Parallel automated reasoning

Maria Paola Bonacina

Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy, EU

Lecture at the SAT/SMT/AR School
Lisbon, 5 July 2019

Motivation for parallel reasoning

- Problems from applications get bigger and bigger
- It is hard to improve sequential performance
- Parallel hardware is available
- Automated reasoning neatly separates inference and control: from sequential to parallel organization of inferences?

Motivation for parallel reasoning

- Several SAT/SMT/AR systems are portfolio systems
- Multiple strategies by interleaving, time slicing, or in parallel
- Portfolio system: framework for parallel experiments or parallel prover/solver?
- Different degrees of integration/interaction
- What is a parallel prover/solver?
- Why is parallel reasoning challenging?

Focus of the lecture

Parallel strategies for

- Automated theorem proving (ATP) in
- First-order logic (FOL)

Further reading:

- Youssef Hamadi and Lakhdar Sais (Editors)
  Handbook of Parallel Constraint Reasoning
  Springer, May 2018
- Chapter 6: Maria Paola Bonacina. Parallel theorem proving (with 230 references)

Theorem-proving strategies

Theorem proving as inference + search

- Inference system: a set of inference rules
- Generate a derivation by applying the inference rules
- An inference system is non-deterministic
- Theorem-proving strategy: inference system + search plan
- A theorem-proving strategy is a deterministic procedure
- Refutationally complete inference system + fair search plan = complete theorem-proving strategy
- Parallelism affects the search component

Taxonomy of theorem-proving strategies

- Ordering-based strategies
- Subgoal-reduction strategies
- Instance-based strategies

- This lecture: ordering-based and subgoal-reduction strategies
  - Less work on parallelizing instance-based strategies
  - That have some commonalities with subgoal-reduction strategies from a parallelization viewpoint

Ordering-based strategies

Ordering-based strategies

- Expansion and contraction of a set of clauses (e.g., resolution, subsumption, paramodulation/superposition, simplification)
- Well-founded partial ordering $\succ$ on terms, literals, clauses:
  - Restrict expansion
  - Define contraction and redundancy
- State of the art for quantifier reasoning + equality reasoning
- Provers: e.g., Otter, EQP, Prover9, Spass, Discount, E, Gandalf, Vampire, Waldmeister, Zipperposition

Expansion inference scheme

An inference

$$\frac{A}{B}$$

where A and B are sets of clauses is an expansion inference if

- $A \subset B$: something is added
- Hence $A \prec B$
  ($\succ$ extended by multiset extension)
- Soundness of expansion: what is added is a logical consequence of what was already there
  $B \setminus A \subseteq Th(A)$ hence $B \subseteq Th(A)$ hence $Th(B) \subseteq Th(A)$

Expansion inference rule: superposition

Example:

$$\frac{f(z,e) \simeq z \quad f(I(x,y),y) \simeq x}{I(x,e) \simeq x}$$

- $f(z,e)\sigma = f(I(x,y),y)\sigma$
- $\sigma = \{z \leftarrow I(x,e),\ y \leftarrow e\}$
- $f(I(x,e),e) \succ I(x,e)$ (by the subterm property)
- $f(I(x,e),e) \succ x$ (by the subterm property)
- Superposition closes a peak:
  $I(x,e) \leftarrow f(I(x,e),e) \rightarrow x$

Expansion inference rule: superposition/paramodulation

$$\frac{S \cup \{l \simeq r \vee C, \quad L[s] \vee D\}}{S \cup \{l \simeq r \vee C,\ L[s] \vee D,\ (L[r] \vee C \vee D)\sigma\}}$$

- s is not a variable
- $l\sigma = s\sigma$ with σ mgu
- $l \simeq r$: para-from literal/clause
- $L[s]$: para-into literal/clause
- $l\sigma \not\preceq r\sigma$ and if $L[s]$ is $p[s] \bowtie q$ then $p\sigma \not\preceq q\sigma$ ($\bowtie$ is $\simeq$ or $\not\simeq$)
- $(l \simeq r)\sigma \not\preceq M\sigma$ for all $M \in C$
- $L[s]\sigma \not\preceq M\sigma$ for all $M \in D$

Contraction inference scheme

An inference

$$\frac{A}{B}$$

where A and B are sets of clauses is a contraction inference if

- $A \not\subseteq B$: something is deleted or replaced
- $B \prec A$: if replaced, replaced by something smaller
- Soundness of contraction adds adequacy:
  what is gone is logical consequence of what is kept
  $A \setminus B \subseteq Th(B)$ hence $A \subseteq Th(B)$ hence $Th(A) \subseteq Th(B)$
  (monotonicity)
- Every step sound and adequate: $Th(A) = Th(B)$

Contraction inference rule: simplification

$$\frac{S \cup \{s \simeq t,\ L[r] \vee C\}}{S \cup \{s \simeq t,\ L[t\sigma] \vee C\}}$$

- $s\sigma = r$ and $s\sigma \succ t\sigma$
- $L[t\sigma] \vee C$ is entailed by the original set (soundness)
- $L[r] \vee C$ is entailed by the resulting set (adequacy)
- $L[r] \vee C$ is redundant

$$\frac{S \cup \{f(x,x) \simeq x,\ P(f(a,a)) \vee Q(a)\}}{S \cup \{f(x,x) \simeq x,\ P(a) \vee Q(a)\}}$$

Ordering-based strategies: derivation

- Input set S
- Inference system: a set of inference rules
- Derivation: $S = S_0 \vdash S_1 \vdash \ldots S_i \vdash S_{i+1} \vdash \ldots$
  $\forall i\ S_{i+1}$ is derived from S_i by an inference
- Refutation: a derivation such that $\Box \in S_k$ for some k
- Refutational completeness: for all unsat S there is refutation
- Persistent clauses: $S_\infty = \bigcup_{i\geq 0} \bigcap_{j\geq i} S_j$
- Once redundant always redundant

Ordering-based inference system

- Expansion rules: ordered resolution, ordered factoring, superposition/ordered paramodulation, equational factoring, reflection (resolution with $x \simeq x$)
- Contraction rules: subsumption, simplification, tautology deletion, clausal simplification (unit resolution + subsumption)
- Refutationally complete

Contraction before expansion

- Simplification-first search plans
- Contraction-first search plans
- Eager-contraction search plans
- Keep sets of clauses interreduced

Forward and backward contraction I

- Forward contraction:
  - Reduce new clause φ by older clauses
  - Find all clauses ψ that can reduce φ
- Backward contraction:
  - Reduce older clause ψ by new clause φ
  - Find all clauses ψ that φ can reduce

Forward and backward contraction II

- Forward contraction before backward contraction
  - Forward contraction implemented as pre-processing clause φ
  - Forward contraction is part of the generation of φ
  - Before forward contraction: raw clause
  - Backward contraction implemented as post-processing φ: detect that ψ can be reduced + forward contraction ψ
  - Clauses generated by backward contraction treated like those generated by expansion
- Backward contraction: highly dynamic database of clauses

Search plans for ordering-based strategies

- Lists `To-Be-Selected` and `Already-Selected`
- Given-clause algorithm: select a given-clause φ from `To-Be-Selected`, do all expansion inferences between φ and all ψ in `Already-Selected`, move φ to `Already-Selected`
- Apply forward contraction to each raw clause
- Two versions for backward contraction:
  - Keep the union of the two lists interreduced
  - Keep only `Already-Selected` interreduced

Subgoal-reduction strategies

Subgoal-reduction strategies

- Linear resolution, model elimination (ME):
  pick a goal clause and try to reduce it to $\Box$
  by reducing goals to subgoals
- ME-tableaux: Tableau as survey of interpretations
  Try to eliminate them all
  Tableau frontier $\sim$ goal clause
- Equality reasoning still an open problem
- Provers: e.g., Setheo, Protein, leanCoP, EKR-Hyper

Ordered linear resolution

- At each step: resolve current goal $L \vee C$ with side clause $L' \vee D$ such that $L\sigma = \neg L'\sigma$
- Next goal: the resolvent $(D \vee C)\sigma$
- Subgoal L reduced to a new bunch of subgoals $D\sigma$
- Side clause: either input or ancestor
- Linear: at every step one parent is previous resolvent
- Ordered: literals in the goal reduced in fixed order e.g., left-to-right (literal-selection rule)

Model elimination

- ME-extension: resolve current goal $L \vee C$ with side clause $L' \vee D$ such that $L\sigma = \neg L'\sigma$
- Next goal: the resolvent $(D \vee [L] \vee C)\sigma$
- Reduced subgoal L saved as framed literal
- ME-reduction: reduce goal $L' \vee D \vee [L] \vee C$ to $(D \vee [L] \vee C)\sigma$ when $L\sigma = \neg L'\sigma$
- ME-contraction: reduce goal $[L] \vee C$ to C
- Side clause: input clause
- Linear input strategy for FOL

Why model elimination?

- $L \vee C$ and $L' \vee D$ with $L\sigma = \neg L'\sigma$:
  no model can satisfy the two clauses by satisfying $L\sigma$ and $L'\sigma$
- $(D \vee [L] \vee C)\sigma$: the framed $L\sigma$ is added to the current candidate model (satisfies $(L \vee C)\sigma$)
- Something in $D\sigma$ must be satisfied to satisfy $(L' \vee D)\sigma$:
  the literals of $D\sigma$ are subgoals of $L\sigma$
- ME-reduction of $L' \vee D \vee [L] \vee C$ to $(D \vee [L] \vee C)\sigma$
  when $L\sigma = \neg L'\sigma$:
  a model with L cannot satisfy $L'\sigma$
- ME-contraction of $[L] \vee C$ to C: no model with L

Subgoal-reduction strategies: derivation

- Derivation: $(S;\varphi_0) \vdash (S;\varphi_1) \vdash \ldots (S;\varphi_i) \vdash \ldots$
  φ_i: goal clauses
- Refutation: $(S;\Box)$ at some stage
- Refutational completeness: if S unsat and $S \setminus \{\varphi_0\}$ sat, there is refutation from $(S;\varphi_0)$
- Redundancy: repeated subgoals
- Lemma learning: when ME-contracting $[L] \vee C$ to C
  learn lemma $\neg L$

Subgoal-reduction strategies: search plan

- Depth-first search (DFS)
  - Literal-selection rule or AND-*rule*
  - Clause-selection rule or OR-*rule*
- Backtracking to get out of dead-end
  (goal clause to which no inference applies)
- Iterative deepening on the number of inferences (resolution or ME-extension) for fairness, hence completeness

Parallelism and deduction

Parallelism and deduction

Parallelism at the

- Term/literal level: fine-grain
  Below the inference level
- Clause level: medium-grain
  At the inference level: parallel inferences
- Search level: coarse-grain
  Multiple processes cooperate searching in parallel for a proof

Fine-grain parallelism for subgoal-reduction

- AND-parallelism: reduce in parallel distinct goal clause literals or tableau leaves
- Literals of the same clause may share variables: conflict
- Example:
  - Subgoals: $\neg P(x)$ and $\neg Q(x, y)$
  - Side clauses: $P(a) \vee C$ and $Q(f(z), z) \vee D$
  - Conflict between $x \leftarrow a$ and $x \leftarrow f(z)$

AND-parallelism not for theorem proving

Fine-grain parallelism for ordering-based strategies

- Rewrite in parallel subterms at distinct positions in a term
- The positions can be:
  - Disjoint positions
  - A variable overlap
  - A non-variable overlap

Disjoint positions: parallel rewriting

- Example:
  - $i(i(x)) \simeq x$
  - $f(x, y) \simeq f(y, x)$
  - $h(i(i(a)), f(a, b)) \rightarrow^{\parallel} h(a, f(b, a))$
- Parallel rewriting: at disjoint positions

Variable overlap: concurrent rewriting

- Example:
  - $h(x, x) \simeq x$
  - $f(y, b) \simeq y$
  - $a \leftarrow h(a, a) \leftarrow f(h(a, a), b) \rightarrow f(a, b) \rightarrow a$
- Same result in either order
- Concurrent rewriting: at disjoint positions and variable overlaps

Non-variable overlap: conflict

- Example:
  - $f(z, e) \simeq z$
  - $f(I(x, y), y) \simeq x$
  - $I(a, e) \leftarrow f(I(a, e), e) \rightarrow a$
- Contraction/contraction Write-write conflict: two contraction steps rewrite the same clause
- Parallel/concurrent rewriting assume non-overlapping equations

Parallel/concurrent rewriting: summary I

Declarative programming languages:

- Fixed set E of input equations
- Goal is to rewrite a term t to its unique normal form
- Regular rewrite system R: non-overlapping and left-linear
- R: confluent, not terminating
- Compile R in ad hoc data structures for concurrent rewriting
- Rewrite engines: Elan, Maude

Parallel/concurrent rewriting: summary II

Theorem proving:

- Equations do overlap
- Goal is refutation
- Superposition (that closes the peak of a write-write rewriting conflict) is necessary
- Large set of generated and kept clauses
- Dynamic set of clauses: growing by expansion and shrinking by contraction
- Concurrent rewriting not for theorem proving

Take-home message

- Conflicts among parallel inferences
- Size and dynamicity of the database of generated and kept clauses

stand in the way of fine-grain parallelism for theorem proving

Parallel inferences

Parallel inferences for subgoal-reduction I

- OR-parallelism: reduce distinct goal clauses in parallel
- Try in parallel the proof attempts that a sequential strategy tries in sequence by backtracking
- Task (φ, j, k)
  - φ: goal clause
  - j: number of ME-extension steps used to generate φ
  - k: limit of iterative deepening
  - Reduce φ to $\Box$ in at most $k - j$ ME-extension steps
  - Active iff $k > j$
- From (φ_i, j, k) to $(\varphi_{i+1}, j + 1, k)$

Parallel inferences for subgoal-reduction II

- Parallel derivation: $(S; G_0) \vdash (S; G_1) \vdash \ldots (S; G_i) \vdash \ldots$
  G_i: set of active tasks
- Processes $p_0, \ldots, p_{n-1}$: all active as soon as $|G_i| > n$
- Each p_h maintains a queue of its active tasks
- Distribution of tasks by task stealing
- Communication by message passing or in shared memory

Parallel inferences for subgoal-reduction: summary

- Static database of clauses S
- Compile S à la Prolog (Prolog Technology Theorem Proving)
- (φ, j, k) encoded as the operations that generate it
- Recall ratio of iterative deepening: in exponential search tree, almost all nodes are on the frontier, re-expanding inner nodes does not matter much
- Provers PARTHEO, PARTHENON, and METEOR

Parallel inferences in ordering-based strategies I

- Parallelize the Otter given-clause algorithm: ROO
- `To-Be-Selected` and `Already-Selected` in shared memory
- Task A: expansion (including forward contraction) with given-clause φ
- Processes $p_0, \ldots, p_{n-1}$ select given-clauses $\varphi_0, \ldots, \varphi_{n-1}$ and each executes Task A
- Can p_h append its set N_h of new clauses to `To-Be-Selected`? No: $\psi \in N_1$ not reduced w.r.t. N_2
- p_h appends them to a third list: `K-list`

Parallel inferences in ordering-based strategies II

- Backward contraction in parallel? No, conflicts
- p_h finds that ψ can be back-contracted: ψ in `To-Be-Deleted`
- Task B: inter-reduce `K-list`, move its clauses to `To-Be-Selected`; backward-contraction of `To-Be-Deleted`
- If `K-list` $!=$ `nil` or `To-Be-Deleted` $!=$ `nil` and none's doing Task B, do it, else do Task A
- Only one p_h does Task B: sequential backward-contraction
- Backward-contraction bottleneck

Parallel inferences: more conflicts

1. Contraction/contraction write-read conflict: one rewrites a φ that another one uses as premise to contract some other ψ
2. Contraction/expansion write-read conflict: one rewrites a φ that an expansion step uses as premise

- Both due to backward contraction
  (clauses subject to forward contraction not used as premises)
- Type (1) harmless as once redundant always redundant

Parallel inferences for ordering-based strategies: summary

- Backward contraction indispensable to counter space growth
- Impact of backward contraction:
  - No read-only data: any clause can be contracted
  - Highly dynamic database of generated and kept clauses
  - Conflicts between parallel inferences
- Stand in the way of medium-grain parallelism for ordering-based strategies

Take-home message

- Subgoal-reduction strategies: somewhat amenable to parallel inferences
- Ordering-based strategies: not amenable to parallel inferences
- From parallel inferences to parallel search

Parallel search

Parallelism at the search level

- Parallelism at the term/literal or clause levels:
  find proof sooner by speeding-up the same search
  that would be done sequentially
- Parallelism at the search level:
  find proof sooner by generating
  multiple different communicating searches

Parallel search I

- Parallel processes $p_0, \ldots, p_{n-1}$
- Each builds its own derivation and its own database of generated and kept clauses
- Success when one p_h finds a proof
- Communication
- Separate databases: no conflicts, no backward-contraction bottleneck
- Duplication harmless for soundness if inferences are sound

Parallel search II

How to differentiate the searches of $p_0, \ldots, p_{n-1}$?

- Distributed search: subdivide the search space among the processes (divide and conquer)
- Multi-search: let the processes use different search plans or different inference systems or both
- Both with communication
- The two can be combined

Parallel search III

- Ordering-based strategies:
  - Distributed search
  - Multi-search
  - Their combination
- Subgoal-reduction strategies:
  - Multi-search

Multi-search

Multi-search for subgoal-reduction I

Differentiate the searches of $p_0, \ldots, p_{n-1}$ by

- Different literal-selection rules
- Different clause-selection rules
- Different limits for iterative deepening
- Different initial goal clauses
- Combinations of these

Multi-search for subgoal-reduction II

- Derivation: $(S; G_0^k) \vdash (S; G_1^k) \vdash \ldots (S; G_i^k) \vdash \ldots$
  G_i^k: set of active tasks at process p_k at stage i
- Communication of tasks
- If p_k has (φ, j, q) and (φ', j', q') with $q < q'$, (φ, j, q) has higher priority for completeness
- Successors of PARTHEO prover: SETHEO, E-SETHEO, SPTHEO, CPTHEO, and P-SETHEO

Heterogeneous multi-search for subgoal-reduction

- Model-elimination (ME) prover
- Resolution engine (e.g., binary resolution, hyperresolution, unit-resulting resolution)
- Used to generate lemmas for ME
- Heuristics to pick best lemmas
- Provers: HPDS, CPTHEO

Multi-search for ordering-based strategies I

- Different search plans
  (e.g., different evaluation functions to select the given-clause)
- Derivation: $S_0^k \vdash S_1^k \vdash \ldots S_i^k \vdash \ldots$
  S_i^k: set of clauses at process p_k at stage i
- Communication:
  - Periodic resync: interleave search plans
  - Share heuristically chosen "good" clauses: combine search plans, "learning"
- Method and prover: Team-Work

Distributed search

Distributed search for ordering-based strategies

- All processes with the same inference system
- Distribute work: subdivide the data or the operations?
- Theorem proving: few inference rules, many clauses
- Subdivide the clauses
- Subdivision of inferences follow
- Notion of subdividing the search space
- Method: theorem proving by Clause-Diffusion

Distributed search: the Clause-Diffusion method

- Deductive processes $p_0, \ldots, p_{n-1}$ that are peers
- All p_j's get input problem, same inference system
- Basic version: also same search plan
- Asynchronous processes: sync on halt, e.g., one found proof
- Search space subdivided by a notion of ownership of clauses: every clause is owned by a process

Clause-Diffusion derivation

- $(O_0; NO_0)^j \vdash (O_1; NO_1)^j \vdash \ldots (O_i; NO_i)^j \vdash \ldots$
- $\forall p_j,\ 0 \leq j \leq n-1,\ \forall i,\ i \geq 0$:
  - O_i^j is the set of clauses owned by p_j
  - NO_i^j is the set of clauses not owned by p_j
  - $S_i^j = O_i^j \uplus NO_i^j$ is the local database of clauses at p_j
  - $S_0^0 = S$ input set: p_0 reads the input
  - $\bigcup_{j=0}^{n-1} S_i^j$ is the global database at stage i
  - Every clause is owned by a process: $\bigcup_{j=0}^{n-1} O_i^j = \bigcup_{j=0}^{n-1} S_i^j$
    And only one: $O_i^j \cap O_i^k = \emptyset$ (exceptions in practice)

Subdivision and diffusion of clauses I

- p_j reads or generates ψ by expansion or backward contraction
- Forward contraction: $\varphi = \psi\downarrow$
- p_j determines owner p_k of φ by an allocation criterion
- Say φ is the m-th clause generated by p_j
- φ's id: $\langle k, m, j\rangle$ globally unique
- $k = j$: φ enters O^j
- $k \neq j$: φ enters NO^j

Subdivision and diffusion of clauses II

- p_j applies φ to backward-contract clauses in S^j
- p_j broadcasts inference message $\langle \varphi, k, m, j \rangle$
- p_q, $q \neq j$, receives $\langle \varphi, k, m, j \rangle$
- Forward contraction: $\alpha = \varphi\downarrow$
- $k = q$: α enters O^q
- $k \neq q$: α enters NO^q
- p_q applies α to backward-contract clauses in S^q

Clause Diffusion: allocation criteria I

- Round-robin
- Input clauses by round-robin then work-load based
  - Measured as number of generated clauses
  - Estimated based on inference messages
- Syntax-based: weight-based
- Variant of any of these: assign a fixed fraction to self

Clause Diffusion: allocation criteria II

- Try to minimize the overlap of the searches by $p_0, \ldots, p_{n-1}$
- Each φ carries id's of parents for proof reconstruction
- Ancestor-graph oriented (AGO) heuristics, e.g.:
  - Input clauses by round-robin then by majority
  - Assign φ to the process that owns the most of its ancestors

Clause Diffusion: subdivision of inferences

- No subdivision of forward-contraction inferences
- No subdivision of backward-contraction inferences that delete clauses (e.g., subsumption)
- Subdivision of expansion inferences:
  p_j performs the inference if it owns the clause paramodulated or superposed into or the negative-literal parent in resolution
- Subdivision of backward-contraction inferences that simplify clauses: $\psi \in S^j$ can backward-simplify $\varphi \in S^j$:
  p_j generates $\varphi\downarrow$ if it owns φ, only deletes φ otherwise

Distributed fairness

- Fairness of a distributed derivation
- Sufficient conditions: local fairness +
  broadcast eventually all persistent irredundant clauses
- Clause-Diffusion satisfies the second one eagerly because of distributed proof reconstruction

Distributed proof reconstruction

- Proof reconstruction at the end of a refutation
- Ordering-based strategies: save clauses deleted by backward contraction
- Proof reconstruction in a distributed derivation:
  - Make sure that whoever finds $\Box$ can do it alone
  - Sufficient condition:
    Broadcast eventually all clauses ever used as premises
- Otherwise: proof reconstruction in post-processing

Distributed global contraction

- If φ redundant w.r.t. the global database at some stage, φ recognized redundant eventually by every process
- If φ redundant in $\bigcup_{j=0}^{n-1} S_i^j$, for all p_j there is a stage l, $l \geq i$, such that φ redundant in S_l^j
- Guaranteed by broadcasting mechanism: global redundancy/contraction reduced to local
- Subdivision of backward contraction: All delete φ and only one generates $\varphi\downarrow$

Clause Diffusion: summary

- A methodology to turn a sequential ordering-based strategy into a distributed one
- Each process executes the sequential strategy, modified with subdivision of work and communication
- If the requirements for distributed fairness are met:
  if the sequential strategy is complete,
  so is the distributed one

The Clause-Diffusion provers I

- Aquarius:
  - Parallelization of Otter
  - PCN for message passing
  - Also multi-search (e.g., different heuristic evaluation functions)
- Peers:
  - Parallelization of code from Otter Parts Store
  - Equational theories possibly with AC function symbols
  - p4 for message passing
  - Pairs algorithm instead of given-clause algorithm

The Clause-Diffusion provers II

Peers-mcd:

- Parallelization of EQP
- Equational theories possibly with AC function symbols
- Blocking, Basic paramodulation
- MPI for message passing
- AGO allocation criteria
- Both given-clause and pairs algorithms

The first big proof: the Robbins theorem

- The Robbins conjecture: Robbins algebra are Boolean
  open in mathematics since 1933
  a challenge for theorem provers since 1990
- EQP proved the Robbins conjecture
- Peers-mcd exhibited super-linear speedup in, e.g.:
  - Two out of three parts of the Robbins proof and almost super-linear speedup in the third
  - The Levi commutator problem in group theory

The Clause-Diffusion provers III

- Peers-mcd: both distributed search and multi-search, distributed mode, multi-search mode, hybrid mode
- Different search plans: given-clause and pairs, different heuristic evaluation functions, different `pick-given-ratio`
- Moufang identities in alternative rings with cancellation laws built-in
- Peers-mcd.d proved them without cancellation laws, with super-linear speedup (w.r.t. EQP) in distributed and hybrid mode with hybrid doing best (no speed-up by multi-search)

Discussion

Lessons learned from experiments I

- Super-linear speed-up possible as sequential and distributed strategies generate different searches
- Fewer clauses generated, higher percentage of retained clauses, different proof
- Effective subdivision of the search space
- The searches by the p_k's do not overlap too much, the successful one finds a proof much sooner
- The proof is not necessarily smaller
- Sub-optimal sequential search plan

Lessons learned from experiments II

- Different search: irregular scalability
- As the point is not to use more computers to do the same steps, no guarantee of scalability
- The problem may not be hard enough to justify using more processes
- Oscillations: the subdivision of the search space depends on the number of processes
- Combining distributed search and multi-search may smooth this effect

Take-home message

- Ordering-based strategies: parallel search
  - Team-Work pioneered multi-search
  - Clause-Diffusion pioneered distributed search
- Parallel ATP compounds the complications of first-order reasoning with those of parallelism

Parallel ATP and parallel SAT-solving

- Distributed search $\sim$ Divide-and-conquer
- Multi-search $\sim$ Portfolio approach

Multi-search for parallel SAT-solving

- Different heuristics for decisions
- Different heuristics for restart
- Randomization

Distributed search for parallel SAT-solving

- Cube-and-conquer as an instance of satisfiability modulo assignment
- Communicating “good” learned clauses
- Activity-based heuristics “intensify” search

More theorem-proving strategies

- Semantically-guided strategies
- Goal-sensitive strategies
- Strategies that combine proof search and model search:
  - Model-based strategies: the state of the derivation contains a representation of a candidate partial model
  - Conflict-driven strategies: nontrivial inferences only to explain and solve conflicts between clauses and candidate model

Future: parallelism and model-based ATP?

- Instance-based strategies (e.g., Inst-Gen, MEC, SGGS)
- Strategies that hybridize tableaux and instance-generation (e.g., hypertableaux)
- SGGS: Semantically-Guided Goal-Sensitive reasoning: model-based and conflict-driven
- Strategies that generalize CDCL to EPR (e.g., NRCL, DPLL(SX)) or FOL (SGGS)

Thank you!