Software Prefetching for Indirect Memory Accesses

Sam Ainsworth and Timothy M. Jones

Computer Laboratory

What should we software prefetch?

- Stride accesses (A[x+N])?

What should we software prefetch?

- Stride accesses (A[x+N])?

✘ Covered by hardware!

What should we software prefetch?

- Stride accesses (A[x+N])?

✘ Covered by hardware!

- Linked data structures (A->next)?

What should we software prefetch?

- Stride accesses (A[x+N])?

✘ Covered by hardware!

- Linked data structures (A->next)?

✘ No memory-level parallelism!

What should we software prefetch?

- Stride accesses (A[x+N])?

✘ Covered by hardware!

- Linked data structures (A->next)?

✘ No memory-level parallelism!

- Indirect Memory Accesses (A[B[x+N]])?

What should we software prefetch?

- Stride accesses (A[x+N])?

✘ Covered by hardware!

- Linked data structures (A->next)?

✘ No memory-level parallelism!

- Indirect Memory Accesses (A[B[x+N]])?

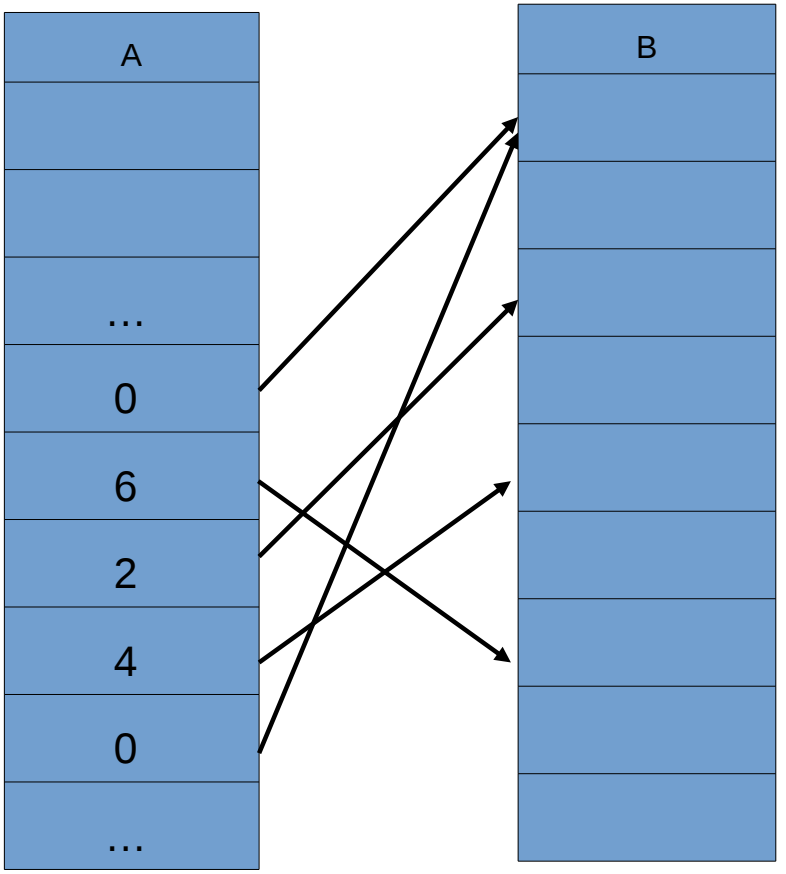
✔ Easy to compute in software, hard to predict in hardware, lots of look-ahead!

Example: Integer Sort (NAS)

```
for (i=0; i<a_size; i++) {
    b[a[i]]++;
}
```

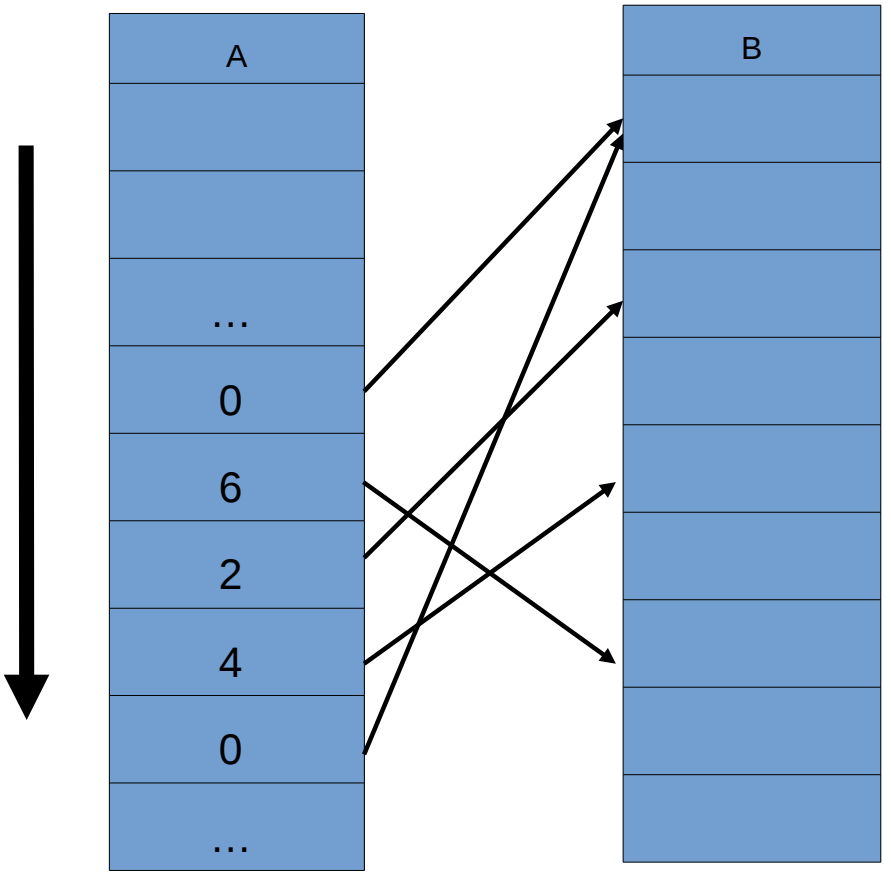
Example: Integer Sort (NAS)

```
for (i=0; i<a_size; i++) {
    b[a[i]]++;
}
```



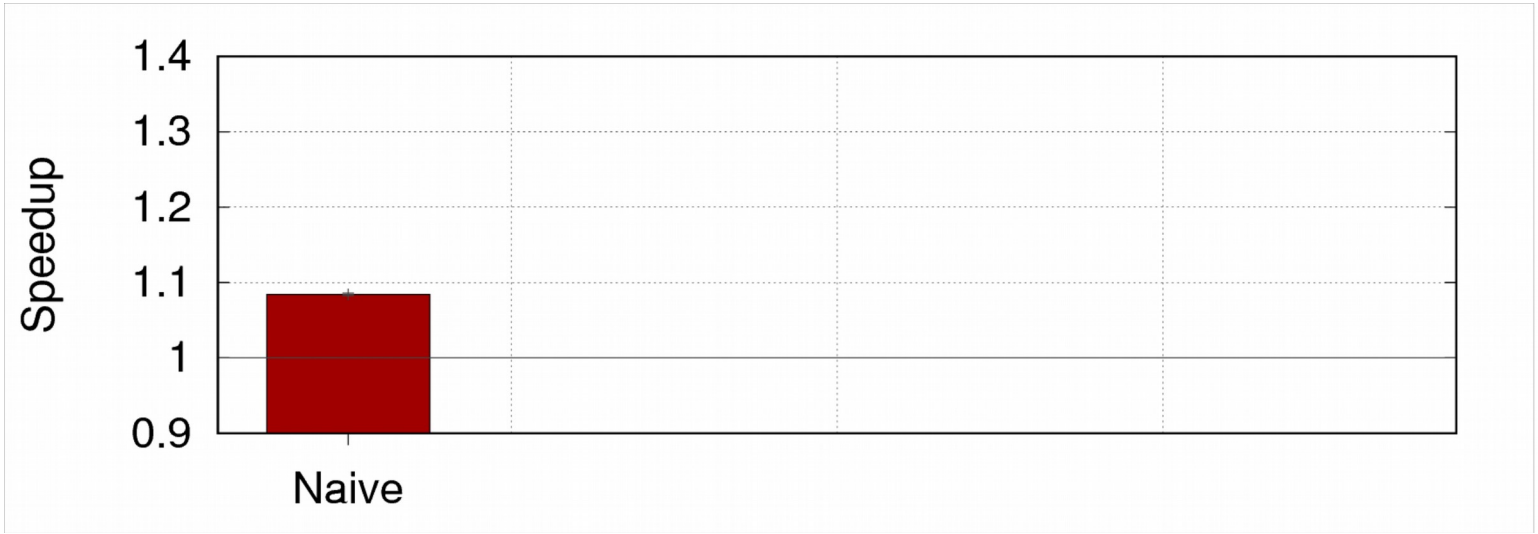
Example: Integer Sort (NAS)

```
for (i=0; i<a_size; i++) {
    b[a[i]]++;
}
```



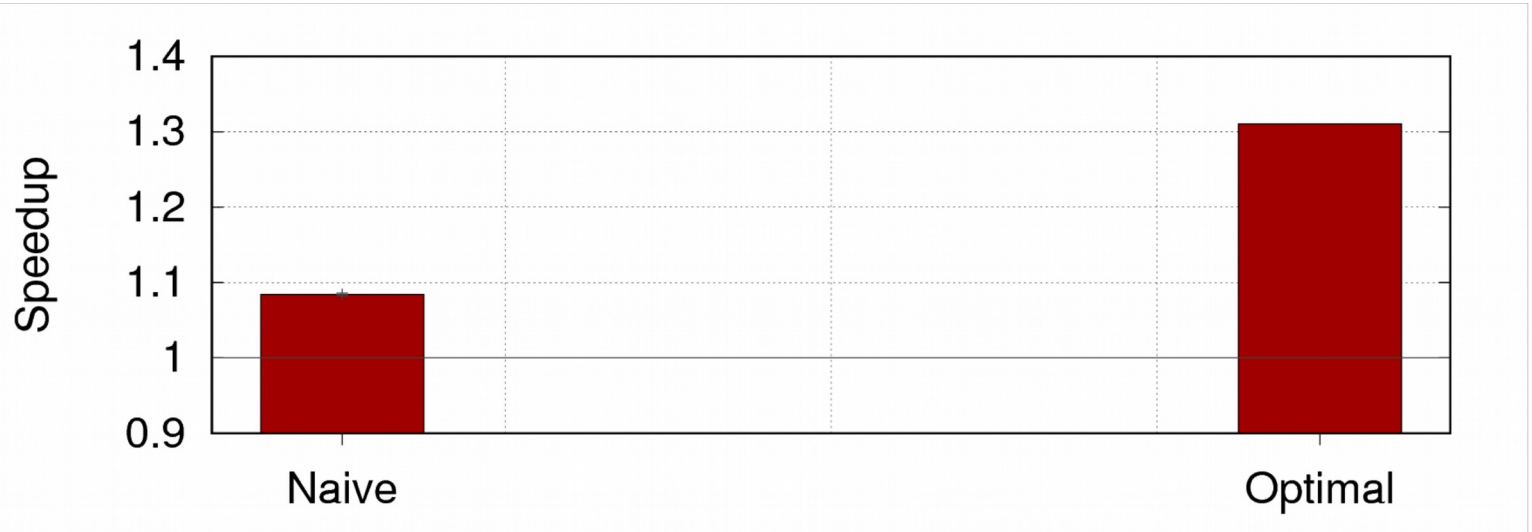
Naive Prefetching

```
for (i=0; i<a_size; i++) {
    SWPF(b[a[i + offset]]);
    b[a[i]]++;
}
```



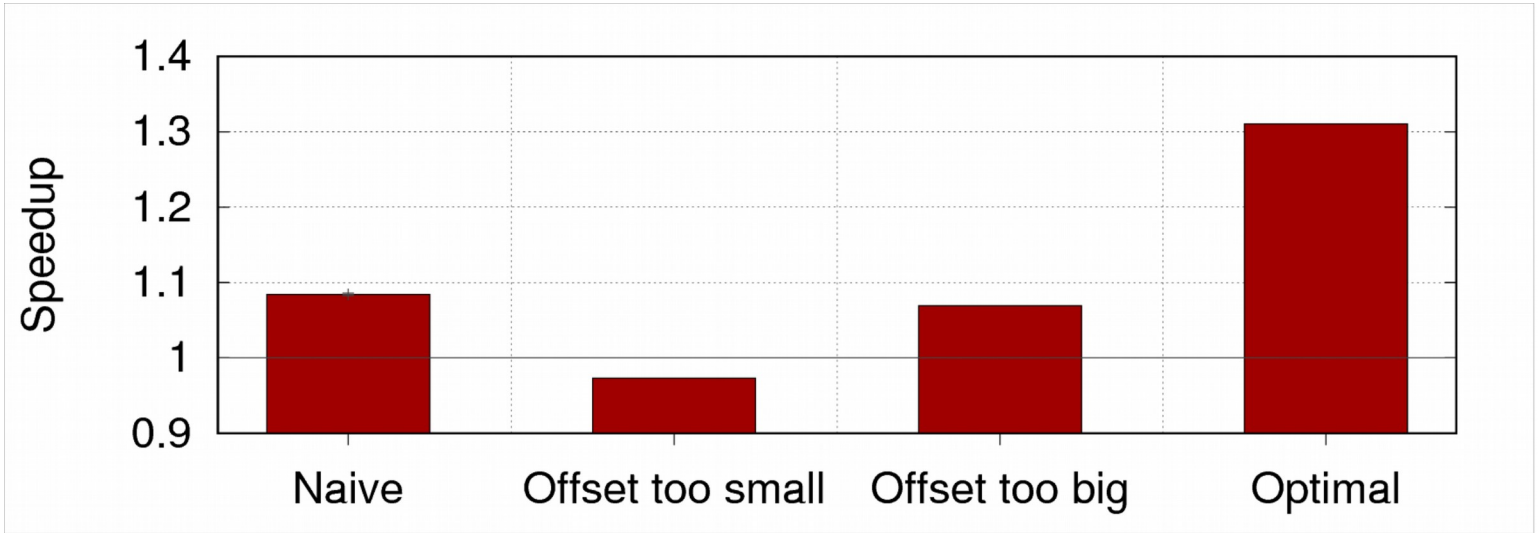
Better Prefetching – Best Offset

```
for (i=0; i<a_size; i++) {
    SWPF(b[a[i + offset]]);
    SWPF(a[i + offset*2]);
    b[a[i]]++;
}
```



Better Prefetching – Bad Offsets

```
for (i=0; i<a_size; i++) {
    SWPF(b[a[i + offset]]);
    SWPF(a[i + offset*2]);
    b[a[i]]++;
}
```



Compiler-Automated Prefetch Insertion Algorithm

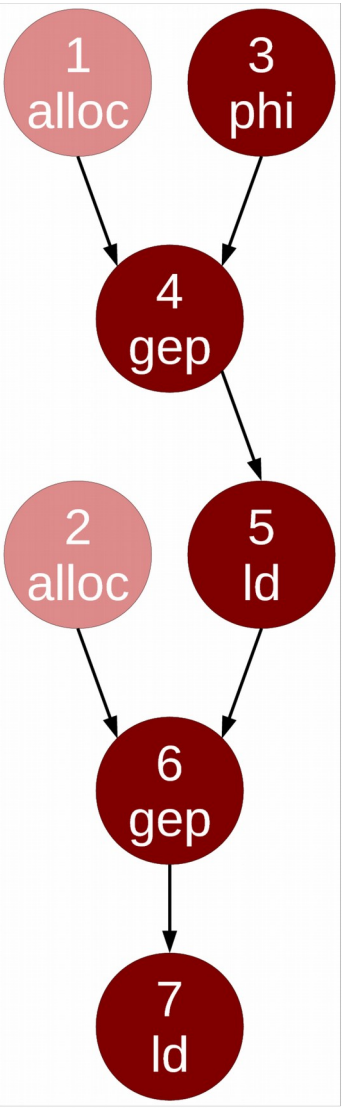
- Identification
- Safety Analysis
- Scheduling

Compiler-Automated Prefetch Insertion Algorithm

- **Identification**
- Safety Analysis
- Scheduling

```
b[a[i]]++
prefetch(b[a[i+?]])
prefetch(a[i+??]])
```

```
start: alloc a, a_size
       alloc b, b_size
loop:  phi i, [#0, i.1]
       gep t1, a, i
       ld t2, t1
       gep t3, b, t2
       ld t4, t3
       add t5, t4, #1
       str t3, t5
       add i.1, i, #1
       cmp size, i.1
       bne loop
```

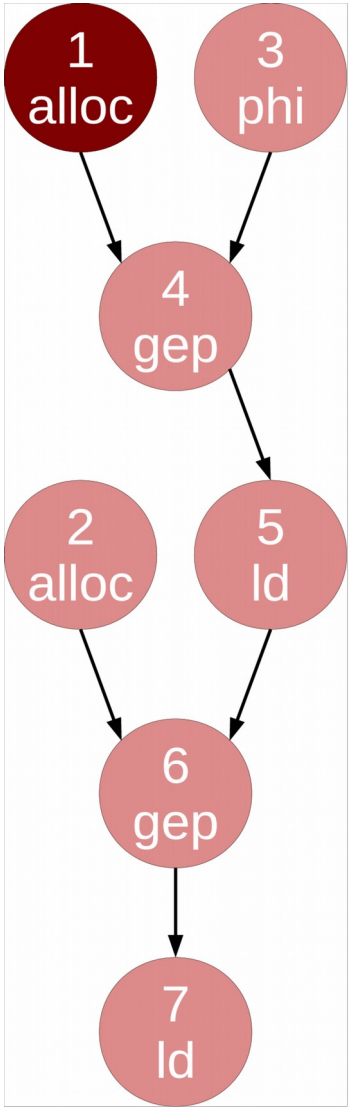


Compiler-Automated Prefetch Insertion Algorithm

- Identification
- **Safety Analysis**
- Scheduling

```
b[a[i]]++
if(i+? < a_size)
  prefetch(b[a[i+?]])
prefetch(a[i+??]])
```

```
start: alloc a, a_size
       alloc b, b_size
loop:  phi i, [#0, i.1]
       gep t1, a, i
       ld t2, t1
       gep t3, b, t2
       ld t4, t3
       add t5, t4, #1
       str t3, t5
       add i.1, i, #1
       cmp size, i.1
       bne loop
```



Compiler-Automated Prefetch Insertion Algorithm

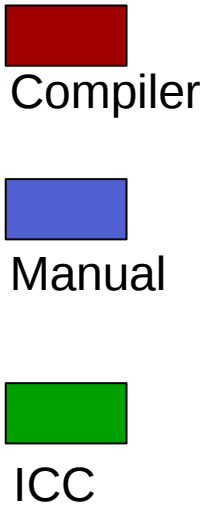
- Identification
- Safety Analysis
- **Scheduling**

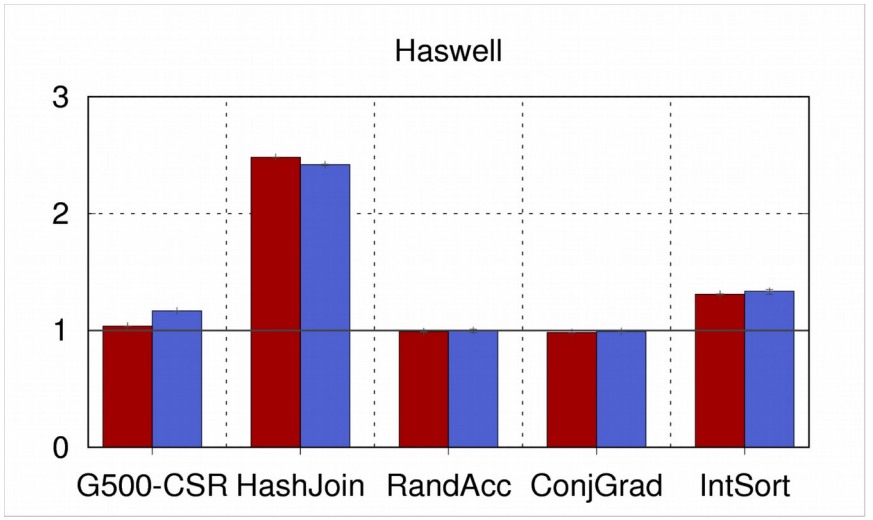
$$\frac{c(t - (l-1))}{t}$$

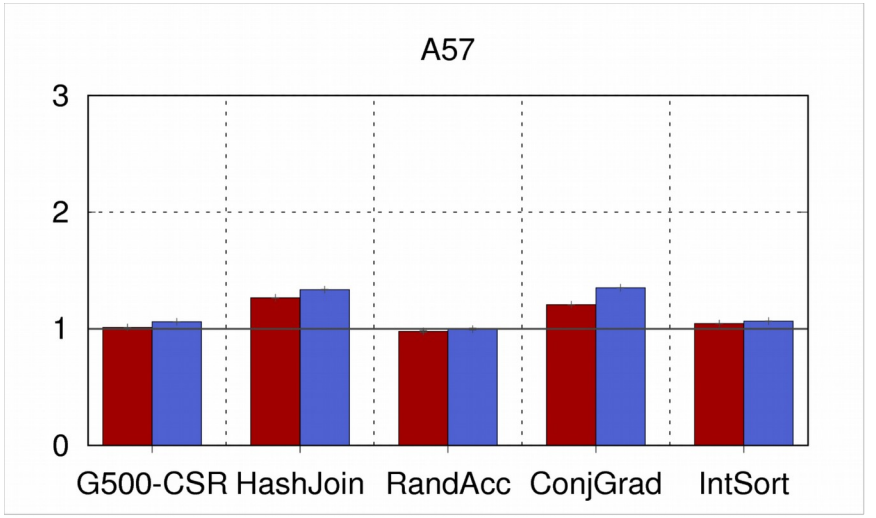
c = microarchitectural constant (64)
t = # loads in sequence
l = # loads in prefetch

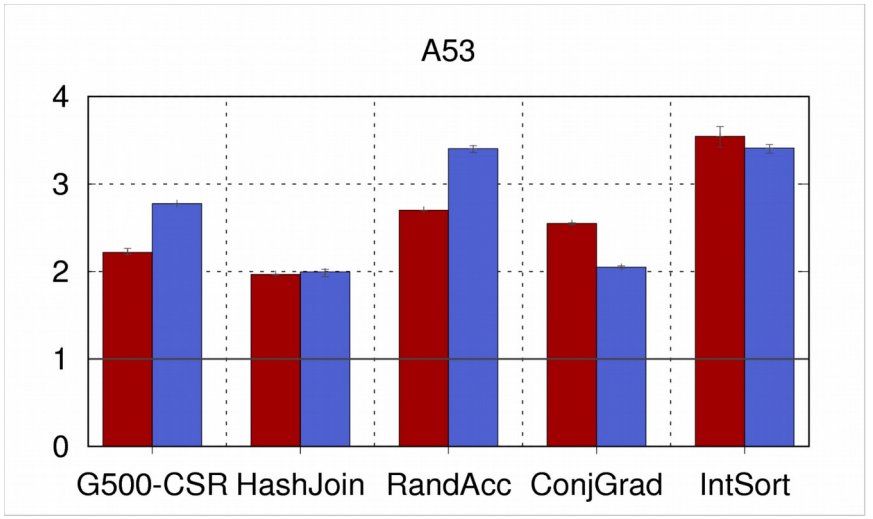
```
b[a[i]]++
if(i+32 < a_size)
  prefetch(b[a[i+32]]) (l=2)
prefetch(a[i+64]])     (l=1)
```

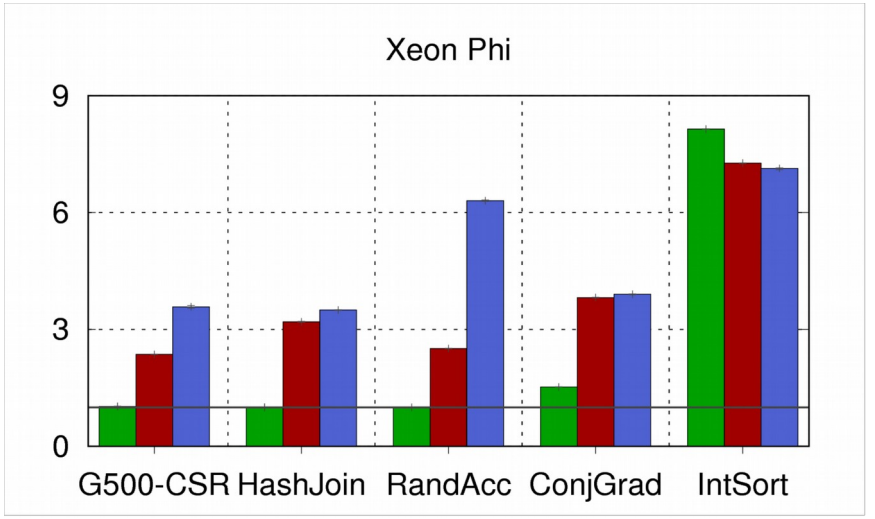
Large Speedups on Real Cores



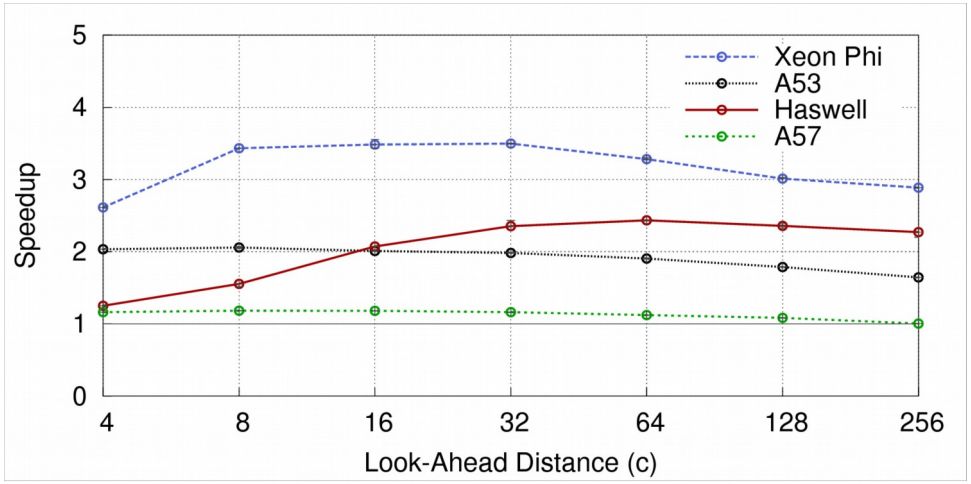




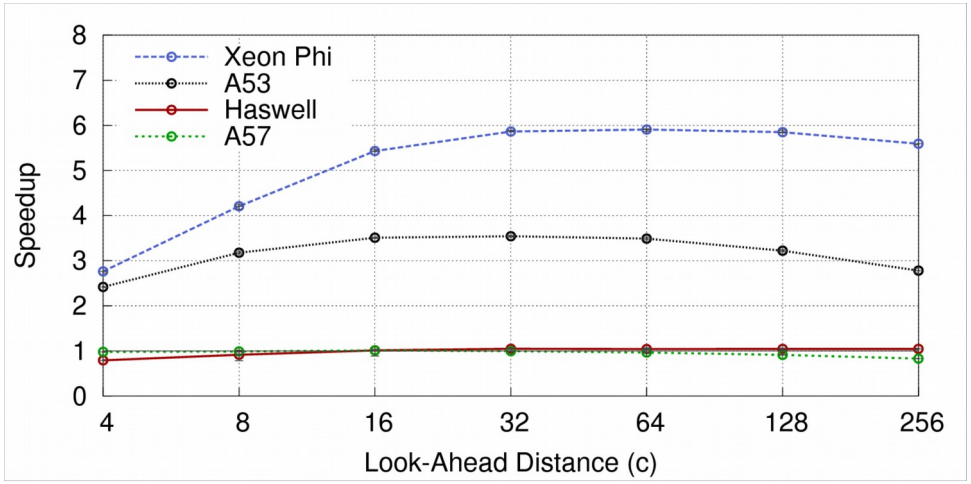




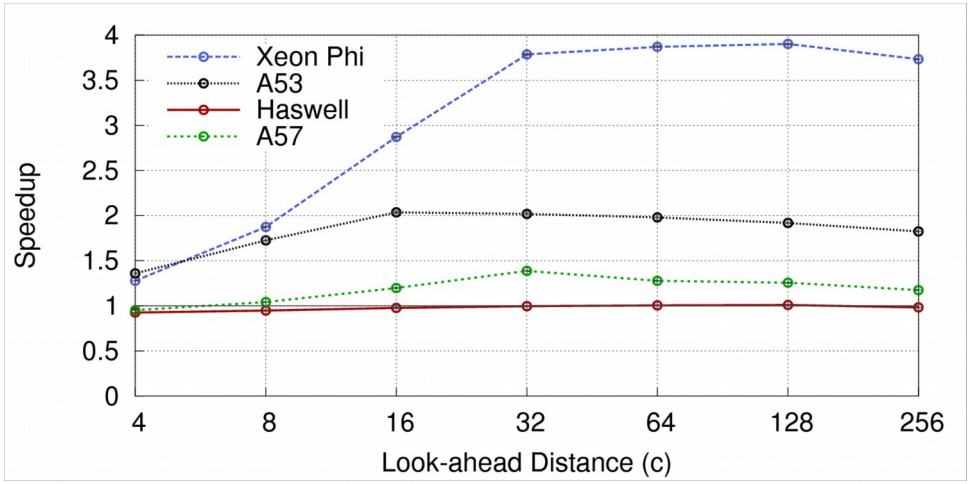
Microarchitectural Constant (c)



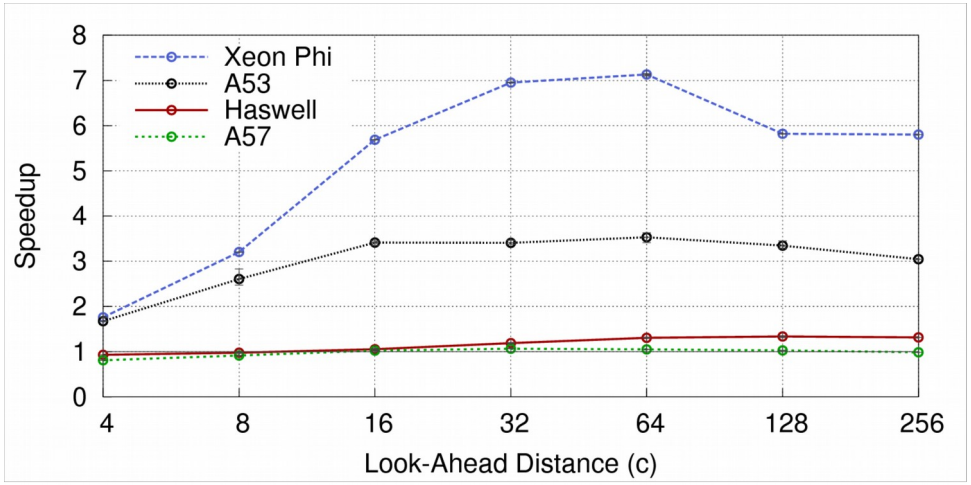
HashJoin



RandAcc



ConjGrad



IntSort