We consider applications that require heavy manipulation of array elements. Usually, \textit{Scientific} or \textit{Engineering} applications. The question is whether their performance can be significantly enhanced by using massively parallel architectures (multi-processors).

There are two categories of code transformations: \textit{Amdahl's Law} and \textit{loop-level parallelism}.

\textbf{Amdahl's Law}

\[ \frac{\text{speedup}}{\text{speed}} = \frac{1}{f + (1 - f) \cdot \frac{p}{d}} \]

where \( d \) is the number of available processors, \( f \) is the fraction of the code (time-wise) that can be parallelized, and \( p \) is the number of processors.

Better than \textit{task-level parallelism}

\[ \left\{ \begin{array}{l} \forall i \exists! Z[i] = X[i] \wedge Y[i] \\ \forall i \exists! Z[i] \wedge Z[i] = Z[i] \wedge Z[i] \end{array} \right\} \]

\textbf{loop-level parallelism}

Loops are the most desirable targets of optimization.
Locality comes in two flavors: 

- **Data locality**: 
  - Processor space — set of processors in the system.
  - Data space — set of array elements accessed.
  - Iteration space — vectors of values assumed by the loop indices.

- **Temporal locality**: 
  - Accesses to spatially close locations are allocated in the same processor time on the same processor.
  - Multiple accesses to the same location are close in time on the same processor.

- **Spatial locality**: 
  - Accesses to spatially close locations are allocated to the same processor.
  - Multiple accesses to spatially close locations are allocated to the same processor.

---

### Additional Locality Enhancing Transformations

- **Loop Fusion**: This is a loop fusion transformation.

  $\forall i \in [0..10]$ do $[i]Z = [i]Z$ + $[i+1]Z$

- **Data Dependence Analysis**: For the example: there is no data dependence across iterations.

  $[i]Z :=[i]Z + [i+1]Z$
Parallelization

If we have \( p \) processors, we may partition the rows among them. But then the overall cost becomes \( \frac{cn}{p} + n^3 = \frac{cn}{p} + \frac{n^3}{p} \), as we approach \( p \) processors. If we have \( d \) processors, we may partition the rows among them. But the picture becomes more complex due to cache interference.

The block size \( B \) is chosen by the loss in communication speed.

The number of cache misses reduces to \( \frac{cn}{p} \), and then if we partition the matrix into blocks of size \( B \times \frac{n}{d} \), the gain in computation speed becomes \( \frac{cn}{p} \) as we approach \( p \) processors. The second expression gets close to \( \frac{cn}{p} \), so the gain in computation speed is offset by the loss in communication speed.

Iteration Spaces

An example of an Iteration Spaces

```plaintext
0 = \lfloor 2 \times 3 \rfloor Z(0,0,0) + 0
```

Representing the set of inequalities

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix} \leq \begin{bmatrix}
2 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]

Solutions of the induced Iteration Space can be characterized as the set of

For example, consider the program

Controlling Order of Execution

The inequality characterization of the iterationspace does not determine the order in which these iterations are executed.

The original program specifies a particular ordering which is consistent with the original access order (p) presents a violation only. We present various ordering of the program consistent with lexicographical visits to the iteration points.

In Fig. 11.14 we present various ordering of the program.

In general, the Iteration Space can be characterized as the set

\[
0 = \lfloor 2 \times 3 \rfloor Z(0,0,0) + 0
\]

where

\[
\{0 \leq q + 2g \mid pZ \geq g\}
\]

For example, if the code is equivalent to

\[
0 = \lfloor 2 \times 3 \rfloor Z(0,0,0) + 0
\]

The Iteration Space is not unique. For example, the code

\[
0 = \lfloor 2 \times 3 \rfloor Z(0,0,0) + 0
\]

Every nested loop structure defines a set of index vectors. The Iteration Space

Parallelization

A. Pnueli

Honors Compilers, NYU, Spring, 2007
Alg. 11.11

Fourier-Motzkin Elimination

Input: A polyhedron \( S \) over variables \( x_1, \ldots, x_n \).

Output: A polyhedron \( S_0 \) over \( x_1, \ldots, x_n \) obtained by projecting \( x_n \) away.

Method

1. For every pair of lower and upper bounds on \( x_n \), obtain

\[ \begin{align*}
L & \leq f \leq U \\
0 & \leq f \\
\end{align*} \]

Eliminate \( f \) from input

Example

Input: \( \{ f \} : \Omega \) \( \{ f \} : \Omega \) \( \{ f \} : \Omega \) \( \{ f \} : \Omega \)

Output: \( \{ f \} : \Omega \) \( \{ f \} : \Omega \) \( \{ f \} : \Omega \) \( \{ f \} : \Omega \)

Example

Alg. 11.13 Loop-Bounds Generation

Input: A convex polyhedron \( S \) over variables \( v_1, \ldots, v_n \).

Output: A set of lower bounds \( L_i \) and upper bounds \( U_i \) for each \( v_i \) expressed only in terms of \( v_j \)'s for \( j < i \).

\[
S_n := S; /* Use Fourier-Motzkin to find the bounds */
\]

for \( i \in [n] \) do

1. Let \( C \) be all the constraints in \( S \) involving \( x_n \).

2. For every pair of lower and upper bounds on \( x_n \), obtain

\[ \begin{align*}
L & \leq f \leq U \\
0 & \leq f \\
\end{align*} \]

Eliminate \( f \) from \( C \) and \( S \)

3. Remove any bounds in \( L_{v_i} \) and \( U_{v_i} \) implied by \( \{ f \} : \Omega \)

4. Otherwise, \( S \) consists of \( S - C \). Plus all the constraints generated in steps 1 and 2.

Example

Alg. 1.13

Output: A set of lower bounds \( L_i \) and upper bounds \( U_i \) for each \( v_i \) expressed only in terms of \( v_j \)'s for \( j < i \).

Input: A polyhedron \( S \) over variables \( v_i \).
Changing Axes

So far, we considered reordering iterations which only concerned the order by which the various indices changed. However, we can also consider coordinate changes by arbitrary affine transformations to the polyhedron. For example, due to the affine transformation such as $z \rightarrow 0$ and $f \rightarrow f_0$, we obtain the program such as $0 = [f \cdot z + u] \odot [u \cdot 0] \Rightarrow 0 \leq f \Rightarrow z + u \cdot z = f$.

Matrix Multiplication

In many cases, it is possible to transform a non-affine program into an affine program such as for $f \in \mathbb{Z}$ with $g \in \mathbb{Z}$ the program which leads to the program

$L = 0; \{ f : \forall \mathbf{i}; f; \forall \mathbf{j} \}$

$\forall \mathbf{i}; f; \forall \mathbf{j} \}$

$0 = f \cdot z + u \cdot f_0 \Rightarrow 0 \leq f \Rightarrow z + u \cdot f_0 = f$.
Lecture 16: Parallelism and Locality

A.Pnueli

Self Reuse

Consider a loop nest of depth $d$ which accesses data of dimension $r < d$. Obviously, the same data must be referenced repeatedly.

There exist $\bar{x} \in \mathbb{Z}^d$ and $\bar{x} \in \mathbb{Z}^d$ such that:

At least one of them is a write.

These accesses are data dependent if $\langle q', r', i', j' \rangle = \mathcal{A}$ and $\langle q', r', i', j' \rangle = \mathcal{A}$.

Consider two static accesses to the same array in possibly different loops, associated with $\langle q, r, i, j \rangle = \mathcal{A}$ and $\langle q', r', i', j' \rangle = \mathcal{A}$.

It is the case that a single static reference of the form $\mathcal{A} + i \mathcal{B} = \mathcal{A} + j \mathcal{B}$.

In particular, we may ask the question of dependence between a

Spatial Self Reuse

This is the case that a single static reference of the form $\mathcal{A} + i \mathcal{B} = \mathcal{A} + j \mathcal{B}$.

Consider a loop nest of depth $d$ which accesses data of dimension $r < d$. Obviously, the same data must be referenced repeatedly.

We should apply an affine transformation to the iteration space, such that the transformed affine access function should depend only on the first $r$ indices of the loop nest.

The dimensionality of the data accessed is determined by the rank of the matrix $\mathcal{A}$ in the affine access function, which guarantees closest proximity (columns in C, rows in Fortran) will depend on the first $r$. In particular, we may ask the question of dependence between a

Group Reuse

Thus, we will transform the C-loop index.

Array Data-Dependence Analysis

$\mathcal{D} \neq \mathcal{I}$. In particular, the question of dependence between a

For example, the program

For example, in the third example of Fig. 11.9, we should reorganize the loop to be of the form

We consider group reuse for the case that the static references are
Consider the loop
\[
\text{for } i \in [1..10] \text{ do}
\]
\[
Z[i] = Z[i-1]
\]
Analyzing dependences, we obtain:
\[
[t - 1]Z = [t]Z
\]
Consider the loop
Consider the following set of inequalities:

In Fig. 11.21, we present the graph for the inequality set obtained by replacing the inequalities by their tightest linear combinations. If the process of constraint tightening terminates, the problem is consistent. Otherwise, when the process of constraint tightening continues, it is possible to take any value of the variables, and the problem is inconsistent.

Example:

Let's consider the inequalities:

\[ 2a + 3b \leq 7 \]
\[ 2a + 4b \leq 10 \]
\[ 1a \leq z_a \]
\[ 1b \leq z_b \]
\[ 1c \leq z_c \]
\[ Ia \leq z_d \]
\[ Ic \leq z_e \]
\[ Ia \leq z_f \]
\[ Ic \leq z_g \]
\[ Ia \leq z_h \]
\[ Ic \leq z_i \]
\[ Ia \leq 1 \]
\[ Ic \leq 2 \]
\[ Ia \leq 3 \]
\[ Ic \leq 4 \]
\[ Ia \leq 5 \]
\[ Ic \leq 6 \]
\[ Ia \leq 7 \]
\[ Ic \leq 8 \]
\[ 1b \leq 9 \]
\[ 1c \leq 10 \]

Consequently, we can take \( a = 1 \) and \( c = 2 \) and obtain:

Variable \( a \) is lower-bounded by 1, and \( c \) is upper-bounded by 4.

\[ 1a \leq 9 \]
\[ 1c \leq 10 \]

In the tightening process terminates, if possible to read a solution from the graph. The tightening process terminates, if possible to read a solution from the graph. Consequently, we can take \( a = 1 \) and \( c = 2 \) and obtain:

Variable \( a \) is lower-bounded by 1, and \( c \) is upper-bounded by 4.