Software Prefetching

Bojian Zheng
bojian@cs.toronto.edu

CSCD70 Compiler Optimizations, Spring 2018

Assignment 3

Q & A

- Many good questions have already been asked on Piazza.
  - Please go through them first before solving the assignment.
- Please ignore the Profiling sections for now, because it seems that the option –stats is missing in lli (thanks to Stone Jin).
- Please name your pass loop-invariant-code-motion as licm seems to contradict with built-in LLVM pass (thanks to Lioudmila Tishkina).

Q & A

- Please write your test cases in do-while loop because special handling is required for for-loop and while-loop (thanks to Terrence Hung).
- Why? Consider the code on the right hand side:

```
j = 5;

for (i = 0; i < ???; ++i)
        j = 10;

printf("%d", j);
```

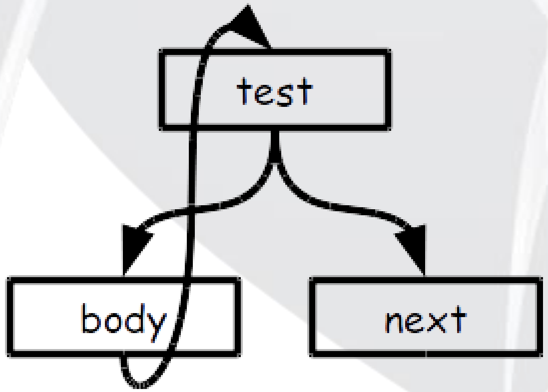
Q & A

- Idea: Body statements are not guaranteed to execute, therefore cannot perform code motion.
- Need to perform the Landing-Pad Transformation first before LICM.

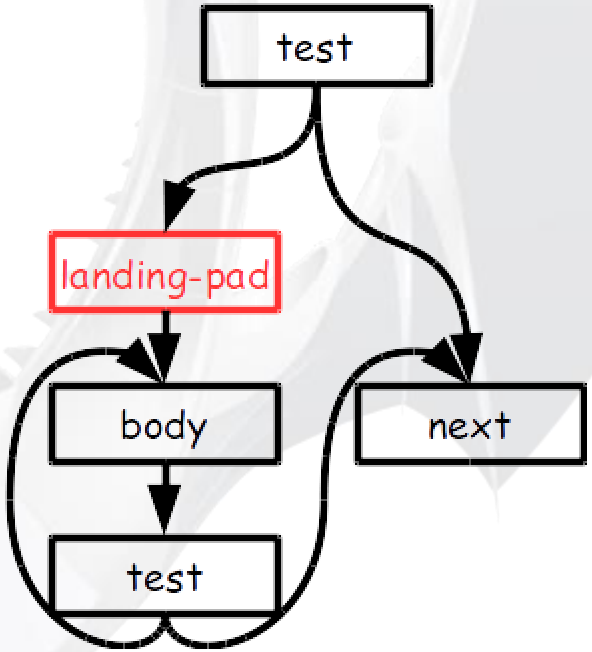
```
j = 5;


for (i = 0; i < ???; ++i)
        j = 10;


printf("%d", j);
```

Landing-Pad Transformation

Before



After



Landing-Pad Transformation

Before

```
j = 5;

for (i = 0; i < ???; ++i)
        j = 10;

printf("%d", j);
```

After

```
j = 5; i = 0;

if (i < ???) { // Landing-Pad
        do {
                j = 10;
                ++i
        } while (i < ???); }

printf("%d", j);
```

Assignment 3 Hints

1. Compute Loop Invariants:
    1) Traverse through the loop, and get all the definitions inside.
    2) Create a mapping from Instruction to bool (isInvariant).
        - Keep iterating until no changes to such mapping occurs.
        - Mark instruction as isInvariant i.f.f. its operands have one of the following properties: (1) constant (2) definition ∉ definitions inside (3) definition ∈ definition inside but definition has already been marked as isInvariant.
        - Do not forget to also include the additional constraints mentioned in the handout (isSafeToSpeculativelyExecute …).

Assignment 3 Hints

2. Compute Dominator Tree:
    - Please refer to the tutorial demo on SSA on how this was done for Dominance Frontier.

3. Compute Loop Exit:
    - llvm::Loop has built-in method call that tells you this.

Assignment 3 Hints

4. Compute candidates for Code Motion:
   - Must be invariant.
   - Must dominate exit blocks.
   - Must have only one definition?
     - No need to worry about this because of SSA.
5. Perform Code Motion:
   - Move candidates to the Loop Preheader, if there exists.

Questions?

1. Compute Loop Invariants.
2. Compute Dominator Tree.
3. Compute Loop Exit
4. Compute candidates for Code Motion.
5. Perform Code Motion.

Software Prefetching

Software Prefetching

- Recall that in our last class, we mentioned the fundamental idea of prefetching – move data close to the processor (e.g. cache) before it is needed.
- Need to answer the following two questions: (1) what to prefetch and (2) when & how to prefetch.

What to prefetch?

- Use Prefetch Predicate:

  Locality Analysis ⇒ Prefetch Predicate (what to prefetch)
- Locality Analysis: Recall from last class that

  $$\text{reuse} \cap \text{localized} = \text{locality}$$

  - where reuse answers the question "under which condition are we going to access the exact same element (Temporal Locality) or elements of the same row (Spatial Locality), under the condition that cache is infinitely large"; localized is determined by how large our working set is compared with our cache size.

Recall: Locality Analysis

```
double A[3][N], B[N][3];

for (i $\in [0, 3)$)
  for (j $\in [0, N - 1)$)
    A[i][j] = B[j][0] + B[j + 1][0];
```

// row-major, 2 elements per cache block, N is small (working set < cache size).

- A[i][j]: Spatial Locality on inner loop j
- B[j + 1][0]: Temporal Locality on outer loop i
- B[j][0]: Group Locality due to leading reference B[j + 1][0]

Miss Instances

```
double A[3][N], B[N][3];

for (i ∈ $[0, 3)$)
  for (j ∈ $[0, N - 1)$)
    A[i][j] = B[j][0] + B[j + 1][0];
```

// row-major, 2 elements per cache block, N is small (working set < cache size).

- Need to understand the miss instances.
- What are the miss instances on A[i][j] and B[j + 1][0]?

Miss Instances - Temporal Locality

```
double A[3][N], B[N][3];

for (i ∈ $[0, 3)$)
  for (j ∈ $[0, N-1)$)
    A[i][j] = B[j][0] + B[j + 1][0];
```

// row-major, 2 elements per cache block, N is small (working set < cache size).

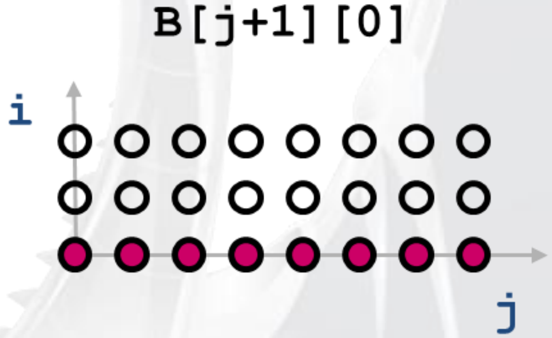
- Consider B[j + 1][0], which has Temporal Locality on outer loop i.

Miss Instances – Temporal Locality

double A[3][N], B[N][3];

```
for (i ∈ $[0, 3)$)
  for (j ∈ $[0, N - 1)$)
    A[i][j] = B[j][0] + B[j + 1][0];
```

// row-major, 2 elements per cache block, N is small (working set < cache size).



Temporal

Miss Instances - Temporal Locality

```
double A[3][N], B[N][3];

for (i $\in [0, 3)$)
  for (j $\in [0, N-1)$)
    A[i][j] = B[j][0] + B[j + 1][0];
```

// row-major, 2 elements per cache block, N is small (working set < cache size).

- Consider B[j + 1][0], which has Temporal Locality on outer loop i.
- Misses happen during our 1st iteration of outer loop i.
- Therefore, predicate is true when i = 0.

Miss Instances - Spatial Locality

```
double A[3][N], B[N][3];

for (i $\in [0, 3)$)
  for (j $\in [0, N - 1)$)
    A[i][j] = B[j][0] + B[j + 1][0];
```

// row-major, 2 elements per cache block, N is small (working set < cache size).

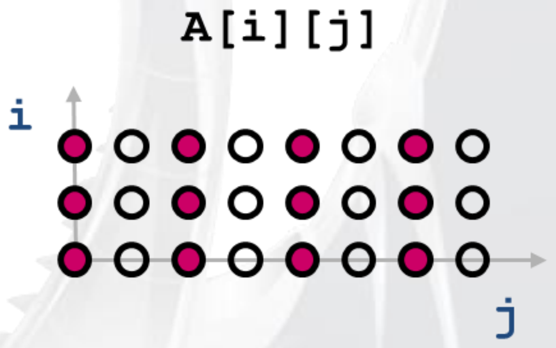
- Consider A[i][j] which has Spatial Locality on inner loop j.

Miss Instances – Spatial Locality

double A[3][N], B[N][3];

```
for (i ∈ $[0, 3)$)
  for (j ∈ $[0, N - 1)$)
    A[i][j] = B[j][0] + B[j + 1][0];
```

// row-major, 2 elements per cache block, N is small (working set < cache size).



Spatial

Miss Instances – Spatial Locality

```
double A[3][N], B[N][3];

for (i $\in [0, 3)$)
  for (j $\in [0, N - 1)$)
    A[i][j] = B[j][0] + B[j + 1][0];
```

// row-major, 2 elements per cache block, N is small (working set < cache size).

- Consider A[i][j] which has Spatial Locality on inner loop j.
- Misses happen every L iteration of inner loop j (where L denotes the # of elements per cache block).
- Therefore, predicate is true when j mod L = 0.

Prefetch Predicate

Locality	**Miss Instances**	**Predicate**
None	Every Iteration	true
Temporal	1st Iteration	$i = 0$
Spatial	Every L Iteration	$i \bmod L = 0$

Prefetch Insertion

- Given that now we have Prefetch Predicate, how are we going to insert them?
- Consider the code on the right hand side:

```
double a[100];

for (i $\in [0, 100)$)
        a[i] = 0;

// 2 elements per cache block
```

Loop Splitting

if

```
double a[100];

for (i $\in [0, 100)$)
    if (i mod 2 == 0)
        prefetch …
    a[i] = 0;

// 2 elements per cache block
```

Loop Unrolling

```
double a[100];

for (i $\in [0, 100)$, i += 2)
    // NO “if” is needed!
    prefetch …
    a[i] = 0; a[i + 1] = 0;

// 2 elements per cache block
```

Loop Splitting

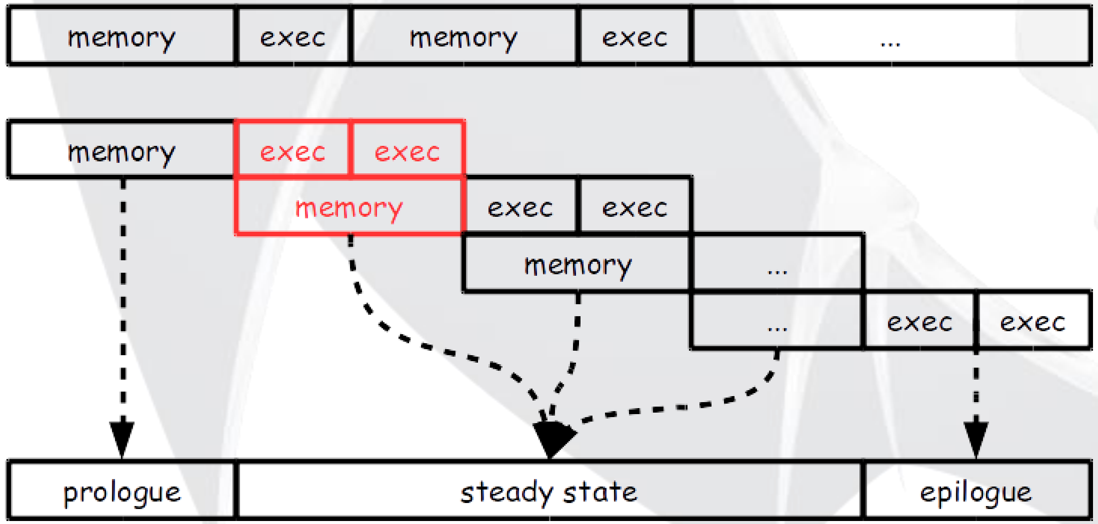
- Idea: Isolate Miss Instances by Loop Splitting.
  - Temporal Locality ⇒ Misses on 1st Iteration ⇒ Peel the 1st Iteration
  - Spatial Locality ⇒ Misses every L Iteration ⇒ Unroll by L

Software Pipelining

```
double a[100];

for (i ∈ $[0, 100)$, i += 2)
        prefetch ______
        a[i] = 0; a[i + 1] = 0;

// 2 elements per cache block
```

- What should "______" be?
  - a[i]? a[i + 2]?

Software Pipelining



- The answer depends on the relative ratio between memory access latency and shortest path through loop body.
- To fully hide the memory latency with execution, prefetch what is needed for the next $\left\lceil \frac{\text{mem}}{\text{exec}} \right\rceil$ iterations

Software Pipelining

```
double a[100];

for (i $\in [0, 100)$, i += 2)
        prefetch ________
        a[i] = 0; a[i + 1] = 0;

// 2 elements per cache block
```

```
double a[100];

for (i $\in [0, 6)$, i+= 2) // prologue
        prefetch a[i]

for (i $\in [0, 94)$, i += 2)  // steady state
        prefetch a[i + 6]
        a[i] = 0; a[i + 1] = 0;

for (i $\in [94, 100)$, i += 2) // epilogue
        a[i] = 0; a[i + 1] = 0;

// 2 elements per cache block, $\left\lceil \frac{\text{mem}}{\text{exec}} \right\rceil = 6$
```

Questions?

- Locality Analysis ⇒ Miss Instances ⇒ Prefetch Predicate

Locality	**Miss Instances**	**Predicate**
None	Every Iteration	true
Temporal	1st Iteration	i = 0
Spatial	Every L Iteration	i mod L = 0

- Loop Splitting & Software Pipelining