Special Meeting 2

Multi-GPU Parallel Constraint Programming

Hakan Hasan

20 June 2025

Constraint programming

- An efficient technique for solving a variety of combinatorial problems (typically in the NP-complete and NP-hard classes).
- In CP a problem is defined over variables that take values in domains and constraints which restricts the allowed combination of values.

A constraint network $\mathscr{CN} = (\mathscr{X}, \mathscr{D}, \mathscr{C})$ is defined by

- a set of n *variables* $\mathscr{X} = \{x_1, x_2, \ldots, x_n\}$,
- a set of n finite *domains* $\mathscr{D} = \{D(x_1), D(x_2), \ldots, D(x_n)\}$ with $D(x_i)$ the set of possible *values* for the variable x_i,
- a set of *constraints* between the variables $\mathscr{C} = \{C_1, C_2, \ldots, C_e\}$. A constraint C_i is defined on a subset of variables $X_{C_i} = \{x_{i_1}, x_{i_2}, \ldots, x_{i_j}\}$ of $\mathscr{X}$ with a subset of the Cartesian product $D(x_{i1}) \times D(x_{i2}) \times \ldots \times D(x_{ij})$ that states which combinations of values of variables $\{x_{i_1}, x_{i_2}, \ldots, x_{i_j}\}$ are compatible.

Constraint programming

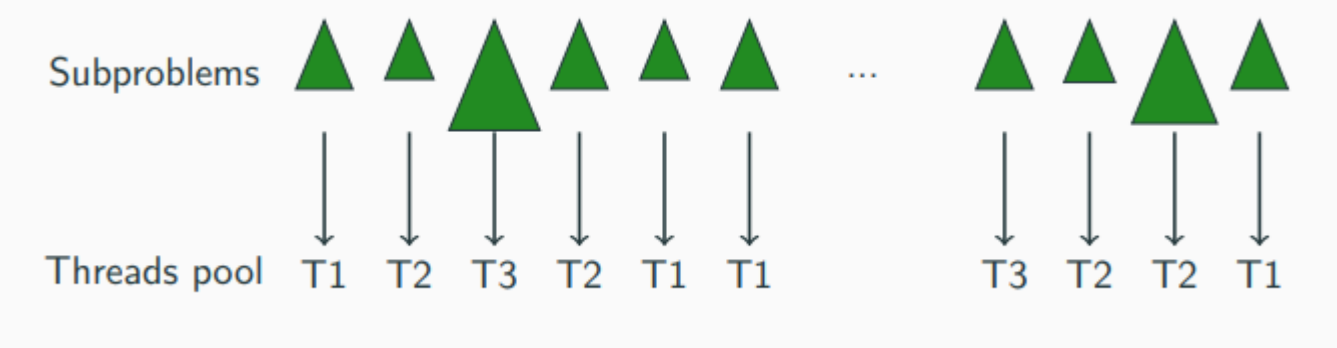
- CP uses for each constraint an algorithm that removes values of variables that are inconsistent with the constraint. These algorithms are called while a domain is modified.
- Then, a search algorithm such as a backtracking or branch-and-bound algorithm is called to find solutions.

The problem

- Current solvers can handle instances with hundreds of thousands variables and millions of constrains.
- But we want to solve even larger constraint problems!
- The natural next step in the development of ever more powerful constraint optimization methods is the use of parallelism.

Embarrassingly parallel search

- Method for solving CP problems in parallel based on embarrassingly parallel computations.
- When having k workers EPS proposes to split the problem into a huge number of sub-problems (e.g., $30k$) and give the subproblems successively and dynamically to the workers.
- EPS expects that for each worker the sum of the resolution time of its subproblems will be equivalent ⇒ load balancing is automatically obtained in a statistical sense.



Key concepts

- Massive static decomposition
- Loose communication
- Non-intrusive implementation

Problem decomposition

- The decomposition challenge is to find the depth at which the search frontier contains approximately p^* nodes.
- EPS statically decomposes the initial problem into a huge number of subproblems that are consistent with propagation.

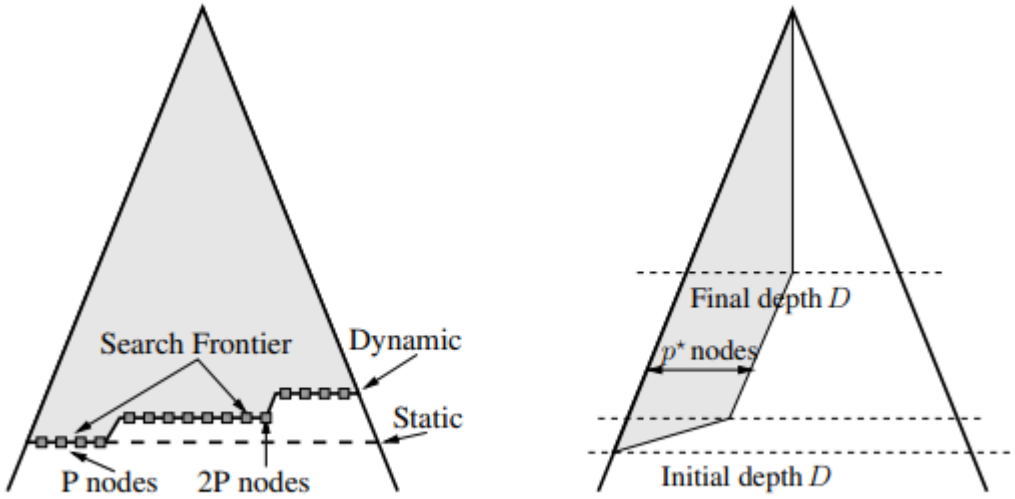
Top-Down decomposition method

- Starts from the root node and incrementally visits the next levels.
- This procedure assume that the variable ordering in the decomposition is static.
- Phases:
  - Start at the root node with an empty list of tuples;
  - Compute a list of p^* tuples that are solver-consistent decomposition by iteratively increasing the decomposition depth (depth-bounded depth first search).

Bottom-Up decomposition method

- This procedure bypasses the limitation of the static variable ordering and handles any branching strategy.
- Aims at identifying the topmost search frontier with approximately p^* open nodes by sampling and estimation.
- Phases:
  - Build a partial tree by sampling the top of real search tree;
  - Estimate the level widths of the real tree;
  - Determine the decomposition depth d^* with a greedy heuristic.

Bottom-Up decomposition method



(a) Decomposition. (b) Estimation.

My work

- Context:
  - Turbo is CP solver using GPU acceleration;
  - Currently supporting single-GPU execution.
- My research objective:
  - Extend Turbo to support multi-GPU execution by splitting large constraint problems into subproblems that can be solved in parallel across multiple GPUs, using MPI for communication.

Thank you!