Parallelization of Discrete Optimization Problems

Advanced Topics in Parallel Processing

Jan Lemeire
Dept. ETRO
November-December 2013

Parallel Systems Course: Chapter VIII

Kumar Chapter 11
DOP Example: Shift- Puzzle
Definition DOP

Set of feasible solutions $S$
- Find any feasible solution

Optional: cost function $f$ for each solution
- Find optimal feasible solution (min or max)

In terms of complexity of solution methods, there are two classes:
- Problems that have efficient algorithms for finding optimal solutions.
  ex Dijkstra
- Problems that don’t have such efficient algorithms (NP-complete)
  ex Traveling Salesman Problem

Algorithms
- Exhaustive search: computational intensive due to large set size
- Heuristic search
Tree representation

Shift Puzzle

Initial

UP

LEFT

DOWN

UP

LEFT

DOWN

UP

LEFT

RIGHT

DOWN

LEFT

RIGHT

...
Sometimes a feasible solution is OK, in other cases the optimal solution should be found.
Sequential Tree Search Algorithms

- **Depth – First search**
  1. Simple Backtracking
  2. Branch and Bound: limit the depth

- **Breadth – First search**
  1. Iterative Deepening: with open node list

- **Heuristic search**
  - Best first search:
    - based on breadth-first
    - With heuristic function
Parallel Depth-first Tree Search

Distribution of tree:

After expansion of root node: send children & problem to slaves
DOP looking for a feasible solution
Parallel Work Anomalies

Sequential work ≠ Parallel work!

\[ T_{\text{seq}} \neq \sum_{i=1}^{p} T_{\text{work}}^i \]

\[ T_{\text{seq}} + T_{\text{anomaly}} = \sum_{i}^{p} T_{\text{work}}^i \]

Search Overhead Factor = \( \frac{\text{parallel work}}{\text{sequential work}} = \frac{\sum T_{\text{work}}^i}{T_{\text{seq}}} \)

In our approach: considered as overhead (can be positive or negative)

Impact on overhead: \( \frac{T_{\text{anomaly}}}{T_{\text{seq}}} \)
Parallel Overhead

- Partitioning: low
- Communication: low
- Synchronization: no
- Returning results: low
- **Idling: HIGH**
  - Due to load imbalances
  
  **Solution:** dynamic load balancing
  "when finished, ask for work"

- Find solutions for:
  1. Donor Selection
  2. Termination Detection
Load imbalances

Figure 11.7  The unstructured nature of tree search and the imbalance resulting from static partitioning.
Donor selection

- **Asynchronous Round Robin**
  Each processor keeps a cyclic list

- **Global Round Robin**
  Master keeps a cyclic list

- **Random polling**
  Random selection of donor
Termination detection

- Via master (centralized)
  - For example: if donor selection happens via master

- Dijkstra’s token algorithm (local)
  - Without DLB: a simple token
  - With DLB: one boolean as token and one boolean per processor

- Tree-Based (local)

- other ...
Tree-based Termination Detection

**Idea:** Associates weights with individual work pieces

- Master starts with weight 1.
- It sends work to \((p-1)\) slaves together with weight \(1/p\) for each. It keeps weight \(1/p\).
- If a slave finishes: it sends its weight to master. Master adds it to its weight.
- If a slave sends a part of its work, it sends halve of its weight.
- Termination when weight at master becomes one and master has finished
Performance Study of DLB

Overhead \( \sim \) number of requests \( n \)

- With \( \beta \): average load imbalance
- \( \beta = (avg - min)/avg \)
- \( \epsilon \): minimal work size after which redistribution of remaining work is not done anymore

Remaining work after each step:

\[
W
\beta.W
\beta. \beta.W
\ldots
\beta^n.W = \epsilon \quad \Rightarrow \quad \beta^n = \epsilon/W
\]

\[
=> \quad n = \log_{\beta} \epsilon/W
\]
Parallel Best-first Search

Keep stack of open nodes, ordered by a heuristic function

1. **Centralised strategy**: keep stack on master
   ⇒ send part of nodes to slaves
   ⇒ Slaves return expanded nodes
   HIGH COMMUNICATION

2. **Local strategy**: stack on each processor
   Synchronization of open node list necessary:
   - Random communication strategy
   - Ring communication strategy
   - Blackboard communication strategy
Figure 11.15  A message-passing implementation of parallel best-first search using the ring communication strategy.
Figure 11.16  An implementation of parallel best-first search using the blackboard communication strategy.
Graph Representation

1. Unfold graph to a tree
   ⇒ Disadvantage: nodes will be checked multiple times if paths can lead to equivalent states

2. Keep a *closed node list*
Check an expanded node
- If parallel: synchronization of list
- retrieve node with a hash function