Loop Fusion

Pratik Fegade
CS 616 Seminar

IIT Bombay

October 16, 2015

Outline

Motivation

Loop Distribution: Necessity

- Distribute loops across every statement possible
- Done to extract as much parallelism as is possible
- The statements can then be marked *parallel* or *sequential* as appropriate

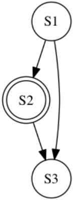
Loop Distribution: The Problem and a Solution

- Leads to too many loops, and hence a large loop overhead
- Often statements originally from the same loop can be fused into a larger loop
- Loop fusion does precisely that

Loop Fusion: Safety Criteria

Ordering Constraint

One cannot fuse two parallel nodes if they have a path between them with a sequential node in it.



In the DDG shown, one cannot fuse S1 and S3, because of the path S1$\rightarrow$S2$\rightarrow$S3

Fusion Preventing Dependences

It may happen that after fusion, a dependence becomes a loop carried one in the reverse direction, which is semantically wrong.

```
    DO I = 1, N
S1:   A(I) = B(I) + C
    ENDDO
    DO I = 1, N
S2:   D(I) = A(I+1) + E
    ENDDO
```

$$S1 \xrightarrow{\delta_\infty} S2$$

```
    PARALLEL DO I = 1, N
S1:   A(I) = B(I) + C
S2:   D(I) = A(I+1) + E
    ENDDO
```

$$S2 \xrightarrow{\overline{\delta}_I} S1$$

Loop Fusion: Profitability Considerations

Parallelism Inhibiting Edges

Fusing two parallel statements may lead to a sequential/difficult-to-parallelize loop, a clearly unprofitable result in general.

```
    DO I = 1, N
S1:   A(I+1) = B(I) + C
    ENDDO
    DO I = 1, N
S2:   D(I) = A(I) + E
    ENDDO
```

$$\mathrm{S1} \xrightarrow{\delta_\infty} \mathrm{S2}$$

```
    DO I = 1, N
S1:   A(I+1) = B(I) + C
S2:   D(I) = A(I) + E
    ENDDO
```

$$\mathrm{S1} \xrightarrow{\delta_I} \mathrm{S2}$$

Typed Fusion: A General Problem

Typed Fusion

Motivation

- We have been labeling nodes (i.e. statements or loops, as we assume loops are maximally distributed), as sequential or parallel
- Labeling based on parallelizablity of statements
- More than two types of statements my be desired

Typed Fusion

Motivation

```
L1: DO I = 1, IMAXD
      DO J = 1, JMAXD
        F(I,J,1) = F(I,J,1)*B(1)
L2: DO K = 2, N-1
      DO J = 1, JMAXD
        DO I = 1, IMAXD
          F(I,J,1) = (F(I,J,K) - A(K)*F(I,J,K-1))*B(K)
```

L1 and L2 have different loop headers and it may be convenient to classify them into two different types.
Note: This may also be achieved by adding bad edges between nodes of different types, but that may lead to many edges.

Typed Fusion

The Problem

The Typed Fusion problem can thus be stated as a tuple $\mathrm{P} = (G, T, m, B, t_0)$, where

- $G = (V, E)$ is a graph
- T is a set of types of nodes
- $m : T \to V$ such that $m(v)$ for a node $v \in V$ is the *type* of v
- $B \subseteq E$ is a set of *bad edges*
- $t_0 \in T$ is the objective type

Typed Fusion

The Problem

A solution to the Typed Fusion problem is a graph $G' = (V', E')$ where V' is obtained from V by fusing nodes of type t_0 subject to the following constraints

- Bad Edge Constraint: No two vertices joined by a bad edge may be fused, and
- Ordering Constraint: No two vertices joined by a path containing a vertex of type $t \neq t_0$ may be fused.

Typed Fusion: Kennedy and McKinley's Algorithm

Typed Fusion

Kennedy and McKinley's Algorithm

- Greedy algorithm to optimally fuse nodes of objective nodes
- Maintains, for each node the highest node of the same type in the fused graph with which it cannot be fused
- Thus, keeps track of the bad paths
  - Paths between nodes of same type with heterogeneous nodes or bad paths
- Finding node to fuse with thus is constant time

Typed Fusion

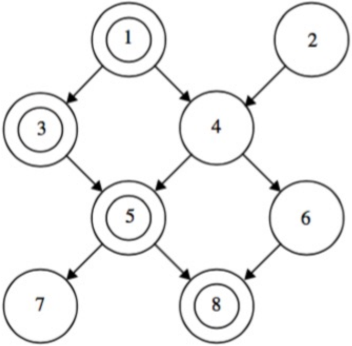
Kennedy and McKinley's Algorithm

- This data is maintained in the array `maxBadPrev`
- Crux of the algorithm lies in the procedure `update_successors` below

```
/* Call to update_successors(n, t)
   (n, m) is an edge
   t is type(n) */
if (t != t0)
  maxBadPrev[m] = MAX(maxBadPrev[m], maxBadPrev[n]);
else // t = t0
  if (type(m) != t0 or (n, m) in B) // bad edge
    maxBadPrev[m] = MAX(maxBadPrev[m], num[n]);
  else // equal types and not fusion preventing
    maxBadPrev[m] = MAX(maxBadPrev[m], maxBadPrev[n]);
```

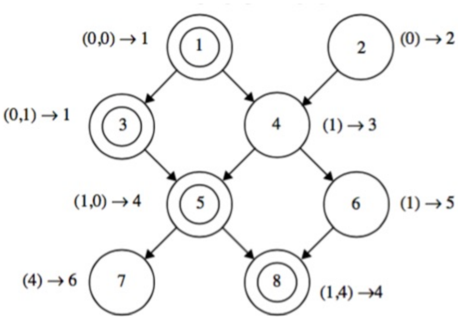
Kennedy and McKinley's Algorithm

An Example



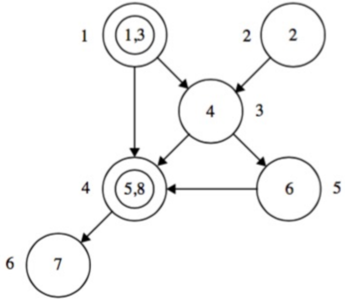
Kennedy and McKinley's Algorithm

An Example



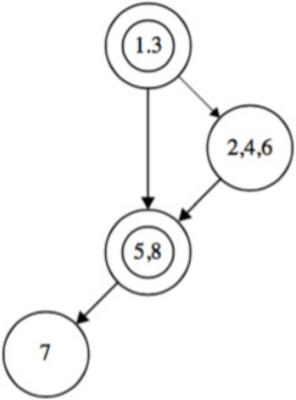
Kennedy and McKinley's Algorithm

An Example



Kennedy and McKinley's Algorithm

An Example



Kennedy and McKinley's Algorithm

Correctness

- Clearly, the algorithm only fuses nodes of type t_0
- `maxBadPrev` can be shown to be correctly maintaining the information it is supposed to be
- Bad paths take into consideration both the bad edge and the ordering constraints

Kennedy and McKinley's Algorithm

Optimality

- Greedily fuse a node if it can be with a node of the current fused graph
- Assuming an optimal solution, can devise a transform that may only shrink it to the greedy solution
- Thus, the greedy solution is the same size as the optimal one

Kennedy and McKinley's Algorithm

Time Complexity

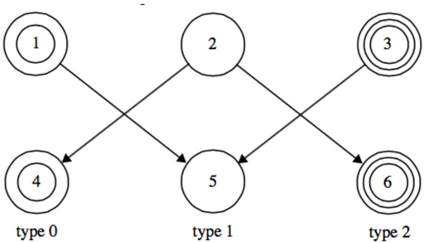
- Each node picked up at most once from W
- In `update_successors`, one iterates over the children of a node
- Thus, each edge is looked at once
- Time complexity is thus $\mathrm{O}(\mathrm{V} + \mathrm{E})$

Ordered and Unordered Typed Fusion

- When the order is given, it is an instance of the ordered typed fusion problem,
- else it is an instance of the unordered typed fusion problem
- The unordered typed fusion problem is an NP-hard problem

Order of Fusion

When there are more than one type of nodes to be fused, the order matters



Cohort Regions

Cohort Regions

- Running independent loops in parallel increases amount of exploited parallelism
- Sets of such loops are called cohorts

Cohort Regions

Some Observations

Any two loops in a cohort regions can be run in parallel. Thus,

- Two parallel loops in a cohort may not have a fusion-preventing or parallelism-inhibiting dependence
- There should not be any dependences between sequential and parallel loops

Cohort Regions

Generation

The problem of cohort fusion can then be solved the *TypedFusion* algorithm if all nodes are treated to be of one type and the following edges are treated as bad.

- Fusion-preventing edges
- Parallelism-inhibiting edges
- Edges between a parallel and a sequential loop

Thank You!