Parallelization of deduction strategies

/* Tutorial given at CADE 1994, Nancy, France, June 26; revised for a class in Spring 1995 at U. Iowa (AI Seminar); revised further for the course "Theorem-proving strategies" in the Intl. MS Program in Computational Logic, TU Dresden, May-June 1999 */

Maria Paola Bonacina
Department of Computer Science
The University of Iowa

Outline

0. A conceptual framework for parallel deduction

1. Parallelization of subgoal-reduction strategies

2. Parallelization of expansion-oriented strategies

3. Parallelization of contraction-based strategies

4. Distributed deduction for contraction-based strategies

5. Discussion

A conceptual framework for parallel deduction : outline

- Deduction strategies
- Classification of deduction strategies
- Classification of types of parallelism
- Critical issues :
  - granularity of parallelism
  - size and dynamics of the data base of clauses
  - organization of memory
  - conflicts between inferences

. . .

1. Parallelization of subgoal-reduction strategies:

   - Prolog technology theorem proving
   - parallel rewriting for equational languages

2. Parallelization of expansion-oriented strategies:

   - connection graph procedures
   - resolution-based methods
   - parallelizations of Buchberger algorithm

3. Parallelization of contraction-based strategies:

   - parallelization of Knuth-Bendix completion
   - resolution-based methods with contraction
   - completion-based methods

A conceptual framework for the study of parallel deduction

Deduction strategies

The first component of a deduction strategy is an *inference system*.

Inference rules:

- *expansion* rules, e.g. resolution,
  hyperresolution,
  paramodulation,
  superposition,
  ....
- *contraction* rules, e.g. simplification,
  normalization,
  subsumption,
  Tautology deletion,
  .....

Examples

resolution (expansion)

$\neg M(a2, b2, r1)$ $\qquad$ $\neg M(x, y, z) \vee M(y, x, z)$

$\neg M(b2, a2, r1)$

simplification (contraction)

$$P(f(f(f(f(\emptyset)))))$$

$$\Big\downarrow 3 \qquad f(f(x)) \to f(x)$$

$$P(f(\emptyset))$$

Deduction strategies

The second component of a deduction strategy is a search plan to control the inference system:

- selection of rules,
- selection of premises.

Example: given

1. $\neg M(a2, b2, r1)$
2. $\neg M(x, y \cdot y, z) \vee M(y, x, z)$
3. $y \cdot y \simeq y$

which step do first ?

Simplification - first search plan.

↳ eager - contraction

Deduction strategies

Strategy: $\mathscr{C} = \langle I ; \Sigma \rangle$

- I: inference system
- Σ: search plan

Theorem proving problem: $S \overset{?}{\models} \varphi$

Derivation:

$$S_0 \vdash S_1 \vdash \ldots \vdash S_i \vdash S_{i+1} \vdash \ldots$$

where $S_0 = S \cup \{\neg \varphi\}$

or

$$(S_0 ; \varphi_0) \vdash (S_1 ; \varphi_1) \vdash \ldots \vdash (S_i ; \varphi_i) \vdash (S_{i+1} ; \varphi_{i+1}) \vdash \ldots$$

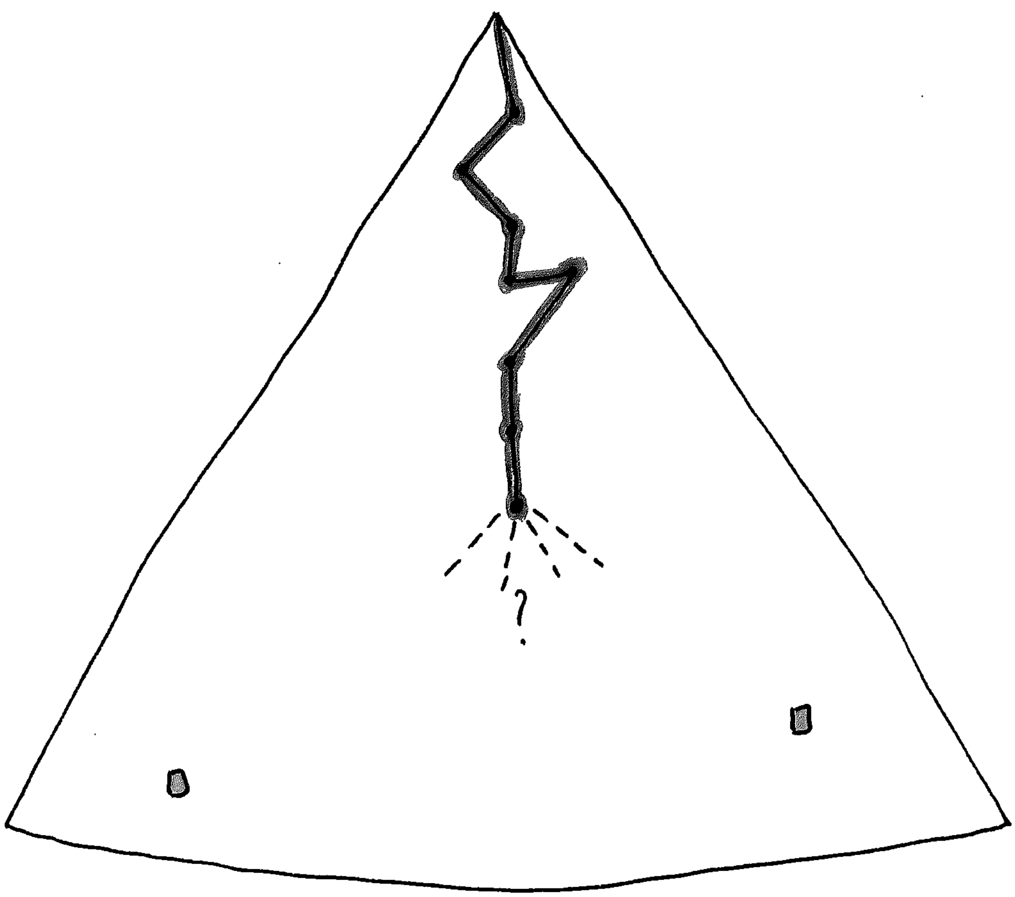
Deduction strategies

Refutational completeness of the inference system
Fairness of the search plan

} completeness of the strategy

Search tree:

▮ : solution



The classification of strategies for the purpose of parallelization

1. Subgoal - reduction strategies.

2. Expansion - oriented strategies.

3. Contraction - based strategies.

(2. and 3.: ordering-based strategies)

Subgoal-reduction strategies

Form of the derivation:

$$(S ; \varphi_0) \vdash (S ; \varphi_1) \vdash \dots \vdash (S ; \varphi_i) \vdash \dots$$

S → set of axioms or rules

φ_i → goal

$$(S ; \varphi_0 ; A_0) \vdash (S ; \varphi_1 ; A_1) \vdash \dots \vdash (S ; \varphi_i ; A_i) \vdash \dots$$

A_i → set of ancestor goals

Form of the typical inference:

$$\frac{(S \cup \{\psi\} ; \varphi ; A)}{(S \cup \{\psi\} ; \varphi' ; A \cup \{\varphi\})}$$

Example: Prolog

S is a Prolog program,

φ_0 is a query to be solved

and each step in a derivation

$$(S; \varphi_0; A_0) \vdash (S; \varphi_1; A_1) \vdash \dots \vdash (S; \varphi_i; A_i) \vdash \dots$$

is an SLD-resolution step (an expansion inference) where a rule in S resolves with the current goal to generate the new goal. (Linear input strategy: the set of ancestors A can be omitted in the formalization of the derivation since it is used only for backtracking.)

Example : term rewriting

S is a set of equations or rewrite rules,

φ_0 is a (ground) term to be reduced to normal form

and each step in a derivation

$$(S;\varphi_0) \vdash (S;\varphi_1) \vdash \ldots \vdash (S;\varphi_i) \vdash (S;\varphi_{i+1}) \vdash \ldots$$

is a rewriting step
(a contraction inference)
where a rule in S is applied
to rewrite φ_i.

Subgoal-reduction strategies

Strategies for:

- functional programming, equational programming, term rewriting,
- logic programming,
- their combinations,
- Prolog technology theorem proving,

. . . tableau-based strategies ...

Fundamental property:
regardless of whether expansion or contraction is used, the strategy works by searching the space of subgoals, while the data base of axioms remains static.

Expansion-oriented strategies

Form of the derivation:

$$(S_0 ; \varphi_0 ; N_0) \vdash (S_1 ; \varphi_1 ; N_1) \vdash \dots \vdash (S_i ; \varphi_i ; N_i) \vdash \dots$$

or

$$(S_0 ; N_0) \vdash (S_1 ; N_1) \vdash \dots \vdash (S_i ; N_i) \vdash \dots$$

S_i: main data base of clauses

N_i: set of "raw clauses"

Form of the typical inferences:

expansion :

$$\frac{(S \cup \{\psi_1, \psi_2\} ; N)}{(S \cup \{\psi_1, \psi_2\} ; N \cup \{\psi_3\})}$$

forward contraction:

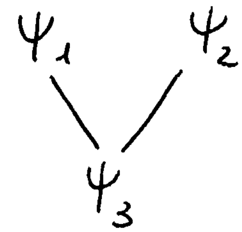
$$\frac{(S ; N \cup \{\psi\})}{(S \cup \{\psi'\} ; N)} \qquad \psi' = \psi\!\downarrow_S$$

Forward contraction

Select ψ_1 and ψ_2 in S_i,

generate raw clauses, e.g.

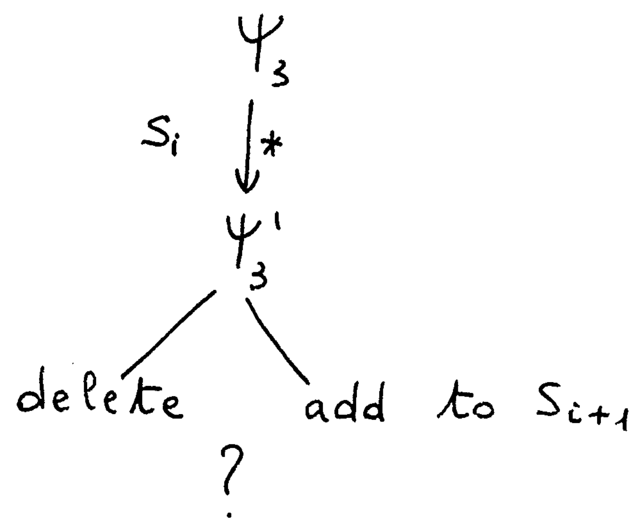
↳ by expansion rules



store raw clauses in N_{i+1},

contract raw clauses:

↳ forward contraction



Example: expansion-oriented, resolution-based Set of Support strategy

The set of clauses S is partitioned in SOS (set of support) and $U = S - SOS$

and a derivation

$$(SOS_0; U_0; N_0) \vdash (SOS_1; U_1; N_1) \vdash \ldots \vdash (SOS_i; U_i; N_i) \vdash \ldots$$

is generated by executing a basic loop:

1. select a clause c from SOS, **move c to U,**
2. generate by resolution all raw clauses into N from c and clauses in U,
3. normalize raw clauses, (forward contraction)
4. add non-trivial normal forms to SOS.

$$\boxed{S = SOS \cup U}$$

Expansion-oriented strategies

Fundamental properties:

search the space of consequences of the axioms and negation of the theorem primarily by expansion;

contraction is applied only to raw clauses in N: only forward contraction;

contraction is not applied to S: the data base is montonically increasing:

$$S_0 \subseteq S_1 \subseteq \ldots \subseteq S_i \subseteq S_{i+1} \subseteq \ldots$$

Contraction-based strategies

Form of the derivation:

$(S_0; \varphi_0) \vdash (S_1; \varphi_1) \vdash \ldots \vdash (S_i; \varphi_i) \vdash \ldots$ (target-oriented)

or

$S_0 \vdash S_1 \vdash \ldots \vdash S_i \vdash \ldots$

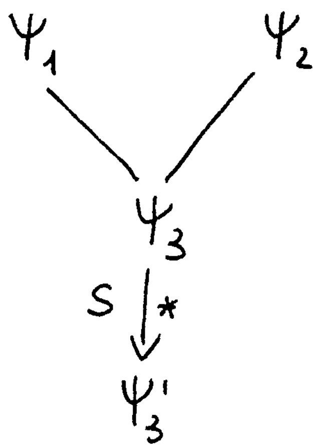
Form of the typical inferences:

expansion: $$\frac{S \cup \{\psi_1, \psi_2\}}{S \cup \{\psi_1, \psi_2, \psi_3\}}$$

contraction $$\frac{S \cup \{\psi\}}{S \cup \{\psi'\}} \quad \psi' = \psi\Downarrow_S$$

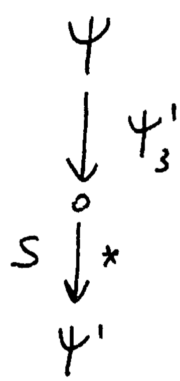
Backward contraction

Contraction-based strategies feature forward and backward contraction:



forward contraction

Use ψ'_3 to contract any ψ in S:



<u>backward contraction</u>

Use ψ' to contract others ....

to keep S inter-reduced.

Example: contraction-based, resolution-based Set of Support strategy

The basic loop to construct a derivation

$(SOS_0; U_0; N_0) \vdash (SOS_1; U_1; N_1) \vdash \ldots \vdash (SOS_i; U_i; N_i) \vdash \ldots$

is modified by the presence of a backward contraction phase:

1. select a clause c from SOS, **move c to U,**
2. generate by resolution all raw clauses from c and clauses in U,
3. normalize raw clauses,
4. add non-trivial normal forms to SOS,
5. apply newly inserted clauses to reduce the clauses in SOS and U: inter-reduce SOS and U.

(any clause in U normalized is put back into SOS)

Contraction-based strategies

Contraction-first search plan:

- use contraction rules eagerly to contain the growth of the generated search space;
- allow expansion inferences only if the premises are fully reduced.

Contraction-based strategies may feature:

+ orderings on terms and clauses,
+ ordering-based restrictions to expansion.

Several successful theorem provers are contraction-based:

Otter, RRL, Reveal .... EQP, SPASS ...

Contraction-based strategies

Fundamental properties:

search the space of consequences of the axioms and negation of the theorem by both expansion and contraction, applying contraction first (eager contraction);

contraction is applied to all clauses in the data base: both forward and backward contraction;

contraction is applied to S:
the data base is not monotonic, it is highly dynamic:

$$\forall i \geq 0 \quad S_i \subseteq S_{i+1} \quad \text{or} \quad S_i \not\subseteq S_{i+1}.$$

$$R(S_0) \leq R(S_1) \leq R(S_2) \leq \ldots$$

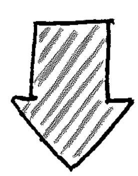
Parallelization

Given a problem with its data and operations, and n processes, how to subdivide the problem among the processes for parallel execution.

- Data-driven parallelization.
- Operations-driven parallelization.

Deduction:

- large data sets
- few types of operations



Data-driven parallelization.

Granularity of parallelism

Granularity of data:

- how large is the fraction of data for each process?
- which type of datum represents a grain of memory with its own access rights?

Granularity of operations:

- how large are the tasks to be executed in parallel by the processes?

Granularity of parallelism for deduction

	granularity of data	granularity of operations
parallelism at the term level (fine grain)	TERM or LITERAL (i.e., subexpression of formula)	SUBTASK OF INFERENCE STEP
parallelism at the clause level (medium grain)	CLAUSE including EQUATION as special case (i.e., formula)	INFERENCE STEP
parallelism at the search level (coarse grain)	SET OF CLAUSES (set of formulae)	MANY INFERENCE STEPS (DERIVATION)

Parallelism at the term level

Processes may access concurrently (disjoint) subterms of a given term and

- cooperate to achieve a single inference step (e.g. in parallel matching) or

- execute homogeneous inference steps in parallel (e.g. in parallel rewriting).

It has been applied to:

- parallel matching,
- parallel rewriting,
- parallel unification,
- parallel narrowing,
- AND-parallelism in logic programming and PTTP,

....

Parallelism at the clause level

Processes may access concurrently distinct clauses and each process executes several, generally homogeneous, inference steps with its clause.

Example: a parallel SOS-strategy where each process selects a different clause from a shared SOS and executes all expansion steps with its selected clause.

It has been applied to:

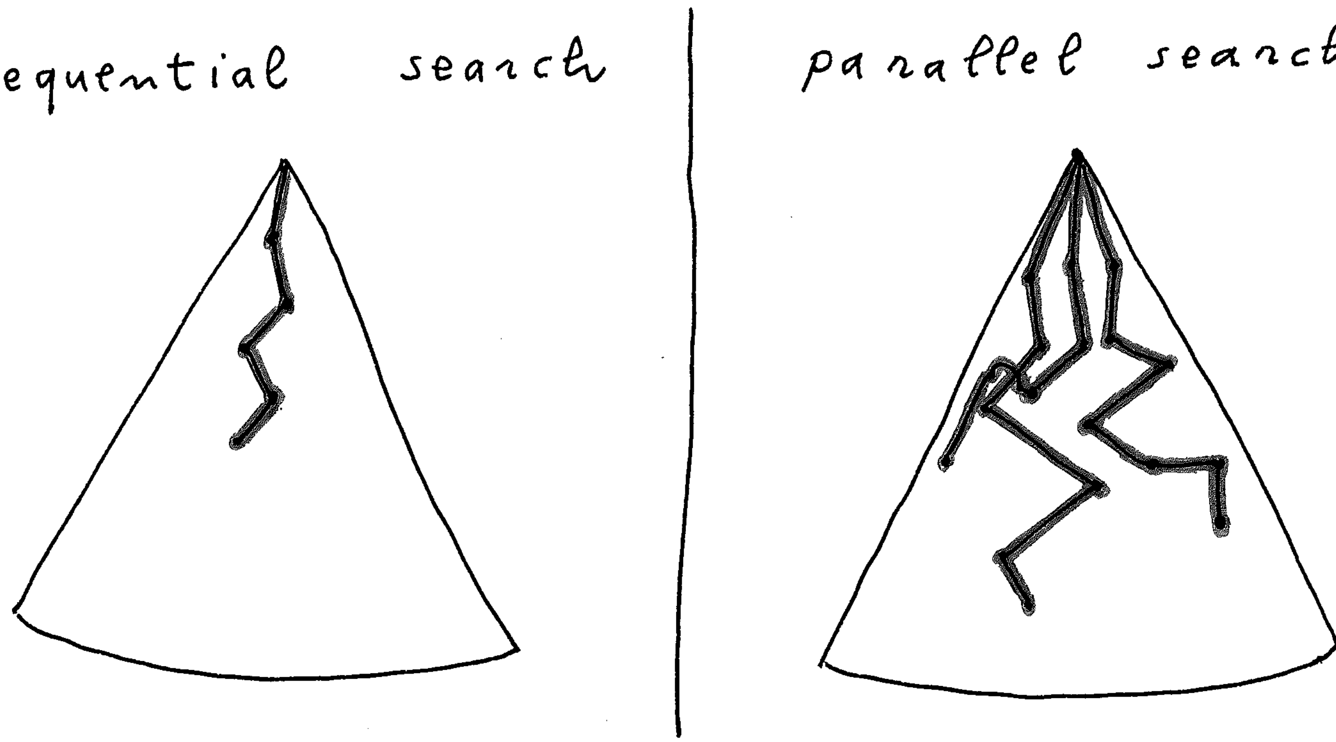
- OR-parallelism in logic programming and PTTP,
- parallel Knuth-Bendix completion,
- parallel narrowing,
- parallel resolution-based theorem proving,

....

Parallelism at the search level

Unlike in parallelism at the term or clause level, the idea is to parallelize search, rather than inferences:

sequential search | parallel search



Concurrent processes develop concurrent derivations.

Parallelism at the search level

Processes may access concurrently distinct sets of clauses and each process develops a derivation, comprising many, generally heterogeneous, inference steps.

Example: a parallel SOS-strategy where each process has its own SOS and U sets.

It has been applied to:

- parallel theorem proving,
- parallel Knuth-Bendix completion

...... other DAI applications ...

Shared memory vs. distributed memory

- Shared memory

• sharing of data • smaller memory	• protection (critical regions, locks, ...) • synchronization delays

- Distributed memory (with message-passing)

• higher degree of parallelism • asynchronous computations	• communication delay • duplication of data • larger memory

Granularity of parallelism and memory organization

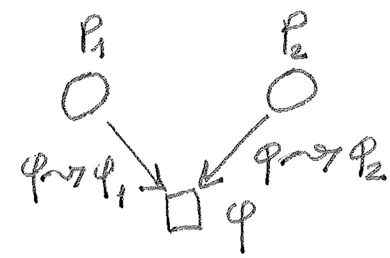
	shared memory	distributed memory
parallelism at the term level	✓ (most approaches)	
parallelism at the clause level	✓ (most approaches)	
parallelism at the search level	✓	✓ (most approaches)

Conflicts

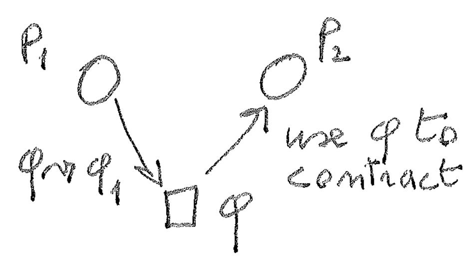
Conflicts between concurrent steps accessing common premises :

assuming that concurrent-read is not a cause of conflict, we have:

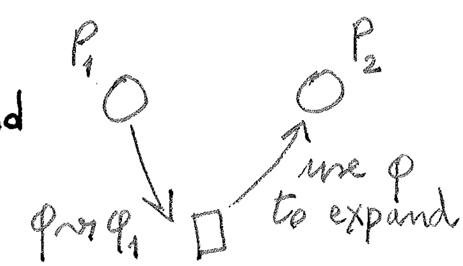
1) write-write conflicts between contraction steps,



2) read-write conflicts between contraction steps,



3) read-write conflicts between expansion steps and contraction steps.



Conflicts

- Parallel expansion steps do not cause conflicts on common premises, because they only need read-access.

- All read-write conflicts (contraction-contraction and expansion-contraction) are due to <u>backward contraction.</u>

- Assuming that raw clauses are not used as premises of expansion steps or to reduce other clauses, forward contraction steps may participate only in write-write conflicts between forward contraction steps.

Approaches to the problem of conflicts

- Fine / medium granularity, shared memory:
  add control to avoid conflicts
  - in the definition of the operations (e.g. concurrent rewrite of disjoint terms),
  - in the implementation.

- Coarse granularity, distributed memory:
  conflicts are prevented because the concurrent processes access physically separate memories.

Critical issues in determining the appropriate granularity and memory organization

- Size of the data base
- Dynamics of the data base
- Presence of read-only data
- Read-access vs. write-access
- Conflicts

Parallelization

of

subgoal-reduction

strategies

Prolog Technology Theorem Proving

A class of methods extending the inference system and search plan of Prolog in such a way that it is complete for FOL.

Basic points:

- inference system based on model-elimination
  [Loveland 69, Stickel 84 ....]

- search plan based on depth-first iterative deepening
  [Korf 85, Stickel - Tyson 85 ....]

- fast implementation on the WAM
  [D.H.D. Warren 83, Stickel 88 ...]

- very efficient use of caching / lemmaizing
  [Michie 68, D.S. Warren 79, D.H.D. Warren - Pereira 82, Vielle 86, Tamaki-Sato 86, Dietrich 87, Elkan 89, Astrachan - Stickel 92 .....]

Parallel Prolog Technology Theorem Proving

Some basic concepts:

- OR-parallelism
- AND-parallelism
- subgoals as tasks ; task-stealing
- representation of subgoals by encoding of WAM operations.

A few among the existing methods:

- PARTHENON [Clarke et al. 90 ; Bose et al. 92]
- PARTHEO [Schumann-Letz 90]
- METEOR [Astrachan-Loveland 91]
- "Nagging" [Sturgill-Segre 94]

......

Term rewriting based strategies for equational programming languages

- Functional programming:

  a regular (non-overlapping and left linear) term rewriting system as a program;

  regularity ⇒ confluence ⇒ uniqueness of normal forms

  [Hoffmann - O'Donnell 82 ......]

- Logic programming

  [Dershowitz - Josephson 84 .....]

Parallel term rewriting

For regular systems, parallel (outermost) rewriting is safe: it is normalizing and there are no conflicts.

$$\Downarrow$$

- Efficient implementation techniques.
- Weakening the regularity requirement.

A few among the existing approaches:

- The Rewrite Rule Machine
  [Goguen - Meseguer et al. 86, 88 ...]
- The Abstract Concurrent Machine
  [Dershowitz - Lindenstrauss 90]
- EQUALS
  [Kaser - Pawagi - Ramakrishnan - Sekar et al. 92]
- Concurrent DAG Rewriting
  [C. Kirchner - Viry 92]

.....

Parallelization of subgoal-reduction strategies

- Separation of rules and subgoals.
- Static data base of rules.
- Search limited to the space of subgoals.

- Pre-processing of the rules.
- The rules are read-only data.
- Specialized data structures for rules and subgoals respectively.
- No conflicts.

- A grain of data can be as small as a term.
- All granularities of parallelism apply.
- Both shared and distributed memory can be used.

Parallelization of expansion-oriented strategies

Some parallel expansion-oriented strategies

- Parallel connection graph procedures for theorem proving:

  [Loganantharaj - Müller 86]

  [Cheng - Juang 87]

  ....

- Parallel resolution-based theorem proving strategies with no backward contraction:

  DARES [Conry - MacIntosh - Meyer 90]

  PARROT [Jindal - Overbeek - Kabat 92]

  ....

- Parallel implementations of the Buchberger algorithm:

  [Vidal 90]

  [Siegl 90]

  [Hawley 91]

  [Chakrabarti - Yelick 93]

  ....

Critical issues in the parallelization of expansion-oriented strategies: the scale of the problem

The strategy searches by expansion the space of consequences of the axioms.

- The data base of clauses (S) becomes very large.
- The data base of clauses is not static: it is monotonically increasing.

The scale of the problem changes with respect to subgoal-reduction strategies.

Critical issues in the parallelization of expansion-oriented strategies: read-only data

Since new clauses need to be added, neither S (the data base of clauses) nor N (the set of raw clauses) are read-only.

Parallel expansion in shared memory:

if S (or N) is implemented as a shared set, then

- either there are write-write conflicts among expansion processes (e.g., concurrent "add" to a shared SOS)
- or S is protected by Exclusive-Write mode and the expansion part of the strategy is forced to be largely sequential.

Critical issues in the parallelization of expansion-oriented strategies : conflicts

- No special assumptions (e.g. non-overlapping ; confluence) can be made on the equations that will be used as simplifiers during the derivation.

- The data base (S) is

large, monotonically increasing $\Longrightarrow$
- no pre-processing
- no specialized data structures

⇓

write-write conflicts in forward contraction.

Critical issues in the parallelization of expansion-oriented strategies : conflicts

In connection graph procedures

[Kowalski]

- a set of clauses is a graph with an edge linking any two literals of opposite sign
- resolution is link resolution.

Link resolution includes the deletion of the link resolved upon :

unrestricted parallel link resolution
(parallelism at the literal / term level)
causes conflicts,
that make it inconsistent.
[Loganantharaj 85]

Parallelization of expansion-oriented strategies (summary)

- The data base of clauses (S) is *large* and *growing* during the derivation.

- Single clauses are *read-only* after insertion in S, but the whole S is *not*.

- *Write-write conflicts*.

⇓

- Term level granularity is too fine.
- Parallelism at the *clause* or *search* level prevail.

- Both shared and distributed memory can be used.

Most approaches to parallel expansion-oriented deduction adopt

either parallelism at the clause level

or parallelism at the search level

Examples:

Parallelism at the clause level:

- PARROT [Jindal-Overbeek-Kabat 92] and
- ROO [Lusk-McCune 92]

implement in shared memory set of support resolution-based strategies.

...

Parallelism at the search level:

- DARES [Conry-MacIntosh-Meyer 90]

implement resolution-based strategies in distributed memory with message-passing.

...

Parallel implementations of the Buchberger algorithm

- Generation of polynomials ~ superposition (expansion)
- Reduction of polynomials ~ simplification (contraction)

- Backward contraction is not as crucial as in theorem proving.
- Most implementation are expansion oriented.

Some approaches:

- Parallelism at the clause level:
  [Siegl 90]

- Parallelism at the search level:
  [Hawley 91]
  [Chakrabarti - Yelick 93]

. . . .

Parallelization of contraction-based strategies

Some parallel contraction-based strategies

- Parallel implementations of Knuth-Bendix completion:

  transition-based Knuth-Bendix completion [Yelick - Garland 92]

  ....

- Parallel resolution-based strategies with contraction and completion-based strategies:

  ROO [Lusk - McCune 92]

  Team-Work method [Denzinger 91] [Avenhaus - Denzinger 92]

  Clause-Diffusion methodology [Bonacina 92] [Bonacina-Hsiang 93]

  Aquarius [Bonacina-Hsiang 93]

  Peers [Bonacina - McCune 94]

  .... Modified Clause-Diffusion [Bonacina 94]

  Peers-mcd [Bonacina 97]

Critical issues in the parallelization of contraction-based strategies

The issues that were critical for expansion-oriented strategies become even more problematic:

1) Size and dynamics of the data base:

   S is very large and highly dynamic (not even monotonic).

2) No read-only data:

   not only S as a whole, but each clause in S is not read-only, because it can be rewritten by backward contraction at any stage of the derivation

⇓

There is no read-only data to be conveniently shared in shared memory.

Critical issues in the parallelization of contraction-based strategies

3) Conflicts:

similar to expansion-oriented strategies, the size and dynamics of S make the use of pre-processing and specialized data structures employed by subgoal-reduction strategies not feasible.

In addition to

- write-write conflicts in forward contraction

backward contraction induces:

- write-write conflicts in backward contraction
- read-write conflicts in backward contraction
- read-write conflicts between expansion and backward contraction steps.

Critical issues in the parallelization of contraction-based strategies

Furthermore,

4) The rate of write-accesses versus read-accesses is higher, because of the write-accesses generated by backward contraction: another factor which is not favorable to shared memory.

5) All clauses may play both roles of "simplifier" and "simplified": another factor which is not favorable to specialized data structures.

All these difficulties are largely due to backward contraction, which is what makes contraction-based strategies generally more efficient than expansion-oriented strategies especially in problems with equality.

Example of the conflicts induced by backward contraction

- Parallelism at the clause level.
- Shared memory (concurrent-read is legal).
- Contraction-based, resolution-based, set of support strategy.

Assume that SOS and U are in shared memory and each parallel process executes the basic cycle:

1. select a clause from SOS
2. generate raw clauses
3. forward contraction of raw clauses
4. add non-trivial normal forms to SOS (add redundant clauses: expansion-contraction conflicts)
5. backward contraction of SOS and U by new clauses and inter-reduction (contraction-contraction conflicts and **expansion-contraction conflicts**).

The backward contraction bottleneck

- Parallelism at the clause level.
- Shared memory.

Each backward contraction step may induce many.

Concurrent backward contraction processes $\Rightarrow$ conflicts $\Rightarrow$ synchronization for conflict prevention

$\Downarrow$

sequentialization, delays.

Only one backward contraction process $\Rightarrow$ bottleneck, starvation of expansion processes.

Parallelization of contraction-based strategies (summary)

- Large, dynamic data base
- Conflicts
- Backward contraction bottleneck

⇓

- Both parallelism at the term level and parallelism at the clause level are too fine.
- Coarse grain parallelism (parallelism at the search level) is the most appropriate ⇒ distributed memory.

	subgoal reduction strategies	expansion oriented strategies	contraction based strategies
size of the data base	small	very large	very large
dynamics of the data base	static	monotonic	dynamic
pre-processing	yes	no	no
read-only data	yes	yes	no
specialized data structures	yes	no	no
conflicts	no	no	yes

Types of parallelisms and strategies

	subgoal reduction strategies	expansion oriented strategies	contraction based strategies
parallelism at the term level	✓		
parallelism at the clause level	✓	✓	
parallelism at the search level	✓	✓	✓

Distributed deduction
for
contraction-based strategies

Parallelism at the search level

The concurrent deductive processes are loosely coupled and asynchronous.

Each process searches for a solution by developing its own derivation.

The processes cooperate by some form of communication (communication scheme).

The processes work on separate sets of clauses ⇒ no conflicts.

Success is reached as soon as one of the processes succeeds.

Distributed memory: agreement of data

Parallelism at the search level
- shared memory
- distributed memory

Distributed memory as a natural choice:
- separate data bases,
- indipendent, asynchronous access.

Furthermore, in parallel deduction there is no need to enforce agreement (coherence) of data for correctness:

φ and φ' are logically equivalent.

Shared memory for normalization

1) Distributed memory : loosely coupled asynchronous concurrent activities.

2) Shared memory : tightly coupled synchronous concurrent activities.

Contraction-based parallel deduction: mostly (1) but also some of (2): normalization.

All simplifiers in one place:

a shared memory component.

Distributed environment

- **Purely distributed:**
  - asynchronous, loosely coupled processors,
  - distributed memory,
  - communication by message-passing.

- **Mixed shared-distributed:**
  - also a shared memory component,
  - combines message-passing with communication through memory.

* Networks of computers
(e.g. networks of work stations)

* Asynchronous multi-processors with distributed memory.

Subdivision of work among parallel processes

Parallelism at the search level:

- partition the search space,
- non-overlapping searching processes.

A few approaches:

- subdivide the clauses, same search plan. (DARES)
- same set of clauses, different search plans. MULTI-SEARCH, e.g. (Team-Work)
- subdivide the clauses, subdivide the inferences, either same or different search plan. DISTRIBUTED SEARCH, e.g. (Clause-Diffusion)

Subdivision of work among parallel processes

- Different search plans: MULTI-SEARCH
  - competition of search plans,
  - interleaving of search plans,
    (Team-Work)
  - cooperation of search plans.
    (Clause-Diffusion)

- Subdivision of clauses and inferences:
  - physical partition of clauses,
    (DARES)
  - distinction between
    physical partition of clauses
    logical partition of clauses

  makes data-driven subdivision of inferences possible.
  (Clause-Diffusion)

DISTRIBUTED SEARCH

Communication

- Periodical ri-generation of a common data base of clauses:
  - periodical synchronization,
  - selection of best set of clauses,
  - selection of "good" clauses.

  (Team-Work)

- Asynchronous message-passing:
  - request-messages and reply-messages, heuristics (DARES)
  - inference-messages, partition of inferences, fairness (Clause-Diffusion).

Contraction

- DARES: expansion-oriented method
  no backward contraction.

- Team-Work: no subdivision of clauses ⇒
  each process does all contraction
  locally like a sequential process.
  There may be specialized
  normalization processes.

- Clause-Diffusion: subdivision of clauses ⇒
  distributed global contraction.

Two approaches:

- localized image sets
  in distributed memory,

- global image set
  in a shared memory component.

A distributed theorem proving strategy

is specified by:

1) the inference system

2) the search plan which schedules
   - the inference steps and
   - the communication phases (synchronous) or the communication steps (asynchronous)

depending on

3) the communication scheme

4) the criteria to subdivide clauses and inferences

5) the mechanism for message-passing (routing / broadcasting / sharing)

6) the scheme for distributed global contraction

. . . .

Distributed derivations

n processes : $P_1, P_2, \dots, P_n$

n strategies : $\varphi_1, \varphi_2, \dots, \varphi_n$

n (generally asynchronous) derivations:

$$S_0^1 \vdash S_1^1 \vdash \dots \vdash S_i^1 \vdash S_{i+1}^1 \vdash \dots$$
$$\vdots$$
$$S_0^k \vdash S_1^k \vdash \dots \vdash S_i^k \vdash S_{i+1}^k \vdash \dots$$
$$\vdots$$
$$S_0^n \vdash S_1^n \vdash \dots \vdash S_i^n \vdash S_{i+1}^n \vdash \dots$$

where

- S is generally a tuple of components depending on the strategy,
- the subindices of the derivations are independent (asynchronous),
- steps are inference steps or communication steps.

Discussion

and

research problems

Considerations on parallel architectures and parallel deduction strategies

- The finer the granularity, the more tightly coupled the processors.

- Common, read-only data: shared memory. High rate of write-access to most or all the data: distributed memory.

Considerations on parallel architectures and parallel deduction strategies

Subgoal-reduction strategies:

- shared memory machines for pre-processing and specialized data structures, e.g. PARTHENON, METEOR ....
- tightly coupled multiprocessors with small nodes and fast communication e.g. RRM, PARTHEO, Concurrent DAG Rewriting .....
- distributed memory environments (networks of workstations) e.g. METEOR, "Nagging" ....

Considerations on parallel architectures and parallel deduction strategies

Expansion-oriented strategies:

- shared memory machines for expansion (concurrent read) and forward contraction (the simplifiers are read-only)

  e.g. parallel connection procedures,
  parallel Buchberger algorithms,
  PARROT,
  ROO ....

Also

- tightly coupled multiprocessors

  and

- distributed memory machines

  e.g. DARES,
  parallel Buchberger algorithms
  ....

Considerations on parallel architectures and parallel deduction strategies

Contraction-based strategies:

- distributed memory environments (networks of workstations) to prevent write-bottlenecks,
  e.g. Team-Work, (DISCOUNT)
  Clause-Diffusion (Aquarius, Peers, Peers-mcd)
  ....

- distributed memory systems with a shared memory component to reduce duplication and communication latency,
  e.g. Clause-Diffusion (not implemented).

Some methodological contributions of this survey

1) Identification of
   classes of strategies,
   types of parallelism.

2) Relations between
   - classes of strategies,
   - granularities of parallelism,
   - parallel architectures and memory organizations.

3) Role of backward contraction.

4) Attention to search and search plans in:
   differentiation of strategies,
   definition of parallelism
   (parallelism at the search level).

Problems and directions for research

1) Distributed deduction for contraction-based strategies:

   a) reducing redundant duplication,
      redundant communication,

   b) distributed global contraction,

   c) communication:
      - granularity of communication,
      - asynchronous / synchronous,

   d) subdivision of work:
      - distribution of data,
      - partition of the search space.

2) Parallel / distributed deduction:

   a) parallel search,

   b) design of parallel search plans,

   c) new search patterns in a partitioned search space.