COMP 515: Advanced Compilation for Vector and Parallel Processors

Prof. Vivek Sarkar
Department of Computer Science
Rice University
vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP515

Homework #3 (REMINDER)

1. **Solve exercise 5.6 in book**
   - Your solution should be legal for all values of K (note that the value of K is invariant in loop I)

Exercise 5.6: What vector code should be generated for the following loop?

```
DO I = 1, 100
    A(I) = B(K) + C(I)
    B(I+1) = A(I) + D(I)
END DO
```

- **Due on Oct 8th**

Coarse-Grain Parallelism (contd)

Chapter 6 of Allen and Kennedy

- **Acknowledgment: Slides from previous offerings of COMP 515 by Prof. Ken Kennedy**
  - **http://www.cs.rice.edu/~ken/comp515/**

Scalar Privatization (Recap)

- The analog of scalar expansion is privatization.
- Temporaries can be given separate namespaces for each iteration.

```
    DO I = 1,N
S1     T = A(I)
S2     A(I) = B(I)
S3     B(I) = T
    ENDDO
```

```
    PARALLEL DO I = 1,N
      PRIVATE t
S1     t = A(I)
S2     A(I) = B(I)
S3     B(I) = t
    ENDDO
```

Array Privatization

- Array variables can be privatized as well (but the underlying analysis can be more complicated than for scalars)

```
    DO I = 1,100
S0      T(1)=X
L1      DO J = 2,N
S1        T(J) = T(J-1)+B(I,J)
S2        A(I,J) = T(J)
        ENDDO
    ENDDO
```

⇨

```
  PARALLEL DO I = 1,100
      PRIVATE t(N)
S0    t(1) = X
L1    DO J = 2,N
S1       t(J) = t(J-1)+B(I,J)
S2       A(I,J)=t(J)
      ENDDO
   ENDDO
```

Loop Distribution

- As we saw in Chapter 5, loop distribution can convert loop-carried dependences to loop-independent dependences.
- Consequently, it often creates opportunity for outer-loop parallelism.
- However, we must add extra barriers to keep distributed loops from executing out of order, so the overhead may override the parallel savings.

Loop Alignment

- Many carried dependencies are due to array alignment issues.
- If we can align all references, then dependencies would *go* away, and loop-level parallelism can be exposed.
- This is also related to Software Pipelining

```
DO I = 2,N
   A(I) = B(I)+C(I)
   D(I) = A(I-1)*2.0
ENDDO
```

→

```
DO I = 1,N   ! Aligned loop
   IF (I .GT. 1) A(I) = B(I)+C(I)
   IF (I .LT. N) D(I+1) = A(I)*2.0
ENDDO
```

→

```
D(2) = A(1)*2.0
DO I = 2,N-1   ! Pipelined loop
   A(I) = B(I)+C(I)
   D(I+1) = A(I)*2.0
ENDDO
A(N) = B(N)+C(N)
```

Code Replication

- **If two dependences between the same statements have different dependence distances, then alignment doesn't help.**
- **We can fix the second case by replicating code:**

```
DO I = 1,N
   A(I+1) = B(I)+C
   X(I) = A(I+1)+A(I)
ENDDO
```

```
DO I = 1,N
   A(I+1) = B(I)+C
   ! Replicated Statement
   IF (I .EQ 1) THEN
      t = A(I)
   ELSE
      t = B(I-1)+C
   END IF
   X(I) = A(I+1)+t
ENDDO
```

Strip Mining

- Converts available parallelism into a form more suitable for the hardware
- Assume THRESHOLD = minimum iters for parallel loop (due to overhead reasons)

```
        DO I = 1, N
            A(I) = A(I) + B(I)
        ENDDO
==>
        k = MAX(THRESHOLD, CEIL (N / P))
        PARALLEL DO I = 1, N, k
                DO i = I, MIN(I + k-1, N)
                        A(i) = A(i) + B(i)
                ENDDO
        END PARALLEL DO
```

Loop Fusion

- Loop distribution was a method for separating parallel parts of a loop.
- Our solution in Section 5 attempted to find the maximal loop distribution.
- The maximal distribution often finds parallelizable components too small for efficient coarse-grain parallelism.
- Two obvious solutions:
  - Strip mine large loops to create larger granularity (with an outer parallel loop and inner sequential loop)
  - Perform maximal distribution, and then fuse together parallelizable loops.
  - Both solutions can be combined as well.

Fusion Safety: Fusion-Preventing Loop-Independent Dependences

Definition: A loop-independent dependence between statements S1 and S2 in loops L1 and L2 respectively is fusion-preventing if fusing L1 and L2 causes the dependence to be carried by the combined loop in the opposite direction.

Example of an illegal loop fusion:

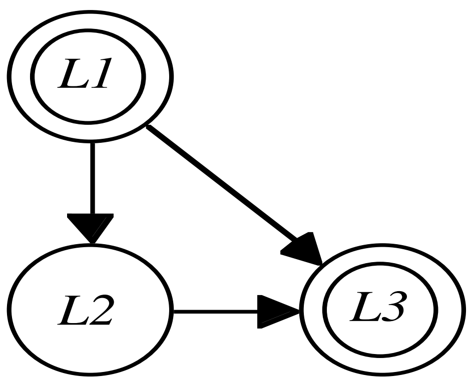
```
    DO I = 1,N
S1      A(I) = B(I)+C
    ENDDO
    DO I = 1,N
S2      D(I) = A(I+1)+E
    ENDDO
```

⇨

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

Fusion Safety: Ordering Constraint

- **We shouldn't fuse loops if the fusion will result in an illegal ordering of the dependence graph.**
- **Ordering Constraint: Two loops can't be legally fused if there exists a path of loop-independent dependencies between them containing a loop or statement not being fused with them i.e., if fusion will result in a cycle in the resulting loop-independent dependences**



Fusing L1 with L3 violates the ordering constraint. {L1,L3} must occur both before and after the node L2, which is not possible.

Fusion Profitability

Parallel loops should generally not be merged with sequential loops.

Definition: An edge between two statements in loops L1 and L2 respectively is said to be parallelism-inhibiting if after merging L1 and L2, the dependence is carried by the combined loop.

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

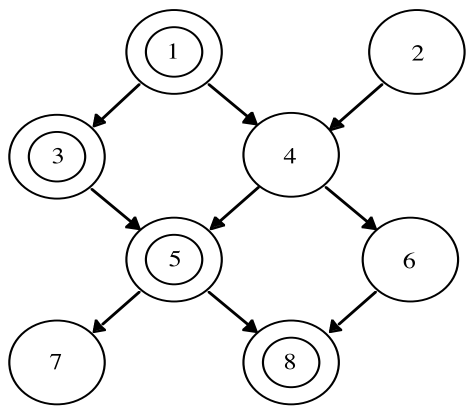
```
      DO I = 1,N
S1       A(I+1) = B(I) + C
S2       D(I) = A(I) + E
      ENDDO
```

Typed Fusion

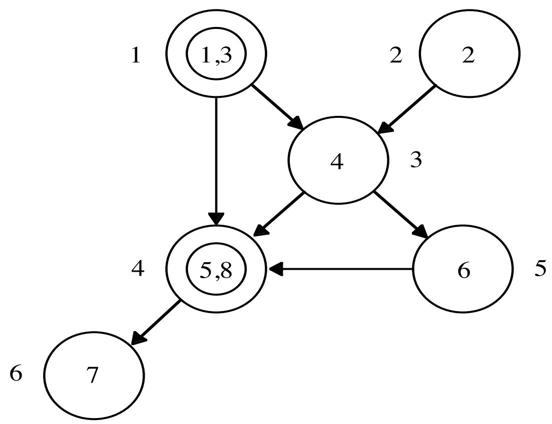
- We start by classifying loops into two types: parallel and sequential.
- We next gather together all edges that inhibit efficient fusion, (i.e., that connect a sequential and a parallel loops) and call them "bad edges".
- Given a graph of loop-independent dependences (V,E), we want to obtain a graph (V',E') by merging vertices of V subject to the following constraints:
  – Bad Edge Constraint: vertices joined by a bad edge aren't fused.
  – Ordering Constraint: vertices joined by path containing non-parallel vertex aren't fused

Typed Fusion Example

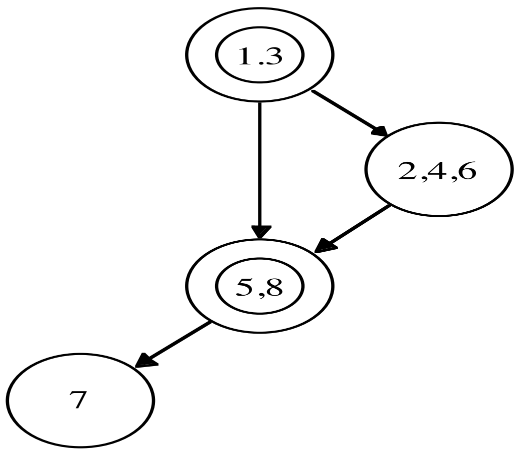
Original loop graph



After fusing parallel loops



After fusing sequential loops



Optimal Weighted Loop Fusion for Parallel Programs

Nimrod Megiddo

IBM Almaden Research Center

Vivek Sarkar

MIT Laboratory for Computer Science

Proceedings of the 1997 ACM Symposium on Parallel Algorithms and Architectures (SPAA), Pages 282-291

Motivation

- Loop fusion is an important program transformation for scientific applications and stream-based applications

- The weighted loop fusion optimization problem is NP-hard

- Approximation solutions are hard because weights can be arbitrary and the program graph has no special structure

- Problem sizes are not very large in practice ($\leq$ 100 nodes)

- Computers are 1000$\times$ faster now compared to when NP-hardness was introduced

$\Rightarrow$ Our goal is to find optimal solutions in a tractable way

Loop Dependence Graph (LDG)

- The LDG is a directed **acyclic** dependence graph (built for a region of acyclic control flow across loop nests).

- **Node** = loop nest.

- **Edge** = data dependence from source node to destination node.

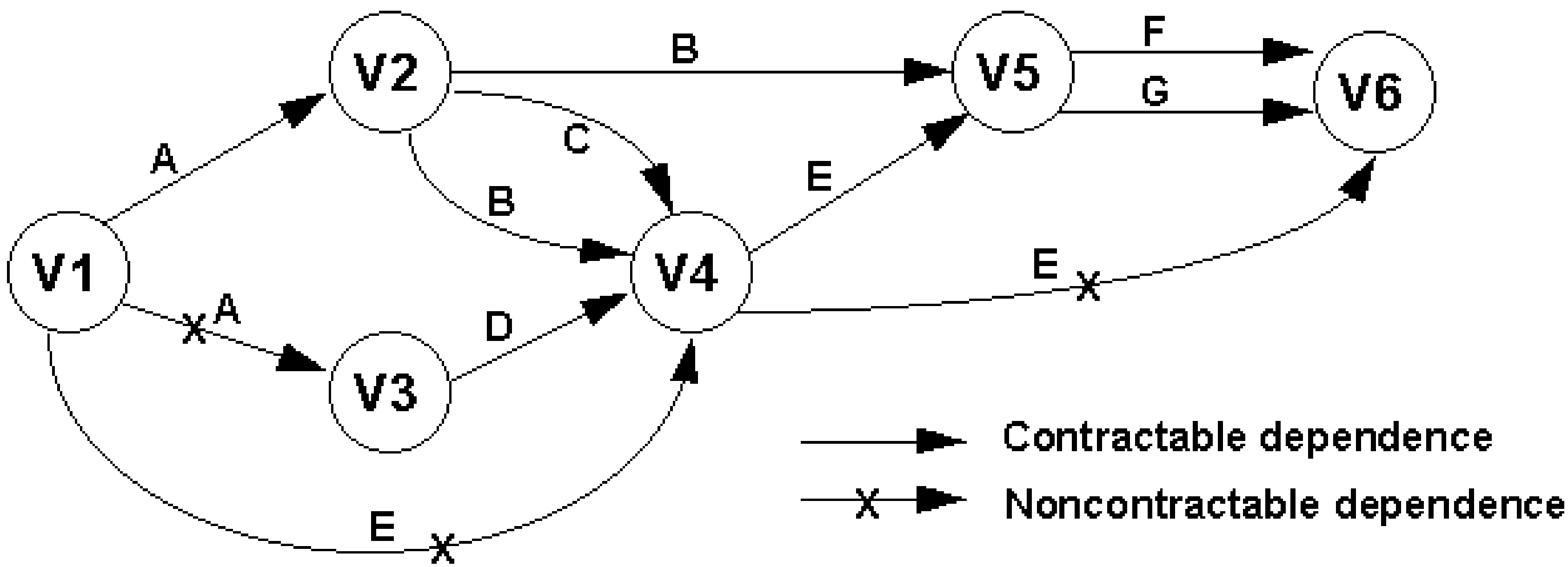
- Each LDG edge is marked as **contractable** or **noncontractable**.

  The source and destination loop nests of a noncontractable edge cannot be fused (otherwise a data dependence will be violated or a parallel loop will be serialized).

Example program

```
      PARDO 10 I = 1, N
10        A(I) =  E(I-1)
      PARDO 20 I = 1, N
          B(I) = A(I)*2 + 3
20        C(I) = B(I) + 99
      PARDO 30 I = 1, N
30        D(I) = A(N-I+1) + A(I)
      PARDO 40 I = 1, N
40        E(I) = B(I) + C(I) * D(I)
      PARDO 50 I = 1, N
          F(I) = B(I)*4 + 2
50        G(I) = E(I)*8 - F(I)
      PARDO 60 I = 1, N
60        H(I) = F(I) + G(I) * E(I-1)
```

LDG for example program



Fusion Partition

Fusion partition = partition of LDG nodes into disjoint fusion clusters

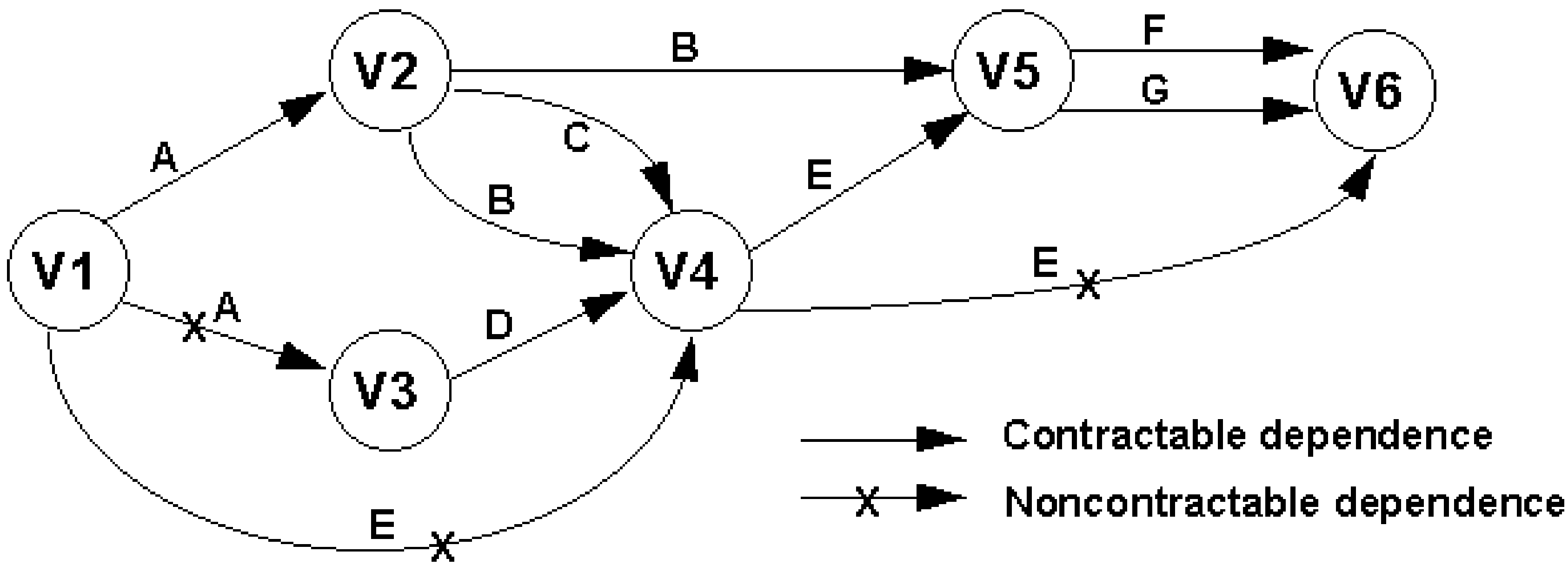
Fusion cluster = set of loop nests to be fused together

A fusion partition is **legal** if and only if:

1. The source and destination nodes of each **noncontractable** edge are placed in distinct clusters, and

2. The reduced graph is acyclic

Examples of Fusion Partitions



Cost of a Fusion Partition

$$\begin{aligned}\text{Define } w_{ij} &= \textbf{weight} \text{ of pair of nodes, } i \text{ and } j \\ &= \text{cost savings obtained by fusing nodes } i \text{ and } j\end{aligned}$$

$\Rightarrow$ cost of fusion partion P is given by

$$F(P) = \sum_{P(i) \neq P(j)} w_{ij}$$

where $P(i)$ = cluster number for node i in fusion partition P.

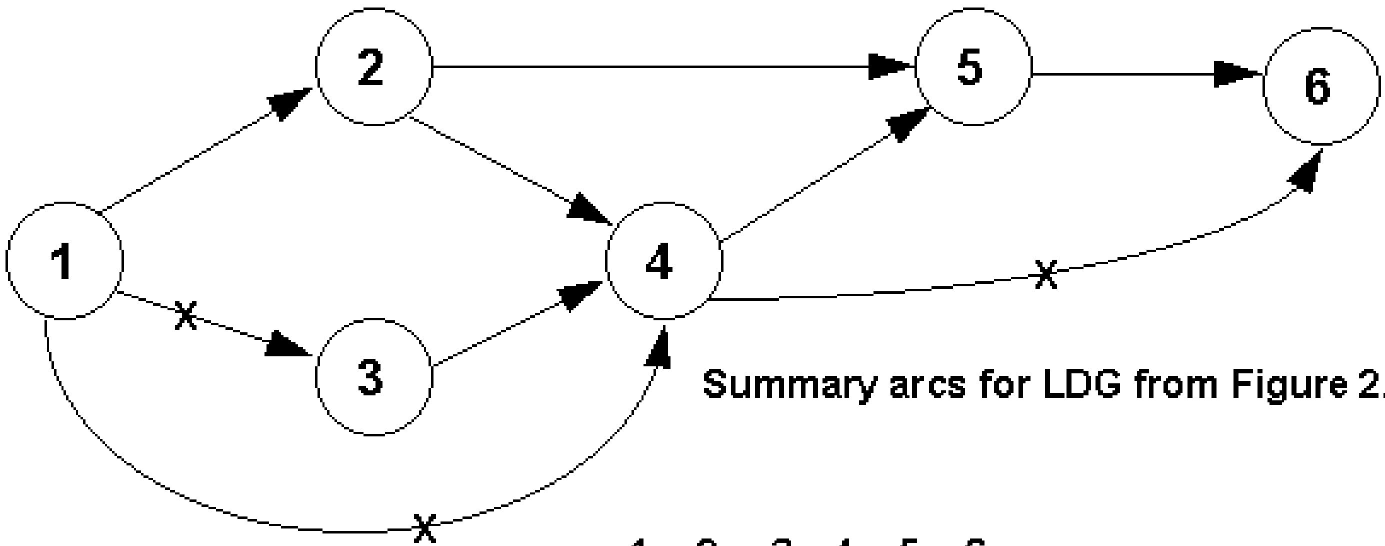
Cost of fusion partition = sum of inter-cluster w_{ij} weights

Computing Weights

Examples of computing weights in different applications of weighted loop fusion:

- **Common Loads:** set w_{ij} = number of common values in load instructions in loop nests i and j
- **Cache Locality:** set w_{ij} = number of common cache lines accessed by loop nests i and j
- **Remote data accesses:** set w_{ij} = number (and size) of common remote data access in loop nests i and j

Example of Cost of a Fusion Partition



Summary arcs for LDG from Figure 2.

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	0	1	2	1	0
3	0	1	0	1	0	0
4	0	2	1	0	2	0
5	0	1	0	2	0	2
6	0	0	0	0	2	0

Table of weights, $w_{ij} = w_{ji}$

Problem Statement for Optimal Weighted Loop Fusion

Given an LDG and weights for pairs of nodes, find a legal fusion partition P with minimum cost, $F(P)$.

The optimized fusion configuration is obtained by fusing all loops belonging to the same cluster, and generating the fused loops in a topological sort order of the reduced LDG.

A Simple Integer Programming Formulation

- Introduce (0,1)-variables x_{ij} such that $x_{ij} = 0$ means that LDG nodes i and j are placed in the same cluster

- **Contractability constraint:** $x_{ij} = 1$ if there is a noncontractable LDG edge from node i to node j

- **Transitivity constraint:** $x_{ik} \leq x_{ij} + x_{jk}$

- Introduce integer variables π_i such that π_i represents a topologically sorted cluster numbering of LDG nodes

- **Equivalence constraint:** If $x_{ij} = 0$ then $\pi_i = \pi_j$. This constraint can be rewritten as $-n \cdot x_{ij} \leq \pi_j - \pi_i \leq n \cdot x_{ij}$.

- **Acyclicity constraint:** If there is an LDG edge from i to j, then $x_{ij} \leq \pi_j - \pi_i$

A Simple Integer Programming Formulation (contd.)

$$
\begin{array}{lrll}
\underset{x,\pi}{\text{Minimize}} & \displaystyle\sum_{(i,j)\in B\cup C} w_{ij}x_{ij} & & \\
\text{subject to} & x_{ik} \le x_{ij} + x_{jk} & \forall \text{ nodes } i,j,k \\
& x_{ij} \le \pi_j - \pi_i & \forall \text{ arcs } (i,j) \\
& -n\cdot x_{ij} \le \pi_j - \pi_i & \forall \text{ nodes } i,j \\
& \pi_j - \pi_i \le n\cdot x_{ij} & \forall \text{ nodes } i,j \\
& x_{ij} = 1 & \forall \text{ noncontractable arcs } (i,j) \\
& x_{ij} \in \{0,1\} & \forall \text{ nodes } i,j
\end{array}
$$

where $C \subseteq A$ is the set of *contractable* arcs and B is the set of unordered node pairs with nonzero weights

This formulation has $O(|N|^2)$ **variables and** $O(|N|^3|)$ **constraints**

More Efficient Formulation

Observations:

- The $x_{ik} \leq x_{ij} + x_{jk}$ triangular inequalities can be dropped without changing the set of feasible solutions.

- We only need to maintain x_{ij} variables for $(i, j) \in B \cup C$

- We only need the $x_{ij} \leq \pi_j - \pi_i$ inequalities for contractable arcs i.e., for $(i, j) \in C$

- We only need the $-n \cdot x_{ij} \leq \pi_j - \pi_i$ inequalities for unordered node pairs with nonzero weights i.e., for $\{i, j\} \in B$

More Efficient Formulation (contd.)

$$
\begin{aligned}
&\underset{x,\pi}{\text{Minimize}} \quad \sum_{(i,j)\in B\cup C} w_{ij}x_{ij} \\
&\text{subject to} \quad x_{ij} \le \pi_j - \pi_i \le n \cdot x_{ij} && \forall \text{ contractable arcs } (i,j) \\
&\qquad -n \cdot x_{ij} \le \pi_j - \pi_i \le n \cdot x_{ij} && \forall \text{ unordered node pairs } (i,j) \\
&&& \text{with nonzero weight} \\
&\qquad \pi_j - \pi_i \ge 1 && \forall \text{ noncontractable arcs } (i,j) \\
&\qquad x_{ij} \in \{0,1\} && ((i,j) \in B \cup C) \ .
\end{aligned}
$$

This formulation has $(|N| + |B| + |C|)$ **variables and** $(2|C| + 2|B| + |NC|)$ **constraints.**

Example of More Efficient Formulation

$$\underset{x,\pi}{\text{Minimize}} \quad x_{23} + (x_{12} + 2x_{24} + x_{25} + x_{34} + 2x_{45} + 2x_{56})$$

subject to

$$\begin{aligned}
x_{12} &\le \pi_2 - \pi_1 \le 6 \cdot x_{12} \\
x_{24} &\le \pi_4 - \pi_2 \le 6 \cdot x_{24} \\
x_{25} &\le \pi_5 - \pi_2 \le 6 \cdot x_{25} \\
x_{34} &\le \pi_4 - \pi_3 \le 6 \cdot x_{34} \\
x_{45} &\le \pi_5 - \pi_4 \le 6 \cdot x_{45} \\
x_{56} &\le \pi_6 - \pi_5 \le 6 \cdot x_{56} \\
-6 \cdot x_{23} &\le \pi_3 - \pi_2 \le 6 \cdot x_{23}
\end{aligned}$$

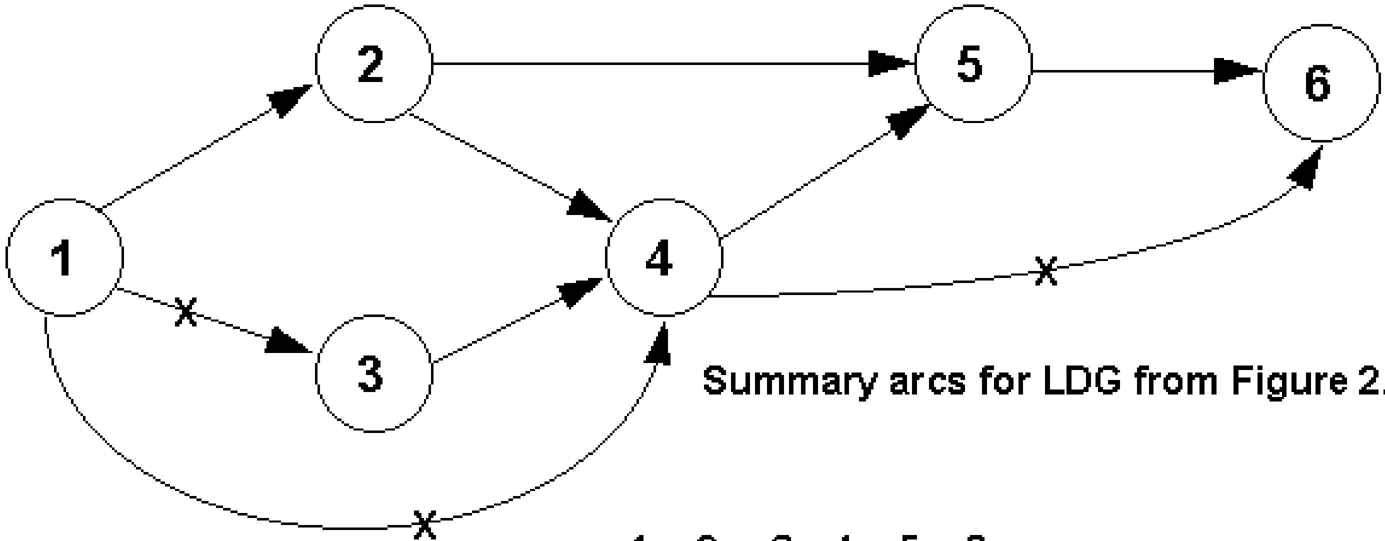
$$\begin{aligned}
\pi_3 - \pi_1 &\ge 1 \\
\pi_4 - \pi_1 &\ge 1 \\
\pi_6 - \pi_4 &\ge 1
\end{aligned}$$

Efficient formulation has 12 variables and 17 constraints

(Simple formulation has 18 variables and 99 constraints)

Optimal Weighted Loop Fusion solution for Example

$P = 1 \mid 2, 3, 4, 5 \mid 6$ (cost = 3)



Summary arcs for LDG from Figure 2.

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	0	1	2	1	0
3	0	1	0	1	0	0
4	0	2	1	0	2	0
5	0	1	0	2	0	2
6	0	0	0	0	2	0

Table of weights, $w_{ij} = w_{ji}$

Optimal Weighted Loop Fusion solution for Example (contd.)

```
      PARDO 10 I = 1, N
10        A(I) =  E(I-1)

      PARDO 20 I = 1, N
          B(I) = A(I)*2 + 3
          C(I) = B(I) + 99
          D(I) = A(N-I+1) + A(I)
          E(I) = B(I) + C(I) * D(I)
          F(I) = B(I)*4 + 2
50        G(I) = E(I)*8 - F(I)

      PARDO 60 I = 1, N
60        H(I) = F(I) + G(I) * E(I-1)
```

Preliminary Experimental Results

Source of LDG	n	$\|B\|$	$\|A\|$	$\|C\|$	Total # iters	Time for total # iters (OSL)
Example	6	1	9	6	15	0.080 seconds
034.mdljdp2	12	0	12	9	19	0.050 seconds
Synthetic	100	0	90	80	60	0.140 seconds

Execution times for solving optimal weighted loop fusion problems using the IBM Optimization Subroutine Library on a 33MHz RS/6000 model 220 workstation.

Extensions

- **Cost term proportional to no. of clusters** — extend objective function to

$$F(P) = \sum_{P(i) \neq P(j)} w_{ij} + \mathrm{S} \times \text{(no. of clusters)}$$

  where S is the synchronization cost incurred per cluster

- **Conformability classes** — equivalence classes such that only nodes from within the same class are allowed to be fused.

  Conformability classes can be used to model non-loop statements, loops with nonconformable bounds, loops with premature exits, etc.

Extensions (contd.)

- **Hierarchical fusion** — apply algorithm recursively on LDG for the body of each fused loop.

- **Control dependences** — extend LDG to be an acyclic PDG. Each control dependence edge is noncontractable.

- **Branch-and-bound method** — compute bounds using linear programming relaxation described in paper.

  Branch-and-bound method automatically stores the best feasible solution seen till current point in time, and can be more efficient than using an optimization library.

Related Work

- **[Allen & Cocke '72]** Introduced loop fusion transformation.
- **[Goldberg & Paige '84]** Showed how loop fusion can be used to optimize stream processing in database queries.
- **[Callahan '87]** Greedy merge algorithm for *unweighted* loop fusion (minimizing the number of clusters).
- **[Gao et al '92]** Heuristic solution to weighted loop fusion using repeated applications of max-flow min-cut algorithm
- **[Kennedy & McKinley '93]** NP-hardness proof for weighted loop fusion. Experimental results show 4–17% improvement in uniprocessor execution times with heuristic algorithm.

Conclusions

- Presented an integer programming formulation for weighted loop fusion

- Size of formulation is linear in size of LDG and weights

- Preliminary execution time measurements show that optimal weighted loop fusion is tractable to solve in practice

Future Work

- Extend prototype implementation by making calls to optimization subroutine library from within the compiler
- Compare performance of heuristic and optimal solutions on Fortran 90 programs for SMPs
- Extend weighted loop fusion model by adding capacity constraints
- Extend branch-and-bound algorithm with incremental recomputation of edge weights
- Investigate development of tractable optimal algorithms for other NP-hard problems in compiler optimization