Common Sub-Expression Elimination Example

You might remember in the Introduction section that we mentioned that a compiler could do a much better job of common sub-expression elimination than a programmer when dealing with matrices that had the same dimensions and subscripts. We would like to show that this is true with the following example. Suppose that we have the following statement:

\[ a[i:j] \leftarrow b[i:j] + c[i:j] \]

where a, b, and c have identical dimensions of 30 rows and 20 columns and are real variables. We can achieve significant common sub-expression elimination that is not available to the programmer without totally redesigning the solution. The 4-tuples that would be generated from the above statement are

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

Click here for the answer.
Repeating the 4-tuples we have

1. I$1 \ast i \quad 20
2. I$2 + I$1 j
3. I$3 \ast i \quad 20
4. I$4 + I$3 j
5. R$1 SLD b \quad I$4
6. I$5 \ast i \quad 20
7. I$6 + I$5 j
8. R$2 SLD c \quad I$6
9. R$3 + R$1 R$2
10. a SST R$3 \quad I$2

From these 4-tuples we have the following dag:

Click [here](#) for the DAG.
Repeating the DAG

From the dag we generate the following 4-tuples:

Click [here](#) for the 4-tuples.
Note: Any time that you have common sub-expression elimination, the intermediate results will be used multiple times. This implies that our register allocation algorithm will have to be modified to consider the implication of intermediate results being used multiple times.
Answer for the matrices

1. $I_1 \cdot j = 50$
2. $I_2 + I_1 = i$
3. $I_3 \cdot j = 50$
4. $I_4 + I_3 = i$
5. $R_1 \ SLD \ b = I_4$
6. $I_5 \cdot j = 50$
7. $I_6 + I_5 = i$
8. $R_2 \ SLD \ c = I_6$
9. $R_3 + R_1 + R_2$
10. $a \SST \ R_3 + I_2$

Resulting DAG

![Resulting DAG Diagram]
The resulting 4-tuples are

```
I$1  *   j   50
I$2  +   I$1  i
R$1 SLD b   I$2
R$2 SLD c   I$2
R$3 +   R$1  R$2
a  SST  R$3  I$2
```