Implications of Register and Memory Temporal Locality for Distributed Microarchitecture

Alireza Khalafi
David Morano
David Kaeli
Northeastern University

Northeastern University
Computer Architecture Research

Augustus Uht
University of Rhode Island

Microarchitecture Research Institute
University of Rhode Island

submitted to
International Symposium on Performance Analysis of Systems Software (ISPASS) 2003
October 1, 2002

outline

- **introduction**
- **related work**
- **example interval calculations**
- **experimental methodology & characterization results**
  - register intervals
  - memory intervals
- **implications for distributed microarchitectures**
  - distributed microarchitecture
  - L0 bypass results
- **summary**

introduction

- **distributed microarchitectures present new problems for operand access and forwarding**
  - centralized operand resources
  - routing congestion around these resources (technology problem)
  - contention for access (performance problem)
- **we explore a variety of inter-access intervals for both registers and memory variables**
- **goal: to explore the feasibility of operand bypass of the traditional centralized associated resources**
  - LSQ and L1 data cache for memory
  - register file, future file, or ROB for registers
- **definition of intervals**
  - access-use (from a read or a write of a variable instance to the next read of the same instance)
  - useful-lifetime (from a write to the last read of the same instance)
  - def-use (frome a write creating an instance to a read of the same)

related work

- **Madison et al (1976)**
  - characterization of temporal locality for memory
- **Verkamo (1985)**
  - memory locality for database applications
- **Franklin and Sohi (1992)**
  - explored three access intervals for registers (for Multiscalar)
    - age (def-use)
    - useful-lifetime (def-last-use)
    - lifetime (def-def) -- did not provide data
  - did not look at memory variables
- **Vijyadhar and Bhaskarpillia (1995)**
  - inter-reference gaps for memory
  - similar to access-def interval but includes write-write occurrences

example calculation

label	*instruction*	*event*	*intervals determined*
I1	r1 <= c	def(r1)	
I2	r2 <= c	def(r2)	
I3	r1 <= r2 + c	def(r1), use(r2)	access-use(r2)=1, def-use(r2)-1, useful-lifetime(r1)=0
I4	r2 <= r1 + c	def(r2), use(r1)	access-use(r1)=1, def-use(r1)=1, useful-lifetime(r2)=1
I5	r3 <= r1 + r2	def(r3), use(r1), use(r2)	access-use(r1)=1, access-use(r2)=1, def-use(r1)=2, def-use(r2)=1

- **a window of instructions are examined for intervals**
- **defs are known coming into the data gathering window**
- **'c' is an arbitrary immediate constant encoded in instructions**
- **final useful-lifetime for *r3* from instruction *I5* is indeterminate**

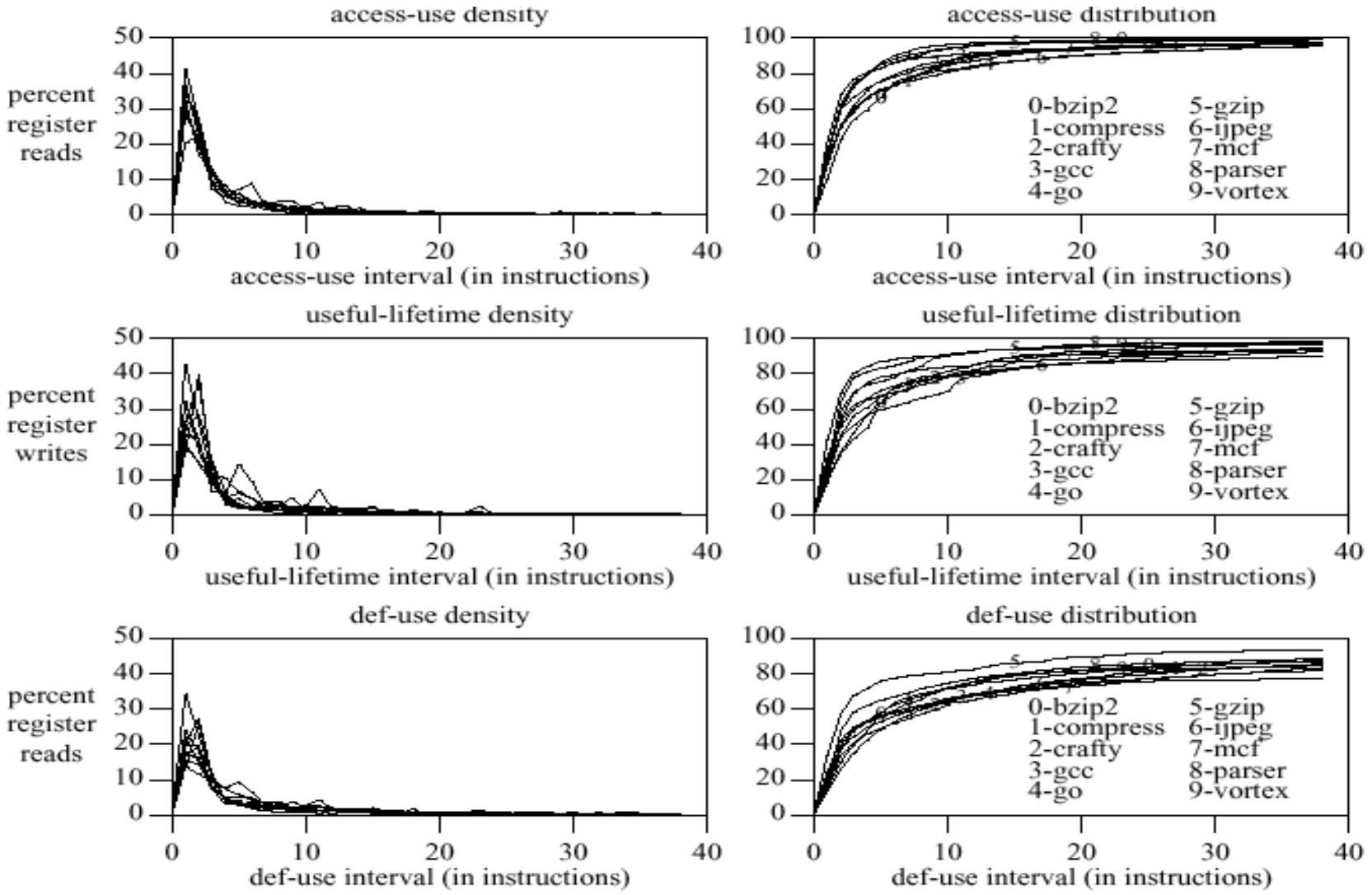
implications of intervals

- **access-use**
  - property of a read
  - useful when distributed caches are available in a microarchitecture
  - used when the cache has a write-allocate policy
- **useful-lifetime**
  - property of a write
  - useful when a microarchitecture has a means of retaining the write operand for some fixed time in units of instructions
  - a prediction of the future
- **def-use**
  - property of a read
  - useful when the operand from a write is simply forwarded and a read wants to try to catch (snarf) the operand
  - a prediction of past write behavior

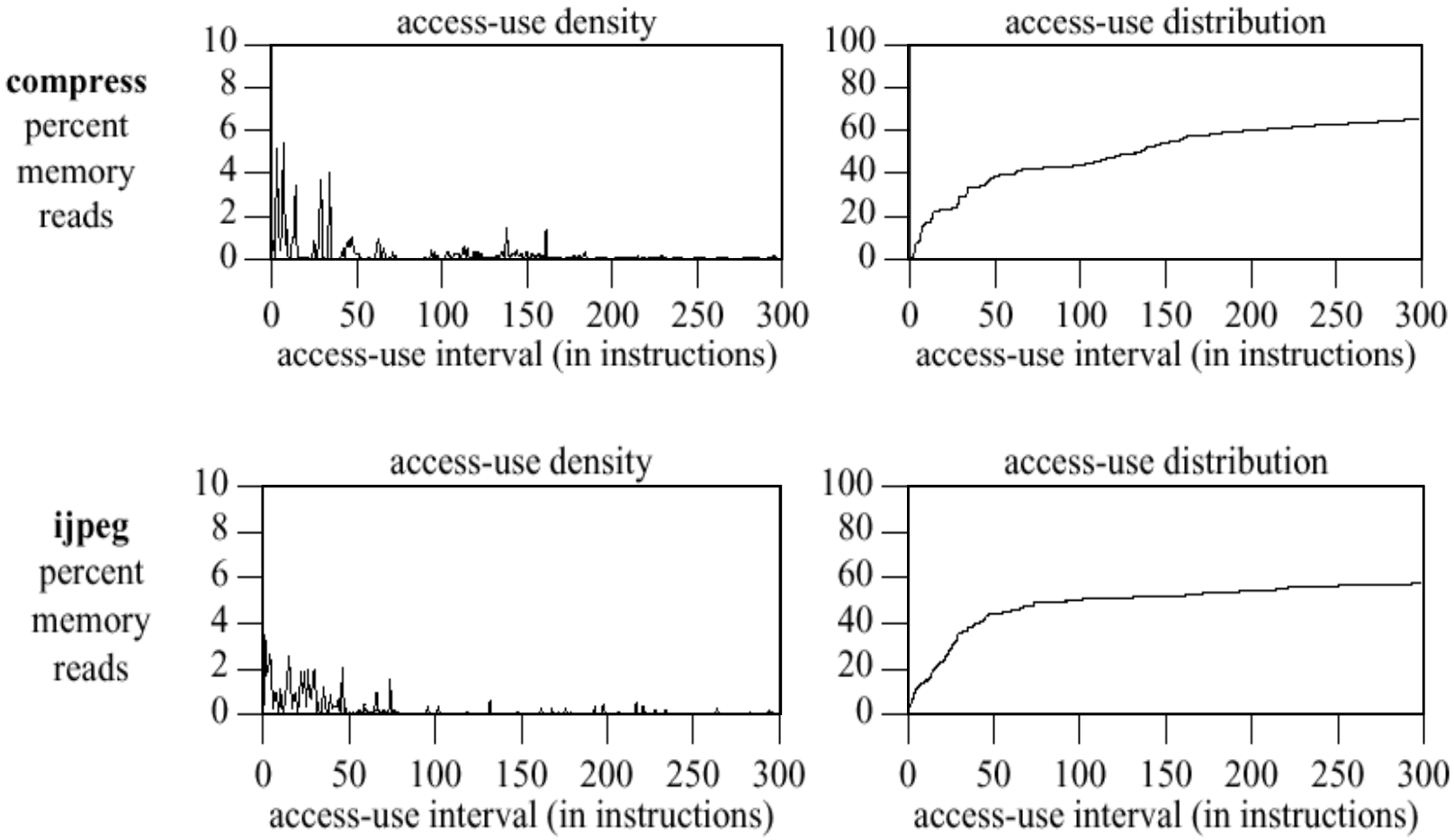
methodology

- **used 10 SpecInt programs**
  - 3 from SpecInt-95
    - GO, COMPRESS, IJPEG
  - 7 from SpecInt-2000
    - BZIP2, CRAFTY, GCC, GZIP, MCF, PARSER, VORTEX
- **compiled for MIPS-1 ISA using SGI native compiler ('-O')**
- **simulated execution for 600 M instructions**
- **data gathering after skipping the first 100 M**
- **defs are live coming into the data gathering window**
- **indeterminant (open) intervals were not counted**

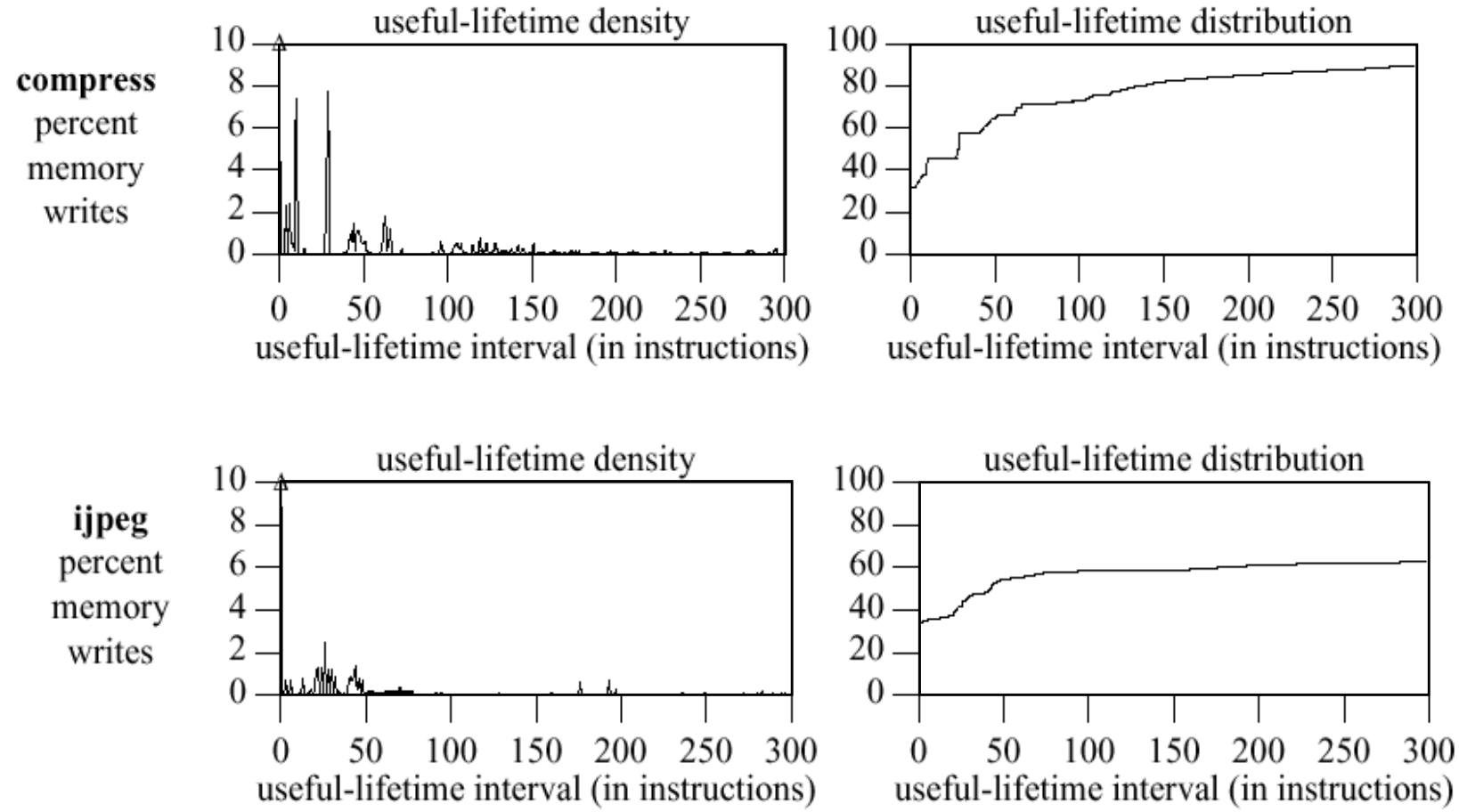
register results (overlaid)



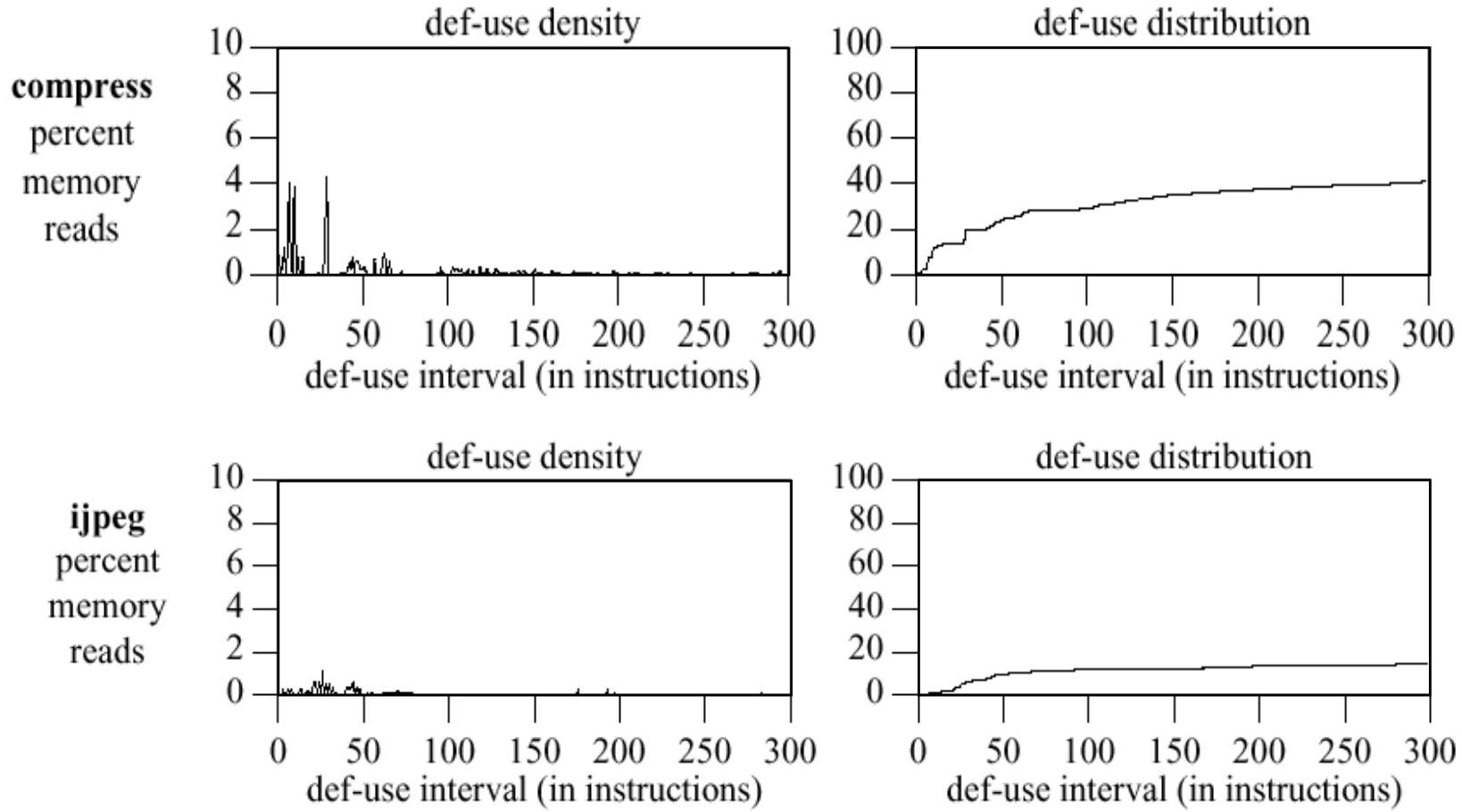
memory results (access-use)



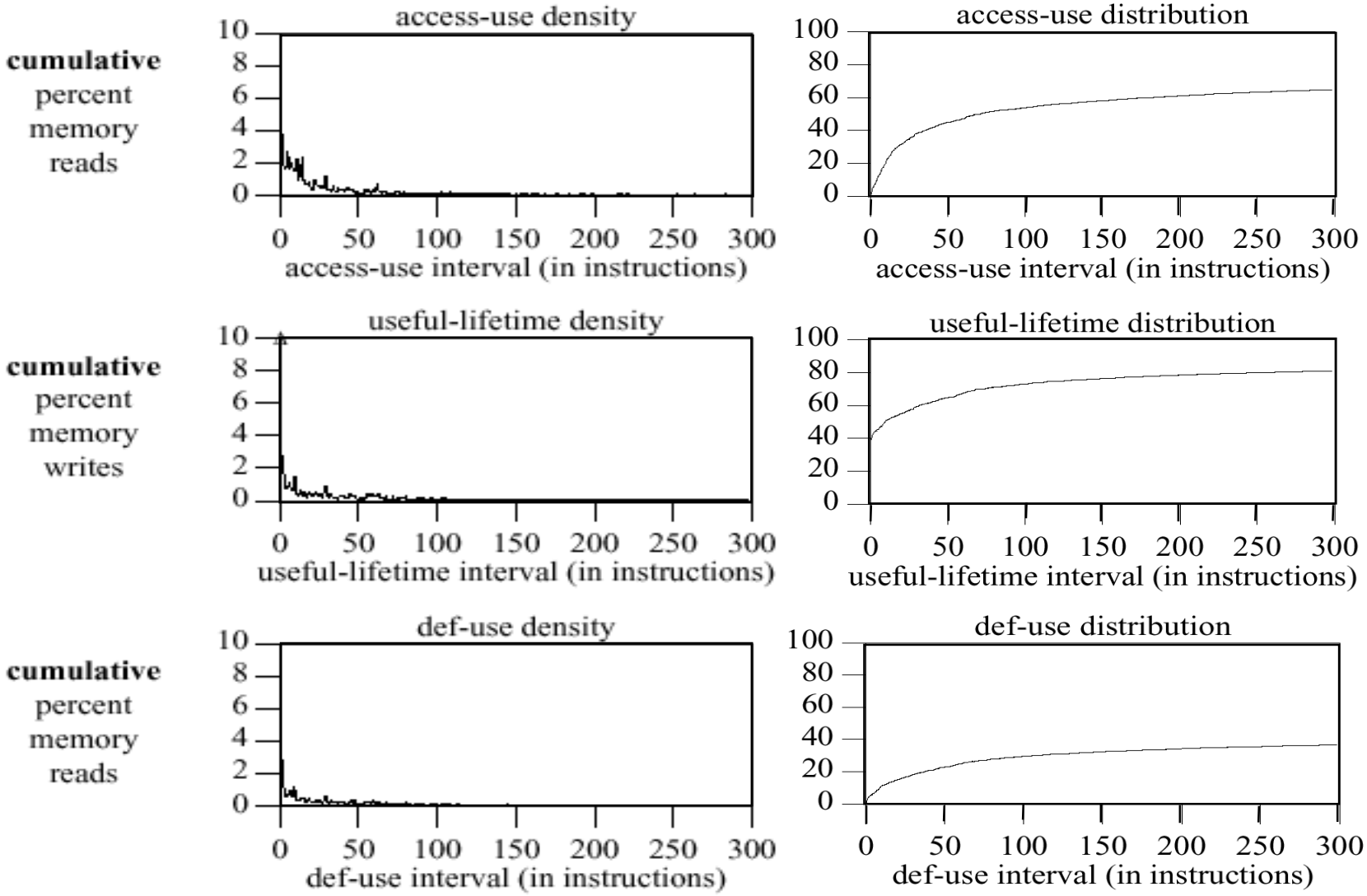
memory results (useful-lifetime)



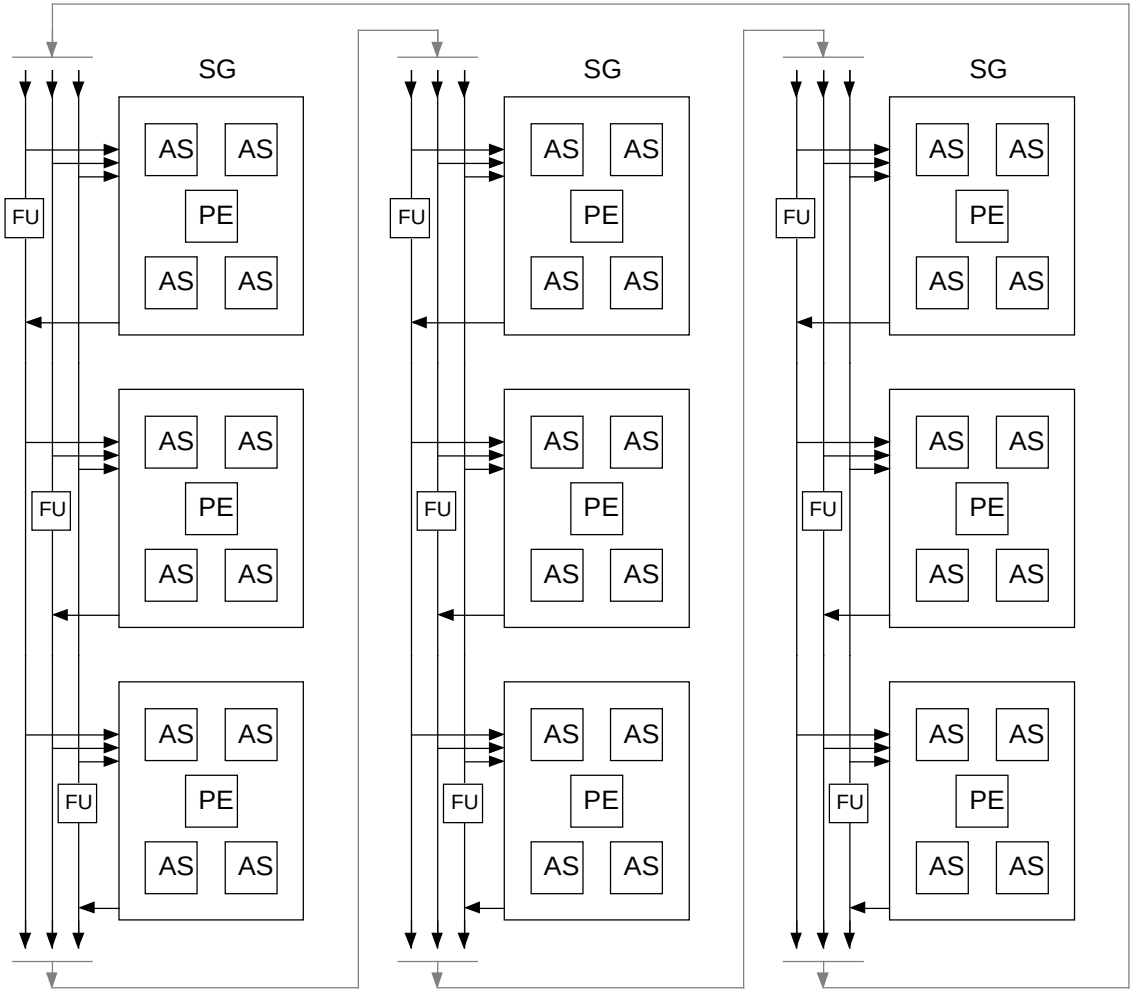
memory results (def-use)



memory results (cumulative)



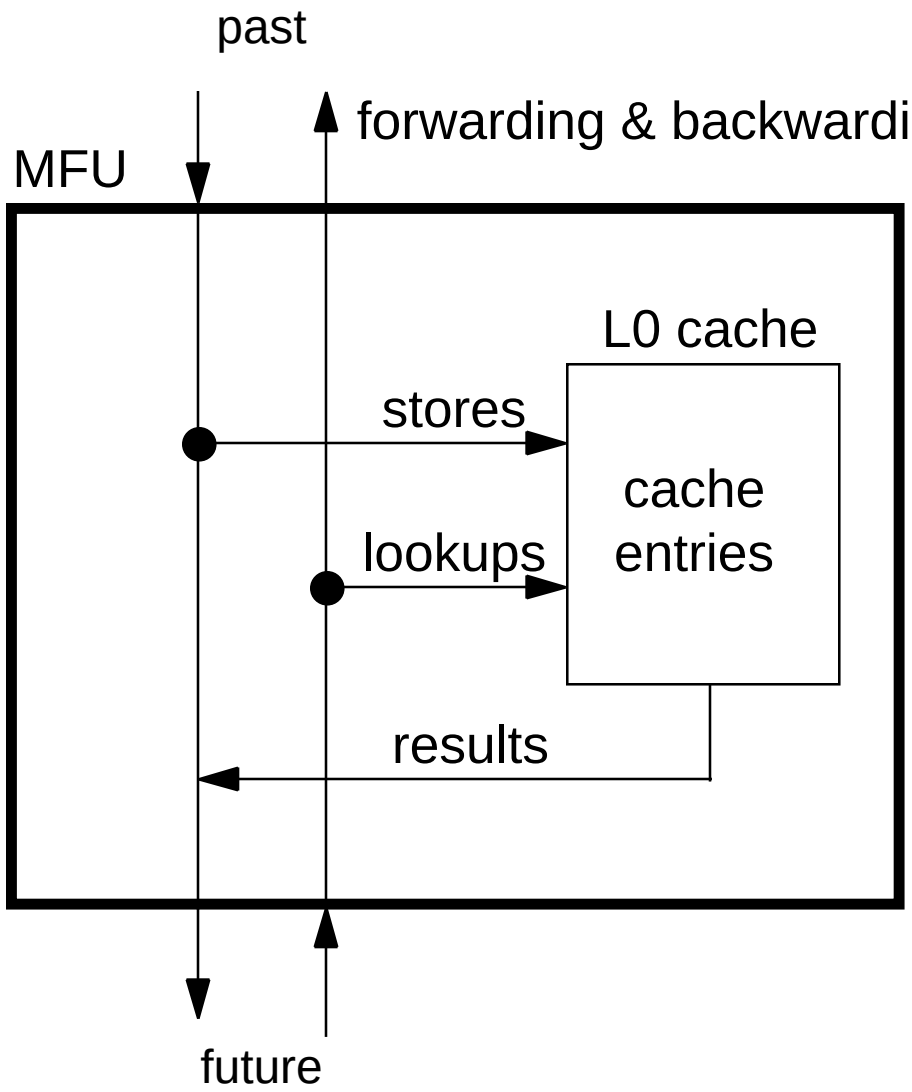
distributed microarchitecture



- **shown: 3 SG rows, 2 AS rows per SG, 3 SG columns**

- **ASes and PEs are grouped into Sharing Groups (SGs)**
- **buses are broken into fixed length segments by Forward Units (FUs)**
- **there are different type FUs for each operand type :**
  - register -- RFU
  - memory - MFU
  - predicate -- PFU
- **both forwward and backward interconnections are provided (backward is not shown)**
- **L0 caches are distributed throughout -- one inside each of the MFUs**

a distributed L0 cache



- **MFU provides**
  - forwarding function
  - backward propagation for requests
  - filtering function
  - cache storage through L0 to satisfy backwarding requests
- **L0 may be a variety of configurations**
  - we employed fully-associative but others are possible

default machine parameters

L0 cache	32 entries, one word each, fully associative
L1 I/D cache	1 clock hit latency
	64 KB, 32 byte block size
	2-way set associative
L2 cache	unified I/D
	10 clock hit latency
	2 MB, 32 byte block size
	direct mapped
main memory	100 clock latency
memory interleave factor	4
forward units (all)	minimum delay 1 clock
forwarding buses (all)	delay 1 clock
	4 in parallel
branch predictor	PAg. 1024 PBHT, 4096 GPHT, 2-bit sat. counter

Memory Bypass Results

memory loads satisfied with bypass

geometry	8-4-8	8-8-8
bzip2	45%	46%
compress	35%	28%
crafty	34%	37%
gcc	24%	22%
go	27%	22%
gzip	54%	53%
ijpeg	14%	13%
mcf	23%	24%
parser	39%	35%
vortex	50%	49%
MEAN	34%	33%

geometry:
- **SG rows**
- **AS rows per SG**
- **SG columns**

latencies:
- **L1 1 ck**
- **L2 10 cks**
- **main memory 100 cks**

summary

- **we have investigated the opportunity for register and memory bypass in general sequential programs**
  - register intervals are very short (~ 90% less than about 30 instructions)
  - memory intervals are much longer (as expected) but still provide opportunity for bypass of L1 and LSQ with about 60% of them less than about 300 instructions (access-use interval)
  - this matches well to the size of upcoming instruction windows
- **bypass results on our microarchitecture**
  - we achieved significant bypass of L1 D-cache and LSQ through the use of distributed L0 caches
  - from 13% (IJPEG) to 53% (GZIP) of all memory loads bypassed the L1 D-cache or LSQ
  - mean bypass of about 33% of all memory loads