Combining Parallel and Distributed Search in Automated Equational Deduction*

Carsten Sinz[1], Jörg Denzinger[2], Jürgen Avenhaus[3], and Wolfgang Küchlin[1]

[1] Symbolic Computation Group, WSI, University of Tübingen, Germany
[2] Computer Science Department, University of Calgary, Canada
[3] Fachbereich Informatik, Universität Kaiserslautern, Germany

Abstract. We present an automated deduction system for equational reasoning combining two different parallelization/distribution schemes: Strategy-compliant parallelization on the level of individual deduction steps (PaReDuX) and distributed cooperation of multiple agents with different search strategies (TEAMWORK). In our experiments we mainly observed a multiplication of the speed-ups of each approach in our combined system.

1 Introduction

Parallelization in the realm of automated deduction has a long and rich history (see, e.g., [SS93, BH94] for surveys), although the task of finding a suitable parallel algorithm is often not easy to accomplish. Usually a static problem decomposition is not feasible, as runtimes of sub-problems (generated by static splitting) can hardly be predicted, and therefore the computational capacity cannot be utilized adequately. Moreover, we are frequently faced with a high degree of data dependencies and with the fact that prover programs are very complex. Both aspects make partitioning of the problem difficult, and are furthermore intensified when data structures in the prover's core have been optimized. On the other hand, speeding up the proof process is highly desirable, as generation of automatic proofs is computationally very expensive.

Because of the aforementioned difficulties, and as most provers use heuristics to make their decisions, parallelization is most easily obtained by running different provers or provers with different heuristics competitively in parallel. In the fortunate case that one of the provers possesses the right settings and heuristics, a proof can possibly be found in reasonable time. This approach can be improved upon by using cooperation in order to exchange valuable information. Another approach to parallelization is to speed up the basic steps of the prover by performing them in parallel, or even splitting them up and executing the atoms they are composed of concurrently. The combination of both approaches can be expected to deliver even better accelerations than the single approaches themselves. However, this combination is seldom put into practice, partly because the success of such a combination is not obvious a priori, partly because implementing such a system is a non-trivial task. Among the reasons for nevertheless trying to combine both methods is—besides the hope for increased speed-ups—the fact that

* This work was supported by DFG under grants Av 10/7-1 and Ku 966/4-1.

such a two-tired parallel system better reflects the actual hardware commonly used for parallel automatic theorem proving: clusters of (symmetric) multi-processor computers (SMPs). Each SMP perfectly matches the demands of the second of the aforementioned parallelization approaches of executing the basic steps in parallel, as common data structures can be held in shared memory and allow very fast data access. Clusters of SMPs connected by networks (i.e. distributed memory multi-processors) better reflect the first approach of cooperating agents, as they have no common memory to store data used by all agents, which matches the fact that information has to be exchanged less frequently and in smaller quantities in the cooperation approach.

Our system TEAMWORK-PaReDuX is a combination of two already existing systems for parallel equational reasoning: PaReDuX [BGKW98] implements a strategy-compliant parallelization on the level of individual deduction steps, TEAMWORK is a framework for distributed cooperating agents that has been applied to equational theorem proving in the DISCOUNT system [DKS97].

2 Theoretical Background

2.1 Unfailing Completion.

We will now give a brief introduction to term rewriting systems and describe the unfailing completion procedure. For a thorough introduction we refer the reader to the work of Dershowitz and Jouannaud [DJ90].

The basic objects we are dealing with are *equations* $s \leftrightarrow t$ and *rewrite rules* $s \rightarrow t$ (directed equations), both of which are built out of *terms* s and t. The only inference rule is to replace equals by equals, where rewrite rules are only applied in the indicated direction. For a definition of *term and equation instances*, *reducibility* and *normal forms* we kindly ask the reader—due to space limitations—to look up in the aforementioned work. Given a reduction ordering $\succ$ (well-founded, closed under substitution and replacement of subterms), the notion of reducibility can be extended to equations: for an equation $e = l \leftrightarrow r$ we denote by $\{e\}_\succ$ the set of all *oriented instances* of e, i.e. the set of rules $e' = l' \rightarrow r'$ such that $l' \succ r'$ and e' is an instance of e. We denote by $\rightarrow^*_\mathcal{R}$ resp. $\leftrightarrow_\mathcal{R}$ the transitive-reflexive resp. symmetric closure of the reduction relation $\rightarrow_\mathcal{R}$.

Unfailing completion [BDP89] is an extension of the reasoning procedure of Knuth and Bendix [KB70] for equational theories. Starting with a given equational theory represented by a set $\mathcal{P}$ of equations, a reduction ordering $\succ$, and a goal $a \leftrightarrow b$ to be proved, the unfailing completion procedure successively transforms $\mathcal{P}$ into two sets $\mathcal{R}$ and $\mathcal{E}$ of rules resp. equations such that the induced reduction relation $\rightarrow_{\mathcal{R} \cup \mathcal{E}_\succ}$ is well-founded and confluent on terms containing no variables. Moreover, to ensure correctness, equality as defined by $\rightarrow_{\mathcal{R} \cup \mathcal{E}_\succ}$ must imply equality by $\mathcal{P}$. Then the goal $a \leftrightarrow b$ is proved as soon as we have found a c such that $a \rightarrow^*_{\mathcal{R} \cup \mathcal{E}_\succ} c \leftarrow^*_{\mathcal{R} \cup \mathcal{E}_\succ} b$. This transformation of $\mathcal{P}$ is accomplished by repeatedly applying an inference rule to a triple $(\mathcal{P}, \mathcal{E}, \mathcal{R})$ producing a successor triple $(\mathcal{P}', \mathcal{E}', \mathcal{R}')$, starting with $\mathcal{E} = \mathcal{R} = \emptyset$. The inference rule is selected out of a set of eight different ones, as described in [BGKW98]. One of the inference rules allows derivation of new equations called *critical pairs* [KB70] from the set $\mathcal{R} \cup \mathcal{E}_\succ$. These critical pairs $s \leftrightarrow t$ all possess the property that

$s \leftarrow_{\mathcal{R}\cup\mathcal{E}_\succ} u \rightarrow_{\mathcal{R}\cup\mathcal{E}_\succ} t$ for some term u, and thus compensate the weakening effect obtained by the ordering of equations. Let us illustrate this by an example. Assume $\mathcal{R} = \{f(f(x,y),z) \rightarrow f(x,f(y,z)),\ f(i(x),x) \rightarrow n\}$ and $\mathcal{E} = \emptyset$. Furthermore, let $u = f(f(i(x'),x'),y'),\ s = f(n,y')$ and $t = f(i(x'),f(x',y'))$. Then $s \leftarrow_{\mathcal{R}} u \rightarrow_{\mathcal{R}} t$, and $s \leftrightarrow t$ is a critical pair that partially compensates the weakening caused by ordering the associativity rule.

UFC($\mathcal{P}$, $\succ$, $a \leftrightarrow b$)

[$\mathcal{P}$: equational theory, $\succ$: term ordering, $a \leftrightarrow b$: equation to be proved]

(1) [Init] $\mathcal{R} := \emptyset;\ \mathcal{E} := \emptyset$.

(2) [Selection] Choose $n = (s \leftrightarrow t) \in \mathcal{P}$.

(3) [Orient/Unfail] Remove n from $\mathcal{P}$;
If $s \succ t$ or $t \succ s$ add n as $l \rightarrow r$ where $l \succ r$ to $\mathcal{R}$, otherwise as $s \leftrightarrow t$ to $\mathcal{E}$.

(4) [Collapse$_\mathcal{E}$]
for each $u \leftrightarrow v \in \mathcal{E}$ do:
if $u \rightarrow_{\{l\rightarrow r\}} u'$ resp. $u \rightarrow_{\{s\leftrightarrow t\}_\succ} u'$ remove $u \leftrightarrow v$ from $\mathcal{E}$, add $u' \leftrightarrow v$ to $\mathcal{P}$.

(5) [Collapse$_\mathcal{R}$]
for each $l' \rightarrow r' \in \mathcal{R}$ do:
if $l' \rightarrow_{\{l\rightarrow r\}} l''$ resp. $l' \rightarrow_{\{s\leftrightarrow t\}_\succ} l''$ remove $l' \rightarrow r'$ from $\mathcal{R}$, add $l'' \leftrightarrow r$ to $\mathcal{P}$.

(6) [Compose]
for each $l' \rightarrow r' \in \mathcal{R}$ do:
if r' is reducible w.r.t. $l \rightarrow r$ resp. $\{s \leftrightarrow t\}_\succ$:
remove $l' \rightarrow r'$ from $\mathcal{R}$;
normalize r' with respect to $\mathcal{R} \cup \mathcal{E}_\succ$ resulting in r'';
add $l' \rightarrow r''$ to $\mathcal{R}$.

(7) [Simplify & Delete]
for each $u \leftrightarrow v \in \mathcal{P}$ do:
if $u \leftrightarrow v$ is reducible w.r.t. $l \rightarrow r$ resp. $\{s \leftrightarrow t\}_\succ$:
normalize $u \leftrightarrow v$ w.r.t. $\mathcal{R} \cup \mathcal{E}_\succ$;
remove $u \leftrightarrow v$ from $\mathcal{P}$ if it is an instance of an equation in $\mathcal{E}$ or if $u = v$.

(8) [Deduce]
compute $mathcalP' = CP_\succ(\mathcal{R} \cup \mathcal{E}, n)$[1];
for each $p \in \mathcal{P}'$ do: normalize p w.r.t. $\mathcal{R} \cup \mathcal{E}_\succ$.

(9) [Merge & Simplify Goal]
add all $p \in \mathcal{P}'$ to $\mathcal{P}$;
normalize $a \leftrightarrow b$ w.r.t. $\mathcal{R} \cup \mathcal{E}_\succ$;
if $a = b$ output "PROVED" and exit, otherwise go to step (2).

Fig. 1. Sequential Completion Strategy for Unfailing Completion: Algorithm UFC

The inference rules for unfailing completion possess a high degree of indeterminism in that neither the order of rule application nor the selection of the equation $s \leftrightarrow t \in \mathcal{P}$ to be considered next is specified. In an effective implementation a *completion strategy* is employed that reduces the indeterminism and additionally ensures completeness.

[1] $CP_\succ(E,n)$ denotes the set of critical pairs between rules $r \in E_\succ$ and $r' \in \{n\}_\succ$.

Huet presented a completion strategy for the ordinary Knuth-Bendix completion algorithm and proved its completeness [Hue81]. In the TEAMWORK-PaReDuX system we adopted a modified version of Huet's completion strategy for unfailing completion [MGB95]. The algorithm UFC we implemented is shown in Fig. 1. We denote the loop comprising steps (2) to (9) of this algorithm as the *base loop* of UFC.

Algorithm UFC still leaves some degree of indeterminism: Which equation n shall we choose in step (2)? In which order should we apply rules from $R \cup E_{\succ}$ to normalize terms in steps (6)-(9)? Although the latter question has also influence on the effectiveness of the completion procedure, finding a good answer to the former question is crucial for a good performance of the algorithm.

2.2 Parallel Search

We will now briefly describe some of the possibilities to parallelize the UFC algorithm, and classify the different approaches depending on the size of the portions of code to be executed in parallel (*grain size*). On the one end of the scale we have as a very coarse grained method the concurrent execution of several instances of the whole sequential UFC algorithm (by so-called *agents*) with different equation selection strategies applied in step (2). This setting of competing agents can be improved by allowing cooperation by exchanging information between agents. TEAMWORK is a representative of this coarse grained parallelization method. On the other end of the scale, realized, e.g., by PaReDuX, we can perceive fine-grained parallel systems in which each step of the algorithm is split, and the resulting portions are executed by individual processors in parallel.

3 The TEAMWORK-PaReDuX System

The TEAMWORK-PaReDuX system is a combination of the parallel UFC algorithm of the PaReDuX-suite and a distribution approach using TEAMWORK's cooperating agents. Both the PaReDuX system and DISCOUNT, a system implementing the TEAMWORK method, have already shown their success in speeding up the unfailing completion procedure [BGK95, DKS97]. In TEAMWORK tasks are working independently over a fixed time interval of several seconds or minutes interspersed by communication periods; PaReDuX, on the other hand, executes individual steps (or groups of steps) of the algorithm UFC in parallel. The different scale on which parallelization is performed in these two systems makes integration promising.

3.1 PaReDuX

The PaReDuX system [BGK95] is a strategy-compliant parallel implementation of algorithm UFC. Strategy compliance means that sequential and parallel program runs are equivalent in the sense that there is a direct correspondence between the program states of the sequential and the parallel version at each passage of the beginning of the base loop. This implies that measurements are reliable, timings do not change when an experiment is repeated, and therefore program development and improvement is facilitated. On the other hand no super-linear speed-ups can be achieved with this approach.

Parallelization within PaReDuX is achieved in two ways. First, by concurrent execution of individual steps of the algorithm, taking data dependencies into account. And second, by executing the *for each*-part of appropriate steps in parallel.

Concurrent Execution of Several Steps. A careful analysis of data-dependencies between the eight steps of the base loop reveals that steps (4) and (5) can be executed in parallel, provided that the write-access to $\mathcal{P}$ is postponed. In this case step (4) only reads and modifies $\mathcal{E}$, whereas in step (5) $\mathcal{R}$ is affected. Similarly, we find that steps (7) and (8) can be executed simultaneously, as in step (7) only $\mathcal{P}$ is modified, which is not read or written in step (8). The other way round, no data used by step (7) is modified in step (8). It turned out that steps (2) to (6) in most cases consume accumulated less than 5% of the runtime [BGKW98]. Therefore in the UFC implementation of PaReDuX only steps (7) and (8) are performed concurrently.

Sub-Division of Individual Algorithm Steps. The rules and equations of $\mathcal{P}$, $\mathcal{E}$ and $\mathcal{R}$ are stored in lists within PaReDuX. In algorithm step (7) the equations of $\mathcal{P}$ are normalized with respect to $\rightarrow_{R \cup E_\succ}$, which is relatively time consuming. Therefore in this step the list $\mathcal{P}$ is recursively split, the resulting sublists are processed concurrently by different processors, and then the result is joined into the updated list for $\mathcal{P}$. This divide-and-conquer-style processing is well supported by the *thread-* or *fork/join*-paradigm, and therefore PaReDuX has adopted this parallelization scheme here. Similarly, the computationally intensive operation of critical pair computation $CP_\succ(\mathcal{R} \cup \mathcal{E}, n)$ in step (8) is performed in parallel by splitting the list representing $\mathcal{R} \cup \mathcal{E}$. The normalization of the resulting equations in $\mathcal{P}'$ is also done by parallel tasks, according to step (7).

To make the fork/join-paradigm available for the special demands of PaReDuX, virtual symbolic threads (*VS-threads*) [KW92] have been employed. Symbolic threads extend the usual (POSIX-)threads by automatic memory management, garbage collection and list processing. Virtual threads decide on their own whether a *fork* is executed as a normal function call or mapped to a kernel thread leading to real concurrency. This allows handling of large sets of (logical) threads without running into possible limitations of the underlying operating system. Moreover, the PaReDuX implementation uses different grain size parameters (concerning list lengths for the divide-and-conquer style list processing) that serve as threshold values below which thread forks are executed as direct function calls. For our experiments we used the default settings for these parameters. An in-depth description of PaReDuX can be found in [BGKW98].

3.2 TEAMWORK

The general idea of the TEAMWORK approach is to have different programs called *experts* working cooperatively on the same problem. All experts work independently on a common problem description during the so-called *working phases*, which are interrupted by *team meetings*. Team meetings allow the exchange of newly gathered information or the reconfiguration of the whole team by taking out unsuccessful experts and replacing them by hopefully better ones. Each expert is assigned a *referee* which is responsible for assessing the expert's work and judging which part of the work could be

of interest to the other experts. One of the experts is called *supervisor* and has a distinguished role to play: it builds new problem descriptions at the end of a team meeting, it chooses experts for the next working phase and it determines the working phase's duration.

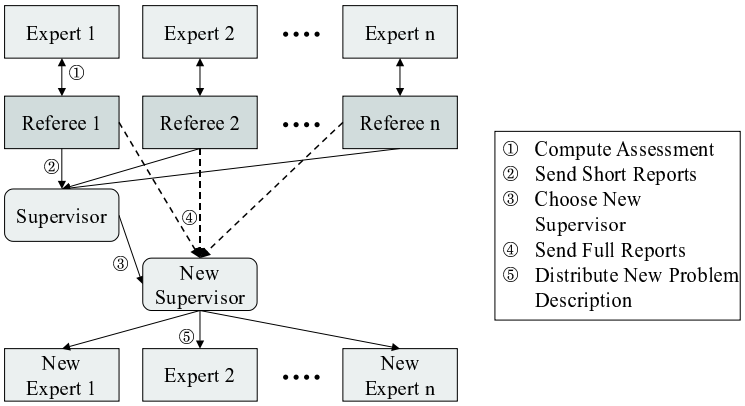

Fig. 2. Actions during a Team Meeting.

The actions taking place during a team meeting are as follows (see Fig. 2): At first, each referee computes an assessment value for its expert (1), which are sent as *short reports* to the supervisor (2). The supervisor designates its own successor, which is usually the best expert of the last working phase, and then passes control to the new supervisor (3), which in turn requests full reports—consisting of a selection of the best results of the last cycle of an expert—from the referees (4). Then the new supervisor integrates these results into its own problem description, and distributes this description as the new problem to a set of freshly chosen experts (5).

When applying the TEAMWORK method to unfailing completion, all experts execute the same basic UFC algorithm, but with different equation selection heuristics applied in step (2). During a team meeting the success of each expert is computed, and the best rules and equations are integrated into the new problem description.

3.3 Combined System

To combine the PaReDuX system with the TEAMWORK method, we take the approach outlined in the last section, but replace the sequential experts by experts executing the PaReDuX-algorithm.

An outline of the architecture of the combined TEAMWORK-PaReDuX system is shown in Fig. 3. As mentioned before, the PaReDuX system is based on VS-threads, which on the one hand build the link to the operating system's threads and on the other hand provide a multi-threaded memory management and list processing functionality. Two or more PaReDuX instances are linked together by TEAMWORK's `TWlib` [DL96]. `TWlib` provides facilities for converting a standard program into a TEAMWORK expert,

for communication between experts, and support for all actions performed during a team meeting.

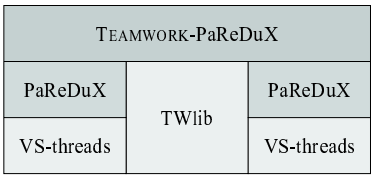

Fig. 3. Architecture of the TEAMWORK-PaReDuX System.

During the working phases all experts run the PaReDuX algorithm, but with different equation selection heuristics. We have implemented six heuristics in our system: *MaxWeight* [DF98], *AddWeight*$_1^2$ [DF98], *OccNest* [AA90, DF98, BGKW98], *OccNest(Mod)* [BGKW98], *GoalInCP* [DF94], and *CPInGoal* [DF94]. Instead of explaining them in detail, we give a rough picture of some ideas of these heuristics, and refer the interested reader to the quoted literature. A common idea is to give precedence to small equations, i.e. terms with a low number of symbol occurrences. This is motivated by the fact, that such equations are statistically applicable in more situations. *MaxWeight* and *AddWeight*$_1^2$ fall into this class of heuristics. The other four strategies are goal sensitive, i.e. they take the equation to be proved into consideration. *OccNest* and *OccNest(Mod)* compare the structure (number of occurrences of each function symbol, functional nesting depth) of an equation with the goal, and choose the best matching one. The remaining goal oriented strategies *GoalInCP* and *CPInGoal* perform an even more sophisticated similarity comparison.

For the assessment taking place at the beginning of the team meeting, we have uniformly used one type of referee, called *statistical referee*. The statistical referee computes the success of an expert using the parameterized assessment function

$$p_0 \cdot |\mathcal{R}| + p_1 \cdot |\mathcal{E}| + p_2 \cdot |\mathcal{P}| + p_3 \cdot \#\text{RED},$$

where #RED denotes the total number of reductions performed with an equation of $\mathcal{E}$ or a rule of $\mathcal{R}$ during the last working phase. We set the vector p to $(-2, -10, -6, 10)$, a value already employed in the DISCOUNT system. The expert with the highest assessment value is selected as the new supervisor and requests full reports from all other experts. A full report consists of up to ten of the best newly derived rules or equations (from $\mathcal{R}$ or $\mathcal{E}$) of each expert based on the evaluation of rules shown below. The transferred equations must all be generated during the last working cycle, and those with the highest values are selected. The evaluation is uniformly computed by the parameterized function

$$q_0 \cdot \#\text{MATCH} + q_1 \cdot \#\text{COLLAPSE} + q_2 \cdot \#\text{COMPOSE} + q_3 \cdot \#\text{GOAL},$$

where #MATCH, #COLLAPSE, #COMPOSE and #GOAL denote statistical information concerning the applicability of the rule or equation n during the last working cycle:

#MATCH denotes the total number of reductions that have been performed with n, #COLLAPSE resp. #COMPOSE the number of rules and equations that have been collapsed resp. composed employing n, and #GOAL indicates the number of reductions on the goal equation that have been performed using n. The value for the vector q was permanently set to $(1, 10, 10, 100)$ in our experiments.

The supervisor then constructs a new, improved problem description containing its own problem representation, augmented by the selected equations and rules of the other experts. Those are added to the supervisor's set $\mathcal{P}$, and the new problem description is redistributed to all other experts, which start their next working cycle with the new data $(\mathcal{P}, \mathcal{E}, \mathcal{R})$ in the base loop of algorithm UFC, i.e. in step (2).

Of course, the choice of the parameter vectors p and q can have a considerable influence on the behavior of the system as a whole. So far, however, we have not conducted experiments to empirically determine the best values for p and q, but rather adopted the values from already existing implementations like DISCOUNT and PaReDuX.

4 Experimental Results

Our experiments to determine the speed-up factors that can be obtained with the combined system as compared to the individual approaches were performed on two Sun Ultra E450 each containing 4 400MHz UltraSparcII processors and 1 GB of main memory, running under Solaris 7. For each problem instance we conducted four program runs:

Sequential: One machine was running as the sole expert using only one processor. We report values for the equation selection heuristic that delivered the best results.

Parallel: We used all four processors of one E450 within one TEAMWORK expert, fixed the equation selection heuristic to the same strategy as in the sequential case.

Distributed: Both E450 were employed, but only one processor on each machine was used. We have chosen the two equation selection heuristics coming up with the best results in the sequential program runs for our experts.

Parallel & Distributed: We used all eight processors of our two SMPs. Two experts were performing the parallel UFC algorithm, leaving the heuristics of the experts the same as in the distributed setting.

We have selected theorems from the TPTP problem library [SS98], version 2.2.1, which is a common reference problem collection for automated theorem prover comparisons. The problems are taken from the domains Boolean Logic (BOO), Combinatory Logic (COL), Group Theory (GRP) and Robbins Algebra (ROB). Moreover, we added the problems LUKA3 and RA007. LUKA3 [DF94] uses the axiomatization of propositional logic by Łukasiewicz and proves $(\neg x \rightarrow \neg y) \rightarrow (y \rightarrow x)$. RA007 proves that $A \leq A \odot 1$ holds in all Relation Algebras using the Tarski/Givant axiomatization [TG87]. We used a fixed cycle time of ten seconds for the length of the working phase, and did not make use of the reactive planning possibilities of TEAMWORK, i.e. we did not change the team setup during a program run.

The results of our experiments are shown in Table 1. The given runtimes are wall-clock times in seconds, where we have set a time limit of one hour for all program executions. An empty entry in the table indicates that the problem could not be solved

Table 1. Experimental Results

Problem	Runtimes				Speed-up			
	Seq.	**Par.**	**Dist.**	**Par. & Dist.**	**Par.**	**Dist.**	**Par. × Dist.**	**Par. & Dist.**
BOO002-2	411.95	154.02	194.80	76.19	2.67	2.11	5.63	5.41
BOO007-4	-	-	554.31	163.68	-	-	-	-
BOO022-1	148.50	56.34	226.44	57.08	2.64	0.66	1.74	2.60
COL003-12	364.89	111.21	262.08	86.56	3.28	1.39	4.60	4.21
GRP002-4	205.93	83.36	14.00	6.22	2.47	14.71	36.33	33.11
GRP119-1	867.70	258.45	423.86	99.81	3.36	2.07	6.96	8.69
GRP122-1	752.73	218.73	155.31	49.23	3.32	4.85	16.10	15.29
GRP175-3	1422.15	555.19	63.70	29.04	2.56	22.33	57.17	48.97
GRP175-4	464.33	163.02	243.50	87.61	2.85	1.91	5.44	5.30
LUKA3	-	-	135.73	33.17	-	-	-	-
RA007	-	-	67.22	25.58	-	-	-	-
ROB004-1	1683.82	670.16	40.73	14.60	2.51	41.34	103.76	115.33

within the time limit. The speed-up factor in column **Par. × Dist.** represents the theoretical outcome of combining the parallelization and distribution method, namely by multiplication of the speed-ups of the individual approaches (the two columns to the left). The rightmost column **Par. & Dist.** reports the empirically determined speed-up of our combined TEAMWORK-PaReDuX system on eight processors.

The speed-ups obtained by parallelization alone cover a range between mediocre speed-ups (2.47) and the theoretically best possible factor of 4 (with 3.36 being the best actually achieved). In the distribution approach synergetic effects due to cooperation of multiple heuristics become apparent and show up as super-linear speed-ups growing to factors of over 41 on only two processors. But in some cases proof search is even slowed down (BOO022-1), because the exchanged equations confuse the experts. The combination of both systems reaches for several examples nearly the multiplicative optimum, and four examples were even better than the predicted theoretical outcome would suggest. We explain the observation of these over-proportional speed-ups by the fact that the TEAMWORK approach is quite sensitive to timing changes. Generating an important result just before a team meeting will send it to the other experts nearly immediately, while generating it just after a meeting means that the other experts have to wait nearly ten seconds for it. Thus, the exact time when a team meeting takes place can considerably effect the overall runtime required for a proof.

5 Conclusion and Future Work

Our experiments have shown that both methods can be successfully united into a single system, with speed-up factors almost multiplying compared to the individual systems. TEAMWORK-PaReDuX has the advantage of reflecting the widespread two-tired hardware architecture of clusters of multi-processor computers in its software architecture. Thus it can make optimal use of today's parallel hardware. Although we presented the combination of two parallelization schemes only for the field of equational deduction,

we expect it to be applicable in neighboring areas, such as Gröbner base computation or theorem proving by resolution, as well. It remains to be shown whether or not our approach is general enough to be a universal scheme for an even broader class of search problems as they occur, e.g., in the realm of artificial intelligence.

Finally, we would like to thank Jochen Walter for his help on implementing the TEAMWORK-PaReDuX system.

References

[AA90] S. Anantharaman and N. Andrianarievelo. Heuristical criteria in refutational theorem proving. In *DISCO'90 (LNCS 429)*, pages 184–193. Springer-Verlag, 1990.

[BDP89] L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In *Rewriting Techniques*, chapter 1, pages 1–30. Academic Press, 1989.

[BGK95] R. Bündgen, M. Göbel, and W. Küchlin. Parallel ReDuX → PaReDuX. In *Rewriting Techniques and Applications (LNCS 914)*, pages 408–413, 1995.

[BGKW98] R. Bündgen, M. Göbel, W. Küchlin, and A. Weber. Parallel term rewriting with PaReDuX. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction—A Basis for Applications, Volume II: Systems and Implementation Techniques*, volume 9 of *Applied Logic Series*, pages 231–260. Kluwer, 1998.

[BH94] M. P. Bonacina and J. Hsiang. Parallelization of deduction strategies: An analytical study. *J. Automated Reasoning*, 13(1):1–33, 1994.

[DF94] J. Denzinger and M. Fuchs. Goal-oriented equational theorem proving using teamwork. In *KI-94 (LNAI 861)*, pages 343–354, 1994.

[DF98] J. Denzinger and M. Fuchs. A comparison of equality reasoning heuristics. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction—A Basis for Applications, Volume II: Systems and Implementation Techniques*, volume 9 of *Applied Logic Series*, pages 361–382. Kluwer, 1998.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Formal Models and Semantics*, volume 2 of *Handbook of Theoretical Computer Science*. Elsevier, 1990.

[DKS97] J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT – a distributed and learning equational prover. *J. Automated Reasoning*, 18(2):189–198, 1997.

[DL96] J. Denzinger and J. Lind. TWlib—a library for distributed search applications. In *Proc. ICS'96-AI*, pages 101–108, 1996.

[Hue81] G. Huet. A complete proof of correctness of the Knuth-Bendix completion algorithm. *J. Comput. Syst. Sci.*, 23:11–21, 1981.

[KB70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebra. In *Computational Problems in Abstract Algebra*. Pergamon Press, 1970.

[KW92] W. Küchlin and J. A. Ward. Experiments with virtual C Threads. In *SPDP'92*, pages 50–55. IEEE Press, 1992.

[MGB95] P. Maier, M. Göbel, and R. Bündgen. A multi-threaded unfailing completion. Technical Report 95–06, WSI, Universität Tübingen, Germany, 1995.

[SS93] C. B. Suttner and J. Schumann. Parallel automated theorem proving. In *Parallel Processing for Artificial Intellegence PPAI-93*. Elsevier, Amsterdam, 1993.

[SS98] G. Sutcliffe and C. Suttner. The TPTP problem library – CNF release v1.2.1. *J. Automated Reasoning*, 21(2):177–203, 1998.

[TG87] A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, 1987.