J Supercomput (2012) 60:4–30
DOI 10.1007/s11227-009-0280-y

Communication-free data alignment for arrays with exponential references in parallelizing compilers for scalable parallel systems

Minyi Guo · Weng-Long Chang · Bo Jiang · Shu-Chien Huang · Sien-Tang Tsai · Michael (Shan-Hui) Ho

Published online: 25 March 2009

Abstract In loops, some arrays are referenced with induction variables. To parallelize such kind of loops, those induction variables should be substituted. Thus, those array references that were substituted are formulated as *nonlinear* expressions. The goal of data alignment is to intelligently map the computations and data onto a set

M. Guo (✉) · B. Jiang
School of Information Engineering, Dalian Maritime University, Dalian, Liaoning 116026, China
e-mail: minyi@u-aizu.ac.jp

B. Jiang
e-mail: newdlhot@163.com

W.-L. Chang
Department of Computer Science and Information Engineering, National Kaohsiung University of Applied Sciences, 415 Chien Kung Road, Kaohsiung 807, Taiwan, ROC
e-mail: changwl@cc.kuas.edu.tw

M. Guo
School of Computer Science and Engineering, The University of Aizu, Aizu-Wakamatsu City, Fukushima 965-8580, Japan

S.-C. Huang
Department of Computer Science, National PingTung University of Education, PingTung, Taiwan, ROC
e-mail: schuang@mail.npue.edu.tw

S.-T. Tsai
Department of Information Management, Southern Taiwan University of Technology, Tainan County 710, Taiwan, ROC
e-mail: tsai@mail.stut.edu.tw

M.(S.-H.) Ho
Department of Information Management, School of Information Technology, Ming Chuan University, 5, Teh-Ming Rd., Gwei-Shan, Taoyuan 333, Taiwan, ROC
e-mail: MHoInCerritos@yahoo.com

of virtual processors which are organized as a Cartesian grid (or a *template* in HPF terms), and to provide data locality for parallelizing compilers so that data access communication costs can be minimized. Most data alignment methods are mainly devised to align the referenced arrays using linear subscripts or quadratic subscripts with n loop index variables, and the methods are well developed. Seldom work, however, is researched on the nonlinear expressions of index variables. This paper proposes a new communication-free data alignment technique to align the referenced arrays using *exponential* subscripts with n loop index variables or other complex nonlinear expressions. The experimental results using SPEC95FP Benchmarks point out that the techniques proposed in the paper can improve the execution time of the subroutines in these benchmarks.

Keywords Parallelizing compiler · Communication-free alignment · Communication optimization · Loop optimization · Data dependence analysis · Load balancing

1 Introduction

For scientific and engineering applications, scalable parallel systems based on distributed memory multicomputers have been increasingly applied [1, 5, 16, 17, 37]. In such systems, the programmers (or compilers) must be responsible for distributing the computations and data in a program over processors and manage communications among tasks. Thus, carefully arranging the computations and data locality in a program can improve its throughput. This allows us to determine which computations are to be distributed onto which processors and what data should be stored locally for the corresponding computations to access with little or no communication cost [5, 32]. As the multicore architectures are being used and developed widely by vendors and academia, the data alignment issues become a hot topic in multicore parallel programming [9, 11, 12, 34, 38].

Induction variable is a scalar integer variable, which is used in a loop to simulate do-variables: it is incremented or decremented by a constant amount with each iteration. Every induction variable can be replaced by a linear or a nonlinear function in the form of do-variables. The transformation is called *induction variable substitution*. Consider a typical do-loop shown in Fig. 1(a), which is frequently used in some real scientific computations such as FFT transform, where K is an induction variable in the two-nested loop. If the induction variable substitution is performed for K, the result after the transformation can be observed as shown in Fig. 1(b). In Fig. 1(b), each array reference contains two raised to the power of the outer loop index I. No existing data alignment method can be applied to solve the problem of communication-free data alignment for the case in Fig. 1(b). Therefore, an efficient and precise method for solving the problem of communication-free data alignment for arrays with *exponential* references is highly important.

In this paper, we offer the alignment techniques to properly map the loop iteration space that implies the computation instances, to the virtual processors. The array elements are respectively referred using exponential subscripts of multiple loop index variables so that no or little communication cost for data accesses is yielded. Based

```
K=2
DO I = 1, N, 1
   DO J =1, 2**I, 1
      K = K + 1
S:   X(K) = Y(K) * Z(K)
   ENDDO
ENDDO
```

Figure 1(a): A do-loop

```
K=2
DO I = 1, N, 1
   DO J =1, 2**I, 1
      K= 2 ** I + J
S:    X(2 ** I + J) = Y(2 ** I + J) * Z(2 ** I + J)
   ENDDO
ENDDO
```

Figure 1(b): After an induction variable substitution for the induction variable K

Fig. 1 A sample do-loop of a FORTRAN program

on elementary linear algebra operations, our alignment methods reduce the mapping problem of the computations and array elements into the problem of determining a null space basis for a matrix. By solving this null space basis, the presented methods can readily decipher the desired mapping functions.

The rest of this paper is organized as follows: in Sect. 2, the main communication-free data alignment notions are introduced and existing famous methods for solving the problem of communication-free data alignment are also briefly described. The theoretical explanations and practical applications of our data alignment techniques are proposed in Sect. 3. The experimental results which reveal that our method can be used to improve the execution time of the subroutines in FFT programs in TFFT2 Benchmark in SPEC95FP Benchmark [18, 29, 33] are presented in Sect. 4. Finally, brief conclusions are drawn in Sect. 5.

2 Background

The approach to solving the problem of general communication-free data alignment has been developed [4, 17, 20, 23, 30]. Most of the methods are based on elementary linear algebra. Existing methods and notions of communication-free data alignment problem are briefly introduced in this section.

2.1 Preliminary data alignment notion

To properly allocate computations and data in a program over multiple processors usually involves two phases called *alignment* and *distribution*. First, the alignment phase intelligently maps computations and data onto a set of virtual processors, which are organized into a Cartesian grid (or a *template* in HPF term), to provide data locality in a program. The distribution phase then folds the virtual processors to the physical processors according to feasible distribution strategies. Our primary concern in this paper is the *alignment*.

In general, the complete communication-free data alignment framework consists of three primary phases in terms of elementary linear algebra [2]. Firstly, identify the constraints on the data mapping and computation. In this phase, the data accesses in a program are inspected and formulated as a system of equations in which the unknowns can be utilized to compute the virtual processors for the computations and data to be mapped onto. Each equation in the system is actually equal to a constraint on the data mapping and computation. Any solution to the system figures out

a so-called communication-free data alignment, which enables the required data elements for a processor to perform a computation to be mapped onto its local memory. Through this, no communication cost owing to data access occurs, and thus optimizes the data locality in a program [32]. Different data access patterns, such as array subscript patterns, will produce various equation systems. Therefore, the data alignment framework must take data access patterns into account to achieve a communication-free data alignment.

Accordingly, the second phase of the framework is to distinguish the constraints (or equations) that need to be intentionally left unsatisfied to reserve parallelism in the computations. Allowing unsatisfied constraints will result in communication costs. Hence, the constraints left unsatisfied should be those that generate as little communication as possible. Finally, in the third phase, the remaining constraints are solved to decipher the computation and data mapping functions. Solving the remaining constraints in terms of linear algebra is equal to determining the null space basis for a matrix. The programmer eventually made basis on the alignment results requirements to provide codes ahead of the nested loops for replicating or broadcasting data onto the processors so that no further communication is needed within the nested loops.

2.2 Existing known methods for solving the alignment problem

Many researchers over the past several years have paid attention to maximize parallelism and minimize the communication cost for any given program run on parallel multiprocessors. For distributed memory parallel machines, the problem on data space of communication-free partition along hyperplanes was considered by Ramanujam and Sadayappan [32]. For this problem, a matrix-based formulation was proposed to identify the existence of communication-free partitions for data arrays. An algorithm based on Gaussian elimination was introduced by Feautrier [10], which computes a placement function for the problem of data and code distributions among the processors. When mapping affine loop nests onto distributed memory parallel computers, for the data and computation alignment problem, an access graph to model affine communications more adequately was presented by Dion and Yves [7]. For a program with nested loops, data access pattern analyzing approaches was proposed by Lam et al. [24, 25] and was used to enable the program to run on a parallel machine in a communication-free manner with certain constraints. On the other hand, for multidimensional redistribution, Guo et al. [13–15] focused on the automatic generation of communication routines. For multistatements in perfect and imperfect loops, Shih et al. [35] confirmed the problem on communication-free partition statement-iteration and data spaces along hyperplanes. The necessary and sufficient conditions for communication-free hyperplane partitions were proposed. If the communication cost for performing do-loops is larger than a threshold value, as noted by Lee [26], it was proven that data redistribution is necessary for executing a sequence of do-loops. An expression-rewriting framework to produce communication sets for arrays in loops with block-cyclic distribution was presented by Hwang and Lee [21]. Efficient methods were offered by Hsu et al. [19, 20] and were used to finish the block-cyclic array redistribution which allows a processor not to construct send/receive data sets for a redistribution. Ozcan et al. proposed a memetic algorithm

(MA) to find the best number of processors and the best data distribution method to be used for each stage of a parallel program [28].

A linear algebra framework proposed by Kandemir et al. [22, 23] automatically figures out the optimal data layouts expressed by hyperplanes for each array reference in a program. An efficient method presented by Boudet et al. [3] solves the alignment problem by considering the alignment and distribution of data arrays while at the same time considering the preservation of parallelism for a given program. An alignment technique offered by Bau et al. [2] was applied to align referenced arrays using linear subscripts with one loop index variable in a communication-free manner. New communication-free alignment methods were proposed by Chu et al. [6] and Chang et al. [4] and were utilized to align the referenced arrays using linear subscripts with three loop index variables.

For array references with *quadratic* subscripts or linear subscripts in a general n do-loops, two new data alignment technologies were proposed by Chang et al. [5] and Wu et al. [36]. The techniques properly map the loop iteration space that implies the computation instances and the array elements, which are respectively referenced onto the virtual processors so that there is no communication cost for data access. Moreover, the proposed alignment techniques do not consider the data dependences. In this approach, there will be two principal advantages. First, the difficulty of alignment can be decreased, enabling our techniques to be applied more broadly than other methods. Second, the original data dependences (if it existed) are likely eliminated or reduced by the resulting alignment function because the dependent iterations and their required data could be mapped onto the same temple element. This fact might benefit the exploitation of parallelism in the distribution phase.

3 The proposed alignment techniques for referenced arrays with exponential subscripts

For referenced arrays in a loop, linear expressions with constant coefficients are the most common subscript patterns. Petersen and Padua [31] pointed out that there are 5,242 linear cases with *symbolic* coefficients, 6,503 *nonlinear* cases, and 4,304 cases with references containing *arrays* in the analyzed Perfect Benchmarks. These were obtained by counting the number of feasible directions of the potential dependences. For data alignment, with our counting criteria for the number of *exponential* cases, which is the number of the nested loops including arrays with *exponential* references, it was discovered by Reilly [33], Paek [29] and Hoeflinger [18] that several important loops in the TFFT2 programs in the SPEC95FP Benchmarks consist of arrays with exponential references after induction variable substitution, scalar expansion transformations, and/or inlining substitutions. These results indicate that the number of arrays with exponential subscripts might be attainable to certain extent. However, no existing method can solve the problem of communication-free data alignment for the arrays referenced using exponential subscripts. Our communication-free data alignment methods for aligning the referenced arrays using exponential subscripts with multiple loop index variables are proposed in this section. To avoid complications, the description for our methods is restricted to formulating and solving the equation

system. For unsatisfied constraints, however, the discussion related is not considered in this study. The features and time complexity analysis to our techniques are provided as well.

3.1 Arrays with exponential references

Assuming that there exist q statements containing t arrays, each with one or more (say m) dimensions, referenced using exponential subscripts enclosed with a general n nested do-loop. In order to align the data elements for multidimensional arrays, a general approach is to use one dimension among others for each array as the alignment basis. The data alignment for multidimensional arrays is considered as the data alignment simply for the adopted dimension of the arrays in the following discussions. Assuming that a reference function for the adopted dimension of an array A_e for $1 \leq e \leq t$ in this common loop is $R_{A_e} = a_{e,1}2^{I_1} + \cdots + a_{e,n}2^{I_n} + b_{e,1}I_1 + \cdots + b_{e,n}I_n + f_e$, where $I_1, I_2, \ldots, I_n$ are index variables of the general loop and $a_{e,1}, \ldots, a_{e,n}, b_{e,1}, \ldots, b_{e,n}$ are coefficients, which in general are integers or fractions in the exponential cases, and f_e is an integer constant. In treating the exponential references, we can extend an iteration vector $\mathbf{i}$ as $\mathbf{i} = [2^{i_1}, \ldots, 2^{i_n}, i_1, \ldots, i_n]^{\mathrm{T}}$, i_v is an index value of I_v for $1 \leq v \leq n$ and T is the transposition operation in the iteration space of this general n nested do-loop. The alignment constraints require the processor performing iteration $\mathbf{i}$, which stands for a computation instance, to own $A_e(R_{A_e})$. From our proposed methods, if there exist two or more *distinct* references (either read or write) to an array, each of the distinct references will be selected as the alignment constraints respectively for this array without considering their data dependences.

Consider an example in Fig. 2, where there is a statement containing three different one-dimensional arrays A, B, and D (i.e., $t = 3$ and $m = 1$). For an iteration vector $\mathbf{i}$ ($\mathbf{i} = [2^i, 2^j, i, j]^{\mathrm{T}}$), the alignment constraint demands that the processor performing the iteration $\mathbf{i}$ must own $A(R_A)$, $B(R_B)$, and $D(R_D)$, where $R_A = 2^I + J$, $R_B = 2^I + J$, and $R_D = 2^I + J$.

Assuming that C is the computation mapping function to map the loop iteration space onto the virtual processors and D_{A_e} is the data mapping function to map the array elements of A_e onto the virtual processors. The alignment problem can be formulated as: Find C and D_{A_e} such that $\forall \mathbf{i} \in$ iteration space of this loop:

$$C(\mathbf{i}) = D_{A_e}(R_{A_e}). \tag{3.1}$$

To map the computations and array elements in a communication-free manner, our alignment methods considered the array subscript patterns that are deemed here as generalized exponential subscripts. Therefore, C and D_{A_e} will be formulated using our technique as follows:

$$R_{A_e} = R'_{A_e}\mathbf{i} + f_e\big(R'_{A_e} = [a_{e,1}, \ldots, a_{e,n}, b_{e,1}, \ldots, b_{e,n}]\big), \tag{3.2}$$

Fig. 2 The FORTRAN do-loop extracted from TFFT2 programs in the SPEC95FP Benchmarks

```
DO I = 1, N, 1
   DO J =1, 2 ** I, 1
S1:    A(2 ** I + J) = B(2 ** I + J) * D(2 ** I + J)
   ENDDO
ENDDO
```

$$C(\mathbf{i}) = C'\mathbf{i} + c_0, \tag{3.3}$$

and

$$D_{A_e}(R_{A_e}) = D'_{A_e}(R_{A_e}) + d_0 = D'_{A_e}(R'_{A_e}\mathbf{i} + f_e) + d_0. \tag{3.4}$$

From (3.3) and (3.4), (3.1) can be converted into the following equation:

$$C'\mathbf{i} + c_0 = D'_{A_e}(R'_{A_e}\mathbf{i} + f_e) + d_0. \tag{3.5}$$

Without loss of generality, (3.5) can be reduced to (3.6):

$$[\,C' \quad c_0\,]\begin{bmatrix}\mathbf{i}\\ 1\end{bmatrix} = [\,D'_{A_e} \quad d_0\,]\begin{bmatrix}R'_{A_e} & f_e\\ 0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{i}\\ 1\end{bmatrix}. \tag{3.6}$$

Let $C = [C'\ c_0]$, $D_{A_e} = [D'_{A_e}\ d_0]$, $F_{A_e} = \left[\begin{smallmatrix}R'_{A_e} & f_e\\ 0 & 1\end{smallmatrix}\right]$ and $\mathbf{i}' = \left[\begin{smallmatrix}\mathbf{i}\\ 1\end{smallmatrix}\right]$, (3.6) can be transformed into the following equation:

$$C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'. \tag{3.7}$$

From (3.7), to determine C and D_{A_e} is to solve the equation $C = D_{A_e}F_{A_e}$ (or $C - D_{A_e}F_{A_e} = \mathbf{0}$) after $\mathbf{i}'$ is eliminated, where $\mathbf{0}$ is a zero matrix. Such an equation can be represented, without loss of generality, in block matrix form [2] as follows:

$$[\,C \quad D_{A_e}\,]\begin{bmatrix}\mathbf{I}\\ -F_{A_e}\end{bmatrix} = \mathbf{0}. \tag{3.8}$$

Here, $\mathbf{I}$ is an identity matrix, and $\mathbf{0}$ (zero matrix), C, D_{A_e} and F_{A_e} are square matrices with the same size as $\mathbf{I}$. By expressing (3.8) in the form of $UV = \mathbf{0}$ and determining a null space basis for VT, the alignment problem is therefore reduced to the standard linear algebra problem of determining a null space basis for a matrix.

Clearly, to construct C, D_{A_e}, and F_{A_e} as square matrices for the nested do-loops with different numbers of loop index variables, (3.3) and (3.4) have to be adjusted. For example, suppose that there is only one index loop variable I_1 for this do-loop, that is, $R_{A_e} = a_{e,1}2^{i_1} + b_{e,1}i_1 + f_e$, then our alignment technique originally formulates $C(\mathbf{i})$ and $D_{A_e}(R_{A_e})$ as follows:

$$C(\mathbf{i}) = c_1 2^{i_1} + c_2 i_1 + c_0$$

and

$$D_{A_e}(R_{A_e}) = d_1\left(a_{e,1}2^{i_1} + b_{e,1}i_1 + f_e\right) + d_0.$$

However, in addition to constructing C and D_{A_e} as square matrices, our alignment technique also considers enabling C and D_{A_e} to be easily determined by constructing F_{A_e} in a form that allows the Gaussian elimination for echelon reduction to be performed effortlessly to simplify the calculation for the aforementioned null space basis. The above equations are subsequently adjusted as follows:

$$C(\mathbf{i}) = (c_{1,1} + c_{2,1} + c_{3,1})2^{i_1} + (c_{1,2} + c_{2,2} + c_{3,2})i_2 + (c_{1,3} + c_{2,3} + c_{3,3})$$

and

$$D_{A_e}(R_{A_e}) = (d_{1,1} + d_{2,1} + d_{3,1})\big(a_{e,1}2^{i_1} + b_{e,1}i_1 + f_e\big) + (d_{1,2} + d_{2,2} + d_{3,2}) + (d_{1,3} + d_{2,3} + d_{3,3})\big(2^{i_1} + i_1 + 1\big).$$

Using our technique, the above equations are reduced to the following equation:

$$\begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} \begin{bmatrix} 2^{i_1} \\ i_1 \\ 1 \end{bmatrix} = \begin{bmatrix} d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,1} & d_{3,2} & d_{3,3} \end{bmatrix} \begin{bmatrix} a_{e,1} & b_{e,1} & f_e \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2^{i_1} \\ i_1 \\ 1 \end{bmatrix}.$$

This makes C and D_{A_e} into 3×3 square matrices. Continuing with this notion, the alignment constraint for the iteration space of this n nested do-loop can be formally expressed, using our technique as:

$$C\mathbf{i}' = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,2n+1} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,2n+1} \\ \vdots & \vdots & \cdots & \vdots \\ c_{2n+1,1} & c_{2n+1,2} & \cdots & c_{2n+1,2n+1} \end{bmatrix} \begin{bmatrix} 2^{i_1} \\ \vdots \\ i_n \\ 1 \end{bmatrix}.$$

The alignment constraint for an array A_e, $1 \le e \le t$, in the general loop can be represented as:

$$D_{A_e}F_{A_e}\mathbf{i}' = \begin{bmatrix} d_{1,1} & \cdots & d_{1,2n+1} \\ \vdots & \ddots & \vdots \\ d_{2n+1,1} & \cdots & d_{2n+1,2n+1} \end{bmatrix} \begin{bmatrix} a_{e,1} & a_{e,2} & \cdots & b_{e,n} & f_e \\ 0 & 0 & \cdots & 0 & 1 \\ 1 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \begin{bmatrix} 2^{i_1} \\ \vdots \\ i_n \\ 1 \end{bmatrix}.$$

Therefore, the alignment problem can be restated as: Find C and D_{A_e} such that $\forall \mathbf{i} \in$ iteration space of this loop:

$$C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'.$$

Here, $\mathbf{i}' = [\mathbf{i}, 1]^{\mathrm{T}}$, as mentioned. The above equation can be reduced to (3.8) to determine C and D_{A_e} as described. This requires the column vector $\mathbf{i}'$ on both sides of the equation to be eliminated to make $(C - D_{A_e}F_{A_e})$ equal to $\mathbf{0}$ for any $\mathbf{i}'$. To do this, we require the following lemma.

Lemma 3.1 *Let P_i be a $p \times 1$ matrix for $1 \le i \le 2n$, $\mathbf{w}$ a p-elements column vector, $\mathbf{0}$ a p-elements zero vector, and y_i a scalar variable for $1 \le i \le 2n$. Then*

$$\forall y_i \quad [P_1 \cdots P_n P_{n+1} \cdots P_{2n} \mathbf{w}] \begin{bmatrix} 2^{y_1} \\ \vdots \\ 2^{y_n} \\ y_1 \\ \vdots \\ y_n \\ 1 \end{bmatrix} = \mathbf{0} \iff P_i = 0$$

for $1 \le i \le 2n$, and $\mathbf{w} = 0$.

Proof

$$[P_1 \cdots P_n P_{n+1} \cdots P_{2n} \mathbf{w}] \begin{bmatrix} 2^{y_1} \\ \vdots \\ 2^{y_n} \\ y_1 \\ \vdots \\ y_n \\ 1 \end{bmatrix} = \mathbf{0}$$

$$\iff \quad \forall y_i \quad \begin{bmatrix} p_{1,1} & \cdots & p_{n,1} & p_{n+1,1} & \cdots & p_{2n,1} & w_1 \\ p_{1,2} & \cdots & p_{n,2} & p_{n+1,2} & \cdots & p_{2n,2} & w_2 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{1,p} & \cdots & p_{n,p} & p_{n+1,p} & \cdots & p_{2n,p} & w_p \end{bmatrix} \begin{bmatrix} 2^{y_1} \\ \vdots \\ 2^{y_n} \\ y_1 \\ \vdots \\ y_n \\ 1 \end{bmatrix} = \mathbf{0}$$

$$\iff \quad \forall y_i \quad \begin{bmatrix} 2^{y_1} p_{1,1} + \cdots + 2^{y_n} p_{n,1} + y_1 p_{n+1,1} + \cdots + y_n p_{2n,1} + w_1 \\ 2^{y_1} p_{1,2} + \cdots + 2^{y_n} p_{n,2} + y_1 p_{n+1,2} + \cdots + y_n p_{2n,2} + w_2 \\ \vdots \\ 2^{y_1} p_{1,p} + \cdots + 2^{y_n} p_{n,p} + y_1 p_{n+1,p} + \cdots + y_n p_{2n,p} + w_p \end{bmatrix} = 0$$

$$\iff \quad \forall y_i \quad 2^{y_1} P_1 + \cdots + 2^{y_n} P_n + y_1 P_{n+1} + \cdots + y_n P_{2n} + \mathbf{w} = \mathbf{0}.$$

We have now proven that

$$\forall y_i \quad 2^{y_1} P_1 + \cdots + 2^{y_n} P_n + y_1 P_{n+1} + \cdots + y_n P_{2n} + \mathbf{w} = \mathbf{0}$$
$$\iff \quad P_1 = \cdots = P_n = P_{n+1} = \cdots = P_{2n} = \mathbf{w} = \mathbf{0}$$

$\Rightarrow$) Assume that there exist certain P_i's $\neq \mathbf{0}$ (e.g., P_q, P_r, P_s, and $P_t \neq \mathbf{0}$) and $\mathbf{w} \neq \mathbf{0}$ and some y_i's $\neq 0$ (e.g., y_q, y_r, y_s, and $y_t \neq 0$) such that

$$2^{y_q} P_q + 2^{y_r} P_r + 2^{y_s} P_s + 2^{y_t} P_t + \mathbf{w} = \mathbf{0}.$$

Because all y_i's can be any value, we have

$$2^{y_q+1} P_q + 2^{y_r} P_r + 2^{y_s} P_s + 2^{y_t} P_t + \mathbf{w} = \mathbf{0}.$$

That is,

$$2^{y_q} P_q + 2^{y_r} P_r + 2^{y_s} P_s + 2^{y_t} P_t + \mathbf{w} + 2^{y_q} P_q = \mathbf{0}.$$

This means that

$$P_q = \mathbf{0}.$$

This contradicts the assumption. Similarly, we have $P_r = \mathbf{0}$, $P_s = \mathbf{0}$, $P_t = \mathbf{0}$ and $\mathbf{w} = \mathbf{0}$. Therefore, we have

$$P_1 = \cdots = P_n = P_{n+1} = \cdots = P_{2n} = \mathbf{w} = \mathbf{0}.$$

$\Leftarrow$) The (if part) is trivial □

From Lemma 3.1, (3.7) can be rewritten as

$$C = D_{A_e} F_{A_e}. \tag{3.9}$$

For $1 \leq e \leq t$, the equation system of (3.9) can be converted into the following matrix equation (3.10):

$$[C\, D_{A_1} \quad \cdots \quad D_{A_t}] \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ -F_{A_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \ddots & & \vdots \\ \vdots & & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & -F_{A_t} \end{bmatrix} = [\mathbf{0} \quad \cdots \quad \mathbf{0}]. \tag{3.10}$$

Here, $\mathbf{I}$ is a $(2n+1) \times (2n+1)$ identity matrix, $\mathbf{0}$ is a $(2n+1) \times (2n+1)$ zero matrix, and $[\mathbf{0} \quad \cdots \quad \mathbf{0}]$ is a $(2n+1) \times ((2n+1) \times t)$ zero matrix.

To solve the matrix equation $[U]_{s \times m}[V]_{m \times n} = [0]_{s \times n}$, in which $[U]_{s \times m}$ is unknown and $[V]_{m \times n}$ is known, we can first transform V into a "rank-revealing" form by performing the required rank-preserving operations—elementary row and column operations. The notion behind this is to obtain a matrix into a form in which its rank can be determined by inspection [2, 8]. One way to achieve this is to perform integer preserving Gaussian elimination [8, 27], wherein the matrix rows or columns are systematically manipulated by elementary row or column operations to yield a matrix in echelon form, thus enabling us to obtain the following factorization (suppose that $V \in Z^{m \times n}$ and $rank(V) = r$):

$$[H]_{m\times m}[V]_{m\times n}[P]_{n\times n} = \begin{bmatrix} R_{1,1} & R_{1,2} \\ 0 & 0 \end{bmatrix}_{m\times n}.$$

Here, H is an $m \times m$ invertible matrix representing the row operations, P is an $n \times n$ unimodular matrix representing the column operations, and $R_{1,1}$ is an $r \times r$ upper triangular invertible matrix. It is a property of this factorization that the transposition of the last $m - r$ rows of H spans the null space of V^{T}. Accordingly, we can obtain the solution for $[U]_{s\times m}$ as follows:

$$U = H(r + 1 : m, 1 : m).$$

This means that only H, the composition of row operations, has to be determined during the elimination.

Considering the example in Fig. 2, the alignment constraint for the iteration space of this loop can be formally expressed as:

$$C\mathbf{i}' = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} \\ c_{5,1} & c_{5,2} & c_{5,3} & c_{5,4} & c_{5,5} \end{bmatrix} \begin{bmatrix} 2^I \\ 2^J \\ I \\ J \\ 1 \end{bmatrix}.$$

The alignment constraints for arrays A, B, and D can be correspondingly represented as:

$$D_A F_A \mathbf{i}' = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} \\ x_{5,1} & x_{5,2} & x_{5,3} & x_{5,4} & x_{5,5} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2^I \\ 2^J \\ I \\ J \\ 1 \end{bmatrix},$$

$$D_B F_B \mathbf{i}' = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} & y_{1,5} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} & y_{2,5} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} & y_{3,5} \\ y_{4,1} & y_{4,2} & y_{4,3} & y_{4,4} & y_{4,5} \\ y_{5,1} & y_{5,2} & y_{5,3} & y_{5,4} & y_{5,5} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2^I \\ 2^J \\ I \\ J \\ 1 \end{bmatrix}$$

and

$$D_D F_D \mathbf{i}' = \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} \\ z_{2,1} & z_{2,2} & z_{2,3} & z_{2,4} & z_{2,5} \\ z_{3,1} & z_{3,2} & z_{3,3} & z_{3,4} & z_{3,5} \\ z_{4,1} & z_{4,2} & z_{4,3} & z_{4,4} & z_{4,5} \\ z_{5,1} & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2^I \\ 2^J \\ I \\ J \\ 1 \end{bmatrix}.$$

The alignment problem can be expressed as follows: Find C, D_A, D_B, and D_D such that $\forall(I, J) \in$ iteration space of this loop:

$$\begin{cases} C = D_A F_A, \\ C = D_B F_B, \\ C = D_D F_D. \end{cases} \tag{3.11}$$

The equation system of (3.11) can be converted into the following matrix equation:

$$[C \quad D_A \quad D_B \quad D_D] \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} \\ -F_A & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -F_B & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -F_D \end{bmatrix} = [\mathbf{0} \quad \mathbf{0} \quad \mathbf{0}]. \tag{3.12}$$

Here, $\mathbf{I}$ is a 5×5 identity matrix and $\mathbf{0}$ is a 5×5 zero matrix. According to the above-mentioned method, a solution matrix of (3.12) is

$$[C \quad D_A \quad D_B \quad D_D] = \begin{bmatrix} 1&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0 \\ 1&1&1&1&1&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0 \\ 1&1&1&1&1&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0 \\ 1&1&1&1&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1 \end{bmatrix}.$$

This gives us

$$C = \begin{bmatrix} 1&0&0&1&0 \\ 0&0&0&0&1 \\ 1&1&1&1&1 \\ 1&1&1&1&1 \\ 1&1&1&1&1 \end{bmatrix}, \quad D_A = \begin{bmatrix} 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \end{bmatrix},$$

$$D_B = \begin{bmatrix} 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \end{bmatrix} \quad \text{and} \quad D_D = \begin{bmatrix} 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \end{bmatrix}.$$

We then obtain the mappings of computations and data as follows:

$$C\mathbf{i}' = \begin{bmatrix} 1&0&0&1&0 \\ 0&0&0&0&1 \\ 1&1&1&1&1 \\ 1&1&1&1&1 \\ 1&1&1&1&1 \end{bmatrix} \begin{bmatrix} 2^I \\ 2^J \\ I \\ J \\ 1 \end{bmatrix} = \begin{bmatrix} 2^I + J \\ 1 \\ 2^I + 2^J + I + J + 1 \\ 2^I + 2^J + I + J + 1 \\ 2^I + 2^J + I + J + 1 \end{bmatrix},$$

$$D_A F_A \mathbf{i}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2^I \\ 2^J \\ I \\ J \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2^I + J \\ 1 \\ 2^I + 2^J + I + J + 1 \\ 2^I + 2^J + I + J + 1 \\ 2^I + 2^J + I + J + 1 \end{bmatrix},$$

$$D_B F_B \mathbf{i}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2^I \\ 2^J \\ I \\ J \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2^I + J \\ 1 \\ 2^I + 2^J + I + J + 1 \\ 2^I + 2^J + I + J + 1 \\ 2^I + 2^J + I + J + 1 \end{bmatrix}$$

and

$$D_D F_D \mathbf{i}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2^I \\ 2^J \\ I \\ J \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2^I + J \\ 1 \\ 2^I + 2^J + I + J + 1 \\ 2^I + 2^J + I + J + 1 \\ 2^I + 2^J + I + J + 1 \end{bmatrix}.$$

Therefore, using our alignment, iteration (I, J) is mapped onto the virtual processor $(2^I + J)$ and the corresponding A, B, and D array elements are mapped into the same virtual processor. The **Align** statements adopted to describe the alignment relation for the array elements among A, B, and D are represented as follows:

$$\text{Align } A\big(2^I + J\big) \text{ with } T\big(2^I + J\big),$$
$$\text{Align } B\big(2^I + J\big) \text{ with } T\big(2^I + J\big), \text{ and}$$
$$\text{Align } D\big(2^I + J\big) \text{ with } T\big(2^I + J\big).$$

Here, the virtual processors are supposed to be organized as a one-dimensional template T. Since no loop-carried output-dependences exist for array A and that the required data elements (the written data element of array A and the corresponding read data element of arrays B and D) for a computation are mapped into the same template element, this nested loop can be executed in parallel without interprocessor communication.

3.2 Array subscripts with $a_{e,1} * J * 2^{L-1} + a_{e,2} * 2^{L-2} + a_{e,3} * K + a_{e,4}$

Assume that there exist q statements containing t arrays of m-dimensions which are referenced using subscripts with the patterns $a_{e,1} * J * 2^{L-1} + a_{e,2} * 2^{L-2} + a_{e,3} * K + a_{e,4}$, where L, J, and K are loop index variables. Suppose that a reference function for the adopted dimension of an array A_e for $1 \leq e \leq t$ in a general loop is $R_{A_e} = a_{e,1} * J * 2^{L-1} + a_{e,2} * 2^{L-2} + a_{e,3} * K + a_{e,4}$, where L, J, and K are index variables of the general loop and $a_{e,1}$, $a_{e,2,}$ and $a_{e,3}$ are coefficients, which in general are integers or fractions in the case, and $a_{e,4}$ is an integer constant.

We consider that two different one-dimensional arrays (i.e., $t = 2$ and $m = 1$), referenced using nonlinear subscripts, are enclosed with a depth-four nested loop in the example shown in Fig. 3. For an iteration vector $\mathbf{i}$ ($\mathbf{i} = [I,\ L,\ J,\ K]^{\mathrm{T}}$), the alignment constraints require the processor performing the iteration $\mathbf{i}$ to own $A(R_A)$ and $B(R_B)$. The primary notion of our method to align referenced arrays using nonlinear subscripts is the same as the previous technique in Sect. 3.1, except that C and D_{A_e} are formulated in different patterns. However, in dealing with the nonlinear cases, more subtle but nontrivial adjustments on (3.3) and (3.4) are required to construct C, D_{A_e}, and F_{A_e} as square matrices for the nested do-loop with different numbers of loop index variables. Similar to the exponential case, the alignment problem can be described as: Find C and D_{A_e} such that $\forall \mathbf{i} \in$ iteration space of this loop:

$$C\mathbf{i}' = D_{A_e} F_{A_e} \mathbf{i}'.$$

In this case, $\mathbf{i} = [I,\ L,\ J,\ K]^{\mathrm{T}}$ and $\mathbf{i}' = [J * 2^{(L-1)}, 2^{(L-2)}, K, 1]^{\mathrm{T}}$. The alignment constraint for the iteration space of this general loop can be formally expressed as

```
DO I = 0, 2**(M – 1)
  DO L = 1, (1 + M) / 2
    DO J = 0, 2**((1 + M – L) – 1)
      DO K = 1, 2**(L – 2)
        …
A(K + J * 2 ** (L – 1) + 2 ** (L – 2)) = B(K + J * 2 ** (L – 1) + 2 ** (L – 2))…
        …
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

Fig. 3 The FORTRAN do-loop extracted from TFFT2 programs in the SPEC95FP Benchmarks

follows, using our alignment technique:

$$C\mathbf{i}' = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{bmatrix} * \begin{bmatrix} J * 2^{(L-1)} \\ 2^{(L-2)} \\ K \\ 1 \end{bmatrix}.$$

The alignment constraint for an array A_e, $1 \le e \le t$, in the general loop can be represented as:

$$D_{A_e} F_{A_e} \mathbf{i}' = \begin{bmatrix} d_{1,1} & d_{1,2} & d_{1,3} & d_{1,4} \\ d_{2,1} & d_{2,2} & d_{2,3} & d_{2,4} \\ d_{3,1} & d_{3,2} & d_{3,3} & d_{3,4} \\ d_{4,1} & d_{4,2} & d_{4,3} & d_{4,4} \end{bmatrix} \begin{bmatrix} a_{e,1} & a_{e,2} & a_{e,3} & a_{e,4} \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} J * 2^{(L-1)} \\ 2^{(L-2)} \\ K \\ 1 \end{bmatrix}.$$

Similar to the discussion in the previous subsection, the column vector $\mathbf{i}'$ on both sides of the equation, $C\mathbf{i}' = D_{A_e} F_{A_e} \mathbf{i}'$, is required to be eliminated for any $\mathbf{i}'$. For this type of loops, we require the following lemma.

Lemma 3.2 *Let P_i be a 4×1 matrix for $1 \le i \le 3$, $\mathbf{w}$ be a four-element column vector, $\mathbf{0}$ be a four-element zero vector, and y_z be a scalar variable for $1 \le z \le 4$. Then*

$$\forall y_z \quad [\, P_1 \quad P_2 \quad P_3 \quad \mathbf{w}\,] \begin{bmatrix} y_3 * 2^{(y_2-1)} \\ 2^{(y_2-2)} \\ y_4 \\ 1 \end{bmatrix} = \mathbf{0} \iff P_i = 0$$

for $1 \le i \le 3$, and $\mathbf{w} = 0$.

Proof

$$[\, P_1 \quad P_2 \quad P_3 \quad \mathbf{w}\,] \begin{bmatrix} y_3 * 2^{(y_2-1)} \\ 2^{(y_2-2)} \\ y_4 \\ 1 \end{bmatrix} = \mathbf{0}$$

$$\iff \quad \forall y_z \quad \begin{bmatrix} p_{1,1} & p_{2,1} & p_{3,1} & w_1 \\ p_{1,2} & p_{2,2} & p_{3,2} & w_2 \\ p_{1,3} & p_{2,3} & p_{3,3} & w_3 \\ p_{1,4} & p_{2,4} & p_{3,4} & w_4 \end{bmatrix} \begin{bmatrix} y_3 * 2^{(y_2-1)} \\ 2^{(y_2-2)} \\ y_4 \\ 1 \end{bmatrix} = \mathbf{0}$$

$$\iff \quad \forall y_z \quad \begin{bmatrix} y_3 * 2^{(y_{21}-1)} * p_{1,1} + 2^{(y_2-2)} p_{2,1} + y_4 * p_{3,1} + w_1 \\ y_3 * 2^{(y_{21}-1)} * p_{1,2} + 2^{(y_2-2)} p_{2,2} + y_4 * p_{3,2} + w_2 \\ \vdots \\ y_3 * 2^{(y_{21}-1)} * p_{1,4} + 2^{(y_2-2)} p_{2,4} + y_4 * p_{3,4} + w_4 \end{bmatrix} = \mathbf{0}$$

$$\iff \quad \forall y_Z \quad y_3 * 2^{(y_2-1)} * P_1 + \cdots + 2^{(y_2-2)} * P_2 + y_4 * P_3 + \mathbf{w} = \mathbf{0}.$$

We have now proven that

$$\forall y_Z \quad y_3 * 2^{(y_2-1)} * P_1 + 2^{(y_2-2)} * P_2 + y_4 * P_3 + \mathbf{w} = \mathbf{0}$$
$$\iff \quad P_1 = P_2 = P_3 = \mathbf{w} = \mathbf{0}$$

$\Rightarrow$) Assume that there exists certain P_i's $\neq \mathbf{0}$ (e.g., P_2 and $P_3 \neq \mathbf{0}$) and $\mathbf{w} \neq \mathbf{0}$, and some y_z's $\neq 0$ (e.g., (y_2 and $y_4 \neq 0$) such that

$$2^{(y_2-2)} * P_2 + y_4 * P_3 + \mathbf{w} = \mathbf{0}.$$

Because all y_z's can be any value, we have

$$2^{(y_2-2)+1} * P_2 + y_4 * P_3 + \mathbf{w} = \mathbf{0}.$$

That is,

$$2^{(y_2-2)} * P_2 + y_4 * P_3 + \mathbf{w} + 2^{(y_2-2)} * P_2 = \mathbf{0}.$$

This means, $P_2 = \mathbf{0}$, which contradicts the assumption. Similarly, we have $P_3 = \mathbf{0}$ and $\mathbf{w} = \mathbf{0}$. Therefore, we have

$$P_1 = P_2 = P_3 = \mathbf{w} = \mathbf{0}.$$

$\Leftarrow$) The (if part) is trivial. □

Again, similar to the linear equation, the equation system in this case can eventually be converted into the following matrix equation:

$$[\,C \quad D_{A_1} \quad \cdots \quad D_{A_t}\,] \begin{bmatrix} \mathbf{I} & \cdots & \mathbf{I} \\ -F_{A_1} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & -F_{A_t} \end{bmatrix} = [\,\mathbf{0} \quad \cdots \quad \mathbf{0}\,].$$

Here, $\mathbf{I}$ is a 4×4 identity matrix, $\mathbf{0}$ is a 4×4 zero matrix, and $[\mathbf{0} \cdots \mathbf{0}]$ is a $4 \times ((4) \times t)$ zero matrix. Note that in solving the above matrix equation, each row with fraction elements in each F_{A_e} can be first multiplied by a factor so that it will only have integer elements. In the example in Fig. 3, the alignment constraint for the iteration space of this loop can be formally expressed as

$$C\mathbf{i}' = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{bmatrix} \begin{bmatrix} y_3 * 2^{(y_2-1)} \\ 2^{(y_2-2)} \\ y_4 \\ 1 \end{bmatrix}.$$

The alignment constraints for arrays A and B can be respectively represented as

$$D_A F_A \mathbf{i}' = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_3 * 2^{(y_2-1)} \\ 2^{(y_2-2)} \\ y_4 \\ 1 \end{bmatrix}$$

and

$$D_B F_B \mathbf{i}' = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} \\ y_{4,1} & y_{4,2} & y_{4,3} & y_{4,4} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_3 * 2^{(y_2-1)} \\ 2^{(y_2-2)} \\ y_4 \\ 1 \end{bmatrix}.$$

The alignment problem can be expressed as follows: Find C, D_A, and D_B such that $\forall(I, L, J, K) \in$ iteration space of this loop:

$$\begin{cases} C = D_A F_A, \\ C = D_B F_B. \end{cases} \tag{3.13}$$

The equations system of (3.13) can be converted into the following matrix equation:

$$[C \quad D_A \quad B_B] \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ -F_A & \mathbf{0} \\ \mathbf{0} & -F_B \end{bmatrix} = [\mathbf{0} \quad \mathbf{0}]. \tag{3.14}$$

Here, $\mathbf{I}$ is a 4×4 identity matrix and $\mathbf{0}$ is a 4×4 zero matrix. According to the method stated in Sect. 3.1, a solution matrix of (3.14) is

$$[C \quad D_A \quad D_B] = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

This gives us

$$C = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad D_A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad D_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We can obtain the mappings of computations and data as follows:

$$C\mathbf{i}' = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} J * 2^{(L-1)} \\ 2^{(L-2)} \\ K \\ 1 \end{bmatrix} = \begin{bmatrix} J * 2^{(L-1)} + 2^{(L-2)} + K \\ 1 \\ J * 2^{(L-1)} + 2^{(L-2)} + K + 1 \\ J * 2^{(L-1)} + 2^{(L-2)} + K + 1 \end{bmatrix},$$

$$D_A F_A \mathbf{i}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} J * 2^{(L-1)} \\ 2^{(L-2)} \\ K \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} J * 2^{(L-1)} + 2^{(L-2)} + K \\ 1 \\ J * 2^{(L-1)} + 2^{(L-2)} + K + 1 \\ J * 2^{(L-1)} + 2^{(L-2)} + K + 1 \end{bmatrix}$$

and

$$D_B F_B \mathbf{i}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} J * 2^{(L-1)} \\ 2^{(L-2)} \\ K \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} J * 2^{(L-1)} + 2^{(L-2)} + K \\ 1 \\ J * 2^{(L-1)} + 2^{(L-2)} + K + 1 \\ J * 2^{(L-1)} + 2^{(L-2)} + K + 1 \end{bmatrix}.$$

Hence, using our alignment, iteration (I, L, J, K) is mapped into the virtual processor $(J * 2 * *(L-1) + 2 * *(L-2) + K)$ and the corresponding A and B array elements are mapped onto the same virtual processor. The **Align** statements adopted to describe the alignment relation for the array elements of both A and B are represented as follows:

$$\begin{aligned} &\text{Align } A\big(J * 2 * *(L-1) + 2 * *(L-2) + K\big) \\ &\quad \text{with } T\big(J * 2 * *(L-1) + 2 * *(L-2) + K\big) \text{ and} \\ &\text{Align } B\big(J * 2 * *(L-1) + 2 * *(L-2) + K\big) \\ &\quad \text{with } T\big(J * 2 * *(L-1) + 2 * *(L-2) + K\big). \end{aligned}$$

Here, the virtual processors are supposedly organized as a one-dimensional template T.

3.3 Features of the proposed techniques

In principle, our techniques have the following features.

Theorem 3.1 *If the alignment function of the template and the reference functions of the written arrays are in common form, i.e., the multiple of the nonconstant items of the former is equal to the multiple of the nonconstant items of the latter, then there are no distributed data updates for writing these arrays.*

Proof We provide the proof for the exponential cases and similarly, the proof for other nonlinear cases can be obtained. We first propose the condition for two dependent iterations to be mapped onto the same template element. Let the alignment constraint for the iteration space of an n nested do-loop be expressed as

$$C\mathbf{i}' = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,2n+1} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,2n+1} \\ \vdots & \vdots & \cdots & \vdots \\ c_{2n+1,1} & c_{2n+1,2} & \cdots & c_{2n+1,2n+1} \end{bmatrix} \begin{bmatrix} 2^{i_1} \\ \vdots \\ 2^{i_n} \\ i_1 \\ \vdots \\ i_n \\ 1 \end{bmatrix}.$$

And the alignment constraint for the written array A_e in the general loop can be represented as

$$D_{A_e}F_{A_e}\mathbf{i}' = \begin{bmatrix} d_{1,1} & \cdots & d_{1,2n+1} \\ \vdots & \ddots & \vdots \\ d_{2n+1,1} & \cdots & d_{2n+1,2n+1} \end{bmatrix} \begin{bmatrix} a_{e,1} & a_{e,2} & \cdots & b_{e,n} & f_e \\ 0 & 0 & \cdots & 0 & 1 \\ 1 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \begin{bmatrix} 2^{i_1} \\ \vdots \\ 2^{i_n} \\ i_1 \\ \vdots \\ i_n \\ 1 \end{bmatrix}.$$

And $C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'$. Take any element of $C\mathbf{i}'$ for example, $c_{k,1} * 2^{i_1} + \cdots + c_{k,n} * 2^{i_n} + c_{k,n+1} * i_{n+1} + \cdots + c_{k,2n} * i_{2n} + c_{k,2n+1}$ for $1 \le k \le (2 * n + 1)$, as the template alignment function by which the template elements are determined for the corresponding iterations (computations) and their required data to be mapped onto, then two dependent iterations, $\mathbf{i}_1 = [2^{i_{1,1}}, \ldots, 2^{i_{1,n}}, i_{1,n+1}, \ldots, i_{1,2n}, 1]$ and $\mathbf{i}_2 = [2^{i_{2,1}}, \ldots, 2^{i_{2,n}}, i_{2,n+1}, \ldots, i_{2,2n}, 1]$, which both access to the same element of array A_e, will be mapped onto the same template array element.

It is obvious that two iterations $\mathbf{i}_1 = [2^{i_{1,1}}, \ldots, 2^{i_{1,n}}, i_{1,n+1}, \ldots, i_{1,2n}, 1]$ and $\mathbf{i}_2 = [2^{i_{2,1}}, \ldots, 2^{i_{2,n}}, i_{2,n+1}, \ldots, i_{2,2n}, 1]$ can be mapped onto the same template element if and only if $c_{k,1} * 2^{i_{1,1}} + \cdots + c_{k,n} * 2^{i_{1,n}} + c_{k,n+1} * i_{1,n+1} + \cdots + c_{k,2n} * i_{1,2n} + c_{k,2n+1} = c_{k,1} * 2^{i_{2,1}} + \cdots + c_{k,n} * 2^{i_{2,n}} + c_{k,n+1} * i_{2,n+1} + \cdots + c_{k,2n} * i_{2,2n} + c_{k,2n+1}$ for $1 \le k \le (2 * n + 1)$. Due to $C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'$, for two iterations $\mathbf{i}_1 = [2^{i_{1,1}}, \ldots, 2^{i_{1,n}}, i_{1,n+1}, \ldots, i_{1,2n}, 1]$ and $\mathbf{i}_2 = [2^{i_{2,1}}, \ldots, 2^{i_{2,n}}, i_{2,n+1}, \ldots, i_{2,2n}, 1]$, if $a_{e,1} * 2^{i_{1,1}} + \cdots + a_{e,n} * 2^{i_{1,n}} + b_{e,1} * i_{1,n+1} + \cdots + b_{e,n} * i_{1,2n} + f_e = a_{e,1} * 2^{i_{2,1}} + \cdots + a_{e,n} * 2^{i_{2,n}} + b_{e,1} * i_{2,n+1} + \cdots + b_{e,n} * i_{2,2n} + f_e$, then $c_{k,1} * 2^{i_{1,1}} + \cdots + c_{k,n} * 2^{i_{1,n}} + c_{k,n+1} * i_{1,n+1} + \cdots + c_{k,2n} * i_{1,2n} + c_{k,2n+1} = c_{k,1} * 2^{i_{2,1}} + \cdots + c_{k,n} * 2^{i_{2,n}} + c_{k,n+1} * i_{2,n+1} + \cdots + c_{k,2n} * i_{2,2n} + c_{k,2n+1}$. For two dependent iterations $\mathbf{i}_1 = [2^{i_{1,1}}, \ldots, 2^{i_{1,n}}, i_{1,n+1}, \ldots, i_{1,2n}, 1]$ and $\mathbf{i}_2 = [2^{i_{2,1}}, \ldots, 2^{i_{2,n}}, i_{2,n+1}, \ldots, i_{2,2n}, 1]$, they access to the same data element of array A_e, that is, $a_{e,1} * 2^{i_{1,1}} + \cdots + a_{e,n} * 2^{i_{1,n}} + b_{e,1} * i_{1,n+1} + \cdots + b_{e,n} * i_{1,2n} + f_e = a_{e,1} * 2^{i_{2,1}} + \cdots + a_{e,n} * 2^{i_{2,n}} + b_{e,1} * i_{2,n+1} + \cdots + b_{e,n} * i_{2,2n} + f_e$. Therefore, the two dependent iterations, $\mathbf{i}_1 = [2^{i_{1,1}}, \ldots, 2^{i_{1,n}}, i_{1,n+1}, \ldots, i_{1,2n}, 1]$ and $\mathbf{i}_2 = [2^{i_{2,1}}, \ldots, 2^{i_{2,n}}, i_{2,n+1}, \ldots, i_{2,2n}, 1]$, will be mapped onto the same template array element. Similarly, for other nonlinear cases, the steps of proof are the same. Thus, it is inferred that two dependent iterations, wherein both write the same element of array A_e, will be mapped onto the same template array element to avoid the distributed data updates in writing the same data element of array A_e. □

From Theorem 3.1, it is apparent that our proposed techniques are not one-to-one mappings but rather many-to-one mappings because dependent iterations with the properties described in Theorem 3.1 will be mapped onto the same template element. Thus, the size of the template array could have a smaller order than the iteration space. On the other hand, the mappings of the read-only arrays will not inflect the correctness of the execution; it is generally helpful to replicate multiple copies of a

read-only data element onto the processors requiring that data element for computation, so that no further inter-processor communication is required within the nested loops.

As mentioned, our proposed alignment techniques do not consider the data dependences. In this approach, there are two principal advantages. First, the difficulty of alignment can be decreased, enabling our techniques to be more broadly applicable than other methods. Second, the original data dependences (if it existed) are likely eliminated or reduced by the resulting alignment function because the dependent iterations and its required data could be mapped onto the same temple element. This fact might benefit the exploitation of parallelism in the distribution phase.

3.4 Time complexity

The proposed alignment techniques reduce the problem of mapping the computations and data in a program to the standard linear algebra problem of determining a null space basis for a matrix. This includes three main phases: The first phase involves determining, according to the alignment constraints, the value for each element in the matrix V. The second phase involves applying Gaussian elimination to determine a basis for U^{T}, the null space of V^{T}. The third phase involves extracting the solution matrices U.

Suppose that V is an $m \times n$ matrix and its rank is r, where r is at most the minimal value of m and n. The worst-case time complexity to determine the values for all of the elements of V is $O(m \times n)$. Determining a null space basis for V^{T} using Gaussian elimination requires $O(r \times m \times n + r^3)$ *arithmetic* operations. The worst-case time complexity to determine a basis for U^{T} is accordingly $O(r \times m \times n + r^3)$. Since $U = H(r+1:m, 1:m)$, the worst-case time complexity to extract the solution matrices U from H is clearly $O(m^2 - r \times m)$. Hence, the worst-case time complexity for the presented techniques is $O(m \times n + r \times m \times n + r^3 + m^2 - r \times m)$, which is similar to the method proposed by Bau et al. [2].

4 Experimental results

We have experimented with the proposed alignment techniques on certain codes extracted from the TFFT2 programs in the SPEC95FP Benchmarks on our PC-cluster environment. Our PC-cluster includes a master, a PC with one P4 (Pentium 4) 1.8 GHz CPU and 256 MB main memory, and 10 slaves, wherein each has one P4 1.5 GHz CPU and 128 MB main memory. The operating system is RedHat Linux 7.1 with the installed parallel software package—MPI-1.2.2.2. We hand-coded these extracted code segments in MPI (Message Passing Interface) with C language and executed them sequentially and in parallel with our MPI environment, respectively.

The code segments extracted from the TFFT2 programs in the SPEC95FP Benchmarks contain referenced arrays using exponential subscripts and other complex nonlinear subscripts, as shown respectively in Tables 1 and 2. The code segment in Table 1 contains three do-loops. The first do-loop in Table 1 is only used for evaluating the performance of every machine tested, so it is not considered in searching for alignment functions for those arrays in statements S_1 and S_2. Those arrays in statements

Table 1 The extracted code segment and the data alignments with our technique in Sect. 3.1

```
DO P = 1, N, 1
  ...
  DO I = 1, M, 1
    DO J = 1, 2^I, 1
    ...
  S1: A(2**I + J) = r1*s1 - r2*s2 + COS(ti)
  S2: B(2**I + J) = r2*s1 + r1*s2 + SIN(ti)
    ...
    ENDDO
  ENDDO
  ...
ENDDO
```

Align $A(2**I + J)$ with $T(2**I + J)$.
Align $B(2**I + J)$ with $T(2**I + J)$.

Table 2 The extracted code segment and the data alignments with our technique in Sect. 3.2

```
DO I = 0, 2**(M - 1)
  ...
  DO L = 1, (1 + M)/2
    DO J = 0, 2**((1 + M - L) - 1)
      DO K = 1, 2**(L - 2)
      ...
S1: A(K + J*2**(L - 1) + 2**(L - 2)) = t1 + t2 - sin(pi)
S2: B(K + J*2**(L - 1) + 2**(L - 2)) = t3 - t4 + cos(pi)
      ...
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

Align $A(K + J*2**(L-1)+2**(L-2))$ with $T(K + J*2**(L-1)+2**(L-2))$.
Align $B(K + J*2**(L-1)+2**(L-2))$ with $T(K + J*2**(L-1)+2**(L-2))$.

S_1 and S_2 for the second do-loop and the third do-loop have no data dependence such that they can intrinsically be executed in parallel. Our proposed method in Sect. 3.1 can align the arrays of this code segment in a communication-free manner which does not cause interprocessor communication.

On the other hand, the code segment in Table 2 contains four do-loops. The first do-loop in Table 2 is also only used to evaluate the performance of every machine tested. Hence, it is not considered in searching for alignment functions for those arrays in statements S_1 and S_2. Those arrays in statements S_1 and S_2 for the second do-loop, the third do-loop, and the fourth do-loop have no data dependence such that they can intrinsically be executed in parallel. Our proposed method in Sect. 3.2 can align the arrays of this code segment in a communication-free manner which does not cause interprocessor communication.

The corresponding sequential and parallel run times for the code segments in Tables 1 and 2 are shown correspondingly in Figs. 4, 5, and 6. Figures 4 to 6 reveal that the difference between the sequential and parallel run time is significant. This is because these tested code segments are computation-intensive such that the cost for finally receiving the computed results appears less significant. Generally, a computation-intensive nested loop which can be aligned in a communication-free manner should well benefit from parallelism.

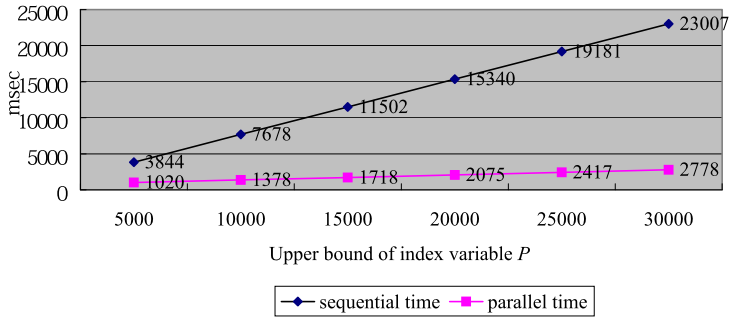

Fig. 4 The overall sequential and parallel run times for the extracted code segment in Table 1

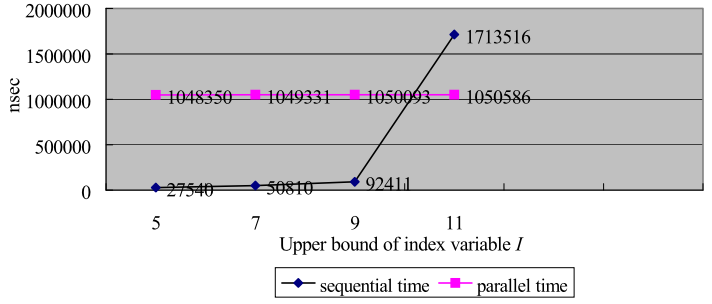

Fig. 5 The overall sequential and parallel run times for the extracted code segment in Table 2

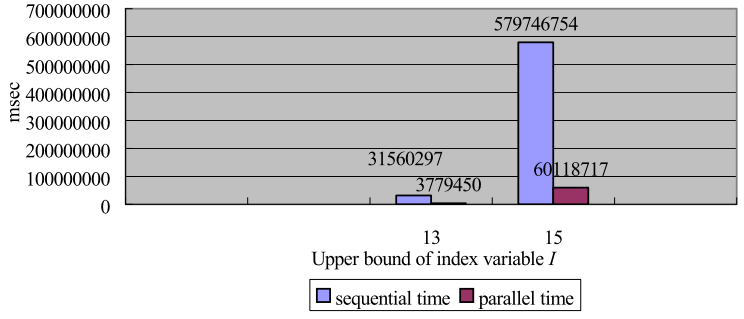

Fig. 6 The overall sequential and parallel run times for the extracted code segment in Table 2

5 Conclusions

Generally, a single nested loop is the main program section to be parallelized. The data alignment, which reduces the interprocessor communication to avoid undermining the benefits of parallelism, is often discussed on a single nested loop basis. On the other hand, to reduce the communication for a program with multiple parallelizable nested loops, a data alignment focusing on the whole program may be essential. How-

ever, the alignment entirely based on the whole program is extremely complicated because various factors, e.g., data dependences across the nested loops, the iteration spaces for different nested loops, and the data access patterns for each nested loop, etc., have to be considered. Alternatively, an intuitive way for aligning data arrays over multiple nested loops is to align the data arrays with the proposed techniques for each individual single nested loop. It is possible that large amounts of communication will occur in this way because the data arrays might have to be remapped onto the template elements and redistributed over the processors across the nested loops. Nevertheless, whether the time to run the program in the above approach is beneficial or not might depend on the characteristics of the program. For example, if the program computation is enormous, the communication cost might become relatively less significant for the program execution. The executed program can perhaps obtain an effective speedup.

For the referenced arrays, linear expressions appeared with the most frequency and most data alignment methods were used mainly to align the array references with linear subscripts. Based on the study of Reilly [33], Paek [29] and Hoeflinger [18] and our survey, the number of arrays with exponential subscripts or other complex nonlinear subscripts might be attained to a certain extent. However, the data alignments for the arrays with exponential subscripts or other complex nonlinear subscripts were previously rarely discussed. In this paper, we propose two alignment techniques to properly map, in a communication-free manner, the computations and array references with exponential subscripts or other complex nonlinear subscripts onto the virtual processors. Our alignment techniques, which are based on elementary linear algebra, reduce the alignment problem to the problem of determining a null space basis for a matrix. By simplifying the process of solving the null space basis, the proposed techniques can easily determine the desired mapping functions. Apparently, many different mapping functions can be obtained by different linear combinations of the null space basis. Furthermore, because dependent iterations with the properties described in Theorem 3.1 will be mapped onto the same template element, the proposed techniques are not one-to-one mappings.

Acknowledgements This work is supported by the National High-Tech Research and Development Plan of China (863 Plan) under Grant Nos. 2008AA01Z106 and 2006AA01Z202, the National Natural Science Foundation of China under Grant Nos. 60811130528, 60725208, and 60533040, Shanghai Pujiang Plan No. 07pj14049.

References

1. Alex A, Codina MJ, Alez GA, Kaeli D (2004) Removing communications in clustered microarchitectures through instruction replication. ACM Trans Archit Code Optim 1(2):127–151
2. Bau D, Kodukula I, Kotlyar V, Pingali K, Stodghill P (1994) Solving alignment using elementary linear algebra. In: Conference record of the 7th workshop on languages and compilers for parallel computing, pp 46–60
3. Boudet V, Rastello F, Yves R (1998) Alignment and distribution is NOT (always) NP-hard. In: Proceeding of 1998 international conference on parallel and distributed systems, vol 5(9), 1998, pp 648–657
4. Chang W-L, Chu C-P, Wu J-H (2001) Communication-free alignment for array references with linear subscripts in three loop index variables or quadratic subscripts. J Supercomput 20(1):67–83

5. Chang W-L, Huang J-W, Chu C-P (2004) Using elementary linear algebra to solve data alignment for arrays with linear or quadratic references. IEEE Trans Parallel Distrib Syst 15(1):28–39
6. Chu C-P, Chang W-L, Chen I, Chen P-S (1998) Communication-free alignment for array references with linear subscripts in two loop index variables or quadratic subscripts. In: Proceedings of the second IASTED international conference on parallel and distributed computing and networks (PDCN'98), Australia, 1998, pp 571–576
7. Dion M, Yves R (1996) Mapping Affine loop nests: new results. Parallel Comput 22(10):1373–1397
8. Edmonds J (1967) Systems of distinct representative and linear algebra. J Res Nat B Stand Sect B 71(4):241–245
9. Franke B, O'Boyle MFP (2005) A complete compiler approach to auto-parallelizing C programs for multi-DSP systems. IEEE Trans Parallel Distrib Syst 16(3):234–245
10. Feautrier P (1993) Toward automatic partitioning of arrays on distributed memory computers. In: ACM international conference on supercomputing, 1993, pp 175–184
11. Gschwind M, Hofstee HP, Flachs B, Hopkins M, Watanabe Y, Yamazaki T (2006) Synergistic processing in cell's multicore architecture. IEEE Micro 26(2):10–24
12. Gebis J, Patterson D (2007) Embracing and extending 20th-century instruction set architectures. Computer 40(4):68–75
13. Guo M, Yamashita Y, Nakata I (1998) Efficient implementation of multi-dimensional array redistribution. IEICE Trans Inf Syst E81-D(11):1195–1204
14. Guo M, Nakata I, Yamashita Y (2000) Contention-free communication scheduling for array redistribution. Parallel Comput 26(8):1325–1343
15. Guo M, Nakata I (2001) A framework for efficient array redistribution on distributed memory multicomputers. J Supercomput 20(3):243–265
16. Guo M (2003) Efficient loop partitioning for parallel codes of irregular scientific computations. IEICE Trans Inf Syst E86-D(9):1825–1834
17. Guo M (2003) Communication generation for irregular codes. J Supercomput 25(3):199–214
18. Hoeflinger J (1998) Interprocedural parallelization using memory classification analysis. PhD thesis, Univ of Illinois at Urbana-Champaign, Center for Supercomputing Res & Dev
19. Hsu C-H, Bai S-W, Chung Y-C, Yang C-S (2000) A generalized basic-cycle calculation method for array redistribution. IEEE Trans Parallel Distrib Syst 11(12):1201–1216
20. Hsu C-H, Lan C-Y, Chen S-C (2006) Optimizing scheduling stability for runtime data alignment. In: EUC 2006 proceedings. Lecture notes in computer science, vol 4097. Springer, Berlin
21. Hwang G-H, Lee JK (1999) An expression-rewriting framework to generate communication sets for HPF programs with block-cyclic distribution. Parallel Comput 25:1105–1139
22. Kandemir M, Choudhary A, Shenoy N, Banerjee P, Ramanujam J (1998) A hyperplane based approach for optimizing spatial locality in loop nests. In: Proc 12th ACM int conf supercomputing, 1998, pp 69–76
23. Kandemir M, Ramanujam J, Choudhary A, Banerjee P (1998) A loop transformation algorithm based on explicit data layout representation for optimizing locality. In: Proc 11th international workshop, LCPC'98, Chapel Hill, NC, USA, 1998, pp 34–50
24. Lam AW, Lam MS (1998) Maximizing parallelism and minimizing synchronization with affine partitions. Parallel Comput 24(3–4):445–475
25. Lam AW, Cheong GI, Lam MS (1999) An affine partitioning algorithm to maximize parallelism and minimize communication. In: 13th ACM international conference on supercomputing, Rhodes, Greece, 1999, pp 228–237
26. Lee PZ (1997) Efficient algorithms for data distribution on distributed memory parallel computers. IEEE Trans Parallel Distrib Syst 8(8):825–839
27. Luenberger DG (1984) Linear and nonlinear programming. Addison-Wesley, Reading
28. Ozcan E, Onbasioglu E (2007) Memetic algorithms for parallel code optimization. Int J Parallel Program 35(1)
29. Paek Y (1997) Compiling for distributed memory multiprocessors based on access region analysis. PhD thesis, Univ of Illinois at Urbana-Champaign, Center for Supercomputing Res & Dev
30. Pan L, Xue J, Lai MK (2007) Toward automatic data distribution for migrating computations. In: The proceedings of 2007 international conference on parallel processing, September 2007
31. Petersen MP, Padua AD (1996) Static and dynamic evaluation of data dependence analysis techniques. IEEE Trans Parallel Distrib Syst 7(11):1121–1132
32. Ramanujam J, Sadayappan P (1991) Compile-time techniques for data distributed in distributed memory machines. IEEE Trans Parallel Distrib Syst 2(4):472–482

33. Reilly J (1995) SPEC95 products and benchmarks. SPEC Newsletter
34. Shikano H, Ito M et al (2008) Heterogeneous multi-core architecture that enables 54x AAC-LC stereo encoding. IEEE J Solid-State Circuits 43(4):902–910
35. Shih K-P, Sheu J-P, Huang C-H (2000) Statement-level communication-free partitioning techniques for parallelizing compilers. J Supercomput 15(3):243–269
36. Wu J-H, Chu C-P (2007) An exact data dependence testing method for quadratic expressions. Inf Sci 177(23)
37. Wolfe M (1996) High performance compilers for parallel computing. Addison-Wesley, Reading
38. Zhao Y, Kennedy K (2007) Dependence-based code generation for a CELL processor. In: Lecture notes in computer science, vol 4382. Springer, Berlin

Minyi Guo received his Ph.D. degree in computer science from University of Tsukuba, Japan. Before 2000, Dr. Guo had been a research scientist of NEC Corp., Japan. He had been Professor at the School of Computer Science and Engineering, The University of Aizu, Japan. Currently Dr. Guo is distinguished Chair Professor at Department of Computer Science and Engineering, Shanghai Jiao Tong University. He is also Adjunct Professor of Dalian Maritime University, China. Doctor Guo has published more than 160 research papers in international journals and conferences. Doctor Guo has served as general chair, program committee or organizing committee chair for many international conferences. He is the founder of International Conference on Parallel and Distributed Processing and Applications (ISPA), and International Conference on Embedded and Ubiquitous Computing (EUC). He is the Editor-in-Chief of the Journal of Embedded Systems. He is also in editorial board of Journal of Pervasive Computing and Communications, International Journal of High Performance Computing and Networking, Journal of Embedded Computing, Journal of Parallel and Distributed Scientific and Engineering Computing, and International Journal of Computer and Applications. Professor Minyi Guo received the National Science Fund for Distinguished Young Scholars in 2007, and is also the PI of NSFC Key Project "Theoretical and Technical Key Points of Pervasive Computing." Doctor Guo's research interests include parallel and distributed processing, parallelizing compilers, pervasive computing, embedded software optimization, molecular computing and software engineering. He is a senior member of IEEE, and member of the ACM, IPSJ, CCF, and IEICE.

Weng-Long Chang received the Ph.D. degree in Computer Science and Information Engineering from National Cheng Kung University, Taiwan, Republic of China, in 1999. He is currently Full Professor at the Department of Computer Science and Information Engineering in National Kaohsiung University of Applied Sciences. His researching interests include quantum algorithms, adiabatic quantum algorithms, DNA-based algorithms, and languages and compilers for parallel computing.

Bo Jiang received his Master Degree in Computer Science from National University of Defense Technology of P.R. China in 1988. He is currently Professor at School of Information Science and Technology, Dalian Maritime University, China. His current research interests include algorithm analysis and complexity, computational geometry, software theory and engineering.

Shu-Chien Huang received the Ph.D. degree in Computer Science and Information Engineering from National Cheng Kung University, Taiwan, Republic of China, in 1999. He is currently Assistant Professor at the Department of Computer Science in National Pingtung University of Education. His researching interests include image processing, quantum algorithms, and languages and compilers for parallel computing.

Sien-Tang Tsai received the Master Degree in Computer Science from University of Georgia, Athens, U.S.A., in 1983. He is currently Associate Professor at the Department of Information Management in Southern Taiwan University of Technology. His researching interests include parallel computing, quantum computing and DNA computing.

Michael (Shan-Hui) Ho received His Ph.D. degree in IS/CS with Management/Accounting minor from University of Texas at Austin in 1988. He had been CFO Assistant, Oracle/DB2 DBA and Project Manager/Senior Software Engineer for Oracle/DB2/SQL with many major US corporations. He is currently Full Professor at the Department of Information Management both in Ming Chuan University and national Taipei University. His researching interests include computation theories, parallel/mobile/quantum/DNA computing, multimedia and software engineering.