Simple Vector Microprocessors for MultiMedia Applications

Paper: http://www.eecg.toronto.edu/~corinna/vector/

Corinna G. Lee and Mark G. Stoodley
University of Toronto

Notable Quotes

David Papworth, Intel Fellow and Chief Architect of Pentium Pro

- hardware: "Processors will add increasing amounts of 'vector-like' execution."
- software: "Vectorization is back!"

ref: Architecture Panel Session at PLDI '98

Current Trends

- multimedia applications are growing in importance
- current hardware trend to support multimedia is to add short vector extensions to state-of-the-art superscalar processors
  - Sun VIS, HP MAX-2, SGI MDMX, Digital MVI, Intel MMX, Intel Katmai, Motorola AltiVec
- BUT control logic for complex superscalar processor is difficult to implement
  - over past 2 years, shippings have been delayed repeatedly to meet target speeds
  - late shippings attributed to complex out-of-order designs

ref: Linley Gwennap, MPR articles, Feb,Oct,Dec 1997

Alternative Hardware Solution

- use simple vector processor design:
  - 2-way, in-order
  - vector length of 64
  - vector width of 8 (i.e., has 8 lanes)
- for this study, focus on multimedia applications
  - important emerging applications area
  - others have demonstrated effectiveness of vector architectures on floating-point applications and SPECint programs

Outline

Motivation

Processor Configurations

Die Area Estimates

Performance Results

Summary and Conclusions

Superscalar and Vector Processors

Feature	Processors: OOO Superscalar	OOO Short Vector	Simple Long Vector
ISA	64b MIPS	64b MIPS with vector extensions	
issue order issue width fetch width re-order buffer size	out of order 4 instructions 4 instructions 56 instructions		in order 2 instructions 2 instructions —
#physical registers	64 int 64 FP	64 int 64 FP 32 8-element Vreg	32 int 32 FP 32 64-element Vreg
datapath	2 IUs 1 LSU	2 IUs 1 LSU 1 VU with 8 IUs	2 IUs 1 LSU 1 VU with 8 IUs
memory system	64-bit data bus, 64-bit address bus R10000-based 2-level cache memory		
C compiler	SGI V5.3 −O2	SGI V5.3 −O2 and VSUIF V1.1.0	

T0-Based Vector Processor: Main Differences From T0

- vector parameters
  - vector length of 64, not 32
  - 1 VU, not 2 VUs
- to match 64b superscalar processor
  - 64-bit data bus, not 128 bits
  - 64-bit datapath, not 32 bits
  - R10000-like latencies for operations
- to improve scalar performance
  - more powerful scalar core: R5000
- to reduce vector die area
  - fully pipelined multiply for narrow data types ($\leq$ 16 bits)

Component Areas for Two Implementations of the Long Vector Processor

Processor Component	Area in mm^2 Scaled to $0.25\mu m$	
	Area Efficient Implementation	High Performance Implementation
64b Vector Datapath		
8 integer units	24.0	36.0
load/store unit	3.0	3.0
64b Vector Register File		
32 64-element vector registers	9.5	19.0
64b MIPS R5000		
scalar integer and FP datapath	10.3	10.3
scalar integer and FP register file	0.5	0.5
instruction issue	0.8	0.8
Clocking and Overhead	4.0	4.0
Total	52.3	73.8

64b Scalar Integer Unit

- area-efficient implementation: 3mm^2

$$
\begin{aligned}
& 2 \times \text{area of 32b T0 scalar datapath} \\
& + 4 \times \text{area for 16x16 T0 multiplier array} \\
& = 2 \times 1\text{mm}^2 + 4 \times 0.25\text{mm}^2
\end{aligned}
$$

  very conservative estimate: area for 21164 IU is $< 1\text{mm}^2$

- high-performance implementation: 4.5mm^2
  - based on OOO integer unit in 21264

Load/Store Unit

- need 64x512 cross-bar to transfer data between 64b memory data bus and 8 64b register buses
- area estimate:

  area for 128x256 T0 crossbar
  $+ 2 \times$ T0 area for shifting/aligning 32b data

- address processing handled by scalar portion of processor
  - only one address bus
  - scalar and vector memory instructions use same memory interface

Vector Register File

- areas are based on layout details for T0 vector register file
  - ref: Krste Asanović (chief architect of T0). *Vector Microprocessors*. PhD thesis, University of California at Berkeley, May 1998.
- includes area for overhead circuitry
  - read sense amplifiers, data latches for writes and reads, multiplexors, drivers, etc.

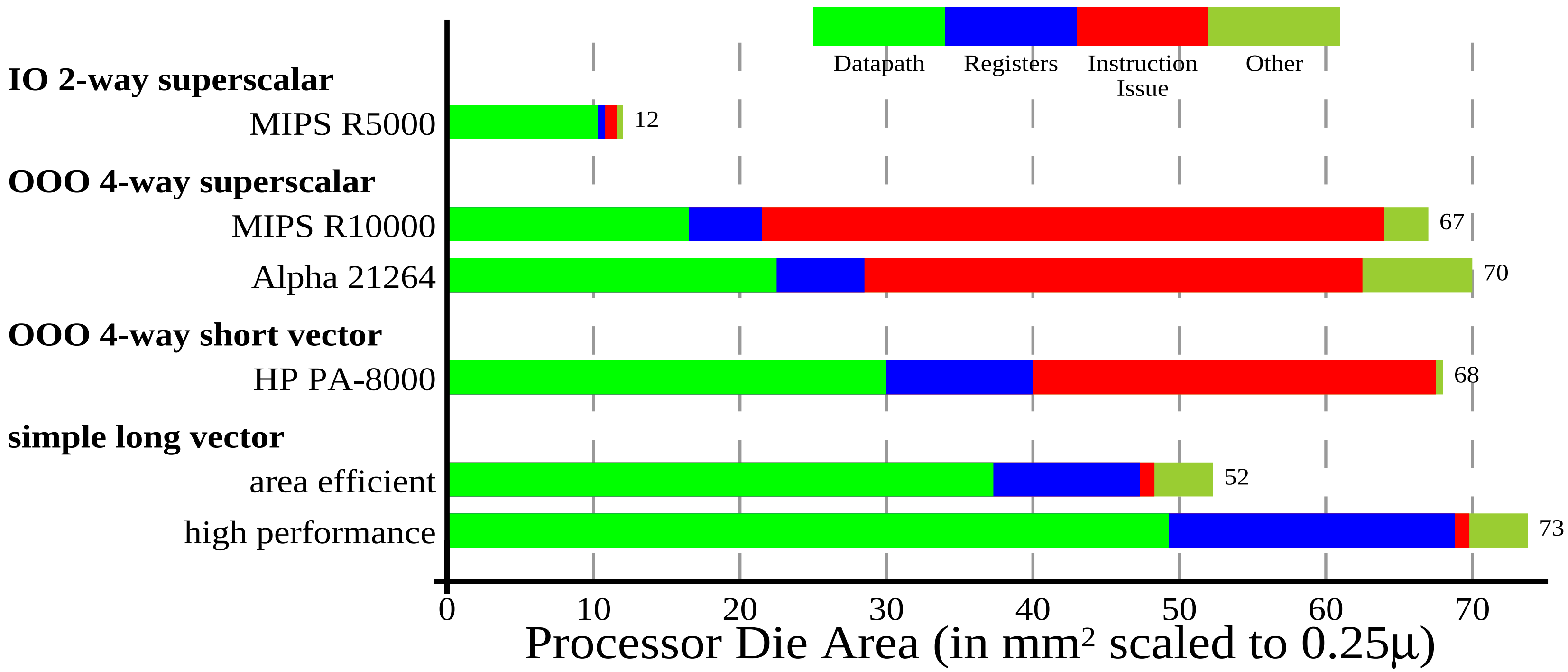
- area-efficient implementation:
  - time-multiplexes word and bit lines
- high-performance implementation:
  - uses extra buses and ports to avoid time-multiplexing

Area Comparison with Existing Superscalar Processors

want area differences to be due to parallel-specific features

- die areas scaled to a 0.25μm process to eliminate areal differences due to differences in line size
- areas are for processor components only; areas for cache and TLB structures, external interface logic, and the pad ring excluded
- areas based on actual VLSI implementations

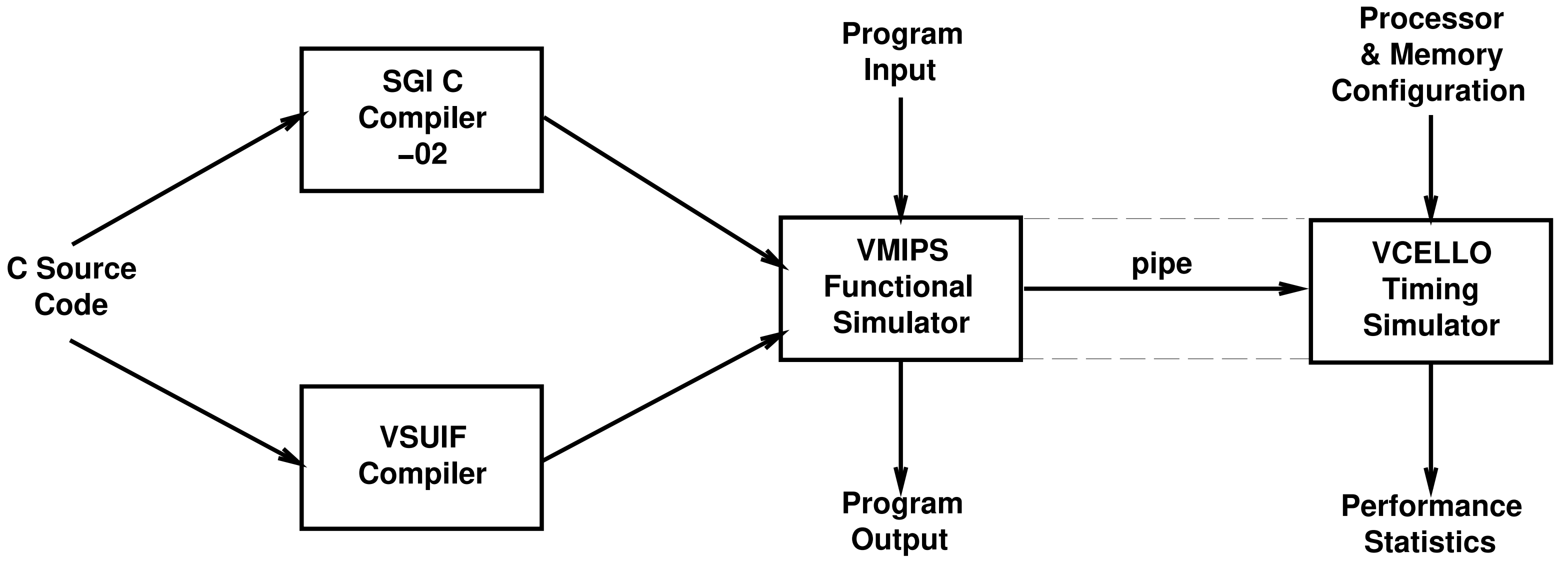
Breakdown of Processor Die Areas



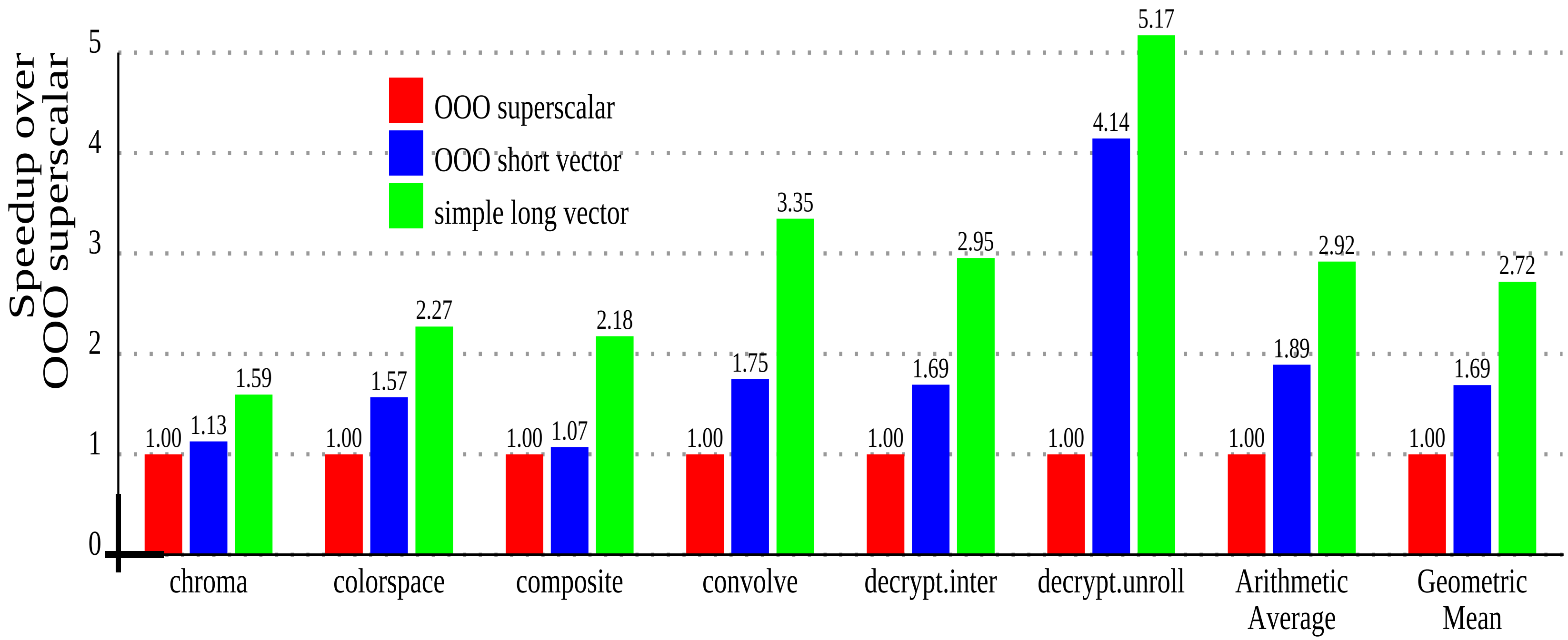
Highly Vectorizable Benchmark Programs

Benchmark	Data Width	Input	Description
`chroma`	8 bit	320x240	Merges two images on the basis of a “whiteness” threshold.
`colorspace`	8 bit	24-bit	Converts an image in RGB to YUV values.
`composite`	8 bit	color	Blends two images together by a blend factor α.
`convolve`	8,16 bit	image(s)	Convolves an image with a 3x3 16-bit kernel.
`decrypt.unroll`	16 bit	16,000-byte	Unrolled version of IDEA decryption.
`decrypt.inter`	16 bit	message	Loop-interchanged version of IDEA decryption.

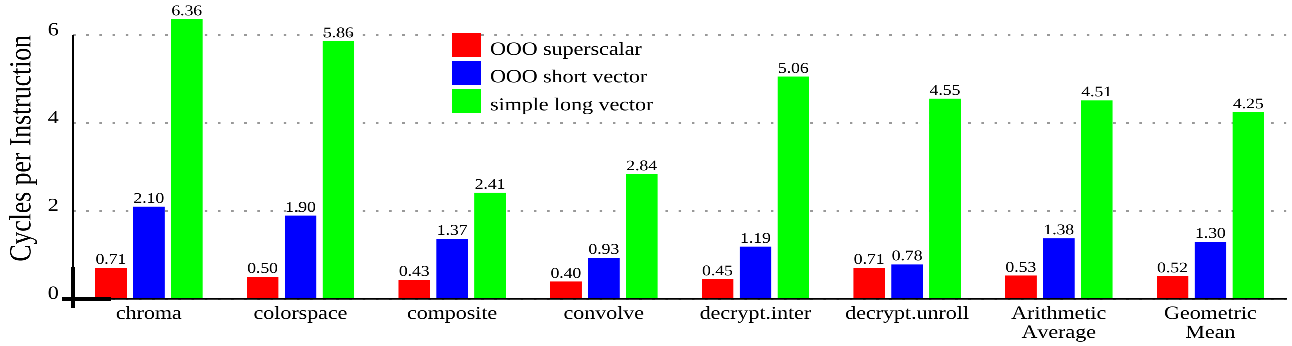
VIVACE Compiler/Simulation Infrastructure

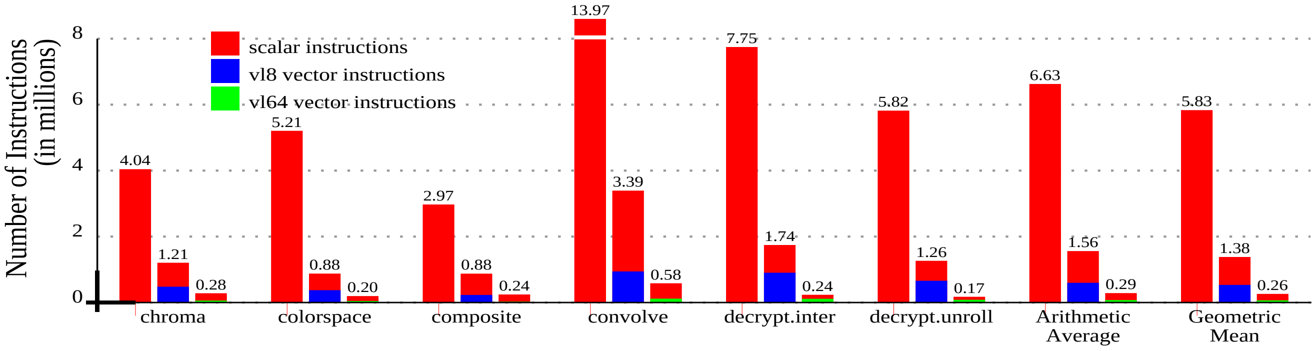


Processor Performance



CPI and Dynamic Instruction Count





Stripmining = Implicitly Loop Unrolling

Loop Version	Static Instructions	Dynamic Instruction Count
rolled	`load r3,0(r2)` `add r2,r2,4`	128

explicitly unrolled 64 times	`load r3,0(r2)`
	`load r3,4(r2)`
	⋮
	`add r2,r2,256`
		65

stripmined with VL=64	`vload v3,(r2)` `add r2,r2,256`	2

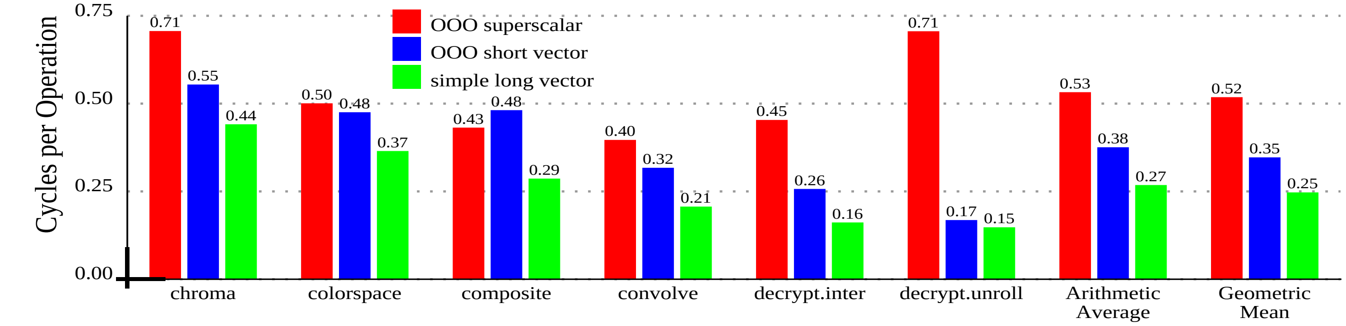
Average Speedup Deconstructed Using CPI Equation

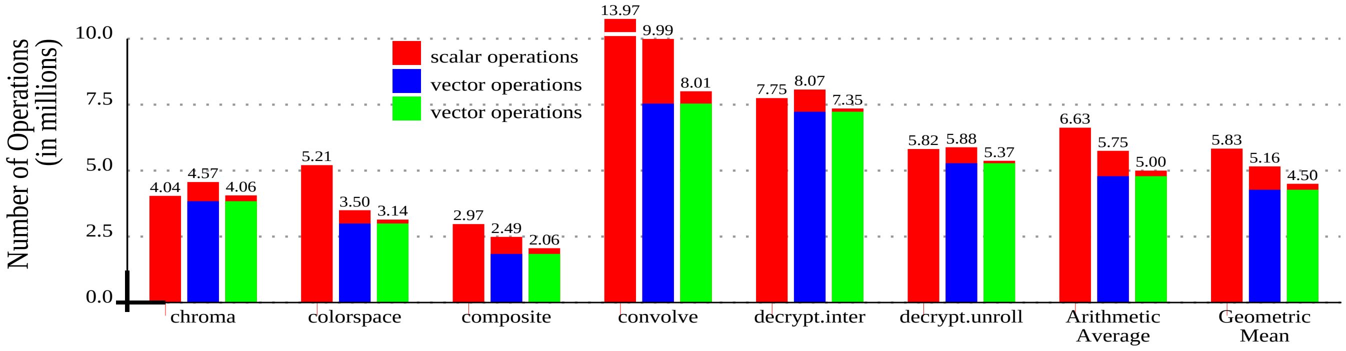
Processor	Cycles per Instruction	Dynamic Instruction Count	Cycle Count	Speedup over Super-scalar
OOO superscalar	$\overline{\mathrm{CPI}}_{\mathrm{SS}}$	$\overline{\mathrm{NI}}_{\mathrm{SS}}$	$\overline{\mathrm{CPI}}_{\mathrm{SS}} \times \overline{\mathrm{NI}}_{\mathrm{SS}}$	1.00×
OOO short vector	$2.50 \times \overline{\mathrm{CPI}}_{\mathrm{SS}}$	$0.24 \times \overline{\mathrm{NI}}_{\mathrm{SS}}$	$0.60 \times \overline{\mathrm{CPI}}_{\mathrm{SS}} \times \overline{\mathrm{NI}}_{\mathrm{SS}}$	1.67×
simple long vector	$8.19 \times \overline{\mathrm{CPI}}_{\mathrm{SS}}$	$0.045 \times \overline{\mathrm{NI}}_{\mathrm{SS}}$	$0.37 \times \overline{\mathrm{CPI}}_{\mathrm{SS}} \times \overline{\mathrm{NI}}_{\mathrm{SS}}$	2.71×

Effectiveness at Using Parallelism?

- usually low CPI means efficient use of hardware
- no longer true because amount of work carried out by an instruction varies tremendously
- instruction not an appropriate unit of work for determining effective use of hardware
- use operation instead
  - amount of work carried out by a functional unit

CPO and Dynamic Operation Count





Effective OLP and ILP in Vector Processors

Type of Parallelism	Vector Processors: OOO short	Vector Processors: simple long	Compiler Assistance
operation-level	√	√	
scalar–scalar ILP	√	×	use simple list scheduling to enable vector–scalar ILP instead
vector–scalar ILP	√	√	
vector–vector ILP	×	√	use loop unrolling or software pipelining to enable

Average Speedup Deconstructed Using CPO Equation

Processor	Cycles per Operation	Dynamic Operation Count	Cycle Count	Speedup over Super-scalar
OOO superscalar	$\overline{\mathrm{CPO}}_{\mathrm{SS}}$	$\overline{\mathrm{NO}}_{\mathrm{SS}}$	$\overline{\mathrm{CPO}}_{\mathrm{SS}} \times \overline{\mathrm{NO}}_{\mathrm{SS}}$	$1.00\times$
OOO short vector	$0.67 \times \overline{\mathrm{CPO}}_{\mathrm{SS}}$	$0.88 \times \overline{\mathrm{NO}}_{\mathrm{SS}}$	$0.59 \times \overline{\mathrm{CPO}}_{\mathrm{SS}} \times \overline{\mathrm{NO}}_{\mathrm{SS}}$	$1.70\times$
simple long vector	$0.48 \times \overline{\mathrm{CPO}}_{\mathrm{SS}}$	$0.77 \times \overline{\mathrm{NO}}_{\mathrm{SS}}$	$0.37 \times \overline{\mathrm{CPO}}_{\mathrm{SS}} \times \overline{\mathrm{NO}}_{\mathrm{SS}}$	$2.70\times$

Cost Benefits of Simple Long Vector Processors

- lower complexity and area cost than those for 4-way OOO superscalar implementation with short vector extensions
  - simpler control
  - more registers and datapath

$\Longrightarrow$ majority of VLSI structures are significantly more regular

- conservative estimate of area

Performance Benefits of Simple Long Vector Processors

- greater performance: 2.7x faster than OOO, 1.6x faster than OOO short vector
  - reduce two components of CPI equation
    * N: stripmining, an inexpensive technique to reduce $N_{dynamic}$ and N_{static} at "cost" of large vector register file
    * CT: simpler and more regular VLSI design
    * at expense of increasing CPI
  - reduce all three components of CPO equation
    * lower CPO indicates more effective use of parallelism
- conservative estimate of performance
  - performance data assumes comparable CT
  - vector compilation could be improved

Summary: Simple Long Vector Processor

- cost and performance benefits
  - lower complexity and area cost than those for 4-way OOO superscalar implementation with short vector extensions
  - greater performance: 2.7x faster than OOO, 1.6x faster than OOO short vector
- area and performance estimates for long vector processor are conservative
- configured like traditional vector architectures with two major enhancements:
  - much wider vectors
  - slightly wider instruction issue
- performance gains obtained with R10000-like 2-level caches

Future Work

- compilation: outer loop vectorization
  - increase number of loops that can be vectorized
  - replace complex loop transformations
  - reduce extraneous memory requirements
  - developed in early 90's by Viet Ngo
- processor architecture: support for conditional execution
  - reduce *nop* slots
- memory architecture
  - techniques to improve cache memory performance