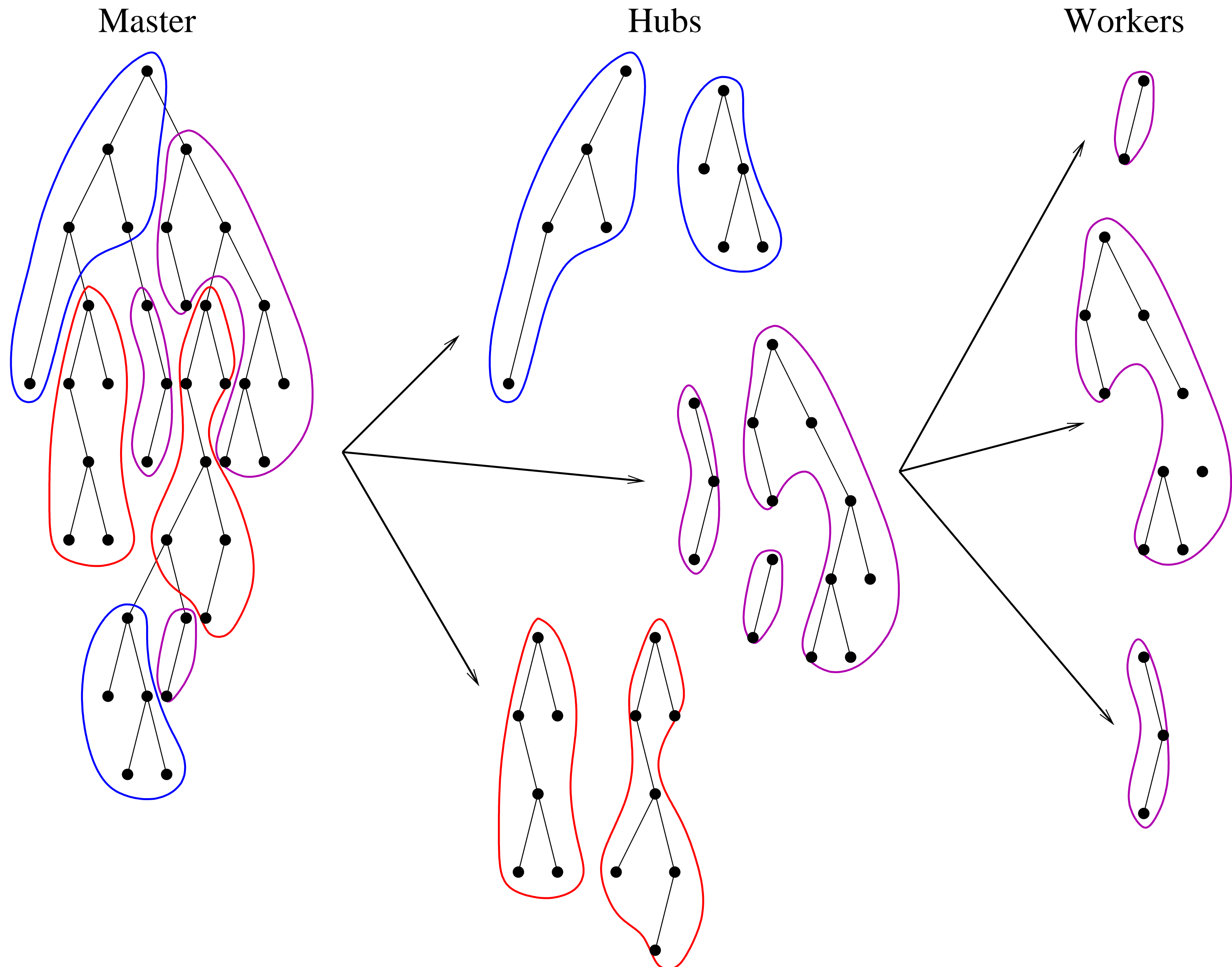
## Hubs

- manage collections of subtrees (may not have full descriptions)
- balances load between workers

## Workers

- processes one subtree.
- hub can interrupt.
- sends branch and quality information to hub.

# Scalability: Master - Hubs - Workers Paradigm



# Scalability: Asynchronous Messaging

Possible communication bottlenecks:

- Too many messages.
  - avoided by the increased task granularity.
  - master-hub-worker paradigm also contributes.
- Too much synchronization (handshaking)
  - almost no handshaking.
  - must take place when a worker finishes exploring a subtree.

# Scalability: Ramp-up/Ramp-down

- Ramp-up time: Time until all processors have useful work to do.
- Ramp-down time: Time during which there is not enough work for all processors.
- Ramp-up time is perhaps the most important scalability issue for branch and bound when the bounding is computationally intensive.
- Controlling Ramp-up/ramp-down
  - Branch more quickly.
  - Use different branching rules (branch more quickly).
  - Hub instructs workers when to change rules.

# Data Handling

- Focused on data-intensive applications.
- Need to deal with HUGE numbers of objects.
- Duplication is a big issue.
- Goal is to avoid such duplication in generation and storage.
- Implementation:

1. Object arrives in encoded form with hash value.
2. Object is looked up in hash map.
3. If it does not exist, then it is inserted.
4. A pointer to the unique copy in the hash map is added to the list.

# Object Pools

- Share objects across nodes in the tree.
- Object pools allow generated objects to be shared.
  - Cut pool is one example.
- Quality measures ensure only the most "important" objects are retained and utilized.
- Object encoding is used to ensure that objects are stored only once.

# Preliminary Computational Results

- Performance measures
  - Speedup
  - Number of nodes in the search tree
  - Amount of idle time
  - Total parallel overhead
  - CPU vs. Wallclock Time
- These results are with a previous generation of the software called SYMPHONY employing many of the same ideas, but with a single node pool.
- The application is the Vehicle Routing Problem.
- The platform is a Beowulf cluster running Linux with 48 dual-processor nodes.
- IBM's Optimization Subroutine Library (OSL) was used for some of the numerical analysis.

# Preliminary Computational Results

| Instance | Tree Size | Ramp Up | Ramp Down | Idle (Nodes) | Idle (Cuts) | CPU sec | Wallclock |
|---|---|---|---|---|---|---|---|
| A − n37 − k6 | 9684 | 0.28 | 1.41 | 8.83 | 29.21 | 947.41 | 248.66 |
| A − n39 − k5 | 1128 | 0.38 | 0.13 | 1.03 | 5.70 | 158.84 | 41.97 |
| A − n39 − k6 | 445 | 0.47 | 0.03 | 0.30 | 0.64 | 27.85 | 7.45 |
| A − n44 − k6 | 1247 | 0.30 | 0.16 | 1.28 | 4.47 | 336.66 | 86.38 |
| A − n45 − k6 | 1202 | 0.30 | 0.18 | 0.94 | 4.70 | 300.00 | 77.40 |
| A − n46 − k7 | 33 | 0.42 | 1.39 | 1.09 | 0.22 | 9.82 | 2.97 |
| A − n48 − k7 | 4931 | 0.68 | 0.83 | 5.12 | 17.65 | 1097.77 | 282.85 |
| A − n53 − k7 | 1521 | 0.64 | 0.18 | 1.66 | 9.50 | 542.19 | 138.98 |
| A − n55 − k9 | 9802 | 0.69 | 1.97 | 11.76 | 25.23 | 1990.86 | 508.84 |
| A − n65 − k9 | 14593 | 0.89 | 4.56 | 21.85 | 104.18 | 8407.69 | 2138.54 |
| B − n45 − k6 | 4243 | 0.45 | 0.57 | 4.03 | 9.85 | 531.04 | 138.15 |
| B − n51 − k7 | 603 | 0.35 | 0.03 | 0.39 | 0.60 | 82.51 | 22.03 |
| B − n57 − k7 | 2964 | 0.41 | 0.62 | 2.25 | 5.38 | 462.13 | 123.09 |
| B − n64 − k9 | 86 | 0.64 | 0.50 | 0.50 | 0.23 | 16.24 | 4.49 |
| B − n67 − k10 | 16924 | 0.60 | 4.28 | 17.69 | 96.48 | 3135.88 | 814.65 |
| Total | 69406 | 7.52 | 16.85 | 78.73 | 314.03 | 18046.89 | 4636.45 |
| Total | 64520 | 27.10 | 23.01 | 78.94 | 363.38 | 15128.37 | 1963.59 |
| Total | 85465 | 93.92 | 67.02 | 106.16 | 737.39 | 19836.47 | 1316.03 |
| Total | 75011 | 373.49 | 100.35 | 111.71 | 1600.49 | 17881.11 | 633.12 |

# Preliminary Conclusions

- We can already achieve close to linear speedup or better with up to 32 processors, even with a single node pool.
- Performance of non-critical work is not a problem – the number of nodes examined stays and the total CPU time is approximately constant.
- For this small number of processors, the node pool central node pool is not a bottleneck.
- Object pools and Ramp-up time are the primary scalability issue for these data-intensive algorithms.
  - For BCP, the inequality pools are the biggest bottleneck.
  - We can try to control this by scaling the number of pools.
  - Ramp up time is more difficult to control.

# What's Currently Available

- SYMPHONY: C library for implementing BCP
  - User fills in stub functions.
  - Supports shared or distributed memory.
  - Documentation and source code available `www.BranchAndCut.org`.
- COIN/BCP: C++ library for implementing BCP
  - User derives classes from library.
  - Documentation and source code available `www.coin-or.org`.
- ALPS/BiCePS/BLIS
  - In development and not yet freely available.
  - Will be distributed from CVS at `www.coin-or.org`.
- The COIN-OR repository `www.coin-or.org`

# Applications

- SYMPHONY has been used for various combinatorial problems:
  - Traveling Salesman Problem
  - Vehicle Routing Problem
  - Capacitated Network Routing
  - Airline Crew Scheduling
- SYMPHONY is also being adapted for constraint programming.
- COIN/BCP has also been used for combinatorial problems
  - Minimum Weight Steiner Tree
  - Max Cut
  - Multi-knapsack with Color Constraints
- We have a number of new applications under development.

# The COIN-OR Project

- Supports the development of interoperable, open source software for operations research.
- Maintains a CVS repository for open source projects.
- Supports peer review of open source software as a supplement to the open literature.
- Software and documentation is freely downloadable from `www.coin-or.org`