Dependence Testing and Vectorization of Multimedia Agents

John G. Allen, Richard Y. D. Xu
Faculty of Information Technology
University of Technology
Sydney, NSW, 2007, Australia.

Jesse S. Jin
Faculty of Design, Communication and IT
University of Newcastle
Callaghan, NSW, 2308, Australia.

Abstract - We present a dependence testing algorithm that considers the short width of modern SIMD registers in a typical microprocessor. The test works by solving the dependence system with the generalized GCD algorithm and then simplifying the solution equations for a particular set of dependence distances. We start by simplifying each solution lattice to generate points that satisfy some small constant dependence distance that corresponds to the width of the register being used. We use the Power Test to efficiently perform Fourier-Motzkin Variable Elimination on the simplified systems in order to determine if dependences exist. The improvements described in this paper also extend our SIMD dependence test to loops with symbolic and triangular lower and upper bounds as well as array indices that contain unknown symbolic additive constants. The resulting analysis is used to guide the vectorization pass of a dynamic multimedia compiler used to compile software agents that process audio, video and image data. We fully detail the proposed dependence test in this paper, including the related work.

Keywords: vectorization, SIMD, dependence, multimedia, parallelism.

1. Introduction

Vectorization of multimedia code, particularly image processing instructions, not only resolves many speed bottlenecks under a single processor environment, it also enables many algorithms to be applied in real-time.

Single Instruction Multiple Data (SIMD) instructions allow multiple data elements to processed in parallel by a single instruction, which permits a small amount of instruction level parallelism to be achieved when SIMD instructions can be applied. This parallelism translates directly to performance improvements in such processor demanding tasks as speech and video coding, image effects and real-time video processing.

The availability of dependence information is still one of the most important factors in determining the legality of transforming scientific and multimedia code into vector form. Historically this has been achieved through a sequence of complex loop transformations such as loop fission and scalar expansion [1], where the vectorization of single statements is determined by the acyclic condensation of the dependence graph [1, 7].

However, this kind of vectorization was not designed with short SIMD parallelism in mind, where register lengths are small and loop unrolling methods are often preferred [2]. It has been observed from such methods that, for dependence to dependence exist, it is a necessary condition the distance between dependent iterations be smaller than the width of the SIMD register being used [3].

This class of dependence testing is often ignored in the literature in favor of methods that try to determine the existence of solutions anywhere within the lower and upper bound of the loop index variables [1, 4, 7]. Furthermore, systems that aim to vectorize scientific and multimedia code often use the results of such tests to guide the legality of their vectorization methods. We believe that such dependence tests are often unsuitable or irrelevant in the context of dependence testing for short SIMD parallelism and hence are sources of imprecision.

The D-test has recently been proposed in the literature by Bulic *et al* [3]. It is an extension to the Banerjee inequalities that takes into account the number of elements processed in the SIMD register. It is not clear whether this test considers real solutions that give rise to the short minimum or maximum dependence distance. It is also not clear whether this test may be applied to triangular loops or loops with symbolic lower and upper bounds other than at run-time. Whether this test can be extended to testing particular direction vectors is not discussed.

The Power Test [5] is a novel test that seeks to reduce the worst-case exponential time complexity of Fourier-Motzkin Variable Elimination (FMVE) by solving a linear programming problem rather than an integer one using LP-relaxation. While SIMD dependence testing is not discussed with this technique we argue that there exists an extension that would make this possible by introducing an additional inequality corresponding to the width of the SIMD register.

The Omega Test [6] is an extension of FMVE to Integer Programming and is well suited to accurate dependence testing due to its all-integer property. Due to its accuracy and the availability of a solid implementation, the test is often considered as a benchmark when comparing a range of dependence tests. We argue that the Omega test is strongly applicable to the class of SIMD dependence testing we are describing in a similar treatment to [5]. That is, by appending additional integer constraints that bound the distance between dependant iterations we propose that we could prune those dependences that do not affect short SIMD vectorization from the Data Dependence Graph.

Other popular tests such as the GCD test, Delta test [1, 4] and the Banerjee inequalities are in their basic form unsuitable for the described class of dependence testing, however application of Banerjee inequalities to SIMD vectorization dependence testing is described in [3]. Since our test has adequately addressed the dependence problem in our compiler, we have left application of the other methods to SIMD testing for future study.

This paper is organized as follows. Section 2 describes the required preliminaries, definitions and conventions used within this paper. Section 3 describes our modified approach based on lattice simplification and some extension to symbolic and triangular bounds. Section 4 describes the evaluation of the simplified inequality system based on the Power Test. Section 5 gives some qualitative results using an example as well as discusses some implementation details based on the SUIF2 framework.

2. Dependence Testing

Our treatment of dependence testing is applicable to array reference pairs with at least one non-constant dependence distance. Note that for simplicity we assume perfectly nested loops.

2.1 Iteration Space

In this paper, we consider normalized perfectly nested loops of depth n where the array indexes are simple linear combinations of the index variables or symbolic constants and the array references have s subscript positions.

```
do $i_1 = l_1, u_1$
  ...
  do $i_n = l_n, u_n$
  $S_1$   $A(f_1(i_1,\dots i_n),\cdots,f_s(i_1,\dots i_n)) = \dots$     (2.1)
  $S_2$   $\dots = A(g_1(i_1,\dots i_n),\cdots,g_s(i_1,\dots i_n))$
  enddo
enddo
```

In this paper the loop limits l_1, u_1 may contain constants and additive symbolic expressions and l_k, u_k are linear functions of $i_1,\dots,i_{k-1}$ for $1 \le k \le n$. The iteration vector $\vec{i}$ for a given iteration of the innermost loop is an element in $\mathbb{Z}^n$ where $\vec{i} = \{i_1, i_2, \dots, i_n\}$ contains the iteration numbers for each of the loops in order of nesting level.

The *iteration space* Φ contains the set of vectors $\vec{i}$ in $\mathbb{Z}^n$ bounded by the loop limits:

$$\Phi = \{\vec{i} \mid l_k \le i_k \le u_k, k = 1,\dots,n\} \qquad (2.2)$$

Index vectors $\vec{i}$ and $\vec{j}$ are *lexicographically ordered* denoted by $\vec{i} \prec \vec{j}$, iff $\vec{i} < \vec{j}$ or there exists an index $l : 1 \le l \le n$ such that $i_1 = j_1 \dots i_{l-1} = j_{l-1}$ and $i_l < j_l$.

2.2 Dependence Distance

If two iterations i_k, j_k access the same memory location and at least one is a write and $\vec{i} \prec \vec{j}$ then the iteration $\vec{j}$ depends on $\vec{i}$, denoted $\vec{i}\,\delta\,\vec{j}$ and *distance vector* is $\vec{d} = \vec{j} - \vec{i}$. Since $\vec{i} \prec \vec{j}$ we have *true dependence* if $\vec{d} \succ \vec{0}$.

2.3 Dependence System

Considering the perfect loop nest described in Section 2.1, a dependence exists from S_1 to S_2 if and only if there exits integer vectors $\vec{i}$ and $\vec{j}$ where $\vec{i} \prec \vec{j}$ and $f(\vec{i}) = g(\vec{j})$.

The array subscripts can be written as $f(\vec{i}) = \vec{i}^T\mathbf{A} + \vec{a}$ where the coefficients and additive expressions are integer constants. The *linear dependence equation* may be conveniently written as:

$$\vec{i}^T\mathbf{A} + \vec{a} = \vec{j}^T\mathbf{B} + \vec{b} \qquad (2.3)$$

Where $\mathbf{A},\mathbf{B} \in \mathbb{Z}^{n\times s}$ and $\vec{a},\vec{b} \in \mathbb{Z}^n$. This is the class of well-known Diophantine equations for which a number of algorithms exist that determine the existence of integer solutions in polynomial time. To determine the set of integer solutions we formulate the dependence system as follows:

$$(\vec{i};\vec{j})\begin{pmatrix}\mathbf{A}\\-\mathbf{B}\end{pmatrix}=\vec{b}-\vec{a} \qquad (2.4)$$

This may be written in the form:

$$(\vec{i};\vec{j})\mathbf{C}=\vec{c} \qquad (2.5)$$

In our implementation, we solve the Diophantine equations by reducing the matrix **C** to echelon form by applying a unimodular transformation **U**. This is achieved by applying a sequence of elementary row operations to **C** equivalent to Gaussian Elimination adapted to integers and applying the same row operations to the identity **I** to obtain a unimodular matrix **U**.

To solve the system of equations we replace 2.5 with an equivalent system by a unimodular transformation:

$$(\vec{i};\vec{j})=\vec{t}\mathbf{U} \qquad (2.6)$$

$$\vec{t}\mathbf{U}\mathbf{C}=\vec{c} \qquad (2.7)$$

Where **U** is a unimodular matrix satisfying $\det(\mathbf{U})=\pm 1$ where a unimodular $\mathbf{U}^{-1}$ exists such that $\vec{t}=(\vec{i};\vec{j})\mathbf{U}^{-1}$. Constructing **U** via Gaussian elimination such that **UC** = **S** is echelon generates the equation:

$$\vec{t}\mathbf{S}=\vec{c} \qquad (2.8)$$

From Equation 2.8 we can solve for some of the components of $\vec{t}$ by simple back-substitution. Note that if **S** has rank $r=\text{rank}(\mathbf{S})$ then the solution vector $\vec{t}$ will contain $m=2n-r$ unsolved variables. The remaining unsolved components of $\vec{t}$ are *free variables* for which any integer value will generate a solution to 2.3. It is sufficient to write $\vec{t}=(\vec{t}_1;\vec{t}_2)$ where $\vec{t}_1$ are constant and $\vec{t}_2$ are free variables. Substituting $\vec{t}$ into 2.6 gives the set of all solutions denoted by $(\vec{i};\vec{j})$, i.e.:

$$\begin{aligned}\vec{i}&=\vec{t}\mathbf{U}_l\\ \vec{j}&=\vec{t}\mathbf{U}_r\end{aligned} \qquad (2.9)$$

Where $\mathbf{U}_l$ and $\mathbf{U}_r$ are the left and right sub-matrices of **U** such that $\mathbf{U}=[\mathbf{U}_l \mid \mathbf{U}_r]$. We are now able to define the *dependence distance* as:

$$\vec{d}=\vec{t}(\mathbf{U}_r-\mathbf{U}_l) \qquad (2.10)$$

We also define a vector $\vec{b}_k$ that represents the variable component of the dependence distance at level k (corresponding to the free variables) and an integer constant c_k that represent the scalar component of the dependence distance at level k:

$$d_k=\vec{b}_k\cdot\vec{t}+c_k \qquad (2.11)$$

For the remainder of this paper we will assume that $\vec{b}_k\neq\vec{0}\ \exists k, 1\le k\le n$ since the inverse case has a trivial SIMD dependence test that cannot be improved with our methods. We will also write the set of all solutions to 2.5 where $\vec{x}$ is an initial solution and $\hat{\mathbf{B}}$ is a basis matrix whose columns spans the null space of **C**:

$$(\vec{i};\vec{j})=\vec{x}+\hat{\mathbf{B}}\vec{t} \qquad (2.12)$$

Where $\vec{t}$ is the vector consisting of the unsolved free variables. In our discussions we will assume that the null space basis matrix is full rank.

3. Symbolic Dependence Testing

In this section we describe our approach to short SIMD vectorization dependence testing based on lattice simplification and Integer Programming.

3.1 Generalized Dependence Distance

While we have a simplification for dependence equations with that have dependence distance that depends on a single free variable, this simplification is omitted here. We proceed with the more general, multiple free variable case.

We start with Equation 2.11 describing the dependence distance where $\vec{b}_k$ is non-zero.

Let w = *VL*-1 (where *VL* is the vector stride), we can solve for the values of the free variables that give a specific dependence distance. Let us define the following sets of equations for the (<) direction vector carried at level k:

$$(<)\begin{cases} \vec{b}_k \cdot \vec{t} = 1 - c_k \\ \vdots \\ \vec{b}_k \cdot \vec{t} = w - c_k \end{cases} \tag{3.1}$$

And similarly for the (>) direction vector:

$$(>)\begin{cases} \vec{b}_k \cdot \vec{t} = -1 - c_k \\ \vdots \\ \vec{b}_k \cdot \vec{t} = -w - c_k \end{cases} \tag{3.2}$$

And finally for the (=) direction vector:

$$(=)\begin{cases} \vec{b}_k \cdot \vec{t} = -c_k \end{cases} \tag{3.3}$$

Using 3.1-3.3 we can solve for the (possibly infinite) values of $\vec{t}$ that generate solutions to Equation 2.5 that satisfy a particular dependence distance when substituted into Equation 2.12.

The question we wish to ask is whether we can generate all such points in the form of a new solution equation that replaces 2.12 for a particular dependence distance. Let us arbitrarily choose the (<) direction vector and a particular dependence distance $d \in \{1,\ldots,w\}$. We start by augmenting $\mathbf{A}' = \vec{b}_k^{\mathrm{T}}$ with the identity to produce $\mathbf{IA}'$ where $\mathbf{A}' \in \mathbb{Z}^{m\times 1}$ ($\mathbf{A}'$ looks like a length m column vector in $\mathbb{Z}^{m\times 1}$) and reducing $\mathbf{A}'$ to upper triangular using the technique described in Section 2. Assuming $\mathbf{A}'$ has an inverse we end up with the following:

$$\vec{t}'\mathbf{S}' = d - c_k \tag{3.4}$$

Where $\mathbf{S}'$ is in row-reduced echelon form. In this way we can test for the existence of integer solutions for each of the possible distances d. If there are no solutions for any dependence distance then we have no dependence. If at least one solution exists then we can solve for $\vec{t}'$ to obtain:

$$\vec{t} = \vec{x}' + \hat{\mathbf{B}}'\vec{t}' \tag{3.5}$$

This is the set of all vectors $\vec{t}$ that generate a particular dependence distance d when substituted into Equation 2.12. Substituting the points $\vec{t}$ into 2.12 gives us the desired result:

$$\begin{aligned}(\vec{i},\vec{j}) &= \vec{x} + \hat{\mathbf{B}}\vec{t} = \vec{x} + \hat{\mathbf{B}}(\vec{t}'\mathbf{U}') \\ &= \underbrace{\vec{x}}_{\text{initial solution}} + \underbrace{\hat{\mathbf{B}}\vec{x}'}_{\text{depends on } d} + \underbrace{\hat{\mathbf{B}}\hat{\mathbf{B}}'\vec{t}'}_{\text{constant for all } d}\end{aligned} \tag{3.6}$$

Proof: Assume $\operatorname{rank}(\mathbf{S}') = 1$. This implies that $\vec{t}' \in \mathbb{Z}^{m-1}$ has m-1 free variables. Expanding 3.5 gives the equations:

$$\begin{aligned} t_k &= x_k' + \alpha_{k,1}'\vec{t}_1' + \ldots + \alpha_{k,m-1}'\vec{t}_{m-1}' \\ &\forall k, 1 \le k \le m-1 \end{aligned} \tag{3.7}$$

Substituting the equations for $\vec{t}$ into Equation 2.12 gives Equation 3.6.

The motivation behind Equation 3.6 is to determine the set of solutions to the original dependence equations that satisfy a constant dependence distance d. According to Equation 3.6, changing d only changes the initial solution of the new result set. Note that this is a simple substitution of the free variables $\vec{t}$ with the corresponding variables in $\vec{t}'$ (with one less free variable) where matrix product is used to perform the substitution.

3.2 Implementation of the Algorithm

To test for a particular direction vector at level k we start by deriving the equivalent system as described by Equation 2.7. We then test for integer solutions to t_1' for each particular distance d_i according to Table 1.

Table 1. Distance sets for each direction vector.

Direction Vector	Distance Set
(<)	$d_i \in \{1,\ldots,w\}$
(>)	$d_i \in \{-1,\ldots,-w\}$
(=)	$d_i \in \{0\}$

Since $\mathbf{S}'$ is row-reduced echelon, it is inexpensive to test each of these distances unless w is large. If any particular distance d_i has an integer solution then we generate for it an initial solution and null-space basis matrix according to Equation 3.5. The initial solution and the basis matrix of the null-space basis matrix are then used to derive the solution set according to Equation 3.6. At this point we have the sub-system defined by Equation 3.9.

$$\vec{l} - (\vec{x} + \hat{\mathbf{B}}\vec{x}') \le \hat{\mathbf{B}}\hat{\mathbf{B}}'\vec{t}' \le \vec{u} - (\vec{x} + \hat{\mathbf{B}}\vec{x}') \tag{3.8}$$

The matrix $\hat{\mathbf{B}}\hat{\mathbf{B}}'$ is constant for all values of d and has one less column than the basis matrix $\hat{\mathbf{B}}$, which spans the original null space. Since the matrix is constant, the projection matrices may be reused for each distance being tested. Only the RHS need be projected to test for a feasible system of inequalities.

To prove the existence of integer points in the region bounded by the inequalities defined in Equation 3.9 we may use any appropriate integer programming method. However, since $\hat{\mathbf{B}}\hat{\mathbf{B}}'$ has one less column than the original basis, we can perform a *consistency check* on each row of system 3.9 prior to resorting to such methods.

3.3 Unknown Additive Symbolic Constants

Symbols and constants that appear anywhere within the loop bounds or array indices are appended to the dependence system as single variables in the same fashion as in [5]. For such symbols we first verify that they are not modified anywhere within the loop nest (modified variables require a more elaborate scheme such as Induction Variable Substitution [1]).

3.4 Symbolic and Triangular Bounds

We represent the trapezoidal dependence system as follows where $\mathbf{L}$ and $\mathbf{U}$ are sparse matrices in $\mathbb{Z}^{m \times m}$ that select the index variables and unknown symbolic quantities determined by Section 3.3 to form trapezoidal or triangular bounds. Since the solution equations provide a mapping from program variables to free variables, we can write the dependence system as follows to derive inequalities that only depend on free variables generated by the solution equations:

$$\vec{l} + \mathbf{L}\left(\vec{x} + \hat{\mathbf{B}}\vec{t}\right) \leq \vec{x} + \hat{\mathbf{B}}\vec{t} \leq \vec{u} + \mathbf{U}\left(\vec{x} + \hat{\mathbf{B}}\vec{t}\right) \qquad (3.9)$$

This works by replacing variables in the lower and upper bound with their corresponding free variables using the relationship in Equation 3.5. Note that if none of the loop bounds are symbolic, then $\mathbf{L}$ and $\mathbf{U}$ will be zero matrices and the equation becomes the same as Equation 3.9 after applying the simplification in Section 3.1. Otherwise, after simplification and a little manipulation, we end up with the following bounds for use by the Power Test for the particular dependence distance d being tested:

$$\begin{aligned} \left(\left(\mathbf{I}-\mathbf{L}\right)\hat{\mathbf{B}}\hat{\mathbf{B}}'\right)\vec{t} &\geq \vec{l} - \left(\mathbf{I}-\mathbf{L}\right)(\vec{x} + \hat{\mathbf{B}}\vec{x}') \\ \left(\left(\mathbf{I}-\mathbf{U}\right)\hat{\mathbf{B}}\hat{\mathbf{B}}'\right)\vec{t} &\leq \vec{u} - \left(\mathbf{I}-\mathbf{U}\right)(\vec{x} + \hat{\mathbf{B}}\vec{x}') \end{aligned} \qquad (3.10)$$

Proof: By substitution of Eq. 3.6 into Eq. 3.10.

Application of the Power Test in this form is convenient as the matrix approach does not require any symbolic computation since each variable in the dependence system will be replaced by one of the so called "free" variables represented in a matrix form. Note that we actually have three inequalities generated for each row since we also have:

$$\vec{l} + \mathbf{L}\left(\vec{x} + \hat{\mathbf{B}}\vec{x}' + \hat{\mathbf{B}}\hat{\mathbf{B}}'\vec{t}'\right) \leq \vec{u} + \mathbf{U}\left(\vec{x} + \hat{\mathbf{B}}\vec{x}' + \hat{\mathbf{B}}\hat{\mathbf{B}}'\vec{t}'\right) \quad (3.11)$$

To generate actual bounds on the free variables using Equation 3.12 we may rewrite it in the following form:

$$\vec{l} - \vec{u} - \left(\mathbf{U}-\mathbf{L}\right)\left(\vec{x} + \hat{\mathbf{B}}\vec{x}'\right) \leq \left(\mathbf{U}-\mathbf{L}\right)\left(\hat{\mathbf{B}}\hat{\mathbf{B}}'\vec{t}'\right) \quad (3.12)$$

This is useful when there is at least one free variable involved in a lower or upper bound (so we can generate an additional bound for it), or if there are free variables in the lower and upper bounds that cancel out with one another to yield a constant comparison of the form $l_k - u_k \leq 0$. Note that a number of calculations in Equations 3.11-3.13 are independent of the dependence distance being tested and therefore may be calculated once only for every dependence distance (for a particular level k) being tested. To further improve performance, sparse computations may be used where it is appropriate to do so.

4. The Power Test

The Power Test [5] attempts to determine the feasibility of particular direction vectors by relaxing the Integer Programming (IP) problem to a Linear Programming one. Using the inequalities derived from dependence system in combination with inequalities implied by the direction vector being tested, the Power Test forms a set of bounds on each of the m free variables. If a contradiction is found for any free variable then the solution space is empty which implies no dependence for that direction vector.

4.1 Deriving Bounds

The Power Test starts with inequalities of the form:

$$l \leq b_k t_k + b_1 t_1 + \ldots + b_{k-1} t_{k-1} \leq u \qquad (4.1)$$

For a particular free variable t_k we have the bounds:

$$\begin{aligned} t_k &\le \left\lfloor (u - b_1 t_1 - \ldots - b_{k-1}t_{k-1})/b_k \right\rfloor \\ t_k &\ge \left\lceil (l - b_1 t_1 - \ldots - b_{k-1}t_{k-1})/b_k \right\rceil \end{aligned} \tag{4.2}$$

The floor and ceiling operators are necessary to keep results in the integer domain. If b_k exactly divides each of the free variables, then the bounds reduce to:

$$\begin{aligned} t_k &\le \left\lfloor \frac{u}{b_k} \right\rfloor - \frac{b_1}{b_k}t_1 - \ldots - \frac{b_{k-1}}{b_k}t_{k-1} \\ t_k &\ge \left\lceil \frac{l}{b_k} \right\rceil - \frac{b_1}{b_k}t_1 - \ldots - \frac{b_{k-1}}{b_k}t_{k-1} \end{aligned} \tag{4.3}$$

Equation 4.3 is used to generate a list of bounds for each free variable in the dependence system. The Power Test may then visit each pair of lower and upper bounds for each free variable to search for contradictions to prove independence. Note that when b_k is negative it is important to reverse the inequality in order to generate a sensible bound (e.g., this will convert a lower bound to an upper bound).

5. Results

The enhancements made to our SIMD dependence test allow us to perform dependence testing on a greater variety of loop nests, including those with symbolic or triangular loop bounds or with unknown symbolic quantities in a subscript position.

While no specific performance details are given in this paper, we have found that in general the use of the Power Test has allowed the test to perform some 20-30 times faster than a naïve implementation based on FMVE (which has exponential time complexity). While this test is not as efficient as the D-test (which is the only other specific test of this kind), we are able to statically provide symbolic dependence testing whilst determining whether solutions exist within the width of the SIMD register being tested.

We also find that our test provides a useful synergy with the Power Test by eliminating a variable from the dependence system and simplifying the solution equations by limiting the set of possible solutions.

5.1 Implementation

We have implemented the symbolic SIMD dependence test as pass module for SUIF 2 [9]. For-loops are normalized using a well-known algorithm to simplify dependence testing. To improve the accuracy, copy propagation, scalar expansion and function inlining are also performed. Simplification and standardization is applied to array index expressions and loop bound expressions by applying in-order visitors. At present no form of induction variable substation (IVS) is currently applied.

5.2 Example

In the following example, we modified the matrix **C** in Equation 2.5 such that the first *m* rows consist of alternating loop index variables for the source and sink of the dependence and the remaining variables correspond to unknown symbolic quantities. Consider the following perfect loop nest with a vector stride of $VL = 2$:

```
do $I_1 = 1,N$
 do $I_2 = 1,N$
 $S_1$  $A(I_1, I_2 + 1) = \ldots$
 $S_2$  $\ldots = A(I_1, I_1 + I_2)$
 enddo
enddo
```

The dependence system in the desired form is:

$$\begin{pmatrix} i_1 & j_1 & i_2 & j_2 & N \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -1 & -1 \\ 0 & 1 \\ 0 & -1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 \end{pmatrix}$$

Reducing the matrix **C** to upper triangular and applying the same sequence of elementary row operations to the identity we obtain **UC** = **S**. Since **U** is a unimodular transformation, the dependence equations may therefore be rewritten as:

$$\begin{pmatrix} i_1 & j_1 & i_2 & j_2 & N \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 \end{pmatrix}$$

This generates the solution vector $\vec{t} = (0, 1, t_3, t_4, t_5)$ from which we derive solutions using Equation 2.9:

$$\begin{aligned} \vec{i} &= (t_3 + 1, t_3 + t_4) \\ \vec{j} &= (t_3 + 1, t_4) \\ N &= t_5 \end{aligned}$$

The distance vector d for this system is therefore:

$$\vec{d} = \vec{j} - \vec{i} = (0, -t_3)$$

Since the dependence distance is non-constant at level $k = 2$ we may test for SIMD dependences using FMVE as follows. The sparse lower and upper bound selection matrices **L** and **U** are simple for this example, with N being the only symbol in the loop bounds. The solution equations in matrix 'orm are defined by:

$$(\vec{i};\vec{j};N) = \vec{x} + \hat{\mathbf{B}}\vec{t} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \vec{t}$$

Suppose we wish to test for dependence at level k = 2 when d_2 = 0, corresponding to (=) direction vector. We start by solving the distance equation $-t_3 = 0$ and applying the simplification described in Section 3.1.

$$d_2 = 0 \Rightarrow (\vec{i};\vec{j};N) = \vec{x} + \hat{\mathbf{B}} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \hat{\mathbf{B}} \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Using Equation 3.11 we generate the inequalities:

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \vec{t} \geq \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 & -1 \\ 0 & -1 \\ 1 & -1 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \vec{t} \leq \begin{pmatrix} -1 \\ -1 \\ 0 \\ 0 \\ \infty \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \vec{t} \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -\infty \end{pmatrix}$$

Projecting this system using the Power Test results in the following feasible bounds on the free variables:

$$1 \leq t_1' \leq t_2' \qquad \left.\begin{matrix} 1 \\ t_1' \end{matrix}\right\} \leq t_2' \leq \infty$$

Testing for a d_2 = -1 corresponding to the (>) direction vector also cannot prove independence. Finally, the distance d_2 = 1 corresponding to the (<) direction vector is disproved by the constant comparison $1 \leq 0$, giving us the set of direction vectors $\{(=,=), (=,>)\}$. Based in these results, the loop nest can be safely vectorized using SIMD instructions.

6. Conclusion

An approach to symbolic SIMD dependence testing is presented. The test uses a simplification to limit the set of potential dependences to have a distance between dependant iterations within the width of the SIMD register. This property, along with the ability to analyze complex loop regions, allows more scientific and multimedia code to be vectorized than with the classical approach. This translates to more parallel code (i.e., SIMD instructions) being generated in a compiler designed specifically for sound, image and video processing.

Since the simplification process successfully eliminates a free variable from the dependence system, the Power Test is more effective and computationally less expensive. In particular, we achieve close to the accuracy of the potentially expensive FMVE algorithm but with lower time complexity. We also give formulae that allow the entire test to be implemented with matrix arithmetic. In practice, we find this test alone to be sufficient to test the legality of automatic parallelization of multimedia code.

7. References

[1] J. R. Allen and K. Kennedy. "Optimizing Compilers for Modern Architectures." Academic Press, San Francisco CA, 2002.

[2] S. Larsen and S. Amarasinghe. "Exploiting Superword Level Parallelism with Multimedia Instruction Sets." *Proc. SIGPLAN Conference on Programming Language Design and Implementation*, Vancouver BC, June 2000.

[3] P. Bulic, V. Gustin. "Fast Data Dependence Analysis in a Multimedia Vectorizing Compiler." *Proc. 12th Euromicro Symposium on Parallel and Distributed Computing*, La Coruna Spain, 176-183, Feb 2004.

[4] U. Banerjee. "Dependence Analysis for Supercomputing." Kluwer Academic Publishers, Norwell MA, 1988.

[5] M. Wolfe, C. W. Tseng. "The Power Test for Data Dependence." *IEEE Transactions on Parallel and Distributed Systems archive*, 3(5): 591-601, Sep 1992.

[6] W. Pugh. "The Omega Test. A Fast and Practical Integer Programming Algorithm for Dependence Analysis." *Comm. ACM*, 35(8): 102-114, Aug 1992.

[7] H. Zima, B. Chapman. "Supercompilers for Parallel and Vector Computers." ACM Press, NY, 1991.

[8] Yijun Yu, Erik H. D'Hollander. "Partitioning Loops with Variable Dependence Distances." *Proc. ICPP 2000*, 209-218, 2000.

[9] SUIF2 website. http://suif.stanford.edu/suif/suif2/. Accessed 01-Dec-2004.