Direct Addressed Caches for Reduced Power Consumption

Emmett Witchel   Sam Larsen   C. Scott Ananian   Krste Asanovic

MIT Lab for Computer Science

The Domain

- We are attempting to reduce power consumed by the caches and memory system.
  - Not discs or screens.
  - 16% of processor + cache energy for StrongARM is dissipated in the data cache.
- We focus on the data cache. The instruction cache is amenable to hardware-only techniques.
- We are interested in power optimizations that are not just existing speed optimizations.
- Exploit compile time knowledge to avoid runtime work.
  - Partially evaluate a program for certain hardware resources.
- We show how software can eliminate cache tag checks which saves energy.

The First Problem — Cache Tags

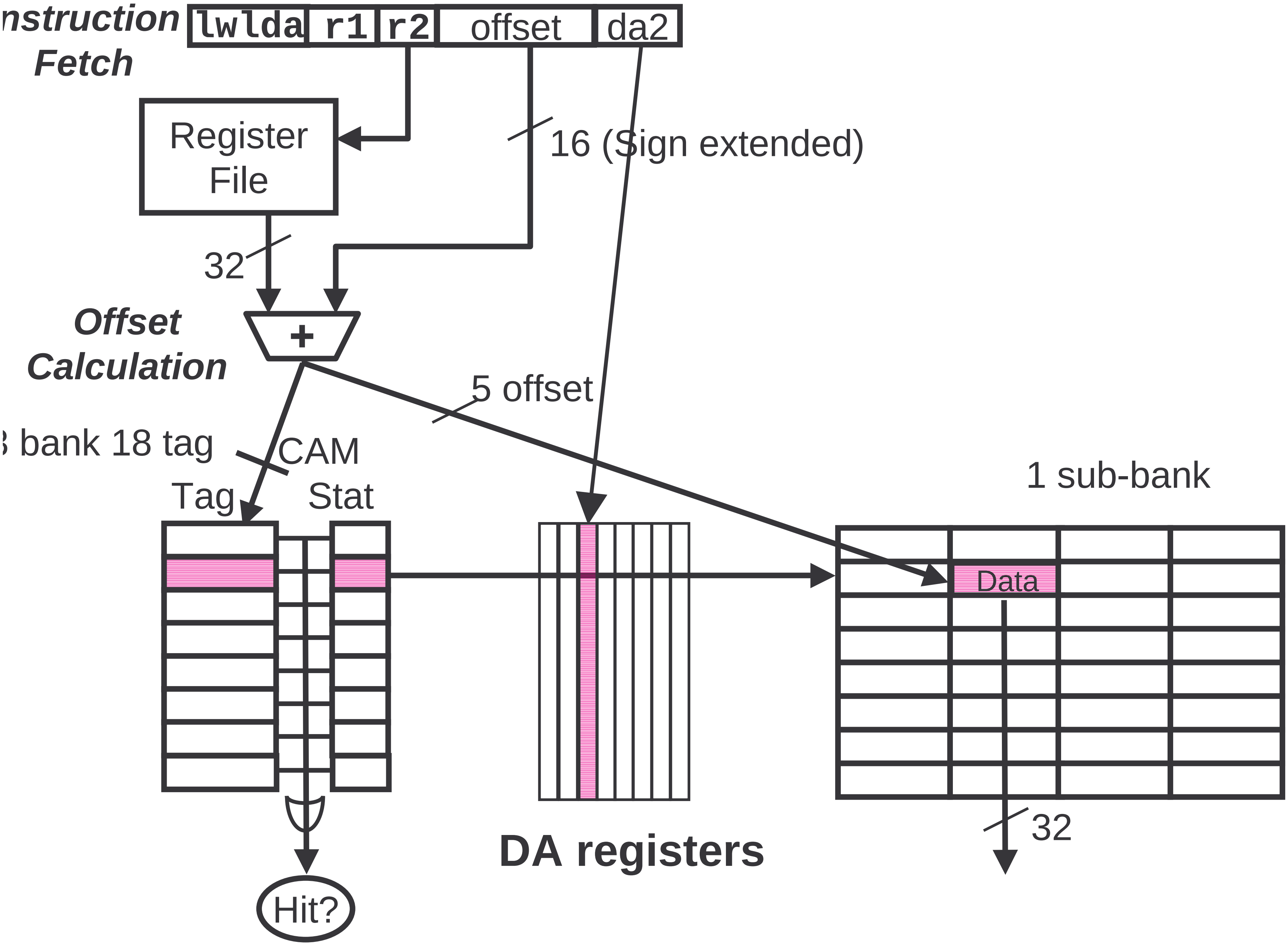
Direct Mapped	Set-Associative	CAM-tag
Each memory location has a unique home.	Each memory location has a small number (e.g., 4) homes.	Each memory location can be anywhere in a sub bank.
High miss rates which means high energy usage.	Moderate miss rates.	Lowest miss rates.
Individual accesses are low power.	Individual accesses are high power because of multiple tag and data reads.	Individual accesses are moderate power. Most of the energy is in the tag check.

- **Both set-associative and CAM-tag caches spend the majority of their energy in the tag check.**

The Solution — Pass Software Information To Hardware

- The compiler often knows when the program is accessing the same piece of memory. Don’t check the cache tags for the second access.
- HW challenge — make this path low power.
- SW challenge — find the opportunities for use.
  - Two compiler algorithms for two languages (C and Java).
- Interface challenge — minimize ISA changes, don’t disrupt HW, don’t expose too much HW detail.
  - New flavors of memory ops are a common ISA change.
- Security challenge — Protect process data from other processes.
  - Snoop on evicts, detect invalid state early in pipeline

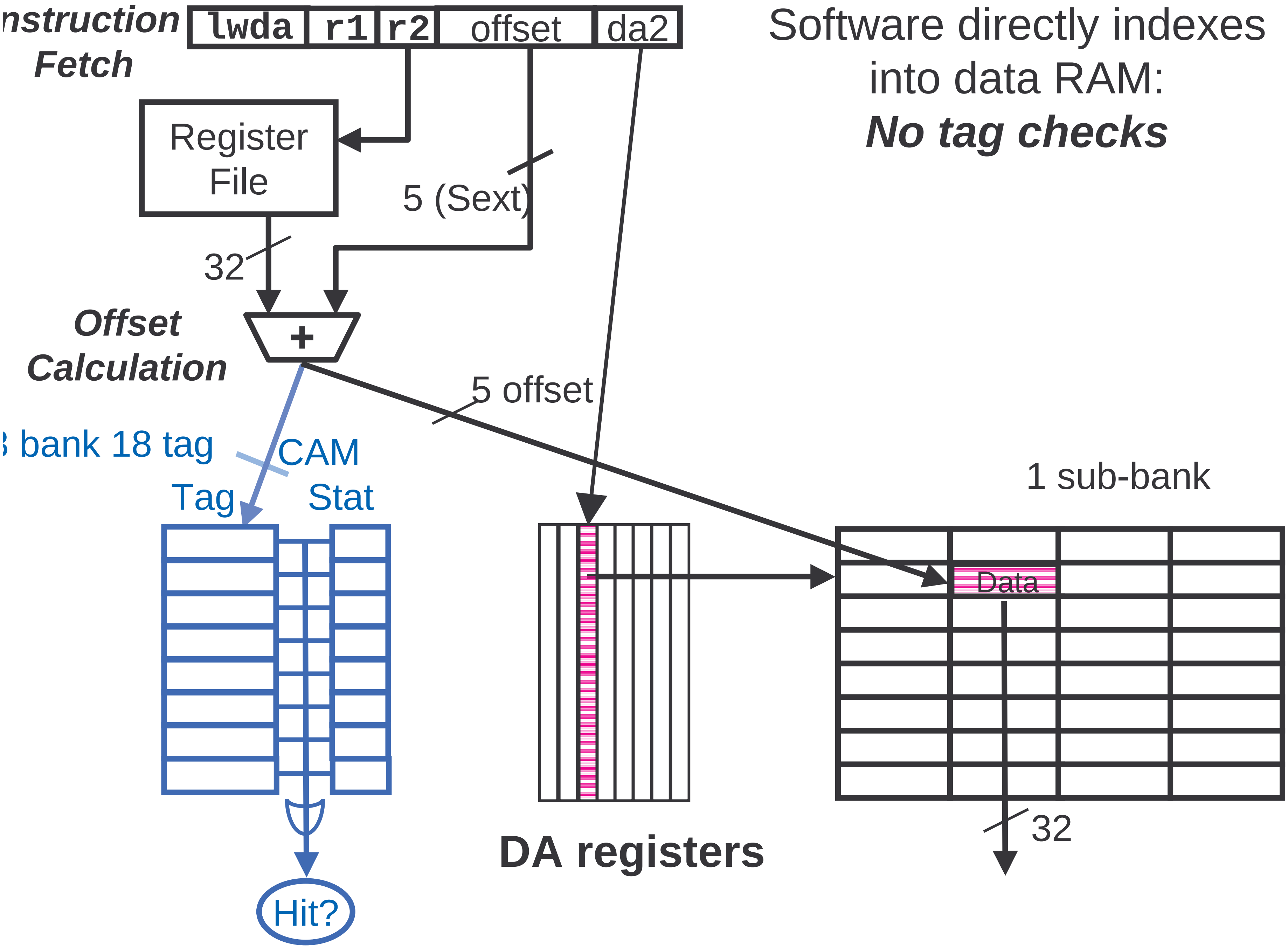
Direct Addressed CAM Tag Cache
Virtually Indexed & Tagged



Direct Addressing

Software directly indexes into data RAM:

No tag checks



Spill Code Using Direct Address Registers

- Old code

```
subu $sp, 64
sw   $ra, 60($sp)
sw   $fp, 56($sp)
sw   $s0, 52($sp)
```

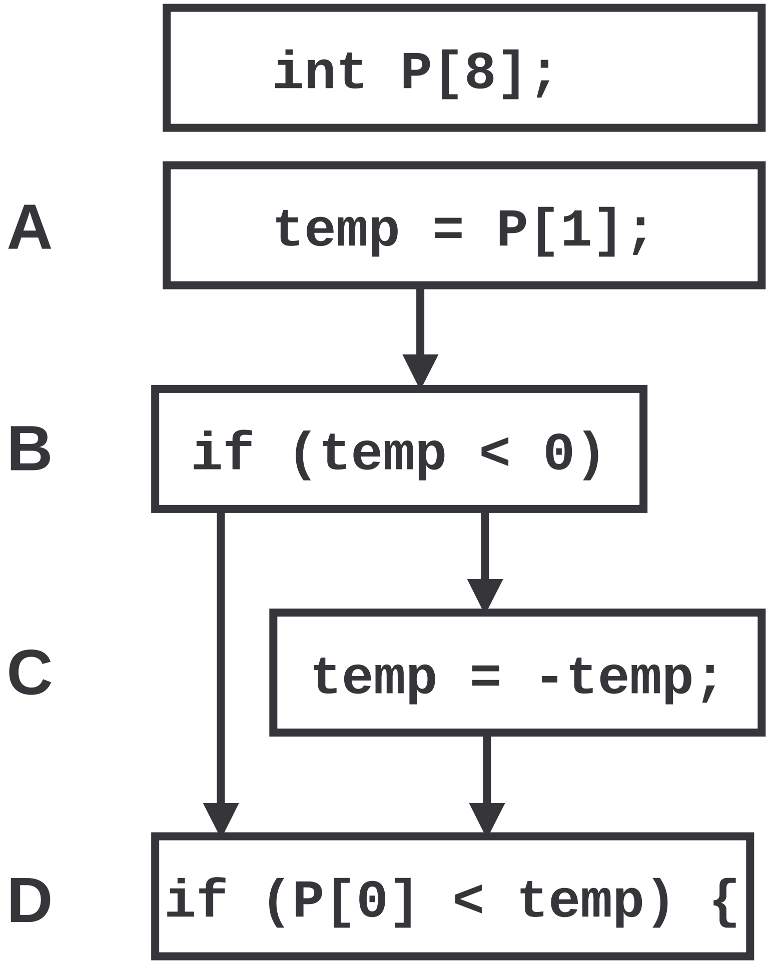
- Transformed code

```
subu  $sp, 64
swlda $ra,60($sp),$da0
swda  $fp,56($sp),$da0
swda  $s0,52($sp),$da0
```

- One tag check per line used for spilling.
- It is a simple transformation.
  - Similar to load/store multiple on StrongARM
    - Ld/st multiple is a limited model – can’t handle read-modify-write.
  - Hardware only schemes capture many references, but add latency.

Compiler Algorithm (C)

Code from gsm in mediabench



```
int P[8];

A  temp = P[1];

B  if (temp < 0)

C  temp = -temp;

D  if (P[0] < temp) {
```

- Find dominance relationship.
  - E.g., Read of P[1] in A dominates read of P[0] in D.
- Determine distance.
  - P[0] is offset –4 from P[1].
  - If dist == 0, done.
- Determine alignment.
  - Stack & static data are aligned by our backend.
  - Loop unrolling to increase alignment.
- Eliminate tag check in the read of P[0].

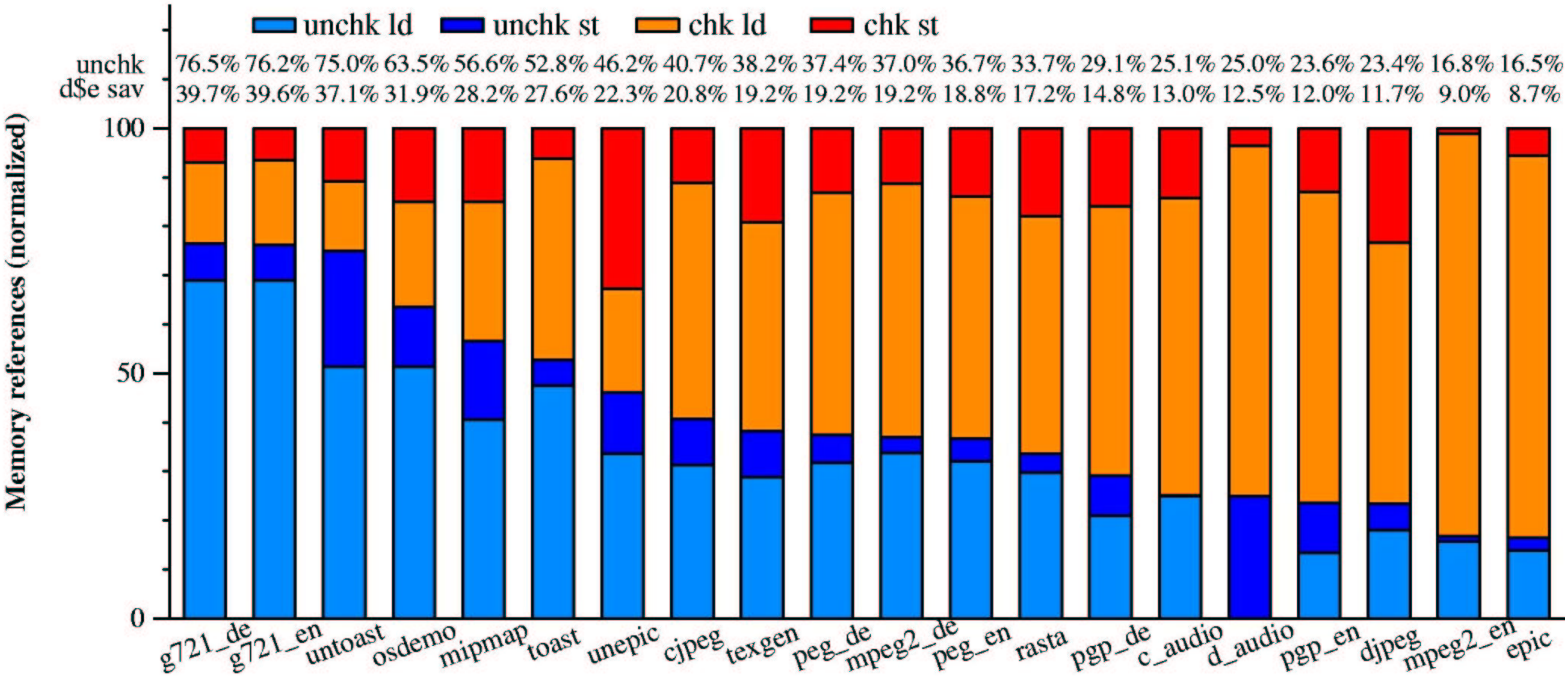
C Compiler Infrastructure

- We use SUIF, with a C backend.
- Loop unrolling to increase aligned references.
- Distance information from memory object offset.
  - Use simple, local information for aliases.
- Profile information to set pre-loop break condition.

```
for(i=0; i<N; i++) {
   A[i] = 0;
}
```

```
for(i=0; i<N; i++) {
   if(&A[I] % line_size == 0)
      break;
   A[I] = 0;
}
for(; i<N; i += 4) {
   A[i + 0] = 0; A[i + 1] = 0;
   A[i + 2] = 0; A[i + 3] = 0;
}
```

Results — C Implementation



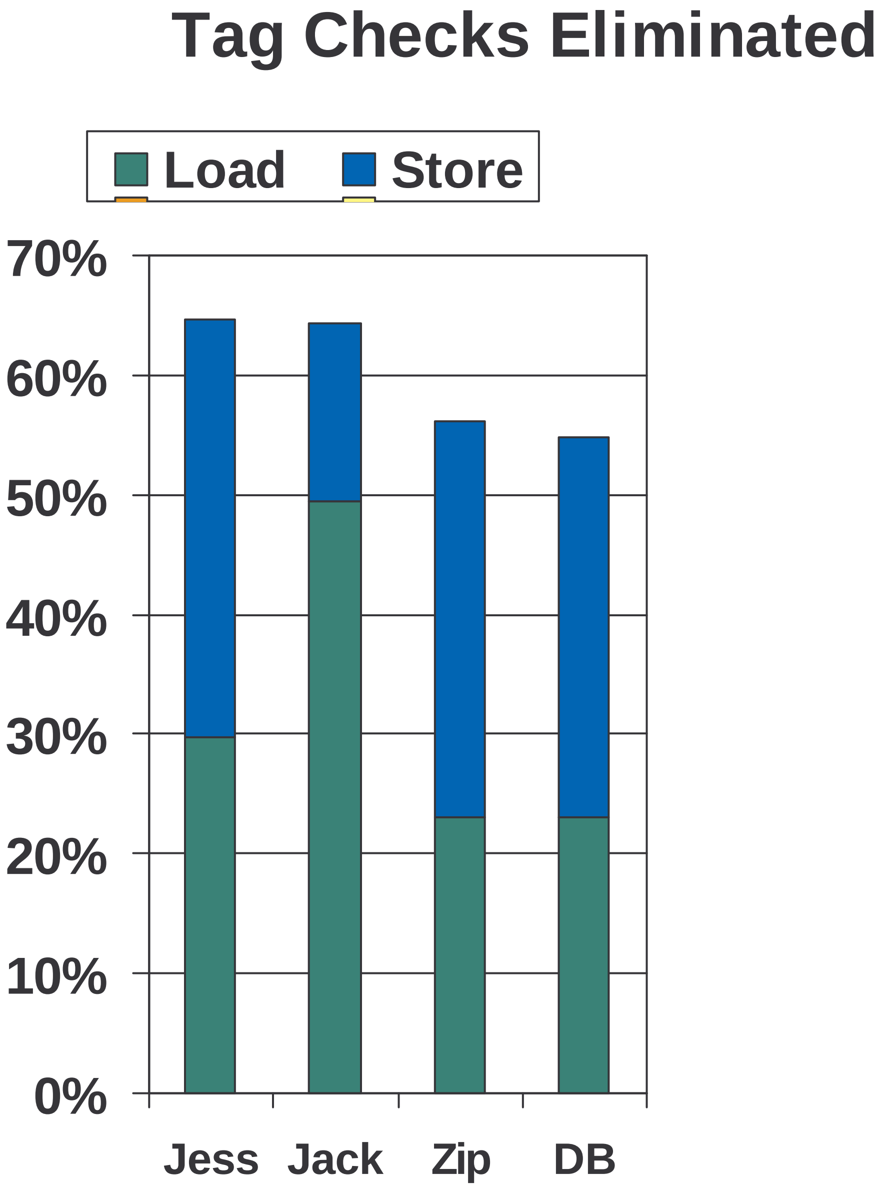
Mediabench

- **Data cache energy reduction 8.7 - 40%.**
- **Function entry/exit code not included — expect greater savings.**

Java Compiler Infrastructure

- FLEX is a bytecode to native compiler developed at MIT.
- We wrote a MIPS back end
  - Modified GNU as to accept new memory operations.
  - Modified ISA simulator to track DAR state.
- Loops are unrolled.
- Object type is tracked for additional opportunity.
  - Allows low level optimization of access to e.g., hash code.

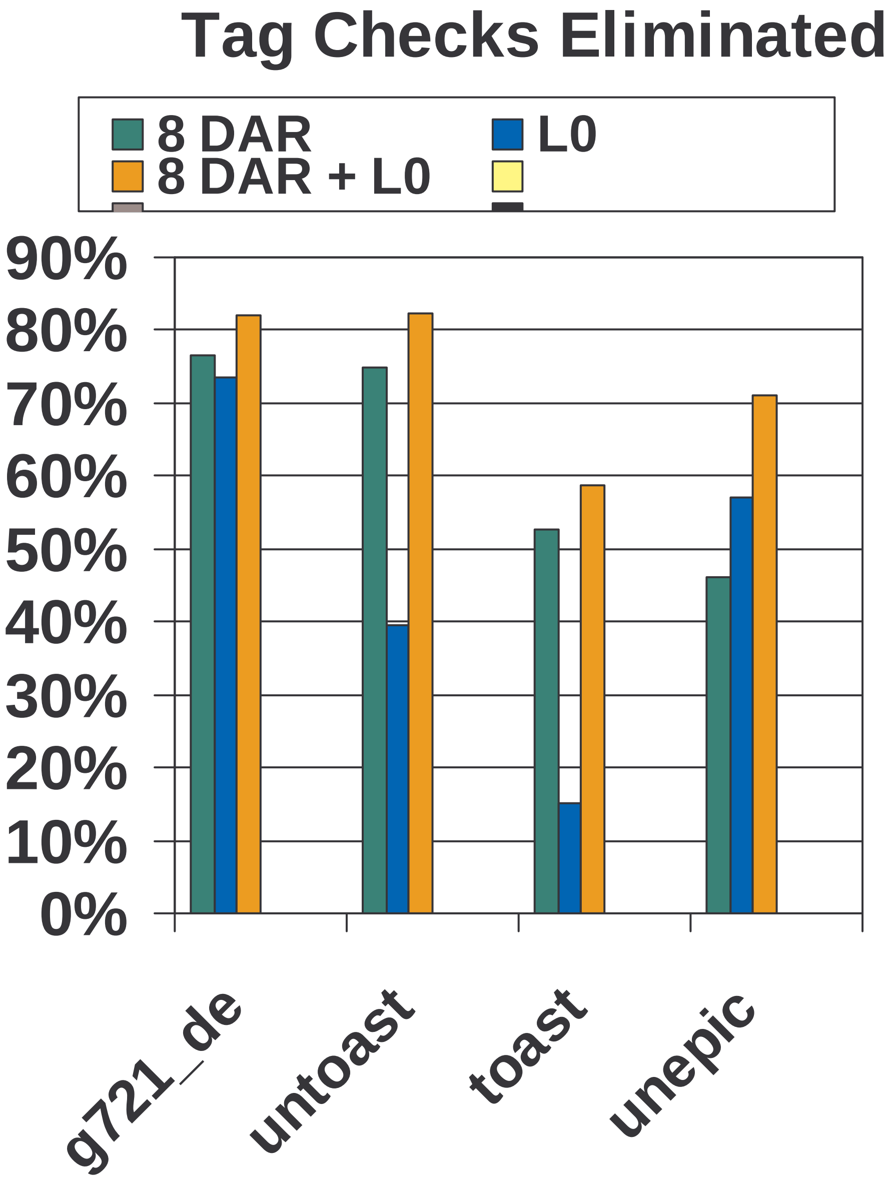
Results — Java Implementation



SPEC JVM ‘98

- One big advantage — function entry/exit code was transformed.
  - Calling convention modified.
- Data cache power savings 26-31%
- No profile feedback.

Results — Comparison with L0 Cache



Mediabench

- DARs usually tie L0 or exceed it.
- When L0 exceeds DARs, DARs help L0.

Related Work

- **Fisher & Ellis used loop unrolling to reduce memory bank conflicts.**
    - **Barua expanded the work with Modulo Unrolling.**
- **Burd and Kin have proposed hardware L0 caches.**
- **Andras’ FlexCache does software way-prediction to software controlled array of tag registers.**

Acknowledgements

- **Mark Hampton — GNU assembler, simulator.**
- **Ronny Krashinsky — Energy modeling.**
- **Sam Larsen — SUIF compiler.**
- **C. Scott Ananian — Java compiler (FLEX)**
- **DARPA, NSF, Infineon**