PART III:

Placement, Communication & Compilation

Outline

- Systolic Synthesis (history)
  - From (i) Single URE, (ii) System of UREs, (iii) ARE
  - Allocation Functions & Unimodular Completion
  - Uniformization
- Lessons from History: Localization, Alignement, Communication
- Compilation of Alpha
  - Naive approach
  - Using the Schedule
  - Allocation and Memory reuse

Systolic Synthesis from a Single URE

- Choose a (one-dimensional) linear schedule, $t(z) = \tau^T z$
- Choose a linear ($n-1$ dimensional) allocation function $a(z)$
- Use $t(z)$ and $a(z)$ to perform a change of basis
- In the new URE, interpret the indices as time and space (processor id's)

Allocation Functions

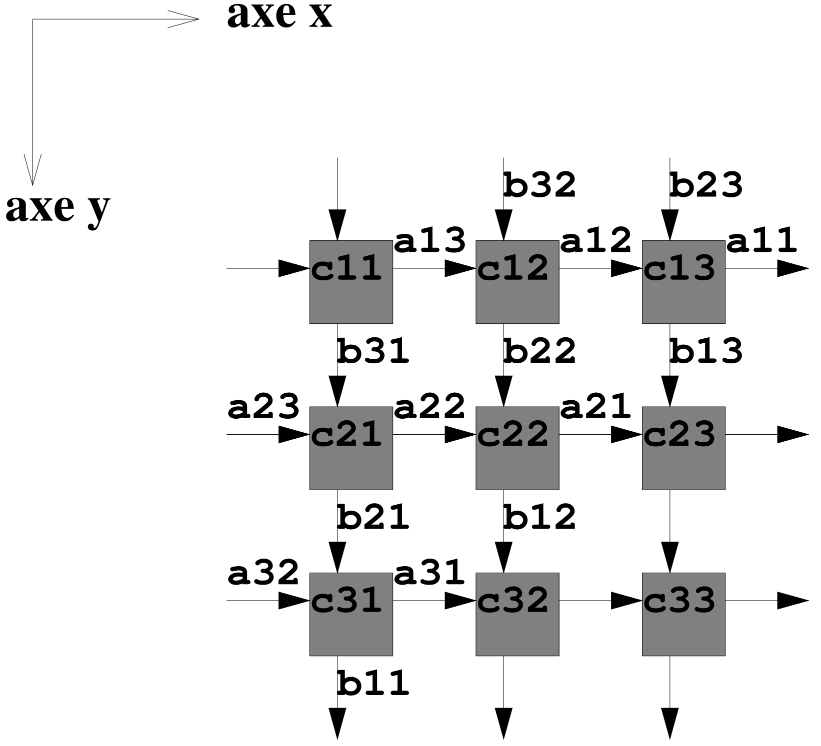
- $a(z) = Pz$, where P is an $n-1 \times n$ matrix
- Specified by a single vector, s (the projection vector)

Matrix Multiplication Example

Schedule : $t(i,j,k) = i + j + k$

Allocation direction : k axis.

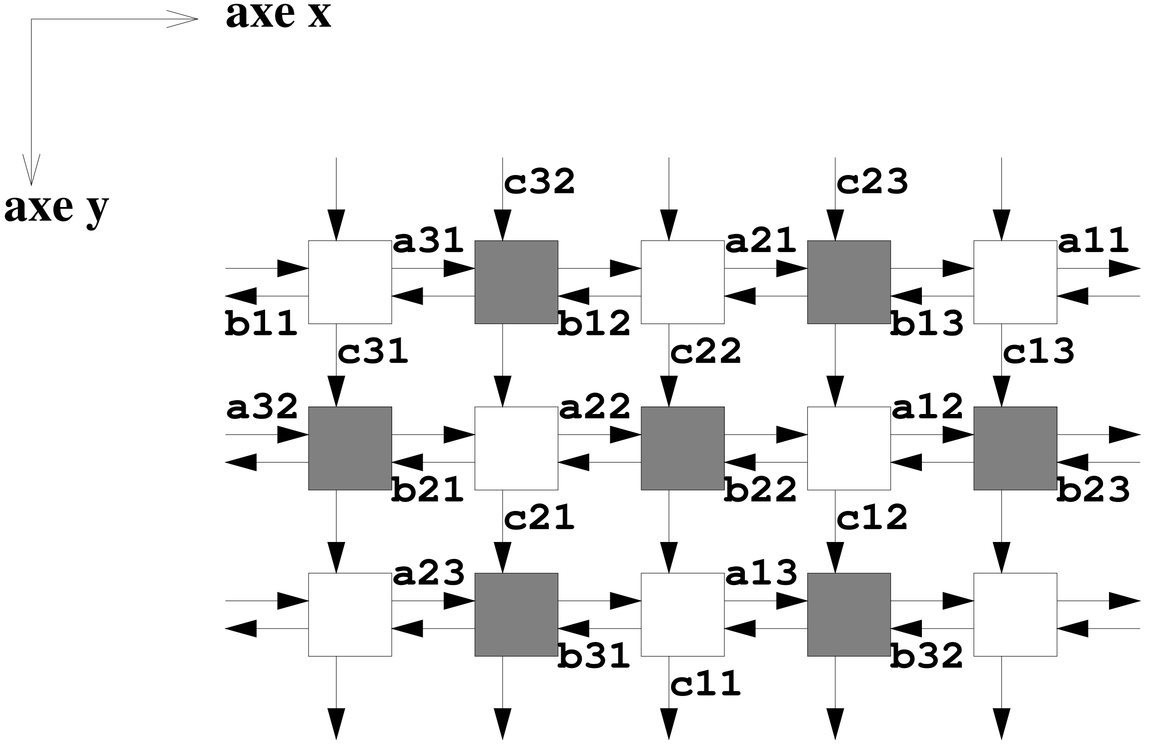
Change of basis $(i,j,k) \rightarrow (t,x,y) = (i+j+k, i, j)$



Projection along line $i = j,\ k = 0$

Allocation direction: $s_2 = (1,1,0)^t$

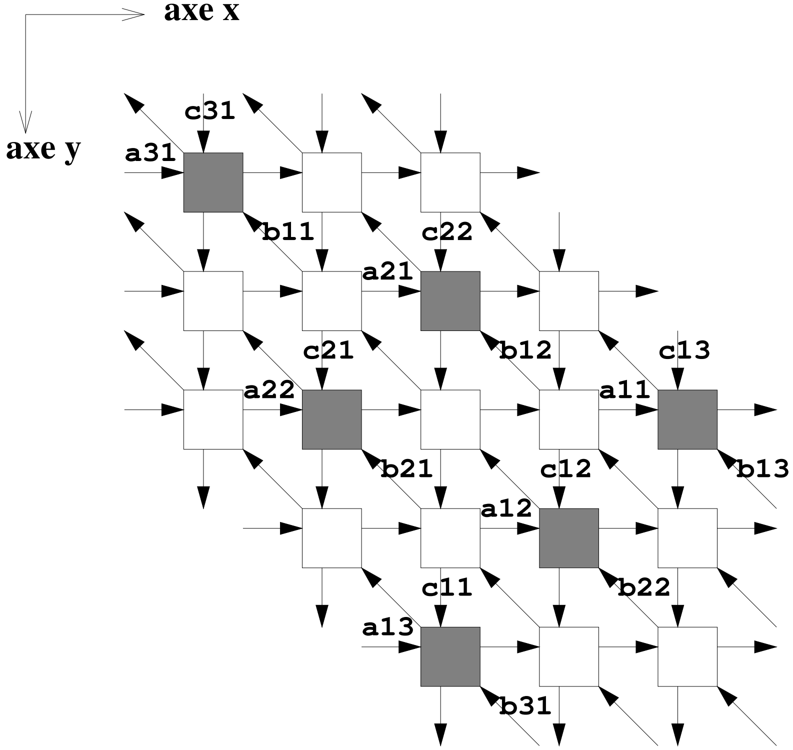
Change of basis $(i,j,k) \rightarrow (t,x,y) = (i+j+k, j-i, k)$



Projection along line $i = j = k$

Allocation direction : $s_3 = (1, 1, 1)^t$

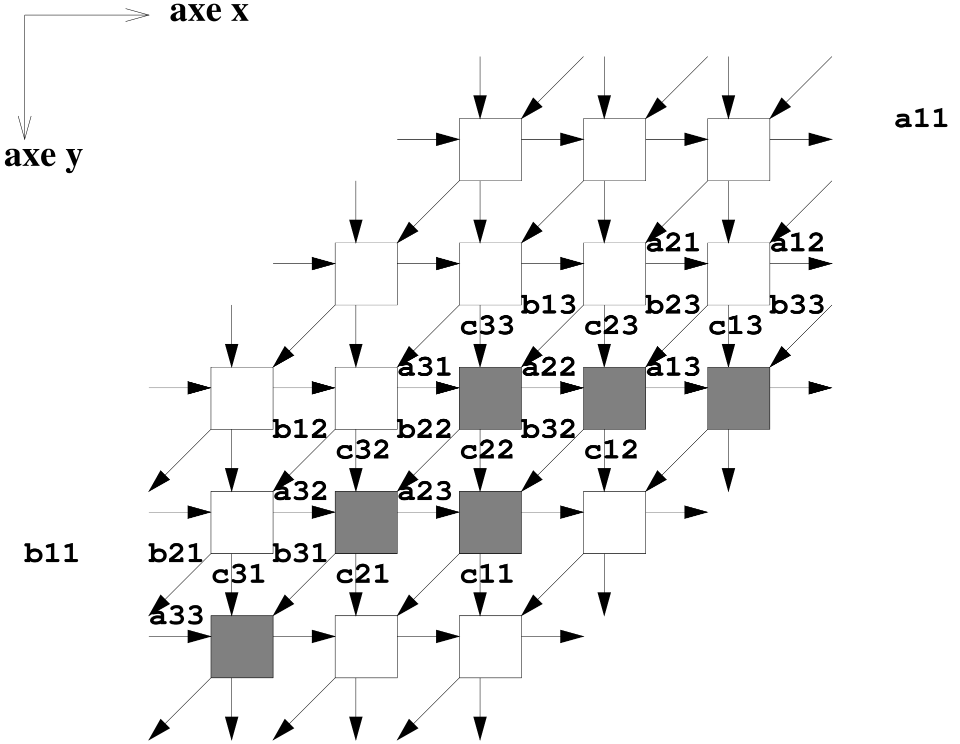
Change of basis: $(i, j, k) \rightarrow (t, x, y) = (i + j + k, j - i, k - i)$



Projection along line $i = j = -k$

Allocation direction: $s_4 = (1, 1, -1)^t$

Change of basis: $(i, j, k) \rightarrow (t, x, y) = (i + j + k, j - i, k + i)$



IRISA

Determining the change of basis (recipe)

1. Let s be a primitive vector such that that $\tau.s \neq 0$.
   (Example : $\tau = (1,1,1)$, and $s = (1,1,1)^t$.)

2. Complete s into a unimodular matrix S using right Hermite Normal Form
   Example :
$$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

3. Take $S^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$ the inverse of S.

4. The matrix P consisting of the last $n-1$ rows of S^{-1} gives the allocation function !

$$P = \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

Why ?

- a given point z is computed at time $\tau.z$ on processor $P.z$.

- the null space of P is generated by s, since $s = S \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ thus $S^{-1}s = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, thus, s is orthogonal to the rows of P.

- Let $z_1, z_2 \in \mathcal{D}$. If $\tau.z_1 = \tau.z_2$, then $P.z_1 \neq P.z_2$ (otherwise, $P.(z_1 - z_2) = 0$, hence, $z_1 - z_2 = k.s$, hence $\tau.s = 0$). But, $\tau.s \neq 0$, a contradiction

- conversely, if $P.z_1 = P.z_2$, then $\tau.z_1 \neq \tau.z_2$.

Systolic Synthesis from a SURE

Key Result: Systolic arrays correspond to SUREs with strongly separating hyperplane [Rao'85]

Schedule $t(z) = \tau^T z$ must have the same linear part for all variables. Then, same approach as for single URE

Observation: If the same change of basis transformation is applied to all variables of a SURE, the dependence functions remain uniform.

Systolic Synthesis from an ARE

The ARE **must** be converted into a URE (or SURE) — Uniformization or Localization

Two methods:

- Pipelining [FM84, RPF86, Roy88, WD88, QV89]
- Folding [CC83, YC88]

Pipelining

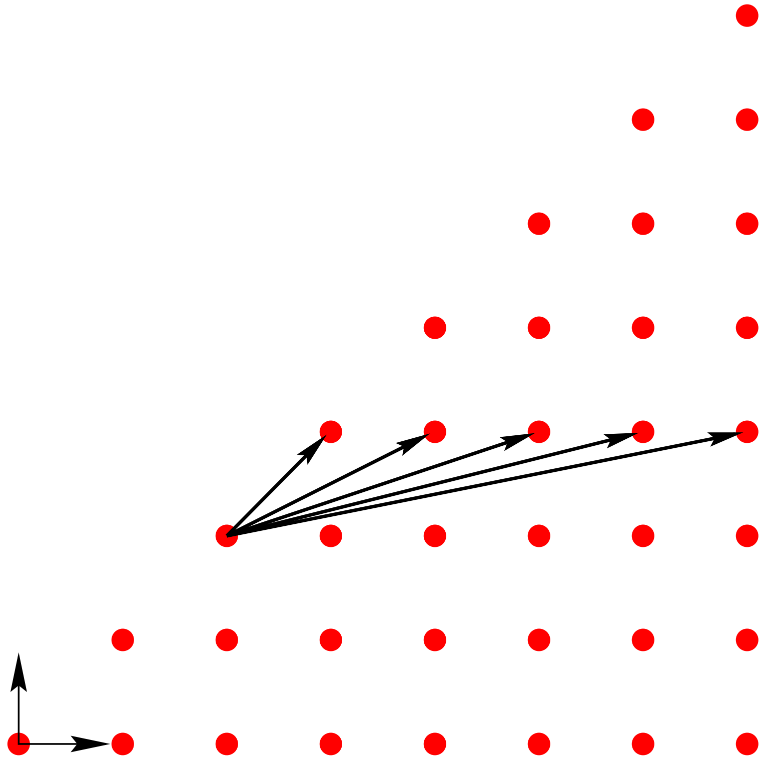
Based on detecting multiple utilizations of the same value.

Dependence function: $z \rightarrow f(z) = Az + a$

The kernel or null space of A identifies the index points where the same value is used.

Two points, z and z' use the same value iff $z - z' \in \mathrm{Null}(A)$

Null Space Pipelining



Example: $X[i,j] = g(\ldots X[i-1, i-1]\ldots)$

Null Space Pipelining

Dependence: $z \to Az + a = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ -1 \end{pmatrix}$

Null(A) is spanned by $[1, 0]^T$

Localization: use an auxillary recurrence

$$Y[i,j] = \begin{cases} \{i = j\} : & \Rightarrow \quad X[i-1, j-1] \\ \{i > j\} : & \Rightarrow \quad Y[i-1, j] \end{cases}$$

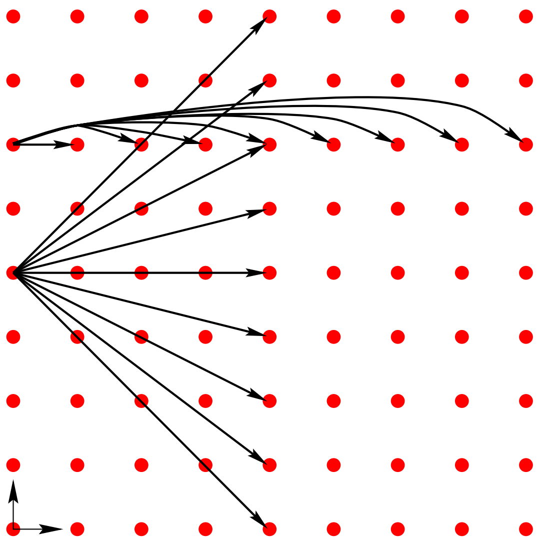
and replacing the ARE by

$$X[i,j] = g(\ldots Y[i,j] \ldots)$$

Necessary and Sufficient Conditions

- A must be singular (many-to-one dependency)
- and idempotent: $A^2 = A$. The producers are **close** to the consumers .

Multistage Pipelining



$$X[i,j] = g(\ldots X[0,j], X[0,i]\ldots)$$

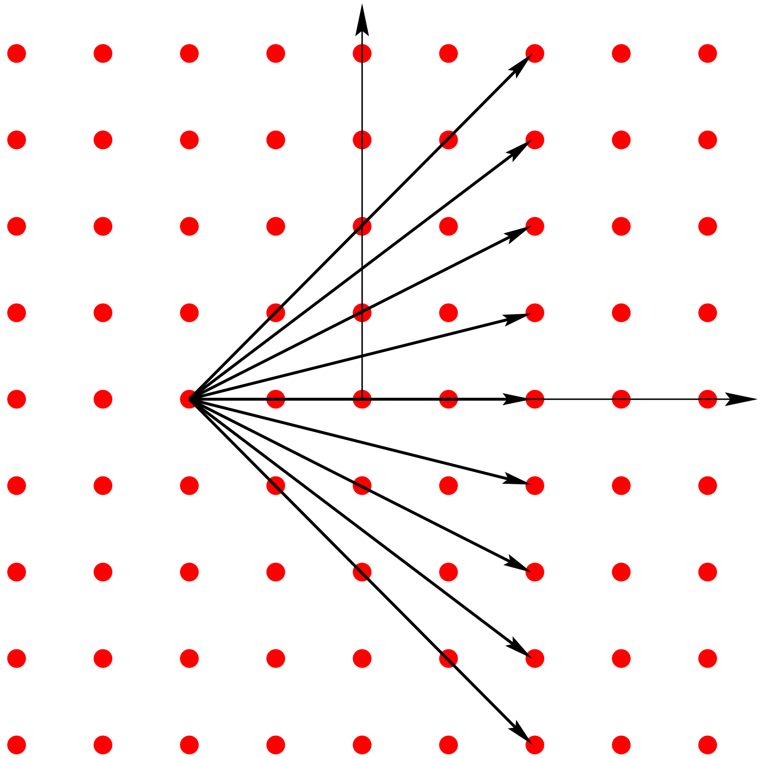
$(i,j \rightarrow 0,j)$ is idempotent, but $(i,j \rightarrow 0,i)$ is not.

Multistage Pipelining

Solution: The second dependency is piggybacked on the first.

$$
\begin{aligned}
X[i,j] &= g(\ldots Y[i,j], Z[i,j] \ldots) \\
Y[i,j] &= \begin{cases} \{i = 0\} : & \Rightarrow \quad X[i,j] \\ \{i > 0\} : & \Rightarrow \quad Y[i-1,j] \end{cases} \\
Z[i,j] &= \begin{cases} \{i < j\} : & \Rightarrow \quad Z[i,j-1] \\ \{i = j\} : & \Rightarrow \quad Y[i,j] \\ \{i > j\} : & \Rightarrow \quad Z[i,j+1] \end{cases}
\end{aligned}
$$

When Null Space Pipelining is Impossible



Example: $X[i,j] = g(\ldots X[-i,0]\ldots)$

consumers not close to producers

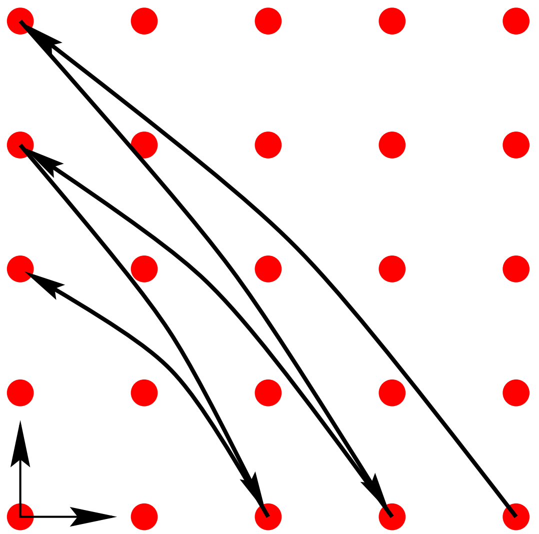
Solution: Pipeline through points other than just the consumers: inefficient, but the best we can do.

$$Z[i,j] = \begin{cases} \{2i \le j; j > 0\}: & \Rightarrow \quad Z[i-1, j-1] \\ \{j = 0\}: & \Rightarrow \quad X[i,j] \\ \{2i \ge j; j < 0\}: & \Rightarrow \quad Z[i+1, j+1] \end{cases}$$

$$Y[i,j] = \begin{cases} \{2i < j\}: & \Rightarrow \quad Y[i, j-1] \\ \{2i = j\}: & \Rightarrow \quad Z[i,j] \\ \{2i > j\}: & \Rightarrow \quad Y[i, j+1] \end{cases}$$

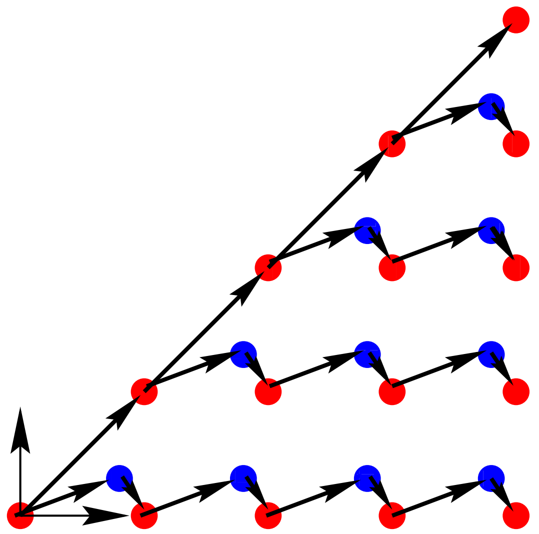
$$X[i,j] = g(\ldots Y[i,j] \ldots)$$

When the Dependency is Non-singular —— Folding



$$X[i,j] = \begin{cases} \{i = j\}: & \Rightarrow \quad X[i-1, j-1] \\ \{i > j\}: & \Rightarrow \quad X[j, i] \\ \{i < j\}: & \Rightarrow \quad X[j-1, i] \end{cases}$$

Fold about the $i-j$ line



$$X[i,j] = \begin{cases} \{i=j\}: & \Rightarrow \quad X[i-1,j-1] \\ \{i>j\}: & \Rightarrow \quad Y[i,j] \end{cases}$$
$$Y[i,j] = X[i-1,j]$$

When is this possible?

- If for some k $A^k = \mathrm{I}$, the non-singular matrix A is the generator of a group.
- There exist exactly $k-1$ folds — piecewise linear change of basis transformations

Relation with General Purpose Parallel Programming

- Alignment
- Data Distribution
- Communication Generation
- Global Operations (reductions, scans, etc.)

Alignment/Data Distribution

Embedding computations and data into a virtual index space such that communication is minimized.

Key point: dependency matrix (access function in nested loop programs) has information to determine potential communication.

[LC91b, GB91, HS91, KLD92, Fea92,DR94b, CGRT93, BKKPS94, DG96, DR95]

Communication

- detect when certain standard communication patterns occur: (broadcasts, scatters, gathers, etc. [LC91b]
- Use this to generate calls to optimized communication libraries
- Optimize communications by mesage vectorization, aggregation, etc.
- Specify abstractly the communication patterns [Raj93]

Global Operations

Consider the recurrence:

$$X[Pz + p] = \{ \ z \in \mathcal{D}\} : \quad \Rightarrow \quad g(\ldots X[Az + a)] \ldots)$$

- If P is non-singular, this definition is no longer functional.
- Interpret as the combination of multiple computations (using an associative operation).
- The set of points whose results are combined is characterized by Null(P)
- Serialization is the dual of Localization [Roy88]

Compilation of Alpha

- Naive approach (demand driven single assignment code) [WR95]
- Imperative (sequential, single assignment) code [QRW95b]
- Imperative (data parallel single assignment) code [QRW95a]
- Imperative (sequential, multiple assignment) code [WR96]

Demand driven approach

- Allocate memory for the domain of all variables
- Generate a loop to visit all points in the domains of all output variables
- At each point evaluate the result (recursively)

Scanning a polyhedron

```
{i,j,k| i>=0; -1+M>=0; -j+N>=0; k>=0; i+j-k>=0}
```

(a) Original polyhedron

```
{ N,M| N,M>0 }
```

(b) Parameter space

```
{j| 0<=j<=N} ::
    {k,j| 0<=k<=j+M} ::
        {i,k,j| 0<=i<=M; i>=k-j}
```

(c) After index separation

```
for (j=0; j<=N; j++)
  for (k=0; k<=j+M; k++)
    for (i=max(0,k-j); i<=M; i++)
```

(d) Corresponding C code

- (a) and (c) — same polyhedron in two different notations
- From (c) we generate C code (d) simply by pretty-printing

Imperative (sequential, single assignment) code

- Choose a (possibly multidimensional) schedule for each variable in the program
- Complete the schedules to obtain a valid change of basis
- Generate loop nests to visit all points in the (union of) the domains of the variables of the transformed Alpha program in lexicographic order of time

Multiple Assignment code

Memory reuse analysis [WR96]

- Schedule is known
- Perform lifetime analysis to determine when it is safe to over-write a given value
- Use it to determine the validity of projection functions (sim-ilar to allocation functions)
- Generate code that
  - Allocates memory only for the projections of variables do-mains
  - Writes multiple times into the same memory location