



Software Controlled Memory Bandwidth

- *Deepak N. Agarwal*

AMD

- *Wanli Liu*

University of Maryland

- *Dr. Donald Yeung*

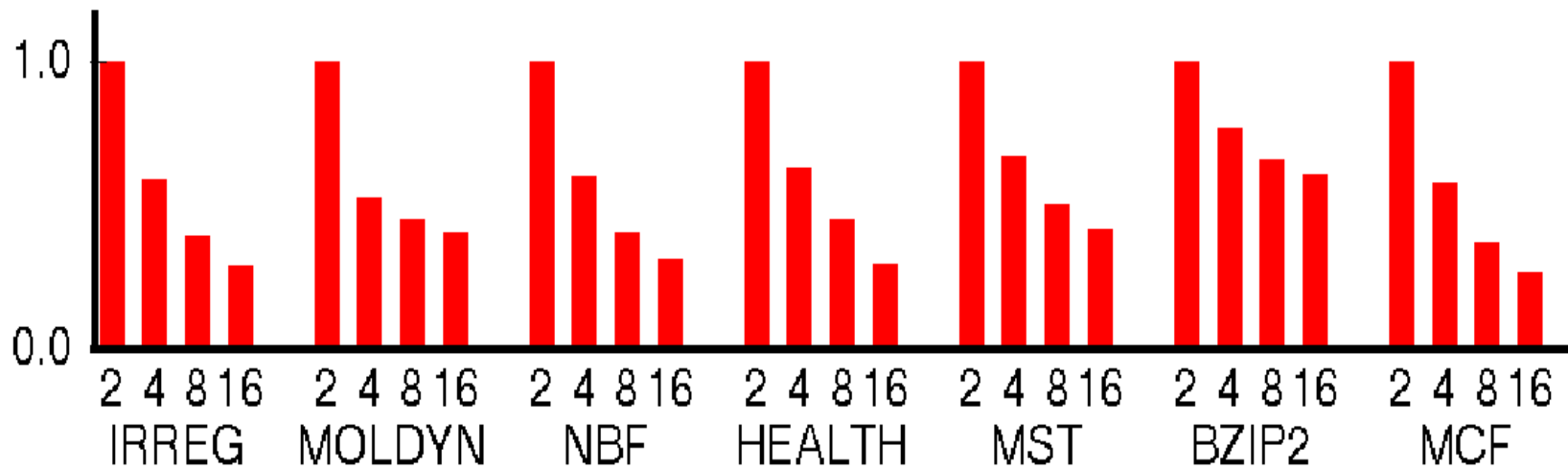
University of Maryland



Factors Stressing Memory Bandwidth

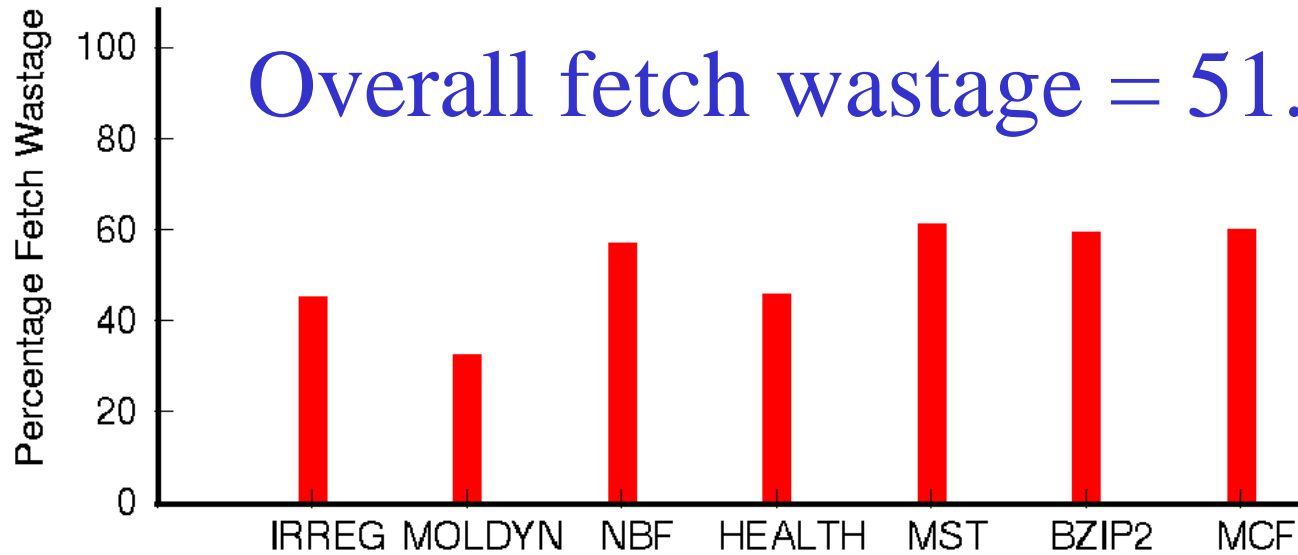
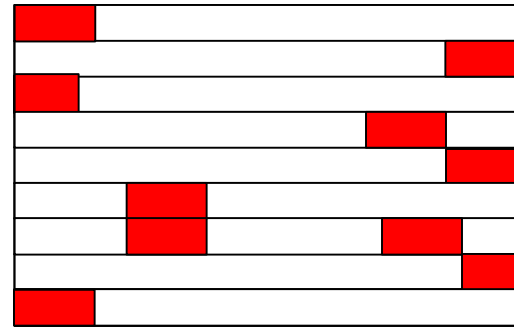
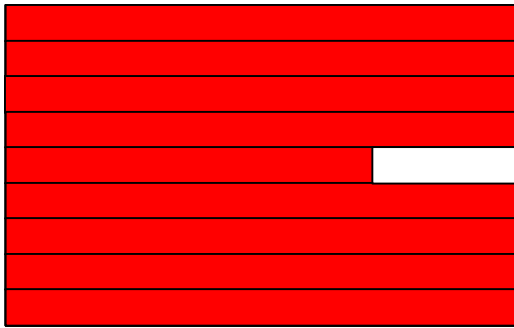
- Processor Improvement
 - Clock Speed Increase
 - More ILP
- Latency Tolerance Techniques Used
 - Non-Blocking Caches, Prefetching, Multi-Threading, etc
- Pin Limitation and Packaging Considerations

Bandwidth Impacts Performance

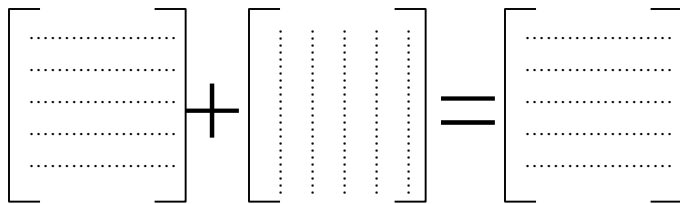


From 2Gb/s to 4Gb/s performance improves
by 38%

Opportunity

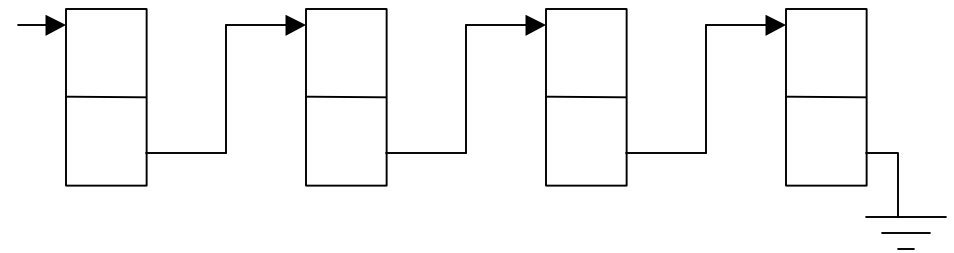


Dense/Sparse Applications



Matrix Addition

```
for(j=0;j<X;j++){  
  for(i=0;i<X;i++){  
    C[j][i]= A[j][i] + B[j][i];  
  }  
}
```



Linked List

```
While(ptr){  
  sum+=ptr->data;  
  ptr=ptr->next;  
}
```



Hardware vs. Software Techniques

Spatial Footprint Predictor (S.Kumar, ISCA'98)

- Hardware Technique
- Selectively Prefetches Required Data Elements

Contribution

- Complexity effective Software Centric Approach
- Sparse Memory Accesses Detected at Source Code Level

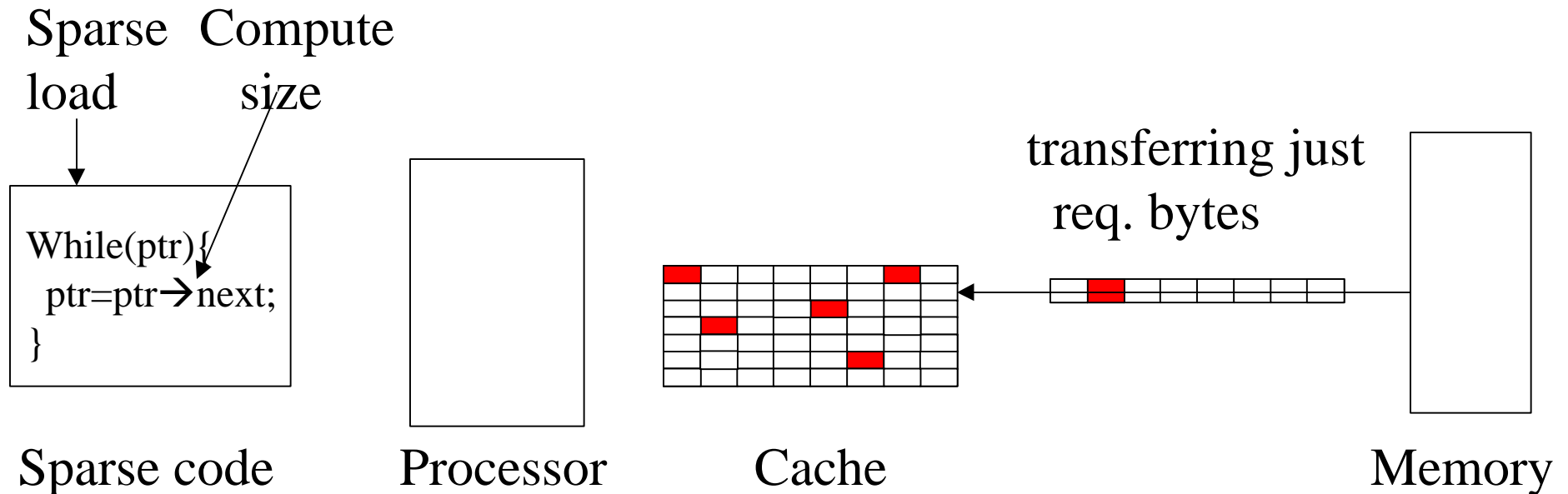


Roadmap

- Motivation
- Our Technique
- Experimental Results
- Conclusion

Approach

- Identify Sparse Memory Accesses
- Compute Transfer Size
- Annotate Selected Memory Instructions





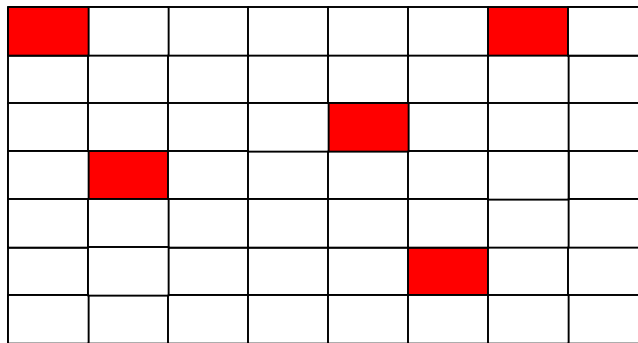
Sparse Memory Access Patterns

- Affine Array Accesses
- Indexed Array Accesses
- Pointer Chasing Accesses



Affine Array Accesses

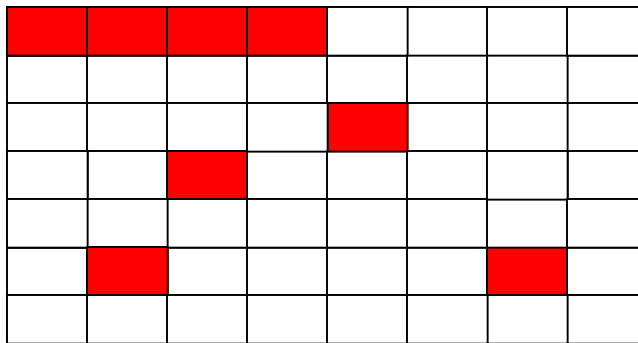
```
for(i=0;i<X;i+=N){  
    sum+= A[i];  
}
```





Indexed Array Accesses

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



Computing Transfer Size

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

Load2

Load1

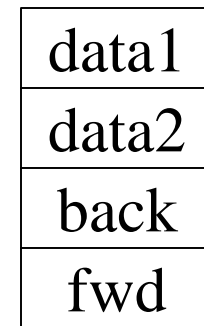
Size #1 – Normal Load

Size #2 – sizeof(A[i])(Sparse Load)

```
While(ptr→fwd){  
    sum+= ptr→data1;  
    ptr  = ptr→fwd;  
}
```

Load #2 Load #1

Structure Layout



Size #1
16 bytes

Size #2
4 bytes



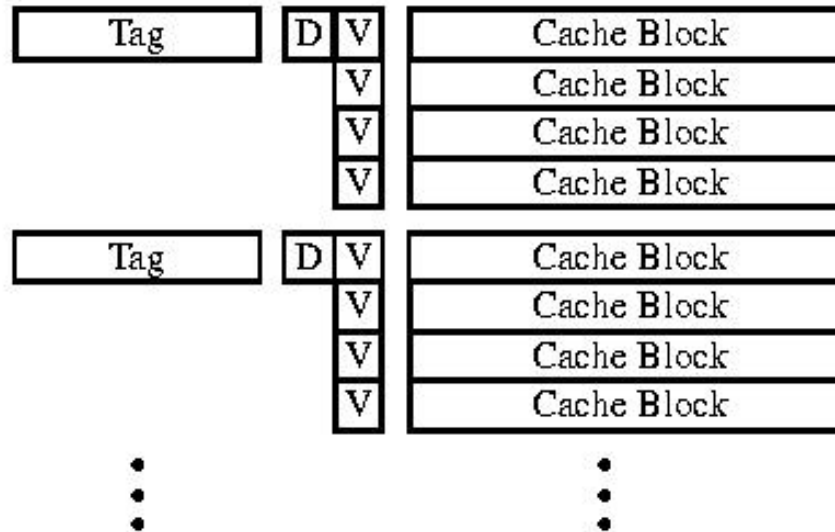
Annotating Memory Instructions

Memory Instructions with Size Information

	load word	load double	store word	store double	prefetch
8 bytes	lw ₈	ld ₈	sw ₈	sd ₈	pref ₈
16 bytes	lw ₁₆	ld ₁₆	sw ₁₆	sd ₁₆	pref ₁₆
32 bytes	lw ₃₂	ld ₃₂	sw ₃₂	sd ₃₂	pref ₃₂



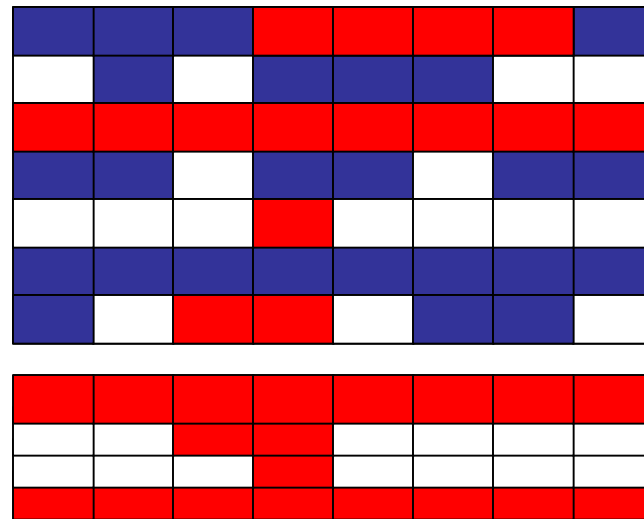
Sectored caches



Fetching Variable Sized Data

·
·
·
·
Ld R0(&R1)
Ld R0(&R2)
Ld8 R0(&R3)
Ld16 R0(&R4)
Ld R0(&R5)
·
·
·

Sector Hit/
Cache Block Miss
Sector Miss



Lower Level Memory



Application Overview

IRREG	Scientific	Indexed Array
MOLDYN	Scientific	Indexed Array
NBF	Scientific	Indexed Array
HEALTH	Olden	Ptr. Chasing
MST	Olden	Ptr. Chasing
BZIP2	SPEC2000	Indexed Array
MCF	SPEC2000	Affine Array, Ptr. Chasing



Experimental Methodology

Cache Simulations

- Traffic and Miss-rate Behavior
- SFP-Ideal (8 Mbytes)
- SFP-Real (32 Kbytes)

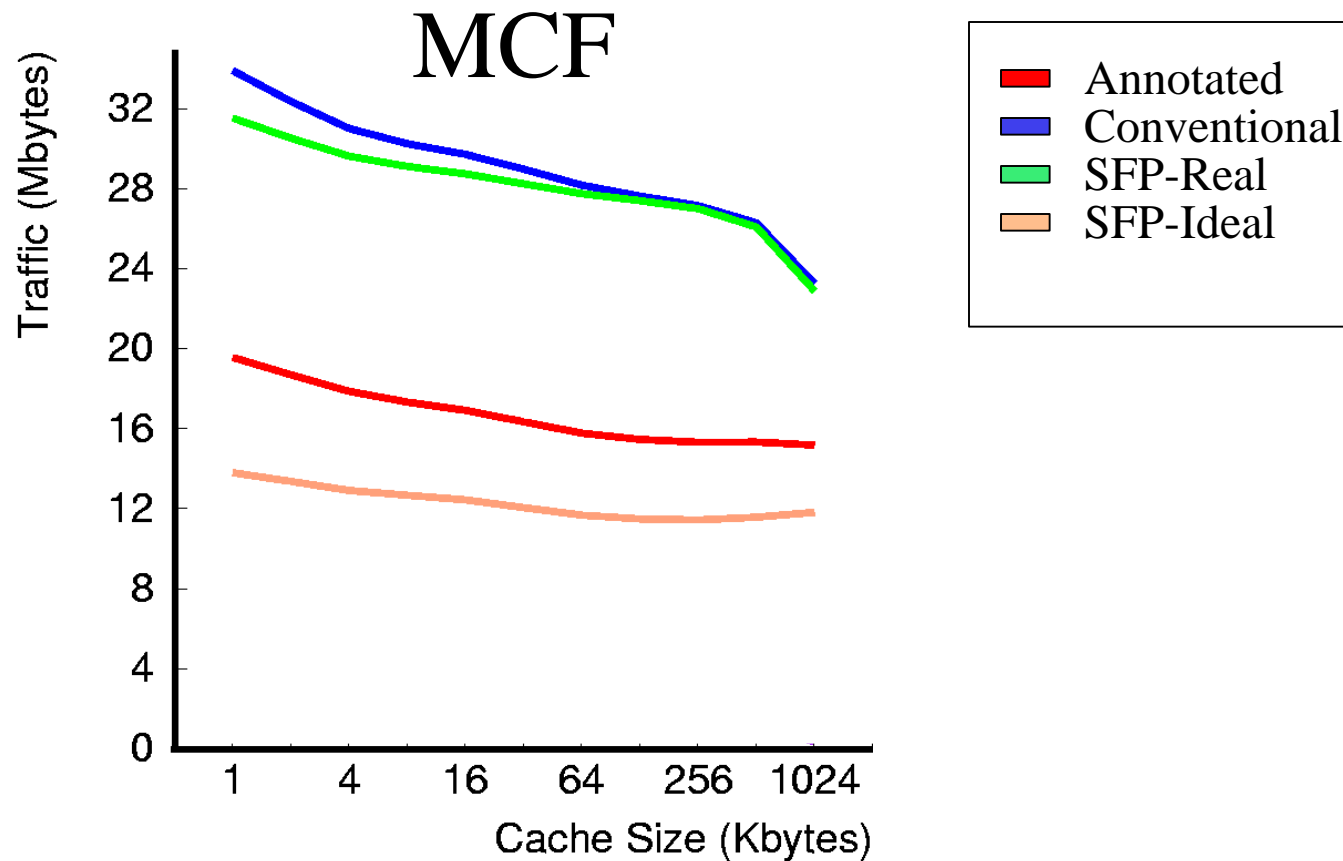
Performance Simulations

- Comparison with Conventional
- Latency Tolerant Study
 - Prefetching
- Bandwidth Sensitivity

Processor and Memory parameters

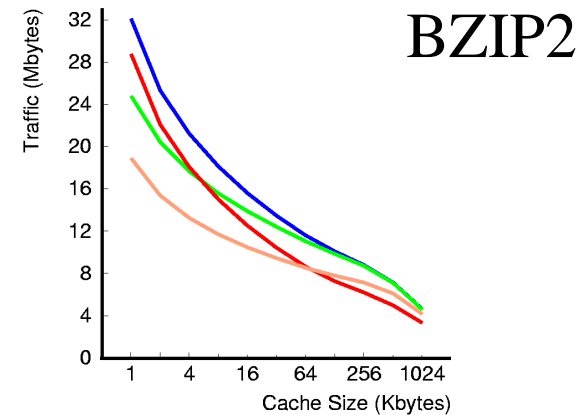
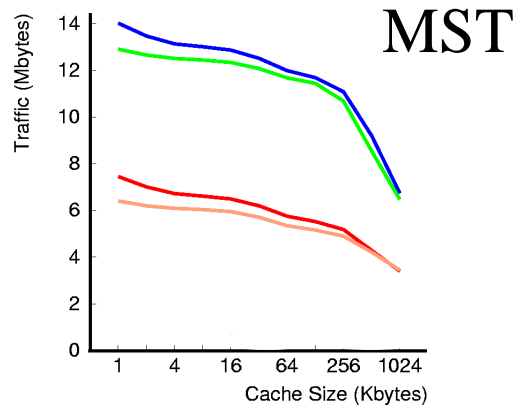
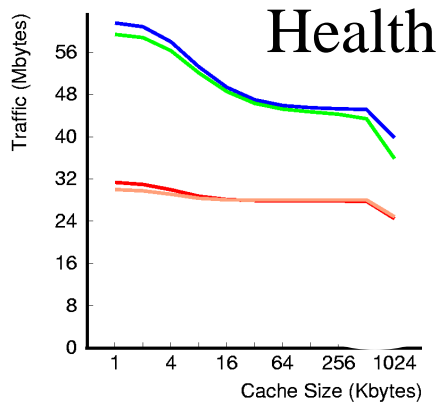
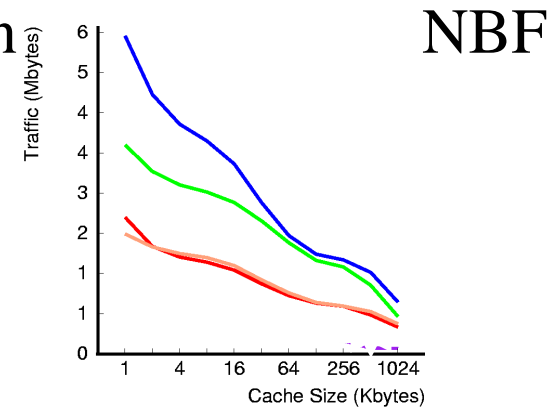
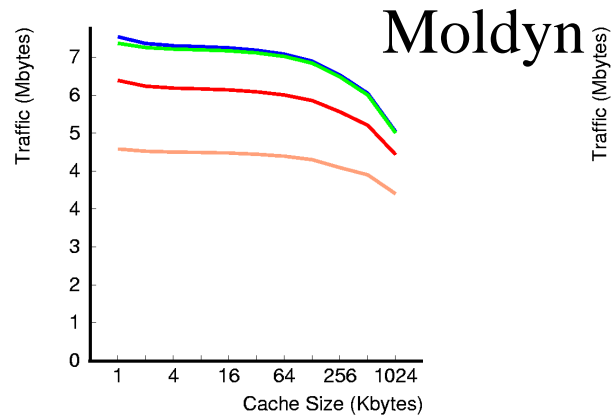
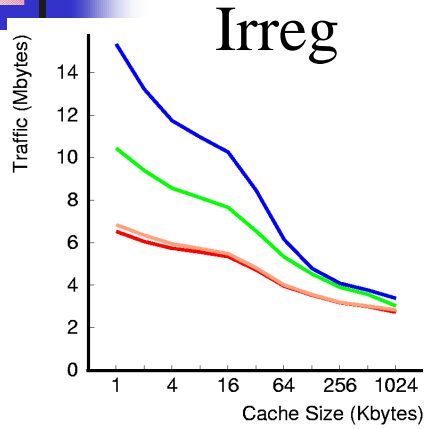
Processor Model	Super scalar
Processor Speed	2 GHz
Issue Width	8
Memory Bandwidth	2 GB/s
Memory Latency	120
Memory Bus Width	8 Bytes
DRAM Banks	64

Traffic Behavior



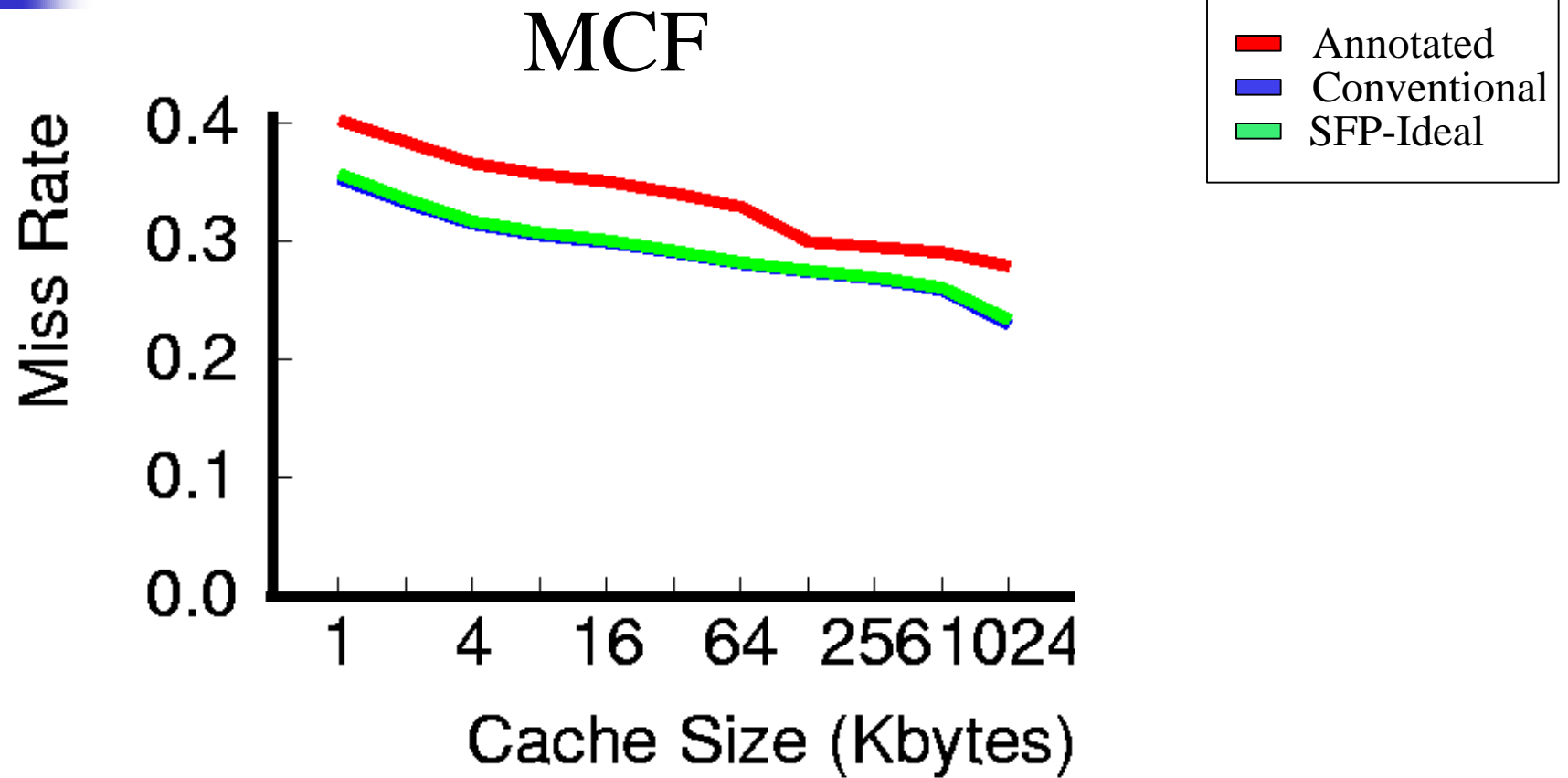
Traffic Reduction for MCF – 57%

Traffic Behavior



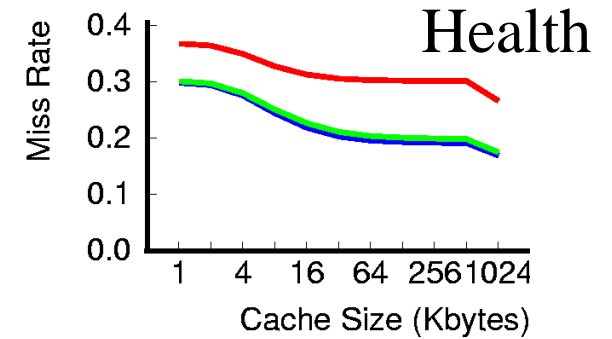
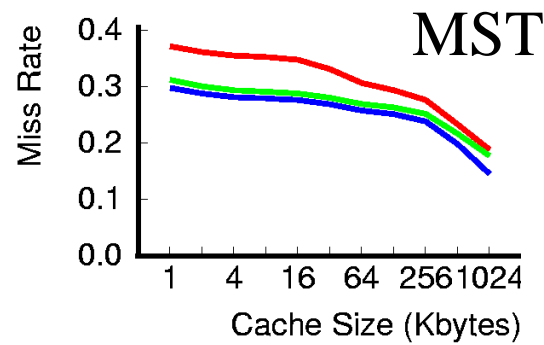
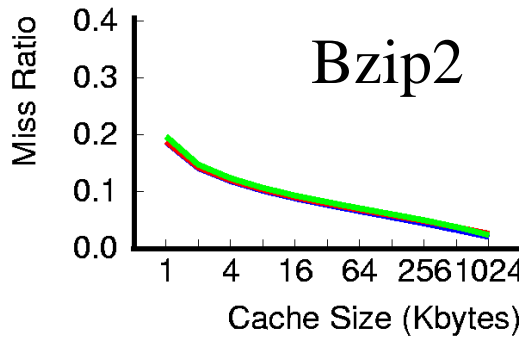
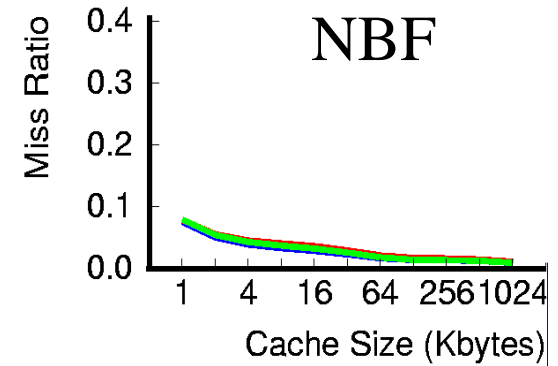
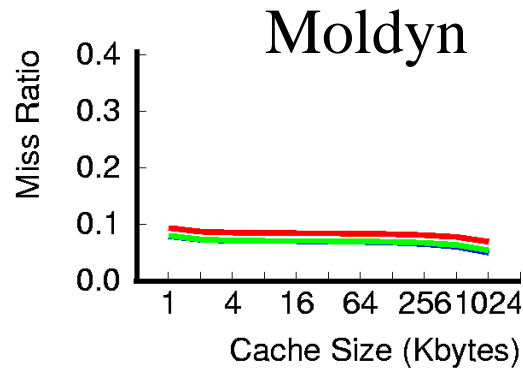
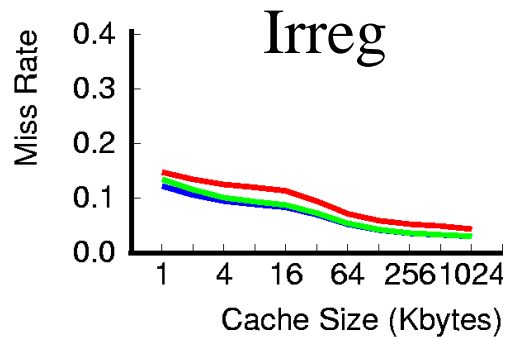
Overall Traffic Reduces by 31 - 71%

Miss-Rates



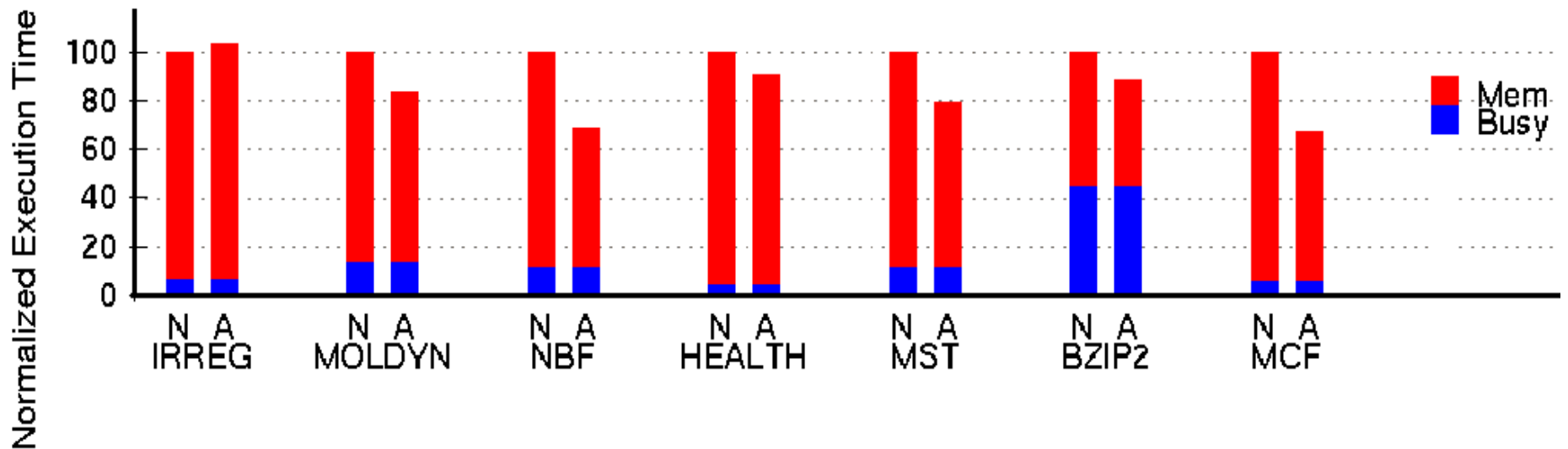
Miss rate increases by 18%

Miss-Rates



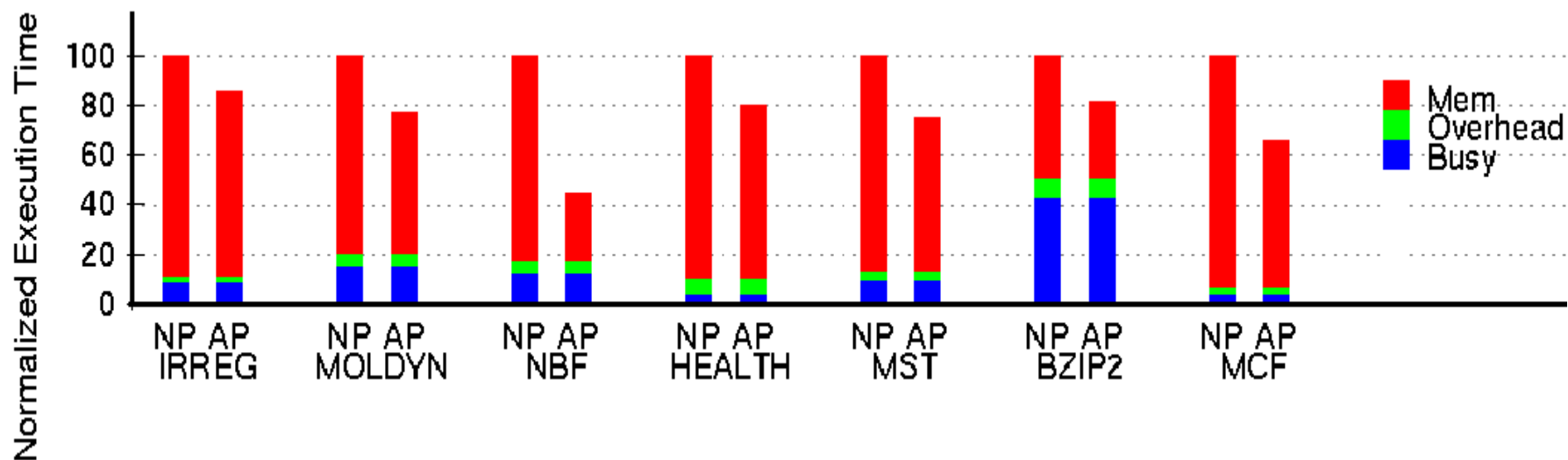
Overall Miss rate increases by 7- 43%

Baseline Performance



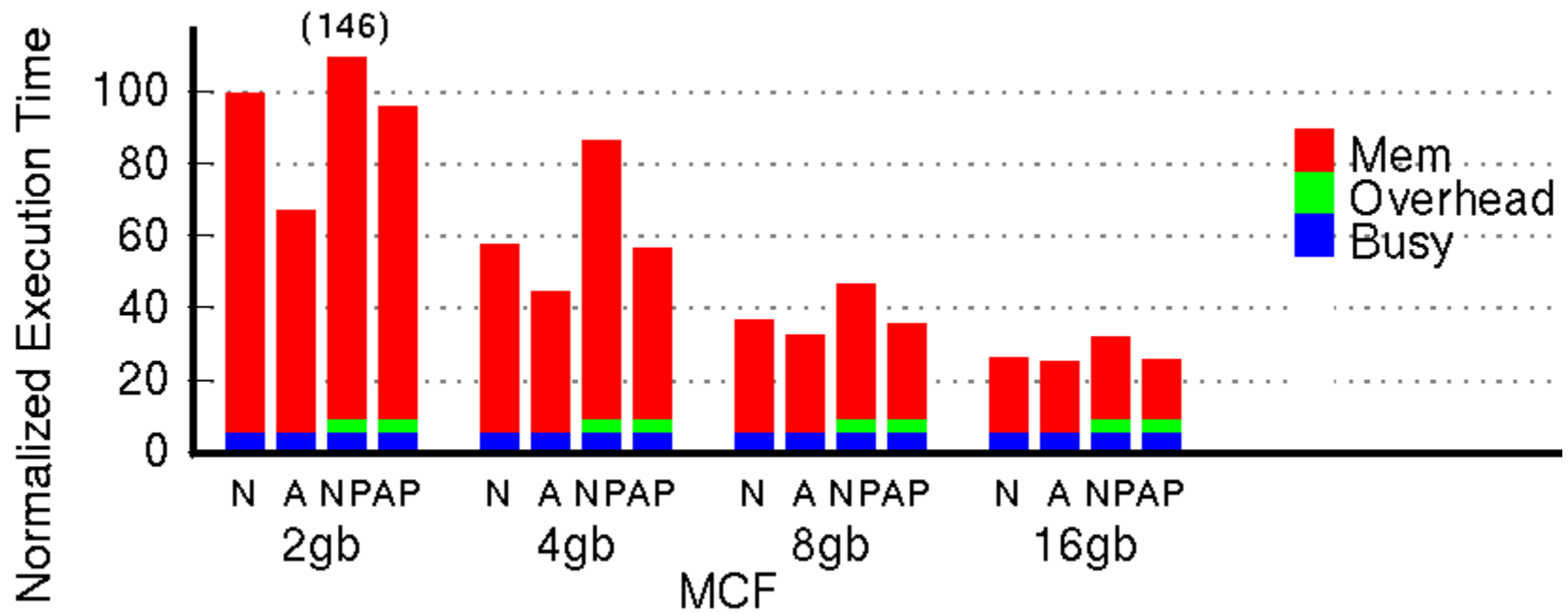
Overall performance improves by 17%

Baseline Performance with Prefetching



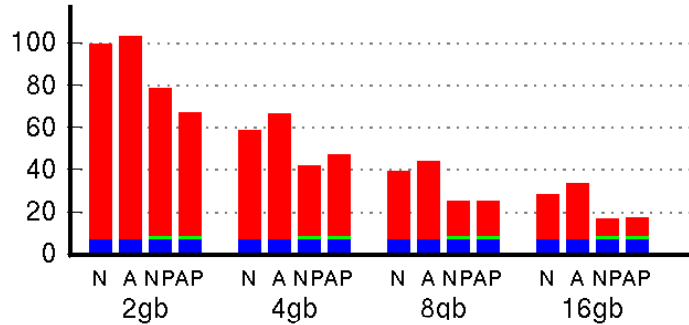
Overall performance improves by 26%

Bandwidth Sensitivity

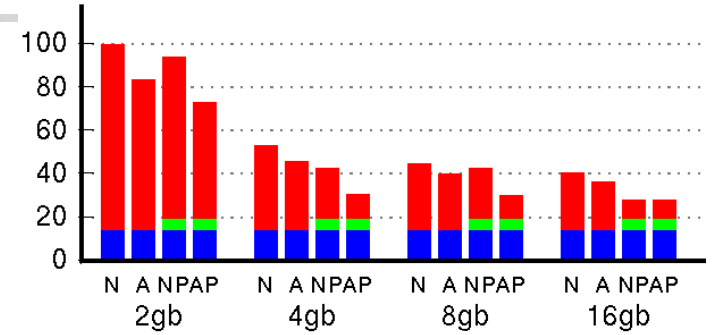


Bandwidth Sensitivity

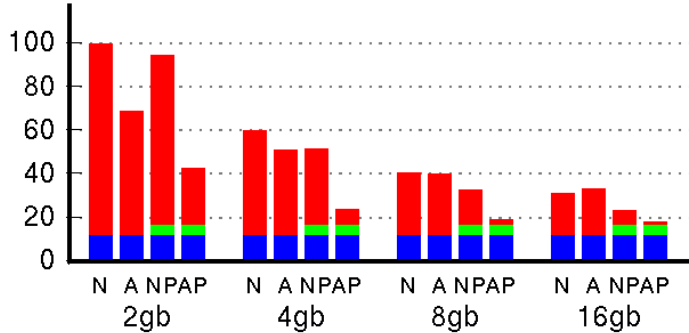
Irreg



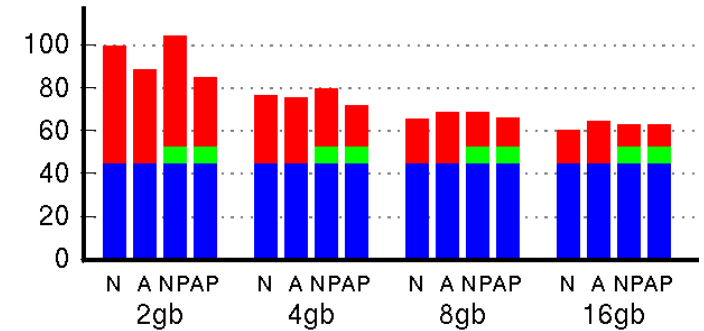
Moldyn



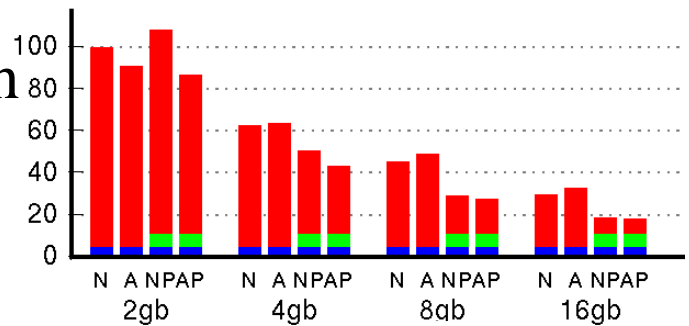
NBF



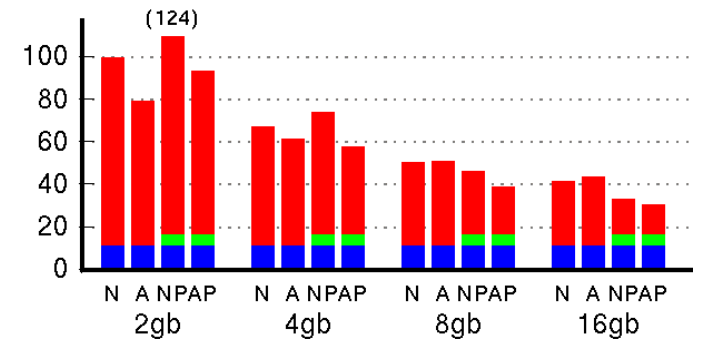
Bzip2



Health



MST





Conclusion

- Complexity effective way for memory bandwidth bottleneck
- Sparse memory references can be identified at source code level
- Software can effectively control memory bandwidth
- Performance numbers:
 - Cache traffic reduces by 31-71%; miss rates increases by 7-43%
 - 17% performance gain over normal caches
 - Annotated s/w prefetching gains 26% over normal prefetching
- Our technique loses effectiveness at higher bandwidth