Software Prefetching for Indirect Memory Accesses

Sam Ainsworth and Timothy M. Jones
University of Cambridge
CGO 2017

Presented by
Nishil Talati and Tarunesh Verma
EECS 583
11/18/2019

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?

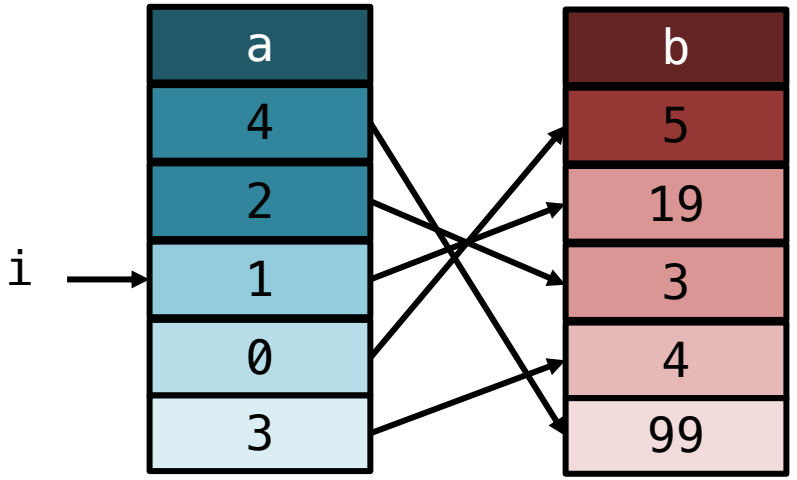
- Stride accesses (A[x+N])?
  - Covered by hardware!
- Linked data structures (A->next)?
  - No memory-level parallelism!
- Indirect Memory Accesses (A[B[x+N]])?

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!

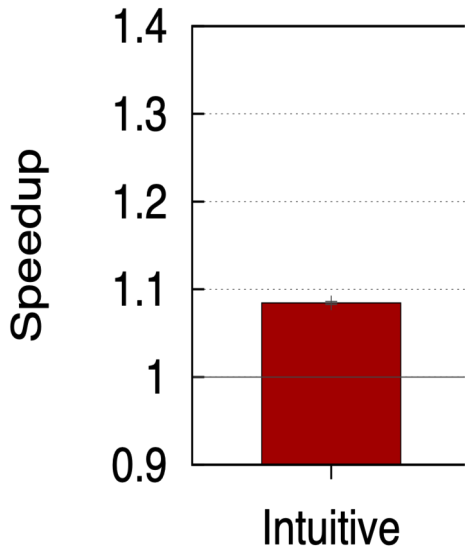
Example: Integer Sort (NAS)

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



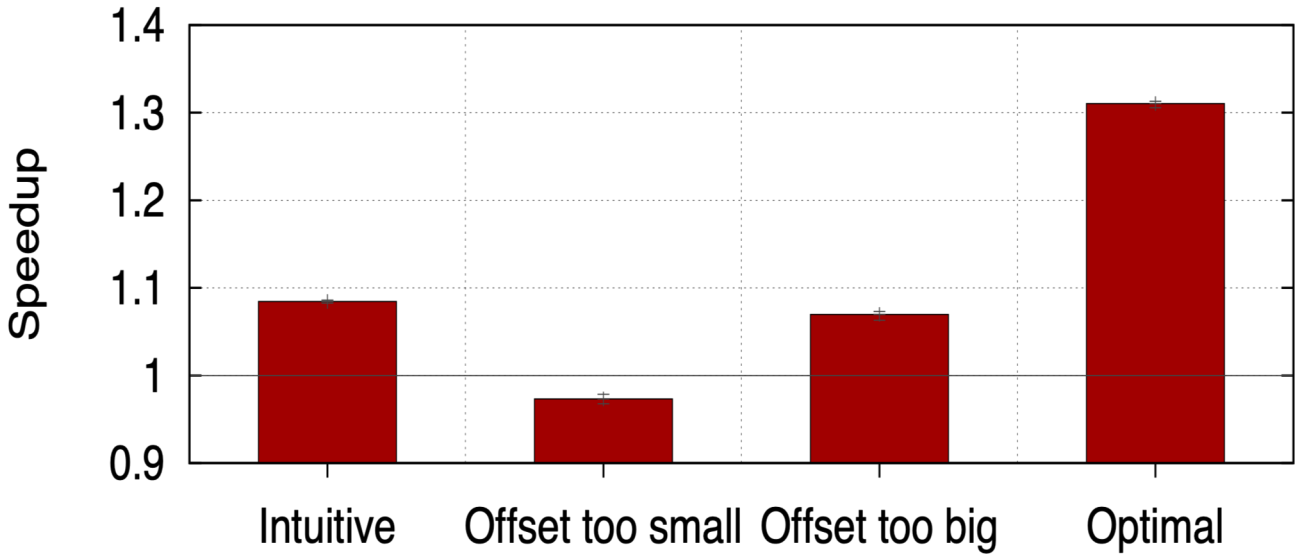
Naive Prefetching

```
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]]++;
}
```

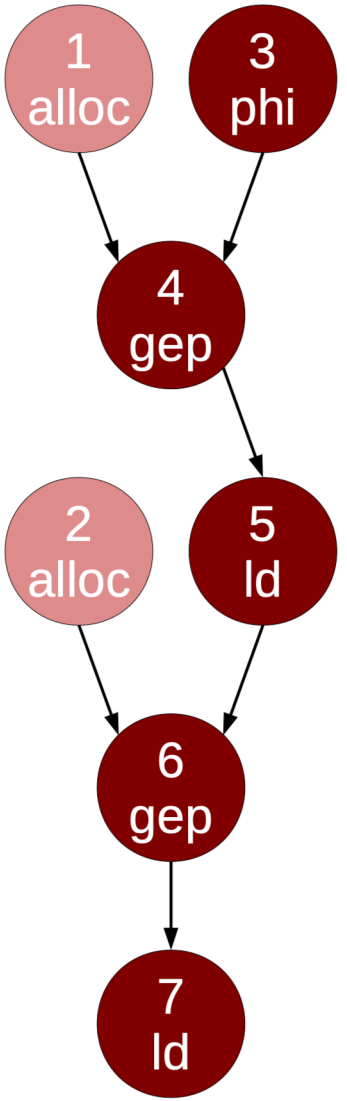


Compiler-Automated Prefetch Insertion Algorithm

- Depth-First Search for Induction Variable
- Safety Analysis
- Scheduling

```
start: alloc a, a_size
       alloc b, b_size
loop:  phi i, [#0, i.1]
       gep t1, a, i
       ld l2, 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
```

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

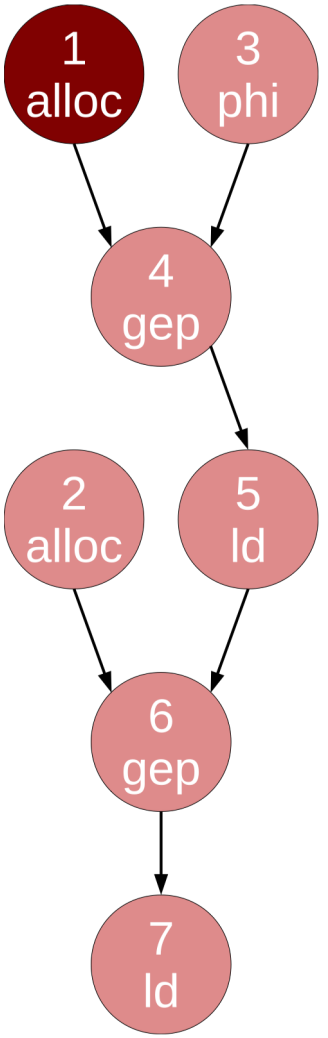


Compiler-Automated Prefetch Insertion Algorithm

- Depth-First Search for Induction Variable
- 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 l2, 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

- Depth-First Search for Induction Variable
- Safety Analysis
- Scheduling

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

c = micro-architecture specific constant (always 64)
t = number of loads in total sequence
l = number of loads in current prefetch

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

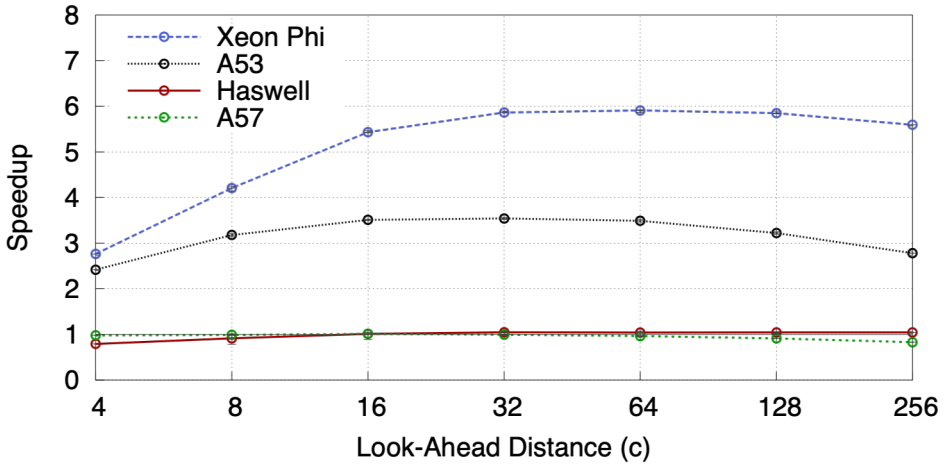
c=64, t=2

Experimental Setup

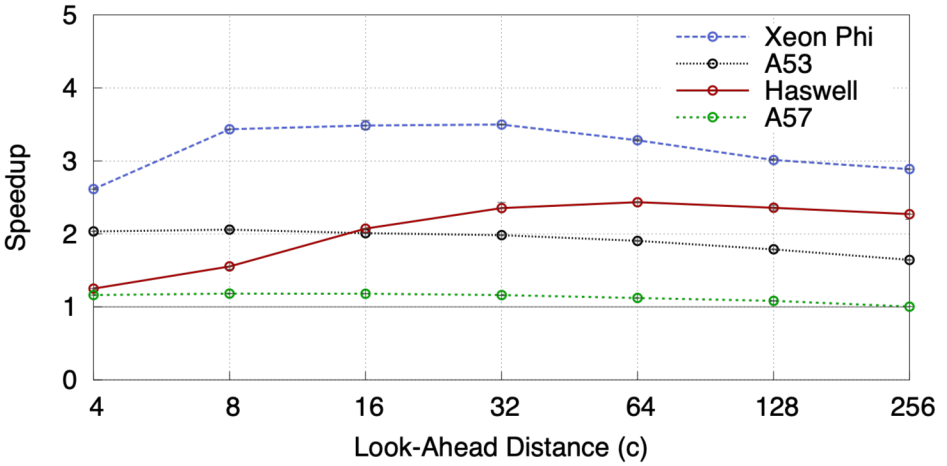
- Implemented as an LLVM pass
- Variety of benchmarks featuring indirects, hashing and multiple indirects, variety of data sizes
- Wide variety of core types (in-order/out-of-order) and ISAs

	System Specifications
Haswell	Intel Core i5-4570 CPU, 3.20GHz, DDR3 DRAM
Xeon Phi	Intel Xeon Phi 3120P CPU, 1.10GHz, GDDR5 DRAM
A57	Nvidia TX1, ARM Cortex-A57 CPU, 1.9GHz, LPDDR4 DRAM
A53	Odroid C2, ARM Cortex-A53 CPU, 2.0GHz, DDR3 DRAM

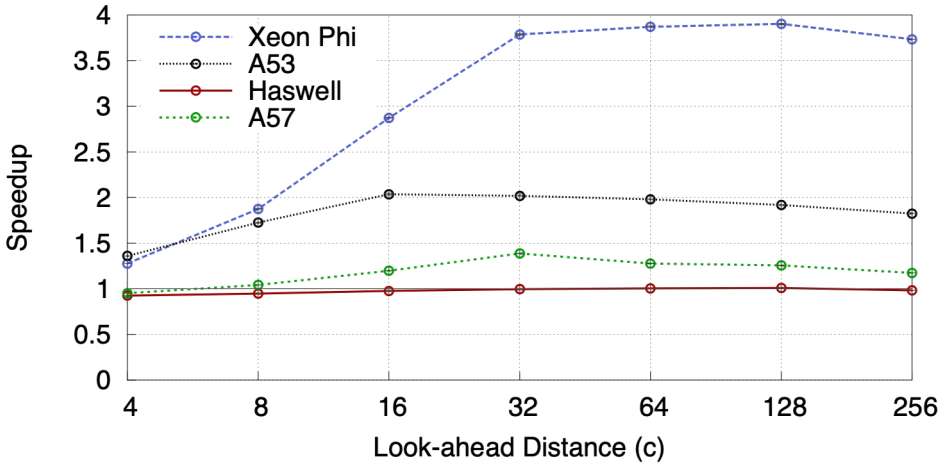
“Micro-architecture Specific” Constant (c)



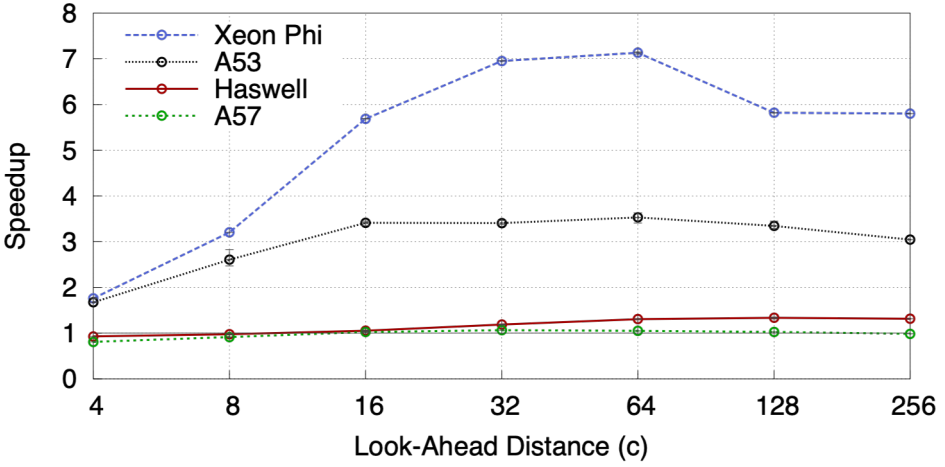
RandomAccess



Hash Join

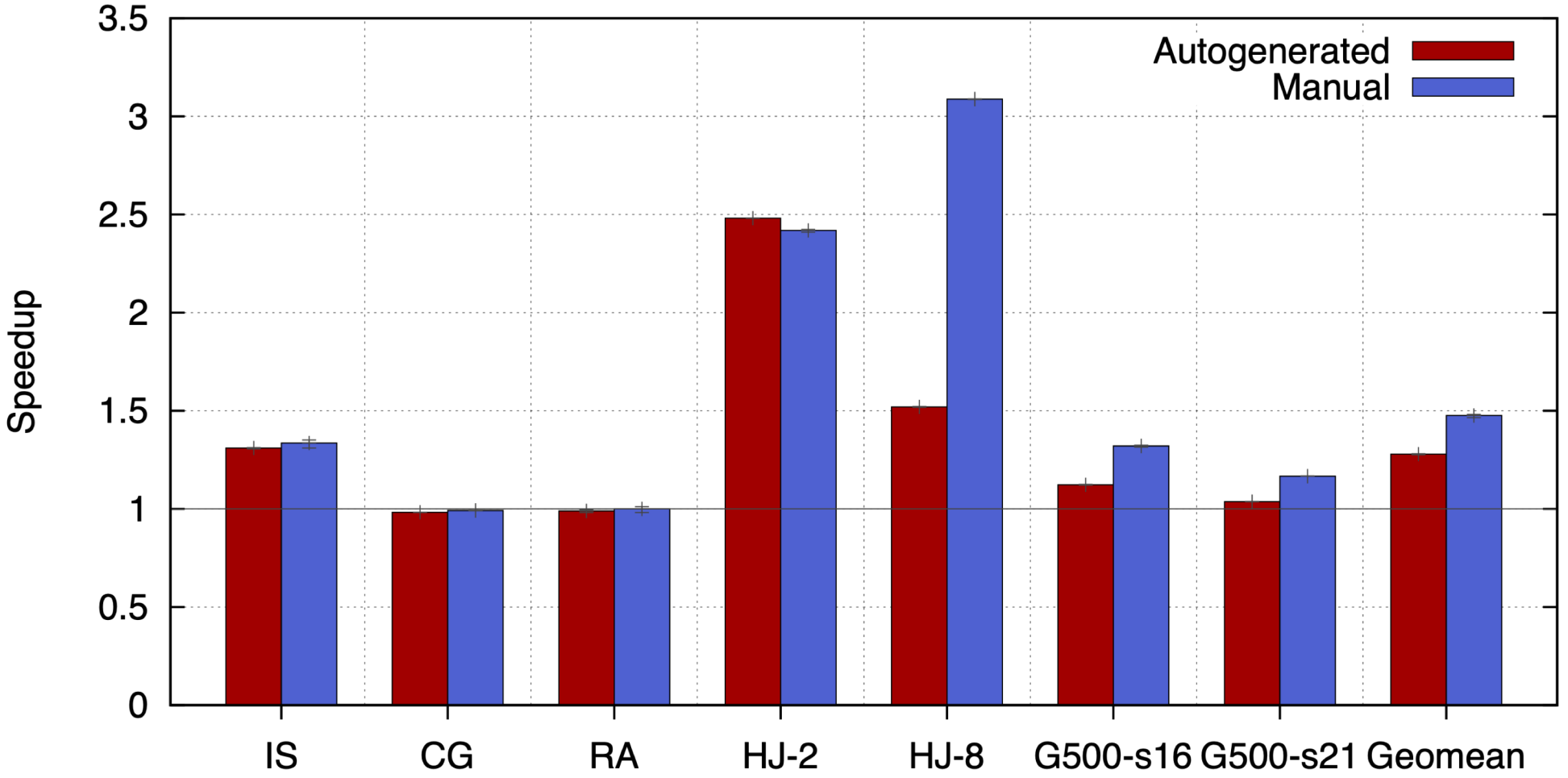


Conjugate Gradient

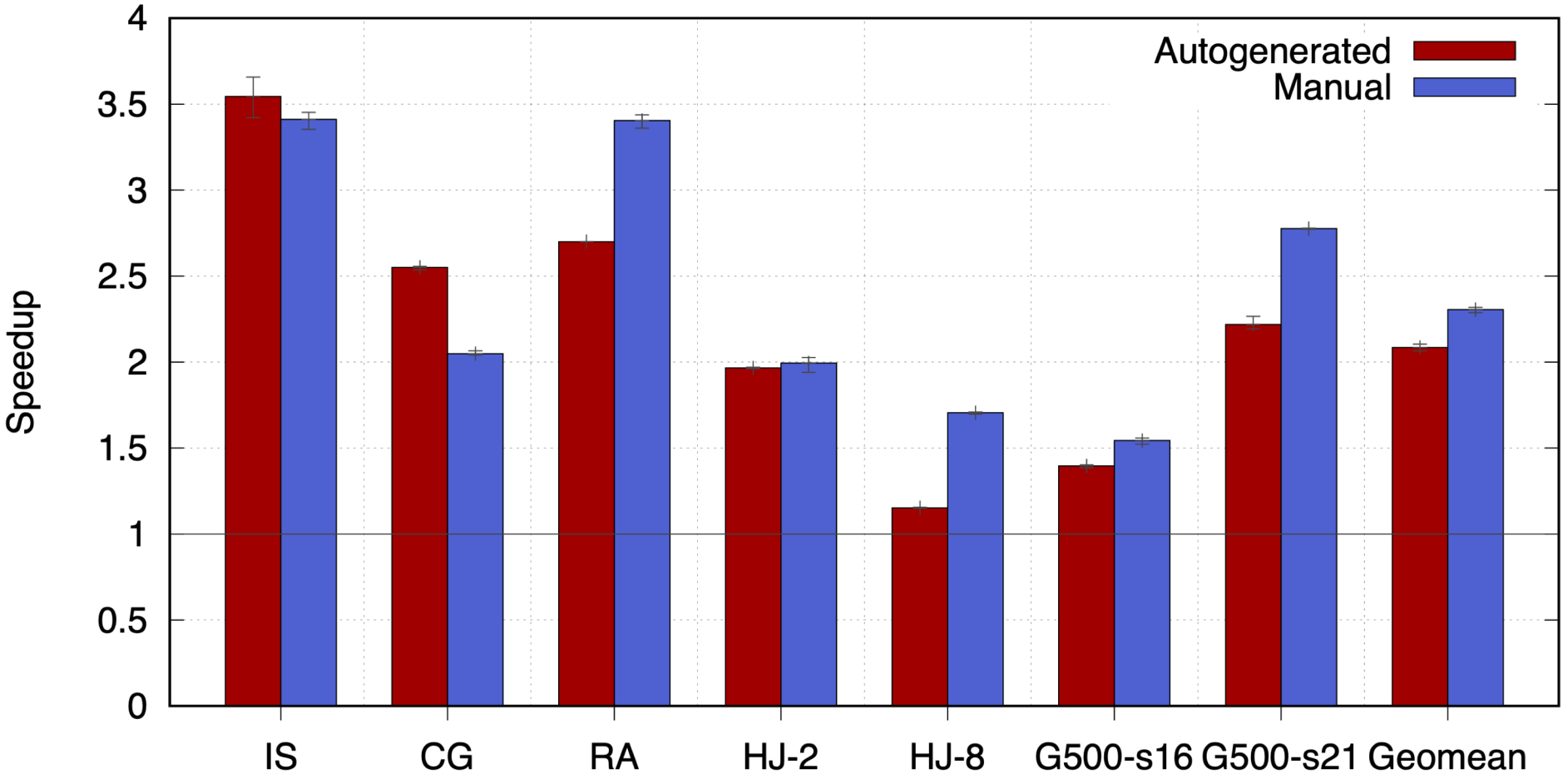


Integer Sort

Results – Intel Haswell



Results – ARM A53



Discussion

- **Advantages**
  - No modifications required to hardware or from the programmer
  - Works for a wide range of micro-architectures
  - Can capture more involved indirection patterns, e.g., a[func(b[i])]

- **Disadvantages**
  - Increase in number of dynamic instructions
  - Difficult to capture dynamic information
    - Detecting sizes of dynamic arrays
    - Constant prefetch look-ahead distance