Parallel Consistency in Constraint Programming

Carl Christian Rolf & Krzysztof Kuchcinski

Lund University
ccr@cs.lth.se
Outline

- Introduction to Constraint Programming (CP)
- Parallelism in CP
- Our Model of Parallel Consistency
- Experimental Results
- Conclusions
- Future Work
Introduction to CP

- Similar to Integer Programming, but more natural modeling
- Constraint programming is declarative, useful for automatic parallelism
- Can be used to formulate problems such as Sudoku, Jobshop scheduling, and aircrew scheduling
- Solving is NP-complete
Solving a CP-Problem

- Constraint problem solving = Search + Consistency
- Search is usually depth-first
- Consistency prunes values that cannot lead to a solution (pruning not complete, hence search)
- Solving is search tree exploration with very heavy nodes
Example of CP-Solving

One branch evaluated at a time
Consistency enforced on every level of the search tree
Questions?

Or are the basics of CP clear to everyone?
Parallelism in CP

- Data parallelism: Split the search tree
- Task parallelism: Split the work in the search nodes
Problems with Data Parallelism

- Problems can’t always be split efficiently - eventually the work is too small
- Communication costs
- Does not suit all problems, e.g., scheduling need customized splitting method
- Consistency often magnitudes more time-consuming than search
Solution

- Combine data and task parallelism
  - When splitting is inefficient, use task parallelism
  - When tasks are too small, split tree instead
- First we need task parallelism, hence this work
Our Model of Parallel Consistency

- The solver has several consistency threads (running on processors P1, P2, and P3 in the example)

- Each iteration of consistency takes data from the store held by the solver
Variants

- Shared updates: the changes to variables are visible to the other constraints before the barrier

- Thread local updates: the changes are only visible after the barrier

- Thread local updates needs no extra synchronization, but slower to detect inconsistency
Shared Updates

- Changes to variables are visible to other threads between constraints
- Updates are written to the store after the barrier
Experimental Results

- n-Sudoku, $n = 1024$
- LA31, 30 by 10 jobshop
- n-Queens, $n = 40,000$
- JaCoP solver, written in Java 5
- Mac Pro with 8 cores
- Speed-up before search
Consistent Store

![Consistent Store Diagram](image-url)

- **Legend:**
  - Light gray: Sudoku
  - Black: LA31
  - Gray: Queens

- **X-axis:** Number of Threads
  - 1, 2, 4, 8

- **Y-axis:** Absolute Speed-up
  - Values range from 0 to 7

- Observations:
  - With 1 thread, LA31 has a higher speed-up compared to Sudoku and Queens.
  - As the number of threads increases to 4 and 8, Queens shows a significant speed-up, surpassing both Sudoku and LA31.
  - The speed-up for Sudoku remains relatively constant across different thread counts.
Observations

- Sudoku is a perfect problem, performs no pruning
- LA31 - global constraints are too small
- Queens - three alldiff constraints dominates execution
Inconsistent Store
Observations

- Many more iterations of consistency, also for Sudoku
- Speed-up drops compared to consistent store
**Processor Load**

- Sudoku perfect, LA31 twelve iterations of consistency, Queens two iterations
Conclusions

- Some problems do not scale well, they need parallel consistency algorithms
- Very hard to retain speed-up during search (due to locking and wait/notify)
- Small difference between thread local updates and shared updates
- Is probably best as an extension to data parallelism
Future Work

- Combine data and task parallelism
- Load balancing in task parallelism
- Ideally: share updates during execution of consistency algorithms
- Long-term future of parallelism in CP: data parallelism + task parallelism + parallel consistency algorithms
Thank You

Questions?