Automatic Parallelization and Granularity Control of Logic and Constraint Programs

Manuel Hermenegildo
`http://www.cliplab.org/~herme`

Departments of Computer Science

University of New Mexico
and Technical University of Madrid

The work presented is a joint effort of the following members of the CLIP group at UNM and UPM: Francisco Bueno, Daniel Cabeza, Manuel Carro, Amadeo Casas, Manuel Hermenegildo, Pedro López, and Germán Puebla.

Supported by several CICYT/MCyT/MEC grants and EU FP-4/5/6 projects.

Introduction / Motivation

- Parallel Processing: high performance / reasonable cost.
- Finally coming of age:
  - Multiprocessor servers, clusters w/high-speed interconnect, ...
  - *Multicore architectures.*

  Not only HPC, but also mainstream systems, even laptops!
- Ideal situation: *Conventional Program* + *Multiprocessor* = *Higher Perf.*
  $\longrightarrow$ (Mostly) automatic parallelization.
- But many challenges:
  - Detecting independent tasks (often hidden by coding style).
    (even if large, irregular executions with pointers and dynamic data structures).
  - Efficient dynamic task scheduling.
  - Parallelization across procedure calls and modules.
  - Ensuring speedup: granularity control, speculation control, etc.

LP and CLP From the Parallelism Point of View

- Interesting from the automatic parallelization point of view:
  - program close to problem description
    $\longrightarrow$ less hiding of intrinsic parallelism
  - well understood mathematical foundation
    $\longrightarrow$ simplifies formal treatment
  - relative purity (well behaved variable scoping, fewer side-effects, generally single assignment)
    $\longrightarrow$ more amenable to automatic parallelization.
  - irregular computations; complex data structures; dynamic memory management; (well behaved) pointers; speculation; search...
    $\longrightarrow$ *real challenges!*
- Interesting techniques used (conditional dep. graphs, abstract interpretation w/interesting domains, cost analysis, dynamic sched. and load balancing, ...)

  (+ high programmer productivity and quite good performance!)

Some Early Design Choices

- Objective: *(More or less) conventional Program + Multiprocessor = Higher Perf.*
- Design decisions (&-Prolog, Aurora, etc., mid 80's):
  - Seek *speed* vs. speedup (beat best seq. execution; remember Amdahl's law).
  - Preserve standard semantics and cost model.
  - Parallel abstract machines derived from the best sequential ones.
    (No graph machines, no dataflow, no "cell" machines, no silver bullets, ...)
  - Platform: SMPs (did lots of work on coherent caches), COMAs, ...
    Later, NUMAs (but, with extensive compiler or programmer support).
  - Language (&-Prolog/Ciao):
    - Does not hide parallelism: allows automatic parallelization.
    - Allows parallelizing by hand (parallel operators, parallel HO, etc.)
  - Compiler & abstract machine:
    - Work hard on sequential performance to match best sequential compiler.
    - Work hard on automatic parallelization and granularity control.

    $\rightarrow$ developed extensive program analysis technology (abstract interpretation).

Parallelism in (Constraint) Logic Programs

- *Or-parallelism:* execute simultaneously different branches of the search space.

  Present in general search problems, enumeration part of constraint problems, etc.

- *And-parallelism:* execute simultaneously different statements or procedure calls.

  → Traditional parallelism (e.g., loop parallelization, task parallelism, divide and conquer, etc.).

```
fib(0, 0).
fib(1, 1).
fib(N, F1+F2) :-
      N>1, F1>=0, F2>=0,
      fib(N-1, F1) &
      fib(N-2, F2).
```

```
qsort([X|L],R) :-
        partition(L,X,L1,L2),
        qsort(L2,R2) &
        qsort(L1,R1),
        append(R1,[X|R2],R).
```

- Explicit vs. implicit: <u>both!</u> (+ source to source transformation.)

Parallelism: Correctness and Efficiency (“No Slowdown”)

- *Correctness:* “same” solutions as sequential execution.
- *Efficiency:* taking a shorter or equal execution time (*speedup*) or, at least, *no-slowdown* over state-of-the-art sequential systems.
- Imperative (a), functional (b), constraint logic programming (c):

s_1	`Y := W+2;`	`(+ (+ W 2)`	`Y = W+2,`
s_2	`X := Y+Z;`	`Z)`	`X = Y+Z,`
	(a)	(b)	(c)

- Constraint programming (with choices):

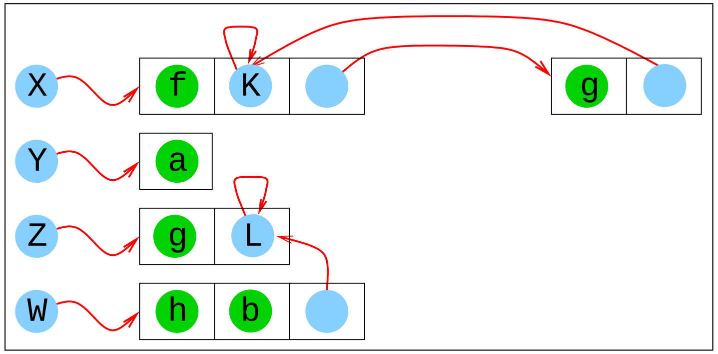
`main:-`	`p(X) :- X=a.`
s_1 `p(X),`	`q(X) :- X=b,` *large computation*.
s_2 `q(X),`	`q(X) :- X=a.`
`write(X).`	

- Fundamental issue: `p` *affects* `q` (prunes its choices); `q` ahead of `p` is *speculative*.
- Dependent vs. independent &-parallelism: just granularity level!

Independence – Strict Independence

- *Independence:* conditions that run-time behavior of parallel tasks must satisfy to guarantee correctness and efficiency.
- Interesting notions of independence developed.
  We assume ideal conditions (no parallelization overhead) in a first stage.
- Early result (*strict independence* [84-89]): correctness and efficiency (search space preservation) guaranteed for `p & q` if there are no "pointers" from `p` to `q`.

```
main :- X=f(K,g(K)), Y=a,
        Z=g(L), W=h(b,L),
        ---------------------->
        p(X,Y),
        q(Y,Z),
        r(W).
```



`p` and `q` are strictly independent, but `q` and `r` are not.

- In the end: pointer / shape analysis (but slightly more civilized case).

Independence – Strict Independence (Contd.)

- Not always possible to determine locally/statically:

```
main :- t(X,Y),       p(X), q(Y).

main :- read([X,Y]), p(X), q(Y).
```

- Alternatives: run-time independence tests, global analysis, ...

```
main :- read([X,Y]), ( indep(X,Y) -> p(X) & q(Y)
                                   ;  p(X) , q(Y) ).

main :- t(X,Y), p(X) & q(Y).      %% (After analysis)
```

Independence – Non-Strict Independence

- NSI [88-92]: only one thread "touches" each shared variable. Example:

  ```
  main :- t(X,Y), p(X), q(Y).
  ```

  ```
  t(X,Y) :- Y = f(X).
  ```

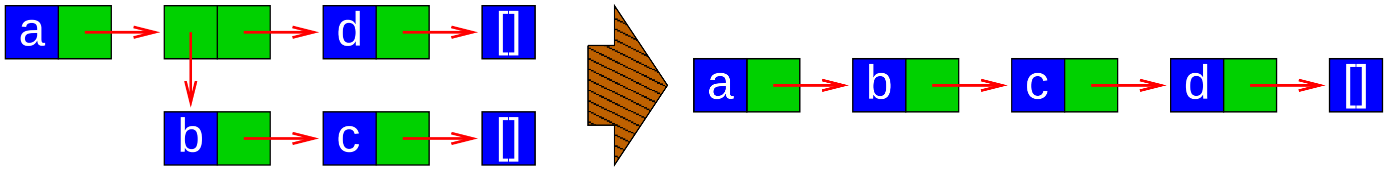
  `p` is independent of `t` (but `p` and `q` are dependent).

- Requires global analysis.

- Very important in programs using "incomplete structures."

  ```
  flatten(Xs,Ys) :- flatten(Xs,Ys,[]).
  ```

  ```
  flatten([], Xs, Xs).
  flatten([X|Xs],Ys,Zs)  :- flatten(X,Ys,Ys1), flatten(Xs,Ys1,Zs).
  flatten(X, [X|Xs], Xs) :- atomic(X), X \== [].
  ```



Independence – Constraint Independence

- Standard Herbrand notions do not carry over to general constraint systems.

```
main :- X > Y, Z > Y, p(X) & q(Z), ...
main :- X > Y, Y > Z, p(X) & q(Z), ...
```

- General notion [91-94]: "all constraints posed by second thread are consistent with output constraints of first thread." (Better also for Herbrand!)
- Sufficient a-priori condition: given $g_1(\bar{x})$ and $g_2(\bar{y})$:

$$(\bar{x} \cap \bar{y} \subseteq def(c)) \ and \ (\exists_{-\bar{x}} c \wedge \exists_{-\bar{y}} c \rightarrow \ \exists_{-\bar{y} \cup \bar{x}} c)$$

  ($def(c)$ is the set of variables constrained to a unique value in c)
- For $c = \{x > y, z > y\}$ $\quad \bar{\exists}_{-\{x\}} c = \bar{\exists}_{-\{z\}} c = \bar{\exists}_{-\{x,z\}} c = true$
  For $c = \{x > y, y > z\}$ $\quad \bar{\exists}_{-\{x\}} c = \bar{\exists}_{-\{z\}} c = true, \quad \bar{\exists}_{\{x,z\}} c = x > z$
- Approximation: presence of "links" through the store.

An Actual System: `Ciao` (&-Prolog's Successor)

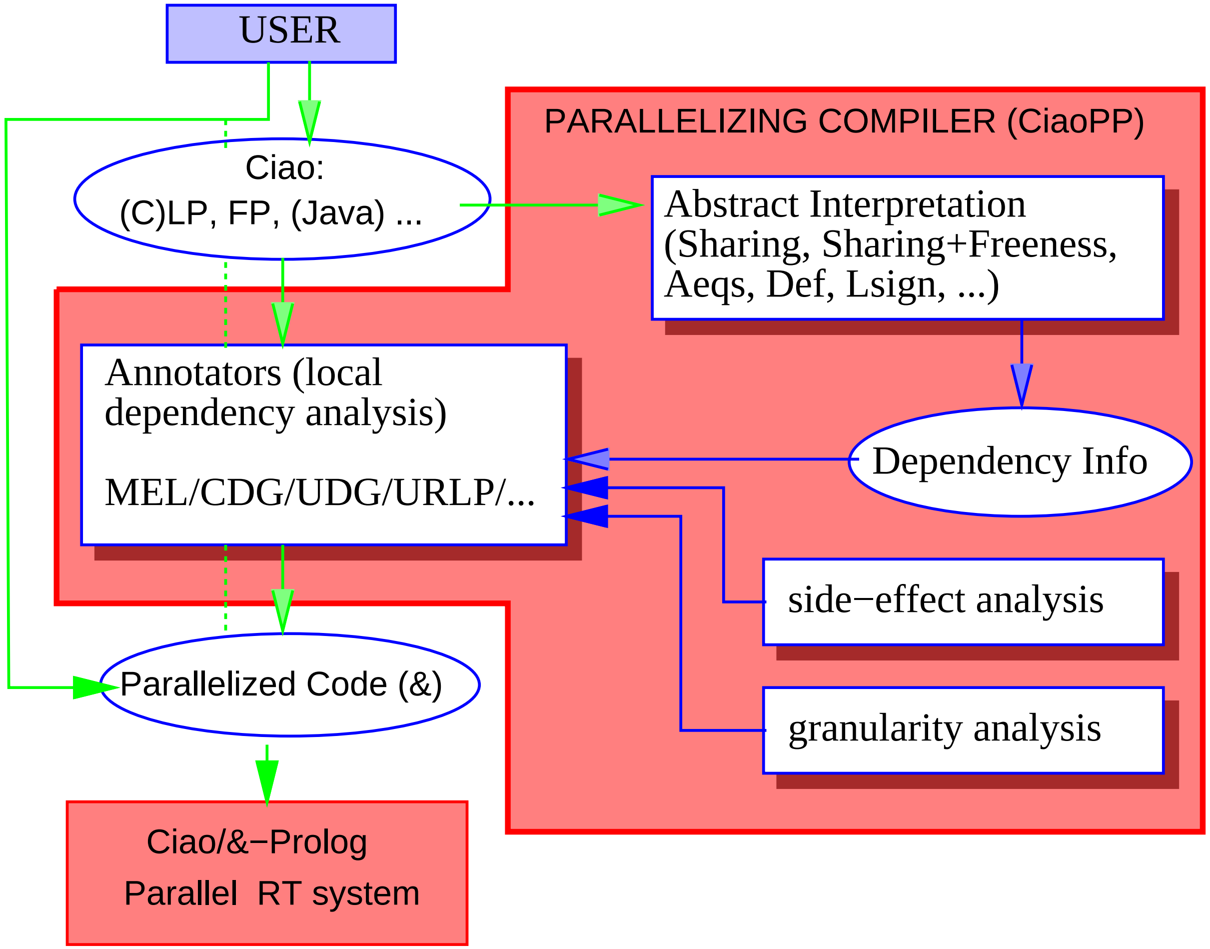
- One of the popular Prolog/CLP systems (supports ISO-Prolog fully).
- At the same time, new-generation *multi-paradigm* language/prog.env. with:
  - Predicates, functions, constraints, higher-order, objects, ...
  - Assertion language for expressing rich program properties.
  - Several control rules (e.g., Andorra).
  - Parallel, concurrent, and distributed execution primitives.
  - Compile-time and run-time tools (`CiaoPP`) for:
    - Automatic parallelization.
    - Resource control.

    + static debugging, verifi cation, program certifi cation, PCC, ...

    All based on modular, incremental, polyvariant *abstract interpretation* and *specialization*.

+ "Industry standard" performance, Robust module/object system, Separate/incremental compilation, (Semi-automatic) interfaces to other languages, databases, etc. Program development environment, LGPL license, ...

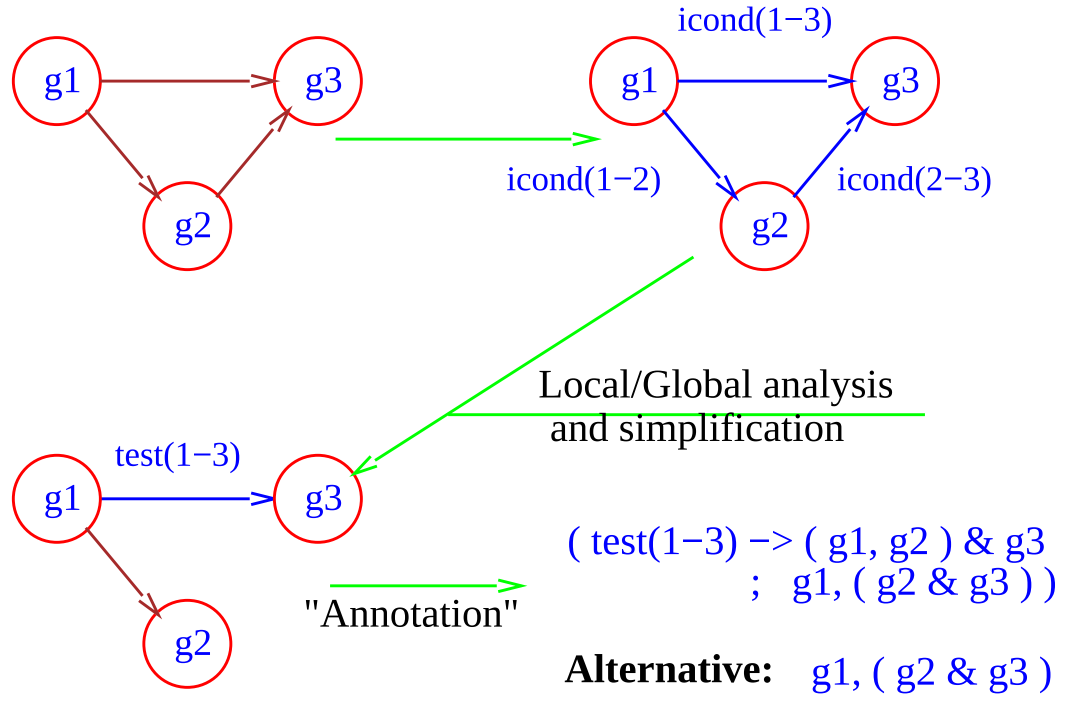
Ciao Parallelizer Overview ("&-Prolog")



Parallelization Process

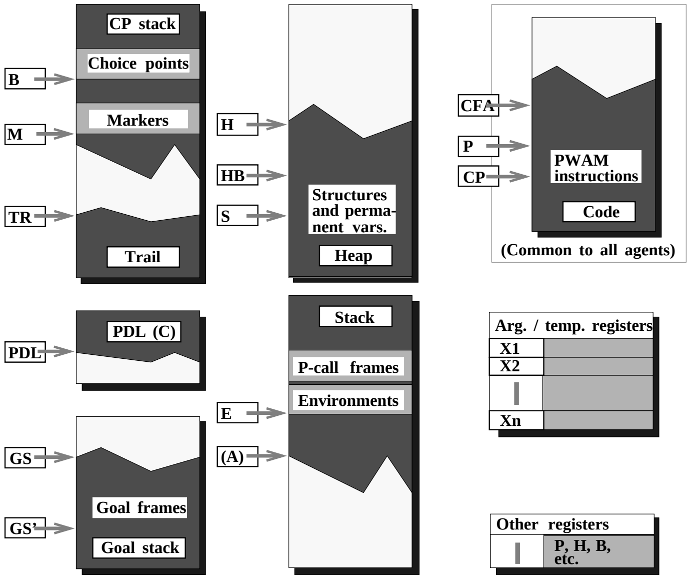
- Conditional dependency graph (of some segment, e.g., a clause):
  - vertices are possible tasks (statements, calls,...),
  - edges=possible dependency (labels=conditions needed for independence).
- Local or global analysis used to reduce/remove checks in the edges.

```
foo(...) :-
    g1(...),
    g2(...),
    g3(...).
```



Parallel Run-time System: PWAM architecture

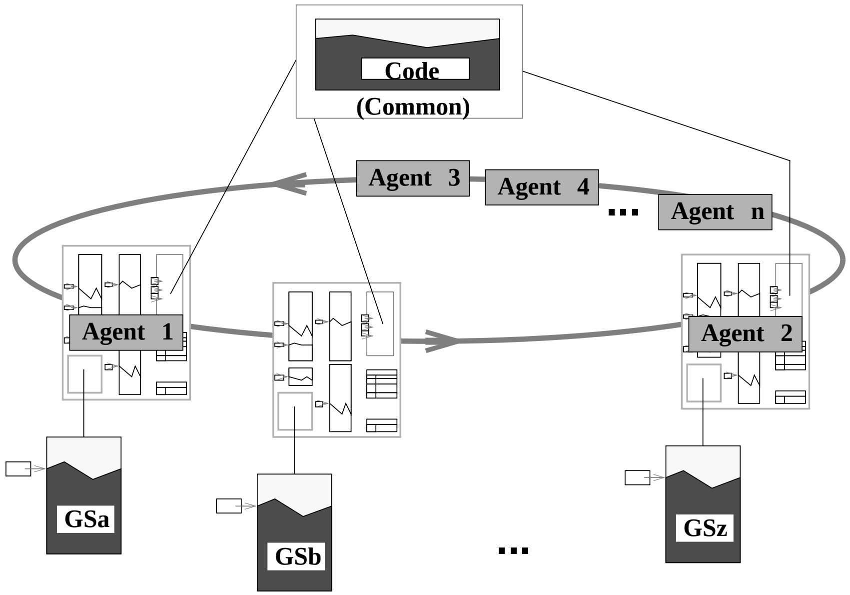
- First Multisequential Model:
  Parallel version of the Warren Abstract Machine (WAM)
- Defined as storage model + instruction set
- First proposal obtaining speedup over state of the art sequential systems.



PWAM Storage Model: A Stack Set

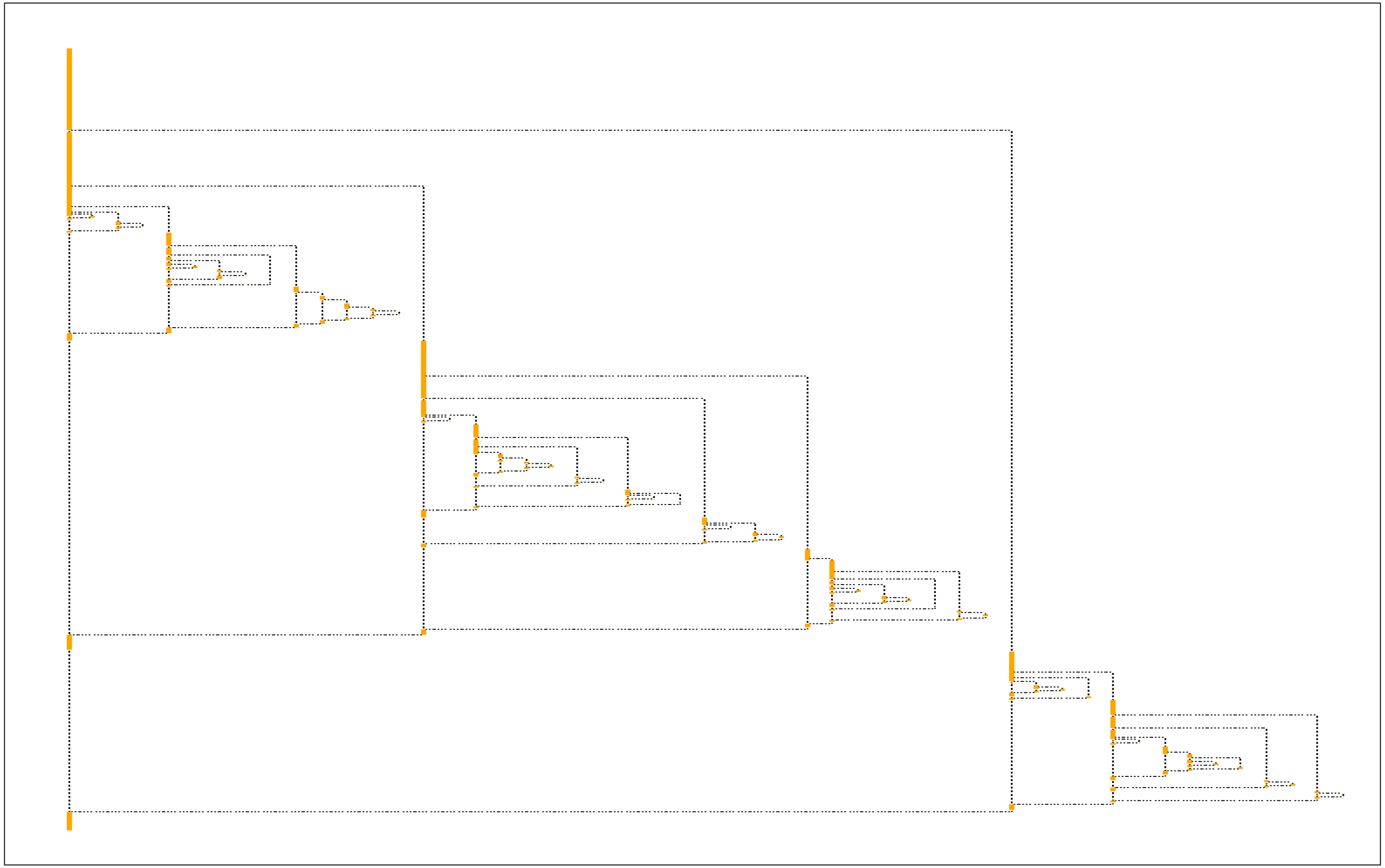
PWAM Run-time System: Agents and Stack Sets

- Dynamic creation/deletion of Stack Sets and Agents.
- Lazy, on demand, (distributed goal stealing based-) scheduling.



- Extensions / optimizations:
  - DASWAM / DDAS System (dependent and-//) [w/Shen]
  - &ACE, ACE Systems (or-, and-, dep-//) [w/Gupta and Pontelli]

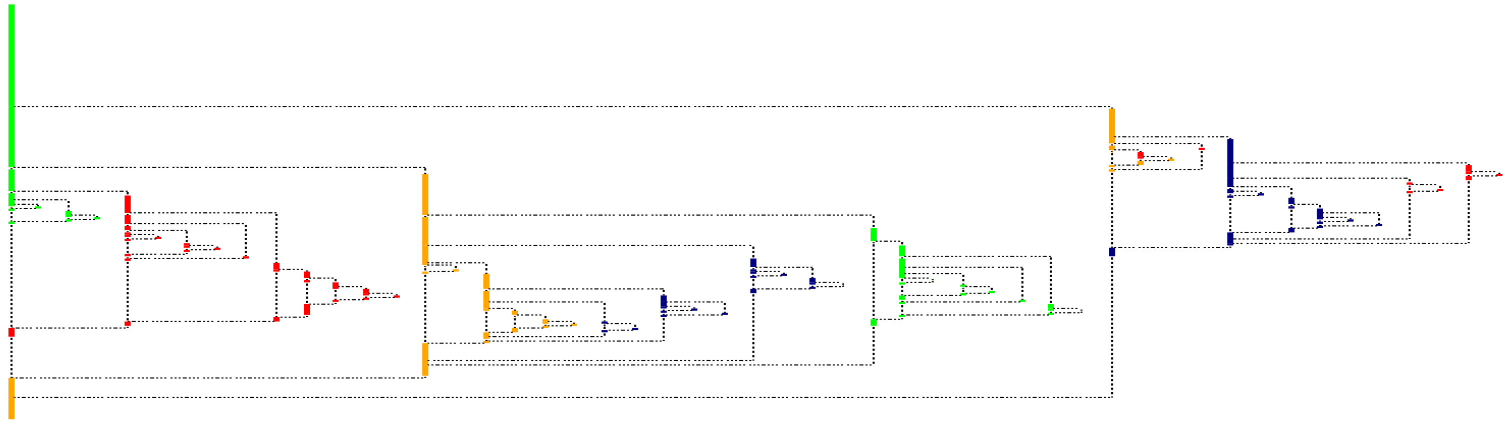
Visualization of And-parallelism - (small) qsort, 1 processor



Visualization of And-parallelism (some explanations)

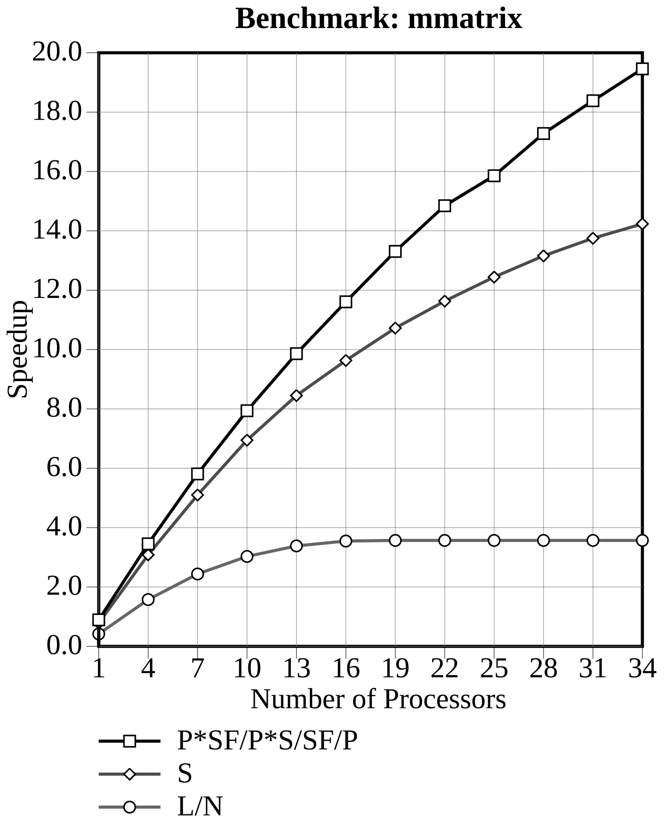
- y axis is time. t_0 is at top of picture. End of the execution at bottom.
- A task fork is represented by a dashed horizontal line.
- Actual task processing represented by a colored line, color is id. of processor performing task.
- Task wait times (e.g., task is available but no processor has picked it up yet) depicted by vertical dotted lines.
- E.g., in `qsort` in the previous slide:
  - First vertical line is the first partition, being done by the orange processor.
  - This forks into two calls to qsort:
    - The left task is taken by the orange processor.
    - Right one available for execution but no other processor to pick it up. Eventually picked up by orange processor after finishing leftmost task (and its subtasks).
  - The small tasks after the joins are the calls to `append`.

Visualization of And-parallelism - (small) qsort, 4 processors

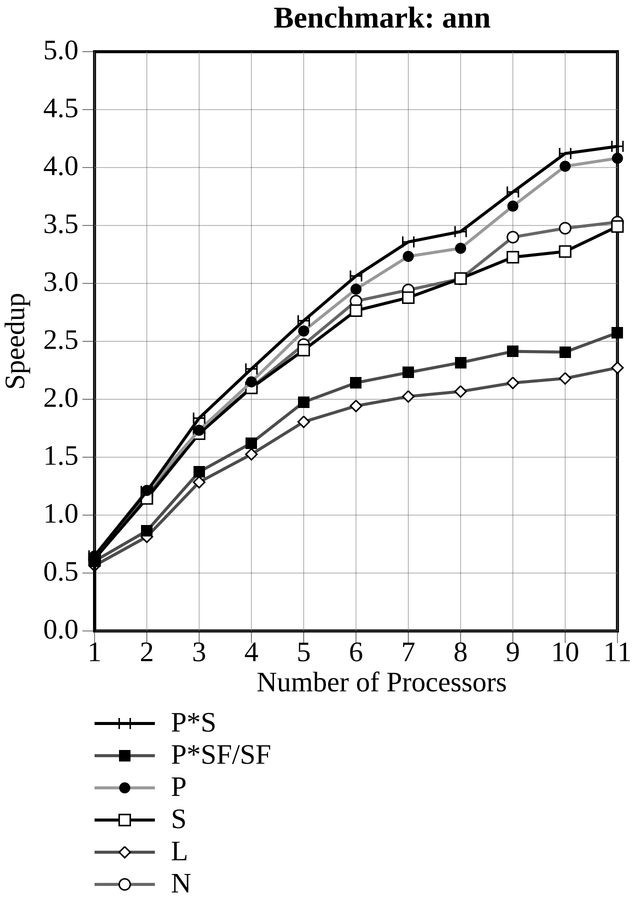


- Speedup!
- Dependent and-parallelism will overlap partition and qsort.

Some Speedups (for different analysis abstract domains)



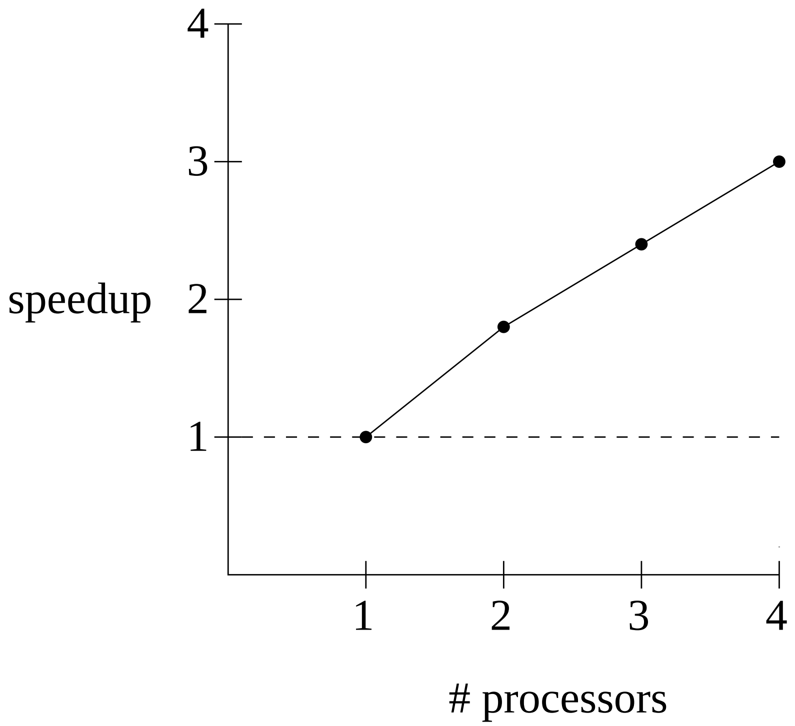
Matrix multiplication



The parallelizer, self-parallelized

Some CLP Results (Run-time System)

- Speedup for `critical` with `go3` input



Dealing with Overheads, Irregularity

- Independence not enough:
  overheads (task creation and scheduling, communication, etc.)
- In symbolic applications compounded because number and size of tasks highly irregular and dependent on run-time parameters.
- Dynamic solutions:
  - Minimize task management and data communication overheads
    (micro tasks, shared heaps, compile-time elimination of locks, ...)
  - Efficient dynamic task allocation (e.g., non-centralized task stealing)
- Quite good results for shared-memory multiprocessors early on
  (e.g., Sequent Balance 1986-89).
- Not sufficient for NUMAs, clusters, WS farms, GRIDs, etc.

Granularity Control

- Replace parallel execution with sequential execution or vice-versa based on bounds (or estimations) on grain size and overheads.
- In general cannot be done (well) completely at compile-time: cost often depends on input (difficult to approximate at compile time, even w/abstract interpretation).

```
..., inc_all(X,Y) & r(Z,M), ...
```

- Our approach:
  - Derive at compile-time *functions* (to be evaluated at run-time) that efficiently approximate task size (lower, upper *bounds*).
  - Transform programs to carry out run-time granularity control.
- Example (assuming threshold is 100 units):

```
..., ( 2*length(X)+1 > 100 -> inc_all(X,Y) & r(Z,M)
                            ;  inc_all(X,Y) , r(Z,M) ), ...
```

- Provably correct techniques (thanks to abstract interpretation)
  Can *ensure speedup*.

Size and Cost Inference in `CiaoPP`

- Upper and lower bounds on argument sizes and procedure cost:
  1. Perform type and mode inference, infer size measures.
  2. Use data dependency graphs to determine the relative sizes of variable bindings at different program points.
  3. Use the size information to set up recurrence equations representing the computational cost of procedures.
  4. Compute lower/upper bounds to the solutions of these recurrence equations to obtain bounds on task granularities.
  5. Non-failure (absence of exceptions) information needed for lower bounds.

Size and Cost Bounds Inference in CiaoPP (Contd.)

E.g., for inc_all:

- Measure (from type/mode inference): list length.
- Argument size relations:

  $\mathtt{Size}^{2}_{\mathtt{inc\_all}}(0) = 0$ (boundary condition from base case),

  $\mathtt{Size}^{2}_{\mathtt{inc\_all}}(n) = 1 + \mathtt{Size}^{2}_{\mathtt{inc\_all}}(n-1)$.

  Sol = $\mathtt{Size}^{2}_{\mathtt{inc\_all}}(n) = n$.

- Procedure cost relations:

  $\mathtt{Cost}^{\mathrm{L}}_{\mathtt{inc\_all}}(0) = 1$ (boundary condition from base case),

  $\mathtt{Cost}^{\mathrm{L}}_{\mathtt{inc\_all}}(n) = 1 + \mathtt{Cost}^{\mathrm{L}}_{\mathtt{inc\_all}}(n-1)$.

  Sol = $\mathtt{Cost}^{\mathrm{L}}_{\mathtt{inc\_all}}(n) = 2\,n + 1$.

Granularity Control: Some Optimizations

- Simplification of cost functions:

```
..., ( length(X) > 50 -> inc_all(X,Y) & r(Z,M)
                       ;  inc_all(X,Y) , r(Z,M) ), ...
```

```
..., ( length_gt(LX,50) -> inc_all(X,Y) & r(Z,M)
                          ;  inc_all(X,Y) , r(Z,M) ), ...
```

- Complex thresholds: use also communication cost functions, load, ...
  Example: Assume $CommCost(inc\_all(X)) = 0.1\ (length(X) + length(Y))$.
  We know $ub\_length(Y)$ (actually, exact size) $= length(X)$; thus:

$$2\ length(X) + 1 > 0.1\ (length(X) + length(X)) \cong$$
$$2\ length(X) > 0.2\ length(X) \equiv$$
$$2 > 0.2$$

  $\Rightarrow$ Guaranteed speedup for any data size.

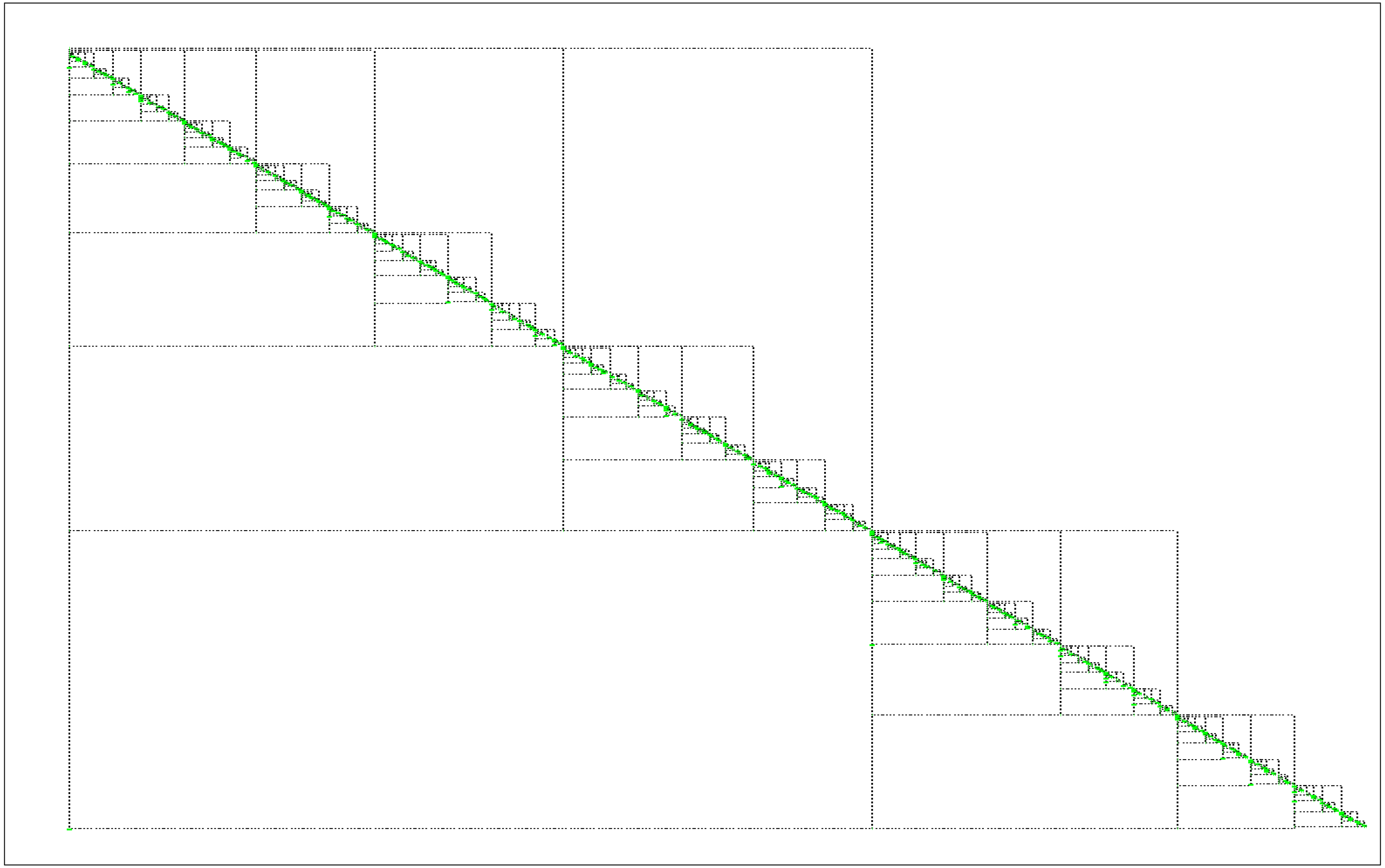
- Data size computations can often be done on-the-fly.
- Static task clustering (loop unrolling).
- Static placement, etc.

Granularity Control System Output Example

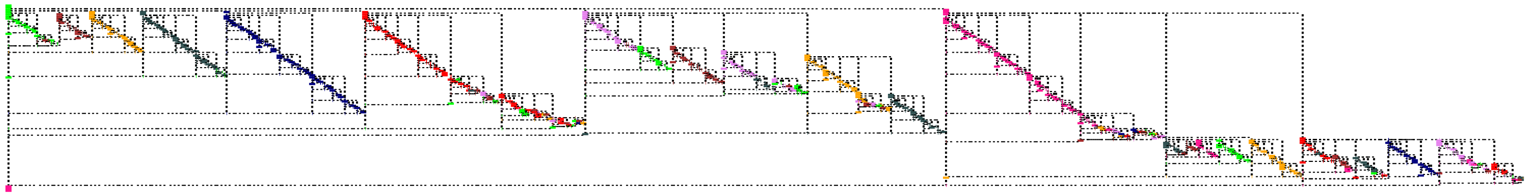
```
g_qsort([], []).
g_qsort([First|L1], L2) :-
  partition3o4o(First, L1, Ls, Lg, Size_Ls, Size_Lg),
  Size_Ls > 20 -> (Size_Lg > 20 -> g_qsort(Ls, Ls2) & g_qsort(Lg, Lg2)
                                 ;  g_qsort(Ls, Ls2) , s_qsort(Lg, Lg2))
               ;  (Size_Lg > 20 -> s_qsort(Ls, Ls2) , g_qsort(Lg, Lg2)
                                 ;  s_qsort(Ls, Ls2) , s_qsort(Lg, Lg2))),
  append(Ls2, [First|Lg2], L2).

partition3o4o(F, [], [], [], 0, 0).
partition3o4o(F, [X|Y], [X|Y1], Y2, SL, SG) :-
  X =< F, partition3o4o(F, Y, Y1, Y2, SL1, SG), SL is SL1 + 1.
partition3o4o(F, [X|Y], Y1, [X|Y2], SL, SG) :-
   X > F, partition3o4o(F, Y, Y1, Y2, SL, SG1), SG is SG1 + 1.
```

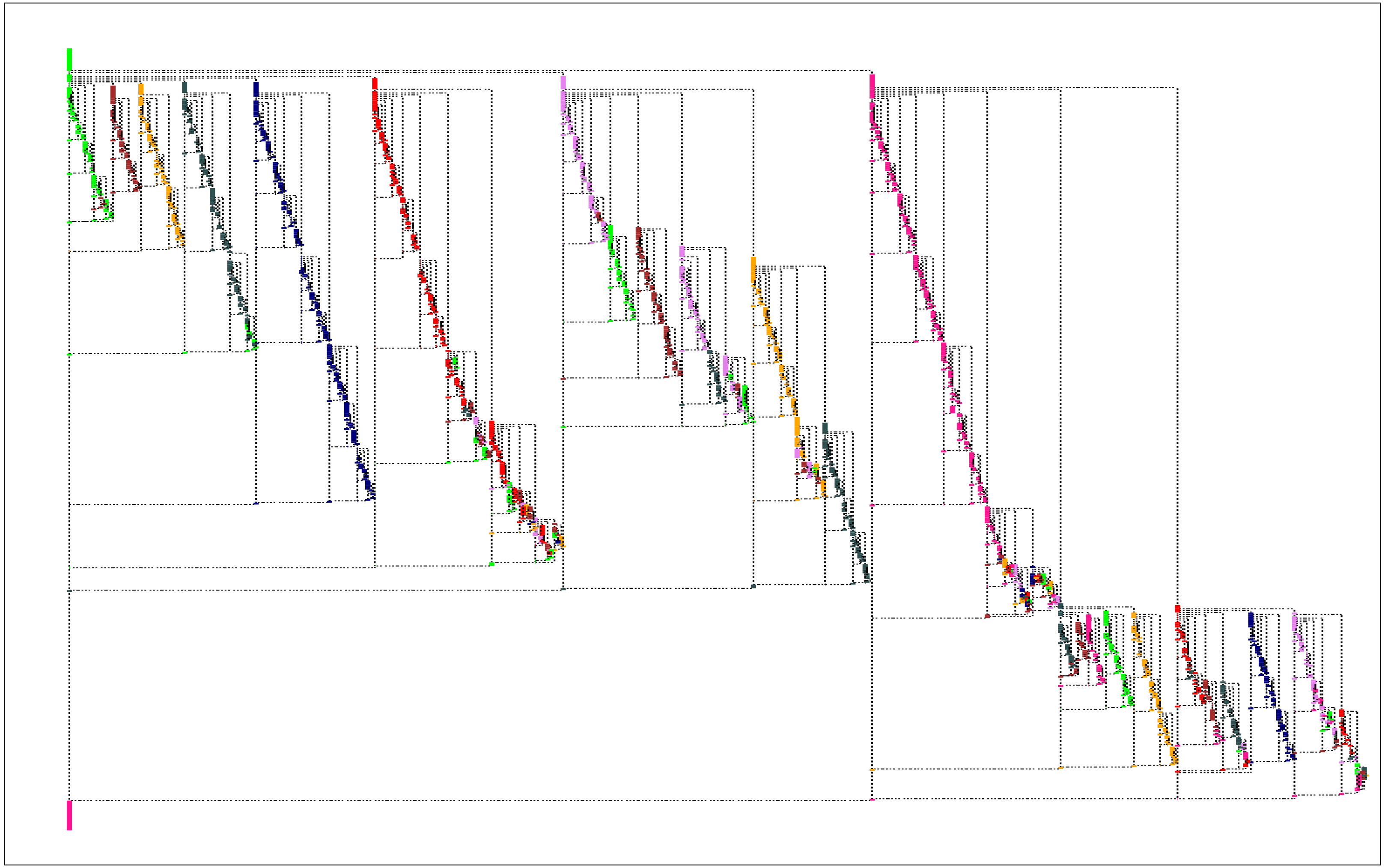
Fib 15, 1 processor



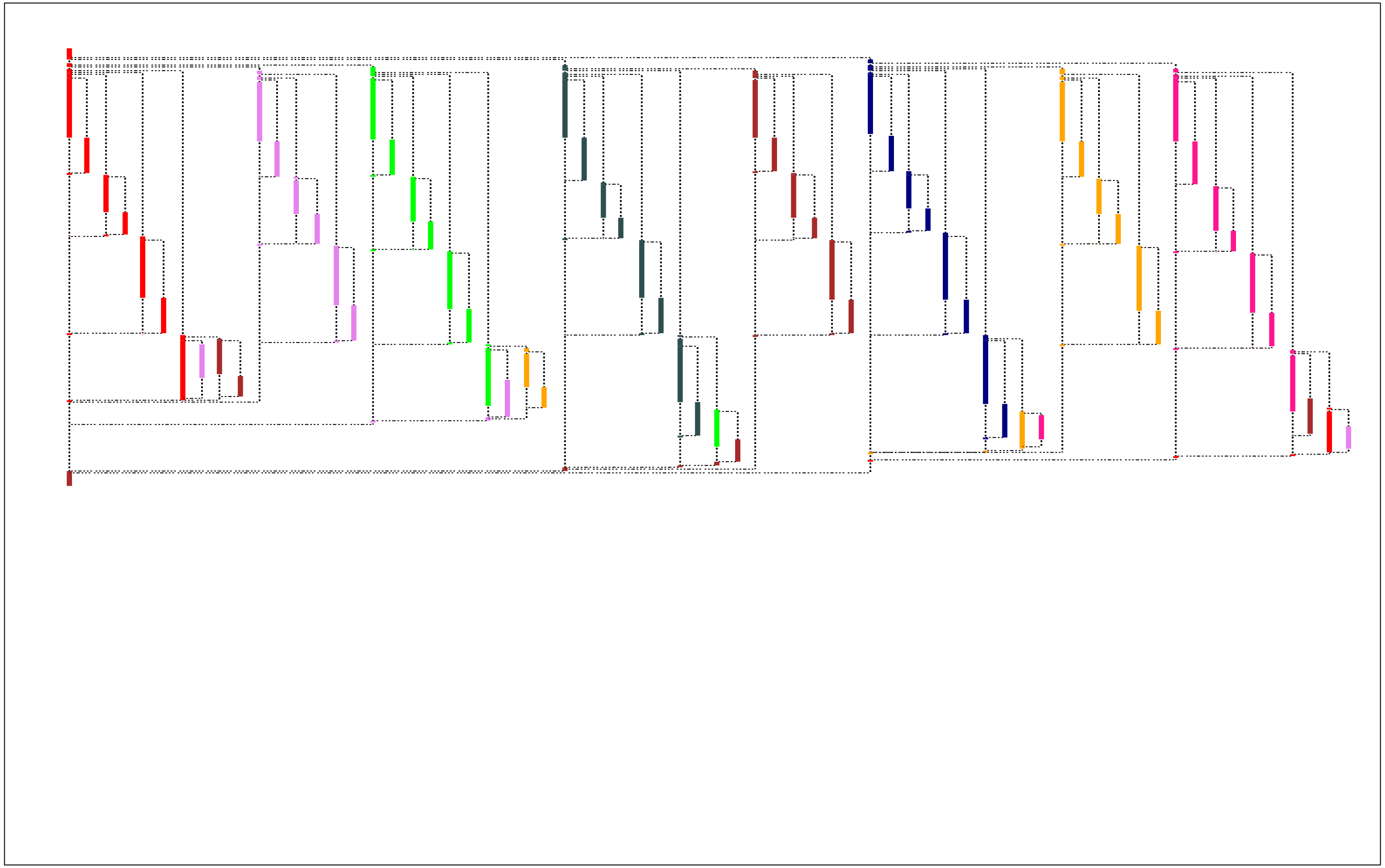
Fib 15, 8 processors (same scale)



Fib 15, 8 processors (full scale)



Fib 15, 8 processors, with granularity control (same scale)



Granularity Control: Experimental Results

- Shared memory:

programs	seq. prog.	no gran.ctl	gran.ctl	gc.stopping	gc.argsize
fi b(19)	1.839	0.729	1.169	0.819	0.549
		1	-60%	-12%	+24%
hanoi(13)	6.309	2.509	2.829	2.399	2.399
		1	-12.8%	+4.4%	+4.4%
unbmatrix	2.099	1.009	1.339	0.870	0.870
		1	-32.71%	+13.78%	+13.78%
qsort(1000)	3.670	1.399	1.790	1.659	1.409
		1	-28%	-19%	-0.0%

.

- Cluster:

programs	seq. prog.	no gran.ctl	gran.ctl	gc.stopping	gc.argsize
fi b(19)	1.839	0.970	1.389	1.009	0.639
		1	-43%	-4.0%	+34%
hanoi(13)	6.309	2.690	2.839	2.419	2.419
		1	-5.5%	+10.1%	+10.1%
unbmatrix	2.099	1.039	1.349	0.870	0.870
		1	-29.84%	+16.27%	+16.27%
qsort(1000)	3.670	1.819	2.009	1.649	1.429
		1	-11%	+9.3%	+21%

.

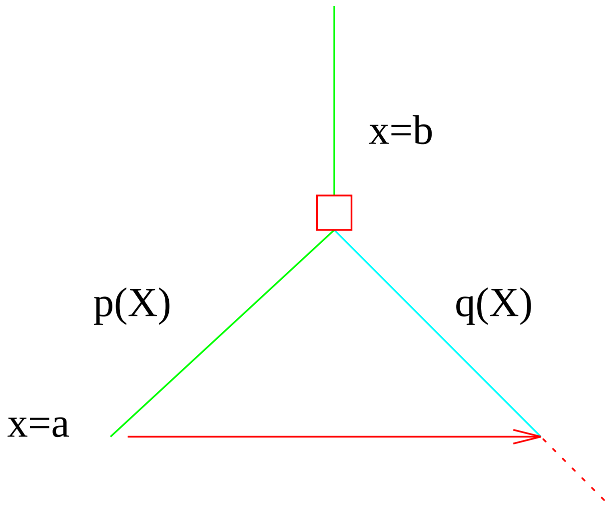
Dealing with Speculation

- Computations can be speculative (or even nonterminating!):

```
foo(X) :- X=b, ..., p(X) & q(X), ...
foo(X) :- X=a, ...

p(X) :- ..., X=a, ...

q(X) :- large computation.
```



  but “no slow-down” guaranteed if

  - left-biased scheduling,
  - instantaneous killing of siblings (failure propagation).

- Left biased schedulers, dynamic throttling of speculative tasks, etc.
- Static detection of non-failure:
  avoids speculativeness / guarantees theoretical speedup
  → *importance of non-failure analysis* (also determinacy).

Wrap-up: strong points

- Several generations of parallelizing compilers for LP and CLP [85-present]:
  - Good compilation speed, proved correct and efficient.
  - Obtained speedups over state-of-the-art sequential systems on applications.
  - Including granularity control.

  Improved on hand parallelizations on several large applications.
- Areas of particularly good progress:
  - Concepts of independence (pointers, search/speculation, constraints...).
  - Inter-proc. & modular anal. (w/recursion, shapes, pointers/aliasing, cost, etc.).
  - Parallelization algorithms for *conditional* dependency graphs.
  - Dealing with irregularity:
    - efficient task representation and fast dynamic scheduling,
    - static inference of task cost functions – granularity control.
  - Mixed static/dynamic parallelization techniques.
  - Applied also to other paradigms (functional, objects, imperative).

Wrap-up: plans and architectural issues

- Areas of improvement:
  - Combine parallelization with extensive optimizations (specialization, low-level optimizations) $\rightarrow$ “run at C speed and in parallel.”
  - Support finer and finer grains of independence.
  - Improve independence detection for structure traversals based on integer arithmetic $\rightarrow$ using, e.g., polyhedra domains.
  - Improve combination of different types of parallelism.
  - Add further support for more implicit dynamic parallelism (e.g., Andorra-styles).
  - Improve treatment of mutating data structures (now done via SSA).
- Architectural lessons:
  - SMPs, or something that behaves like them! (COMAs, etc.)
  - Coherent caches.
  - Fast locks, communication; support for boxing, unboxing, etc.; ...
  - Beware of Amdahl’s law: we need uniprocessor performance.

Selected Bibliography

- Survey/Tutorials/Philosophical (on parallelism):

  - M. Hermenegildo, R. Warren. Designing a High-Performance Parallel Logic Programming System. Computer Architecture News, Special Issue on Parallel Symbolic Programming, Vol. 15, Num. 1, pages 43-53, ACM, March 1987.
  - M. Hermenegildo. Parallelizing Irregular and Pointer-Based Computations Automatically: Perspectives from Logic and Constraint Programming. Parallel Computing, Vol. 26, Num. 13-14, pages 1685-1708, Elsevier Science, December 2000. (Paper corresponding to Europar'97 invited talk.) on line
  - G. Gupta, E. Pontelli, K. Ali, M. Carlsson, M. Hermenegildo. Parallel Execution of Prolog Programs: a Survey. ACM Transactions on Programming Languages and Systems, Vol. 23, Num. 4, pages 472-602, ACM Press, July 2001. on line

- The Ciao multiparadigm programming system:

  - Manuel V. Hermenegildo, Germán Puebla, Francisco Bueno, Pedro López-García. Integrated Program Debugging, Verifi cation, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). Science of Computer Programming, Vol. 58, Num. 1-2, pages 115-140, Elsevier Science, October 2005. (Paper corresponding to SAS'03 invited talk.) on line
  - F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, G. Puebla (Eds.). The Ciao Multiparadigm Programming System. Reference Manual (V1.10). August 2004. System and on-line version of the manual available at http://clip.dia.fi .upm.es/Software/Ciao. 1088 pages.

Selected Bibliography (cont.)

- Automatic parallelization:

  - K. Muthukumar, M. Hermenegildo. The CDG, UDG, and MEL Methods for Automatic Compile-time Parallelization of Logic Programs for Independent And-parallelism. Int'l. Conference on Logic Programming, pages 221-237, MIT Press, June 1990.
  - K. Muthukumar, F. Bueno, M. García de la Banda, M. Hermenegildo. Automatic Compile-time Parallelization of Logic Programs for Restricted, Goal-level, Independent And-parallelism. Journal of Logic Programming, Vol. 38, Num. 2, pages 165-218, Elsevier - North-Holland, February 1999. on line
  - G. Puebla, M. Hermenegildo. Abstract Multiple Specialization and its Application to Program Parallelization. J. of Logic Programming. Special Issue on Synthesis, Transformation and Analysis of Logic Programs, Vol. 41, Num. 2&3, pages 279-316, Elsevier - North Holland, November 1999. on line
  - M. Hermenegildo, M. Carro. Relating Data-Parallelism and (And-) Parallelism in Logic Programs. The Computer Languages Journal, Vol. 22, Num. 2/3, pages 143-163, Elsevier Science, July 1996. on line
  - M. García de la Banda, F. Bueno, M. Hermenegildo. Towards Independent And-Parallelism in CLP. Programming Languages: Implementation, Logics, and Programs, LNCS, Num. 1140, pages 77-91, Springer-Verlag, September 1996.

Selected Bibliography (cont.)

- Datafbw analysis for parallelization (detecting independence):

  - R. Warren, M. Hermenegildo, S. K. Debray. On the Practicality of Global Flow Analysis of Logic Programs. Fifth International Conference and Symposium on Logic Programming, pages 684-699, MIT Press, August 1988.
  - K. Muthukumar, M. Hermenegildo. Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation. 1989 North American Conference on Logic Programming, pages 166-189, MIT Press, October 1989. on line
  - F. Bueno, M. García de la Banda, M. Hermenegildo. Effectiveness of Abstract Interpretation in Automatic Parallelization: A Case Study in Logic Programming. ACM Transactions on Programming Languages and Systems, Vol. 21, Num. 2, pages 189-238, ACM Press, March 1999. on line
  - K. Muthukumar, M. Hermenegildo. Compile-time Derivation of Variable Dependency Using Abstract Interpretation. Journal of Logic Programming, Vol. 13, Num. 2/3, pages 315-347, Elsevier - North-Holland, July 1992. on line
  - K. Muthukumar, M. Hermenegildo. Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation. 1991 International Conference on Logic Programming, pages 49-63, MIT Press, June 1991. on line
  - D. Cabeza, M. Hermenegildo. Extracting Non-Strict Independent And-Parallelism Using Sharing and Freeness Information. 1994 International Static Analysis Symposium, LNCS, Num. 864, pages 297-313, Springer-Verlag, September 1994.

Selected Bibliography (cont.)

- Granularity control, resource awareness:
  - Inference of cost functions and data sizes, granularity control:

    - S.K. Debray, N.-W. Lin, M. Hermenegildo. Task Granularity Analysis in Logic Programs. Proc. of the 1990 ACM Conf. on Programming Language Design and Implementation, pages 174-188, ACM Press, June 1990.
    - P. López-García, M. Hermenegildo, S.K. Debray. Towards Granularity Based Control of Parallelism in Logic Programs. Proc. of First International Symposium on Parallel Symbolic Computation, PASCO'94, pages 133-144, World Scientifi c, September 1994. on line
    - P. López-García, M. Hermenegildo, S.K. Debray. A Methodology for Granularity Based Control of Parallelism in Logic Programs. Journal of Symbolic Computation, Special Issue on Parallel Symbolic Computation, Vol. 22, pages 715-734, Academic Press, 1996. on line
    - S.K. Debray, P. López-García, M. Hermenegildo, N.-W. Lin. Estimating the Computational Cost of Logic Programs. Static Analysis Symposium, SAS'94, LNCS, Num. 864, pages 255–265, Springer-Verlag, September 1994. (Paper corresponding to SAS'94 invited talk.) on line
    - S.K. Debray, P. López-García, M. Hermenegildo, N.-W. Lin. Lower Bound Cost Estimation for Logic Programs. 1997 International Logic Programming Symposium, pages 291-305, MIT Press, Cambridge, MA, October 1997. on line

Selected Bibliography (cont.)

- Granularity control, resource awareness (continued):
  - Related techniques (data sizes, nonfailure analysis, determinacy analysis):
    - P. López-García, M. Hermenegildo. Effi cient Term Size Computation for Granularity Control. International Conference on Logic Programming, pages 647-661, MIT Press, Cambridge, MA, June 1995. on line
    - S.K. Debray, P. López-García, M. Hermenegildo. Non-Failure Analysis for Logic Programs. 1997 International Conference on Logic Programming, pages 48-62, MIT Press, Cambridge, MA, June 1997. on line
    - F. Bueno, P. López-García, M. Hermenegildo. Multivariant Non-Failure Analysis via Standard Abstract Interpretation. 7th International Symposium on Functional and Logic Programming (FLOPS 2004), LNCS, Num. 2998, pages 100-116, Springer-Verlag, April 2004. on line
    - P. López-García, F. Bueno, M. Hermenegildo. Determinacy Analysis for Logic Programs Using Mode and Type Information. Proceedings of the 14th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'04), LNCS, Num. 3573, pages 19-35, Springer-Verlag, August 2005. on line
  - Other applications (code mobility/proof-carrying code):
    - M. Hermenegildo, E. Albert, P. López-García, G. Puebla. Abstraction Carrying Code and Resource-Awareness. Proc. of 7th ACM-SIGPLAN International Symposium on Principles and Practice of Declarative Programming (PPDP'05), 11 pages, ACM Press, July 2005. (Paper corresponding to invited talk.) on line
    - M. Hermenegildo, E. Albert, P. López-García, G. Puebla. Some Techniques for Automated, Resource-Aware Distributed and Mobile Computing in a Multi-Paradigm Programming System. Proc. of EURO-PAR 2004, LNCS, Num. 3149, pages 21-37, Springer-Verlag, August 2004. (Paper corresponding to invited talk.) on line
  - Other applications ("static performance debugging"):
    - (See SAS'03 invited talk paper in the section on the Ciao multiparadigm system, on line.)

Selected Bibliography (cont.)

- Notions of independence:
  - Independence in logic programs:
    - M. Hermenegildo, F. Rossi. Strict and Non-Strict Independent And-Parallelism in Logic Programs: Correctness, Effi ciency, and Compile-Time Conditions. Journal of Logic Programming, Vol. 22, Num. 1, pages 1-45, Elsevier - North Holland, 1995. on line
    - M. Hermenegildo, F. Rossi. On the Correctness and Effi ciency of Independent And-Parallelism in Logic Programs. 1989 North American Conference on Logic Programming, pages 369-390, MIT Press, October 1989.
  - Independence in constraint logic programs:
    - M. García de la Banda, M. Hermenegildo, K. Marriott. Independence in CLP Languages. ACM Transactions on Programming Languages and Systems, Vol. 22, Num. 2, pages 269-339, ACM Press, March 2000. on line
    - M. García de la Banda, M. Hermenegildo, K. Marriott. Independence in Constraint Logic Programs. 1993 International Logic Programming Symposium, pages 130-146, MIT Press, Cambridge, MA, October 1993.
  - Independence in (constraint) logic programs with delays (i.e., lazy execution, concurrency):
    - M. García de la Banda, M. Hermenegildo, K. Marriott. Independence in Dynamically Scheduled Logic Languages. 1996 International Conference on Algebraic and Logic Programming, LNCS, Num. 1139, pages 47-61, Springer-Verlag, September 1996.
    - F. Bueno, M. Hermenegildo, U. Montanari, F. Rossi. Partial Order and Contextual Net Semantics for Atomic and Locally Atomic CC Programs. Science of Computer Programming, Vol. 30, pages 51-82, North-Holland, January 1998. Special CCP95 Workshop issue. on line

Selected Bibliography (cont.)

- Parallel abstract machines and execution models:
  - M. Hermenegildo. An Abstract Machine for Restricted AND-parallel Execution of Logic Programs. Third International Conference on Logic Programming, Lecture Notes in Computer Science, Num. 225, pages 25-40, Springer-Verlag, Imperial College, July 1986.
  - M. Hermenegildo, R. I. Nasr. Effi cient Management of Backtracking in AND-parallelism. Third International Conference on Logic Programming, LNCS, Num. 225, pages 40-55, Springer-Verlag, Imperial College, July 1986.
  - M. Hermenegildo, E. Tick. Memory Referencing Characteristics and Caching Performance of AND-Parallel Prolog on Shared-Memory Architectures. New Generation Computing, Vol. 7, Num. 1, pages 37-58, Springer Verlag, October 1989. on line
  - M. Hermenegildo, K. Greene. The &-Prolog System: Exploiting Independent And-Parallelism. New Generation Computing, Vol. 9, Num. 3,4, pages 233-257, Springer Verlag, 1991.
  - G. Gupta, M. Hermenegildo, E. Pontelli, V. Santos-Costa. ACE: And/Or-parallel Copying-based Execution of Logic Programs. International Conference on Logic Programming, pages 93-110, MIT Press, June 1994.
  - E. Pontelli, G. Gupta, D. Tang, M. Carro, M. Hermenegildo. Improving the Effi ciency of Nondeterministic And-parallel Systems. The Computer Languages Journal, Vol. 22, Num. 2/3, pages 115-142, Pergamon/Elsevier, July 1996. on line

Selected Bibliography (cont.)

- Scheduling:
  - M. Hermenegildo. Relating Goal Scheduling, Precedence, and Memory Management in AND-Parallel Execution of Logic Programs. Fourth International Conference on Logic Programming, pages 556-575, MIT Press, University of Melbourne, May 1987.
  - K. Shen, M. Hermenegildo. Flexible Scheduling for Non-Deterministic, And-parallel Execution of Logic Programs. Proceedings of EuroPar'96, LNCS, Num. 1124, pages 635-640, Springer-Verlag, August 1996. on line
- Visualization of parallelism:
  - M. Carro, L. Gómez, M. Hermenegildo. Some Paradigms for Visualizing Parallel Execution of Logic Programs. 1993 International Conference on Logic Programming, pages 184-201, MIT Press, June 1993. on line