Data Relocation and Prefetching for Programs with Large Data Sets

Yoji Yamada, John Gyllenhaal, Grant Haab, Wen-mei Hwu

University of Illinois at Urbana-Champaign

Background

Cache

- Can be used to decrease time to access data
- Limitations
  - size
  - associativity

Characteristics of array-based applications

- Processing large arrays in a loop nest creates large working sets
- Sequential and non-sequential access patterns

Problems

- Capacity misses due to large working set and non-sequential accesses
- Conflict misses due to non-sequential accesses
- Cache line fetch overhead for non-sequential accesses

Concept of Data Relocation and Prefetching

To solve the problems due to large working set

- prefetch necessary data for innermost loop iteration space in cache just before entering the innermost loop

```
for (i = 0; i < N; i++) {
   for (j = 0; J < 3; j++) {
      ... = A[i][2*j] + B[j][i];
   }
}
```

prefetch {A[0][0], A[0][4], A[0][6], B[0][0], B[1][0], B[2][0]}

prefetch {A[1][0], A[1][4], A[1][6], B[0][1], B[1][1], B[2][1]} during computation for i = 0

prefetch {A[2][0], A[2][4], A[2][6], B[0][2], B[1][2], B[2][2]} during computation for i = 1

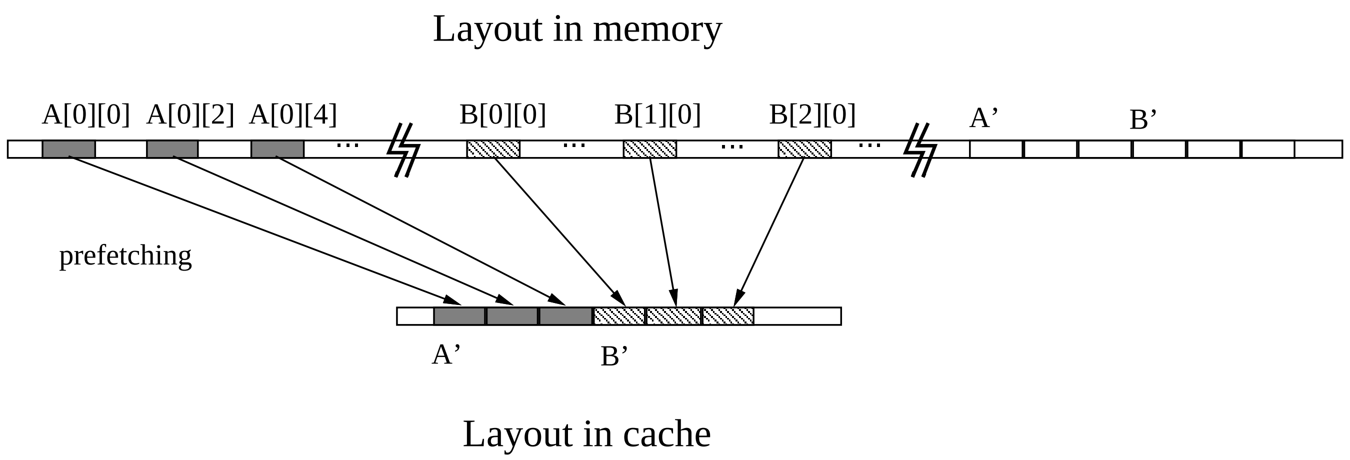
...

i = 0	**prefetching**	**computation**		
i = 1		**prefetching**	**compuation**	
i = 2			**prefetching**	**computation**

Concept of Data Relocation and Prefetching

To solve the problems due to non-sequential array accesses

- relocate array elements
  - compresses non-sequential accesses to sequential accesses
  - aligns arrays to non-overlapping locations
- perform the relocation during prefetching



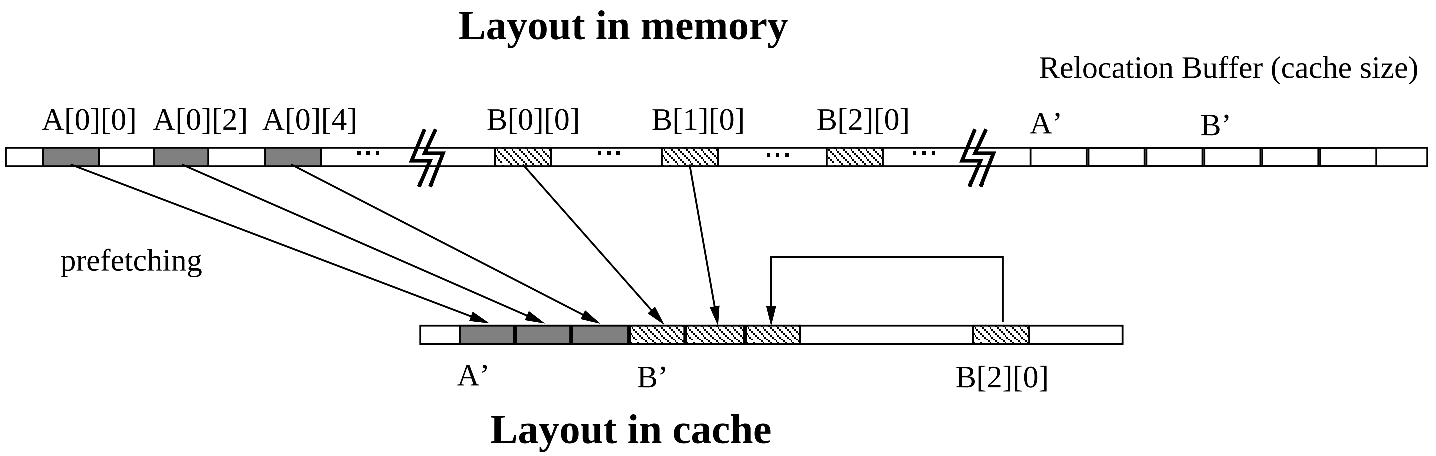
Architectural Support: Precollect Operation

a) Original Code

```
for (i = 0; i < N; i++)
   for (j = 0; j < 3; j++)
      ... = A[i][2*j] + B[j][i];
```

b)Transformed Code

```
for (i = 0; i < N; i++) {
   precollect(&A[0][0],A’,8,16,3);
   precollect(&B[0][0],B’,8,24,3);
   for (j = 0; j < 3; j++)
      ... = A’[j] + B’[j];
}
```



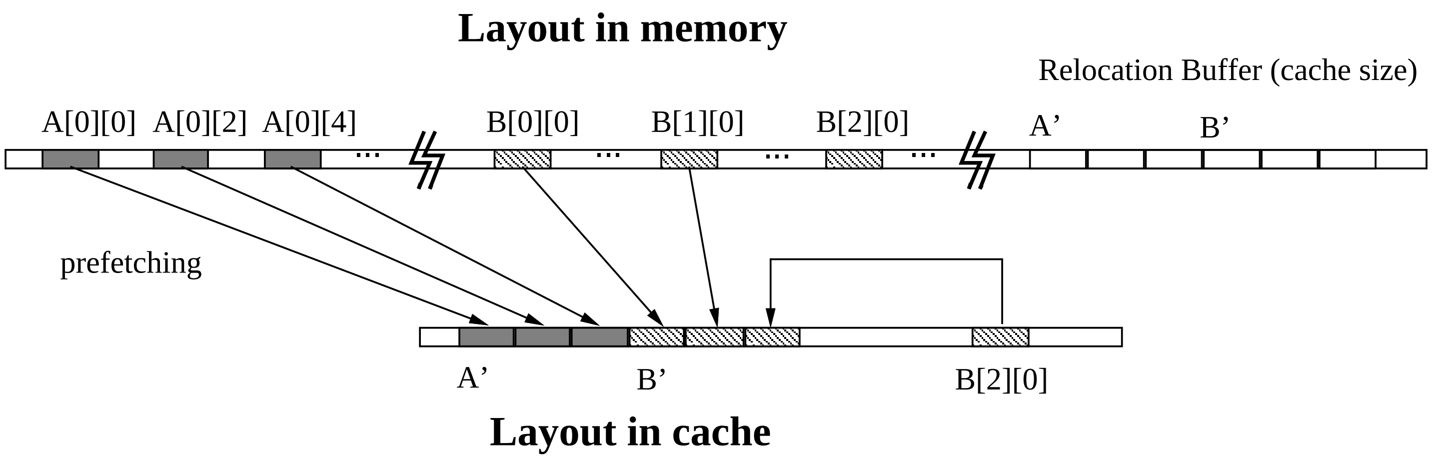
Architectural Support: Await Operation

a) Original Code

```
for (i = 0; i < N; i++)
   for (j = 0; j < 3; j++)
      ... = A[i][2*j] + B[j][i];
```

b)Transformed Code

```
for (i = 0; i < N; i++) {
   precollect(&A[0][0],A’,8,16,3);
   precollect(&B[0][0],B’,8,24,3);
   await(A’);
   await(B’);
   for (j = 0; j < 3; j++)
      ... = A’[j] + B’[j];
}
```



Compiler Support: Example

A) Original Loop Nest

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    C[i][j] = A[i][2*j] + B[j][i];
```

B) After Loop Unrolling

```
for (i'=0; i'<N; i'+=2) {
  /* first outer loop body */
  i = i';
  for (j=0; j<N; j++)
    C[i][j] = A[i][2*j] + B[j][i];

  /* second outer loop body */
  i++;
  if (i < N) {
    for (j=0; j<N; j++)
      C[i][j] = A[i][2*j] + B[j][i];
  }
}
```

C) After Relocation Operation Insertion

```
/* Bd = dimension(B[]) * 8 bytes/element */
precollect(&A[0][0], A', 8, 16, N);
precollect(&B[0][0], B', 8, Bd, N);
preallocate(C', 8, N);

for (i'=0; i'<N; i'+=2) {
  i = i';
  if (i+1 < N) {
     precollect(&A[i+1][0], A'', 8, 16, N);
     precollect(&B[0][i+1], B'', 8, Bd, N);
     preallocate(C'', 8, N);
  }
  await(A'); await(B'); await(C');
  for (j=0; j<N; j++)
    C'[j] = A'[j] + B'[j];
  distribute(&C[i][0], C', 8, 8, N);

  i++;
  if (i < N) {
    if (i+1 < N) {
      precollect(&A[i+1][0], A', 8, 16, N);
      precollect(&B[0][i+1], B', 8, Bd, N);
      preallocate(C', 8, N);
    }
    await(A''); await(B''); await(C'');
    for (j=0; j<N; j++)
      C''[j] = A''[j] + B''[j];
    distribute(&C[i][0], C'', 8, 8, N);
  }
}
finishup();
```

Data Relocation and Prefetching Support

- Architecture Support: Five new assembly-level operations
  - *Precollect*
  - *Preallocate*
  - *Distribute*
  - *Await*
  - *Finishup*
- Compiler Support (implemented in the IMPACT Compiler):
  - insertion of operations
  - supporting loop transformations
- Hardware Support: *Data relocation and prefetch unit*
  - processes all DRP operations
  - pipelined, non-blocking implementation

Architecture Support: Data relocation and prefetch operations

Precollect operation

- Operation
  - Prefetch array elements from their original locations
  - Load them into the cache using relocated addresses
  - Set dirty bit so written back to memory if replaced in cache
  - Does not block the cache or stall the processor
- Operands
  - Starting address of the original array
  - Starting address of the relocated array
  - Size of an array element
  - Stride of accesses to the original array
  - Number of elements of the array to be relocated

Architecture Support: Data relocation and prefetch operations

Await operation

- Operation
  - Check if associated *Precollect* or *Preallocate* operation has finished
  - If not, block the cache until associated operation is finished
- Operands
  - Starting address of relocated array (for operation association)

Replacement of High-level DRP Operations with Low-level Instructions

a) Precollect Operation

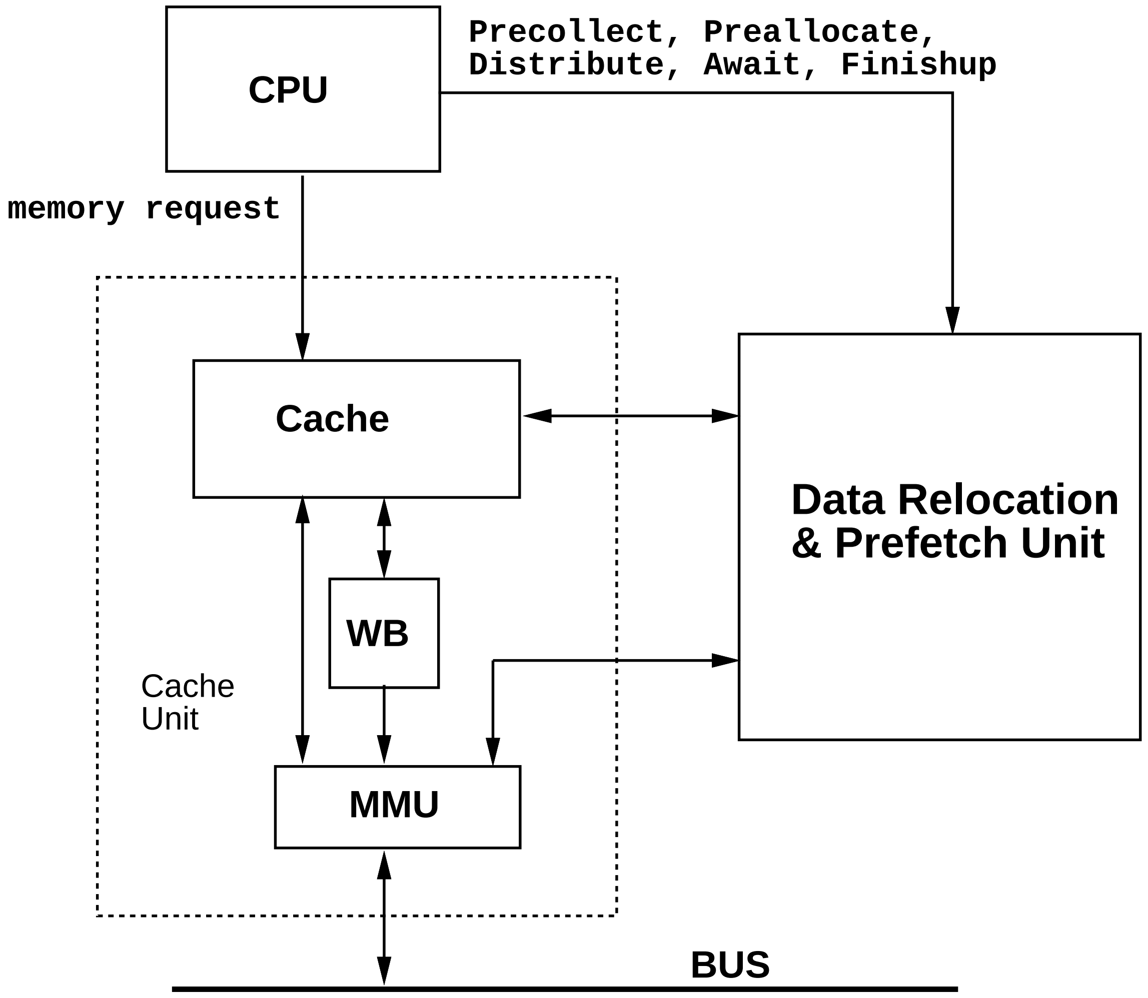
```
Lop_setup [Rt] [(element_size)(stride)]

Lop_precollect [buffer_addr] [(array_addr)(Rt)(# of data)]
```

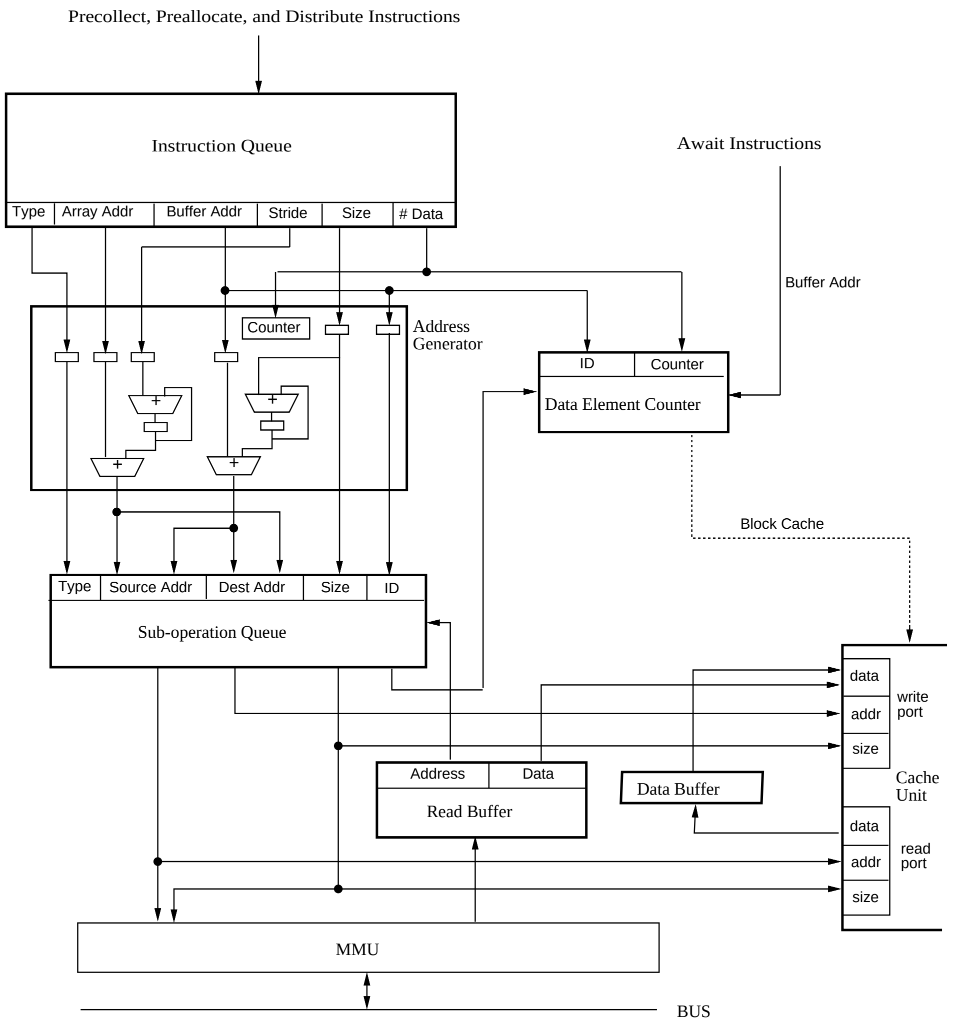
b) Await Operation

```
Lop_await [buffer_addr]
```

Hardware Support: Data Relocation and Prefetch Unit Interfaces



Data Relocation and Prefetch Unit Data Path



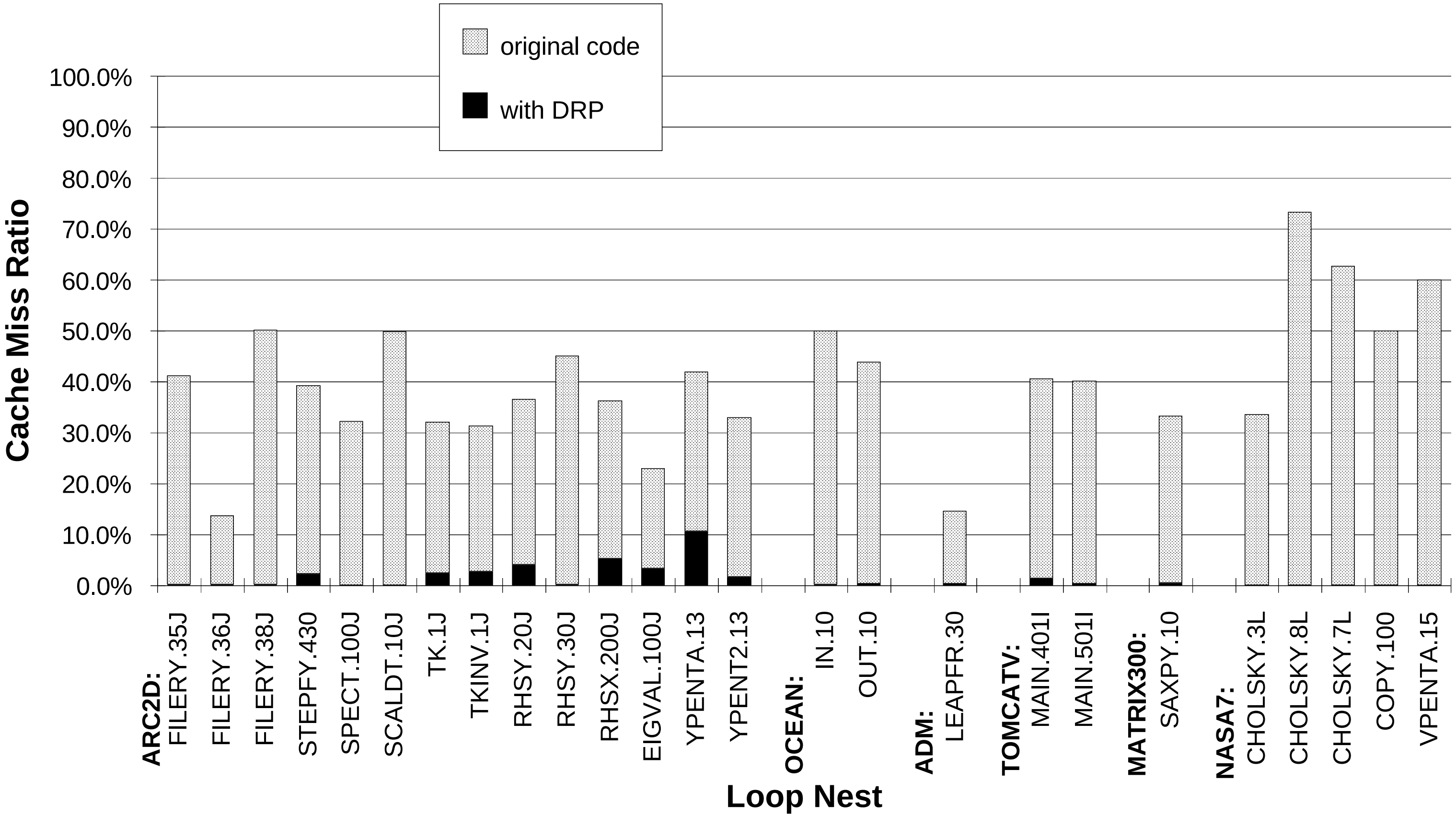
Experimental Evaluation

- Benchmark Programs
  - PERFECT: ARC2D, OCEAN, ADM
  - SPEC: TOMCATV, NASA7, MATRIX300
- Emulation to Verify Correctness of Transformations
  - HP-PA 7100 Architecture
- Detailed Cycle-by-Cycle Simulation
  - Detailed processor and memory models
  - Uniform sampling at 1% rate to reduce simulation time

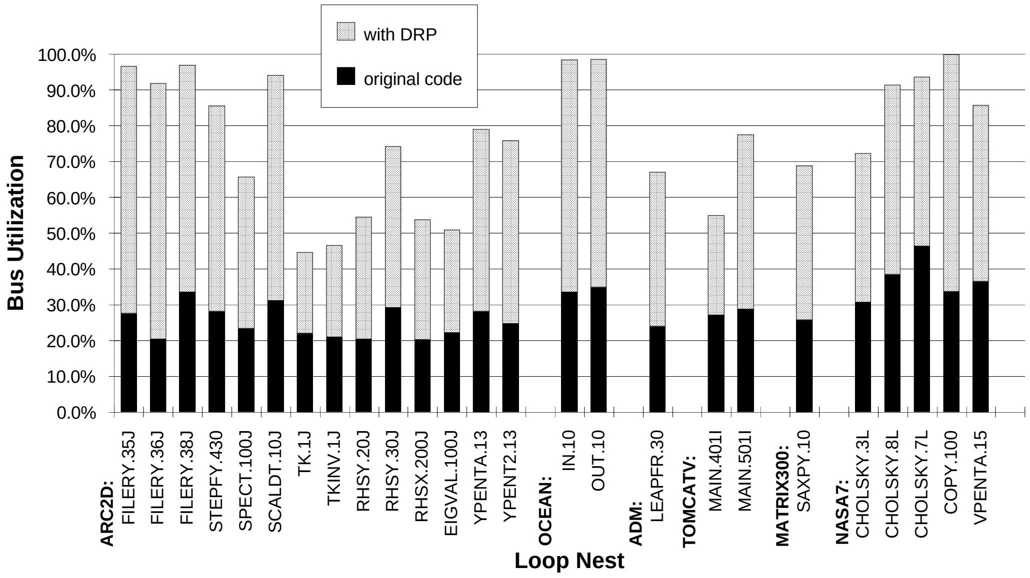
Simulation Model

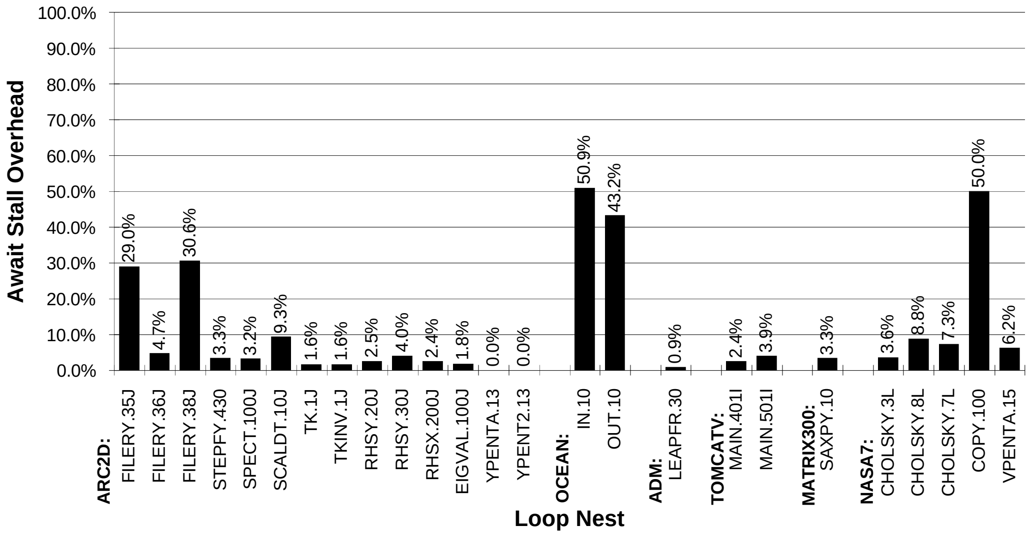
- Processor Model
  - four-issue superscalar with pipelined functional units
  - functional units can execute any type of instruction
  - scoreboarding with register renaming
  - 64 integer and 64 DP floating-point registers
  - instruction latencies are same as HP-PA 7100
- Memory System Model
  - separate instruction and data caches
  - each 8k-byte, blocking caches with 16-byte block
  - data cache is 4-way multiported, write-back, no write-alloc.
  - 8-entry combining write buffer
  - split-transaction memory bus, 64 bits/cycle data bandwidth
  - pipelined memory with a 10-cycle latency

DRP Cache Miss Results

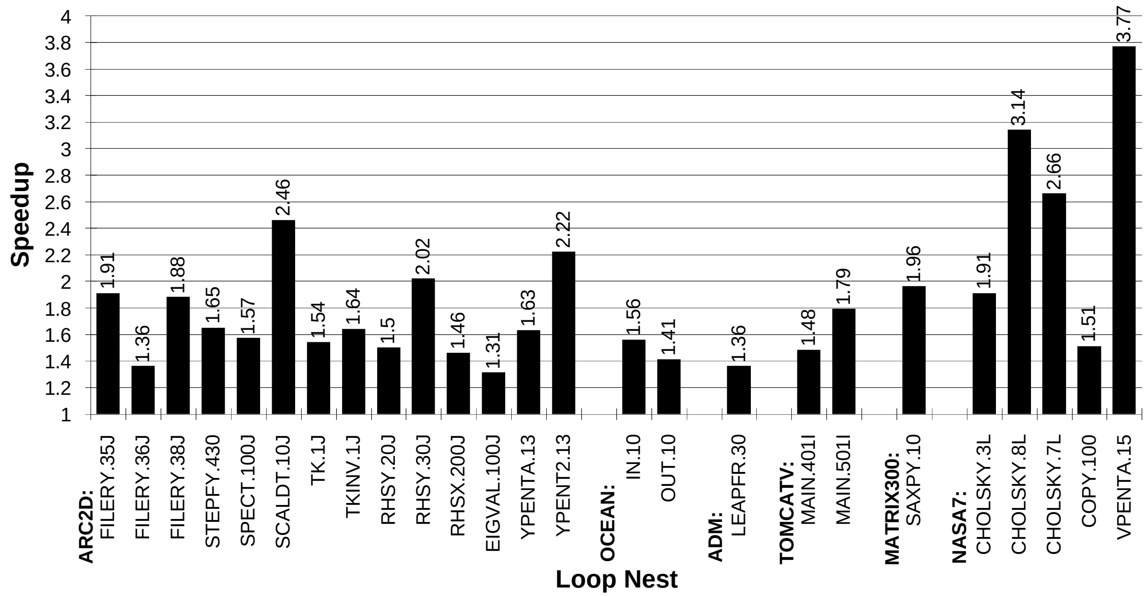


DRP Bus Utilization and Await Stall Overhead Results





DRP Speedup Results



Benchmark	Transformed Loop Nest Execution %	Execution Time in Billions of Cycles: Base	Execution Time in Billions of Cycles: DRP	Execution Speedup
ARC2D	31.9%	9.248	8.300	1.11
OCEAN	19.8%	11.165	10.512	1.06
ADM	2.2%	1.732	1.703	1.02
TOMCATV	40.1%	1.442	1.065	1.35
MATRIX300	96.3%	3.181	1.586	2.01
NASA7	15.5%	13.010	11.788	1.10

Data Relocation and Prefetching Future Work

- Tune compiler algorithm to reduce overhead and increase applicability
- Reduce DRP bus utilization by combining requests for sequential accesses into single requests
- Investigate hardware techniques to eliminate the await instruction
- Evaluate DRP performance for other cache protocols