Parallelism in Constraint Programming

Carl Christian Rolf
Okay, people, let’s get started on our orientation. I have 666 PowerPoint slides to cover.
‘Tiny Sudoku’

X, Y, Z, Q must have values between 1 and 2
The values in each row and column must be unique
Tiny Sudoku in Constraint Programming

X, Y, Z, Q must have values between 1 and 2
The values in each row and column must be unique

\[
\begin{array}{cc}
X & Y \\
Z & Q \\
\end{array}
\]

\[
\begin{align*}
\{X, Y, Z, Q\} & \in \{1..2\}, \\
X & \neq Y, \\
X & \neq Z, \\
Y & \neq Q, \\
Z & \neq Q, \\
solve.
\end{align*}
\]
Solving Tiny Sudoku in CP

SOLVE
\{X, Y, Z, Q\} \in \{1..2\}
X \neq Y, X \neq Z, Y \neq Q, Z \neq Q

X = 1

X = 1
\{Y, Z, Q\} \in \{1..2\}

Evaluate constraints:
X \neq Y, X \neq Z, Y \neq Q, Z \neq Q

X = 1, Y = 2,
Z = 2, Q = 1
END
Parallel Consistency for Tiny Sudoku

SOLVE
\{X, Y, Z, Q\} \in \{1..2\}
X \neq Y, X \neq Z, Y \neq Q, Z \neq Q

X = 1
\{Y, Z, Q\} \in \{1..2\}

Evaluate constraints:

CPU 1
X \neq Y

CPU 2
X \neq Z

CPU 3
Y \neq Q

CPU 4
Z \neq Q

X = 1, Y = 2,
Z = 2, Q = 1
END
Parallel Consistency: Summary

- The ‘task-parallelism’ of constraint programming
- Previous work has not handled global constraints
- Might have to run more iterations of consistency
- Load-balancing depends on the problem model
- Hard to share data during consistency
Parallel Consistency: Performance

Run on an eight core Mac Pro

<table>
<thead>
<tr>
<th>Number of Consistency Threads</th>
<th>Sudoku</th>
<th>LA31</th>
<th>Queens</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>2</td>
<td>[2]</td>
<td>[2]</td>
<td>[2]</td>
</tr>
<tr>
<td>8</td>
<td>[8]</td>
<td>[8]</td>
<td>[8]</td>
</tr>
</tbody>
</table>
Parallel Consistency: Conclusions

• Excellent performance for regular problems

• Some problems do not scale well, they need constraint-specific parallel consistency algorithms
Combining Parallel Consistency with Parallel Search
Parallel Search for Tiny Sudoku

**CPU 1**

**SOLVE**
\[
\{X, Y, Z, Q\} \in \{1..2\}
\]
\[
X \neq Y, X \neq Z, Y \neq Q, Z \neq Q
\]

\[X = 1\]

\[\{Y, Z, Q\} \in \{1..2\}\]

Evaluate constraints:
\[
X \neq Y, X \neq Z, Y \neq Q, Z \neq Q
\]

\[X = 1, Y = 2, Z = 2, Q = 1\]

**END**

**CPU 2**

\[X = 2\]

\[\{Y, Z, Q\} \in \{1..2\}\]

Evaluate constraints:
\[
X \neq Y, X \neq Z, Y \neq Q, Z \neq Q
\]

\[X = 2, Y = 1, Z = 1, Q = 2\]

**END**
Combining Parallelisms: Tiny Sudoku

CPU 1

SOLVE
{X, Y, Z, Q} ∈ \{1..2\}
X ≠ Y, X ≠ Z, Y ≠ Q, Z ≠ Q

X = 1

{Y, Z, Q} ∈ \{1..2\}

Evaluate constraints:

CPU 1
X ≠ Y

CPU 2
X ≠ Z

CPU 3
Y ≠ Q

CPU 4
Z ≠ Q

X = 1, Y = 2, Z = 2, Q = 1

END

CPU 5

X = 2

{Y, Z, Q} ∈ \{1..2\}

Evaluate constraints:

CPU 5
X ≠ Y

CPU 6
X ≠ Z

CPU 7
Y ≠ Q

CPU 8
Z ≠ Q

X = 2, Y = 1, Z = 1, Q = 2

END
Combining Parallelisms: Summary

• Never studied before in constraint programming

• Easier to achieve a speed-up than if the solver only offers one type of parallelism

• Does not fit all types of problems, often needs problem-specific optimization of, e.g., thread allocation
Combining Parallelisms: Results

Finding 200 solutions to 100x100 Sudoku. Run on 8-core Mac Pro

Consistency threads per search thread: 1, 2, 4, 8

Absolute Speed-up

Number of Search Threads
Combining Parallelisms: Conclusions

- Needs better control of mutual exclusion than currently offered by Java
- The problem must suit both types of parallelism to get a large performance benefit
- Problem-specific optimizations are necessary for good performance
Relative-Measured Load-Balancing
Load-Balancing for Tiny Sudoku

CPU 1

SOLVE
\{X,Y,Z,Q\} ∈ \{1..2\}
X ≠ Y, X ≠ Z, Y ≠ Q, Z ≠ Q

X = 1

{Y, Z, Q} ∈ \{1..2\}

Evaluate constraints:
X ≠ Y, X ≠ Z, Y ≠ Q, Z ≠ Q

X = 1, Y = 2,
Z = 2, Q = 1
END

CPU 2

Waiting for work

CPU 3

SOLVE
\{X,Y,Z,Q\} ∈ \{1..2\}
X ≠ Y, X ≠ Z, Y ≠ Q, Z ≠ Q

X = 1

{Y, Z, Q} ∈ \{1..2\}

Evaluate constraints:
X ≠ Y, X ≠ Z, Y ≠ Q, Z ≠ Q

X = 1, Y = 2,
Z = 2, Q = 1
END

Should CPU 1 or CPU 2 be allowed to send work?
Relative-Measured Load-Balancing

Busy solvers always compete for sending work to idle solvers
Relative-Measured Load-Balancing: Summary

• Infeasible to get an exact measure of the work-size due to the way CP solves problems

• We let solvers compete based on their work-size estimates

• We can use any measure that can be partially ordered

• Using measures from several solvers increases accuracy by eliminating systematic errors
Relative-Measured Load-Balancing: Performance

Golomb-12, proving optimality. Slowest and fastest measures

Random Polling  Least Labeled First  Most Labeled First

Absolute Speed-up

Number of Computers
Relative-Measured Load-Balancing: Conclusions

• Relative measures lets even simple estimates outperform random polling by over 20%

• Advanced measures can easily be used

• Performance benefit increases with the number of solvers
Dynamic Balancing of Communication and Computation
Dynamic Balancing for Tiny Sudoku

CPU 1

\[\text{SOLVE} \quad \{X,Y,Z,Q\} \in \{1..2\} \]
\[X \neq Y, X \neq Z, Y \neq Q, Z \neq Q\]

\[X = 1\]
\[\{Y, Z, Q\} \in \{1..2\}\]

Evaluate constraints:
\[X \neq Y, X \neq Z, Y \neq Q, Z \neq Q\]

\[X = 1, Y = 2, Z = 2, Q = 1\]

END

CPU 2

\[X = 2\]
\[\{Y, Z, Q\} \in \{1..2\}\]

Evaluate constraints:
\[X \neq Y, X \neq Z, Y \neq Q, Z \neq Q\]

\[X = 2, Y = 1, Z = 1, Q = 2\]

END
Tiny Sudoku: Zoomed in

**CPU 1**

**SOLVE**

\[
\{X, Y, Z, Q\} \in \{1..2\}
\]

**Evaluate constraints:**

\[
X \neq Y, X \neq Z, Y \neq Q, Z \neq Q
\]

**X = 1**

**Y, Z, Q\} \in \{1..2\}**

**X = 1, Y = 2, Z = 2, Q = 1**

**CPU 2**

**X = 2**

**Y, Z, Q\} \in \{1..2\}**

**Evaluate constraints:**

\[
X \neq Y, X \neq Z, Y \neq Q, Z \neq Q
\]

**X = 2, Y = 1, Z = 1, Q = 2**

Send either a copy of the variables, the domains, and the constraints; or just the assignments made by CPU 1.
Balancing Communication and Computation: Summary

- Copying sends a lot of data, but needs very little processing.
- Recomputation often needs a lot of processing, but sends little information.
- We estimate the network load to avoid getting stuck in performance bottlenecks.
Balancing Communication and Computation: Results

Proving optimal Golomb ruler of size 12

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Copying</th>
<th>Recomputation</th>
<th>Dual Com</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>
Balancing Communication and Computation: Conclusions

• Switching dynamically between copying and recomputation often increases performance

• Simple measure to estimate where the performance bottlenecks are
Distributed Constraint Programming with Agents (DCP)
Tiny Sudoku in DCP

Agent 1

\[ X \in \{1..2\} \]

\[ X \neq Z \]

Agent 2

\[ Y \in \{1..2\} \]

\[ Y \neq Q \]

Agent 3

\[ Z \in \{1..2\} \]

Agent 4

\[ Q \in \{1..2\} \]

\[ Z \neq Q \]

Solving starts in one agent, constraints communicate prunings
Our use of DCP

• To be used in UAVs in catastrophe areas
• We want independent agents to cooperate, to for instance share a heat camera
• We want to find good, preferably optimal, schedules
Example of Job Shop Scheduling

Two jobs, consisting of three tasks. Tasks have to execute on specific machines in a certain order.
DCP: In Contrast to the Traditional Approach

• A full constraint solver in each agent
• A set of variables in each agent
• A set of $n$-ary (global) constraints in each agent
• No use of memory-demanding table constraints
• Advanced search methods
## DCP: Experimental Results

### Traditional Model

<table>
<thead>
<tr>
<th>Problem</th>
<th>Time</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA01</td>
<td>&gt;30min</td>
<td>936</td>
</tr>
<tr>
<td>LA04</td>
<td>&gt;30min</td>
<td>976</td>
</tr>
<tr>
<td>LA05</td>
<td>&gt;30min</td>
<td>720</td>
</tr>
<tr>
<td>MT06</td>
<td>87.7s</td>
<td>55</td>
</tr>
</tbody>
</table>

### Our Model

<table>
<thead>
<tr>
<th>Problem</th>
<th>Time</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA01</td>
<td>3.8s</td>
<td>666</td>
</tr>
<tr>
<td>LA04</td>
<td>10.8s</td>
<td>590</td>
</tr>
<tr>
<td>LA05</td>
<td>0.7s</td>
<td>593</td>
</tr>
<tr>
<td>MT06</td>
<td>3.0s</td>
<td>55</td>
</tr>
</tbody>
</table>

We proved the optimum of all problems, the traditional model of none.
DCP: Conclusions

• Our model outperforms traditional models by orders of magnitude

• The best traditional approaches will remain slower, as speed-up is limited
Overall Conclusions

- We’ve developed and evaluated several new ways to parallelize constraint solving with global constraints.
- Our model of distributed constraint programming vastly outperforms traditional approaches.
- Parallelism in constraint programming can, with no understanding of parallel programming, give well-scaling performance for many kinds of problems.
Lastly, a nice Quote

“We finish this review with a single paper, probably one that best represents the state of the art [31].”


From:

Thank You!