# Parallel Constraint Logic Programming System GDCC

## ABSTRACT

GDCC is a parallel constraint logic programming language, where Constraint Logic Programming (*CLP*) is an amalgamation of logic programming paradigms with constraint programming paradigms. It is a very high level, flexible and efficient parallel programming language for problem solving.

We demonstrate the effectiveness of GDCC in various application fields by showing several example systems.

## DEMONSTRATIONS

1) GDCC

Showing the basic features of GDCC by using simple examples.

2) Handling Robot Design Support System

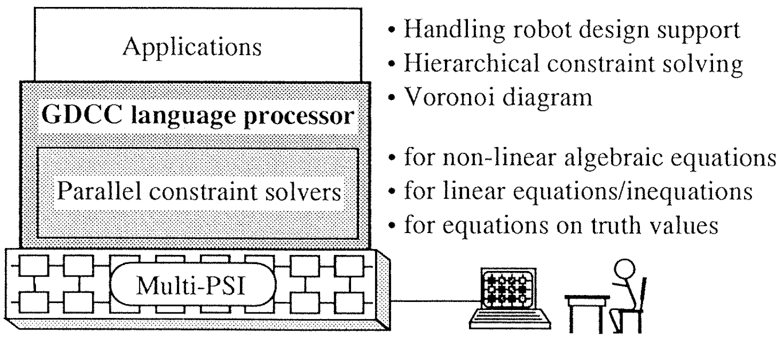
Showing the power and the flexibility of GDCC in the Design field.

3) Hierarchical Constraint Solving in Parallel

Showing the power and the efficiency of GDCC by writing a parallel hierarchical constraint solving system.

4) Voronoi Diagram

Showing the high level and flexibility of GDCC in the field of Computational Geometry.


Applications
GDCC language processor
Parallel constraint solvers
Multi-PSI
• Handling robot design support
• Hierarchical constraint solving
• Voronoi diagram
• for non-linear algebraic equations
• for linear equations/inequations
• for equations on truth values

GDCC System

## 1. GDCC LANGUAGE AND SYSTEM

### Purposes of the system

1) To provide a Highly Declarative Language based on Constraint Logic Programming Paradigms
2) To provide a Powerful and Flexible Problem Solving Framework
3) To provide an Efficient Problem Solving Framework by Employing Parallelism

### Problem Solving in Constraint Logic Programming

Constraints are relations that hold in a problem. Problem solving using Constraint Logic Programming is different from that using conventional programming languages:

- Problem solving using a conventional programming language
    1. Problem analysis
    2. Finding relationships between objects
    3. Finding a solution method
    4. Programming as a procedure to solve the problem
- Problem solving using a Constraint Logic Programming language
    1. Problem analysis
    2. Finding relationships between objects
    3. Programming as a set of constraints that hold in the problem

$\boxed{\text{How to solve} \Rightarrow \text{What to solve}}$

### Features of GDCC

1) Two levels of parallelism

   To realize a flexible and efficient constraint logic programming language, GDCC employs two levels of parallelism: one is the parallelization of the language and the other is the parallelization of constraint solvers.
2) Multiple constraint solvers
    1. Algebraic constraint solver
    2. Rational/Integer linear constraint solver
    3. Constraint solver for truth values

3) Block Mechanism

1. Multiple Environment

   Since the function to approximate the real roots of uni-variate equations is incorporated with the algebraic constraint solver; a mechanism is needed to handle the situation in which a variable may have multiple values.

2. Localize Failures

   To realize search function in a committed-choice language, a mechanism is needed to localize failures.

3. Specification of Synchoronization points between the inference engine and the constraint solvers

   To maximize or minimize a function with respect to a certain set of constraints, a mechanism is needed to specify the set of constraints, and to evaluate a goal with respect to the set of constraints. For example,

$$\mathit{maximize}(X+Y) \text{ under a set of constraints } \begin{cases} 0 \le X, \ X \le 1 \\ 0 \le Y, \ Y \le 1 \end{cases}$$

## GDCC programming examples

Heron's Formula

The following GDCC program deduces a property on a triangle, known as "Heron's formula" from three known properties: Pythagorean Theorem of a right-angle triangles, the formula for calculating the surface area of a triangle, and the fact that every triangle can be divided into two right-angle triangles.

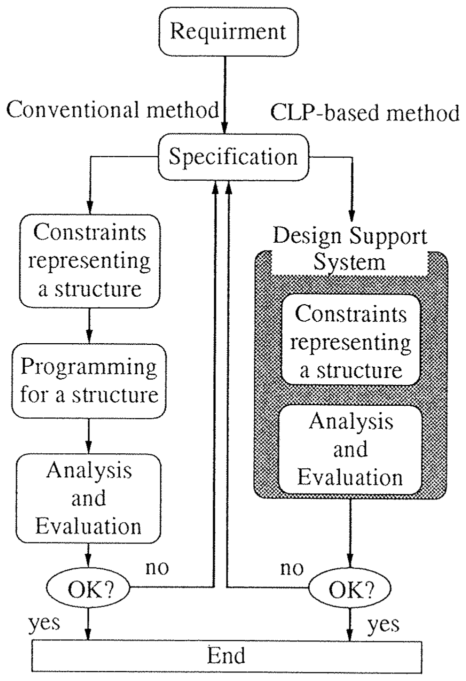
```
:- module heron.
:- public tri/4.

tri(A,B,C,S) :- true |
      alloc(0,CA,CB,H),
        alg#C=CA+CB,
        alg#CA**2+H**2=A**2,
        alg#CB**2+H**2=B**2,
        alg#H*C=S.
```

## 2. HANDLING ROBOT DESIGN SUPPORT SYSTEM

### Features

A Robot Design Support System that uses the Flexibility of Constraint Logic Programming.

### Handling Robot Design Process



1) Any robot structure can be handled by only changing the specification.
2) There is no need to write an individual program to analyze the design.

### Functions of the system

The Design Support System can:

- Create constraints by
    - Solving Forward Kinematics.

- Analyze and evaluate the robot by
  - Solving Inverse Kinematics,
  - Calculating the Torque working on each Joint, and
  - Calculating the Manipulability of Robot being designed.

## Demonstration

The handling robot in the following "handling robot design support system" demonstrations is a robot with 3 joints and 3 arms.

1) Forward Kinematics

Deducing the position of the end-effector $(P_x, P_y, P_z)$ in terms of the length of each arm $(l_1, l_2, l_3)$ and the rotation angle of each joint $(\theta_1, \theta_2, \theta_3)$.

$$\textit{Specification} \Rightarrow \begin{cases} P_x = P_x(\theta_1, \theta_2, \theta_3, l_1, l_2, l_3) \\ P_y = P_y(\theta_1, \theta_2, \theta_3, l_1, l_2, l_3) \\ P_z = P_z(\theta_1, \theta_2, \theta_3, l_1, l_2, l_3) \end{cases}$$

2) Inverse Kinematics

Calculating the desired joint rotation angles $(\theta_1, \theta_2, \theta_3)$ to move the end-effector to a given position $(P_x, P_y, P_z)$. Since this problem have more than two solutions, filtering to reduce the number of solutions is demonstrated by restricting the rotation angle of a certain joint.

$$\textit{Specification} \Rightarrow \begin{cases} \theta_1 = \theta_1(P_x, P_y, P_z, l_1, l_2, l_3) \\ \theta_2 = \theta_2(P_x, P_y, P_z, l_1, l_2, l_3) \\ \theta_3 = \theta_3(P_x, P_y, P_z, l_1, l_2, l_3) \end{cases}$$

3) Calculation of Torque and Evaluation of Manipulability

By using the forward kinematics results $(P_x, P_y, P_z)$ and giving the force working on the end effector $(F_x, F_y, F_z)$, equations to calculate the torque working on each joint $(T_1, T_2, T_3)$ and the manipulability $(D)$ are deduced. Then, by placing constraints on moving the end-effector, the equations for torques and manipulability are simplified.

$$\left.\begin{array}{l} P_x = P_x(\theta_1, \theta_2, \theta_3, l_1, l_2, l_3) \\ P_y = P_y(\theta_1, \theta_2, \theta_3, l_1, l_2, l_3) \\ P_z = P_z(\theta_1, \theta_2, \theta_3, l_1, l_2, l_3) \\ F_x,\ F_y,\ F_z \end{array}\right\} \Rightarrow \left\{\begin{array}{l} T_1 = T_1(\theta_1, \theta_2, \theta_3, F_x, F_y, F_z) \\ T_2 = T_2(\theta_1, \theta_2, \theta_3, F_x, F_y, F_z) \\ T_3 = T_3(\theta_1, \theta_2, \theta_3, F_x, F_y, F_z) \\ D = D(\theta_1, \theta_2, \theta_3) \end{array}\right.$$

## 3. HIERARCHICAL CONSTRAINT SOLVING IN PARALLEL

### Constraint Hierarchy

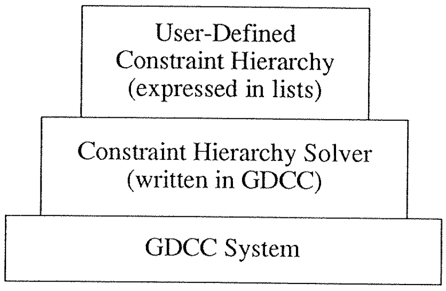
1. Introduces various strengths of constraints
2. Important in synthesis problems

### Features

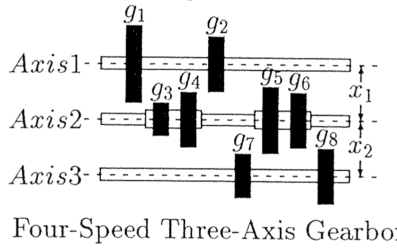
Solving Constraint Hierarchy in Parallel by GDCC using the Block Mechanism

1. We show how to solve constraint hierarchy by the block mechanism.
2. We show speed-up by parallel execution on Multi-PSI.

### System Architecture



### Demonstration: Gear-Box Design



Four-Speed Three-Axis Gearbox

We specify the sum of the two intervals between axes and output the speed ratios, each of which is produced by combining two meshing pairs. We assume that there are standard radiuses of gears which take discrete values. We decide each gear radius so that standard gears can hopefully be used as many as possible.

## Example of Design Problem

**Hard Constraints** (speed ratio and distance):

$ratio(\langle g_1, g_3\rangle, \langle g_5, g_7\rangle) = 1$ (a)

$ratio(\langle g_2, g_4\rangle, \langle g_5, g_7\rangle) = 2$ (b)

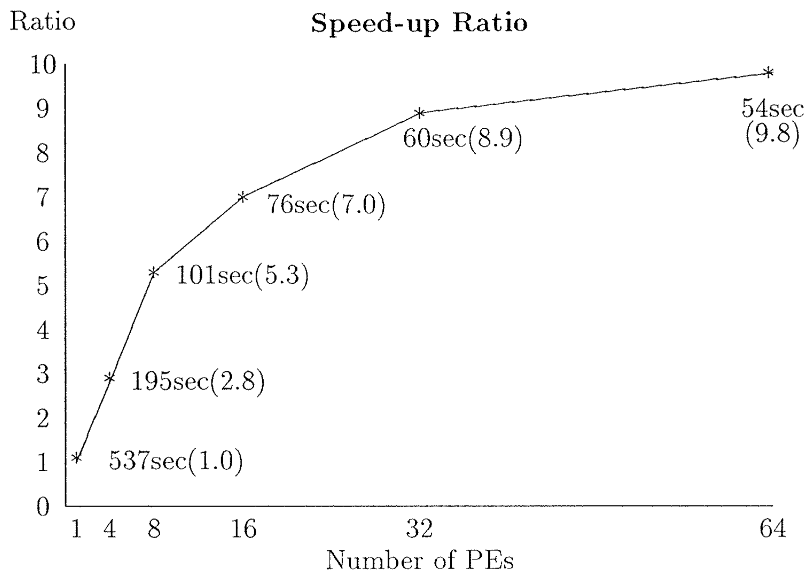
$ratio(\langle g_1, g_3\rangle, \langle g_6, g_8\rangle) = 4$ (c)

$ratio(\langle g_2, g_4\rangle, \langle g_6, g_8\rangle) = 8$ (d)

$x_1 + x_2 = 10$ (e)

**Soft Constraints** (use of standard gear):
The radius should preferably be 1, 2, 3, or 4.

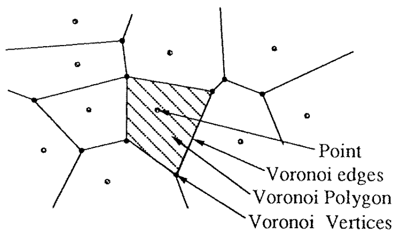
## Result



## 4. VORONOI DIAGRAM

### Voronoi Diagram

The Voronoi Diagram of a finite set $S$ of points in the plane is a partition of the plane so that each region of the partition is a set of points which are closer to one point in $S$ in the region than to any other point in $S$.



### Features

The Voronoi Diagram Program uses the High-level and Flexibility of Constraint Logic Programming.

1) The algorithm is relatively straightforward.

2) The algorithm is $O($ The number of points $N)$.

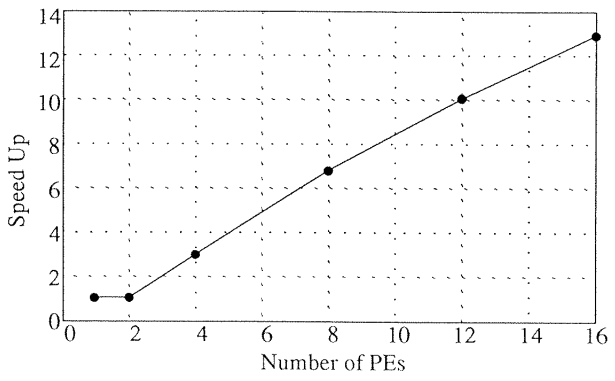
3) The definition of the distance can be changed easily.

### Demonstration

1) Constructing the Voronoi diagram for given points

2) Changing the definition of the Voronoi diagram

### Result


Speed Up
Number of PEs