Related Entries

►Functional Languages
►Parallelization, Automatic
►Prolog Machines

Bibliographic Notes and Further Reading

Pointers to a brief selection of the related literature have been provided within the text and are included in the references. For a much more detailed coverage of this very large topic the reader is referred to the comprehensive surveys in [7, 15, 18].

Bibliography

1. Ali K, Karlsson R (1990) The Muse approach to or-parallel Prolog. Int J Parallel Program 19(2):129–162
2. Bevemyr J, Lindgren T, Millroth H (1993) Reform Prolog: the language and its implementation. In: Warren DS (ed) Proceedings of tenth international conference on logic programming. MIT Press, Cambridge, pp 283–298
3. Bueno F, García de la Banda M, Hermenegildo M (March 1999) Effectiveness of abstract interpretation in automatic parallelization: a case study in logic programming. ACM Trans Program Lang Syst 21(2):189–238
4. Conery JS (1987) Parallel execution of logic programs. Kluwer, Norwell
5. Costa VS, Warren DHD, Yang R (1996) Andorra-I compilation. New Gener Comput 14(1):3–30
6. Green C (1969) Theorem proving by resolution as a basis for question-answering systems. In: Machine intelligence, 4
7. Gupta G, Pontelli E, Ali K, Carlsson M, Hermenegildo M (July 2001) Parallel execution of Prolog programs: a survey. ACM Trans Program Lang Syst 23(4):472–602
8. Hermenegildo M (December 2000) Parallelizing irregular and pointer-based computations automatically: perspectives from logic and constraint programming. Parallel Comput 26(13–14):1685–1708
9. Hermenegildo M, Carro M (July 1996) Relating data-parallelism and (and-) parallelism in logic programs. Comput Lang J 22(2/3):143–163
10. Hermenegildo M, Greene K (1991) The &-Prolog system: exploiting independent and-parallelism. New Gener Comput 9(3,4):233–257
11. Kacsuk P (1992) Distributed data driven Prolog abstract machine (3DPAM). In: Kacsuk P, Wise MJ (eds) Implementations of distributed Prolog. Wiley, Aachen, pp 89–118
12. Lloyd JW (1987) Foundations of logic programming, 2nd extended edn. Springer, New York
13. Lusk E, Butler R, Disz T, Olson R, Stevens R, Warren DHD, Calderwood A, Szeredi P, Brand P, Carlsson M, Ciepielewski A, Hausman B, Haridi S (1988) The aurora or-parallel Prolog system. New Gener Comput 7(2/3):243–271
14. Muthukumar K, Bueno F, García de la Banda M, Hermenegildo M (February 1999) Automatic compile-time parallelization of logic programs for restricted, goal-level, independent and-parallelism. J Log Program 38(2):165–218
15. Shapiro EY (September 1989) The family of concurrent logic programming languages. ACM Comput Surv 21(3):412–510
16. Shen K (November 1996) Overview of DASWAM: exploitation of dependent and-parallelism. J Log Program 29(1–3):245–293
17. Smith D (1996) MultiLog and data or-parallelism. J Log Program 29(1–3):195–244
18. Tick E (May 1995) The deevolution of concurrent programming languages. J Log Program 23(2):89–124
19. Ueda K (1987) Guarded Horn clauses. In: Shapiro EY (ed) Concurrent Prolog: collected papers. MIT Press, Cambridge, pp 140–156

LogP Bandwidth-Latency Model

►Bandwidth-Latency Models (BSP, LogP)

Loop Blocking

►Tiling

Loop Nest Parallelization

Utpal Banerjee
University of California at Irvine, Irvine, CA, USA

Synonyms

Parallelization

Definition

Loop Nest Parallelization refers to the problem of finding parallelism in a perfect nest of sequential loops. It typically deals with transformations that change the execution order of iterations of such a nest **L** by creating a different nest **L′**, such that **L′** is equivalent to **L** and some of its loops can run in parallel.

Discussion

Introduction

For a better understanding of the current essay, the reader should first go through the essay ►Unimodular Transformations in this encyclopedia.

The program model is a perfect nest **L** of m sequential loops. An *iteration* of **L** is an instance of the body of **L**. The program consists of a certain set of iterations that are to be executed in a certain sequential order. This execution order imposes a *dependence structure* on the set of iterations, based on how they access different memory locations. A loop in **L** *carries a dependence* if the dependence between two iterations is due to the unfolding of that loop. A loop can *run in parallel* if it carries no dependence.

In this essay, one considers transformations that change **L** into a perfect nest $\mathbf{L}'$ with the same set of iterations executing in a different sequential order. (The number of loops may differ from **L** to $\mathbf{L}'$.) The new nest $\mathbf{L}'$ is *equivalent* to the old nest **L** (i.e., they give the same results), if whenever an iteration depends on another iteration in **L**, the first iteration is executed after the second in $\mathbf{L}'$. If a loop in $\mathbf{L}'$ carries no dependence, then that loop can run in parallel. The goal is to transform **L** into an equivalent nest $\mathbf{L}'$, such that some inner and/or outer loops of $\mathbf{L}'$ can run in parallel.

Two major loop nest transformations are discussed here: unimodular and echelon. Unimodular transformations have already been introduced in this encyclopedia in the essay under that name. There it is shown by examples how to achieve inner and outer loop parallelization via unimodular transformations. Detailed algorithms for achieving those goals are given in this essay. Echelon transformations are explained after that, and it is pointed out how a combination of both transformations can maximize loop parallelization.

Mathematical Preliminaries

Lexicographic order and Fourier's method of elimination, as explained in the essay ►Unimodular Transformations, will be used in the following. In this section, some of the elementary definitions and algorithms of matrix theory are stated, customized for integer matrices. From now on, a matrix is an integer matrix, unless explicitly designated to be otherwise.

The transpose of a matrix **A** is denoted by $\mathbf{A}'$. The $m \times m$ unit matrix is denoted by $\mathcal{I}_m$. A *submatrix* of a given matrix **A** is the matrix obtained by deleting some rows and columns of **A**. If **A** is an $m \times p$ matrix and **B** an $m \times q$ matrix, then the *column-augmented matrix* $(\mathbf{A};\mathbf{B})$ is the $m \times (p+q)$ matrix whose first p columns are the columns of **A** and the last q columns are the columns of **B**. Thus,

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 0 & 0 \\ 0 & 5 \end{pmatrix} \text{ and } \mathbf{B} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$\text{yield } (\mathbf{A};\mathbf{B}) = \left(\begin{array}{cc|c} 1 & 2 & 1 \\ 0 & 0 & 0 \\ 0 & 5 & 0 \end{array}\right).$$

A *row-augmented matrix* is defined similarly.

The *(main) diagonal* of an $m \times n$ matrix (a_{ij}) is the vector $(a_{11}, a_{22}, \ldots, a_{pp})$ where $p = \min(m,n)$. The *rank* of a matrix **A**, denoted by $\text{rank}(\mathbf{A})$, is the maximum number of linearly independent rows (or columns) of **A**.

For a given $m \times n$ matrix **A**, let ℓ_i denote the column number of the leading (first nonzero) element of row i. (For a zero row, ℓ_i is undefined.) Then **A** is an *echelon matrix*, if for some integer ρ in $0 \le \rho \le m$, the following holds:

1. The rows 1 through ρ are nonzero rows.
2. The rows $\rho + 1$ through m are zero rows.
3. For $1 \le i \le \rho$, each element in column ℓ_i below row i is zero.
4. $\ell_1 < \ell_2 < \cdots < \ell_\rho$.

If there is such a ρ, then $\rho = \text{rank}(\mathbf{A})$. A zero matrix is an echelon matrix for which $\rho = 0$. Three nonzero echelon matrices with the values $\rho = 4, 3, 2$, respectively, are given below:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \begin{pmatrix} 5 & 2 & 3 \\ 0 & -1 & 0 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 5 & 2 & 3 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

For any given matrix, there are three types of *elementary row operations*:

1. *Reversal*: multiply a row by −1.
2. *Interchange*: interchange two rows.
3. *Skewing*: add an integer multiple of one row to another row.

The *elementary column operations* are defined similarly.

An *elementary matrix* is any matrix obtained from a unit matrix by one elementary row operation. Thus, there are three types of elementary matrices. The elementary matrices can also be derived by column operations. For example, the same matrix is obtained from the 3 × 3 unit matrix if we interchange rows 1 and 3, or interchange columns 1 and 3.

A *unimodular matrix* is a square integer matrix with determinant 1 or −1. Each elementary matrix is unimodular, and each unimodular matrix can be expressed as the product of a finite number of elementary matrices (Lemma 2.3 in [2]). For a given matrix **A**, performing an elementary row operation is equivalent to forming a product of the form **EA**, where **E** is the corresponding elementary matrix. Applying a finite sequence of row operations to **A** is the same as forming a product of the form **UA**, where **U** is a unimodular matrix.

Reducing an $m \times n$ matrix **A** to *echelon form* means finding an $m \times m$ unimodular matrix **U** and an $m \times n$ echelon matrix **S**, such that $\mathbf{UA} = \mathbf{S}$. Figure 1 gives an algorithm that reduces any given matrix **A** to echelon form. Note that applying the same row operation to the two matrices **U** and **S** separately is equivalent to applying that operation to the column-augmented matrix $(\mathbf{U}; \mathbf{S})$.

For a given matrix **A**, one may need to find a unimodular matrix **V** and an echelon matrix **S** such that $\mathbf{A} = \mathbf{VS}$. This **V** could be the inverse of the unimodular matrix **U** of Algorithm 1. However, if **U** is not needed, Algorithm 1 can be modified to yield directly a unimodular **V** and an echelon **S** such that $\mathbf{A} = \mathbf{VS}$. This is the algorithm of Fig. 2. In Algorithm 2, whenever a row operation is applied to **S**, it is followed by a column operation applied to **V**. The row operation is equivalent to forming a product of the form **ES**, where **E** is an elementary matrix. The column operation that follows is equivalent to forming the product $\mathbf{VE}^{-1}$. Hence, the relation $\mathbf{A} = \mathbf{VS}$ remains unchanged since $\mathbf{VS} = (\mathbf{VE}^{-1})(\mathbf{ES})$.

Each algorithm can be modified by using extra reversal operations, as needed, so that any given rows of the computed echelon matrix are nonnegative.

The program model for all transformations is a perfect nest of loops $\mathbf{L} = (L_1, L_2, \ldots, L_m)$ shown in Fig. 3. The limits p_r and q_r are integer-valued linear functions of $I_1, I_2, \ldots, I_{r-1}$. All distance vectors are assumed to be uniform. If there is no distance vector **d** with $\mathbf{d} \succ_r \mathbf{0}$, then the loop L_r carries no dependence and is able to run in parallel.

Unimodular Transformations

Let **U** denote an $m \times m$ unimodular matrix. The unimodular transformation of the loop nest **L** induced by **U** is

Algorithm 1 Given an $m \times n$ integer matrix **A**, this algorithm finds an $m \times m$ unimodular matrix **U** and an $m \times n$ echelon matrix $\mathbf{S} = (s_{ij})$, such that $\mathbf{UA} = \mathbf{S}$. It starts with $\mathbf{U} = \mathcal{I}_m$ and $\mathbf{S} = \mathbf{A}$, and works on the matrix $(\mathbf{U}; \mathbf{S})$. Let i_0 denote the row number in which the last processed column of **S** had a nonzero element.

```
set U ← I_m, S ← A, i_0 ← 0
do j = 1, n, 1
    if there is at least one nonzero s_ij with i_0 < i ≤ m
    then
        set i_0 ← i_0 + 1
        do i = m, i_0 + 1, −1
            do while s_ij ≠ 0
                set σ ← sgn (s_{i−1,j} * s_ij)
                    z ← ⌊|s_{i−1,j}|/|s_ij|⌋
                subtract σz times row i from row (i − 1) in (U; S)
                interchange rows i and (i − 1) in (U; S)
```

Loop Nest Parallelization. Fig. 1 Echelon Reduction Algorithm

Algorithm 2 Given an $m \times n$ integer matrix $\mathbf{A}$, this algorithm finds an $m \times m$ unimodular matrix $\mathbf{V}$ and an $m \times n$ echelon matrix $\mathbf{S} = (s_{ij})$, such that $\mathbf{A} = \mathbf{VS}$. It starts with $\mathbf{V} = \mathcal{I}_m$ and $\mathbf{S} = \mathbf{A}$, and works on the matrices $\mathbf{V}$ and $\mathbf{S}$. Let i_0 denote the row number in which the last processed column of $\mathbf{S}$ had a nonzero element.

set $\mathbf{V} \leftarrow \mathcal{I}_m$, $\mathbf{S} \leftarrow \mathbf{A}$, $i_0 \leftarrow 0$
do $j = 1, n, 1$
  if there is at least one nonzero s_{ij} with $i_0 < i \le m$
  then
    set $i_0 \leftarrow i_0 + 1$
    do $i = m, i_0 + 1, -1$
      do while $s_{ij} \neq 0$
        set $\sigma \leftarrow \text{sgn}\,(s_{i-1,j} * s_{ij})$
          $z \leftarrow \lfloor |s_{i-1,j}|/|s_{ij}| \rfloor$
        subtract σz times row i from row $(i-1)$ in $\mathbf{S}$
        add σz times column $(i-1)$ to column i in $\mathbf{V}$
        interchange rows i and $(i-1)$ in $\mathbf{S}$
        interchange columns i and $(i-1)$ in $\mathbf{V}$

Loop Nest Parallelization. Fig. 2 Modified Echelon Reduction Algorithm

$$
\begin{array}{ll}
L_1: & \textbf{do } I_1 = p_1, q_1 \\
L_2: & \quad \textbf{do } I_2 = p_2, q_2 \\
\vdots & \quad\quad \vdots \\
L_m: & \quad\quad \textbf{do } I_m = p_m, q_m \\
 & \quad\quad\quad H(\mathbf{I})
\end{array}
$$

Loop Nest Parallelization. Fig. 3 Loop Nest **L**

$$
\begin{array}{ll}
L'_1: & \textbf{do } K_1 = \alpha_1, \beta_1 \\
L'_2: & \quad \textbf{do } K_2 = \alpha_2, \beta_2 \\
\vdots & \quad\quad \vdots \\
L'_m: & \quad\quad \textbf{do } K_m = \alpha_m, \beta_m \\
 & \quad\quad\quad H'(\mathbf{K})
\end{array}
$$

Loop Nest Parallelization. Fig. 4 Loop Nest $\mathbf{L}'$

a loop nest $\mathbf{L}'$ of the form shown in Fig. 4, where $\mathbf{K} = \mathbf{IU}$ and $H'(\mathbf{K}) = H(\mathbf{KU}^{-1})$. The transformed program has the same set of iterations as the original program, executing in a different sequential order. The transformation $\mathbf{L} \Rightarrow \mathbf{L}'$ is Valid if and only if $\mathbf{dU} \succ \mathbf{0}$ for each distance vector $\mathbf{d}$ of $\mathbf{L}$. In that case, the vectors $\mathbf{dU}$ are precisely the distance vectors of the new nest $\mathbf{L}'$.

Inner Loop Parallelization

Let D denote the set of distance vectors of $\mathbf{L}$. For the transformation $\mathbf{L} \Rightarrow \mathbf{L}'$ to be valid, the matrix $\mathbf{U}$ must be so chosen that $\mathbf{dU} \succ \mathbf{0}$ for each $\mathbf{d} \in D$. After a valid transformation by $\mathbf{U}$, the set of distance vectors of $\mathbf{L}'$ is $\{\mathbf{dU} : \mathbf{d} \in D\}$. A loop L'_r of $\mathbf{L}'$ can run in parallel if there is no $\mathbf{d} \in D$ satisfying $\mathbf{dU} \succ_r \mathbf{0}$. Thus, a unimodular matrix $\mathbf{U}$ will transform the loop nest $\mathbf{L}$ into an equivalent loop nest $\mathbf{L}'$ where the loops $L'_2, L'_3, \ldots, L'_m$ can all run in parallel, if and only if $\mathbf{U}$ satisfies the condition

$$\mathbf{dU} \succ_1 \mathbf{0} \qquad (\mathbf{d} \in D). \tag{1}$$

The following informal algorithm shows in detail that a unimodular matrix with this property can always be constructed.

Algorithm 3 Given a perfect nest $\mathbf{L}$ of m loops, as shown in Fig. 3, with a set of distance vectors D, this algorithm finds an $m \times m$ unimodular matrix $\mathbf{U} = (u_{rt})$ that transforms $\mathbf{L}$ into an equivalent loop nest $\mathbf{L}'$, such that the inner loops $L'_2, L'_3, \ldots, L'_m$ of $\mathbf{L}'$ can all run in parallel.

1. If $D = \emptyset$, then all loops of $\mathbf{L}$ can execute in parallel; take for $\mathbf{U}$ the $m \times m$ unit matrix and exit. Otherwise, go to step 2.
2. Condition (1) is equivalent to the following system of inequalities

$$d_1 u_1 + d_2 u_2 + \cdots + d_m u_m \ge 1 \qquad (\mathbf{d} \in D), \tag{2}$$

where $\mathbf{d} = (d_1, d_2, \ldots, d_m)$ and the fixed second subscript of u_{r1} has been dropped for convenience.

For each r in $1 \le r \le m$, define the set

$$D_r = \{\mathbf{d} \in D : \mathbf{d} \succ_r \mathbf{0}\}.$$

At least one of these sets is nonempty. Since $d_1 = d_2 = \cdots = d_{r-1} = 0$ for each $\mathbf{d} \in D_r$, one can break up the system (2) into a sequence of subsystems of the following type:

$$\left.\begin{aligned} d_m u_m &\geq 1 \quad (\mathbf{d} \in D_m) \\ d_{m-1}u_{m-1} + d_m u_m &\geq 1 \quad (\mathbf{d} \in D_{m-1}) \\ &\vdots \\ d_1 u_1 + d_2 u_2 + \cdots + d_{m-1}u_{m-1} + d_m u_m &\geq 1 \quad (\mathbf{d} \in D_1). \end{aligned}\right\}$$

Since $d_r > 0$ for $\mathbf{d} \in D_r$, this sequence can be rewritten as

$$\left.\begin{aligned} u_m &\geq 1/d_m && (\mathbf{d} \in D_m) \\ u_{m-1} &\geq (1 - d_m u_m)/d_{m-1} && (\mathbf{d} \in D_{m-1}) \\ &\vdots \\ u_1 &\geq (1 - d_2 u_2 - \cdots - d_m u_m)/d_1 && (\mathbf{d} \in D_1). \end{aligned}\right\} \quad (3)$$

3. If $D_m = \emptyset$, then take $u_m = 0$. Otherwise, take $u_m = 1$. Suppose $u_m, u_{m-1}, \ldots, u_{r+1}$ have been chosen, where $1 \leq r < m$. If $D_r = \emptyset$, then take $u_r = 0$. Otherwise, u_r has a set of lower bounds of the form:

$$u_r \geq (1 - d_{r+1}u_{r+1} - d_{r+2}u_{r+2} - \cdots - d_m u_m)/d_r \quad (\mathbf{d} \in D_r).$$

Take for u_r the smallest nonnegative integer that would work:

$$u_r = \left\lceil \max_{\mathbf{d} \in D_r} \{(1 - d_{r+1}u_{r+1} - d_{r+2}u_{r+2} - \cdots - d_m u_m)/d_r\} \right\rceil^+ .$$

4. Find s in $1 \leq s \leq m$ such that u_s is the first nonzero element in the sequence: $u_m, u_{m-1}, \ldots, u_1$. Note that $u_s = 1$. Set

$$\mathbf{U} = \begin{pmatrix} u_1 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ u_{s-1} & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{pmatrix},$$

such that row s of $\mathbf{U}$ is $(1, 0, 0, \ldots, 0)$, its first column is $(u_1, u_2, \ldots, u_m)$, and the submatrix obtained by deleting row s and the first column of $\mathbf{U}$ is the $(m-1) \times (m-1)$ unit matrix. Since $\det \mathbf{U} = 1$, $\mathbf{U}$ is unimodular.

5. Note that the inverse of $\mathbf{U}$ is given by

$$\mathbf{U}^{-1} = \begin{pmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 1 & \cdots & 0 & -u_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & -u_{s-1} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \end{pmatrix},$$

such that column s is $(1, -u_1, \ldots, -u_{s-1}, 0, \ldots, 0)$, row 1 is $(0, \ldots, 0, 1, 0, \ldots, 0)$, and the submatrix obtained by deleting column s and the first row is the $(m-1) \times (m-1)$ unit matrix. From the equation $(I_1, I_2, \ldots, I_m) = (K_1, K_2, \ldots, K_m)\mathbf{U}^{-1}$, it follows that

$$\left.\begin{aligned} I_1 &= K_2 \\ &\vdots \\ I_{s-1} &= K_s \\ I_s &= K_1 - u_1 K_2 - \cdots - u_{s-1}K_s \\ I_{s+1} &= K_{s+1} \\ &\vdots \\ I_m &= K_m. \end{aligned}\right\}$$

In the system of inequalities

$$p_r(I_1, \ldots, I_{r-1}) \leq I_r \leq q_r(I_1, \ldots, I_{r-1}) \quad (1 \leq r \leq m),$$

substitute for each I_r the corresponding expression in $K_1, K_2, \ldots, K_m$. Get the loop limits $\alpha_r(K_1, \ldots, K_{r-1})$ and $\beta_r(K_1, \ldots, K_{r-1})$ of Fig. 4 by

solving this system using Fourier's method of elimination.

6. Now each distance vector **d** of **L** satisfies the condition $\mathbf{dU} \succ_1 \mathbf{0}$. Hence, the transformed nest $\mathbf{L}'$ of Fig. 4 created by this matrix **U** is equivalent to the original nest **L**, and the inner loops of $\mathbf{L}'$ can all execute in parallel.

Remark 1 There exist infinitely many vectors $\mathbf{u} = (u_1, u_2, \ldots, u_m)$ satisfying (3) and $\gcd(u_1, u_2, \ldots, u_m) = 1$. For each such vector **u**, there exist infinitely many unimodular matrices with **u** as the first column. (See Sect. 3.3 in [2].) Any such matrix will satisfy the goal of Algorithm 3. Among all possible choices for **u**, an ideal vector is one that minimizes the number of iterations $(\beta_1 - \alpha_1 + 1)$ of the outermost loop L_1' that must run sequentially. This poses an optimization problem; see Chap. 3 in [3] for details.

Since $\mathbf{K} = \mathbf{IU}$ and $(u_1, u_2, \ldots, u_m)$ is the first column of **U**, one has $K_1 = u_1 I_1 + u_2 I_2 + \cdots + u_m I_m$. An equation of the form $K_1 = c$, where c is a constant, represents a hyperplane in the vector space $\mathbf{R}^m$ with coordinate axes $I_1, I_2, \ldots, I_m$. The integral values of K_1 from α_1 to β_1 then represent a packet of parallel hyperplanes through the index space of the loop nest **L**, perpendicular to the vector **u**. Geometrically speaking, Algorithm 3 finds a sequence of parallel hyperplanes, such that the planes are taken sequentially, and iterations for all index points on each plane are executed in parallel. This algorithm is derived from Lamport's 1974 CACM paper [8]. The name *Hyperplane Method*, often used to describe it, also comes from [8].

Example 1 To better understand how Algorithm 3 creates the matrix **U** from a given set of distance vectors D, consider a loop nest $\mathbf{L} = (L_1, L_2, L_3, L_4)$ with the set:

$$D = \{(0,0,0,2), (0,3,1,-2), (0,4,-6,0), (1,-5,3,1), (2,1,0,0), (3,0,-2,1)\}.$$

The only loop that carries no dependence is L_3. Hence, only L_3 can run in parallel. The goal is to find a unimodular matrix **U** that will transform **L** into an equivalent loop nest $\mathbf{L}'$, where all the inner loops can execute in parallel. The first column (u_1, u_2, u_3, u_4) of **U** must satisfy the inequality

$$d_1 u_1 + d_2 u_2 + d_3 u_3 + d_4 u_4 \geq 1$$

for each $(d_1, d_2, d_3, d_4) \in D$. After simplification, one gets a set of six inequalities:

$$\left.\begin{array}{rcl} u_4 & \geq & 1/2 \\ u_2 & \geq & (1 - u_3 + 2u_4)/3 \\ u_2 & \geq & (1 + 6u_3)/4 \\ u_1 & \geq & 1 + 5u_2 - 3u_3 - u_4 \\ u_1 & \geq & (1 - u_2)/2 \\ u_1 & \geq & (1 + 2u_3 - u_4)/3. \end{array}\right\}$$

For each u_r, select the smallest possible nonnegative integral value that will work. So, take $u_4 = 1$. Since $D_3 = \emptyset$ (i.e., there is no distance vector **d** with $\mathbf{d} \succ_3 \mathbf{0}$), take $u_3 = 0$. The constraints on u_2 are then $u_2 \geq 1$ and $u_2 \geq 1/4$. Take $u_2 = 1$. Finally, the lower bounds on u_1 are given by $u_1 \geq 5$, $u_1 \geq 0$, and $u_1 \geq 0$, so that one takes $u_1 = 5$. The unimodular matrix **U** as constructed in Step 4 of Algorithm 3 is

$$\mathbf{U} = \begin{pmatrix} 5 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Here $s = 4$. Row 4 of **U** is $(1,0,0,0)$, its first column is $(u_1, u_2, u_3, u_4) = (5,1,0,1)$, and the submatrix obtained by deleting the fourth row and the first column is the 3×3 unit matrix. The set of distance vectors of the equivalent loop nest $\mathbf{L}' = (L_1', L_2', L_3', L_4')$ is the set of vectors **dU** for $\mathbf{d} \in D$, that is, the set

$$\{(2,0,0,0), (1,0,3,1), (4,0,4,-6), (1,1,-5,3), (11,2,1,0), (16,3,0,-2)\}.$$

None of the inner loops L_2', L_3', L_4' carries a dependence, and hence they can all run in parallel.

Outer Loop Parallelization

For a given $m \times n$ matrix $\mathbf{A} = (a_{ij})$, the rows are denoted by $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m$ and the columns by $\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^n$. Note that the proof of the following theorem uses both the *set* D of distance vectors and the distance *matrix* **D** of **L**, while the previous algorithm used only D.

Theorem 1 *Consider a nest* $\mathbf{L}$ *of* m *loops. Let* $\mathbf{D}$ *denote the distance matrix of* $\mathbf{L}$ *and* ρ *the rank of* $\mathbf{D}$*. Then there exists a valid unimodular transformation* $\mathbf{L} \Rightarrow \mathbf{L}'$ *such that the outermost* $(m-\rho)$ *loops and the innermost* $(\rho-1)$ *loops of* $\mathbf{L}'$ *can execute in parallel.*

Proof Let D denote the set of distance vectors of $\mathbf{L}$. A unimodular matrix $\mathbf{U}$ will induce a valid transformation $\mathbf{L} \Rightarrow \mathbf{L}'$ if and only if $\mathbf{dU} > \mathbf{0}$ for each $\mathbf{d} \in D$. After a valid transformation by $\mathbf{U}$, the distance vectors of $\mathbf{L}'$ are the vectors $\mathbf{dU}$, where $\mathbf{d} \in D$. The outermost $(m-\rho)$ loops and the innermost $(\rho-1)$ loops of $\mathbf{L}'$ can execute in parallel, if and only if $\mathbf{dU} >_r \mathbf{0}$ is false for each $\mathbf{d} \in D$ when $r \neq m-\rho+1$. Let $n = m-\rho$. Then each $\mathbf{d} \in D$ must satisfy $\mathbf{dU} >_{n+1} \mathbf{0}$.

The transpose $\mathbf{D}'$ of $\mathbf{D}$ has m rows and its columns are the distance vectors of $\mathbf{L}$. By Algorithm 1, find an $m \times m$ unimodular matrix $\mathbf{V}$ and an echelon matrix $\mathbf{S}$ such that $\mathbf{VD}' = \mathbf{S}$. ($\mathbf{S}$ and $\mathbf{D}'$ have the same size.) The number of nonzero rows of $\mathbf{S}$ is the rank ρ of $\mathbf{D}$, and the number of zero rows is $n = m-\rho$. Since the lowest n rows of $\mathbf{S}$ are zero vectors, it follows that each of the lowest n rows of $\mathbf{V}$ is orthogonal to each $\mathbf{d} \in D$.

Next, find an m-vector $\mathbf{u}$ by Algorithm 3 such that $\mathbf{du} > 0$ for each $\mathbf{d} \in D$. Let $\mathbf{A}$ denote the $m \times (n+1)$ matrix where the columns $\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^n$ are the lowest n rows of $\mathbf{V}$, and $\mathbf{a}^{n+1} = \mathbf{u}$. Then each $\mathbf{d} \in D$ satisfies the equations:

$$\mathbf{da}^1 = 0,\ \mathbf{da}^2 = 0, \ldots,\ \mathbf{da}^n = 0,\ \mathbf{da}^{n+1} > 0. \qquad (4)$$

By Algorithm 2, find an $m \times m$ unimodular matrix $\mathbf{U}$ and an $m \times (n+1)$ echelon matrix

$$\mathbf{T} = \begin{pmatrix} t_{11} & t_{12} & t_{13} & \cdots & t_{1n} & t_{1,n+1} \\ 0 & t_{22} & t_{23} & \cdots & t_{2n} & t_{2,n+1} \\ 0 & 0 & t_{33} & \cdots & t_{3n} & t_{3,n+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & t_{nn} & t_{n,n+1} \\ 0 & 0 & 0 & \cdots & 0 & t_{n+1,n+1} \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix},$$

such that $\mathbf{A} = \mathbf{UT}$ and the diagonal element $t_{n+1,n+1}$ is nonnegative. The unimodular transformation $\mathbf{L} \Rightarrow \mathbf{L}'$ induced by $\mathbf{U}$ satisfies the theorem.

After writing the relation $\mathbf{A} = \mathbf{UT}$ in the form

$$(\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^n, \mathbf{a}^{n+1}) = (\mathbf{u}^1, \mathbf{u}^2, \ldots, \mathbf{u}^n, \mathbf{u}^{n+1}, \mathbf{u}^{n+2}, \ldots, \mathbf{u}^m) \cdot \mathbf{T},$$

it becomes clear that

$$\left.\begin{aligned} \mathbf{a}^1 &= t_{11}\mathbf{u}^1 \\ \mathbf{a}^2 &= t_{12}\mathbf{u}^1 + t_{22}\mathbf{u}^2 \\ &\vdots \\ \mathbf{a}^n &= t_{1n}\mathbf{u}^1 + t_{2n}\mathbf{u}^2 + \cdots + t_{nn}\mathbf{u}^n \\ \mathbf{a}^{n+1} &= t_{1,n+1}\mathbf{u}^1 + \cdots + t_{n,n+1}\mathbf{u}^n + t_{n+1,n+1}\mathbf{u}^{n+1}. \end{aligned}\right\} \qquad (5)$$

Since $\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^n$ form distinct rows of a unimodular matrix $\mathbf{V}$, they are linearly independent. Hence, the diagonal elements $t_{11}, t_{22}, \ldots, t_{nn}$ of $\mathbf{T}$ must be nonzero. Multiply the equations of (5) by any $\mathbf{d} \in D$ and use (4) to get

$$\left.\begin{aligned} t_{11}(\mathbf{du}^1) &= 0 \\ t_{12}(\mathbf{du}^1) + t_{22}(\mathbf{du}^2) &= 0 \\ &\vdots \\ t_{1n}(\mathbf{du}^1) + t_{2n}(\mathbf{du}^2) + \cdots + t_{nn}(\mathbf{du}^n) &= 0 \\ t_{1,n+1}(\mathbf{du}^1) + \cdots + t_{n,n+1}(\mathbf{du}^n) + t_{n+1,n+1}(\mathbf{du}^{n+1}) &> 0. \end{aligned}\right\}$$

Since $t_{11}, t_{22}, \ldots, t_{nn}$ are all nonzero and $t_{n+1,n+1} \geq 0$, this implies

$$\mathbf{du}^1 = 0,\ \mathbf{du}^2 = 0, \ldots,\ \mathbf{du}^n = 0,\ \mathbf{du}^{n+1} > 0,$$

that is, $\mathbf{dU} >_{n+1} \mathbf{0}$, for each $\mathbf{d} \in D$. This completes the proof. ■

Example 2 Consider a loop nest $\mathbf{L} = (L_1, L_2, L_3)$ whose distance matrix $\mathbf{D}$ and its transpose $\mathbf{D}'$ are given by

$$\mathbf{D} = \begin{pmatrix} 6 & 4 & 2 \\ 0 & 1 & -1 \\ 1 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{D}' = \begin{pmatrix} 6 & 0 & 1 \\ 4 & 1 & 0 \\ 2 & -1 & 1 \end{pmatrix}.$$

Here $m = 3$. By Algorithm 1, find two matrices

$$\mathbf{V} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & -2 \\ 1 & -1 & -1 \end{pmatrix} \quad \text{and} \quad \mathbf{S} = \begin{pmatrix} 2 & -1 & 1 \\ 0 & 3 & -2 \\ 0 & 0 & 0 \end{pmatrix},$$

such that $\mathbf{V}$ is unimodular, $\mathbf{S}$ is echelon, and $\mathbf{VD}' = \mathbf{S}$. Then $\rho = 2$ and $n = 1$. Since the bottom row of $\mathbf{S}$ is zero, it follows that the bottom row $(1, -1, -1)$ of $\mathbf{V}$ is orthogonal to each column of $\mathbf{D}'$, that is, to each distance vector $\mathbf{d}$ of $\mathbf{L}$. This will be the first column of a 2×3 matrix $\mathbf{A}$ that is being constructed.

Next, one needs a vector $\mathbf{u}$ such that $\mathbf{du} > 0$, or equivalently, $\mathbf{du} \geq 1$ for each distance vector $\mathbf{d}$. The set of inequalities to be satisfied is:

$$\left.\begin{array}{rcrcrcl} 6u_1 & + & 4u_2 & + & 2u_3 & \geq & 1 \\ & & u_2 & - & u_3 & \geq & 1 \\ u_1 & & & + & u_3 & \geq & 1 \end{array}\right\}$$

or

$$\left.\begin{array}{lcl} u_2 & \geq & 1 + u_3 \\ u_1 & \geq & (1 - 4u_2 - 2u_3)/6 \\ u_1 & \geq & 1 - u_3. \end{array}\right\}$$

Algorithm 3 returns the vector $\mathbf{u} = (1, 1, 0)$. This will be the second column of the matrix $\mathbf{A}$. Thus,

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ -1 & 1 \\ -1 & 0 \end{pmatrix}.$$

By Algorithm 2, find two matrices

$$\mathbf{U} = \begin{pmatrix} -1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{T} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix},$$

such that $\mathbf{U}$ is unimodular, $\mathbf{T}$ is echelon (with a nonnegative diagonal element on second row), and $\mathbf{A} = \mathbf{UT}$. Since

$$(6, 4, 2)\mathbf{U} = (0, 10, 6), \ (0, 1, -1)\mathbf{U} = (0, 1, 0), \text{ and } (1, 0, 1)\mathbf{U} = (0, 1, 1)$$

are all positive vectors, the transformation $(L_1, L_2, L_3) \Rightarrow (L_1', L_2', L_3')$ induced by $\mathbf{U}$ is valid. The distance vectors of (L_1', L_2', L_3') are $(0, 10, 6)$, $(0, 1, 0)$ and $(0, 1, 1)$. Since the loops L_1' and L_3' do not carry a dependence, they can run in parallel.

Echelon Transformation

Some background needs to be prepared before echelon transformations can be defined. Let N denote the number of distance vectors of the loop nest $\mathbf{L}$ of Fig. 3, and $\mathbf{D}$ its distance matrix. Apply Algorithm 2 to the $N \times m$ matrix $\mathbf{D}$ to find an $N \times N$ unimodular matrix $\mathbf{V}$ and an $N \times m$ echelon matrix $\mathbf{S} = (s_{tr})$ with nonnegative rows, such that $\mathbf{D} = \mathbf{VS}$. Let ρ denote the number of positive rows of $\mathbf{S}$, so that $\rho = \text{rank}(\mathbf{D})$.

The top ρ rows of $\mathbf{S}$ are positive rows and the bottom $(N - \rho)$ rows are zero rows. Let $\widehat{\mathbf{S}}$ denote the $\rho \times m$ submatrix of $\mathbf{S}$ consisting of the positive rows, and $\widehat{\mathbf{V}}$ the $N \times \rho$ submatrix of $\mathbf{V}$ consisting of the leftmost ρ columns. Then, $\mathbf{D}$ can be written in the form:

$$\mathbf{D} = \widehat{\mathbf{V}}\widehat{\mathbf{S}}. \tag{6}$$

Lemma 1 *The rows of $\widehat{\mathbf{V}}$ are positive vectors.*

Proof Let $\mathbf{v}$ denote any row of $\widehat{\mathbf{V}}$. Then $\mathbf{v}\widehat{\mathbf{S}} = \mathbf{d}$, where $\mathbf{d}$ is a distance vector. Since $\mathbf{d}$ must be positive, $\mathbf{v}$ cannot be the zero vector. Hence, it has the form $\mathbf{v} = (0, \ldots, 0, v_t, \ldots, v_\rho)$, where $1 \leq t \leq \rho$ and $v_t \neq 0$. Let $\ell = \ell_t$ denote the column number of the leading (first nonzero) element on row t of $\widehat{\mathbf{S}}$. Then, $s_{t\ell} > 0$, column ℓ of $\widehat{\mathbf{S}}$ has the form $(s_{1\ell}, s_{2\ell}, \ldots, s_{t\ell}, 0, \ldots, 0)$, and each column j for $1 \leq j < \ell$ has the form $(s_{1j}, s_{2j}, \ldots, s_{t-1,j}, 0, \ldots, 0)$. Hence, the product $\mathbf{v}\widehat{\mathbf{S}}$ has the form $(0, \ldots, 0, v_t s_{t\ell}, \ldots)$. Now, $v_t s_{t\ell} \neq 0$ since $v_t \neq 0$ and $s_{t\ell} > 0$. However, being the leading element of a distance vector, $v_t s_{t\ell}$ must be positive. This means $v_t > 0$, since $s_{t\ell} > 0$. Thus, $\mathbf{v}$ is a positive vector. □

Lemma 2 *For $1 \leq t \leq \rho$, let ℓ_t denote the column number of the leading element on row t of $\widehat{\mathbf{S}}$. For each $\boldsymbol{I} \in \boldsymbol{Z}^m$, there exists a unique $\boldsymbol{Y} \in \boldsymbol{Z}^m$ and a unique $\boldsymbol{K} \in \boldsymbol{Z}^\rho$, such that*

$$0 \leq Y_{\ell_t} \leq s_{t\ell_t} - 1 \quad (1 \leq t \leq \rho) \tag{7}$$

and

$$\boldsymbol{I} = \boldsymbol{Y} + \boldsymbol{K}\widehat{\boldsymbol{S}}. \tag{8}$$

Proof Note that the leading elements $s_{t\ell_t}$ are all positive by construction. Let $\mathbf{I} = (I_1, I_2, \ldots, I_m) \in \mathbf{Z}^m$. Define $\mathbf{K} = (K_1, K_2, \ldots, K_\rho) \in \mathbf{Z}^\rho$ by

$$\begin{aligned} K_1 &= \lfloor I_{\ell_1} / s_{1\ell_1} \rfloor \\ K_2 &= \lfloor (I_{\ell_2} - s_{1\ell_2} K_1) / s_{2\ell_2} \rfloor \\ &\vdots \\ K_\rho &= \lfloor (I_{\ell_\rho} - s_{1\ell_\rho} K_1 - s_{2\ell_\rho} K_2 - \cdots - s_{\rho-1,\ell_\rho} K_{\rho-1}) / s_{\rho\ell_\rho} \rfloor. \end{aligned}$$

After defining $\mathbf{K}$, define $\mathbf{Y} \in \mathbf{Z}^m$ by $\mathbf{Y} = \mathbf{I} - \mathbf{K}\widehat{\mathbf{S}}$. Then $\mathbf{Y}$ and $\mathbf{K}$ are well defined and (8) holds.

For any real number x, one has $0 \le x - \lfloor x \rfloor < 1$. If I and s are integers with $s > 0$, then $0 \le I/s - \lfloor I/s \rfloor < 1$ implies $0 \le I - s\lfloor I/s \rfloor \le s - 1$. Hence, if $I = Y + s\lfloor I/s \rfloor$, then $0 \le Y \le s - 1$. Since (8) implies

$$\begin{aligned} I_{\ell_1} &= Y_{\ell_1} + s_{1\ell_1} K_1 \\ I_{\ell_2} - s_{1\ell_2} K_1 &= Y_{\ell_2} + s_{2\ell_2} K_2 \\ &\vdots \\ I_{\ell_\rho} - s_{1\ell_\rho} K_1 - s_{2\ell_\rho} K_2 - \cdots - s_{\rho-1,\ell_\rho} K_{\rho-1} &= Y_{\ell_\rho} + s_{\rho\ell_\rho} K_\rho, \end{aligned}$$

it follows from the definition of $(K_1, K_2, \ldots, K_\rho)$ that $Y_{\ell_1}, Y_{\ell_2}, \ldots, Y_{\ell_\rho}$ satisfy (7). □

The index variables $I_1, I_2, \ldots, I_m$ of the loop nest of Fig. 3 satisfy the constraints:

$$p_r(I_1, I_2, \ldots, I_{r-1}) \le I_r \le q_r(I_1, I_2, \ldots, I_{r-1}) \qquad (1 \le r \le m).$$

For $I_1, I_2, \ldots, I_m$ in these inequalities, substitute the corresponding expressions in $Y_1, Y_2, \ldots, Y_m, K_1, K_2, \ldots, K_\rho$ from (8). Add (7) to that system to get a system of inequalities in $Y_1, Y_2, \ldots, Y_m, K_1, K_2, \ldots, K_\rho$. Apply Fourier's method to the combined system and eliminate the variables

$$K_\rho, K_{\rho-1}, \ldots, K_1, Y_m, Y_{m-1}, \ldots, Y_1$$

(in this order) to get bounds of the form:

$$\begin{aligned} \alpha_1 &\le Y_1 \le \beta_1 \\ \alpha_2(Y_1) &\le Y_2 \le \beta_2(Y_1) \\ &\vdots \\ \alpha_m(Y_1, Y_2, \ldots, Y_{m-1}) &\le Y_m \le \beta_m(Y_1, Y_2, \ldots, Y_{m-1}) \\ \alpha_{m+1}(Y) &\le K_1 \le \beta_{m+1}(Y) \\ \alpha_{m+2}(Y, K_1) &\le K_2 \le \beta_{m+2}(Y, K_1) \\ &\vdots \\ \alpha_{m+\rho}(Y, K_1, K_2, \ldots, K_{\rho-1}) &\le K_\rho \le \beta_{m+\rho}(Y, K_1, K_2, \ldots, K_{\rho-1}), \end{aligned}$$

where $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_m)$. For each index point $\mathbf{I}$ of $\mathbf{L}$, there is a unique $(m+\rho)$-vector $(\mathbf{Y}, \mathbf{K})$ satisfying these constraints, and conversely. The nest $\mathbf{L}'$ of $(m+\rho)$ loops shown in Fig. 5, where $\mathbf{I} = \mathbf{Y} + \mathbf{K}\widehat{\mathbf{S}}$ and $H'(\mathbf{Y}, \mathbf{K}) = H(\mathbf{I})$, has the same set of iterations as $\mathbf{L}$, but the order of execution is different. The transformation $\mathbf{L} \Rightarrow \mathbf{L}'$ is called the *echelon transformation* of the loop nest $\mathbf{L}$.

Two iterations $H(\mathbf{i})$ and $H(\mathbf{j})$ in $\mathbf{L}$ become iterations $H'(\mathbf{y}, \mathbf{k})$ and $H'(\mathbf{z}, \mathbf{l})$, respectively, in $\mathbf{L}'$, where $\mathbf{i} = \mathbf{y} + \mathbf{k}\widehat{\mathbf{S}}$ and $\mathbf{j} = \mathbf{z} + \mathbf{l}\widehat{\mathbf{S}}$. The transformed nest $\mathbf{L}'$ is *equivalent* to the original nest $\mathbf{L}$, if whenever $H(\mathbf{j})$ depends on $H(\mathbf{i})$ in $\mathbf{L}$, $H'(\mathbf{y}, \mathbf{k})$ precedes $H'(\mathbf{z}, \mathbf{l})$ in $\mathbf{L}'$, that is, $(\mathbf{y}, \mathbf{k}) \prec (\mathbf{z}, \mathbf{l})$.

Theorem 2 *The loop nest $\mathbf{L}'$ obtained from a nest $\mathbf{L}$ by echelon transformation is equivalent to $\mathbf{L}$, and the m outermost loops of $\mathbf{L}'$ can run in parallel.*

Proof To prove the equivalence of $\mathbf{L}'$ to $\mathbf{L}$, consider two iterations $H(\mathbf{i})$ and $H(\mathbf{j})$ of $\mathbf{L}$, such that $H(\mathbf{j})$ depends on $H(\mathbf{i})$. Let $(\mathbf{y}, \mathbf{k})$ denote the value of $(\mathbf{Y}, \mathbf{K})$ corresponding to the value $\mathbf{i}$ of $\mathbf{I}$, and $(\mathbf{z}, \mathbf{l})$ the value corresponding to $\mathbf{j}$. Since $\mathbf{d} = \mathbf{j} - \mathbf{i}$ is a distance vector of $\mathbf{L}$, it follows from (6) that $\mathbf{d} = \mathbf{v}\widehat{\mathbf{S}}$ for some row $\mathbf{v}$ of $\widehat{\mathbf{V}}$.

$$\begin{aligned} L'_1: &\quad \textbf{do } Y_1 = \alpha_1, \beta_1 \\ &\quad\ \vdots \\ L'_m: &\quad\ \ \textbf{do } Y_m = \alpha_m, \beta_m \\ L'_{m+1}: &\quad\ \ \ \ \textbf{do } K_1 = \alpha_{m+1}, \beta_{m+1} \\ &\quad\ \ \ \ \ \ \vdots \\ L'_{m+\rho}: &\quad\ \ \ \ \ \ \textbf{do } K_\rho = \alpha_{m+\rho}, \beta_{m+\rho} \\ &\quad\ \ \ \ \ \ \ \ H'(\mathbf{Y}, \mathbf{K}) \end{aligned}$$

Loop Nest Parallelization. Fig. 5 Echelon Transformation

Then

$$\mathbf{j} = \mathbf{i} + \mathbf{d} = \mathbf{y} + \mathbf{k}\widehat{\mathbf{S}} + \mathbf{v}\widehat{\mathbf{S}} = \mathbf{y} + (\mathbf{k} + \mathbf{v})\widehat{\mathbf{S}}.$$

Since $\mathbf{y}$ satisfies (7), $(\mathbf{y}, \mathbf{k}+\mathbf{v})$ is the image of $\mathbf{j}$ under the mapping $\mathbf{I} \mapsto (\mathbf{Y}, \mathbf{K})$. But the image of $\mathbf{j}$ is also $(\mathbf{z}, \mathbf{l})$ by assumption. Since this mapping is well defined, it follows that $\mathbf{z} = \mathbf{y}$ and $\mathbf{l} = \mathbf{k} + \mathbf{v}$. Hence,

$$(\mathbf{z}, \mathbf{l}) - (\mathbf{y}, \mathbf{k}) = (\mathbf{z} - \mathbf{y}, \mathbf{l} - \mathbf{k}) = (\mathbf{0}, \mathbf{v}).$$

Since $\mathbf{v}$ is positive by Lemma 2, this implies $(\mathbf{y}, \mathbf{k}) \prec (\mathbf{z}, \mathbf{l})$. Hence, $\mathbf{L}'$ is equivalent to $\mathbf{L}$ by definition.

It is clear from the above discussion that the distance vectors of the nest $\mathbf{L}'$ are of the form $(\mathbf{0}, \mathbf{v})$, where $\mathbf{v}$ is a row of $\widehat{\mathbf{V}}$. This means the m outermost Y-loops carry no dependence, and hence they can run in parallel. □

Corollary 1 *In $\boldsymbol{L}'$, the distance matrix of the innermost nest of $\boldsymbol{K}$-loops is $\widehat{\boldsymbol{V}}$.*

Example 3 Consider the loop nest $\mathbf{L}$:

L_1 : **do** $I_1 = 1, 100$
L_2 : **do** $I_2 = 1, 200$
L_3 : **do** $I_3 = I_1, 300$
$X(I_1, I_2, I_3) = X(I_1 - 1, I_2 - 3, I_3) +$
$X(I_1 - 2, I_2 - 10, I_3) + X(I_1 - 1, I_2 + 1, I_3)$

whose distance matrix is

$$\mathbf{D} = \begin{pmatrix} 1 & 3 & 0 \\ 2 & 10 & 0 \\ 1 & -1 & 0 \end{pmatrix}.$$

By Algorithm 2, find two matrices

$$\mathbf{V} = \begin{pmatrix} 1 & 1 & 0 \\ 2 & 3 & 1 \\ 1 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{S} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

such that $\mathbf{V}$ is unimodular, $\mathbf{S}$ is echelon, and $\mathbf{D} = \mathbf{VS}$. The leading elements of nonzero rows of $\mathbf{S}$ are positive. It is clear that $\rho = \text{rank}(\mathbf{D}) = \text{rank}(\mathbf{S}) = 2$. The submatrix $\widehat{\mathbf{S}}$ of $\mathbf{S}$ consisting of its nonzero rows is given by

$$\widehat{\mathbf{S}} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 4 & 0 \end{pmatrix}.$$

Here $s_{1\ell_1} = 1$ and $s_{2\ell_2} = 4$. Define a mapping $(I_1, I_2, I_3) \mapsto (Y_1, Y_2, Y_3, K_1, K_2)$ of the index space of $\mathbf{L}$ into $\mathbf{Z}^5$ by the equation

$$(I_1, I_2, I_3) = (Y_1, Y_2, Y_3) + (K_1, K_2) \cdot \begin{pmatrix} 1 & -1 & 0 \\ 0 & 4 & 0 \end{pmatrix}, \quad (9)$$

and the constraints

$$0 \le Y_1 \le 0 \quad \text{and} \quad 0 \le Y_2 \le 3. \quad (10)$$

Equation (9) is equivalent to the system:

$$\left.\begin{aligned} I_1 &= Y_1 + K_1 \\ I_2 &= Y_2 - K_1 + 4K_2 \\ I_3 &= Y_3. \end{aligned}\right\} \quad (11)$$

Substituting for I_1, I_2, I_3 from (11) into the constraints defining the loop limits of $\mathbf{L}$, one gets the following set of inequalities:

$$\left.\begin{aligned} 1 \le\ & Y_1 + K_1 && \le 100 \\ 1 \le\ & Y_2 - K_1 + 4K_2 && \le 200 \\ Y_1 + K_1 \le\ & Y_3 && \le 300. \end{aligned}\right\} \quad (12)$$

Eliminate the variables K_2, K_1, Y_3, Y_2, Y_1 from (10) and (12) by Fourier's method:

$$\left.\begin{aligned} \lceil (1 - Y_2 + K_1)/4 \rceil \le\ & K_2 && \le \lfloor (200 - Y_2 + K_1)/4 \rfloor \\ 1 - Y_1 \le\ & K_1 && \le \min(100 - Y_1, Y_3 - Y_1) \\ 1 \le\ & Y_3 && \le 300 \\ 0 \le\ & Y_2 && \le 3 \\ 0 \le\ & Y_1 && \le 0. \end{aligned}\right\}$$

Using the fact that $Y_1 = 0$, simplify expressions and get the following loop nest $\mathbf{L}'$ equivalent to $\mathbf{L}$:

L_2 : **do** $Y_2 = 0, 3$
L_3 : **do** $Y_3 = 1, 300$
L_4 : **do** $K_1 = 1, \min(100, Y_3)$
L_5 : **do** $K_2 = \lceil (1 - Y_2 + K_1)/4 \rceil,$
$\lfloor (200 - Y_2 + K_1)/4 \rfloor$
$X(K_1, Y_2 - K_1 + 4K_2, Y_3) =$
$X(K_1 - 1, Y_2 - K_1 + 4K_2 - 3, Y_3) +$
$X(K_1 - 2, Y_2 - K_1 + 4K_2 - 10, Y_3) +$
$X(K_1 - 1, Y_2 - K_1 + 4K_2 + 1, Y_3)$

In $\mathbf{L}'$, the Y_2 and the Y_3 loops can run in parallel. (The Y_1-loop with a single iteration has been omitted.)

The distance matrix of the nest of $\mathbf{K}$-loops is the submatrix consisting of the two leftmost columns of $\mathbf{V}$, that is, the matrix:

$$\widehat{\mathbf{V}} = \begin{pmatrix} 1 & 1 \\ 2 & 3 \\ 1 & 0 \end{pmatrix}.$$

Since the K_2-loop carries no dependence, it can run in parallel.

Remark 2 When $\rho = m$, no valid unimodular transformation can give a parallel outer loop (see Corollary 1 to Theorem 3.8 in [3]). A simple example would be a nest of two loops with two distance vectors $(2, 0)$ and $(0, 3)$, for which $\rho = 2 = m$. However, echelon transformation of this nest will yield a parallel Y_1-loop with 2 iterations and a parallel Y_2-loop with 3 iterations.

If both transformations are available, the best strategy would be to apply the echelon transformation first to get a nest of m parallel outermost Y-loops and ρ sequential innermost K-loops. Then apply Algorithm 3 to the nest of K-loops to get one sequential outermost loop followed by $(\rho - 1)$ parallel inner loops.

Related Entries

►Code Generation
►Parallelism Detection in Nested Loops, Optimal
►Parallelization, Automatic
►Unimodular Transformations

Bibliographic Notes and Further Reading

Unimodular Transformation. Research on this topic goes back to Leslie Lamport's paper in 1974 [8], although he did not use this particular term. This essay is based on the theory of unimodular transformations as developed in the author's books on loop transformations [2, 3]. The general theory grew out of the theory of unimodular transformations of double loops described in [1]. (Watch for some notational differences between these references and the current essay.) The paper by Michael Wolf and Monica Lam [13] covers many useful aspects of unimodular transformations. See also Michael Dowling [5], Erik H. D'Hollander [4], and François Irigoin and Rémi Triolet [6, 7].

Consider a sequential loop nest where each iteration (other than the first) depends on the previous one. The Hyperplane method would still transform it into a nest where the outermost loop is sequential and all inner loops are parallel. There is no great surprise here. It simply means that each inner loop in the new nest will have a single iteration. (Such a loop can run in parallel according to the definition given.) See [13] for the case when there are too many distance vectors.

Echelon Transformation. In his PhD thesis [9], David Padua developed the greatest common divisor method for finding a partition of the index space. Peir and Cytron [10] described a partition based on minimum dependence distances. Shang and Fortes [12] offered an algorithm for finding a maximal independent partition. D'Hollander's paper [4] builds on and incorporates these approaches. The general theory of echelon transformations presented here is influenced by these works. See also Polychronopoulos [11].

See [3] for details on unimodular, echelon, and other transformations used in loop nest parallelization, and more references.

Bibliography

1. Banerjee U (1990) Unimodular transformations of double loops. In: Proceedings of the third workshop on languages and compilers for parallel computing, Irvine, 1–3 August 1990. Available as Nicolau A, Gelernter D, Gross T, Padua D (eds) (1991) Advances in languages and compilers for parallel computing. MIT, Cambridge, pp 192–219
2. Banerjee U (1993) Loop transformations for restructuring compilers: the foundations. Kluwer Academic, Norwell
3. Banerjee U (1994) Loop transformations for restructuring compilers: loop parallelization. Kluwer Academic, Norwell
4. D'Hollander EH (July 1992) Partitioning and labeling of loops by unimodular transformations. IEEE Trans Parallel Distrib Syst 3(4):465–476
5. Dowling ML (Dec 1990) Optimal code parallelization using unimodular transformations. Parallel Comput 16(2–3):157–171
6. Irigoin F, Triolet R (Jan 1988) Supernode partitioning. In: Proceedings of 15th annual ACM SIGACT-SIGPLAN symposium on principles of programming languages, San Diego, pp 319–329
7. Irigoin F, Triolet R (1989) Dependence approximation and global parallel code generation for nested loops. Parallel Distrib Algorithms (Cosnard M et al. eds), Elsivier (North-Holland), New York, pp 297–308

8. Lamport L (Feb 1974) The parallel execution of DO loops. Commun ACM 17(2):83–93
9. Padua DA (Nov 1979) Multiprocessors: discussions of some theoretical and practical problems. PhD thesis, Report 79-990, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana
10. Peir J-K, Cytron R (Aug 1989) Minimum distance: a method for partitioning recurrences for multiprocessors. IEEE Trans Comput C-38(8):1203–1211
11. Polychronopoulos CD (Aug 1988) Compiler optimizations for enhancing parallelism and their impact on architecture design. IEEE Trans Comput C-37(8):991–1004
12. Shang W, Fortes JAB (June 1991) Time optimal linear schedules for algorithms with uniform dependences. IEEE Trans Comput 40(6):723–742
13. Wolf ME, Lam MS (Oct 1991) A loop transformation theory and an algorithm to maximize parallelism. IEEE Trans Parallel Distrib Syst 2(4):452–471

Loop Tiling

►Tiling

Loops, Parallel

Roland Wismüller
University of Siegen, Siegen, Germany

Synonyms

Doall loops; Forall loops

Definition

Parallel loops are one of the most widely used concepts to express parallelism in parallel languages and libraries. In general, a parallel loop is a loop whose iterations are executed at least partially concurrently by several threads or processes. There are several different kinds of parallel loops with different semantics, which will be discussed below. The most prominent kind is the Doall loop, where the iterations are completely independent.

Discussion

Introduction

In a task parallel program, where the execution of different pieces of code is distributed to parallel processors, there are two principal ways of specifying parallel activities. The first one is to specify several *different* code regions, i.e., tasks, which should be executed in parallel. This construct is typically called *parallel regions* or *parallel case*. It offers only a very limited scalability, since the maximum degree of parallelism is determined by the number of regions. The second way is to use *parallel loops*. In a parallel loop, the parallel processors execute the *same* code region, namely, the loop body, but with different data. Thus, parallel loops are a special kind of SPMD programming. Typically, parallel loops are used within a shared memory programming model, for example, OpenMP and Intel's Threading Building Blocks. However, they have also been included in some distributed memory programming models, for example, Occam.

In a sequential loop, the loop body is executed for each element in the loop's index domain in a fixed order implied by the domain. For example, in

```
for (i=0; i<N; i++)
    s[i] = sin(PI*i/N);
```

the assignments to `s` occur in the order `s[0]`, `s[1]`, ..., `s[N]`, although there is obviously no particular reason which mandates this order of the loop iterations. In a parallel loop, the restrictions on the ordering of the loop iterations are relaxed, which allows the iterations to execute – at least partially – in parallel. In the example, all iterations can execute in parallel, since they are completely independent of each other.

Types of Parallel Loops

Today's parallel programming interfaces offer different constructs for parallel loops, which typically fall into one of the following classes which are described by Polychronopoulos [7] and Wolfe [10]: First, there are parallel loops which can be viewed as sequential loops extended by additional properties. A *Doall loop* assures that its iterations can be executed completely independently, while a *Doacross loop* may contain forward dependences. These loops can also be executed sequentially without changing their semantics. In addition, some parallel languages offer a *Forall loop*; however, the term is used with a nonuniform meaning. For example, the Forall loop in Vienna Fortran [3] is actually a Doall loop, while in High Performance Fortran (HPF)