Hot Compilers for Future Hot Chips

Saman Amarasinghe, Jennifer Anderson, Robert French, Mary Hall, Monica Lam, Shih-Wei Liao, Brian Murphy, Chau-Wen Tseng, Chris Wilson, Robert Wilson

Stanford University

Contact: Monica Lam
(lam@cs.stanford.edu, http://suif.stanford.edu)

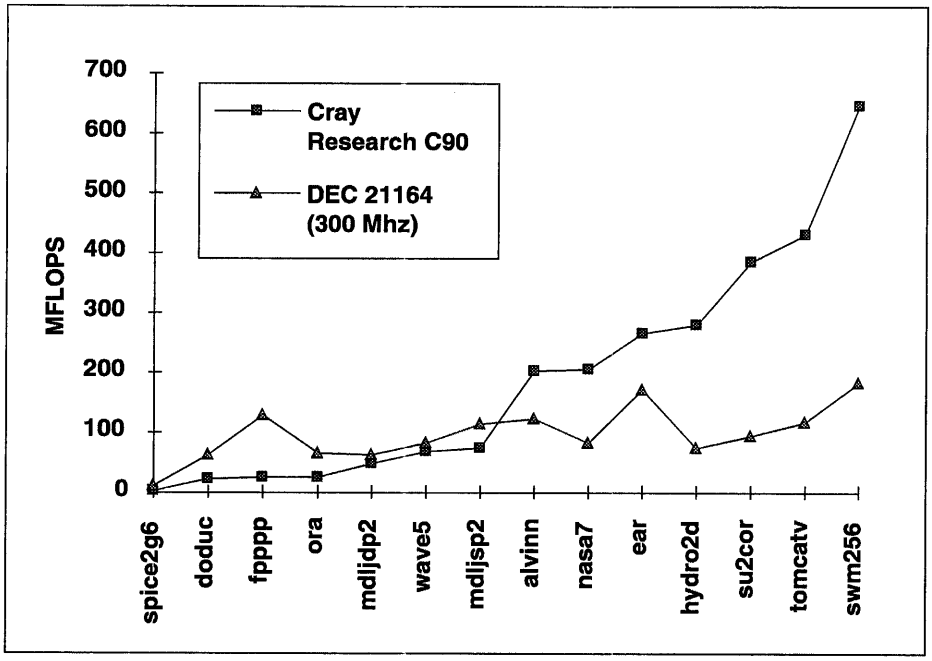
Microprocessor Directions

- **Current trends**
  - Higher clock rate
  - More instruction level parallelism (ILP)
- **Alternate direction: Multiple superscalar processors on a chip**
  - Hardware advantages over Very Wide superscalars: Simpler, Faster
    - Less wires
    - Multiported caches → separate caches
    - Multiported registers → separate register files
    - Data dependence between instructions → disjoint sets
  - Software advantages:
    - Supports multiprogramming
    - Supports explicit parallel programming
- **Key question: Performance of individual applications?**

  Are multiprocessors competitive to processors with high ILP?

6.1-03

SPEC92fp Programs

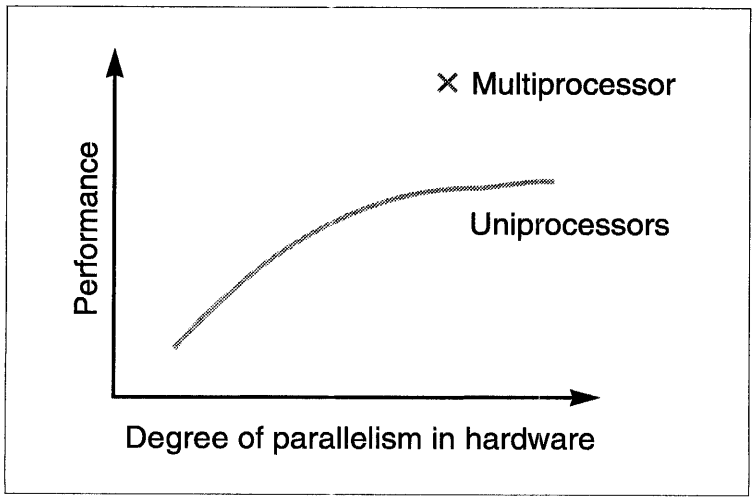


3

6.1-04

Potential Advantage of Multiprocessors

- **High ILP → Wide variation of program performance**
- **Diminishing return of ILP**



4

Challenges for Multiprocessor Compilers

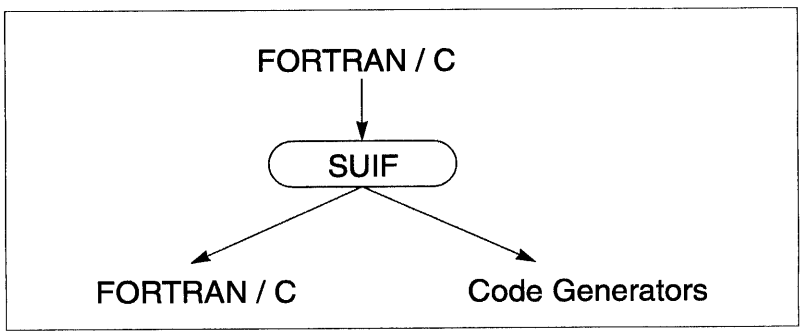
- **Programs with high ILP**
  - Must minimize synchronization and communication cost
- **Programs with low ILP**
  - Multiprocessors' unique advantage →

    Extract independent threads of computation
- **Programs written in C**
  - Pointer analysis

5

SUIF Compiler Research Project

- **SUIF (Stanford University Intermediate Format)**



  - A fully functional compiler research infrastructure
  - Base version has been released and is freely available
  - Techniques described will be incorporated in future releases
- **Techniques expected to be transferred to industry in a few years**

6

6.1-07

I. Pointer Analysis

- **Important for C code**
  - To reorder/parallelize loads&stores through pointers
- **Efficient and accurate analysis for**
  - Parameter passing
  - Pointer assignments
  - Direct array address calculations (*p++)
  - Function pointers
- **Technique: Interprocedural pointer analysis algorithm**
  - Avoids re-analyzing a function by determining the relevant pointer aliases at call sites
  - [Wilson&Lam, ACM PLDI Conference '95]
- **Future research**
  - Efficient analysis of recursive data structures

7

6.1-08

Preliminary Results

Program	Lines	Procedures	Analysis Time DEC 5000 (sec)
alvinn	272	8	0.29
grep	430	9	0.57
diff	668	23	1.31
lex315	776	16	0.85
compress	1503	14	1.29
loader	1539	29	1.43
football	2354	57	5.12
compiler	2360	37	5.86
assembler	3361	51	4.37
eqntott	3454	60	6.17
ear	4284	68	2.41
simulator	4663	98	9.89

- **Can find all loop-level parallelism in both SPEC92fp C programs**

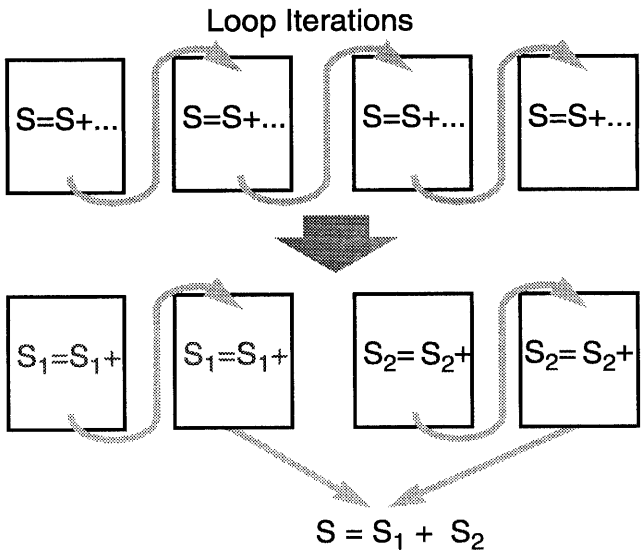
8

II. Finding Coarse-Grain Parallelism

First Generation Parallelizers	SUIF
Optimizations on **scalar** variables	Optimizations on **array** variables
Within procedures	**Across** procedures

- **Reference**
  - [Hall et al., Supercomputer 95]

Reductions

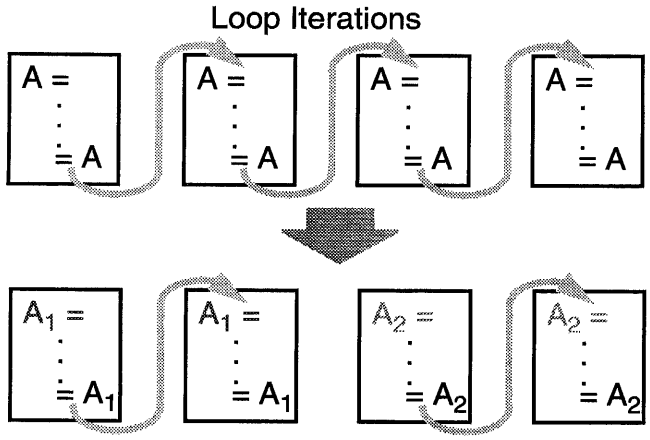


- **Generalized to reductions on (portions of) arrays**
  - in either dense or sparse representation

6.1-11

Privatization

- **Each processor keeps a private copy of temporary data**



- **Generalized to privatizing arrays**
  - Requires data flow analysis on individual array elements

11

6.1-12

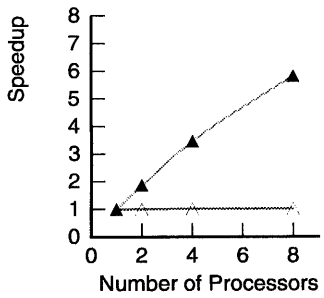
Interprocedural Analysis

- **Algorithm**
  - Based on interval analysis
  - Complexity: Analyze each function 3 times
  - Incremental compilation possible by saving summary information
- **Important to have a complete suite of interprocedural analyses**
  - Constant propagation
  - Loop invariant analysis
  - Induction variable analysis
  - Scalar privatization
  - Scalar reduction recognition
  - Array data dependence analysis
  - Array privatization
  - Reductions to regions of arrays

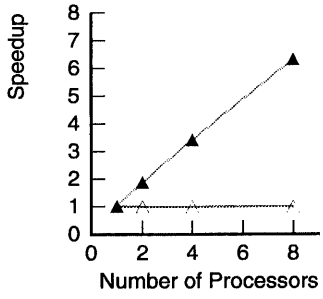
12

NAS Parallel Benchmarks on SGI Challenge

Improvement due to more parallelism:

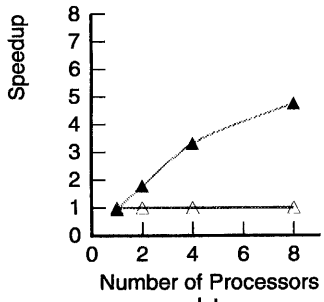


cgm

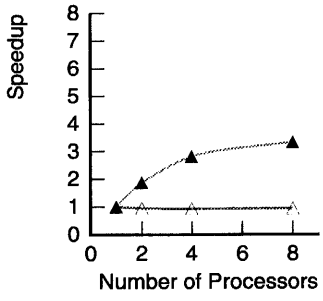


embar(APR)

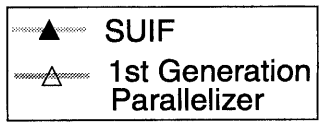
Improvement due to less frequent synchronization:



appbt



appsp



13

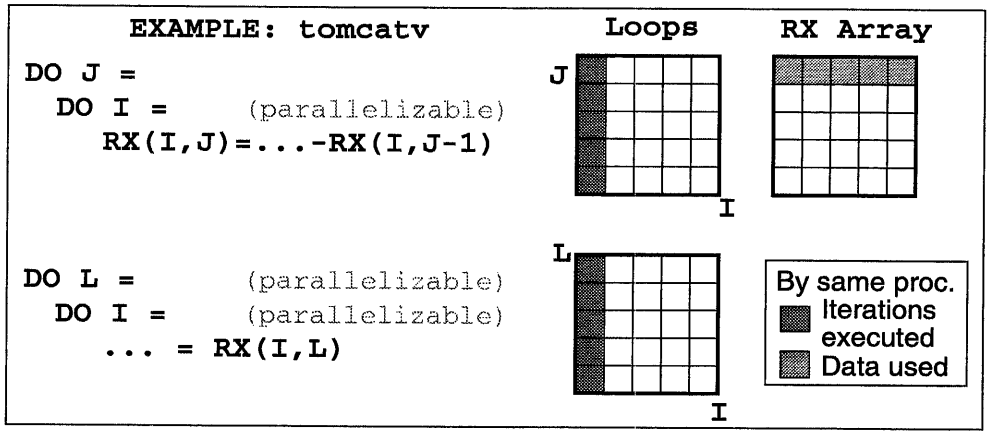
Observations

- **Parallelizers are maturing**
  - Interprocedural analysis
  - Array reduction and privatization
- **An interactive parallelizer will be a useful tool**

14

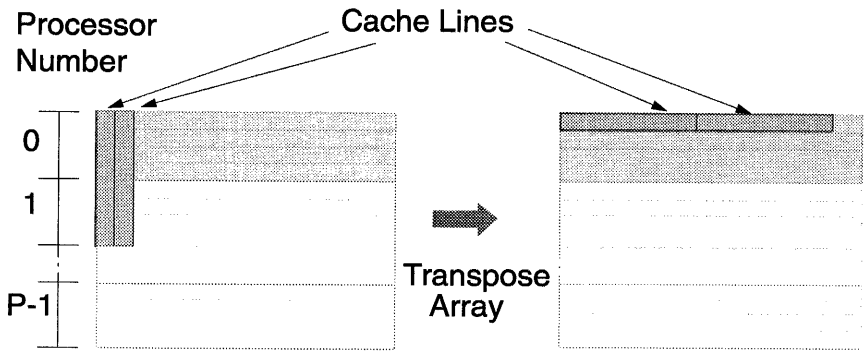
III. Optimizations

- **Minimize synchronization & cache traffic across loops**
  - 1st generation parallelizers
    - Insert barriers between every parallel loop
    - Distribute iterations in each loop independently
  - New global analysis
    - Smart assignment of iterations to eliminate barriers and cache traffic



15

Data Restructuring



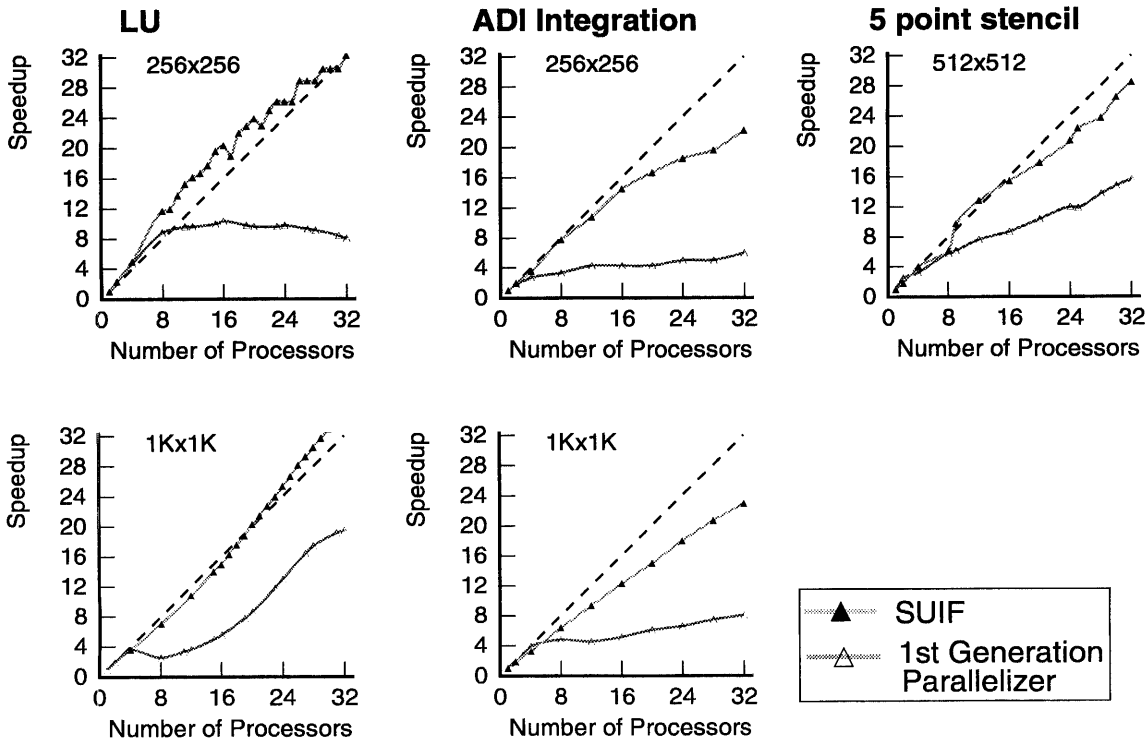
- **Problem**
  - False sharing: processors sharing same cache line but not data
  - Conflicts: set associativity maps data to same cache position
- **Solution:**
  - Make data accessed by each processor contiguous via data transposes and blocking

16

Stanford DASH Multiprocessors

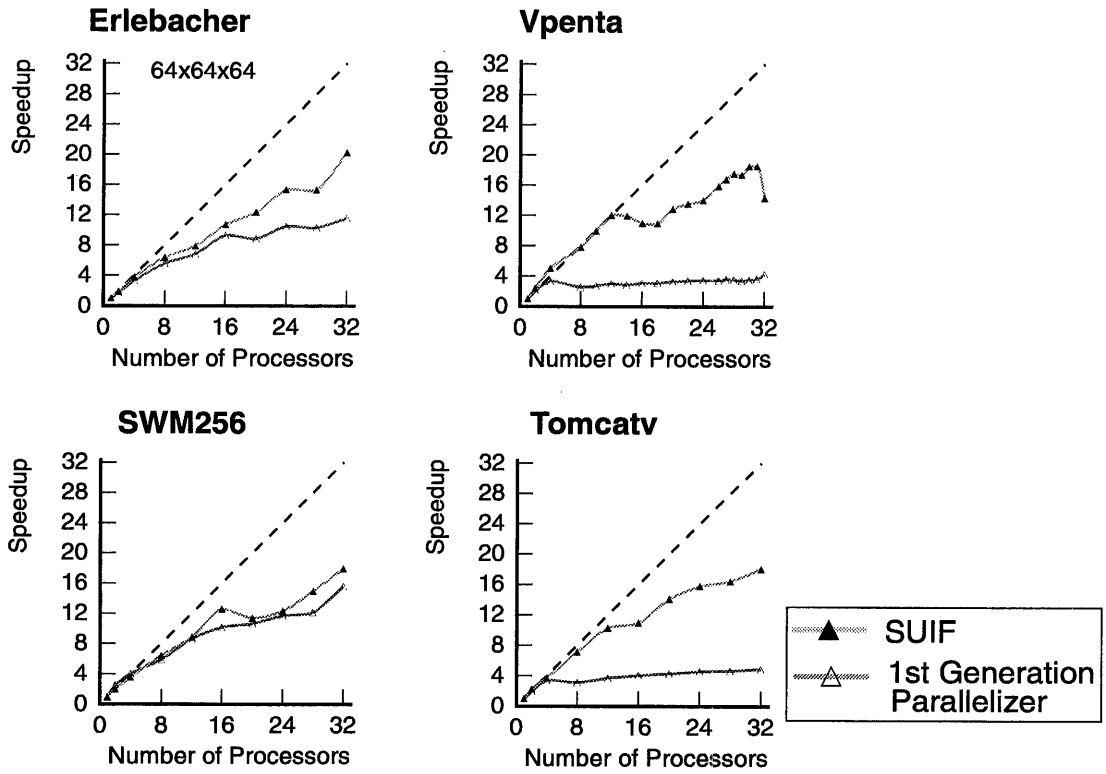
- **Characteristics**
  - Directory-based, cache-coherent NUMA
  - 32 33MHz MIPS R3000 in 8 clusters
  - 64KB 1st level direct-mapped cache, 256KB 2nd level cache
- **Techniques also useful for small-scale multiprocessors made out of fast processors**

Experimental Results: Kernels



6.1-19

Programs



19

6.1-20

Results on SPEC92 fp

... will be reported in the meeting

20

Conclusions

- **Maturation of parallelizing compilers**
  - Pointer analysis
  - Outer loop parallelism
  - Synchronization and cache optimizations
  - Interactive compilers will be effective in finding parallelism
- **Build multiprocessors with superscalar processors instead of very wide superscalars**
  - More cost-effective
  - More versatile
    - Thread and instruction level parallelism
    - Multiprogramming
  - Fine-grain parallelism possible with better support
    → multiprocessors on a chip

21