On Solving a Problem in Algebraic Geometry by Cluster Computing*

Wolfgang Schreiner, Christian Mittermaier, and Franz Winkler

Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, Linz, Austria
FirstName.LastName@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at

Abstract. We describe a parallel solution to the problem of reliably plotting a plane algebraic curve. The sequential program is implemented in the software library CASA on top of the computer algebra system Maple. The parallel version is based on Distributed Maple, a parallel programming extension written in Java. We evaluate its performance on a cluster of workstations and PCs, on a massively parallel multiprocessor, and on a cluster that couples workstations and multiprocessor.

1 Introduction

We describe a parallel solution to the problem of reliably plotting a plane algebraic curve. Our starting point is the software library CASA (computer algebra software for constructive algebraic geometry) which has been developed on top of the computer algebra system Maple under the direction of the third author [2].

The basic objects of CASA are algebraic sets represented e.g. as systems of polynomial equations. Algebraic sets represented by bivariate polynomials model plane curves; an important problem is the *reliable* plotting of such curves. Numerical methods often yield qualitatively wrong solutions, i.e., plots where some "critical points" (e.g. singularities) are missing. For instance, the left diagram in Figure 1 shows a plot generated by the Maple function `implicitplot`. The numerical approximation fails to capture two singularities; even if we refine the resolution of the underlying grid, only one of the missing singularities emerges.

On the contrary, the CASA function `pacPlot` produces the diagram shown to the right. This plot of an algebraic curve a is generated by a hybrid combination of exact symbolic algorithms for the computation of all critical points and of fast numerical methods for the interpolation between them [3] (see Figure 1):

1. Compute the critical points of a in the y-direction and sort them according to their y-coordinates.
2. Intersect a with the horizontal lines that lie in the middle of the stripes determined by the y-coordinates of the critical points.
3. Trace a from each intersection point in both directions towards the border points of the stripe.

* Supported by the FWF projects P11160-TEC (HySaX) und SFB F013/F1304.

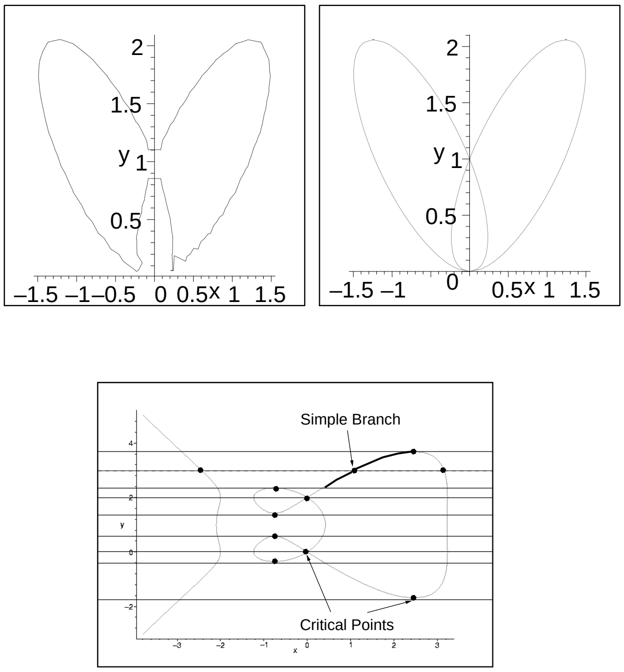

Fig. 1. Plotting an Algebraic Curve

`pacPlot` spends virtually all computation time in Step 1, a symbolic preprocessing phase that applies exact arithmetic to compute those points that determine the topological structure of a, essentially the singularities and the extrema in one coordinate direction. Figure 2 sketches the structure of the algorithm that computes a set of intervals that isolate these critical points.

The computation time of the critical point computation dominates the total plotting time and even on fast workstations only curves with total degree up to 8 or so can be plotted in a time that is acceptable for interactive usage. In the diploma thesis of the second author [1], this function has been parallelized.

2 The Parallelization Approaches

Investigating the dynamical behavior, it turns out that it does not pay off to parallelize the outermost loop which iterates over all systems in S: typically there are not more than two systems and only one system is not immediately recognized as trivial. Therefore we apply parallelism on several other levels:

1. parallel resultant computation,
2. parallel real root isolation,

$\boldsymbol{P}$ **:= criticalPoints($\boldsymbol{a(x, y)}$):**
$P := \emptyset$
$S := \{(p(y), q(x, y)) \mid$
$\exists p'(y) : (p'(y), q(x, y)) \in \text{triangulize}(a(x, y), \frac{\partial a(x,y)}{\partial x}), p(y) \in \text{factor}(p'(y))\}$
for $(p(y), q(x, y)) \in S$ **do**
$r(x) := \text{resultant}_x(p(y),\ q(x, y))$
$X := \text{realroot}(r(x))$
$Y := \text{realroot}(p(y))$
for $x \in X$ **do**
$q'(y) := \text{squarefree}(q(x, y),\ x.0,\ p(y))$
$q''(y) := \text{squarefree}(q(x, y),\ x.1,\ p(y))$
for $y \in Y$ **do**
if $\text{test}(q'(y),\ q''(y),\ y,\ p(y))$ **then** $P := P \cup \{(x, y)\}$
end
end
end
end

Fig. 2. Computation of Critical Points

3. parallel solution test,
4. parallel interval refinement.

Parallel Resultant Computation A single resultant computation may take a good deal of the computation time of the whole algorithm. Our parallelization approach applies a modular method to compute the resultant in multiple homomorphic images of the domain and to combine the homomorphic results to the result in the original domain. We thus get a divide and conquer structure where both the divide phase (the modular resultant computation) and the conquer phase (the "Chinese Remaindering" computation) can be parallelized [4].

Parallel Real Root Isolation The isolation of the real roots of the resultant by Uspensky's method, a recursive divide and conquer search algorithm, may also take a significant computation time. A naive parallelization of this algorithm typically yields poor speedups due to the narrowness of the highly unbalanced search tree. A parallelization of the arithmetic in each recursive invocation step is only feasible for tightly coupled multiprocessors. Therefore our approach is to broaden and to flatten the search tree (to a certain extent) by applying speculative parallelism [1].

Parallel Solution Test and Interval Refinement The tests which of the candidates (x, y) are indeed solutions of the given system can be performed in parallel in a straight-forward fashion. Likewise, we can apparently refine all isolating intervals in parallel to the desired accuracy [1].

We have implemented all four ideas on the basis of Distributed Maple.

3 Distributed Maple

Distributed Maple is an environment for writing parallel programs on the basis of the computer algebra system Maple developed by the first author [4]. It allows to create tasks and to execute them by Maple kernels running on various machines of a network. Each node of a session comprises two components (see Figure 3):

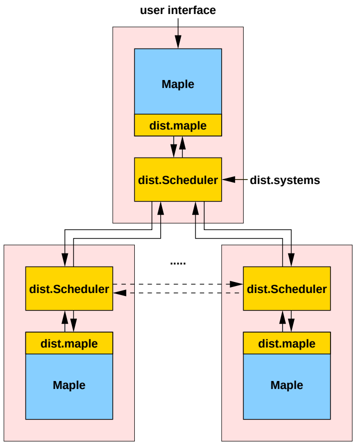

Fig. 3. Distributed Maple Architecture

Scheduler The Java program `dist.Scheduler` coordinates the node interaction. The scheduler process attached to the frontend kernel starts instances of the scheduler on other machines and communicates with them via sockets.

Maple Interface The program `dist.maple` running on every Maple kernel implements the interface between kernel and scheduler. Both components use pipes to exchange messages (which may embed any Maple objects).

The user interacts with Distributed Maple via the Maple frontend by a number of programming commands, in particular:

`dist[start](`*f*`, `*a*`, ...)` creates a task evaluating *f(a, ...)* and returns a task reference *t*. Tasks may create other tasks and arbitrary Maple objects (including task references) may be passed as arguments and returned as results.

`dist[wait](`*t*`)` blocks the execution of the current task until the task represented by *t* has terminated and returns its result. Multiple tasks may independently wait for and retrieve the result of the same task *t*.

This parallel programming model is based on para-functional principles which is sufficient for many kinds of computer algebra algorithms. In addition, the environment supports a non-deterministic form of task synchronization for speculative parallelism and shared data objects which allow tasks to communicate by side effects on a global store.

4 Experimental Results

We have benchmarked the parallel critical point computation in three environments with four random algebraic curves for which the sequential program takes 6870s, 470s, 155s, and 11748s (measured on a PIII@450MHz PC):

- The cluster of our institute which comprises 4 Silicon Graphics Octanes (2 R10000@250Mhz each) and 16 Linux PCs (various Pentium processors),
- a Silicon Graphics Origin multiprocessor (64 R12000@300 Mhz, 24 processors used) located at the university campus,
- a mixed configuration consisting of our 4 dual-processor Octanes and 16 processors of the Origin multiprocessor connected via an ATM line.

The raw computing power of each environment is 18.3, 17.1, and 18.7 (multiplication factors compared with a single PIII@450MHz processor); the benchmark results are as follows:

Execution Times (s)							
Example	Environment	1	2	4	8	16	24
1 (6870s)	Cluster	14992	8035	2732	1186	552	488
	Origin	7290	4217	1789	872	446	513
	Mixed	14992	8035	2732	1368	597	519
2 (470s)	Cluster	810	648	328	173	95	108
	Origin	667	541	297	166	112	116
	Mixed	810	648	328	210	116	127
3 (155s)	Cluster	267	191	112	67	46	45
	Origin	196	147	90	63	54	54
	Mixed	267	191	112	74	58	56
4 (11748s)	Cluster	25178	15559	6820	3562	1915	1563
	Origin	13397	8223	4009	2281	1726	1420
	Mixed	25178	15559	6820	4042	2004	1599

A detailed analysis of the results and our conclusions are given in a longer version of this paper at `http://www.risc.uni-linz.ac.at/software/distmaple`.

References

[1] C. Mittermaier. Parallel Algorithms in Constructive Algebraic Geometry. Master's thesis, Johannes Kepler University, Linz, Austria, 2000.

[2] M. Mnuk and F. Winkler. CASA - A System for Computer Aided Constructive Algebraic Geometry. In *DISCO'96 — International Symposium on the Design and Implementation of Symbolic Computation Systems*, volume 1128 of *LNCS*, pages 297–307, Karsruhe, Germany, 1996. Springer, Berlin.

[3] T. Q. Nam. Extended Newton's Method for Finding the Roots of an Arbitrary Systemof Equations and its Applications. In *IASTED'94: 12th International Conference on Applied Informatics*, Annecey, France, 1994.

[4] W. Schreiner. Developing a Distributed System for Algebraic Geometry. In B. H. Topping, editor, *EURO-CM-PAR'99 Third Euro-conference on Parallel and Distributed Computing for Computational Mechanics*, pages 137–146, Weimar, Germany, March 20-25, 1999. Civil-Comp Press, Edinburgh.