1. Review

- The DAG representation of basic blocks
- The data-flow analysis schema
- Reaching definitions
- Control-flow equations for reaching definitions
- Live-variable analysis

2. Common Subexpression Elimination

- Local common subexpression elimination

  In the following BEFORE basic block, the assignments to \texttt{t7} and \texttt{t10} compute the subexpressions \texttt{4 * i} and \texttt{4 * j}, which have been eliminated in the AFTER block by local common subexpression elimination:

\[
\begin{align*}
\text{BEFORE} & \quad \text{AFTER} \\
\texttt{t6 = 4 * i} & \quad \texttt{t6 = 4 * i} \\
\texttt{x = a[t6]} & \quad \texttt{x = a[t6]} \\
\texttt{t7 = 4 * i} & \quad \texttt{t8 = 4 * j} \\
\texttt{t8 = 4 * j} & \quad \texttt{t9 = a[t8]} \\
\texttt{t9 = a[t8]} & \quad \texttt{a[t7] = t9} \\
\texttt{a[t7] = t9} & \quad \texttt{a[t6] = t9} \\
\texttt{t10 = 4 * j} & \quad \texttt{t8 = 4 * j} \\
\texttt{a[t10] = x} & \quad \texttt{a[t8] = x} \\
\texttt{goto B2} & \quad \texttt{goto B2}
\end{align*}
\]
- Global common subexpression elimination
  - In the following flow graph, block B5 computes the common subexpressions $4 \times i$ and $4 \times j$, which are computed in blocks B2 and B3, respectively.
o Notice that block B5 can be replaced by the following block since block B2 has computed 4*i into t2 and a[t2] into t3:

\[
\begin{align*}
x &= t3 \\
t8 &= 4 \times j \\
t[9] &= a[t8] \\
a[t2] &= t9 \\
a[t8] &= x \\
goto B2
\end{align*}
\]

o This block can be replaced by following block by noticing that block B3 has computed 4*j into t4 and a[t4] into t5:

\[
\begin{align*}
x &= t3 \\
t[9] &= a[t4] \\
a[t2] &= t9 \\
a[t4] &= x \\
goto B2
\end{align*}
\]

o We now notice that block B3 has already computed a[t4] into t5 so we can replace the second of third statements by the assignment a[t2] = t5 to obtain the following optimized block:

\[
\begin{align*}
x &= t3 \\
a[t2] &= t5 \\
a[t4] &= x \\
goto B2
\end{align*}
\]

So far we have reduced the original nine-statement block B5 into a four-statement block.

3. Copy Propagation

- A three-address statement of the form \( u = v \) is called a copy statement, or copy for short.
- We can introduce copy statements to avoid recomputing common subexpressions:
Copies introduced during common subexpression elimination

4. Dead-Code Elimination

- Statements that compute values that never get subsequently used can be eliminated.
- Often copy propagation turns copy statements into dead code.
- Consider the reduced basic block for B5:

\[
\begin{align*}
x &= t_3 \\
a[t_2] &= t_5 \\
a[t_4] &= x \\
goto B2
\end{align*}
\]

After copy propagation this block becomes:

\[
\begin{align*}
x &= t_3 \\
a[t_2] &= t_5 \\
a[t_4] &= t_3 \\
goto B2
\end{align*}
\]

We now observe \( x \) is never used so the first statement can be eliminated. The block now becomes

\[
\begin{align*}
a[t_2] &= t_5 \\
a[t_4] &= t_3 \\
goto B2
\end{align*}
\]

5. Code Motion

- Loop-invariant computations are best moved outside loops.
- Consider the while-statement:
while (i <= limit – 2)
Code motion will produce a faster equivalent loop when the limit computation is performed once before entering the loop:

    t = limit – 2
    while (i <= t)

6. Induction Variables and Reduction in Strength

• A variable $x$ is an \textit{induction variable} if its value always changes by a constant whenever it is assigned a new value.

  o For example, $i$ and $t_2$ are induction variables in block $B_2$ of the flow graph in Section 2 above.

• \textit{Strength reduction} is the replacement of an expensive operation such as multiplication by a cheaper one such as addition.

• Reduction in strength and induction-variable elimination can be used to speed up loops. See ALSU, Figs. 9.8 – 9.10, pp. 592-595 for an extended example.

7. Reading

• ALSU, Section 9.1

aho@cs.columbia.edu