
Exploiting instruction-level parallelism through tree-instructions

Jaime H. Moreno, Mayan Moudgill
IBM Thomas J. Watson Research Center

Motivation

- **Modern computer architectures**
 - ▶ **object-code compatibility across implementations**
 - object-code might be "tuned" to an implementation
 - ▶ **exploitation of instruction-level parallelism (ILP)**
 - increasingly wider-issue processors
 - ▶ **dynamic scheduling of instructions**
 - adapting object code to different implementations
 - checking of dependencies
 - policies for issuing instructions
 - management of resources
 - increasingly complex for higher ILP

Motivation (cont.)

- **Static scheduling as alternative to dynamic scheduling**
 - ▶ **explicit representation of ILP**
 - **object-code reflects organization of processor**
 - ▶ **very-long instruction word processors (VLIW)**

- **Traditional separation between architecture and implementation sacrificed**
 - ▶ **conscious decision**
 - **better exploitation of hardware resources**
 - ▶ **object-code compatibility is lost**

Motivation (cont.)

- **Proposed solutions to object-code compatibility problem include**
 - ▶ **Binary-to-binary translation**
 - ▶ **Delayed split-issue [Rau93]**
 - ▶ **Reschedule object-code at page-fault time [Conte95]**
 - ▶ **...**

- **Dynamic extraction of ILP**
 - ▶ **Fill-unit [Melvin88, Franklin94]**
 - ▶ **DIF [Nair97]**
 - ▶ **DAISY [Altman97]**
 - ▶ **some current microprocessors (PentiumPro, ...)**

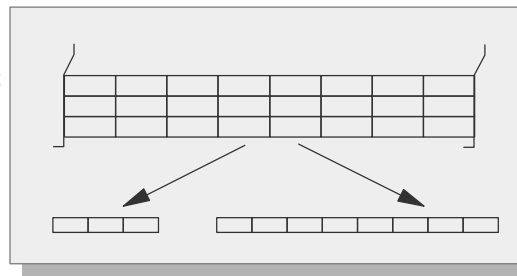
- **Active area of research**

Objectives

- **Explicit representation of ILP**
 - ▶ no dynamic checking of dependencies
 - ▶ no specific processor implementation
 - ▶ simple issue policy
 - ▶ suitable for processors with varying issue-width

- **Object-code compatibility across implementations**
 - ▶ monotonic performance increase with issue-width
 - ▶ minor degradation with respect to compilation targeted to each implementation

- **Premise**
 - ▶ compile for wide-issue target but execute on implementations of various (smaller) widths



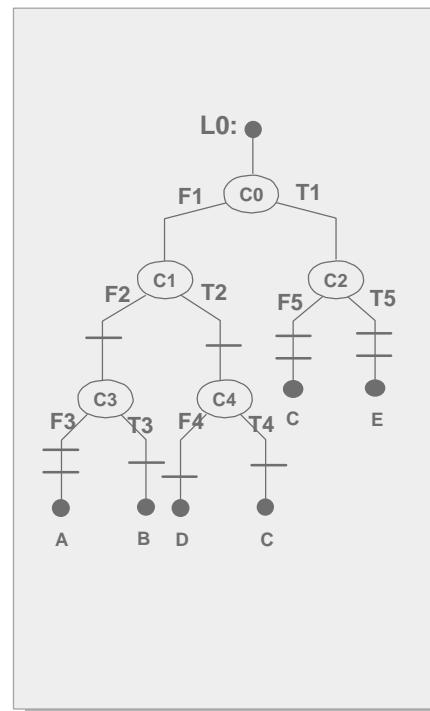
Exploiting instruction-level parallelism through tree-instructions
Moudgill

J. Moreno, M.

Approach

- **Programs represented as sequences of tree-instructions**
 - extension to [Ebcioğlu88, Moon93]

- **A tree-instruction**
 - ▶ multiway branch (unlimited)
 - ▶ primitive operations (unlimited)
 - ▶ operations and branches executable in parallel
 - ▶ single path selected at execution time
 - operations in selected path completed
 - operations in other paths discarded
 - ▶ paths ordered from left-to-right
 - probability of being taken
 - ▶ sequential constraints in each path
 - as if operations were executed one at a time

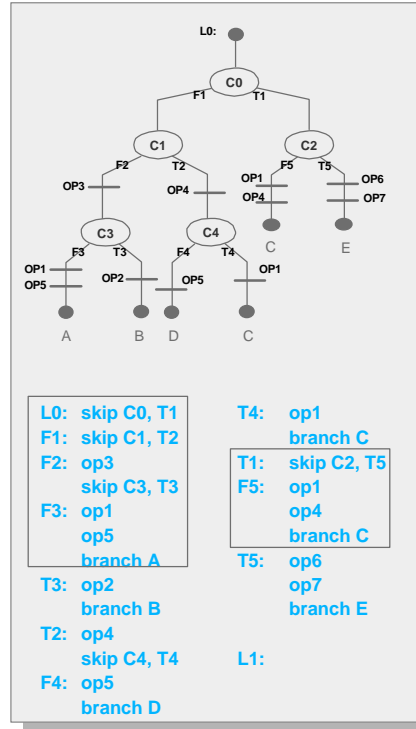


Exploiting instruction-level parallelism through tree-instructions
Moudgill

J. Moreno, M.

Representation of tree-instructions in memory

- **Contiguous sequence of primitive operations**
 - ▶ depth-first traversal of tree
- **skip conditional operation**
 - ▶ branch within tree (short positive displacement)
- **unconditional branches**
- **End of tree in memory delimited implicitly**
 - ▶ operation after unconditional branch not reachable by any skip instruction; or
 - ▶ # of branches = # of skips + 1

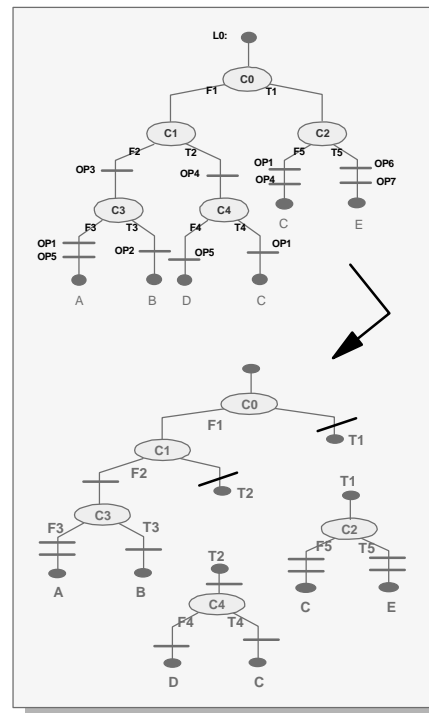


Exploiting instruction-level parallelism through tree-instructions

J. Moreno, M. Moudgill

Execution of tree-instructions

- **If enough resources**
 - ▶ entire tree executed at once
- **If not**
 - ▶ tree *pruned* according to resources
 - ▶ subtrees executed in successive cycles
 - ▶ subtrees
 - same structure as original tree
 - smaller
- **Pruning at**
 - ▶ skip instructions
 - ▶ in path (implicit branch)
- **No dependence checking required**
 - ▶ only resource constraints
- **If taken path included in first subtree**
 - ▶ other subtrees not fetched
 - "dynamic unspeculation"

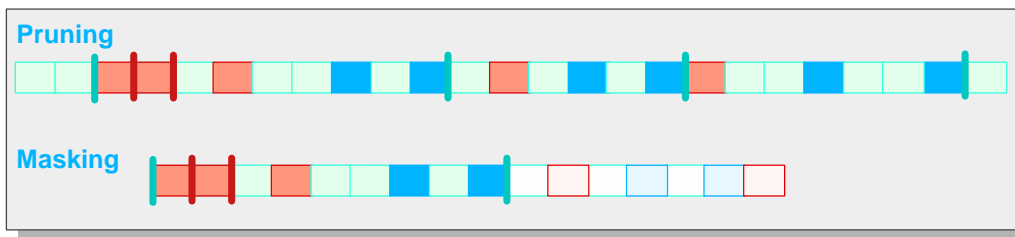


Exploiting instruction-level parallelism through tree-instructions Moudgill

J. Moreno, M.

Execution of tree-instructions (cont.)

- **Pruning step**
 - ▶ at I-cache reload time
 - ▶ subtrees fitted to implementation
 - ▶ I-cache representation augmented with subtrees boundaries
- **Masking step**
 - ▶ at I-cache fetch time
 - ▶ full fetch-width read from I-cache
 - ▶ mask-out operations beyond end of subtree
- **Pruning can be constrained to ease implementation requirements**
 - ▶ i.e., prune at doubleword boundaries

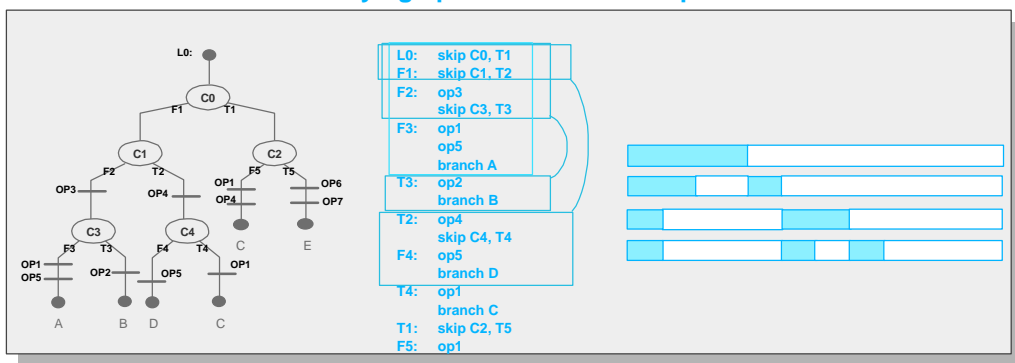


Exploring instruction-level parallelism through tree-instructions
Moudgill

J. Moreno, M.

Execution of subtrees

- Evaluate multiway branch and select path taken
- Execute all operations fetched
- Generate address of next tree-instruction to be executed
- Complete operations only in taken path
- **Multiway branch unit generates two results [RC20505]**
 - ▶ address of next tree-instruction
 - ▶ *execution mask* identifying operations to be completed



Exploring instruction-level parallelism through tree-instructions
Moudgill

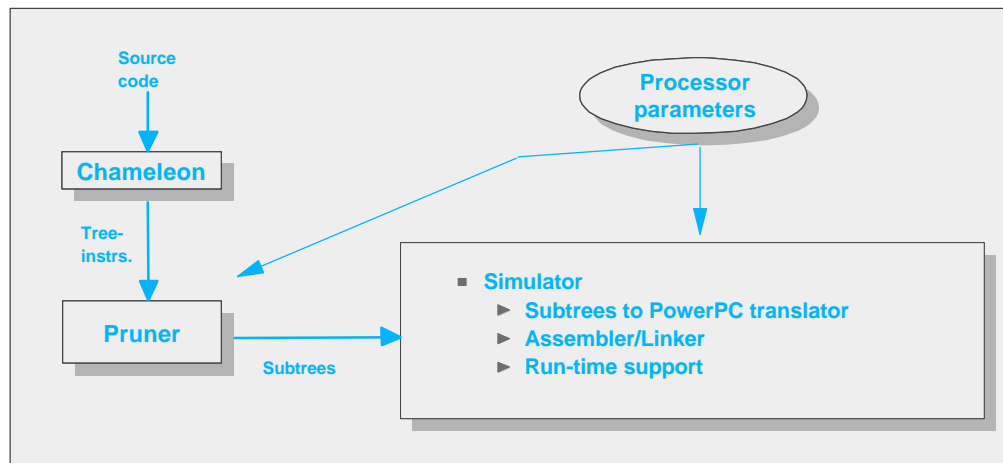
J. Moreno, M.

Experimental evaluation

- **PowerPC based processor architecture; deviations include [RC20733]**
 - ▶ 64 registers, 16 condition registers
 - ▶ support for speculative non-trapping load operations
 - ▶ some complex operations deleted
 - ▶ "record" form of operations may target any condition register
 - ▶ shorter displacement field in memory operations (11 bits)
 - ▶ some operations encoded in 64 bits
 - ▶ some new operations
 - ▶ some support for conditional execution: conditional move, conditional store

Experimental evaluation (cont.)

- **Components of experimental environment [RC20495]**
 - ▶ *Chameleon*, optimizing research compiler
 - ▶ Pruner
 - ▶ Simulator



Experimental evaluation: methodology

- **Count number of tree-instructions (subtrees) executed**
 1. **code compiled according to the resources of an implementation**
 2. **pruning code compiled for largest implementation**

Experimental evaluation: processor configurations

Configurations	Integer	Memory	Branch	FP	Maximum
A (16/8/8/2/16)	16	8	8	2	16
B (12/6/6/2/12)	12	6	6	2	12
C (12/4/4/2/12)	12	4	4	2	12
D (8/4/4/2/8)	8	4	4	2	8
E (8/4/2/2/8)	8	4	2	2	8
F (6/3/3/2/6)	6	3	3	2	6
G (6/3/2/2/6)	6	3	2	2	6

Operation	Latency
Integer	1
Float. point	3
Load	1
Divide	10
Multiply	3

Chameleon, optimizing research compiler

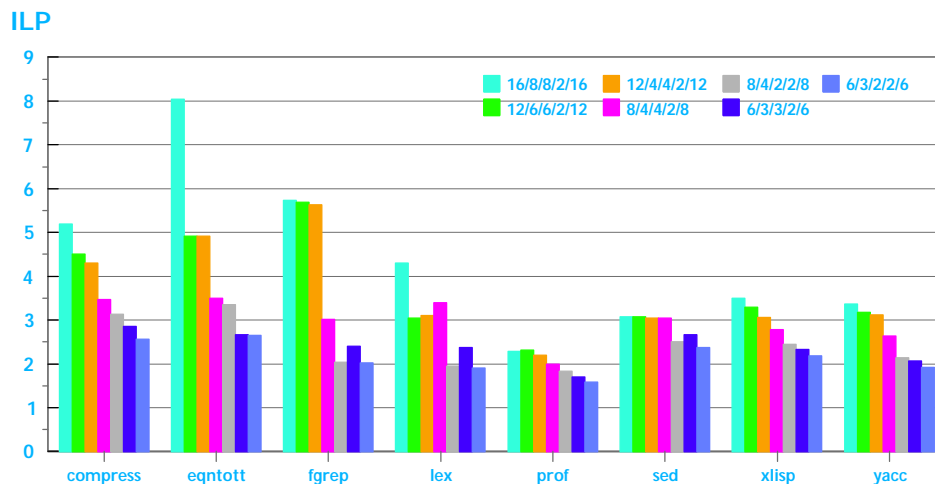
- **Aggressive suite of optimizations**
 - ▶ **Traditional:** constant propagation, loop-invariant code motion, etc.
 - ▶ **ILP-increasing:** loop transformations, memory disambiguation, etc.
 - ▶ **Architectural based:** designed to exploit architecture features
- **Synthetic branch probabilities [Ball93]**
 - ▶ can also use profiling directed feedback
- **Scheduler**
 - ▶ enhanced version of selective scheduling [Moon92]
 - ▶ all loops subject to *software pipelining*
 - all instructions in a loop examined for scheduling (not fixed window)
 - heuristics for selecting scheduled slot (not greedy)
 - sensitivity to register pressure (not schedule if might spill register)
- **Further information elsewhere [RC20495, RC20694, IBM JRD June 97]**

Exploring instruction-level parallelism through tree-instructions
Moudgill

J. Moreno, M.

Chameleon's ILP extraction capabilities

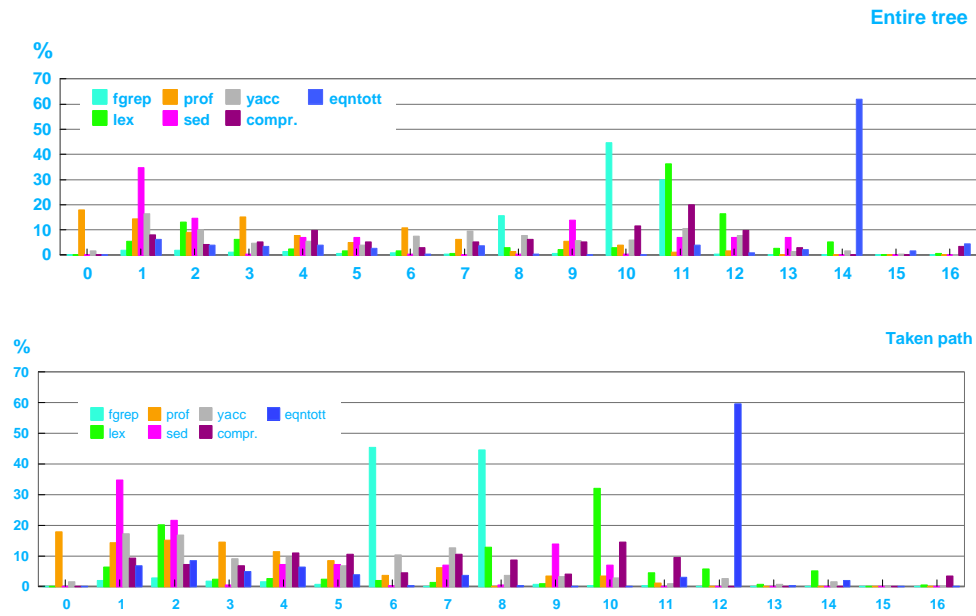
- **Path-length ratio with respect to optimized code generated by *x/c***
 - ▶ standard production compiler, -O2



Exploring instruction-level parallelism through tree-instructions
Moudgill

J. Moreno, M.

Dynamic distribution of operations per tree

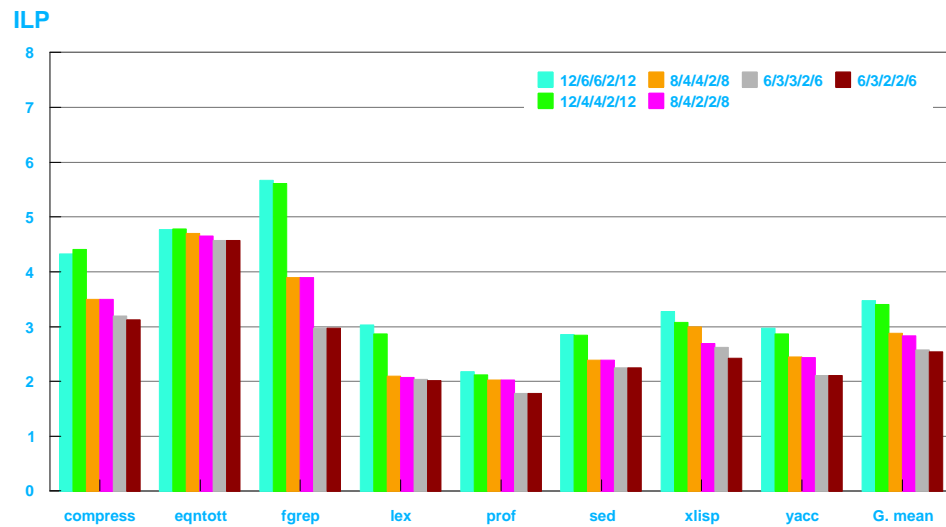


Exploring instruction-level parallelism through tree-instructions Moudgill

J. Moreno, M.

Path-length ratio of pruned code

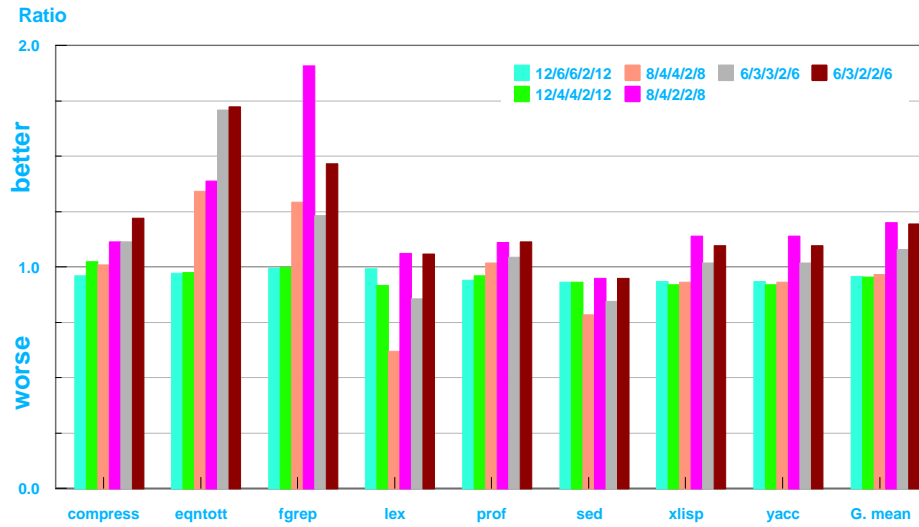
- with respect to optimized code generated by x/c



Exploring instruction-level parallelism through tree-instructions Moudgill

J. Moreno, M.

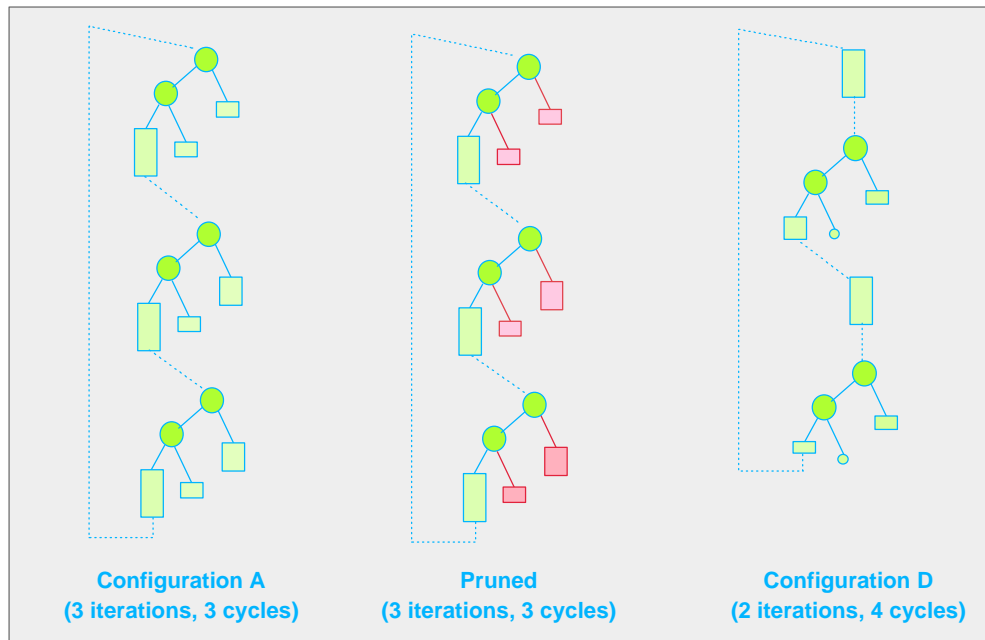
Ratio of pruned code to code compiled for small implementation



Exploring instruction-level parallelism through tree-instructions
Moudgill

J. Moreno, M.

Pruning sometimes better: function *strchr* in *fgrep*



Exploring instruction-level parallelism through tree-instructions
Moudgill

J. Moreno, M.

Pruning behavior

- **compress**
 - ▶ **1820 tree-instructions**

Configuration	Trees potentially pruned	Significant trees pruned	Pruned leftmost path	Pruned other paths
B (12/6/6/2/12)	106	28	4	24
C (12/4/4/2/12)	98	27	3	24
D (8/4/4/2/8)	257	54	10	44
E (8/4/2/2/8)	271	50	10	40
F (6/3/3/2/6)	418	71	20	51
G (6/3/2/2/6)	409	73	21	52

Concluding remarks

- **Programs represented as tree-instructions**
- **Explicit representation of instruction-level parallelism**
 - ▶ **preserving object-code compatibility**
 - ▶ **no dependence checking among operations**
- **Effective mechanism for executing same code on various implementations**
 - ▶ **ILP increases with issue-width**
 - ▶ **minor degradation with respect to code compiled for specific implementation**
 - **less than 10% in most cases**
 - **some cases even exhibit better ILP**
 - **potentially some negative effect due to comparatively larger code size (not yet quantified)**

Concluding remarks (cont.)

- **Monotonic increase in ILP with pruning**
 - ▶ **compiling/tuning for various issue-width more sensitive to optimization algorithms**
- **Compiler *could* always produce code achieving same/better ILP**
 - ▶ ***but it is not easy!***
- **Pruning allows focusing effort on optimizing code for large configuration**
- **Pruning could be performed at various places**
 - ▶ **I-cache replacement**
 - ▶ **Page-fault replacement (as in [Conte95])**

Further information

- **IBM Research CyberDigest**
 - ▶ **<http://www.watson.ibm.com:8080/>**
(search by authors' name)
- **IBM Research web site**
 - ▶ **<http://www.research.ibm.com/vliw/>**
(follow link to publications)