Forall Context

Genghan Zhang

May 2022

1 Introduction

When can we determine the reduction of an Einstein notation with provenance graph? In other words, the question is whether a workspace will be added to one or multiple times? It's safe to assume that += is used if a workspace is accessed multiple times, otherwise = (Theorem 1). So how can we infer such information from the concrete index notation?

2 IndexSetRel

The first step is the relation between the set of indexVars in lhs and rhs. We denote them as $\mathcal{L}$ and $\mathcal{R}$. We just assume the reduction operator is +=. We enumerate all the possible relations and find that only for two relations we can't determine the reduction operator directly from the relation.

Table 1: Relation, operator and example

Case	Relation	Operator	example
1	$(\mathcal{L} \cap \mathcal{R} \neq \emptyset) \wedge (\mathcal{L} \cap \mathcal{R} \neq \mathcal{L}) \wedge (\mathcal{L} \cap \mathcal{R} \neq \mathcal{R})$	+=	$ws(i,j) = A(i,k) * B(k,j)$
2	$\mathcal{L} \subsetneqq \mathcal{R}$	+=	$ws(i) = A(i,k) * B(i,k)$
3	$\mathcal{R} \subsetneqq \mathcal{L}$	=	$ws(i,k) = A(i) * B(i)$
4	$(\mathcal{L} \cap \mathcal{R} = \emptyset) \wedge (\mathcal{L} \neq \emptyset) \wedge (\mathcal{R} \neq \emptyset)$	+=	$ws(i1) = A(i) * B(i)$
		=	$B_{new}(i1) = B(i)$
5	$\mathcal{L} = \mathcal{R}$	+=	$ws(i1) = A_{new}(i1) * B_{new}(i1)$
		=	$ws(i) = A(i) * B(i)$

Case 4 can only happen in provenance graph, because such Einstein notation is not valid if indexes iterate on each dimension. However, if we do $split(i, i0, i1)$ first, $i1$ can be restored using i. Therefore, indexes actually iterate on part the shared dimension (the reduction dimension of the child node). How are operators in Case 4 and 5 determined? That's why we need **Forall Context**.

3 Forall Context

We only consider the statement without sequence. First, we introduce *last level forall* (LLF). $LLF := \forall_{index} \quad assignment$. Each LLF has a *forall context*. We build a *forall context tree* (FCT) based on the provenance graph to define forall context.

3.1 Build FCT

Traverse the provenance graph. When we meet a whereNode, if FCT is empty, add an *applicant node* (AN) and put Ø into it (which is called *content* of an AN). Otherwise, assign a left child and a right child to the current AN and put Ø into them. The left child of an AN corresponds to the consumer of current whereNode, and the right to the producer. When we meet a forallNode, add the indexVar to the content of current AN. Content of leaves of FCT will be the indexVar of LLF in the provenance graph.

Definition 1. *Forall Context of an LLF is defined to be the union of ANs' contents along the path from the root to the leaf corresponding to the LLF.*

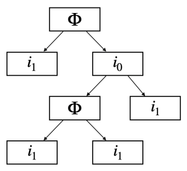
Theorem 1. *If a workspace is accessed only once in the current forall context, then the operator must be = , otherwise it must be +=.*

3.2 Example

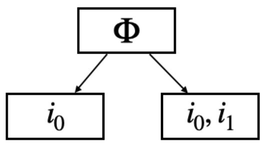
For the IndexStmts below:

```
Stmt1 = suchthat(where(forall(i1, A += ws(i1)), forall(i0, where(where(forall(i1,
ws(i1) += B_new(i1) * C_new(i1)), forall(i1, C_new(i1) = C(i))), forall(i1,
B_new(i1) = B(i))))), split(i, i0, i1, 32))

Stmt2 = suchthat(where(forall(i0, A += ws(i0)), forall(i0, forall(i1,
ws(i0) += B(i) * C(i)))), split(i, i0, i1, 32))

Stmt3 = suchthat(where(forall(i0, A += ws(i0)), forall(i0, where(where(forall(i1,
ws(i0) += B_new(i1) * C_new(i1)), forall(i1, C_new(i1) = C(i))), forall(i1,
B_new(i1) = B(i))))), split(i, i0, i1, 32))
```

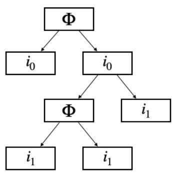
the FCTs are in figure 1:



(a) FCT of Stmt1



(b) FCT of Stmt2



(c) FCT of Stmt3

Figure 1: FCT of the examples

The forall context of LLFs in Stmt1 are $\{i_1\}, \{i_0, i_1\}, \{i_0, i_1\}, \{i_0, i_1\}$. The forall context of LLFs in Stmt2 are $\{i_1\}, \{i_0, i_1\}$. The *depth* of an LLF is defined as the number of elements in the leaf. All LLFs in Stmt1 have depth 1. In Stmt2, the left leaf has depth 1, and the right leaf has depth 2.

3.3 Code

The code is in index_notation.cpp IndexSetRel Assignment::getIndexSetRel().

4 Redundant IndexStmt Rewrite

4.1 Visitor

Algorithm 1: RedundantVisitor

```
Data: $to\_change \leftarrow empty$, $ctx\_stack \leftarrow empty$, $num\_stack \leftarrow empty$
Function visit($ForallNode$):
    stackPush($forallVar$);
    Update $num\_stack$;
end
Function visit($WhereNode$):
    $num\_stack.push\_back(0)$;
    visit($consumer$);
    stackPop();
    $num\_stack.push\_back(0)$;
    visit($producer$);
    stackPop();
end
Function visit($AssignmentNode$):
    $a \leftarrow current\_node$;
    if IndexSetRel($a$) is equal then
        if some indexVar in lhs has sibling in ctx_stack then
            $to\_change.push\_back(a)$;
        end
    end
    if IndexSetRel($a$) is none then
        if ctx_stack except the top contains indexVars in lhs then
            $to\_change.push\_back(a)$;
        end
    end
end
return $to\_change$;
```

In the visit function for Assignment Nodes, the top of *num_stack* describes the depth of LLF and the top of *ctx_stack* is the indexVar of LLF.

None (mutually exclusive): If *ctx_stack* except the top contains indexVars in *rhs*, the indexVar of outer forall must be the reduction dimension. Therefore, the operator must be +=.

Equal: If some indexVar in *rhs* has *sibling* in *ctx_stack*, the indexVar of outer forall must be the reduction dimension. Therefore, the operator must be +=.

4.2 Rewriter

Change the AssignmentNodes in *to_change* to "+="

4.3 Code

The code is in Precompute::apply, after the PrecomputeRewriter is applied.