Nov. 25 2023 @ PAAP2023

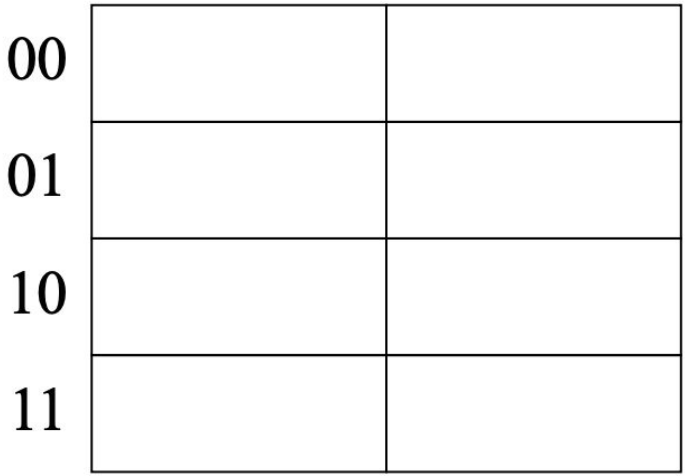
Global Store Statement Aggregation

Tomohiro Sano and Yasunobu Sumikawa
Takushoku University, Japan

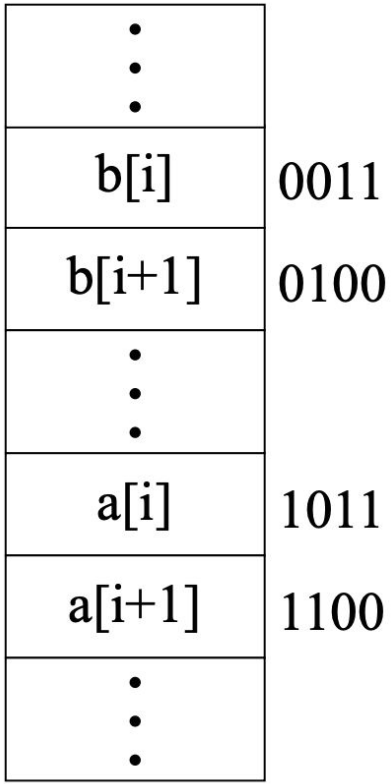
Background

```
main(){
  a[i]=x;
  b[i]=y;
  a[i+1]=z;
}
```

C program



Cache memory



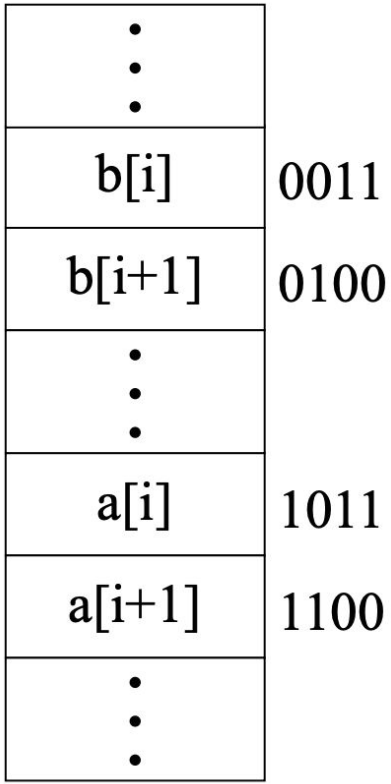
Main memory

Background

```
main(){
  a[i]=x;
  b[i]=y;
  a[i+1]=z;
}
```

C program

Cache memory

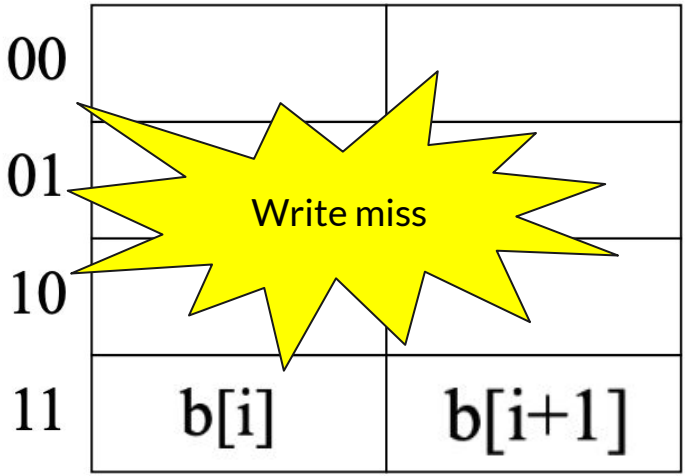


Main memory

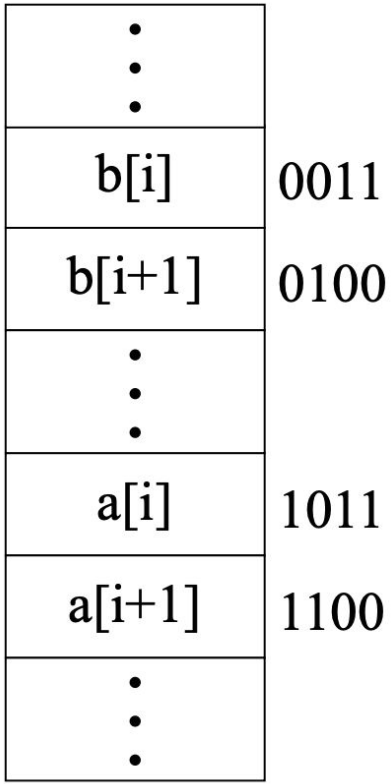
Background

```
main(){
  a[i]=x;
  b[i]=y;
  a[i+1]=z;
}
```

C program



Cache memory

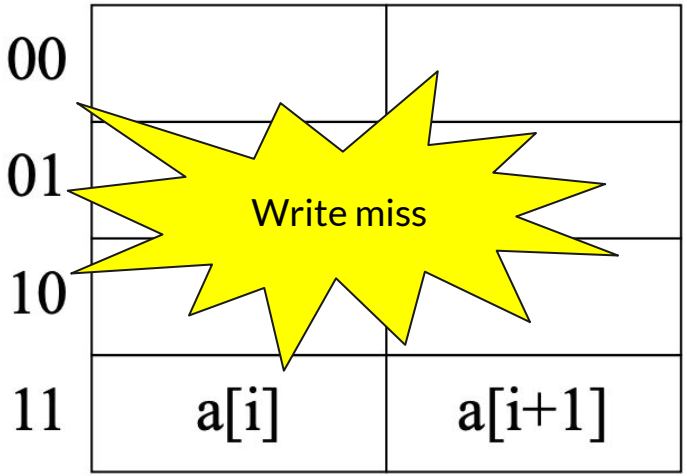


Main memory

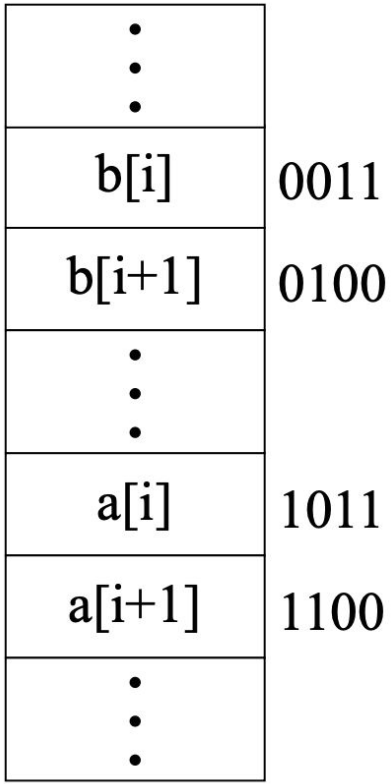
Background

```
main(){
  a[i]=x;
  b[i]=y;
  a[i+1]=z;
}
```

C program



Cache memory

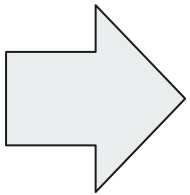


Main memory

Objective

We propose a novel code motion algorithm named Global Store statement Aggregation (GSA)

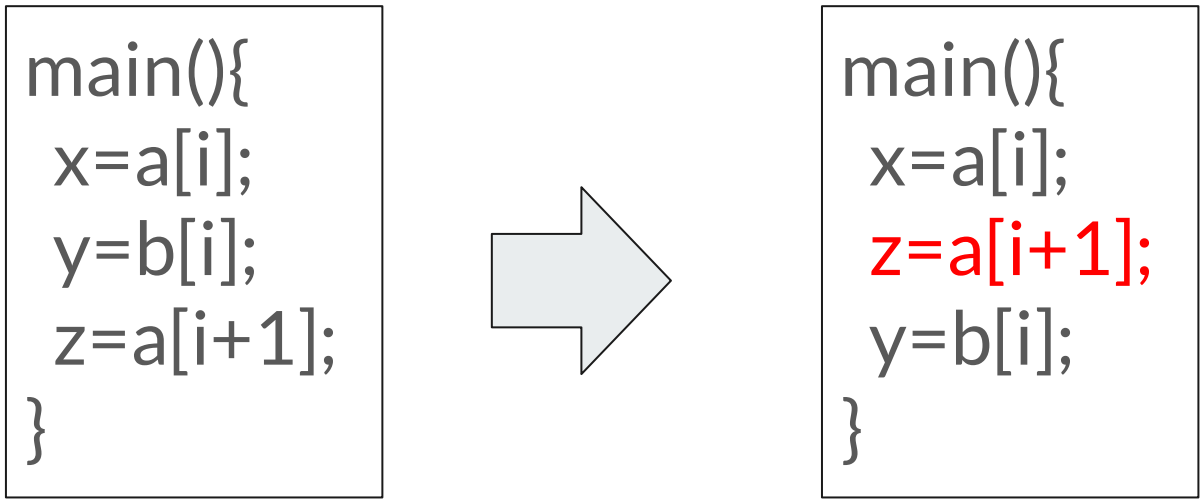
```
main(){
  a[i]=x;
  b[i]=y;
  a[i+1]=z;
}
```



```
main(){
  b[i]=y;
  a[i]=x;
  a[i+1]=z;
}
```

Related Works

Aggregating load statements has been proposed



Y. Sumikawa and M. Takimoto, “Global load instruction aggregation based on code motion,” in 5th PAAP, 2012, pp. 149–156.

Algorithm overview

1. Traversing the control flow graph (CFG) with reverse topological sort order
2. Sinking each store statement extended by PDE
3. Array reference analysis during the sinking

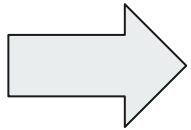
Algorithm overview

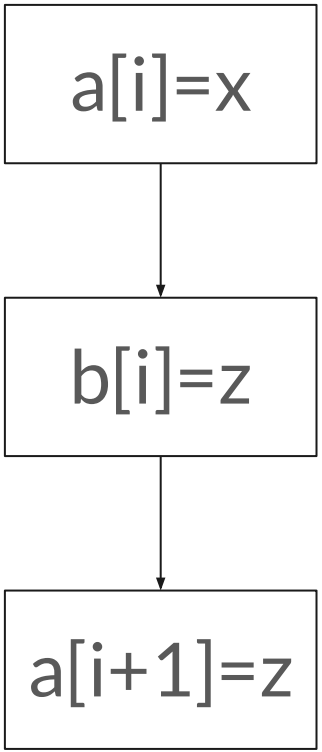
1. Traversing the control flow graph (CFG) with reverse topological sort order
2. Sinking each store statement extended by PDE
3. Array reference analysis during the sinking

CFG creation

```
main(){
  a[i]=x;
  b[i]=y;
  a[i+1]=z;
}
```

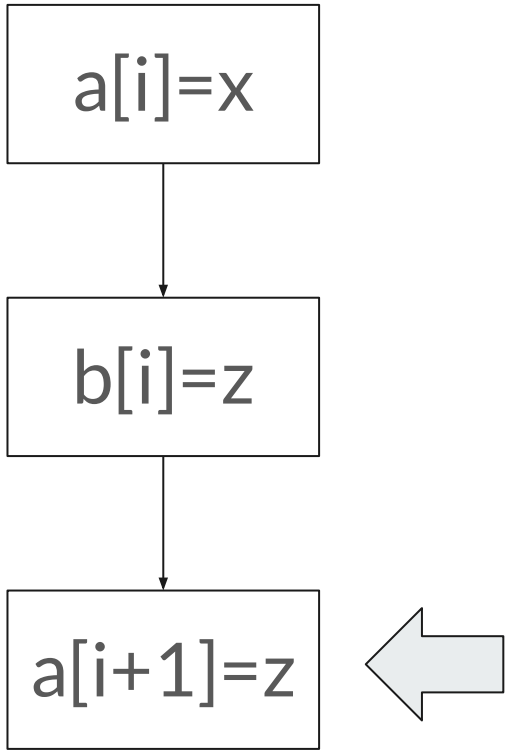
C program



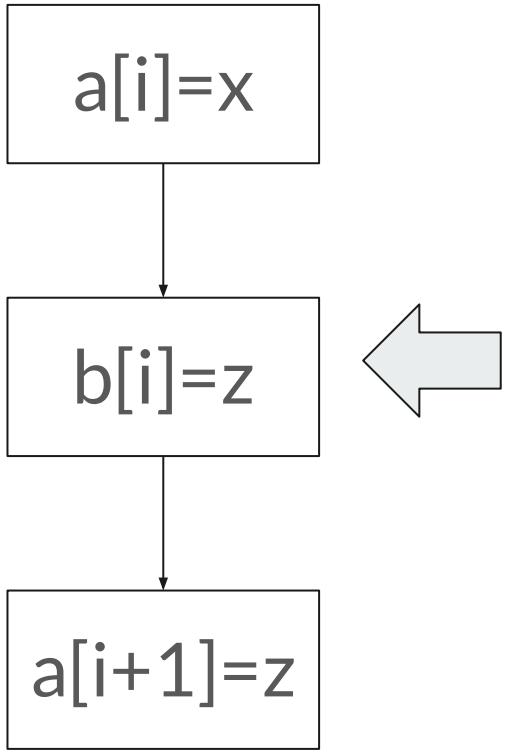


CFG (control flow graph)

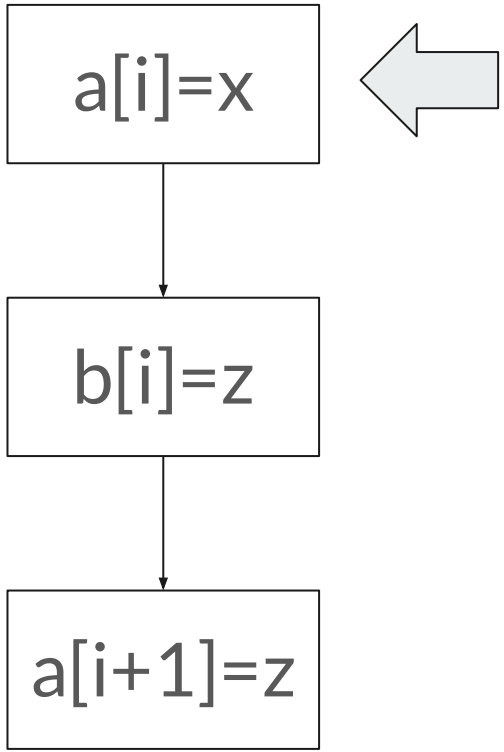
CFG traversing



CFG traversing



CFG traversing

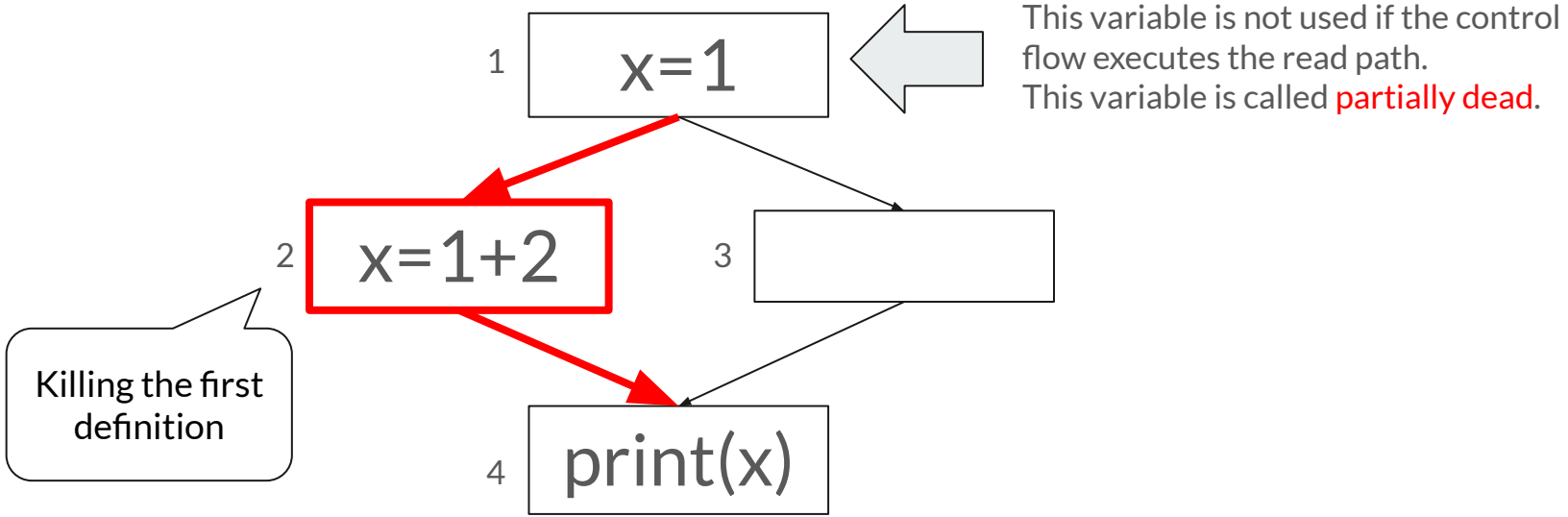


Algorithm overview

1. Traversing the control flow graph (CFG) with reverse topological sort order
2. Sinking each store statement extended by PDE
3. Array reference analysis during the sinking

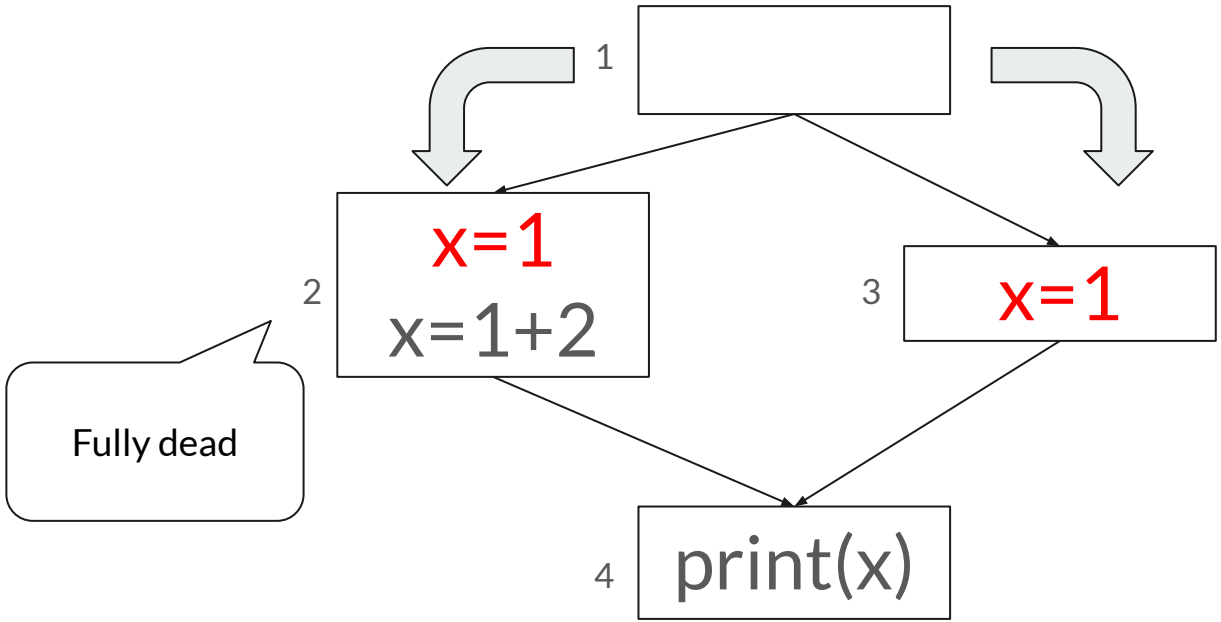
PDE

PDE eliminates partially dead variables

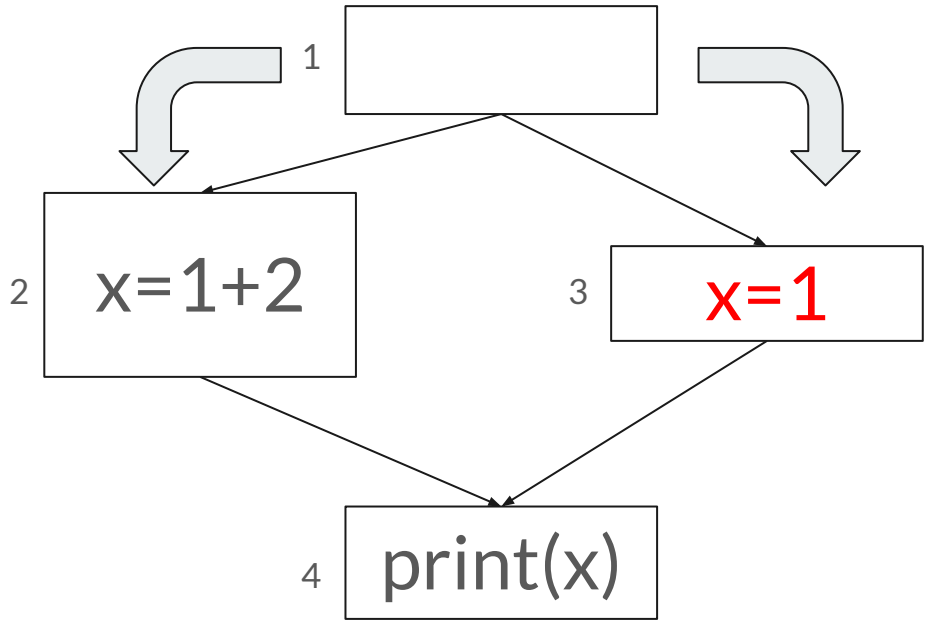


J. Knoop, O. Ruthing, and B. Steffen, “Partial dead code elimination,” ¨ SIGPLAN Not., vol. 29, no. 6, pp. 147–158, jun 1994.

Sinking of PDE

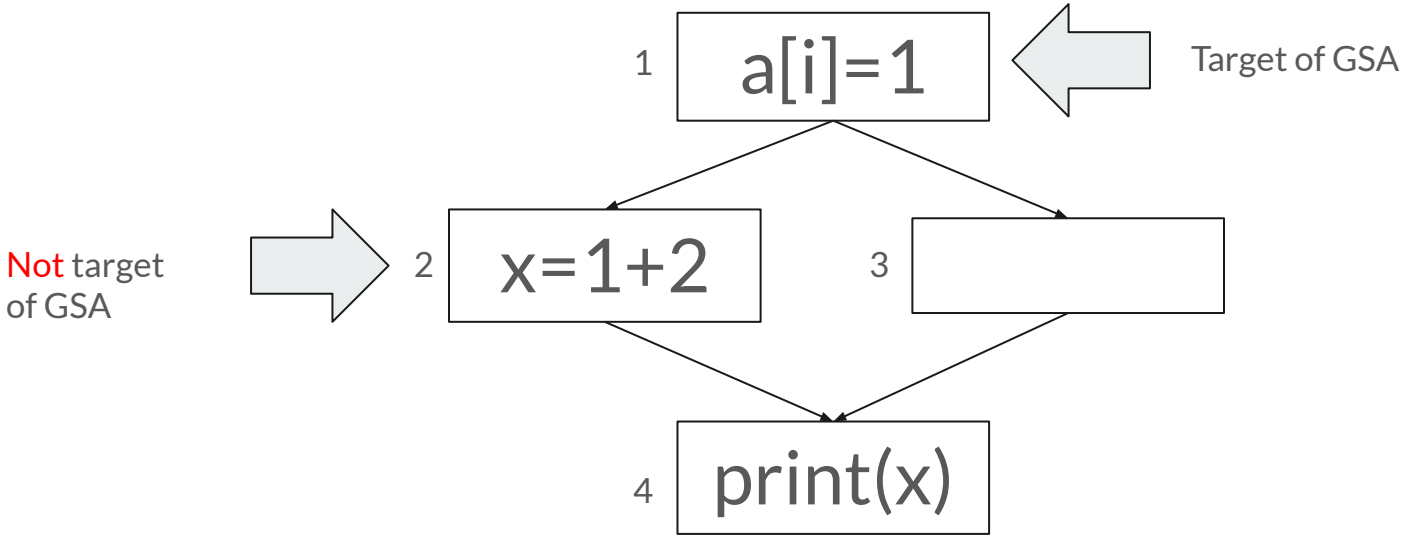


Elimination



PDE extension

GSA performs PDE sinking only for store statements



Algorithm overview

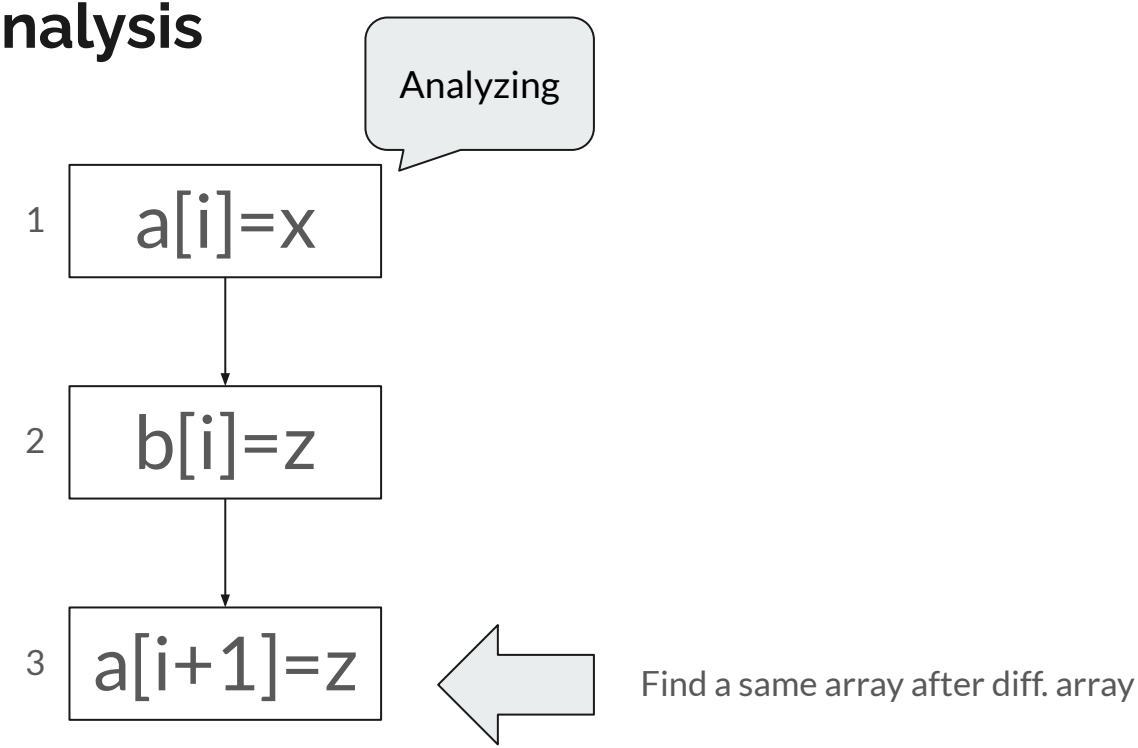
1. Traversing the control flow graph (CFG) with reverse topological sort order
2. Sinking each store statement extended by PDE
3. Array reference analysis during the sinking

Array reference analysis

1. Forwarding analysis
2. Backwarding analysis

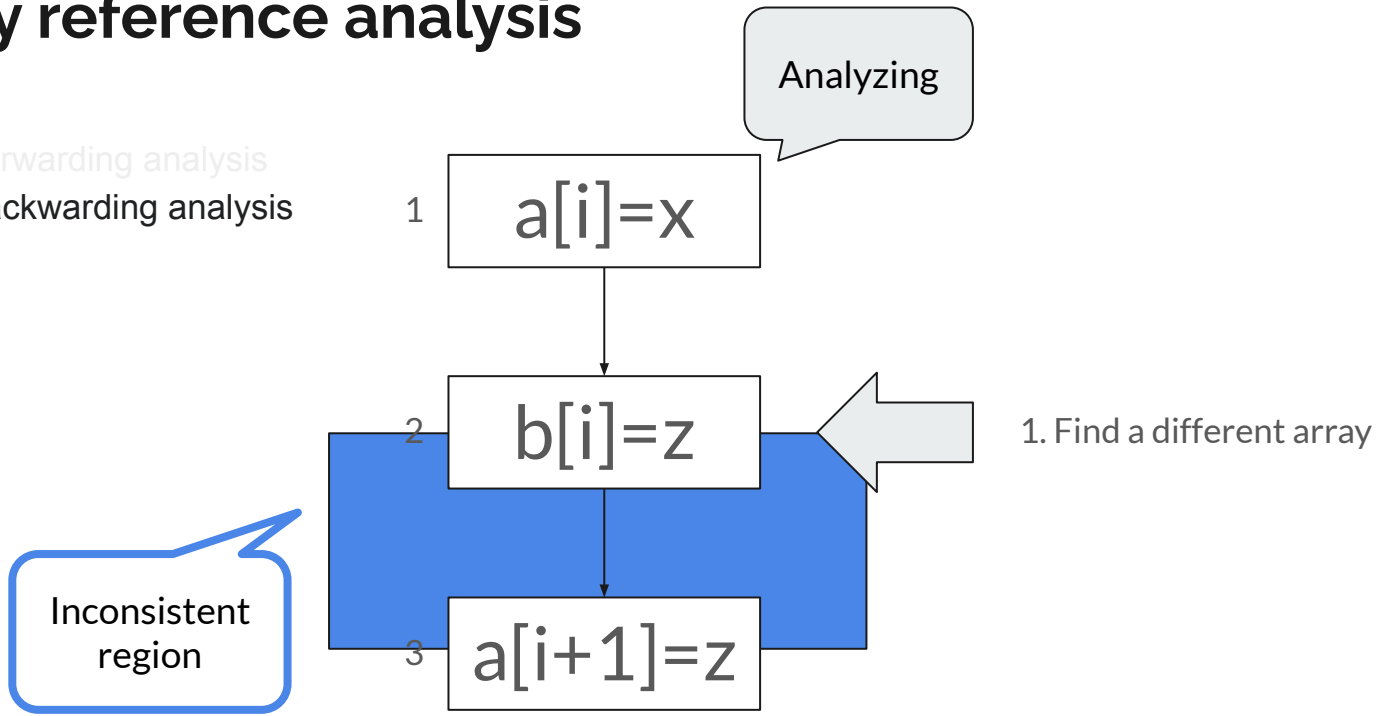
Array reference analysis

1. Forwarding analysis
2. Backwarding analysis

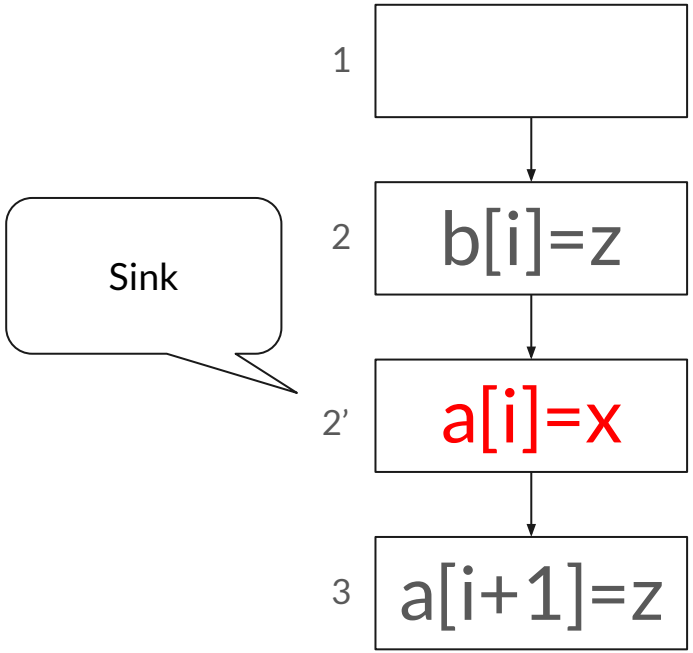


Array reference analysis

1. Forwarding analysis
2. Backwarding analysis



Sinking using the array reference analysis



Evaluation

Compiler: COINS

Baseline: PDE

Benchmark program: distcountsort (**Count**), radixsort (**Radix**), and arrays(**Array**)

CPU: Intel Corei7-11700 2.50GHz

OS: Ubuntu 64bit

L1d and Li1 cache memories: 384 KiB, 256 KiB

L2 cache memory: 4 MiB

L3 cache memory: 16 MiB

Results | Total number of cache misses

	A. PDE	B. GSA	(A-B) / A
Count	25,480	25,266	0.84%
Radix	6,662	6,575	1.31%
Array	18,362	17,343	5.55%

Results | Number of last level cache store misses

	A. PDE	B. GSA	(A-B) / A
Count	17,918	17,889	0.16%
Radix	1,200	1,171	2.42%
Array	11,661	11,450	1.81%

Results | Execution time

	A. PDE	B. GSA	(A-B) / A
Count	1,564.8	1,342	14.24%
Radix	566.6	562.5	0.72%
Array	551.5	531.5	3.63%

Conclusion

- We proposed a novel code motion based cache optimization algorithm, named global store statement aggregation (GSA).
- GSA aims at reducing write misses by making store statements accessing the same array continuously.

Future Work

- we will examine *moving all store and load statements at the same time to enhance GSA*.

Thank you for your attention!