Decoupling Address Generation from Loads and Stores to Improve Data Access Energy Efficiency

Michael Stokes, Ryan Baird, Zhaoxiang Jin*,
David Whalley, Soner Onder*

Computer Science Department
Florida State University

*Computer Science Department
Michigan Technological University

June 19, 2018

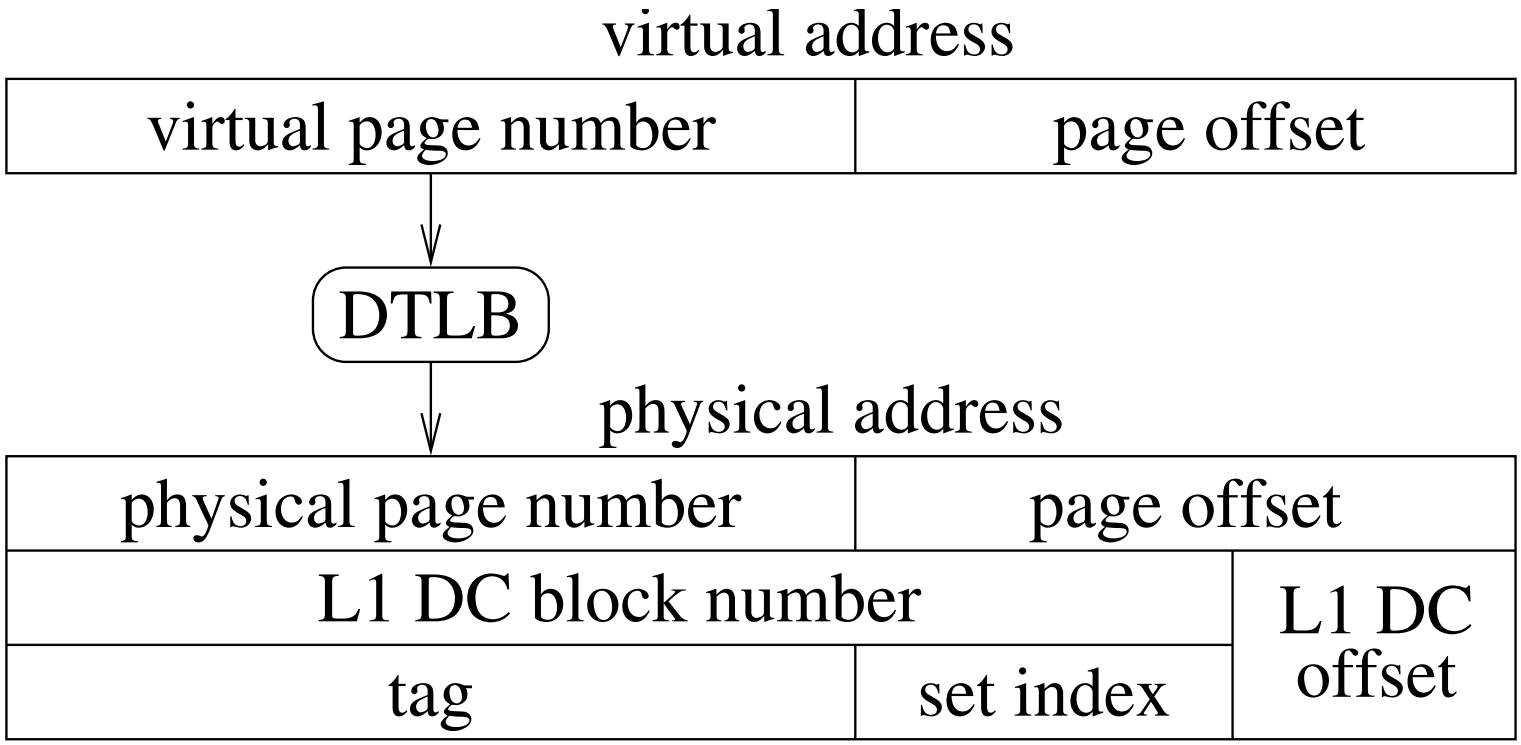
Motivation

- Energy Efficient Processor Design
  - Extend battery life
  - Reduce generated heat
  - Reduce energy cost
- DAGDA is a technique that reduces data access energy
- Achieves set-associative cache access hit-rate with direct-mapped cache access energy without increasing access time

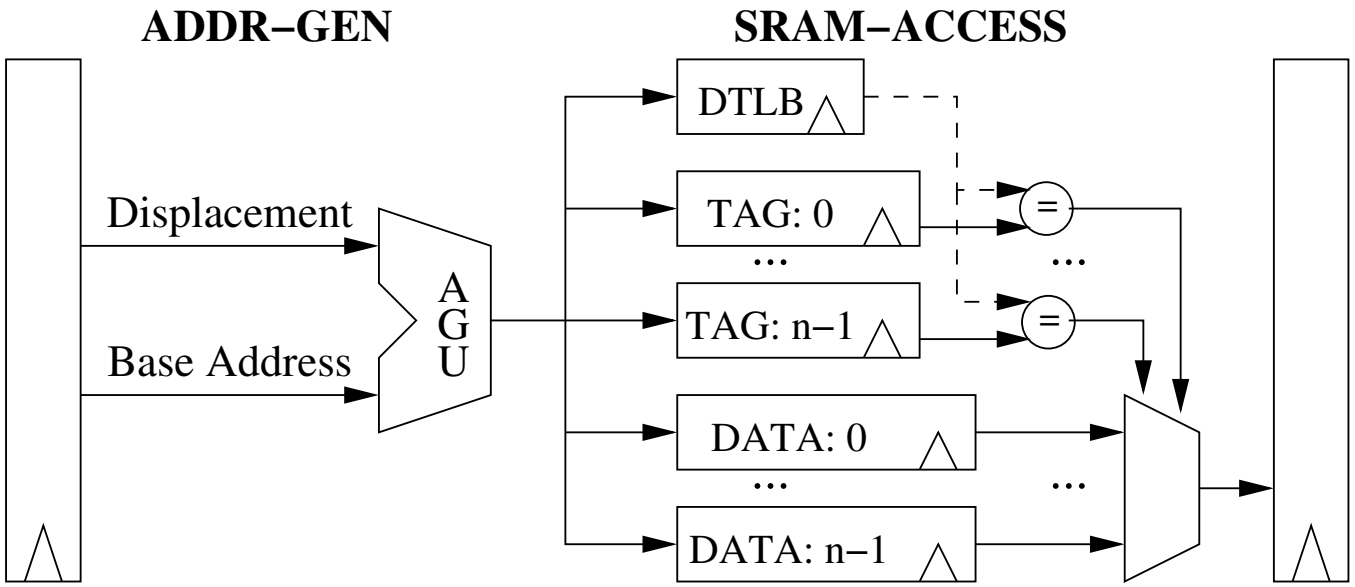
Set-Associative Cache Access

- A traditional set-associative cache access must perform the following steps:
  - Calculate the virtual address by adding the register and offset
  - Translate the virtual address to a physical address by accessing the DTLB
  - Determine the correct way by comparing the tag portion of the physical address with the tags associated with the ways of the set
  - Access the desired word from the appropriate way, if the tag comparison is a hit

VIPT Cache Access Overview

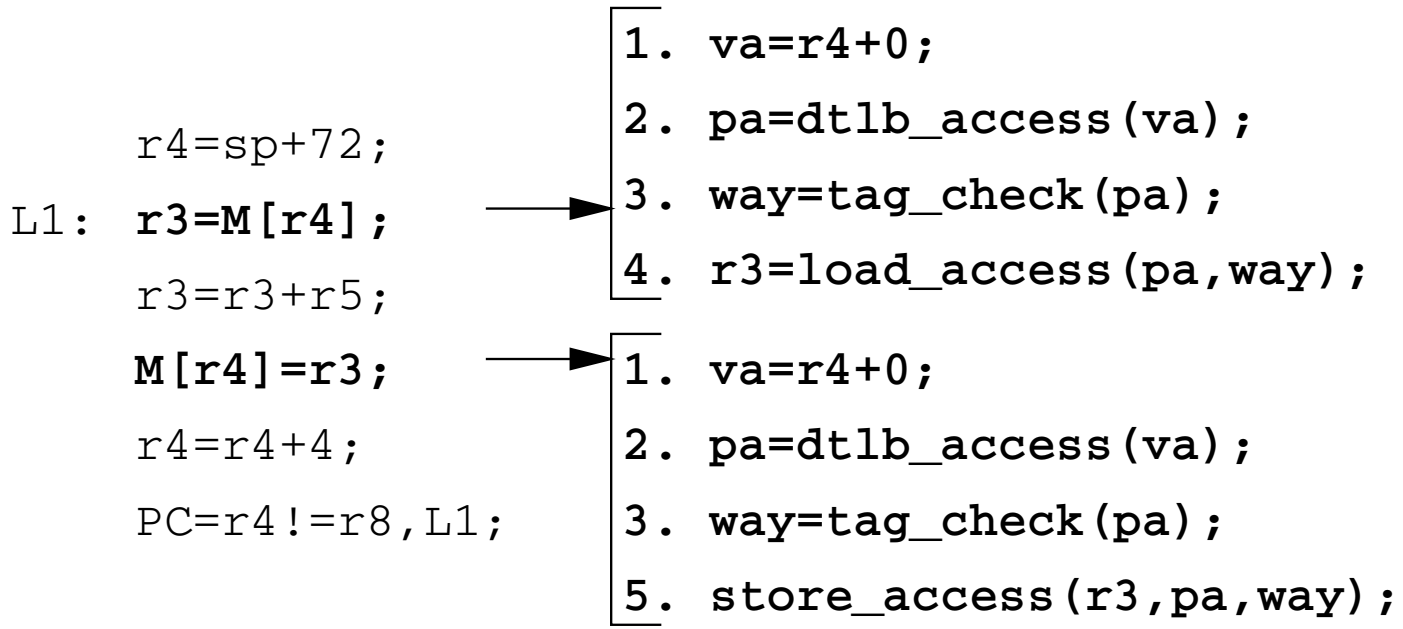


VIPT Cache Access

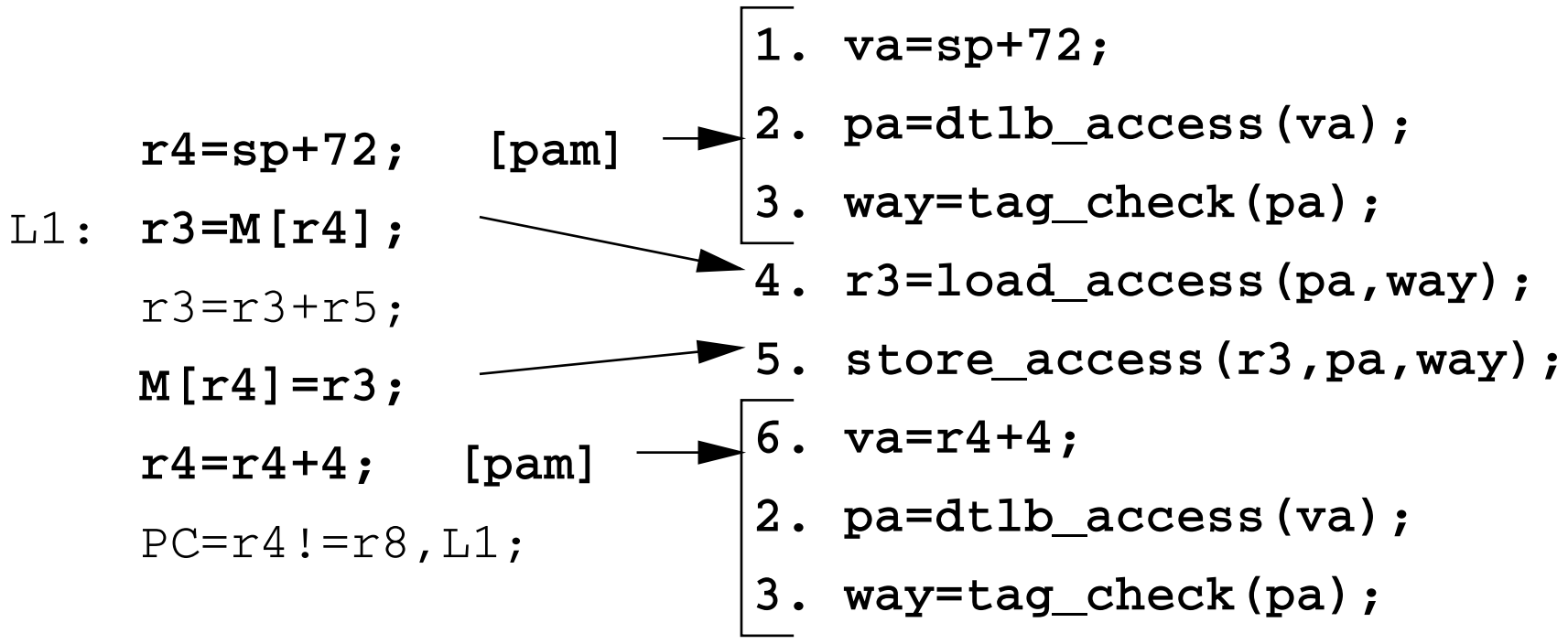


- A virtually-indexed, physically-tagged cache accesses the DTLB, tag array, and data arrays in parallel
- This removes the DTLB and tag array from the critical path

Conventional Micro-Operations

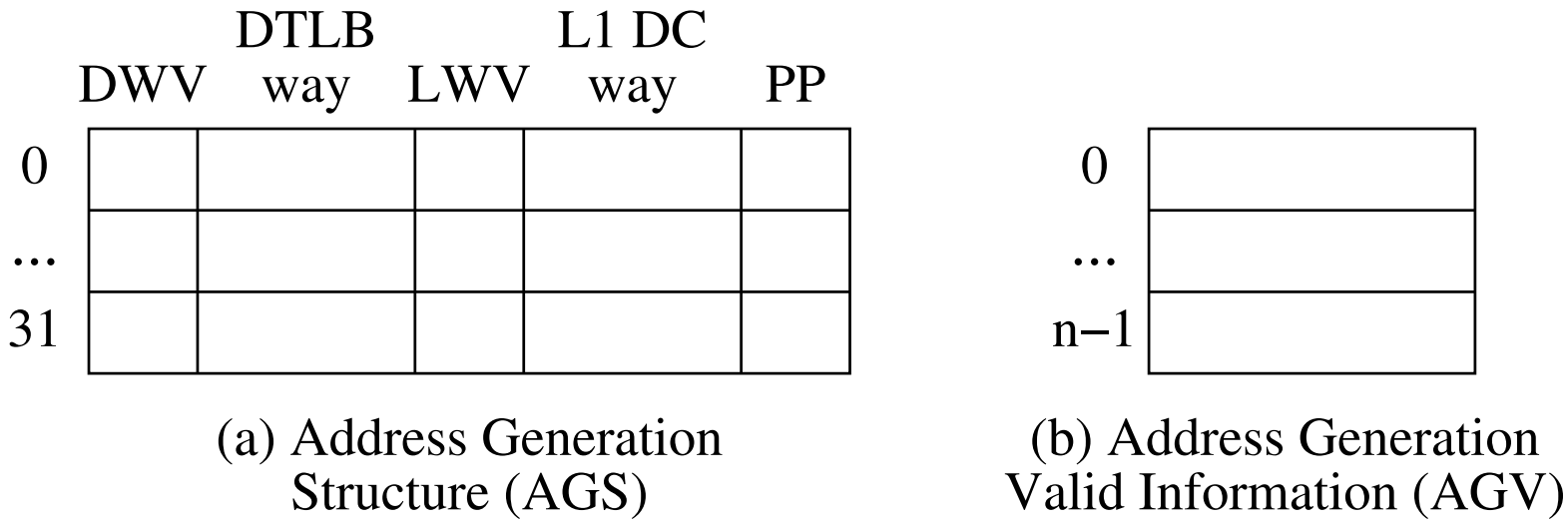


Decoupled Micro-Operations



Memoizing Cache Access Information

- Saving cache-access information requires a new structure
  - A PAM operation associates this information with the destination register
  - A load/store operation uses this information associated with its source register



(a) Address Generation Structure (AGS)

(b) Address Generation Valid Information (AGV)

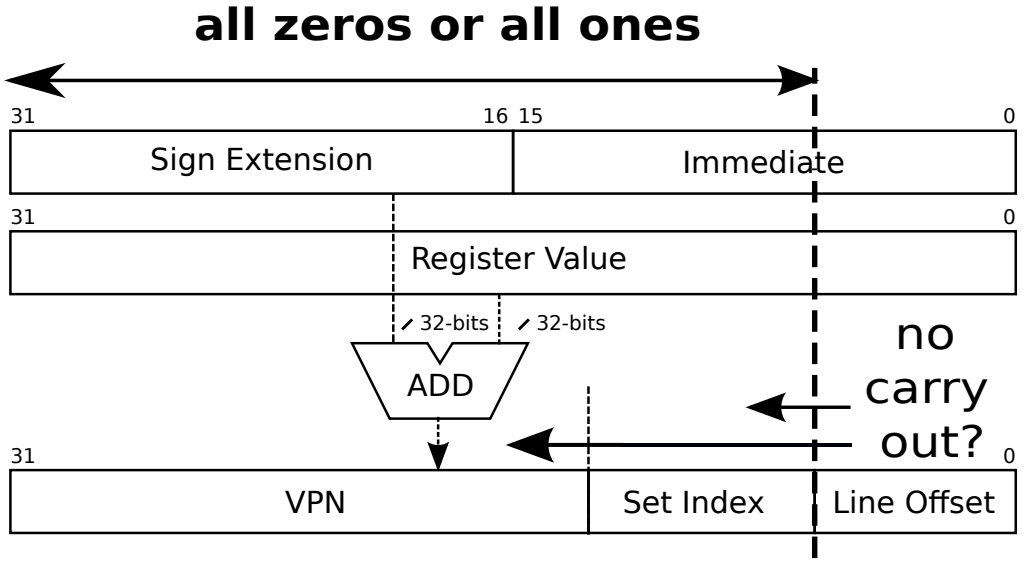
Avoiding Redundant DTLB and Tag Array Accesses

- Often, the PAM instruction's calculated virtual address shares the same line as the source register
- If so, the DTLB access and L1 DC tag check can be avoided

```
      r20=...;[pam]
L3:   r2=M[r20];
      ...
      r20=r20+4;[pam]
      PC=r20!=r21,L3;
```

Detecting AGS Re-Use

- If we're adding a positive value and there *is no* carry out from the offset field (set index), the calculated address shares the same line (page) as the source register
- If we're adding a negative value and there *is* carry out from the offset field (set index), the calculated address shares the same line (page) as the source register



Pipeline Modifications

- In a traditional MIPS pipeline, the EX stage calculates the effective address prior to a memory access
- With DAGDA, we calculate the effective address in the prepare-to-access memory (PAM) instruction
- Therefore, we can place the memory access stage before the EX stage.

DAGDA Stages Used by Instructions

- The DAGDA pipeline can perform an operation on the loaded value

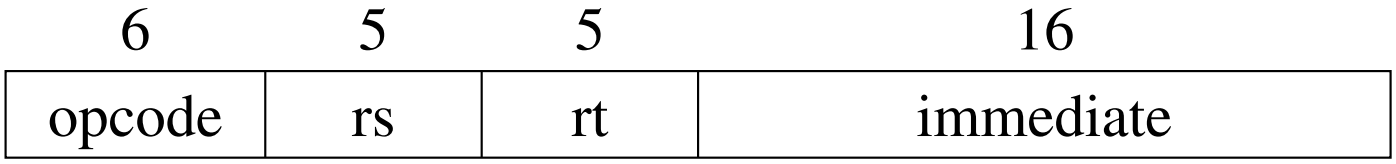
Instruction	Pipeline Stages					
ALU inst	IF	ID	RF	*DA*	EX	WB
pam ALU inst	IF	ID	RF	**AG**	**TC**	WB
load inst	IF	ID	RF	DA	*EX*	WB
pam load inst	IF	ID	RF	DA	**TC**	WB
store inst	IF	ID	RF	DA	EX	WB

DAGDA Instruction Pipeline Example

- One instruction needs to be placed between a PAM instruction and a load to avoid a stall

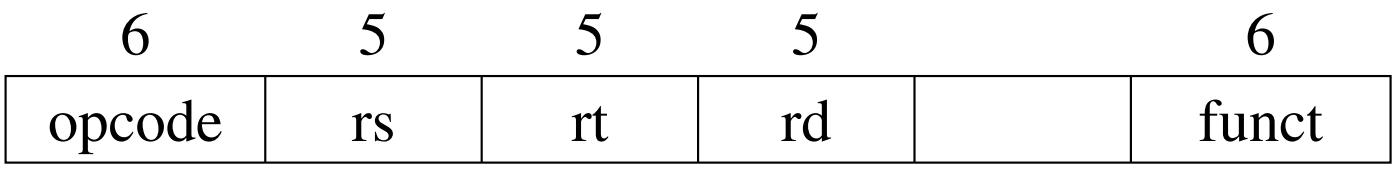
Instruction	1	2	3	4	5	6	7	8	9	10
1. pam add	IF	ID	RF	**AG**	**TC**	WB				
2. other		IF	ID	RF	DA	EX	WB			
3. pam load			IF	ID	RF	**DA**	**TC**	WB		
4. other				IF	ID	RF	DA	EX	WB	
5. load					IF	ID	RF	**DA**	EX	WB

New Instruction Format



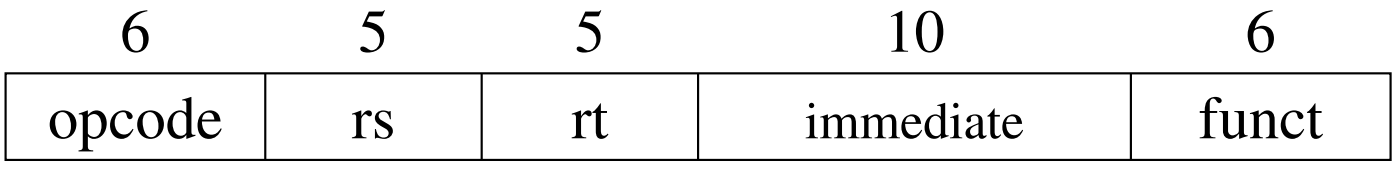
ex: rt=M[rs+immed]; # load

(a) Original MIPS I Format Used for Loads and Stores



ex: rd=M[rs]+rt; # load+addreg

(b) MIPS R Format Used with Loads




```
ex: rt=M[rs]+immed;              # load+addimmed
ex: rt=M[rs]; rs=rs+immed;       # load+postincr
ex: M[rs]=rt; rs=rs+immed;       # store+postincr
```

(c) New Short Immediate Format Used with Loads and Stores

Optimizations Using New Encoding

```
     ...
     PC=L2;
L1:  ...
     M[r7]=r3;
     ...
L2:  ...
     r7=r7+4; [pam]
     PC=r7!=r8,L1;
```

(a) Original Loop

```
     ...
     r7=r7+4; [pam]
     PC=L2;
L1:  ...
     M[r7]=r3; r7=r7+4; [pam]
     ...
L2:  ...
     PC=r7!=r8,L1;
```

(b) After Transformation

Benchmarks Used and Compiler

- MiBench benchmarks used
- The VPO (Very Portable Optimizer) was used to compile the benchmarks

Category	Benchmarks
automotive	bitcount, qsort, susan
consumer	jpeg, tiff
network	dijkstra, patricia
office	ispell, stringsearch
security	blowfish, rijndael, pgp, sha
telecom	adpcm, CRC32, FFT, GSM

Processor and Cache Configuration

- Processor Configuration

page size	8KB
L1 DC	32KB size, 4 way associative, 1 cycle hit, 10 cycle miss penalty
DTLB	32 entries, fully associative

- The ADL simulator was used to estimate the results
  - Simulator was modified to capture pipeline stalls.
  - Single cycle stall for a PAM-followed-by-load hazard (DAGDA)

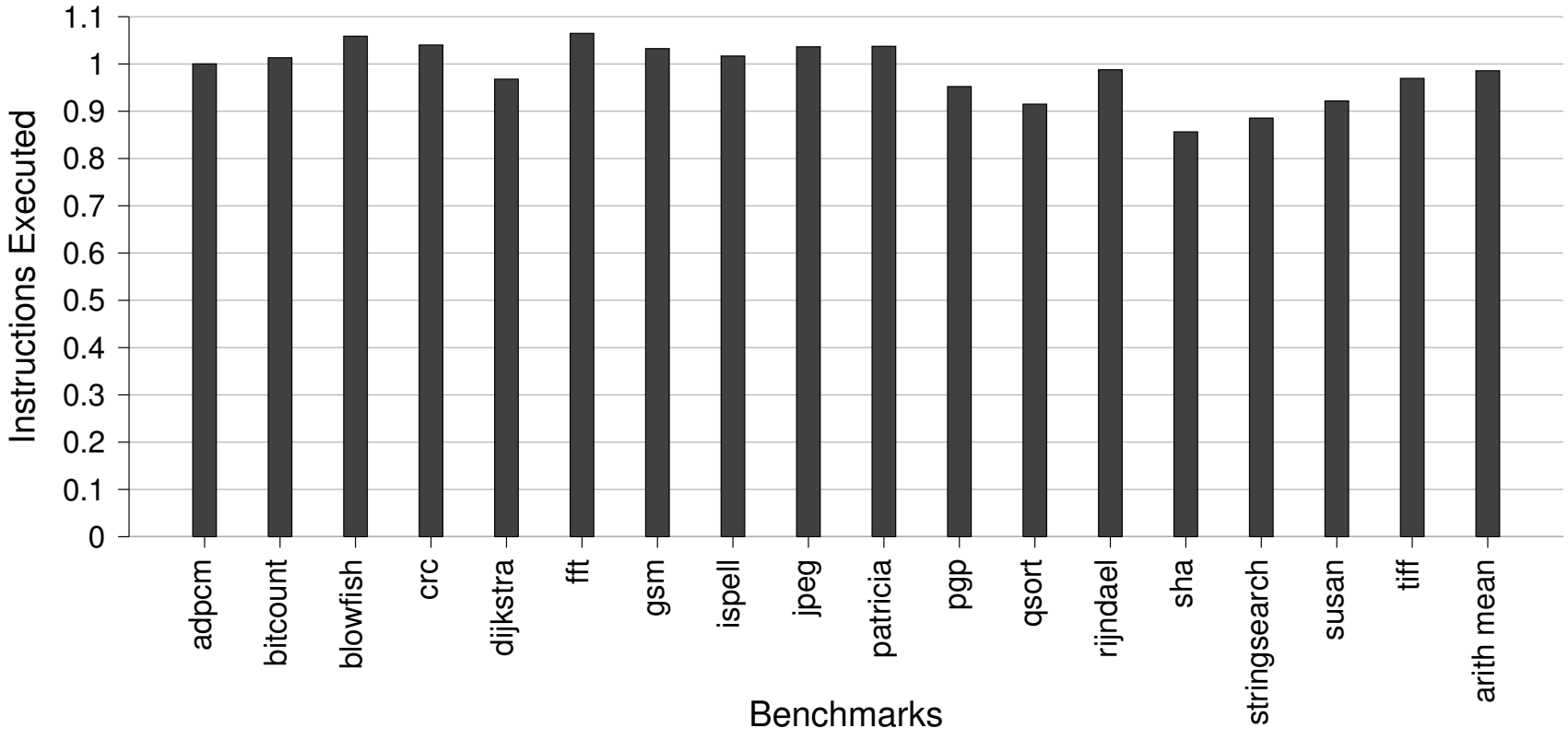
L1 DC and DTLB Component Energy

- Used CACTI to estimate the L1 DC and DTLB energy
- Used a 22-nm CMOS process technology with LSP

Component	Energy
Read L1 DC Tags - All Ways	0.782 pJ
Read L1 DC Data - All Ways	8.236 pJ
Write L1 DC Data - One Way	1.645 pJ
Read L1 DC Data - One Way	2.059 pJ
Read DTLB - Fully Associative	0.823 pJ
Read DTLB - One Way	0.215 pJ
Write AGS - 1 Entry	0.320 pJ
Read AGS - 1 Entry	0.147 pJ
Write AGV - 1 Bit in All 4 Entries	0.240 pJ
Read AGV - 32 Bits in All 4 Entries	0.500 pJ

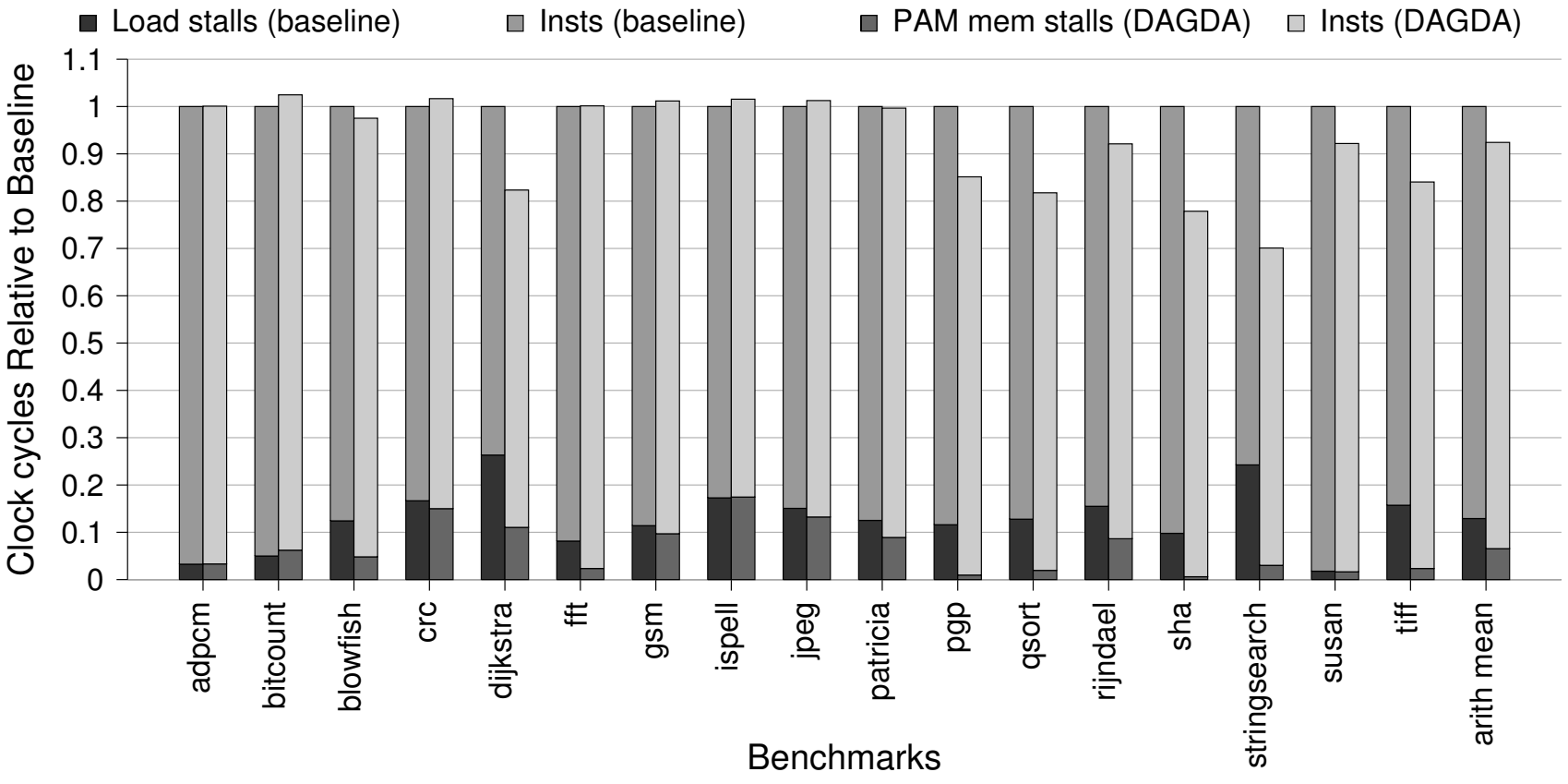
Instruction Count Impact

- The instructions executed was reduced on average by 1.4%



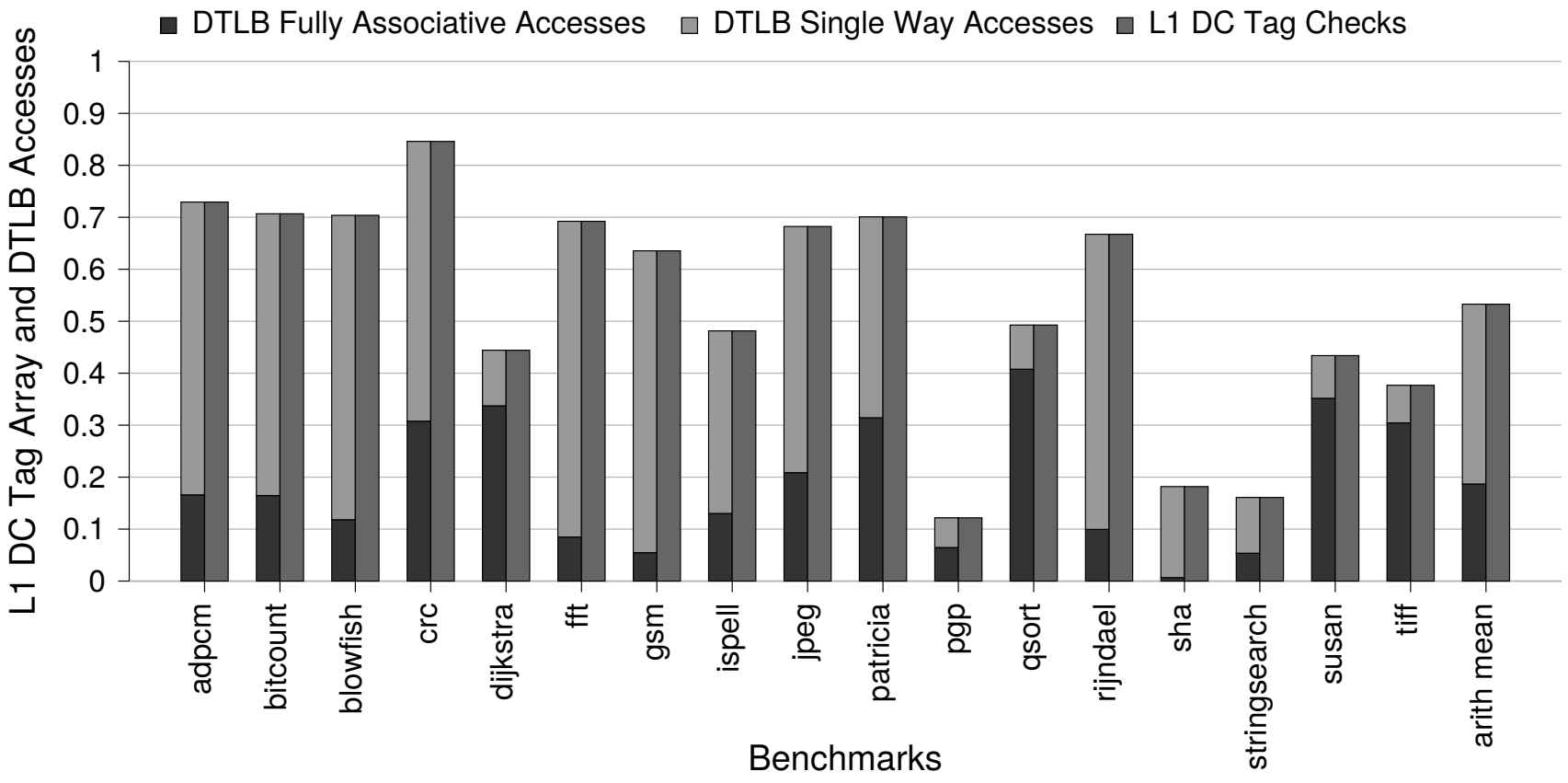
Cycle Count Impact

- The cycle count was reduced on average by 7.6%



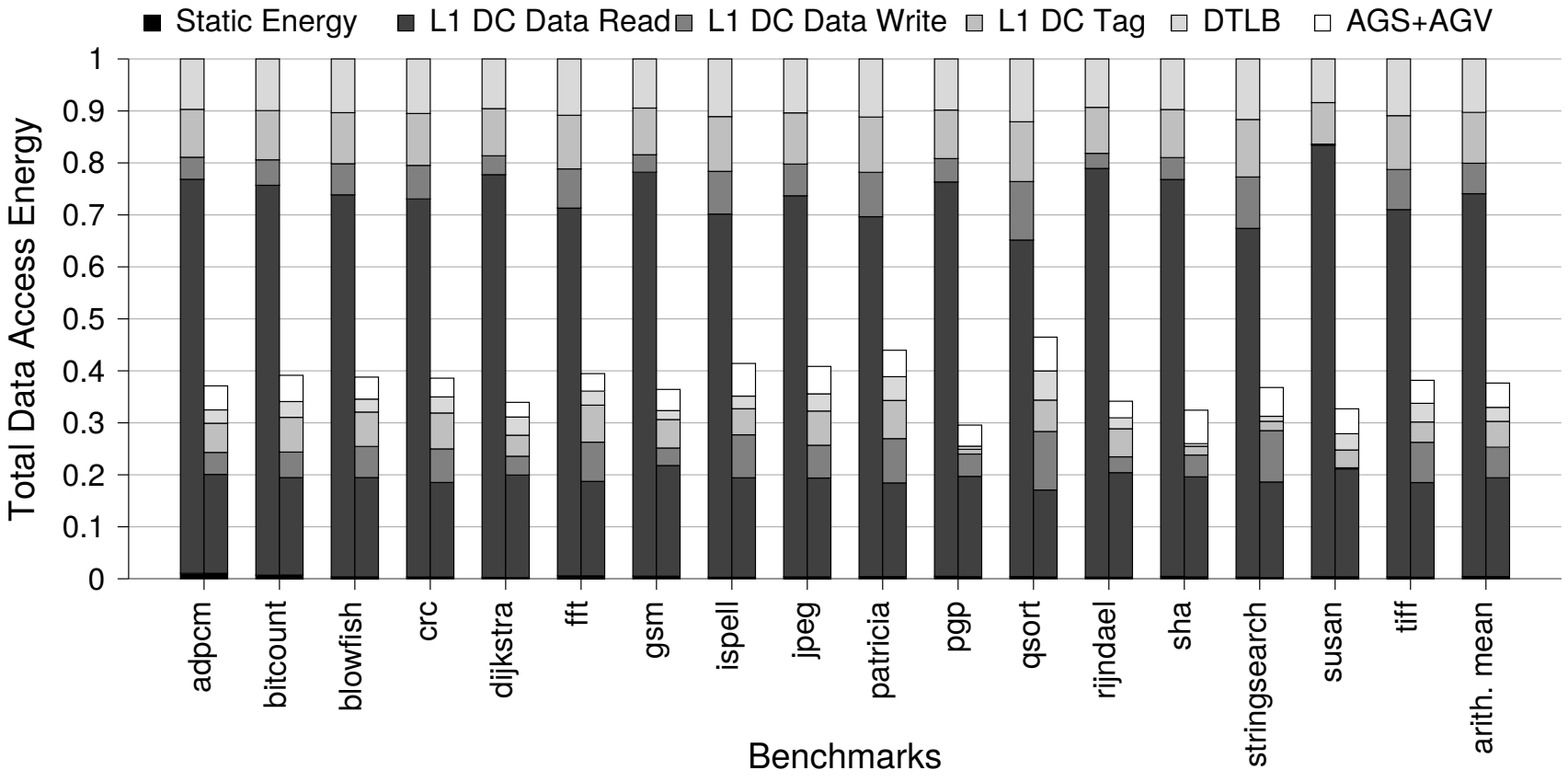
L1 DC Tag Array and DTLB Accesses

- L1 DC tag checks were avoided 47% of the time and fully associative DTLB accesses were avoided 82% of the time



Data Access Energy

- The total data access energy was reduced by 62%



Conclusions

- DAGDA reduces data access energy by enabling loads to directly access a single data array way of a set-associative cache and by avoiding a large fraction of L1 DC tag checks and DTLB accesses
- DAGDA is able to offer performance improvements with its modified ISA
  - The total number of instructions executed is reduced
  - PAM operations can prefetch data accesses into the L1 DC in the case of an L1 DC miss

Questions?