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)

```c
for (i=0; i<a_size; i++) {
    b[a[i]]++;
}
```
Example: Integer Sort (NAS)

```c
for (i=0; i<a_size; i++) {
    b[a[i]]++;
}
```
Example: Integer Sort (NAS)

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

```c
for (i=0; i<a_size; i++) {
    SWPF(b[a[i + offset]]);
    b[a[i]]++;
}
```
Better Prefetching – Best Offset

```c
for (i=0; i<a_size; i++) {
    SWPF(b[a[i + offset]]);
    SWPF(a[i + offset*2]);
    b[a[i]]++;
}
```
Better Prefetching – Bad Offsets

```c
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

\[
\begin{align*}
\text{start: } & \text{ alloc } a, a\text{\_size} \\
& \text{ alloc } b, b\text{\_size} \\
\text{loop: } & \phi i, [\#0, i.1] \\
& \text{ gep } t1, a, i \\
& \text{ ld } t2, t1 \\
& \text{ gep } t3, b, t2 \\
& \text{ ld } t4, t3 \\
& \text{ add } t5, t4, \#1 \\
& \text{ str } t3, t5 \\
& \text{ add } i.1, i, \#1 \\
& \text{ cmp } \text{ size, i.1} \\
& \text{ bne loop}
\end{align*}
\]
Compiler-Automated Prefetch Insertion Algorithm

- Identification
- Safety Analysis
- Scheduling

\[
\text{start: alloc } a, \text{ a}_\text{size} \\
\text{alloc } b, \text{ b}_\text{size} \\
\text{loop: phi } i, [\#0, i.1] \\
\text{gep } t1, a, i \\
\text{ld } t2, t1 \\
\text{gep } t3, b, t2 \\
\text{ld } t4, t3 \\
\text{add } t5, t4, #1 \\
\text{str } t3, t5 \\
\text{add } i.1, i, #1 \\
\text{cmp } \text{size}, i.1 \\
\text{bne } \text{loop}
\]

\[
\begin{align*}
b[a[i]] & \text{++} \\
\text{if}(i+? < \text{a}\_\text{size}) & \\
\text{prefetch}(b[a[i+?]]) & \\
\text{prefetch}(a[i+??]) &
\end{align*}
\]
Compiler-Automated Prefetch Insertion Algorithm

- Identification
- Safety Analysis
- Scheduling

\[
c(t - (l-1)) = t
\]

\[c = \text{microarchitectural constant (64)}\]
\[t = \# \text{ loads in sequence}\]
\[l = \# \text{ loads in prefetch}\]

\[
b[a[i]]++
if(i+32 < a\_size)
prefetch(b[a[i+32]]) \quad (l=2)
prefetch(a[i+64]]) \quad (l=1)
\]
Large Speedups on Real Cores

Haswell

A57

A53

Xeon Phi

Compiler

Manual

ICC
Microarchitectural Constant (c)

HashJoin

RandAcc

ConjGrad

IntSort