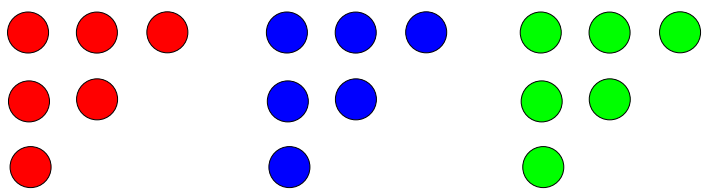


# Component-level Parallelization of Triangular Decompositions

**Marc Moreno Maza & Yuzhen Xie**

**University of Western Ontario, Canada**

**February 1, 2007**

Interactive Parallel Computation in Support of Research in Algebra, Geometry and Number Theory
MSRI Workshop 2007, Berkeley.

# Solving polynomial systems symbolically

- Strengths of symbolic solving:
  - parametric polynomial systems solving,
  - real algebraic geometry,
  - solving in non-zero characteristic.
- Increasing number of applications:
  - cryptology, robotics, signal processing
  - geometric modeling (AGGM community),
  - algebraic study of dynamical systems (in biology, chemsitry, ...).
- Strengths of triangular decomposition:
  - size of coefficients, running time complexity estimates,
  - fast arithmetic, sharp modular methods,
  - geometrical information.

# Gröbner bases and triangular decompositions

$$\begin{cases} x^2+y+z=1 \\ x+y^2+z=1 \\ x+y+z^2=1 \end{cases} \quad \underline{\text{has Gröbner basis}} :$$

$$\begin{cases} z^6-4z^4+4z^3-z^2=0 \\ 2z^2y+z^4-z^2=0 \\ y^2-y-z^2+z=0 \\ x+y+z^2-1=0 \end{cases} \quad \underline{\text{and triangular decomposition}} :$$

$$\begin{cases} z=1 \\ y=0 \\ x=0 \end{cases} \bigcup \begin{cases} z=0 \\ y=1 \\ x=0 \end{cases} \bigcup \begin{cases} z=0 \\ y=0 \\ x=1 \end{cases} \bigcup \begin{cases} z^2+2z-1=0 \\ y=z \\ x=z \end{cases}$$

# Triangular decompositions

- The zero set $V(F)$ admits a decomposition (unique when minimal)

$$V(F) = V(F_1) \cup \cdots \cup V(F_e),$$

s.t. $F_1, \ldots, F_e \subset \mathbb{K}[X]$ and every $V(F_i)$ **cannot** be decomposed further.

- Moreover, up to technical details, each $V(F_i)$ is the zero set of a **triangular system**, which can view as a **solved system**

$$\begin{cases} T_n(x_1, \ldots, x_d, x_{d+1}, x_{d+2}, \ldots, x_{n-1}, \mathbf{x_n}) = 0 \\ T_{n_1}(x_1, \ldots, x_d, x_{d+1}, x_{d+2}, \ldots, \mathbf{x_{n-1}}) = 0 \\ \vdots \\ T_{d+2}(x_1, \ldots, x_d, x_{d+1}, \mathbf{x_{d+2}}) = 0 \\ T_{d+1}(x_1, \ldots, x_d, \mathbf{x_{d+1}}) = 0 \\ h(x_1, \ldots, x_d) \neq 0 \end{cases}$$

# Parallelizing the computation of Gröbner bases

**Input:** $F \subset \mathbb{K}[X]$ and an admissible monomial ordering $\leq$.

**Output:** $G$ a reduced Gröbner basis w.r.t. $\leq$ of the ideal $\langle F \rangle$ generated by $F$.

**repeat**

(S) $B :=$ MinimalAutoreducedSubset($F$, $\leq$)

(R) $A :=$ S_Polynomials($B$) $\cup\, F$;

$R :=$ Reduce($A$, $B$, $\leq$)

(U) $R := R \setminus \{0\}$; $F := F \cup R$

**until** $R = \emptyset$

**return** $B$

# The parallel characteristic set method

**Input:** $F \subset \mathbb{K}[X]$.

**Output:** $C$ an autoreduced characteristic set of $F$ (in the sense of Wu).

**repeat**

(S) $B :=$ MinimalAutoreducedSubset($F$, $\leq$)

(R) $A := F \setminus B$;

$R :=$ PseudoReduce($A$, $B$, $\leq$)

(U) $R := R \setminus \{0\}$; $F := F \cup R$

**until** $R = \emptyset$

**return** $B$

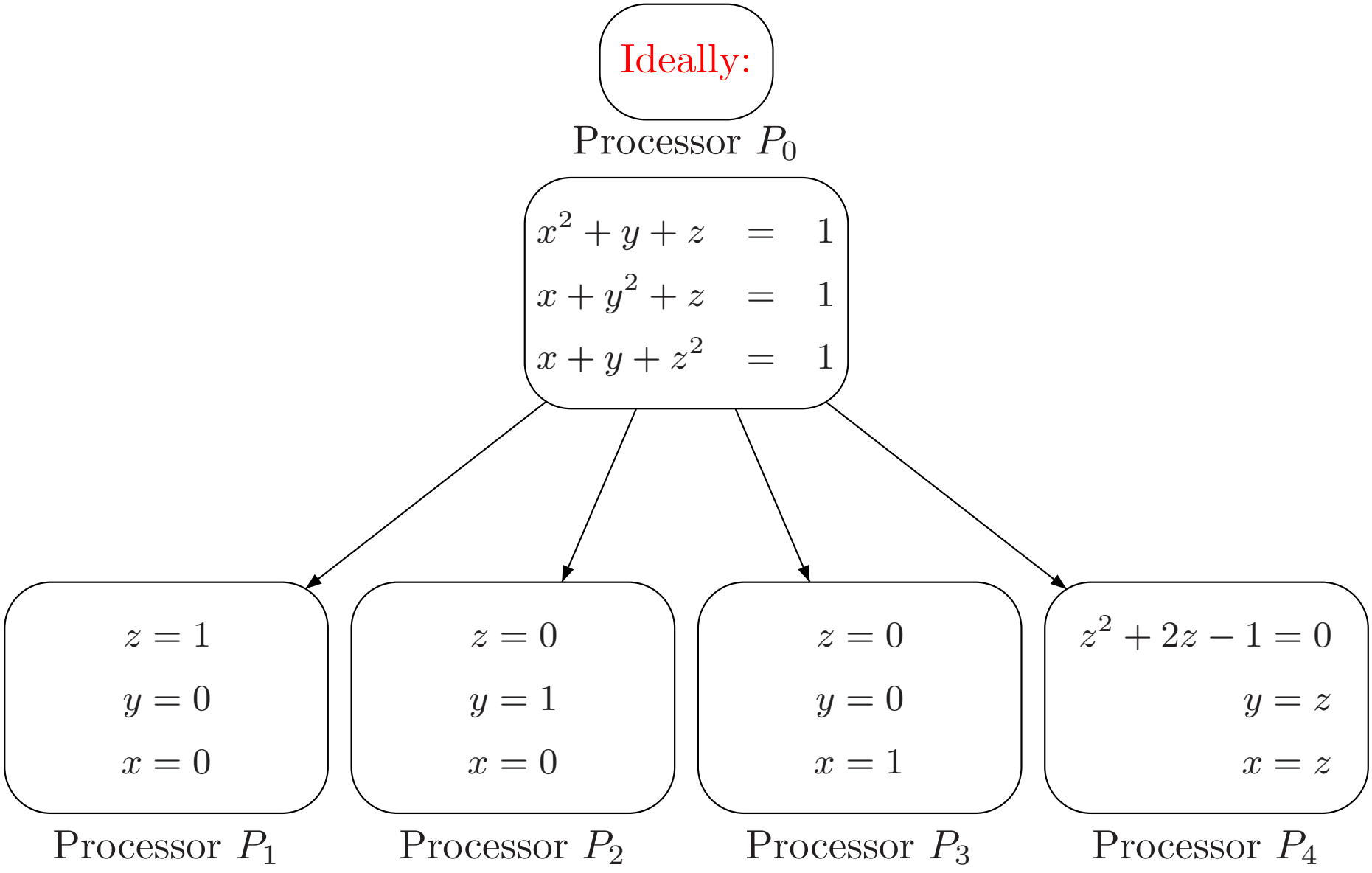
- Repeated calls to this procedure computes a decomposition of $V(F)$.
- Cannot start computing the 2nd component before the 1st is completed.

## Solving polynomial systems symbolically and in parallel!

- Related work :
  - Parallelizing the computation of Gröbner bases (R. Bündgen, M. Göbel & W. Küchlin, 1994) (S. Chakrabarti & K. Yelick, 1993 - 1994) (G. Attardi & C. Traverso, 1996) (A. Leykin, 2004)
  - Parallelizing the computation of characteristic sets (I.A. Ajwa, 1998), (Y.W. Wu, W.D. Liao, D.D. Liu & P.S. Wang, 2003) (Y.W. Wu, G.W. Yang, H. Yang, H.M. Zheng & D.D. Liu, 2005)

# Solving polynomial systems symbolically and in parallel!

- **New motivations**:
  - renaissance of parallelism,
  - new algorithms offering better opportunities for parallel execution.
- **Our goal**:
  - multi-level parallelism:
    - coarse grained "component-level" for tasks computing geometric objects,
    - medium/fine grained level for polynomial arithmetic within each task.
  - to develop a solver for which the number of processes in use depends on the geometry of the solution set


Ideally:
Processor $P_0$
$x^2 + y + z = 1$
$x + y^2 + z = 1$
$x + y + z^2 = 1$
$z = 1$
$y = 0$
$x = 0$
$z = 0$
$y = 1$
$x = 0$
$z = 0$
$y = 0$
$x = 1$
$z^2 + 2z - 1 = 0$
$y = z$
$x = z$
Processor $P_1$
Processor $P_2$
Processor $P_3$
Processor $P_4$

# Dynamic and very irregular computations (1/2)

• How do splits occur during decompositions? Given a polynomial ideal $\mathcal{I}$ and polynomials $p, a, b$, there are two rules:

- $\mathcal{I} \longmapsto (\mathcal{I} + p, \mathcal{I} : p^\infty)$.
- $\mathcal{I} + \langle a\, b \rangle \longmapsto (\mathcal{I} + \langle a \rangle, \mathcal{I} + \langle b \rangle)$.

• The second one is more likely to **split computations evenly**. But geometrically, it means that a component is **reducible**.

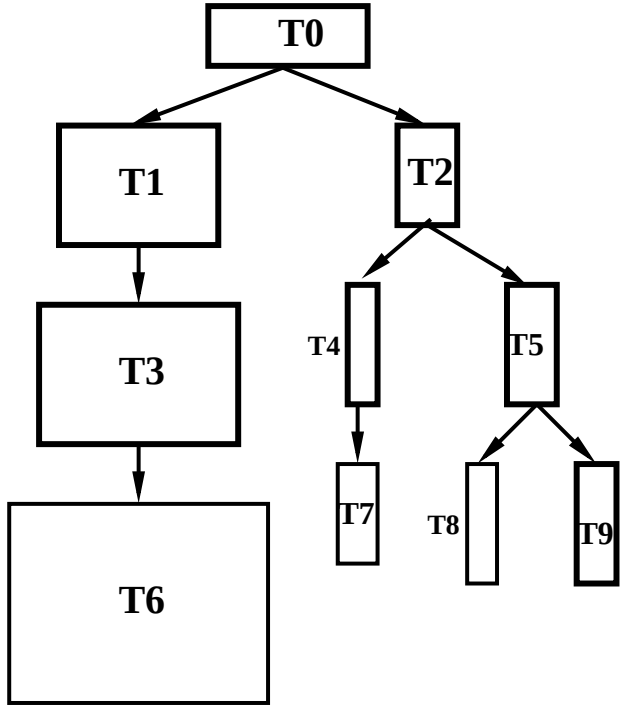
• Unfortunately, most polynomial systems $F \subseteq \mathbb{Q}[X]$ are **equiprojectable**, that is, they can be represented by a single triangular set.

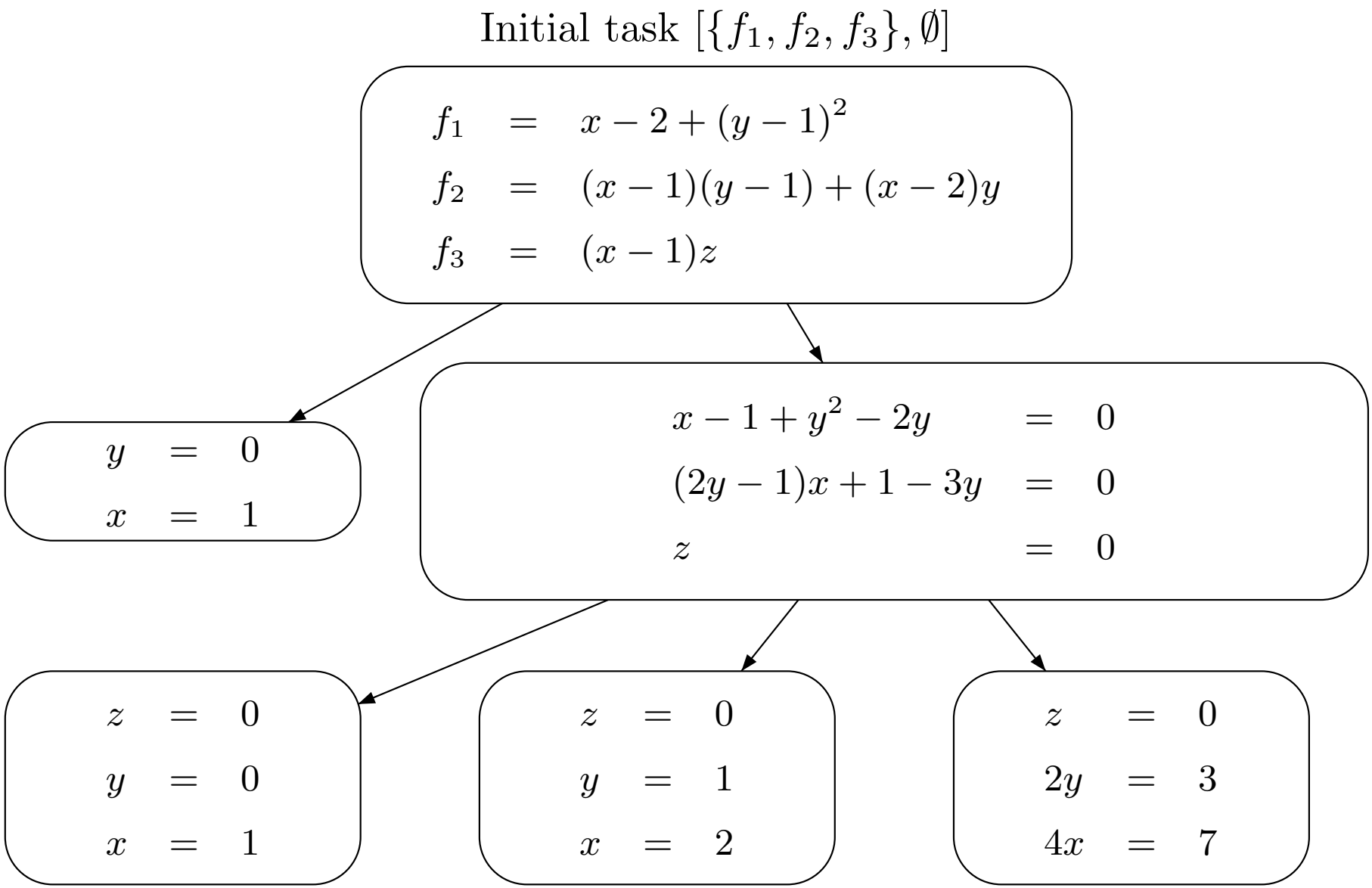
- This is true in theory, shape lemma: (Becker, Mora, Marinari & Traverso, 1994).
- and in practice: `SymbolicData.org`.

• However, for $F \subseteq Z/pZ[X]$ where $p$ prime, the second rule is more likely to be used.

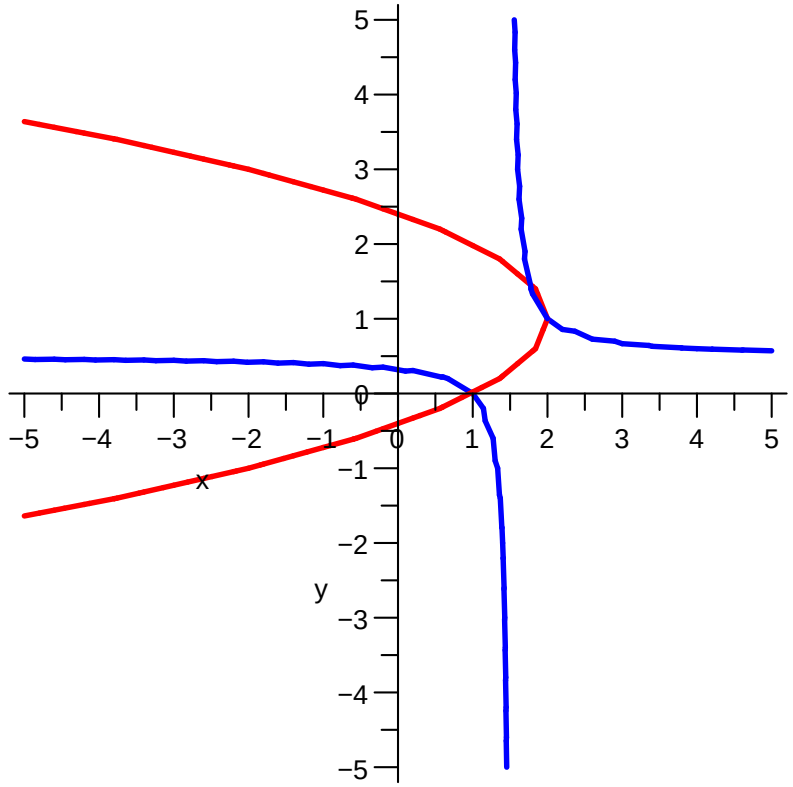
# Dynamic and very irregular computations (2/2)

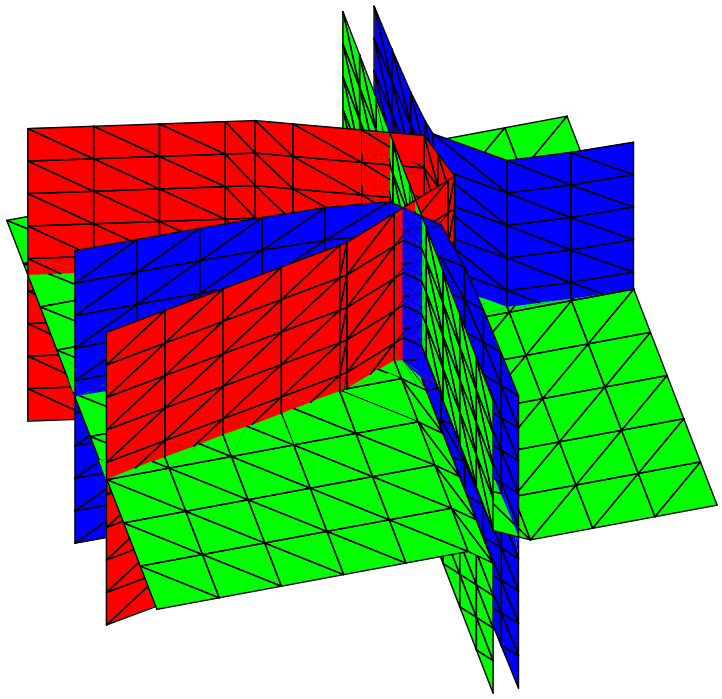
- Very irregular tasks (CPU time, memory, data-communication)




Initial task $[\{f_1, f_2, f_3\}, \emptyset]$
$f_1 = x - 2 + (y-1)^2$
$f_2 = (x-1)(y-1) + (x-2)y$
$f_3 = (x-1)z$
$y = 0$
$x = 1$
$x - 1 + y^2 - 2y = 0$
$(2y-1)x + 1 - 3y = 0$
$z = 0$
$z = 0$
$y = 0$
$x = 1$
$z = 0$
$y = 1$
$x = 2$
$z = 0$
$2y = 3$
$4x = 7$

# Even worse: redundant and irregular tasks



The red and blue surfaces intersect on the line $x - 1 = y = 0$ contained in the green plane $x = 1$. With the other green plane $z = 0$, they intersect at $(2, 1, 0)$, $(\frac{7}{4}, \frac{3}{2}, 0)$ but also at $x - 1 = y = z = 0$, which is redundant.

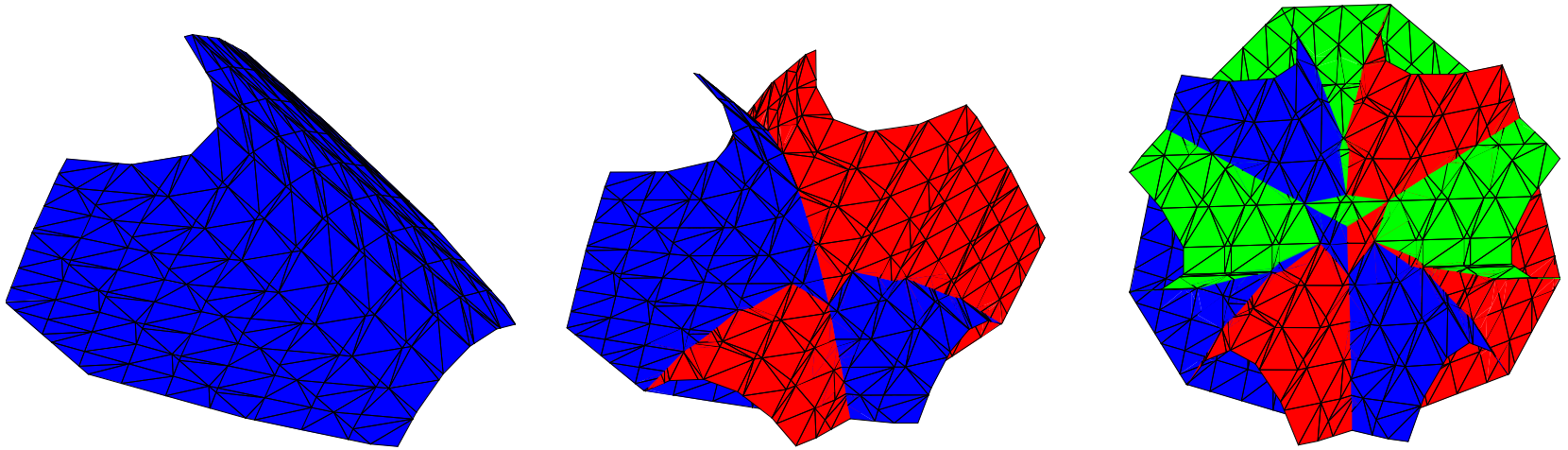
# How to create and exploit parallelism?

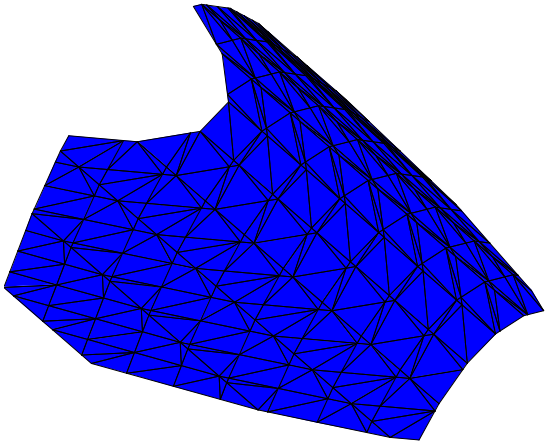
- We start from an algorithm (Triade, MMMM 2000)
  - based on incremental solving for exploiting geometrical information,
  - solving by decreasing order of dimension for removing redundant components efficiently.
- This algorithm
  - uses a task model for the intermediate computations,
  - but a naive parallelization would be unsuccessful, for the reasons mentioned earlier.
- So, we rely also
  - on the modular algorithm (Dahan, MMM, Schost, Wu and Xie, 2005)
  - in order to create opportunities for parallel execution with input systems over $\mathbb{Q}$.
- We adapt the Triade algorithm to exploit the opportunities created by the modular techniques.

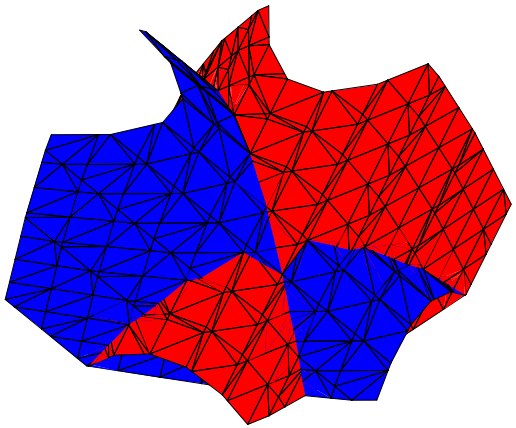
# Triangular Decompositions: a geometrical approach

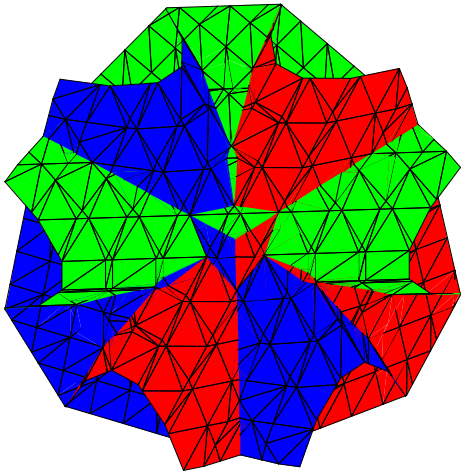
Incremental solving: by solving one equation after the other, lead to a **more geometric approach**

$$\begin{cases} x^2 + y + z = 1 \end{cases} \qquad \begin{cases} x^2 + y + z = 1 \\ x + y^2 + z = 1 \end{cases} \qquad \begin{cases} x^2 + y + z = 1 \\ x + y^2 + z = 1 \\ x + y + z^2 = 1 \end{cases}$$









$$\left\{ \; x^2 + y + z = 1 \right.$$

$$\left\{ \begin{array}{lcl} x & + & y^2 + z = 1 \\ y^4 & + & (2z-2)y^2 \\ & + & y - z + z^2 = 0 \end{array} \right.$$

$$\left\{ \begin{array}{r} x + y = 1 \\ y^2 - y = 0 \\ z = 0 \end{array} \right.$$

$$\left\{ \begin{array}{r} 2x + z^2 = 1 \\ 2y + z^2 = 1 \\ z^3 + z^2 - 3z = -1 \end{array} \right.$$

# Triangular sets and regular chains

- Let $\mathbf{T} = T_1, \ldots, T_s \subset \mathbb{K}[x_1 < \cdots < x_n]$. with main variables $x_{\ell_1} < \cdots < x_{\ell_s}$, that is, a triangular set.
- For $i \leq s$, the initial $h_i$ is the leading coefficient of $T_i$ in $X_{\ell_i}$.
- Define $\mathrm{Sat}_i(\mathbf{T}) = (T_1, \ldots, T_i) : (h_1 \ldots h_i)^\infty$.

$\mathbf{T}$ is a regular chain if $h_i$ is regular mod $\mathrm{Sat}_{i-1}(\mathbf{T})$ for all $i \geq 2$.

- The quasi-component $W(\mathbf{T}) := V(\mathbf{T}) \setminus V(h_1 \cdots h_{\ell_s})$ satisfies $\overline{W(\mathbf{T})} = V(\mathrm{Sat}_n(\mathbf{T}))$ where $dim(\mathrm{Sat}_n(\mathbf{T})) = n - |T|$.
- The algebraic variables of $T$ are those which appear as main variables.

Example

$$\left| \begin{array}{l} T_2 = (X_1 + X_2){X_3}^2 + X_3 + 1 \\ T_1 = {X_1}^2 + 1. \end{array} \right. , \text{ with } \left| \begin{array}{l} \mathsf{mvar}(T_2) = X_3 \\ \mathsf{mvar}(T_1) = X_1 \end{array} \right. .$$

# Task Model

- A task is any $[F, S]$ whith $F \subset \mathbb{K}[X]$ finite and $S \subset \mathbb{K}[X]$ triangular set.

- The task $[F, S]$ is solved if $F = \emptyset$ and $S$ is a regular chain.

- Solving $[F, S]$ means computing regular chains $T_1, \ldots, T_e$ such that:

$$V(F) \cap W(S) \quad \subseteq \quad \cup_{i=1}^{e} W(T_i) \quad \subseteq \quad V(F) \cap \overline{W(S)}.$$

- $[F_1, S_1] \prec [F_2, S_2]$ if either $S_1 \prec S_2$ or $(\exists f_1 \in F_1)\ (\forall f_2 \in F_2)\ f_1 \prec f_2$ and $S_1 \simeq S_2$.

- The tasks $[F_1, S_1], \ldots, [F_e, S_e]$ form a delayed split of $[F, S]$ if for all $1 \leq i \leq e$ we have $[F_i, S_i] \prec [F, S]$ and the following holds

$$V(F) \cap W(S) \quad \subseteq \quad \cup_{i=1}^{e} V(F_i) \cap W(S_i) \quad \subseteq \quad V(F) \cap \overline{W(S)}.$$

# The main operations

- Triangularize$(F, S)$ returns regular chains $T_1, \ldots, T_e$ solving $[F, S]$.

- When $F = \emptyset$ we write Extend$(S)$ instead of Triangularize$(F, S)$.

- For $p$ a polynomial and $T$ a regular chain, Decompose$(p, T)$ returns a delayed split of $[\{p\}.T]$. Hence Decompose$(p, T)$ computes $V(p) \cap W(T)$ by lazy evaluation.

- For $p, t$ polynomials with same main variable and $T$ a regular chain, GCD$(p, t, T)$ returns pairs $(g_1, T_1), \ldots, (g_e, T_e)$ such that

  - whenever $|T_i| = |T|$ holds, $g_i$ is a GCD of $p$ and $t$ modulo $T_i$,

  - $T_1, \ldots, T_e$ solve $T$, that is, $W(T) \subseteq \cup_{i=1}^{e} W(T_i) \subseteq \overline{W(T)}$.

- RegularizeInitial$(p, T)$ returns regular chains $T_1, \ldots, T_e$ solving $T$ and such that $p \mod \mathrm{Sat}(T_i)$ is constant or its initial is regular.

These operations rely on the D5 Principle (Della Dora et al., 1985).

# The *Triangularize* algorithm

**Input:** a task $[F, T]$

**Output:** regular chains $T_1, \ldots, T_e$ solving $[F, T]$

Triangularize$(F, T)$ == **generate**

1 $R := [[F, T]]$

2 # $R$ is a list of tasks

3 **while** $R \neq [\,]$ **repeat**

4 choose and remove a task $[F_1, U_1]$ from $R$

5 $F_1 = \emptyset \Longrightarrow$ **yield** $U_1$

6 choose a polynomial $p \in F_1$

7 $G_1 := F_1 \setminus \{p\}$

8 **for** $[H, T] \in$ Decompose$(p, U_1)$ **repeat**

9 $R :=$ cons $([G_1 \cup H, T], R)$

# The *Decompose* algorithm

**Input:** a polynomial $p$ and a regular chain $T$ such that $p \notin \mathrm{Sat}(T)$.

**Output:** a delayed split of $[\{p\}, T]$.

$\mathsf{Decompose}(p, T) ==$ **generate**

1 **for** $C \in \mathrm{RegularizeInitial}(p, T)$ **repeat**

2 $f := \mathsf{Reduce}(p, C)$

3 $f = 0 \Longrightarrow$ **yield** $[\emptyset, C]$

4 $f \in \mathbb{K} \Longrightarrow$ **iterate**

5 $v := \mathrm{mvar}(f)$

6 $v \notin \mathrm{mvar}(C) \Longrightarrow$ **yield** $[\{\mathrm{init}(f), p\}, C]$

7 **for** $D \in \mathsf{Extend}(C \cup \{f\})$ **repeat yield** $[\emptyset, D]$

8 **for** $[F, E] \in \mathsf{AlgebraicDecompose}(f, C_{<v} \cup C_{>v}, C_v)$

9 **repeat yield** $[F, E]$

# The *AlgebraicDecompose* algorithm

**Input:** $p, T, t$ as in the definition of GCD.

**Output:** a delayed split of $[\{p\}, T \cup \{t\}]$.

AlgebraicDecompose$(p, T, t)$ == **generate**

1 Let $h_T$ be the product of the initials in $T$
2 $f := t\, h_T$
3 **for** $[g_i, T_i] \in \mathsf{GCD}(t, p, T)$ **repeat**
4 $|T_i| > |T| \Longrightarrow$
5 **for** $T_{i,j} \in \mathsf{Extend}(T_i \cup \{f\})$ **repeat yield** $[p, T_{i,j}]$
6 $g_i \in \mathbb{K} \Longrightarrow$ **iterate**
7 $\mathrm{mvar}(g_i) < v \Longrightarrow$ **yield** $[\{g_i, p\}, T_i \cup \{t\}]$
8 $\deg(g_i, v) = \deg(t, v) \Longrightarrow$ **yield** $[\emptyset, T_i \cup \{t\}]$
9 **yield** $[\emptyset, T_i \cup \{g_i\}]$
10 **yield** $[\{\mathrm{init}(g_i), p\}, T_i \cup \{t\}]$

# Solving by decreasing order of dimension

• A task $[G, C]$ output by Decompose$(p, T)$ is solved, that is $G = \emptyset$, if and only $\overline{W(C)}$ is a component of $V(p) \cap \overline{W(T)}$ with maximum dimension.

⇒ Tasks in Triangularize can be chosen such that regular chains are generated by decreasing order of dimension (= increasing height).

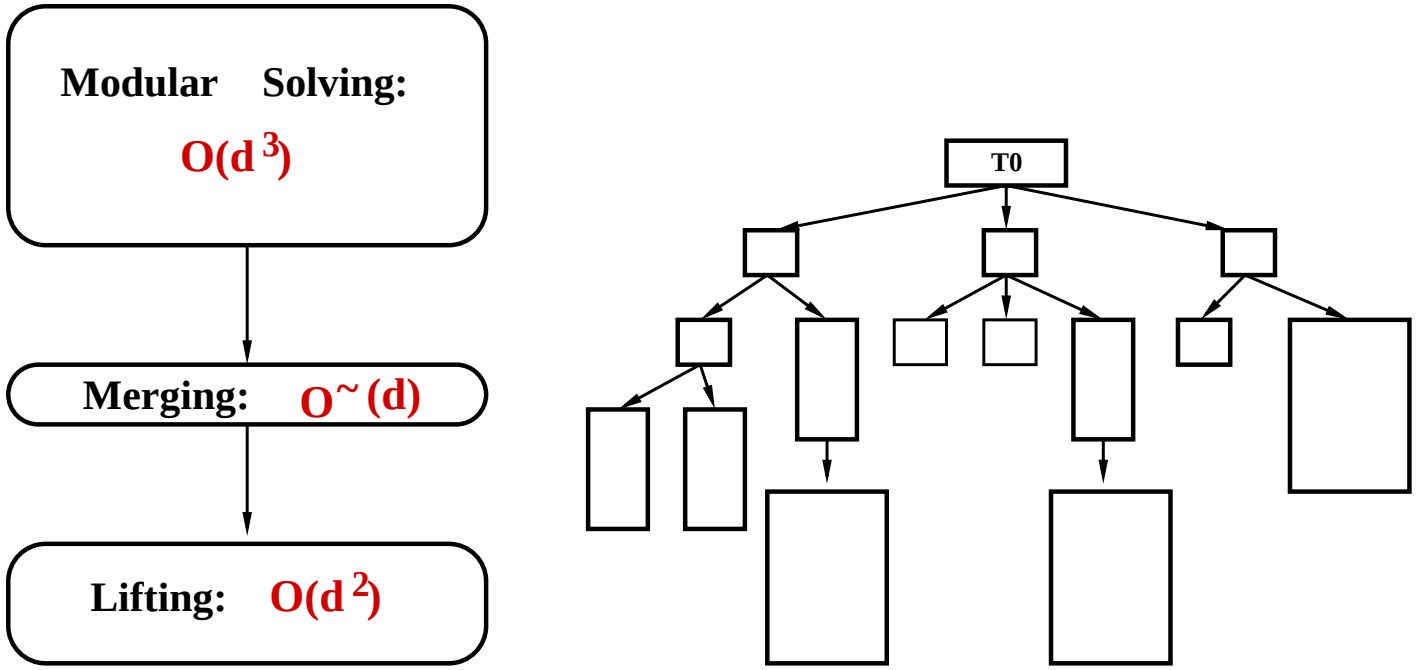
Allow efficient control of redundant components.

• To do so:

- each task $[F_i, T_i]$ is assigned a priority based on *rank* and *dimension* considerations.
- a task chosen in $R$ has maximum priority.

⇒ This leads to a first parallel version of Triangularize where all tasks of maximum priority are executed concurrently.

# Create parallelism: Using modular techniques




For solving $F \subseteq \mathbb{Q}[X]$ we use modular methods. Indeed, for a prime $p$:

- irreducible polynomials in $\mathbb{Q}[X]$ are likely to factor modulo $p$,
- for $p$ big enough, the result over $\mathbb{Q}$ can be recovered from the one over $Z/pZ[X]$.

(X. Dahan, M. Moreno Maza, É. Schost, W. Wu & Y. Xie, 2005)

# Effect of modular solving

| Sys | Name | $n$ | $d$ | $p$ | *Degrees* |
|---|---|---|---|---|---|
| 1 | eco6 | 6 | 3 | 105761 | [1,1,2,4,4,4] |
| 2 | eco7 | 7 | 3 | 387799 | [1,1,1,1,4,2,<br>4,4,4,4,4,2] |
| 3 | CassouNogues2 | 4 | 6 | 155317 | [8] |
| 4 | CassouNogues | 4 | 8 | 513899 | [8,8] |
| 5 | Nooburg4 | 4 | 3 | 7703 | [18,6,6,3,3,4,4,4,4,2,2,2,<br>2,2,2,2,1,1,1,1,1] |
| 6 | UteshevBikker | 4 | 3 | 7841 | [1,1,1,1,2,30] |
| 7 | Cohn2 | 4 | 6 | 188261 | [3,5,2,1,2,1,1,16,12,10,8,8,<br>4,6,4,4,4,4,2,1,1,1,1,1,1,1,<br>1,1,1,1,1,1,1] |

# Exploit parallelism!

- Driving idea: limit the irregularity of tasks. In particular,
  - we want to avoid inexpensive computations leading to expensive data communication.
  - we want to balance the work among the workers.
- Means:
  - Reduce the number of parallel steps in Triangularize to be at most $n$.
  - Replace Decompose by an operation SplitByHeight with a stronger requirement, and thus, more work.
  - Be able to estimate the cost of processing a task by SplitByHeight.
- We strengthen the notion of a task $[F, T]$: for every $f \in F$ the initial of $f$ is regular w.r.t. $T$.

$\Rightarrow$ When calling Decompose$(p, T)$ is called, we know which operation will be performed.

# The *Split-by-height* strategy

**Input:** a task $[F, T]$

**Output:** a delayed split of $[F, T]$ such that for all output task $[G, U]$: $|U| = |T| \implies G = \emptyset$.

SplitByHeight$(F, T)$ == **generate**

1 $R := [[F, T]]$ # $R$ is a list of tasks
2 **while** $R \neq [\,]$ **repeat**
3 choose and remove a task $[F_1, U_1]$ from $R$
4 $|U_1| > |T| \implies$ **yield** $[F_1, U_1]$
5 $F_1 = \emptyset \implies$ **yield** $[F_1, U_1]$
6 choose a polynomial $p \in F_1$
7 $G_1 := F_1 \setminus \{p\}$
8 **for** $[H, T] \in$ Decompose$(p, U_1)$ **repeat**
9 $R :=$ cons $([G_1 \cup H, T], R)$

## A new Triangularize$(F, T)$ based on SplitByHeight

**Input:** a task $[F, T]$

**Output:** regular chains $T_1, \ldots, T_e$ solving $[F, T]$.

Triangularize$(F, T)$ == **generate**

1 $R := [[F, T]]$ # $R$ is a list of tasks
2 **while** $R \neq [\,]$ **repeat**
3 choose and remove $[F_1, U_1] \in R$ with max priority
4 $F_1 = \emptyset \Longrightarrow$ **yield** $U_1$
5 **for** $[H, T] \in$ SplitByHeight$(F_1, U_1)$ **repeat**
6 $R :=$ cons $([H, T], R)$
7 sort $R$ by decreasing priority

## Challenges in the implementation

- Encoding complex mathematical objects and sophisticated algorithms.
- Handling dynamic process creation and management (MPI is not sufficient).
- Scheduling of highly irregular tasks.
- Managing huge intermediate data (with coefficient swell).
- Handling synchronization.

# Preliminary implementation: Environment

- We use the ALDOR programming language:
  - designed to express the structures and algorithms in computer algebra,
  - which provides a two-level object model of *categories* and *domains*,
  - which provides interoperability with other languages, such as C, for high-performance computing.
- We use the **BasicMath** library (100,000 lines) in ALDOR which provides a sequential implementation of the Triade algorithm.
- We targeted multi-processed parallelism on SMPs.
- Unfortunately, there was no available support in ALDOR for our needs.

# Communication and synchronization

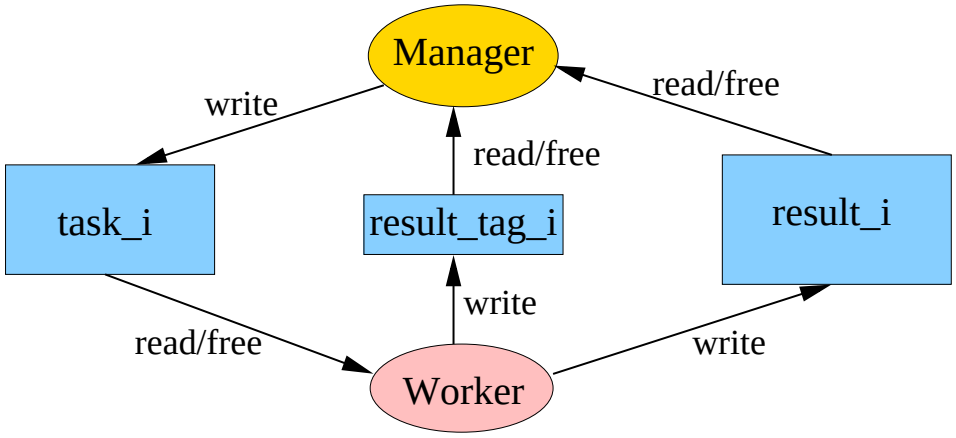
• We have two functional modules, which are stand-alone executable: `Manager` and `Worker`.

• A primer `run()` in ALDOR allows us to launch independent processes from the `Manager`.

• We rely on *shared memory segments*: we have developed a `SharedMemorySegment` domain in ALDOR.

• Multivariate polynomials are transfered as primitive array of machine integers via **Kronecker substitution**.

• Synchronization for data communication between `Manager` and `Worker`'s is controlled by a **mailbox protocol**.



# Experimentation

- Machine: Silky in Canada's Shared Hierarchical Academic Research Computing Network (SHARCNET), SGI Altix 3700 Bx2, 128 Itanium2 Processors (1.6GHz), 64-bit, 256 GB memory, 6 MB cache, SUSE Linux Enterprise Server 10 (ia64).
- Tests on 7 well-known problems:
  - Sequential runs **with and without regularized initials**
  - Parallel with ***Task Pool Dimension and Rank Guided Scheduling*** (TPDRG) in Triangularize, using 3 to 21 processors.
  - Parallel with ***Greedy Scheduling*** in Triangularize, using the number of processors
    * giving the best TPDRG time,
    * giving the best TPDRG time + 2 processors.
  - CPU time in **read and write** for data communication.

# Recall: features of the testing problem

| Sys | Name | $n$ | $d$ | $p$ | *Degrees* |
|---|---|---|---|---|---|
| 1 | eco6 | 6 | 3 | 105761 | [1,1,2,4,4,4] |
| 2 | eco7 | 7 | 3 | 387799 | [1,1,1,1,4,2,<br>4,4,4,4,4,2] |
| 3 | CassouNogues2 | 4 | 6 | 155317 | [8] |
| 4 | CassouNogues | 4 | 8 | 513899 | [8,8] |
| 5 | Nooburg4 | 4 | 3 | 7703 | [18,6,6,3,3,4,4,4,4,2,2,2,<br>2,2,2,2,1,1,1,1,1] |
| 6 | UteshevBikker | 4 | 3 | 7841 | [1,1,1,1,2,30] |
| 7 | Cohn2 | 4 | 6 | 188261 | [3,5,2,1,2,1,1,16,12,10,8,8,<br>4,6,4,4,4,4,2,1,1,1,1,1,1,1,<br>1,1,1,1,1,1,1] |

## Table 1: Wall time (s) for sequential (with vs without *regularized initial*) vs parallel

| Sys | *noregSeq* (s) | *regSeq* (s) | *slowBy* | #CPUs | *SigPara* (s) | *SPD* |
|---|---|---|---|---|---|---|
| 1 | 3.6 | 4.0 | 0.01 | 5 | 1.9 | 1.9 |
| 2 | 707.5 | 727.9 | 0.01 | 9 | 119.4 | 5.9 |
| 3 | 463.02 | 476.2 | 0.01 | 9 | 207.3 | 2.3 |
| 4 | 2132.9 | 2162.4 | 0.01 | 11 | 894.3 | 2.5 |
| 5 | 4.1 | 4.1 | 0.01 | 11 | 1.6 | 2.6 |
| 6 | 866.3 | 866.2 | - | 13 | 451.9 | 2.0 |
| 7 | 298.3 | 305.2 | 0.01 | 11 | 96.4 | 3.1 |

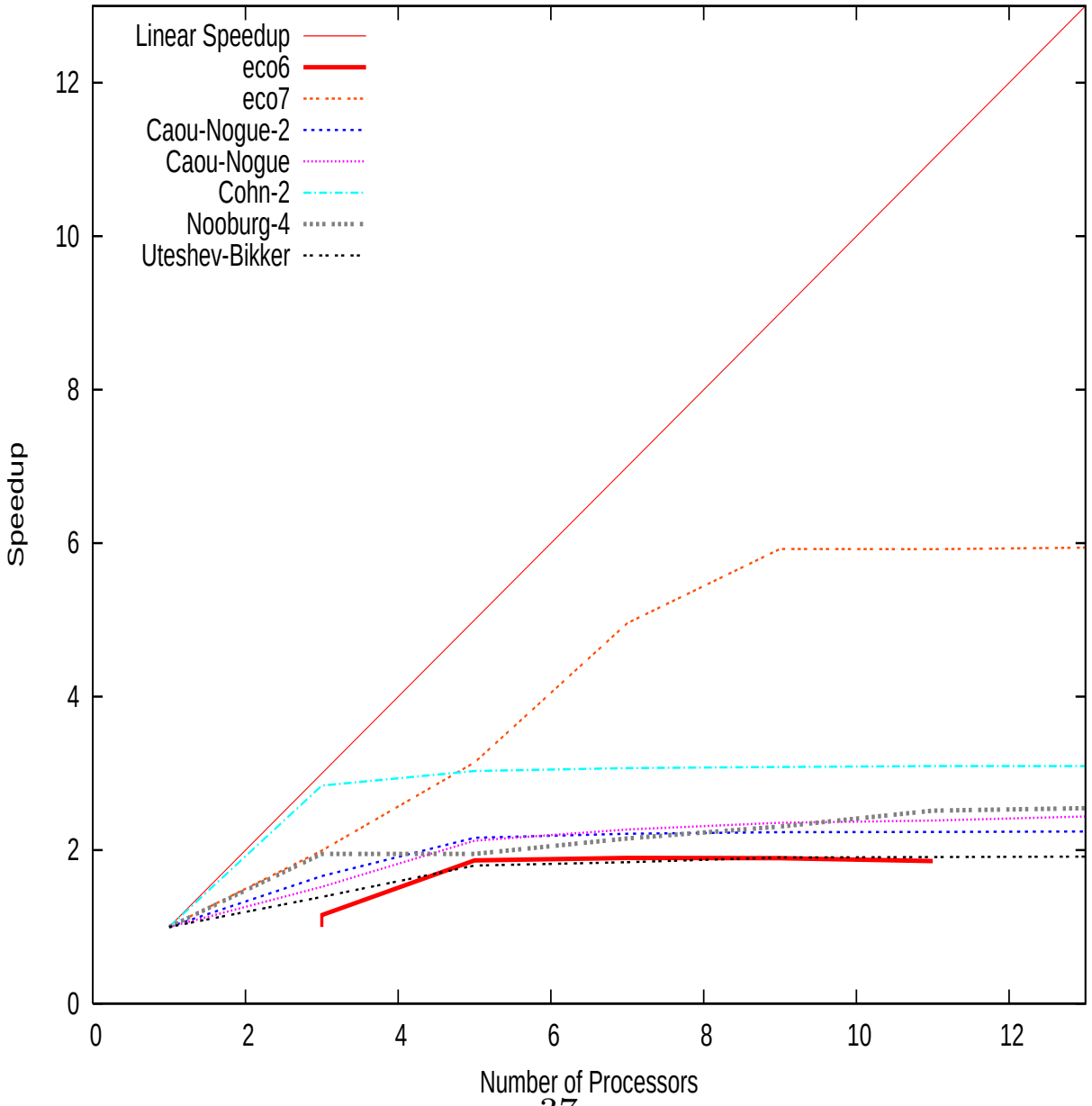
## Table 2: Parallel timing (s) vs #processor

| #CPUs | Sys 1 | Sys 2 | Sys 3 | Sys 4 | Sys 5 | Sys 6 | Sys 7 |
|---|---|---|---|---|---|---|---|
| 3 | 3.1 | 355.1 | 278.7 | 1401.4 | 2.1 | 622.9 | 105.0 |
| 5 | 1.9 | 225.3 | 214.2 | 1004.7 | 2.1 | 481.7 | 98.4 |
| 7 | 1.9 | 142.7 | 209.2 | 939.4 | 1.9 | 470.2 | 97.2 |
| 9 | 1.9 | 119.4 | 207.3 | 905.2 | 1.8 | 455.2 | 96.7 |
| 11 | 2.0 | 119.5 | 207.1 | 894.3 | 1.6 | 453.1 | 96.4 |
| 13 | - | 119.1 | 206.4 | 874.5 | 1.6 | 451.9 | 96.4 |
| 17 | - | 120.1 | 211.7 | 865.5 | 1.6 | 451.6 | 96.2 |
| 21 | - | 119.2 | - | 852.5 | - | 451.3 | 96.5 |

## Table 3: Speedup(s) vs #processor

| #CPUs | Sys 1 | Sys 2 | Sys 3 | Sys 4 | Sys 5 | Sys 6 | Sys 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1.15 | 1.99 | 1.66 | 1.52 | 1.95 | 1.39 | 2.84 |
| 5 | 1.87 | 3.14 | 2.16 | 2.12 | 1.95 | 1.80 | 3.03 |
| 7 | 1.89 | 4.96 | 2.21 | 2.27 | 2.15 | 1.84 | 3.07 |
| 9 | 1.89 | 5.92 | 2.23 | 2.36 | 2.31 | 1.90 | 3.09 |
| 11 | 1.86 | 5.92 | 2.24 | 2.39 | 2.51 | 1.91 | 3.10 |
| 13 | - | 5.94 | 2.24 | 2.44 | 2.55 | 1.92 | 3.09 |

# Speedup vs number of processors



## Table 4: Best TPDRG timing vs *Greedy* scheduling (s)

| System | *TPDRG* (best) | #CPUs (A) | Greedy (A) | #CPUs (B) | Greedy (B) |
|---|---|---|---|---|---|
| 1 | 1.911 | 7 | 1.792 | 9 | 1.778 |
| 2 | 119.093 | 13 | 120.505 | 15 | 120.516 |
| 3 | 206.375 | 13 | 213.206 | 15 | 213.354 |
| 4 | 852.488 | 20 | 896.787 | 22 | 939.618 |
| 5 | 1.611 | 13 | 1.631 | 15 | 1.632 |
| 6 | 451.357 | 20 | 500.504 | 22 | 469.354 |
| 7 | 96.203 | 17 | 100.779 | 19 | 96.169 |

All the examples but one are zero-dimensional (= finite number of solutions): a situation where redundant components are very rare.

So the *TPDRG* scheduling, which enhances the *solving by decreasing order of dimension*, is not a necessity.

However, it compares with the Greedy scheduling!

## Table 5: CPU time in read and write for data communication vs workers' total

| System | DataSize (#integer) | Read (ms) | Write (ms) | Workers (ms) | $Read + Write$ vs Total |
|---|---|---|---|---|---|
| 1 | 7717 | 80 | 56 | 1031 | 13.2% |
| 2 | 56689 | 383 | 172 | 174329 | 0.3% |
| 3 | 112802 | 4394 | 301 | 125830 | 3.7% |
| 4 | 430217 | 33760 | 426 | 585112 | 5.8% |
| 5 | 13307 | 123 | 55 | 1121 | 15.9% |
| 6 | 254145 | 9773 | 386 | 233582 | 4.4% |
| 7 | 77426 | 819 | 317 | 31858 | 3.6% |

# Conclusions

• By using modular methods, we have created opportunities for coarse grained **component-level** parallel solving of polynomial systems in $\mathbb{Q}[X]$

• To exploit these opportunities, we have transformed the Triade algorithm.

• We have strengthened its notion of a task and replaced the operation Decompose by SplitByHeight in order to

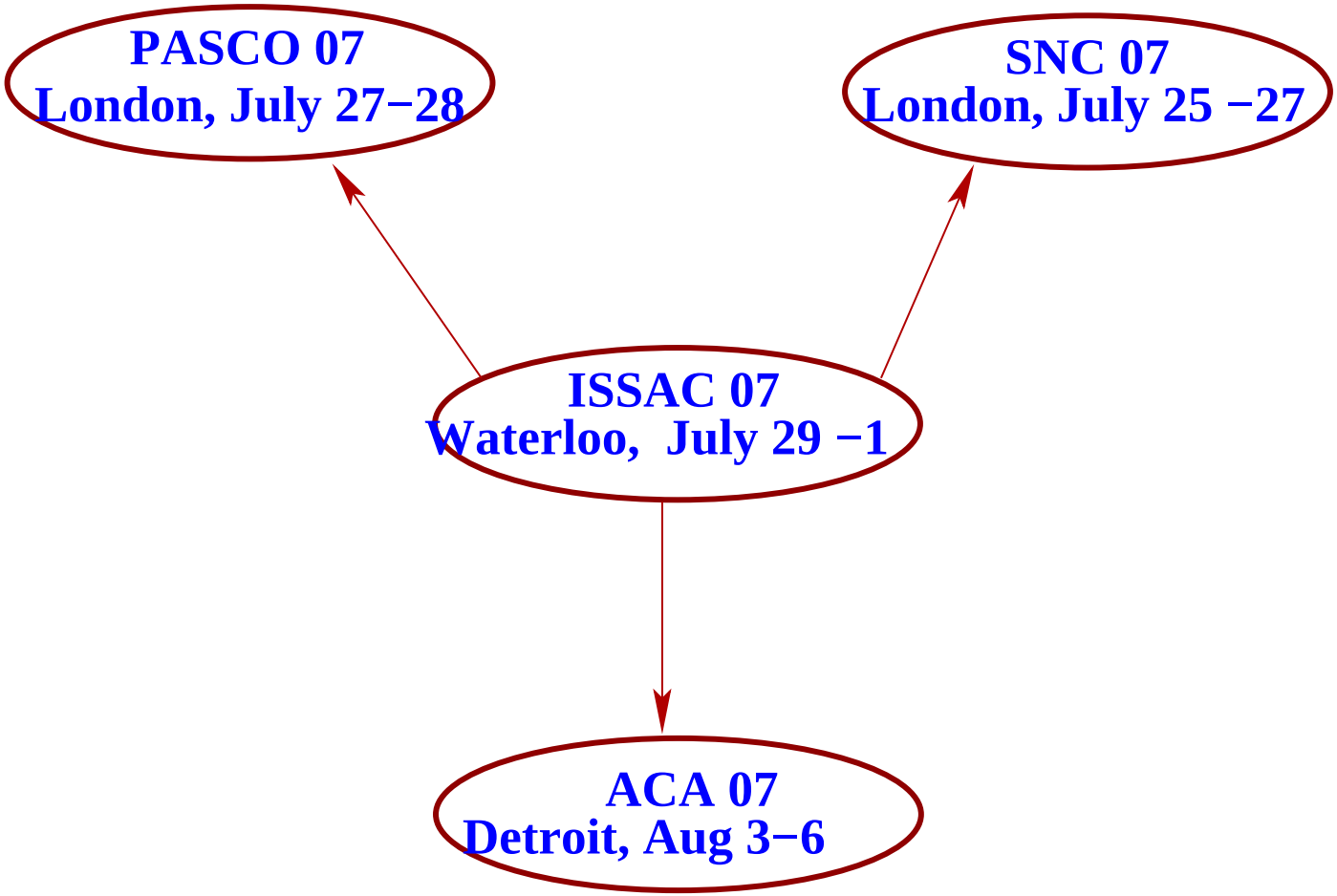
- reduce the depth of the task tree,
- create more work at each node,
- be able to estimate the cost of each task within each parallel step.

• We have realized a preliminary implementation in ALDOR using multi-processed parallelism with shared memory segment for data-communication.

# Toward Efficient Multi-level Parallelization

- We aim at developing a model for threads in Aldor to support parallelism for symbolic computations targeting SMP and multi-cores.
  - in particular, parametric types, such as polynomial data types, shall be properly treated.
- We aim at investigating multi-level parallel algorithms for triangular decompositions of polynomial systems.
  - coarse grained level for tasks to compute geometric of the solution sets.
  - medium/fine grained level for polynomial arithmetic such as multiplication, GCD/resultant, and factorization.
  - hence, to increase speed-up.

# Propaganda


PASCO 07
London, July 27−28
SNC 07
London, July 25 −27
ISSAC 07
Waterloo, July 29 −1
ACA 07
Detroit, Aug 3−6