Vol. 42 No. 4 IPSJ Journal Apr. 2001

Regular Paper

Modulo Interval Arithmetic and Its Application to Program Analysis

Tsuneo Nakanishi[†] and Akira Fukuda[†]

Interval arithmetic, an arithmetic system on intervals of real numbers, is useful for program analysis which deals with range information of variables or expressions such as array reference analysis, data dependence analysis and value range analysis. However, since loop indices or array subscripts often take contiguous integers with a stride, the interval representing densely contiguous real numbers is not accurate representation for program analysis and degrades opportunity of parallelization or code optimization. In this paper *modulo interval arithmetic*, an arithmetic system on sets of contiguous integers with strides included in real intervals, is presented. Modulo interval arithmetic has both arithmetic operations and set operations which are useful for various program analysis. Moreover, this paper discusses application of modulo interval arithmetic to program analysis for parallelizing compilers.

1. Introduction

The interval, which is defined for two arbitrary real numbers $a, b \in \boldsymbol{R}$ $(a \leq b)$, is the set of real numbers $\{x \in \boldsymbol{R} \mid a \leq x \leq b\}$ and denoted by $[a, b]$. Interval arithmetic [12),18)] is an arithmetic defined on intervals. Interval arithmetic is used to estimate errors produced by floating point calculation. Interval arithmetic or its similar ideas are useful also for program analysis, for example, array reference analysis, data dependence analysis [2)] and value range analysis [6),14),17),20)]. However, real numbers are usually used for representing physical data in real programs and loop indices and array subscripts, which are analyzed in these program analyses, mostly take integers. Thus the interval is too rough representation for program analysis, especially to deal with sparse integer intervals such as index values of loops with strides.

Subarray representation is important to analyze data dependences among procedures or to optimize data partitioning or transference on distributed memory multiprocessors. Researchers in high performance computing have developed various kinds of subarray representations [1),3),11),15),19)]. Some of these subarray representations are based on the triplet notation, which is used in Fortran90 or HPF [9)] programs, or other subarray representations use sets of inequalities bounding subarrays. These subarray representations have basic set operations for data dependence analysis in general. Considering facility of implementation and manageability on run-time, the triplet notation will be the most simple and practical subarray representation. Note that the triplet notation is a kind of intervals consisting of contiguous integers possibly with a stride.

Based on these observations, this paper proposes modulo interval arithmetic, an extension of interval arithmetic, which has arithmetic operations and set operations on sets of integers included in intervals of real numbers. Modulo interval, denoted by $[a, b]_{n(r)}$, means the set of integers in $[a, b]$ which are divisible by n with a remainder of r. Although additional information of the modulo interval to the interval are only the modulus n and the residue r, the modulo interval is a simple and useful representation with rich well-defined mathematical properties to express a set of iterations and subarrays.

The advantages of the modulo interval over other subarray representations will be its simplicity and generality. Arithmetic and set operations of the modulo interval are so simple that complicated implementation is not required like inequality-based subarray representations. Simplicity of these operations owes the modulus part of the modulo interval which is not contained in the triplet notation. Most inequality-based subarray representations are exclusively used for subarray representation and thus have only set operations for array reference analysis and data dependence analysis. On the other hand, modulo interval arithmetic has not only set operations but also arithmetic operations. The arithmetic operations expand applications of modulo interval arithmetic to other program analysis, such as value range analysis, besides

† Graduate School of Information Science, Nara Institute of Science and Technology

array reference analysis and data dependence analysis.

This paper is organized as follows: Section 2 summarizes interval arithmetic as a background of modulo interval arithmetic. Section 3 presents the definition of the modulo interval and mathematical properties of modulo interval arithmetic. In Section 4, modulo interval arithmetic is applied to program analysis performed by parallelizing compilers. Section 5 describes related works. Section 6 concludes the paper.

2. Interval Arithmetic

An interval is defined for two arbitrary real numbers a and b with $a \leq b$ and denoted by $[a, b]$. An interval $[a, b]$ means the set of real numbers $\{x \in \boldsymbol{R} \mid a \leq x \leq b\}$.

Any arithmetic operation $\odot$ on real numbers is redefined as an arithmetic operation on intervals by the following definition where A and B are intervals.

$$A \odot B = \{x = a \odot b \mid a \in A, b \in B\}$$

By the above definition and denseness of real numbers, addition $(+)$, subtraction $(-)$, multiplication $(\times)$ and division $(/)$ of intervals $[a, b]$ and $[c, d]$ are derived as follows except division is not derived if $0 \in [c, d]$.

$$\begin{aligned} [a,b] + [c,d] &= [a+c, b+d] \\ [a,b] - [c,d] &= [a-d, b-c] \\ [a,b] \times [c,d] &= [\min\{ac, ad, bc, bd\}, \\ &\qquad \max\{ac, ad, bc, bd\}] \\ [a,b] \,/\, [c,d] &= [\min\{a/c, a/d, b/c, b/d\}, \\ &\qquad \max\{a/c, a/d, b/c, b/d\}] \end{aligned}$$

Moreover, exponentiation of an interval $[a, b]$ is derived as follows where $n \geq 0$.

$$[a,b]^n = \begin{cases} [a^n, b^n] \\ \quad \text{(if } a \geq 0 \text{ or } n \text{ is odd)} \\ [b^n, a^n] \\ \quad \text{(if } b < 0 \text{ and } n \text{ is even)} \\ [0, \max\{a^n, b^n\}] \\ \quad \text{(otherwise)} \end{cases}$$

Addition and multiplication on intervals are both associative and commutative. The distributive law does not hold for interval arithmetic in general. Instead of the distributive law, the subdistributive law shown below holds for intervals A, B and C.

$$A \times (B + C) \subseteq A \times B + A \times C$$

See Ref. 12) for the details of interval arithmetic.

3. Modulo Interval Arithmetic

In this section the modulo interval and its mathematical properties are introduced.

3.1 Definition

A modulo interval is defined for two arbitrary real numbers a and b with $a \leq b$, a non-zero integer n referred to as a *modulus*, and an integer r referred to as a *residue*, and denoted by $[a, b]_{n(r)}$. A modulo interval $[a, b]_{n(r)}$ means the set of integers $\{x \in \boldsymbol{Z} \mid a \leq x \leq b, x = nm + r, m \in \boldsymbol{Z}\}$. Examples are shown below.

$$\begin{aligned} [7, 21]_{5(4)} &= \{9, 14, 19\} \\ [-6, 10]_{-4(2)} &= \{-6, -2, 2, 6, 10\} \\ [-7, 9]_{4(-3)} &= \{-7, -3, 1, 5, 9\} \\ [3, 7]_{2(3)} &= \{3, 5, 7\} \end{aligned}$$

A modulo interval $[a, b]_{n(r)}$ is *normalized* if both a and b are in $[a, b]_{n(r)}$ and $0 \leq r < n$.

Some definitions and notations are described below for formal discussion.

The *sign* of an integer a, denoted by $\mathrm{sig}(a)$, gives $+1$, 0 and -1 if $a > 0$, $a = 0$ and $a < 0$, respectively. The *quotient* of an integer a divided by an integer b, denoted by $a \operatorname{div} b$, is defined as $\mathrm{sig}(a/b) * \lfloor |a|/|b| \rfloor$. The *remainder* of an integer a divided by an integer b, denoted by $a\%b$, is defined as $a - (a \operatorname{div} b) * b$. $0 \leq a\%b < b$ if and only if $a \geq 0$ and $0 \geq a\%b > b$ if and only if $a < 0$.

The *greatest common divisor* of integers m and n, denoted by $\gcd(m, n)$, is defined as the greatest positive integer which divides both m and n if m and n are non-zero integers. $\gcd(m, n)$ is defined as zero if m or n is zero.

The *least common multiple* of integers m and n, denoted by $\mathrm{lcm}(m, n)$, is defined as the least positive integer which are divisible both by m and by n if m and n are non-zero integers. $\mathrm{lcm}(m, n)$ is defined as zero if m or n is zero.

3.2 Arithmetic Operations

Any arithmetic operation $\odot$ on integers is redefined as an arithmetic operation on modulo intervals by the following definition where A and B are modulo intervals and z is an integer.

$$\begin{aligned} A \odot B &= \{x = a \odot b \mid a \in A, b \in B\} \\ A \odot z &= \{x = a \odot z \mid a \in A\} \\ z \odot B &= \{x = z \odot b \mid b \in B\} \end{aligned}$$

Addition, subtraction and multiplication on modulo intervals have the following property.

Property 1 The relations below hold for addition, subtraction and multiplication on modulo intervals.

$$[a,b]_{m(r)} + [c,d]_{n(s)} \subseteq [a+c, b+d]_{\gcd(m,n)(r+s)}$$
$$[a,b]_{m(r)} - [c,d]_{n(s)} \subseteq [a-d, b-c]_{\gcd(m,n)(r-s)}$$
$$[a,b]_{m(r)} \times [c,d]_{n(s)} \subseteq [\min\{ac,ad,bc,bd\}\ , \max\{ac,ad,bc,bd\}]_{\gcd(m,n)(rs)}$$

PROOF. Let x denote an integer in $[a,b]_{m(r)}$ and y an integer in $[c,d]_{n(s)}$. x and y are expressed as $x = mu + r$ and $y = nv + s$ with appropriate integers u and v, respectively. Thus the sum of x and y is represented with $g = \gcd(m,n)$ as follows.

$$x + y = (mu + r) + (nv + s) = (mu/g + nv/g)g + (r + s)$$

Since both m and n are divisible by the greatest common divisor of m and n (namely g), $mu/g+nv/g$ is an integer. It follows that $x+y$ divided by g makes a remainder of $r+s$. On the other hand, since $a \le x \le b$ and $c \le y \le d$, $a+c \le x+y \le b+d$ holds. These prove the first relation concerning addition on modulo intervals.

The other relations are proved likewise. □

Property 1 is given as a set of inclusion relations. However, the relations concerning addition and subtraction on modulo intervals are reduced to equivalent relations in the special case described in the following property.

Property 2 The relations below hold for addition and subtraction on normalized modulo intervals of an identical modulus.

$$[a,b]_{n(r)} + [c,d]_{n(s)} = [a+c, b+d]_{n(r+s)}$$
$$[a,b]_{n(r)} - [c,d]_{n(s)} = [a-d, b-c]_{n(r-s)}$$

PROOF. Since both $[a,b]_{n(r)}$ and $[c,d]_{n(s)}$ are normalized, a, b, c and d are expressed as $a = nu_a + r$, $b = nu_b + r$, $c = nu_c + s$ and $d = nu_d + s$ with appropriate integers u_a, u_b, u_c and u_d, respectively. Moreover, $a \le b$, $c \le d$ and $n > 0$ hold and it follows that $u_a \le u_b$ and $u_c \le u_d$.

An integer x in $[a+c, b+d]_{n(r+s)}$ is expressed as $x = nu + r + s$ with an appropriate integer u such that $u_a+u_c \le u \le u_b+u_d$. Thus u is expressed as $u = u_a+u_c+\Delta$ with an integer Δ such that $0 \le \Delta \le (u_b + u_d) - (u_a + u_c)$.

Consider the case such that $0 \le \Delta < u_b - u_a$. Since x is expressed as $x = \{n(u_a + \Delta) + r\} + (nu_c+s)$, it follows that $n(u_a+\Delta)+r \in [a,b]_{n(r)}$ and $nu_c + s \in [c,d]_{n(s)}$.

Consider the case such that $u_b - u_a \le \Delta \le (u_b + u_d) - (u_a + u_c)$. Since x is expressed as $x = (nu_b + r) + \{n(u_a + u_c - u_b + \Delta) + s\}$, it follows that $nu_b + r \in [a,b]_{n(r)}$ and $n(u_a + u_c - u_b + \Delta) + s \in [c,d]_{n(s)}$.

$[a+c, b+d]_{n(r+s)} \subseteq [a,b]_{n(r)} + [c,d]_{n(s)}$ is proved by the consequences of the above cases. $[a,b]_{n(r)} + [c,d]_{n(s)} \subseteq [a+c, b+d]_{n(r+s)}$ holds by Property 1. These prove the first relation concerning addition of normalized modulo intervals of an identical modulus.

The relation on subtraction is proved likewise. □

Addition, subtraction and multiplication of a modulo interval and an integer have the following property.

Property 3 The relations below hold for addition, subtraction and multiplication of a modulo interval and an integer.

$$[a,b]_{n(r)} + z = [a+z, b+z]_{n(r+z)}$$
$$z + [a,b]_{n(r)} = [z+a, z+b]_{n(z+r)}$$
$$[a,b]_{n(r)} - z = [a-z, b-z]_{n(r-z)}$$
$$z - [a,b]_{n(r)} = [z-b, z-a]_{n(z-r)}$$
$$[a,b]_{n(r)} \times z = [az, bz]_{nz(rz)}$$
$$z \times [a,b]_{n(r)} = [az, bz]_{nz(rz)}$$

PROOF. By the definition of the modulo interval, $[a,b]_{n(r)} + z$ is expressed as the set of integers $\{x = m + z \mid m \in [a,b]_{n(r)}\}$. This set is reduced to a modulo interval as follows.

$$\begin{aligned}&\{x = m + z \mid m \in [a,b]_{n(r)}\}\\ &= \{x = m + z \mid m = nu + r, u \in \boldsymbol{Z},\\ &\qquad a \le m \le b\}\\ &= \{x \mid x = nu + r + z, u \in \boldsymbol{Z},\\ &\qquad a + z \le x \le b + z\}\\ &= [a+z, b+z]_{n(r+z)}\end{aligned}$$

This proves the first relation concerning addition of a modulo interval and an integer.

The other relations are proved likewise. □

Note that the relations described in Property 3 are given as equivalent relations.

Exponentiation on modulo intervals has Property 4. Although exponentiation does not appear frequently in array subscripts of real programs, induction variable elimination for parallelization transforms subscripts of triangular packed matrices, appeared in BLAS [10], into quadratic expressions [7] for example .

Property 4 The relation below holds for exponentiation on modulo intervals where z is a non-negative integer.

$$[a,b]^z_{n(r)} \subseteq \begin{cases} [a^z, b^z]_{n(r^z)} \\ \quad \text{(if } a \geq 0 \text{ or } z \text{ is odd)} \\ [b^z, a^z]_{n(r^z)} \\ \quad \text{(if } b < 0 \text{ and } z \text{ is even)} \\ [0, \max\{a^z, b^z\}]_{n(r^z)} \\ \quad \text{(otherwise)} \end{cases}$$

PROOF. Trivial. This property is proved by the property concerning exponentiation on intervals described in Section 2 and Property 1. □

If the modulus of a modulo interval is a prime number, the following property holds.

Property 5 The relation below holds for exponentiation on modulo intervals where n is a non-negative prime number.

$$[a,b]^n_{n(r)} \subseteq \begin{cases} [a^n, b^n]_{n(r)} \\ \quad \text{(if } a \geq 0 \text{ or } n \text{ is odd)} \\ [b^n, a^n]_{n(r)} \\ \quad \text{(if } b < 0 \text{ and } n \text{ is even)} \\ [0, \max\{a^n, b^n\}]_{n(r)} \\ \quad \text{(otherwise)} \end{cases}$$

PROOF. Trivial. This property is proved by Fermat's Theorem[4)] and Property 4. Fermat's Theorem, which is well-known as a fundamental theorem of public key cryptography, is the theorem states that $a^p \equiv a \pmod p$ holds for any integer a where p is a prime number. □

3.3 Set Operations

Since a modulo interval is a set of integers, set operations are defined on modulo intervals.

Intersection of modulo intervals has the following property.

Property 6 The intersection of modulo intervals $[a,b]_{m(r)}$ and $[c,d]_{n(s)}$ is given as below where z is an integer which is divisible by m with a remainder of r and by n with a remainder of s.

$$[a,b]_{m(r)} \cap [c,d]_{n(s)} = \begin{cases} [\max\{a,c\}, \min\{b,d\}]_{\mathrm{lcm}(m,n)(z)} \\ \quad \text{(if } s-r \text{ is divisible by} \\ \quad \ \gcd(m,n).) \\ \emptyset \\ \quad \text{(if } s-r \text{ is not divisible by} \\ \quad \ \gcd(m,n).) \end{cases}$$

PROOF. $[a,b]_{m(r)} \cap [c,d]_{n(s)}$ is expressed as follows.

$$\begin{aligned} &[a,b]_{m(r)} \cap [c,d]_{n(s)} \\ &= \{x \mid x = mu + r = nv + s, u, v \in \boldsymbol{Z}, \\ &\qquad \max\{a,c\} \leq x \leq \min\{b,d\}\} \end{aligned}$$

The above expression includes a Diophantine equation $mu+r = nv+s$ with variables u and v. The Diophantine equation has integer solutions if and only if $s-r$ is divisible by $\gcd(m,n)$. Thus $[a,b]_{m(r)} \cap [c,d]_{n(s)} = \emptyset$ if $s-r$ is not divisible by $\gcd(m,n)$.

If $s-r$ is divisible by $\gcd(m,n)$, the Diophantine equation has the following general integer solutions where u_0 and v_0 are the particular integer solutions of a Diophantine equation $mu - nv = \gcd(m,n)$ and t is an arbitrary integer.

$$\begin{aligned} u &= ((s-r)u_0 + nt)/\gcd(m,n) \\ v &= ((s-r)v_0 + mt)/\gcd(m,n) \end{aligned}$$

$mu+r$ and $nv+s$ are expressed as follows where $k = (s-r)/\gcd(m,n)$.

$$\begin{aligned} &mu + r \\ &= (mu_0(s-r) + mnt)/\gcd(m,n) + r \\ &= \mathrm{lcm}(m,n)t + mku_0 + r \\ &nv + s \\ &= (nv_0(s-r) + mnt)/\gcd(m,n) + s \\ &= \mathrm{lcm}(m,n)t + nkv_0 + s \end{aligned}$$

Since $(mku_0+r)-(nkv_0+s) = k(mu_0-nv_0)-(s-r) = (s-r)/\gcd(m,n)*\gcd(m,n)-(s-r) = 0$, mku_0+r and nkv_0+s are equal to an integer and thus $mu+r = nv+s$. Let z denote the integer. The above $mu+r$ and $nv+s$ are included in $[\max\{a,c\}, \min\{b,d\}]_{\mathrm{lcm}(m,n)(z)}$ if $\max\{a,c\} \leq mu+r = nv+s \leq \min\{b,d\}$. z is divisible by m with a remainder of r and by n with a remainder of s. Thus $[a,b]_{m(r)} \cap [c,d]_{n(s)} = [\max\{a,c\}, \min\{b,d\}]_{\mathrm{lcm}(m,n)(z)}$ if $s-r$ is divisible by $\gcd(m,n)$. □

It is required to find an integer z which is divisible by m with a remainder of r and by n with a remainder of s to obtain $[a,b]_{m(r)} \cap [c,d]_{n(s)}$ by Property 6. z is found by the following algorithm.

Algorithm 1 The algorithm below finds an integer which is divisible by m with a remainder of r and by n with a remainder of s. This is a simplified version of the algorithm which appears in Ref. 2) as *Extended Euclid's Algorithm*.

```
(1) $(x_1, c_1) := (1, |m|)$, $(x_2, c_2) := (0, |n|)$.
(2) If $c_2 = 0$, $z := m(s-r)(\mathrm{sig}(m)x_1)/c_1 + r$
    and terminate the algorithm.
(3) $q := c_1 \,\mathrm{div}\, c_2$.
(4) $(t_x, t_c) := (x_1, c_1) - q(x_2, c_2)$, $(x_1, c_1) :=$
    $(x_2, c_2)$, $(x_2, c_2) := (t_x, t_c)$.
(5) Go to (2).
```

PROOF. See Ref. 2). □

Union of modulo intervals has the following property.

Property 7 The relation below holds for modulo intervals and a positive integer z.

$$[a,b]_{n(r)} = \bigcup_{k=0}^{z-1} [a,b]_{nz(kn+r)}$$

PROOF. Let x denote an integer in $[a,b]_{n(r)}$. x is expressed as $x = nu + r$ with an appropriate integer u and reduced as follows.

$$\begin{aligned} x &= nu + r \\ &= n\{z(u \operatorname{div} z) + u\%z\} + r \end{aligned}$$

$u \operatorname{div} z$ and $u\%z$ are integers. If $0 \leq u\%z < z$, x is expressed as follows and $x \in \bigcup_{k=0}^{z-1} [a,b]_{nz(kn+r)}$ holds.

$$\begin{aligned} x &= n\{z(u \operatorname{div} z) + u\%z\} + r \\ &= nz(u \operatorname{div} z) + n(u\%z) + r \end{aligned}$$

If $-z < u\%z < 0$, x is expressed as follows and $x \in \bigcup_{k=0}^{z-1} [a,b]_{nz(kn+r)}$ holds since $0 < z + u\%z < z$.

$$\begin{aligned} x &= nz(u \operatorname{div} z) + n(u\%z) + r \\ &= nz(u \operatorname{div} z - 1) + n(z + u\%z) + r \end{aligned}$$

Let y denote an integer in $\bigcup_{k=0}^{z-1} [a,b]_{nz(kn+r)}$. y is expressed as $y = nzv + kn + r$ with appropriate integers v and k such that $0 \leq k < z - 1$. y is reduced as follows.

$$\begin{aligned} y &= nzv + kn + r \\ &= n(zv + k) + r \end{aligned}$$

$zv + k$ is an integer. Thus $y \in [a,b]_{n(r)}$ holds. Consequently, $[a,b]_{n(r)} = \bigcup_{k=0}^{z-1} [a,b]_{nz(kn+r)}$ is proved. □

Property 7 is useful to reduce errors produced by modulo interval calculations. For example, the result of $[3,15]_{4(3)} + [5,8]_{3(2)}$ is given as a subset of $[8,23]_{1(0)}$ by Property 1. However, by equalizing moduli of operand modulo intervals to their least common multiple with Property 7, the result is given as a smaller, namely more accurate, subset as follows.

$$\begin{aligned} &[3,15]_{4(3)} + [5,8]_{3(2)} \\ &= ([3,15]_{12(3)} \cup [7,7]_{12(7)} \cup [11,11]_{12(11)}) \\ &\quad + [5,8]_{3(2)} \\ &\subseteq [8,23]_{3(2)} \cup [12,15]_{3(0)} \cup [16,19]_{3(1)} \end{aligned}$$

3.4 Division

Division of a modulo interval by an integer has the following property.

Property 8 The relations below hold for division of a modulo interval by an integer where z is a non-zero integer which divides the modulus of the modulo interval.

$$\begin{aligned} &[a,b]_{n(r)} \operatorname{div} z \\ &= [\min\{a \operatorname{div} z, b \operatorname{div} z\}, \\ &\qquad \max\{a \operatorname{div} z, b \operatorname{div} z\}]_{n \operatorname{div} z (r \operatorname{div} z)} \end{aligned}$$

$$[a,b]_{n(r)} \%z = r\%z$$

PROOF. Let x denote an integer in $[a,b]_{n(r)}$. x is expressed as $x = nu + r$ with an appropriate integer u. The quotient of x divided by z is represented as follows.

$$\begin{aligned} x/z &= (nu + r)/z \\ &= (n/z)u + r/z \end{aligned}$$

Since n is divisible by z, n/z is an integer and equal to $n \operatorname{div} z$. $x \operatorname{div} z$ is represented as follows.

$$x \operatorname{div} z = (n \operatorname{div} z)u + r \operatorname{div} z$$

This proves the first relation concerning the quotient of a modulo interval divided by an integer.

On the other hand, $x\%z$ is represented as follows.

$$\begin{aligned} x\%z &= (nu + r)\%z \\ &= (nu\%z + r\%z)\%z \end{aligned}$$

Since n is divisible by z, $nu\%z$ is zero and $x\%z$ is equal to $r\%z$. This proves the second relation concerning the remainder of a modulo interval divided by an integer. □

In case n is not divisible by z, Property 7 enables division by multiplying the modulus of a modulo interval by z as follows.

$$\begin{aligned} &[2,70]_{7(2)} \operatorname{div} 3 \\ &= ([2,23]_{21(2)} \cup [9,30]_{21(9)} \cup [16,16]_{21(16)}) \\ &\quad \operatorname{div} 3 \\ &= [0,7]_{7(0)} \cup [3,10]_{7(3)} \cup [5,5]_{7(5)} \end{aligned}$$

4. Application to Program Analysis

In this section modulo interval arithmetic is applied to program analysis performed by parallelizing compilers. The examples shown in this section deal with only single loops and one-dimensional arrays for simplicity without loss of generality. The discussion in this section can be applied to multiple nested loops and multi-dimensional arrays. Although some devices are possibly desired to improve accuracy of analysis, such devices will be discussed in our future papers.

4.1 Array Reference Analysis

Since remote memory access or inter-processor communication is a dominant overhead for parallel computing on distributed-

```
L1: DO I=1, 1000
      ... = A(3I-1) + A(3I) + A(3I+1)
    EndDO
L2: DO I=1, 999, 2
      ... = A(3I-1) + A(3I) + A(3I+1)
    EndDO
```

Fig. 1 Array reference analysis (example).

```
L3: DO I=4, 1000, 4
      A(I) = ...
    EndDO
L4: DO I=1, 999, 2
      ... = A(3I-1) + A(3I) + A(3I+1)
    EndDO
```

Fig. 2 Data dependence analysis (example 1).

```
L5: DO I=4, 1000, 4
      A(I) = ...
    EndDO
L6: DO I=1, 999, 2
      ... = A(3I)
    EndDO
```

Fig. 3 Data dependence analysis (example 2).

memory multiprocessors, parallelizing compilers for distributed-memory multiprocessors must perform array reference analysis for partitioning and distributing arrays on distributed memories to reduce remote memory accesses or inter-processor communications. The modulo interval can be employed as a subarray representation for array reference analysis and its arithmetic operations are useful to obtain subarrays referenced in loops.

The index I of Loop $L1$ shown in **Fig. 1** takes integers in $[1,1000]_{1(0)}$. The subscript of Array A takes the set of integers evaluated as follows by using modulo interval arithmetic in execution of $L1$.

$$\begin{aligned}&(3*[1,1000]_{1(0)}-1)\cup(3*[1,1000]_{1(0)})\\&\cup(3*[1,1000]_{1(0)}+1)\\&=([3,3000]_{3(0)}-1)\cup([3,3000]_{3(0)})\\&\quad\cup([3,3000]_{3(0)}+1)\\&=[2,2999]_{3(2)}\cup[3,3000]_{3(0)}\cup[4,3001]_{3(1)}\\&=[2,3001]_{1(0)}\end{aligned}$$

Thus $A(2:3001)$ will be referenced in $L1$.

The index I of Loop $L2$ shown in Fig. 1 takes integers in $[1,999]_{2(1)}$. The subscript of Array A takes the set of integers evaluated as follows in execution of $L2$.

$$\begin{aligned}&(3*[1,999]_{2(1)}-1)\cup(3*[1,999]_{2(1)})\\&\cup(3*[1,999]_{2(1)}+1)\\&=([3,2997]_{6(3)}-1)\cup([3,2997]_{6(3)})\\&\quad\cup([3,2997]_{6(3)}+1)\\&=[2,2996]_{6(2)}\cup[3,2997]_{6(3)}\cup[4,2998]_{6(4)}\end{aligned}$$

Thus $A(2:4)$, $A(8:10)$, $A(14:16)$, ..., and $A(2996:2998)$ will be referenced in $L2$.

4.2 Data Dependence Analysis

For two tasks T_1 and T_2 of a program, T_1 and T_2 cannot run in parallel if T_1 or T_2 reads the value of a variable written by the other task, that is, there exists a data dependence between T_1 and T_2. Let $O(T_1)$ denote the set of variables written by T_1 and $I(T_2)$ the set of variables read by T_2. If $O(T_1)\cap I(T_2)=\emptyset$, T_1 and T_2 can run in parallel since there exists no data dependence between T_1 and T_2 (Bernstein's condition[5)]).

Consider whether Loops $L3$ and $L4$ in **Fig. 2** can run in parallel or not. The subscript of Array A takes the set of integers in $[4,1000]_{4(0)}$ in execution of $L3$ and the set of integers in $[2,2996]_{6(2)}\cup[3,2997]_{6(3)}\cup[4,2998]_{6(4)}$ in execution of $L4$. ($L2$ and $L4$ are the identical loop. The set of integers taken by the subscript of Array A is evaluated in Section 4.1.) The intersection of these modulo intervals is not empty as follows.

$$\begin{aligned}&[4,1000]_{4(0)}\\&\cap([2,2996]_{6(2)}\cup[3,2997]_{6(3)}\cup[4,2998]_{6(4)})\\&=([4,1000]_{4(0)}\cap[2,2996]_{6(2)})\\&\cup([4,1000]_{4(0)}\cap[2,2997]_{6(3)})\\&\cup([4,1000]_{4(0)}\cap[2,2998]_{6(4)})\\&=[8,992]_{12(8)}\cup[4,1000]_{12(4)}\\&\neq\emptyset\end{aligned}$$

Since the intersection shown above is given as the exact set not as a superset, it is concluded that $L3$ and $L4$ cannot run in parallel absolutely.

Next, consider whether Loops $L5$ and $L6$ in **Fig. 3** can run in parallel or not. The modulo intervals representing sets of integers taken by subscripts of Array A in executions of $L5$ and $L6$ are $[4,1000]_{4(0)}$ and $[3,2997]_{6(3)}$, respectively. The intersection of the modulo intervals is empty as follows by Property 6.

$$[4,1000]_{4(0)}\cap[3,2997]_{6(3)}=\emptyset$$

Since there exists no dependence between $L5$

```
L7: DO I=4, 1000, 4
S:     A(3I) = ...
T:     ... = A(2I-1)
    EndDO
```

Fig. 4 Data dependence analysis (example 3).

and $L6$, the loops can run in parallel. If interval arithmetic is used for this analysis instead of modulo interval arithmetic, a false data dependence between $L5$ and $L6$ will be reported as follows.

$$[4,1000] \cap [3,2997] = [4,1000] \neq \emptyset$$

Modulo interval arithmetic is effective for data dependence analysis in loops with strides.

Finally, consider whether iterations of Loop $L7$ in **Fig. 4** can run in parallel or not. If there exist i and i' included in $[4,1000]_{4(0)}$ such that $3i = 2i' - 1$, the array element written by Statement S at $I = i$ is read by Statement T at $I = i'$ or the array element read by Statement T at $I = i'$ is written by Statement S at $I = i$. Since a flow dependence and an anti-dependence exist between S and T in the former and the latter cases, respectively, iterations of Loop $L7$ cannot run in parallel. To check if there exist such i and i', it is enough to check whether the set of integers taken by $3i - 2i' + 1$ includes zero or not. i and i' take integers in $[4,1000]_{4(0)}$ and thus the set of integers taken by $3i - 2i' + 1$ is evaluated as follows.

$$\begin{aligned} &3 * [4,1000]_{4(0)} - 2 * [4,1000]_{4(0)} + 1 \\ &= [12,3000]_{12(0)} - [8,2000]_{8(0)} + 1 \\ &= [-1987,2993]_{4(1)} \end{aligned}$$

Obviously, $0 \notin [-1987,2993]_{4(1)}$. This proves iterations in $L7$ can run in parallel.

These data dependence analysis by modulo interval arithmetic are equivalent to the famous GCD test.

4.3 Value Range Analysis

It is often required to analyze value ranges taken by expressions or variables at specified points of a given program for parallelization and code optimization. Value range information of array subscripts is useful for data dependence analysis. Value range information of a variable is useful to reduce memory consumption by demoting the variable to a smaller data type. Especially in recent years researchers in silicon compilation tackle with value range analysis to reduce silicon area or power consumption [14),17),20)].

Modulo interval arithmetic is also useful for value range analysis and can provide more detailed value range information than value range analysis with interval arithmetic. In our preliminary work [13)] value range analysis with modulo intervals is modeled as a problem of the monotone data flow system [21)]. Value range information of variables at arbitrary points of a given program is represented by modulo intervals and propagated in the control flow graph of the program. At each node of the control flow graph, propagated value range information are merged with modulo interval set operations and the merged information is processed with modulo interval arithmetic operations corresponding to ordinary arithmetic operations included in the assignment statement of the node. This procedure is performed iteratively at every node until value range information held at every node converges.

5. Related Works

The idea of the modulo interval is motivated by a previous work of our research project on loop parallelization and processor assignment [8)]. In the work a set of loop iteration index values is represented by a *coset*, denoted by $n\boldsymbol{Z} + r$, a set of integers which are divisible by n with a remainder of r. The coset is used only for representation and no operation is defined on cosets in the work.

Paek, et al.[15)] proposes Linear Memory Access Descriptor, or LMAD, as an array reference descriptor. The following is a formal description of LMAD.

$$\mathcal{A}^{\delta_{i_1},\delta_{i_2},\ldots,\delta_{i_d}}_{\sigma_{i_1},\sigma_{i_2},\ldots,\sigma{i_d}} + \tau$$

LMAD represents multi-dimensional subarrays by linearizing the original multi-dimensional arrays which include them. δ_{i_k} is the stride of array scanning by the loop of nest level k. σ_{i_k} is the span of array scanning by the loop of nest level k, that is, the difference of the offsets of the leftmost and the rightmost elements scanned by the loop of nest level k. τ is the offset of the leftmost element scanned by the loop nest. The modulo interval itself has less capability than LMAD to represent a subarray. However, the modulo interval represents a subarray equivalently to LMAD by representing the residue with a modulo interval recursively. Although most mathematical properties on modulo intervals still hold on such recursive modulo intervals, recursive modulo intervals are outside the scope of this paper.

Most subarray representations, including LMAD, have only set operations and operations to simplify their representations, since they are specially designed for array reference analysis. On the other hand, the modulo interval has simple and well-defined arithmetic operations based on the number theory and interval arithmetic as well as set operations. These operations have no difficulty for implementation. Array reference analysis and data dependence analysis require set operations. Scalar analysis, such as value range analysis, requires arithmetic operations for static evaluation of expressions. Program information obtained or required by these analyses is interrelated. Since the modulo interval has both arithmetic operations and set operations, the modulo interval is advantageous as a universal representation for program information handled by various program analysis techniques. Moreover, the modulo interval can be combined with existing subarray representations for more accurate analysis due to its simplicity, generality and manageability.

6. Conclusion

In this paper the modulo interval and its mathematical properties have been presented. Since the modulo interval is a subset of the interval, the modulo interval has mathematical properties similar to ones of the interval. The modulo interval represents a set of integers more accurately than the interval with additional information to intervals, namely the modulus and the residue. The modulus and the residue of the modulo interval are useful to represent integers taken by loop indices or array subscripts.

Modulo interval arithmetic is an extension of interval arithmetic and contains arithmetic operations and set operations. Both arithmetic operations and set operations have interesting mathematical properties derived from the number theory. The modulo interval is dealt with as numbers or as sets depending on context of program analysis. Moreover, this paper has demonstrated how to apply modulo interval arithmetic to program analysis performed by parallelizing compilers: array reference analysis, data dependence analysis and value range analysis.

Most properties of modulo interval arithmetic described in Section 3 are provided as inclusion relations not as equivalent relations. Moreover, an arithmetic operation of modulo intervals of different moduli provides a subset of a modulo interval whose modulus is the greatest common divisor of the moduli of operand modulo intervals as a result. The result of modulo interval arithmetic can be estimated bigger than the exact result. This degrades accuracy of program analysis and reduces opportunity of parallelization and code optimization. A sophisticated application of modulo interval arithmetic should be performed to keep the result of modulo interval arithmetic tightly close to the exact result. It is a future work to develop an algorithm of performing modulo interval arithmetic with smaller errors. See Ref. 16) for a partial work to reduce errors of modulo interval arithmetic.

Acknowledgments The authors would like to thank Prof. Fujio Kako, Ms. Noriko Soyama, and Ms. Kazuko Kambe of Nara Women's University for their valuable comments and suggestions. This work is supported by "Research for the Future" Program of Japan Society for the Promotion of Science.

References

1) Balasundaram, V.: A Mechanism for Keeping Useful Internal Information in Parallel Programming Tools: The Data Access Descriptor, *J. Parallel and Distributed Computing*, Vol.9, No.2, pp.154–170 (1990).

2) Banerjee, U.: *Dependence Analysis for Supercomputing*, Kluwer Academic Publishers (1988).

3) Callahan, D. and Kennedy, K.: Analysis of Interprocedural Side Effects in a Parallel Programming Environment, *J. Parallel and Distributed Computing*, Vol.5, No.5, pp.517–550 (1988).

4) Graham, R.L., Knuth D.E. and Patashnik O.: *Concrete Mathematics*, Addison-Wesley (1989).

5) Hwang, K., *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill (1993).

6) Harrison, W.H.: Compiler Analysis of the Value Ranges for Variables, *IEEE Trans. Softw. Eng.*, Vol.SE-3, No.3, pp.243–250 (1977).

7) Kambe, K. and Kako, F.: Reduction of Quadratic Equations for Data Dependence Tests, *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications* (*PDPTA*) *2000*, Vol.IV, pp.1874–1880 (2000).

8) Kitasuka, T., Joe, K., Schouten, D. and Fukuda A.: A Loop Parallelization Technique

for Linear Dependence Vector, *Proc. IFIP WG10.3 Working Conf. on Parallel Architectures and Compilation Techniques* (*PACT*) *'95*, pp.285–289 (1995).

9) Koelbel, C.H., Loveman, D.B., Shreiber, R.S., Steele Jr., G.L. and Zosel, M.E.: *The High Performance Fortran Handbook*, MIT Press (1994).

10) Lawson, C.L., Hanson, R.J., Kincaid, D. and Krogh, F.T.: Basic Linear Algebra Subprograms for FORTRAN usage, *ACM Trans. Math. Softw.*, Vol.5, No.3, pp.308–323 (1979).

11) Li, Z. and Yew, P.-C. Efficient Interprocedural Analysis for Program Parallelization and Restructuring, *Proc. ACM/SIGPLAN Parallel Programming: Experience with Applications, Languages and Systems* (*PPEALS*) *1988*, pp.85–97 (1988).

12) Moore, R.E.: *Interval Analysis*, Prentice-Hall (1966).

13) Nakanishi, T. and Fukuda, A.: Design of Modulo Interval Propagation in the JSPS MIRAI Compiler (in Japanese), *IPSJ SIG Notes* (*2000-HPC-82*), Vol.2000, No.73, pp.95–100 (2000).

14) Ogawa, O., Takagi, K., Itoh, Y., Kimura, S. and Watanabe, K.: Hardware Synthesis from C Programs with Estimation of Bit Length of Variables, *IEICE Trans. Fundamentals*, Vol.E82-A, No.11, pp.2338–2346 (1999).

15) Paek, Y., Hoeflinger, J. and Padua, D.: Simplification of Array Access Patterns for Compiler Optimizations, *Proc. 1998 ACM SIGPLAN Conf. on Programming Language Design and Implementation* (*PLDI*), pp.60–71 (1998).

16) Soyama, N., Nakanishi, T., Kako, F. and Fukuda, A.: An Algorithm of Modulo Interval Arithmetic and Its Applications (in Japanese), *IPSJ SIG Notes* (*2000-ARC-139*), Vol.2000, No.74, pp.43–48 (2000).

17) Stephenson, M., Babb, J. and Amarasinghe, S.: Bitwidth Analysis with Application to Silicon Compilation, *Proc. 2000 ACM SIGPLAN Conf. on Programming Language Design and Implementation* (*PLDI*), pp.108–120 (2000).

18) Sunaga, T.: Theory of an Interval Algebra and Its Application to Numerical Analysis, *RAAG Memoirs*, Vol.2, pp.29–46 (1958).

19) Triolet, R., Irigion, F. and Feautrier, P.: Direct Parallelization of CALL Statements, *Proc. SIGPLAN '86 Symp. on Compiler Construction*, pp.176–185 (1986).

20) Yamashita, H., Tomiyama, H., Inoue, A., Nurprasetyo, E.F., Okuma, T. and Yasuura, H.: Variable Size Analysis for Datapath Width Optimization, *Proc. Asia Pacific Conf. on Hardware Description Languages* (*APCHDL '98*), pp.69–74 (1998).

21) Zima, H.: *Supercompilers for Parallel and Vector Computers*, ACM Press (1990).

(Received September 10, 2000)
(Accepted December 1, 2000)

Tsuneo Nakanishi was born in 1970. He received the B.E. degree in communication engineering from Osaka University, Japan, in 1993; and the M.E. and D.E. degrees in information science from Nara Institute of Science and Technology, Japan, in 1995 and 1998, respectively. From 1996 to 1998, he was a research fellow of the Japan Society for the Promotion of Science. From 1998 he has been an assistant professor of Graduate School of Information Science, Nara Institute of Science and Technology, Japan. His research interests include compilers, embedded systems and parallel computing. He is a member of the ACM, the IEEE Computer Society and the IPSJ.

Akira Fukuda was born in 1954. He received the B.E., M.E. and D.E degrees in computer science and communication engineering from Kyushu University in 1977, 1979 and 1985, respectively. From 1977 to 1981, he worked for the Nippon Telegraph and Telephone Corporation, where he engaged in research on performance evaluation of computer systems and the queueing theory. From 1981 to 1991 and from 1991 to 1993, he worked for the Department of Information Systems and Department of Computer Science and Communication Engineering, Kyushu University, Japan, respectively. From 1994 he has been a professor of Graduate School of Information Science, Nara Institute of Science and Technology, Japan. His research interests include parallel and distributed operating systems, parallelizing compilers and performance evaluation of computer systems. He received 1990 IPSJ Research Award and 1993 IPSJ Best Author Award. He is a member of the ACM, the IEEE Computer Society, the IEICE, the IPSJ and the Operations Research Society of Japan.