Department of Computer Science
University of Manchester

Manchester M13 9PL, England

Technical Report Series

UMCS–92–1–1

Michael O'Boyle and G.A. Hedayat

Load Balancing of Parallel Affine Loops by Unimodular Transformations

Load Balancing of Parallel Affine Loops by Unimodular Transformations*

Michael O'Boyle and G.A. Hedayat
Department of Computer Science, University of Manchester,
Oxford Road, Manchester M13 9PL, UK
mob@cs.man.ac.uk gholam@cs.man.ac.uk

Abstract

This paper is concerned with the automatic mapping of array computation to processors efficiently. One of the major overheads associated with the mapping of computation is load imbalance. An optimising compiler should find a mapping such that this overhead is minimised.

This paper describes formally the mapping of loop iterations to processors so as to minimise load imbalance. The class of perfectly load balanced affine loops is defined whereby using unimodular transformations, it is shown that a large class of loops are equivalent.

An algorithm is detailed which can determine both whether a loop structure may be load balanced, and the necessary transformations to do so. This algorithm has a worst case complexity of $O(m^3)$ where m is the dimensionality of the iteration space. The analysis is extended to the case when many loops are to be partitioned where it is shown that a transformation may be constructed which simultaneously balances all appropriate loops. Finally it is shown that if a loop is perfectly load balanced, then there exists a transformation such that it may be placed outermost so as to aid partitioning.

Technical Reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from m1.cs.man.ac.uk (130.88.13.4) in the directory /pub/TR. The files are stored as PostScript, in compressed form, with the report number as filename. Alternatively, reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, U.K.

1 Introduction

Compilation should minimise parallel time by utilising machine parallelism and reducing overhead. The first stage of compilation is therefore to identify and match program parallelism to machine parallelism. We define machine parallelism simply as the number of processors p. It is necessary to identify and divide a nested loop array computation into p sub computations.

It is assumed that the range of any loop is greater than the number of processors, thus compiling for load balancing is the task of selecting one or more iterators which are partitioned into groups and scheduled across the processors. In effect each processor performs a sub-set of some of the loop iterations. This sub-dividing of the iteration space has been referred to as tiling [7]. A similar process has been called loop elimination by [2] when applied to distributed memory multiprocessors.

Previous work on program restructuring has been mainly focused on revealing program parallelism and exploiting a machine's memory hierarchy efficiently [10]. In this paper it is assumed that the program is already in a form where all parallelism has been revealed, and it is now the task of the compiler to transform the program so as to minimise load imbalance. We believe that this is the first paper to use unimodular program transformations for this purpose.

The contributions of this paper are as follows:

- Load Balancing can be completely automated for a sub-class of programs.
- Load Balancing of array computation is defined as an optimisation problem where perfect load balance is described as an invariant condition.
- A necessary and sufficient condition for perfect load balance of parallel nested affine loops is derived.
- An efficient polynomial algorithm to transform such a particular iterator in a loop nest into a load balanced form is described.
- An algorithm to simultaneously transform several iterators into load balanced form is presented.
- Reordering of load balanced parallel affine loops is shown to be always possible.
- An algorithm is presented whereby all load balanced iterators within a loop nest may be moved outermost.

In this paper we focus on identifying the iterators to be partitioned, transforming them into a suitable form and then reordering the iterators such that they are outermost. Moving the iterators to be partitioned as far out of the loop nest as possible minimises the number of task spawns required.

To motivate the rest of this paper consider the example in figure 1. Here we assume that the number of processors $p = 10$. If the i loop is partitioned and statically scheduled across the processors such that the first processor receives the first 10 iterations, the second processor the

```
DOALL i = 1,100
  DOALL j = 1,100
    DOALL k =1,i
      a[i,j] += b[i,k]+c[k,j]
```

Figure 1: A nested DOALL loop

next 10 etc., then the first processor will perform 5,500 iterations, the last processor will perform 95,500, with the average being 50,500. If however the j loop were chosen all processors would perform 50,500 iterations. If it is assumed that the time to execute such a program is dominated by the processor performing the most iterations, then clearly partitioning with respect to j is preferable. If i were chosen to be partitioned then each processor would have to just spawn one task, whilst choosing j would require 100 spawns. Ideally j should be chosen to be partitioned, and placed outermost in the loop nest. This idea is the basis for this paper. We make the following observation: **The iterator that neither makes reference to any other iterator in its loop bounds, nor is referenced by any other, may be partitioned to give perfect load balance**. Perfect load balance occurs when each iteration of a particular iterator involves exactly the same amount of computation. Later sections formalise this idea and provide mechanisms to transform loops accordingly.

This paper is divided into 7 sections. The second section introduces the notation used throughout this paper and formally describes load balancing for parallel affine loops . The third section describes the unimodular transformation used to transform the loops into load balanced form. An existence condition is derived which forms the basis of the transformation algorithm. The fourth section extends the ideas of section 3 to the case when multiple loops are to be load balanced. The fifth section shows that a load balanced affine loop may always be moved outermost and defines the necessary transformation. Section 6 reviews related work and section 7 concludes this paper.

2 The Problem

2.1 Representation

In this paper we restrict our attention to parallel nested loops. The loop bounds and array references are restricted to the affine form described in figure 2. The number of loops is m, L_k and U_k are integer linear combinations of the iterators $J = [j_1, j_2, \ldots, j_{k-1}]^T, k \leq m$ and H is computation whose array occurrences make affine references to the enclosing iterators . It is assumed the amount of work in H is invariant of the iteration space. In practice this means that H contains no conditional evaluation. It follows that for perfect load balance, it is sufficient that each processor receives the same number of computation points. More general forms of H are considered in [6]. The lower and upper bounds of all loops described by figure 2 may

DOALL $j_1 = l_1, u_1$
DOALL $j_2 = L_2(j_1) + l_2, U_2(j_1) + u_2$
$\vdots$
DOALL $j_m = L_m(j_1, \ldots, j_{m-1}) + l_m,$
$U_m(j_1, \ldots, j_{m-1}) + u_m$
$H(j_1, \ldots, j_m)$

Figure 2: Parallel Nested Affine Loops

be expressed in matrix form.

$$LJ \geq l \tag{1}$$

$$UJ \leq u \tag{2}$$

L and U are both $(m \times m)$ lower unit triangular integer matrices, l and u are $(m \times 1)$ integer vectors and $J = [j_1, j_2, \ldots, j_m]^T$.

Equations 1 and 2 can be reformed thus

$$\begin{bmatrix} -L \\ U \end{bmatrix} J^m \leq \begin{bmatrix} -l \\ u \end{bmatrix} \tag{3}$$

which is in the general form of a polytope:

$$AJ^m \leq b \tag{4}$$

Within the polytope $AJ^m \leq b$ there exists a lattice of computation points $Latt(A.b)$ with integer co-ordinates, often referred to as the **iteration space**. Thus the points that J^m ranges over are the integer lattice points enclosed by the above polytope [8]. Due to the restriction on H, the amount of work associated with each point is uniform throughout the lattice. A load balanced mapping is one where the number of computation points assigned to each processor is the same.

Let q be any point in $Latt(A.b)$ and $|q|_x$ be the number of such points assigned to a processor x. As the points in this lattice may all be evaluated independently, we can state load balancing in this case to be find a mapping, π, where :

$$\pi : q \mapsto x \forall q \in Latt(A^m.b^m), x \in 1, \ldots, p \tag{5}$$

such that

$$|q|_x = |q|_y, \forall x, y \in 1, \ldots, p \tag{6}$$

In general this is not achievable and must be expressed as an optimisation problem where 6 is replaced by

$$Minimise(\max_x(|q|_x) - \frac{\sum_{x=1}^{p} |q|_x}{p}) \tag{7}$$

This paper determines the form of loops and the necessary transformations such that 5 and 6 may be satisfied.

2.2 Perfect Load Balance as Invariance

We seek a method of partitioning the lattice into p subsections such that the number of points scheduled to each is equal. Such a scheme provides perfect load balance. If we consider just orthogonal partitions of the polytope (partitions that are perpendicular to an iterator axis), then we seek the parallel iterator that may possess this property.

Each iterator $j \in J^m$ has a lattice, $Latt(A^{m-1}.b^{m-1})$, of computation points associated with it. We need to find a parallel iterator, $j_b \in J^m$, such that the number of points in its associated lattice is invariant of such an iterator.

We first formally present the invariance condition necessary for load balancing in terms of the loop structure and provide an example to illustrate this.

Definition 1 *Let e_b^T be the bth row of the identity matrix, then we define the* **invariance condition** *to be:*

$$e_b^T L = e_b^T U = e_b^T \tag{8}$$

$$L e_b = U e_b = e_b \tag{9}$$

The significance of the invariance condition is that the iterator j_b satisfying 8 and 9 is invariant if it does not make reference to other iterators nor is it referenced by any other iterator. In general both L and Uwill be of the form

$$\left[\begin{array}{c|c|c} L_f & 0 & 0 \\ \hline y_L & 1 & 0 \\ \hline A_L & x_L & L_g \end{array}\right] \left[\begin{array}{c|c|c} U_f & 0 & 0 \\ \hline y_U & 1 & 0 \\ \hline A_U & x_U & U_g \end{array}\right] \tag{10}$$

where L_f and U_f are $(b-1)\times(b-1)$ lower unit diagonal triangular matrices, L_g and U_g are $(m-b)\times(m-b)$ lower unit diagonal triangular matrices, y_L and y_U are $1\times(b-1)$ vectors, x_L and x_U are $(m-b)\times 1$ vectors and A_L and A_U are arbitrary integer $(m-b)\times(b-1)$ matrices.

As stated previously a candidate iterator for partitioning is one which does not refer to the bounds of any other loop. Therefore for an iterator $j_b \in J^m$ to satisfy 8 and 9 we require the following:

$$y_L = y_U = 0 \tag{11}$$

and

$$x_L = x_U = 0 \tag{12}$$

Therefore it is necessary to search each row in L, U to see if these conditions are satisfied. In general for a given set of loops this will not be true.

To illustrate these points, consider the following loop.

```
DOALL i = 1,n
  DOALL j = 1, n
    DOALL k = 2*i, 3*i
      a[i,k] += b[i,i]-c[k-j]
```

Figure 3: Triple Loop

The range of each of the iterators is represented by two matrix inequalities where each row corresponds to a unique iterator, and each matrix corresponds to the lower and upper bounds of the loop respectively :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \tag{13}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \leq \begin{bmatrix} n \\ n \\ 0 \end{bmatrix} \tag{14}$$

Here conditions 11 and 12 hold only for the second loop j.

However it is possible that none of the iterators have this form but may be transformed (with corresponding adjustments to the array occurrences) to a load balanced form. This is the subject of the next section.

3 Transformations

Legal transformations include any reordering of the computation that maintains the data dependency of the original program. By restricting this reordering to **DOALL** loops which contain no cross-iteration dependencies, all data dependencies are preserved. Additionally after transformation the loop should be in the structure defined by 1 and 2. More general forms of loop structure are studied in [6].

3.1 Change of Basis

Given an L, U and a form L^b, U^b, where the iterator j_b is in a load balanced form satisfying 11 and 12, find a transformation, π, such that:

$$\pi : L \mapsto L^b \tag{15}$$

$$\pi : U \mapsto U^b \tag{16}$$

We look at a restricted set of unimodular transformations [1] which satisfy 15 and 16 by post multiplication by a unit lower triangular matrix T, which changes the basis of $J \mapsto TJ_b$. The system of inequalities defined by 1 and 2 remains unchanged by this transformation.

$$L(TT^{-1})J \geq l \tag{17}$$

$$U(TT^{-1})J \leq u \tag{18}$$

$$(LT)(T^{-1}J) \geq l \tag{19}$$

$$(UT)(T^{-1}J) \leq u \tag{20}$$

If in addition an integer matrix T exists such that $LT = L^b$, $UT = U^b$, with J_b as the new iterators, then 19 and 20 may be written:

$$L^b J_b \geq l \tag{21}$$

$$U^b J_b \leq u \tag{22}$$

3.2 Existence Condition

In this section the necessary and sufficient conditions for the existence of a unimodular unit lower triangular matrix T is addressed.

Necessity: Assume there is a T such that $LT = L^b, UT = U^b$. The form of L, U, T is as follows:

$$\left[\begin{array}{c|c|c} L_f & 0 & 0 \\ \hline y_L & 1 & 0 \\ \hline A_L & x_L & L_g \end{array}\right] \left[\begin{array}{c|c|c} U_f & 0 & 0 \\ \hline y_U & 1 & 0 \\ \hline A_U & x_U & U_g \end{array}\right] \left[\begin{array}{c|c|c} T_f & 0 & 0 \\ \hline y_T & 1 & 0 \\ \hline A_T & x_T & T_g \end{array}\right] \tag{23}$$

To satisfy the invariance condition, we require L^b and U^b to be of the following form:

$$\left[\begin{array}{c|c|c} L_f^b & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline A_L^b & 0 & L_g^b \end{array}\right] \left[\begin{array}{c|c|c} U_f^b & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline A_U^b & 0 & U_g^b \end{array}\right] \tag{24}$$

Condition 11 implies that:

$$y_L T_f + y_T = 0 \tag{25}$$

$$y_U T_f + y_T = 0 \tag{26}$$

Therefore

$$(y_U - y_L) T_f = 0 \tag{27}$$

As $T_f \neq 0$ then

$$y_U = y_L \tag{28}$$

This is the first condition for existence of T. Condition 12 implies that:

$$L_g x_T = -x_L \tag{29}$$

$$U_g x_T = -x_U \tag{30}$$

Thus the solutions for x_T given by 29 and 30 must be consistent. Equations 29 and 30 may be written:

$$\left(\begin{bmatrix} 1 & 0 \\ x_L & L_g \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ x_U & U_g \end{bmatrix} \right) \begin{bmatrix} 1 \\ x_T \end{bmatrix} = 0 \tag{31}$$

$$\begin{bmatrix} 1 & 0 \\ x_L & L_g \end{bmatrix} \begin{bmatrix} 1 \\ x_T \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{32}$$

Together these form the second and third condition for existence of T. Clearly 28,31 and 32 must hold if a transformation is to be determined. This establishes the necessity.

Sufficiency: Assume 28 holds, and there is x_T satisfying 31 and 32 then:

$$T = \left[\begin{array}{c|c|c} I_{b-1} & 0 & 0 \\ \hline -y_L & 1 & 0 \\ \hline 0 & x_T & I_{m-b} \end{array}\right] \tag{33}$$

is the desired unimodular transformation corresponding to iterator j_b.□

3.3 Algorithm 1

The following algorithm determines whether a transformation T exists and if it does finds it. In addition, the relationship between the new iteration space and the old one is determined, so that the relevant array occurrences may be altered accordingly.

for each $j_b \in J^m$

1. Check $y_L = y_U$. If not terminate.
2. Choose an arbitrary lower unit diagonal T_f e.g. unity.
3. Calculate $y_T = -y_L T_f$
4. Solve $\left(\begin{bmatrix} 1 & 0 \\ x_L & L_g \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ x_U & U_g \end{bmatrix}\right)\begin{bmatrix} 1 \\ x_T \end{bmatrix} = 0$
5. Check if consistent. If not terminate.
6. Solve$\begin{bmatrix} 1 & 0 \\ x_L & L_g \end{bmatrix}\begin{bmatrix} 1 \\ x_T \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
7. Check steps 4 and 6 are consistent. If not terminate.
8. Choose an arbitrary lower unit diagonal T_g e.g. unity.
9. Choose an arbitrary matrix A_T e.g. the null matrix.
10. Construct T
11. Calculate $L^b = LT, U^b = UT$
12. For each $j \in J$ in each array occurrence substitute $J = TJ_b$

The complexity of this algorithm is dominated by steps 4 and 6. Thus the upper bound complexity is $O(m^2)$. If this process is repeated for all the iterators then the upper bound complexity is $O(m^3)$. To illustrate this algorithm, consider the following program:

```
DOALL i = 1,n
  DOALL j = n+i-1,2*n+i+1
    DOALL k in 1+2*i+2*j , n+3*i+2*j
        a[i,j] = Min(c[i,k]*d[i-j,k])
```

Figure 4: Nested DOALL Loop

In its present form it does not satisfy the invariance condition. The upper and lower bounds for each of the iterators are as follows:

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -2 & -2 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \geq \begin{bmatrix} 1 \\ n-1 \\ 1 \end{bmatrix} \tag{34}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -3 & -2 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \leq \begin{bmatrix} n \\ 2n+1 \\ n \end{bmatrix} \tag{35}$$

By applying algorithm 1, it is possible to determine if this program may be transformed into an invariant form. Test the first iterator i:

1. Not applicable.
2. Not applicable.
3. Not applicable.
4. Find x_T where
$$\left(\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -3 & -2 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -2 & -2 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$
There is no x_T.
5. Not consistent. $1 \neq 0$. Terminate.

   Thus loop i is rejected.

Now try the second iterator j i.e. $b = 2$.

1. $y_U = -1$ and $y_L = -1$ therefore $y_U = y_L$ is satisfied.
2. $T_f = 1$
3. $y_T = -(-1)1 = 1$
4. Find x_T where
$$\left(\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 \\ x \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$
This is true for all x.

5. No contradiction

6. Find x_T where
   $$\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$
   This implies x=2. No inconsistency

7. 5 and 6 give consistent results for x

8. $T_g = 1$

9. $A_T = 0$

10. $T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix}$

11. $L^b =$
    $$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -2 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix}$$
    $U^b =$
    $$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -3 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{bmatrix}$$

12. $\begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ k' \end{bmatrix} = \begin{bmatrix} i' \\ j' + i' \\ k' + 2j' \end{bmatrix}$

If the same procedure is applied to iterator k ,$b = 3$, it is seen that it fails on the first step. $y_L = [-2, -2], y_U = [-3, -2], y_U \neq y_L$. So loop k is not a candidate for partitioning for load balancing.

As only loop j is in invariant form we rewrite the program as:

```
DOALL i = 1,n
 DOALL j = n-1,2*n+1
  DOALL k in 1+4*i , n+5*i
    a[i,j+i] = Min(c[i,k+2*j]*d[-j,k+2*j])
```

```
Figure 5: Transformed DOALL Loop
```

Note that the constant terms in the loop are unaffected by the transformation. The array references are adjusted in accordance with step 12. No other loop depends on the new j loop, therefore parallelising and partitioning with respect to this loop will give a load balanced implementation.

4 Multiple Iterators

If each of the m parallel loops are successful candidates for load balancing, then there are potentially 2^m-1 permutations which may be used to partition the iteration space. To determine whether a combination of iterators can partition the iteration space in a load balanced manner, a modification of algorithm 1 is required. In this section we propose a new algorithm to construct a new transformation $T_s^{'}$ which has the combined effect of transforming each load balanced loop $j = i_1, \ldots, i_s$ where $s \leq m$ is the number of invariant loops and $i_1 < \cdots < i_s$ are the values of loops to be load balanced.

Let T_k be the individual transformation on a particular iterator k as given by algorithm 1. The first theorem below shows that given the transformation $T_{\ell-1}^{'}$ that makes invariant all loops $i_1 < i_2 < .. < i_{\ell-1}$ and the transformations T_{i_ℓ} determined by Algorithm 1 for iterator i_ℓ alone, $T_\ell^{'}$ can be constructed to include the new iterator i_ℓ . Two lemmas are required to prove this theorem. The first is a technical condition to aid the proof, the second ensures that the form of the transformation is legal.

$T_s^{'}$ is defined by a corollary to the main theorem and provides the basis for a simple algorithm to simultaneously make invariant all load balanced loops.

Lemma 1 *Given $T_{\ell-1}^{'}$ such that $LT_{\ell-1}^{'}, UT_{\ell-1}^{'}$ are jointly invariant for $j \in i_1, \ldots, i_{\ell-1}$ i.e.*

$$LT_{\ell-1}^{'}e_{i_k} = UT_{\ell-1}^{'}e_{i_k} = e_{i_k} \forall k \in 1, \ldots, \ell-1 \tag{36}$$

$$e_{i_k}^T LT_{\ell-1}^{'} = e_{i_k}^T UT_{\ell-1}^{'} = e_{i_k}^T \forall k \in 1, \ldots, \ell-1 \tag{37}$$

Then $T_{\ell-1}^{'}$ can be chosen so that:

$$T_{\ell-1}^{'}e_{i_k} = e_{i_k} \forall k \geq \ell \tag{38}$$

Proof of Lemma 1 *This follows immediately by observing that invariance conditions 36,37 impose constraints only on elements in the $m \times (\ell-1)$ sub-matrix of $T_{\ell-1}^{'}$ and hence the $(m-\ell+1) \times (m-\ell+1)$ right hand corner sub-matrix of $T_{\ell-1}^{'}$ can be chosen as identity matrix without violating conditions 36,37.* □

Theorem 1 *Define $T_\ell^{'}$ by:*

$$T_\ell^{'} = T_{\ell-1}^{'} - e_{i_\ell}e_{i_\ell}^T LT_{\ell-1}^{'} + T_{i_\ell}e_{i_\ell}e_{i_\ell}^T \tag{39}$$

Then $T_\ell^{'}$ is the transformation that satisfies the invariance condition for all load balanced loops $j \in i_1, \ldots, i_\ell$ i.e.

$$LT_\ell^{'}e_{i_k} = UT_\ell^{'}e_{i_k} = e_{i_k} \forall k \in 1, \ldots, \ell \tag{40}$$

$$e_{i_k}^T LT_\ell^{'} = e_{i_k}^T UT_\ell^{'} = e_{i_k}^T \forall k \in 1, \ldots, \ell \tag{41}$$

The proof needs the following preliminary Lemma.

Lemma 2 *$T_\ell^{'}$ given in 39 is unit lower triangular.*

Only an outline of the proof is presented

Proof of Lemma 2 *Substitute for T'_ℓ from 39 and use 36 and 37 to show*

$$e_i^T T'_\ell e_j = 0 \; \forall i < j \tag{42}$$

and

$$e_i^T T'_\ell e_i = 1 \tag{43}$$

Proof of Theorem 1 *It is sufficient to show 40 and 41 for LT'_ℓ.*

step1. Show 40 and 41 are true $\forall k \in 1, \ldots, \ell - 1$ [Proof Omitted]

step2. Show 40 and 41 are true for $k = \ell$

$$LT'_\ell e_{i_\ell} = L(T'_{\ell-1} - e_{i_\ell} e_{i_\ell}^T LT'_{\ell-1} + T_{i_\ell} e_{i_\ell} e_{i_\ell}^T) e_{i_\ell} \tag{44}$$

By Lemma 1 $T'_{\ell-1} e_{i_\ell} = e_{i_\ell}$

$$LT'_\ell e_{i_\ell} = Le_{i_\ell} - Le_{i_\ell} e_{i_\ell}^T Le_{i_\ell} + LT_{i_\ell} e_{i_\ell} \tag{45}$$

Observe that ${e_{i_\ell}}^T Le_{i_\ell} = 1$ and $LT_{i_\ell} e_{i_\ell} = e_{i_\ell}$ by invariance of i_ℓ for LT_{i_ℓ}. Hence:

$$LT'_\ell e_{i_\ell} = Le_{i_\ell} - Le_{i_\ell} + e_{i_\ell} = e_{i_\ell} \tag{46}$$

Similarly:

$$e_{i_\ell}^T LT'_\ell = e_{i_\ell}^T LT'_{\ell-1} - e_{i_\ell}^T Le_{i_\ell} e_{i_\ell}^T LT'_{\ell-1} + e_{i_\ell}^T LT_{i_\ell} e_{i_\ell} e_{i_\ell}^T \tag{47}$$

Noting that $e_{i_\ell}^T LT_{i_\ell} = e_{i_\ell}^T$ and simplifying the expression $\Rightarrow$

$$e_{i_\ell}^T LT'_\ell = e_{i_\ell}^T e_{i_\ell} e_{i_\ell}^T = e_{i_\ell}^T \tag{48}$$

this is the invariance condition and thus T'_ℓ is the required transformation. □

It has been shown that T'_ℓ is the transformation that load balances ℓ iterators provided $T'_{\ell-1}$ is given. The following corollary constructs the transformation T'_s that load balances all s iterators.

Corollary 1 $T'_s = T'_{s-1} - e_{i_s} e_{i_s}^T LT'_{s-1} + T_{i_s} e_{i_s} e_{i_s}^T$ *is the transformation for joint invariance of $j \in i_1, \ldots, i_s$*

Proof of Corollary 1 *set $T'_1 = T_{i_1}$ and recursively apply theorem 1 for $k = 2, \ldots, s$*□

4.1 Algorithm 2

It is now possible to give a simple algorithm that uses the result of theorem 1 to construct a transformation that transforms all load balanceable loops into invariant form.

1. Apply Algorithm 1 to give the invariant iterators $j \in i_1, \ldots, i_s$ and the canonical transformations $T_{i_1}, \ldots, T_{i_s}$ If $s \leq 1$ Stop.

2. Set $T'_1 = T_{i_1}$

3. For $k \in 2, \ldots, s$

$$T'_k = T'_{k-1} - e_{i_k} e_{i_k}^T L T'_{k-1} + T_{i_k} e_{i_k} e_{i_k}^T \tag{49}$$

Complexity: The extra cost of computing the transformation T'_s, step 3 of Algorithm 2 has an upper bound less than $O(s.{i_s}^2) \leq O(m^3)$. As Algorithm 1 has an upper bound complexity of $O(m^3)$ this new algorithm does not alter the overall complexity of the scheme.

To illustrate this algorithm, consider the following slightly contrived example:

```
DOALL i = 1,n
 DOALL j = -i+1,n-i
  DOALL k in -1-i-j ,-j+3
   DOALL l in 1-i-2*j-k , 2*n-2*j-k-i
    DOALL m in 2*j+6*i+k-l-1 , i-k-l+n
     a[i,j] +=2*(b[i,k] * c[k,l] + d[m,m-1])/b[j,j]
     e[i,i+j,i] += f[i,j,k,l,m]* g[j,k,l,m]
```

Figure 6: Five Nested Loop

This example has been chosen to show that load balancing of parallel affine loops is non-trivial in more complex cases, such as when more than one iterator is a candidate for load balancing.

This loop nest has the following upper and lower bound matrices on J^5:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 1 & 0 \\ -6 & -2 & -1 & 1 & 1 \end{bmatrix} \quad l = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \tag{50}$$

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 1 & 0 \\ -1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad u = \begin{bmatrix} n \\ n \\ 3 \\ 2n \\ n \end{bmatrix} \tag{51}$$

On applying algorithm 1, j_2 and j_4 are the only candidates for load balancing, s=2. The corresponding transformation matrices for both iterators are as follows:

$$T_{j_2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 1 \end{bmatrix} \tag{52}$$

$$T_{j_4} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -1 & -2 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{53}$$

To find the transformation T'_s, $s = 2$ for joint invariance of j_2, j_4, apply step 2 of Algorithm 2.

$$T'_1 = T_{j_2} \tag{54}$$

$$T'_2 = T'_1 - e_{j_4} e^T_{j_4} L T'_1 + T_{j_4} e_{j_4} e^T_{j_4} \tag{55}$$

This gives

$$T'_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 \\ 0 & 2 & 0 & -1 & 1 \end{bmatrix} \tag{56}$$

Applying T'_2 to L and U gives:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -3 & 0 & -2 & 0 & 1 \end{bmatrix} l = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \tag{57}$$

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} u = \begin{bmatrix} n \\ n \\ 3 \\ 2n \\ n \end{bmatrix} \tag{58}$$

Note that rows and columns 2 and 4 are in invariant form for both upper and lower bounds. The array occurrences must be expressed with respect to the new iteration basis

$$\begin{bmatrix} i \\ j \\ k \\ l \\ m \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 \\ 0 & 2 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ k' \\ l' \\ m' \end{bmatrix} \tag{59}$$

$$\begin{bmatrix} i \\ j \\ k \\ l \\ m \end{bmatrix} = \begin{bmatrix} i' \\ j' - i' \\ k' - j' \\ l' - k' - j' + i' \\ m' - l' + 2j' \end{bmatrix} \tag{60}$$

This give the following transformed program:

```
DOALL i = 1,n
 DOALL j = 1,n
  DOALL k in -1 ,i+3
    DOALL l in 1 , 2*n
      DOALL m in 3*i+2*k -1 , n
        a[i,j-i] +=2*(b[i,k-j]*c[k-j,l-k-j+i]
                   +d[m-l+2*j,m-l+2*j-1])/b[j-i,j-i]
        e[i,j,i] +=f[i,j-i,k-j,l-k-j+i,m-l+2*j]*
                   g[j-i,k-j,l-k-j+i,m-l+2*j]
```

Figure 7: Transformed Invariant DOALL Loop

Although loops j and l are in load balanced form, they are not outermost. The next section describes a method whereby the loops may be always re-ordered so that they are outermost.

5 Reordering Iterators

Having determined which iterator(s) are to be used to partition the computation lattice, it is desirable to have these iterators as far out as possible in the loop structure of the translated form to reduce the number of spawned tasks. It can be shown that all perfectly load balanced parallel affine iterators can be moved to the outermost scope and remain affine.

In this section we show that load balanced iterators can be moved to the outermost nest by a sequence of unimodular transformations, whilst preserving the affine structure of the loop.

Theorem 2 shows that one invariant iterator can be moved one nest level up by a unimodular transformation. By extending this result it is possible to move multiple iterators to the outermost nest by a succession of these unimodular transformations. To prove this theorem some preliminary definitions and lemmas are first required.

Let $E_{i,k}$ be the permutation of identity with row i and k interchanged. $E_{i,k}$ is unimodular, and $E_{i,k}^{-1} = E_{i,k}$. Interchange of iterator j_i with j_k can be represented as:

$$J' = E_{i,k} J \; j_i, j_k \in J \tag{61}$$

Let L, U be in load balanced form with iterator $j_i \in J$ invariant. Define:

$$L' = E_{i-1,i} L E_{i-1,i} \tag{62}$$

$$U' = E_{i-1,i} U E_{i-1,i} \tag{63}$$

$$J' = E_{i-1,i} J \tag{64}$$

$$l' = E_{i-1,i} l \tag{65}$$

$$u' = E_{i-1,i} u \tag{66}$$

Lemma 3 *L', U' are unit lower triangular.*

The importance of Lemma 3 is in establishing that the unimodular transformation described in 62 and 63 preserve the affine structure of the loop.

Proof of Lemma 3 *L, U are in load balanced form, and j_i is the invariant iterator. Thus*

$$e_i^T L e_{i-1} = e_i^T U e_{i-1} = 0 \tag{67}$$

As loop interchange is restricted to neighbouring iterator of j_i, in this case j_{i-1}, the only possible non-zeros in the upper triangular part of $L^{'}$ and $U^{'}$ are the $(i-1, i)$ elements. From 62 and 63 it is easily seen that

$$e_{i-1}^T L^{'} e_i = e_i^T L e_{i-1} \tag{68}$$
$$e_{i-1}^T U^{'} e_i = e_i^T U e_{i-1} = 0 \tag{69}$$

Combine with 67 and observe that the effect of transformations in 62 and 63 on the diagonal elements of L, U are to interchange the (i, i) with $(i-1, i-1)$ elements both equal to 1. This establishes that $L^{'}, U^{'}$ are unit lower triangular. □

Lemma 4 *The iteration spaces represented by*

$$LJ \geq l \tag{70}$$

$$UJ \leq u \tag{71}$$

and

$$L^{'}J^{'} \geq l^{'} \tag{72}$$

$$U^{'}J^{'} \leq u^{'} \tag{73}$$

are equivalent.

Proof of Lemma 4 *By Lemma 3, 72 and 73 represent a legal affine loop. It remains to show the equivalence of system of inequalities 70,71 and 72,73. To this end, we can observe that inserting $E_{i-1,i}E_{i-1,i}$ =Identity in 70 and71 preserves the system of inequalities.*

$$LE_{i-1,i}E_{i-1,i}J \geq l \tag{74}$$

$$UE_{i-1,i}E_{i-1,i}J \leq u \tag{75}$$

Now we substitute from 64:

$$LE_{i-1,i}J^{'} \geq l \tag{76}$$

$$UE_{i-1,i}J^{'} \leq u \tag{77}$$

Now multiply both sides of inequalities in 76 and 77 by $E_{i-1,i}$. This amounts to reordering the inequalities, thus preserves the iteration space. Substitute from 62 to 66 $\Rightarrow$:

$$L^{'}J^{'} \geq l^{'} \tag{78}$$

$$U^{'}J^{'} \leq u^{'} \square \tag{79}$$

Theorem 2 *Let $L^{'}$,$U^{'}$ be defined as in 62 and 63. Then $j_{i-1} \in J^{'}$ is an invariant iterator for $L^{'}, U^{'}$.i.e.*

$$L^{'}e_{i-1} = U^{'}e_{i-1} = e_{i-1} \tag{80}$$

and

$$e_{i-1}^{T}L^{'} = e_{i-1}^{T}U^{'} = e_{i-1}^{T} \tag{81}$$

Proof of Theorem 2 *It suffices to show 80 and 81 for $L^{'}$. By assumption, j_i is an invariant iterator for L, U. Thus*

$$Le_i = e_i \tag{82}$$

$$e_i^T L = e_i^T \tag{83}$$

Observe the identities:

$$E_{i-1,i}e_{i-1} = e_i \tag{84}$$

$$e_{i-1}^{T}E_{i-1,i} = e_i^T \tag{85}$$

Substitute 84 in 82:

$$LE_{i-1,i}e_{i-1} = E_{i-1,i}e_{i-1} \tag{86}$$

Multiply both sides of 86 by $E_{i-1,i}$ and substitute from 62:

$$L^{'}e_{i-1} = e_{i-1} \tag{87}$$

Similarly, substitute 85 in 83:

$$e_{i-1}^{T}E_{i-1,i}L = e_{i-1}^{T}E_{i-1,i} \tag{88}$$

Multiply both sides of 88 by $E_{i-1,i}$ and substitute from 62:

$$e_{i-1}^{T}L^{'} = e_{i-1}^{T} \tag{89}$$

which is the invariant condition and thus j_{i-1} is an invariant iterator. □

Theorem 2 shows that the new transformed iterators have any one load balanced loop one loop nest further out than before.

Observation 1 *Similarly, it can be shown that any load balanced iterator j_i can be moved one loop nest further in by applying permutation transformation $J^{'} = E_{i,i+1}J$. Thus two neighbouring iterators that are both load balanced will remain so upon interchange.*

Given the set of iterators J_B where iterators $j = i_1, i_2, \ldots, i_s$ are the values of the iterators in load balanced form for

$$L^B J_B \geq l \tag{90}$$

$$U^B J_B \leq u \tag{91}$$

We propose to find a unimodular transformation E such that:

$$J_o = EJ_B \tag{92}$$

and J_o is the iteration vector with the first s iterators load balanced.

To this end let E^i be the transformation that moves a particular iterator j_i to the outermost scope. It is defined thus:

$$E^i = E_{1,2} \times E_{2,3} \times \cdots \times E_{i-1,i} \tag{93}$$

It should be noted that in general

$$E^i \neq E_{1,i} \tag{94}$$

E is now defined as:

$$E = E^{i_s} \times E^{i_{s-1}} \times \cdots \times E^{i_1} \tag{95}$$

E is unimodular as it is the product of unimodular transformations.

Let

$$L^o = EL^B E^{-1} \tag{96}$$

$$U^o = EU^B E^{-1} \tag{97}$$

$$l^o = El \tag{98}$$

$$u^o = Eu \tag{99}$$

L^o, U^o, l^o, u^o are in the form described by figure 1. This is shown by repeated application of lemma 3. The set of inequalities given by:

$$L^o J_o \geq l^o \tag{100}$$

$$U^o J_o \leq u^o \tag{101}$$

are equivalent to 90 and 91. This can be shown by observing that:

$$EL^B(E^{-1}E)J_B \geq El \tag{102}$$

$$EU^B(E^{-1}E)J_B \leq Eu \tag{103}$$

$$(EL^B E^{-1})(EJ_B) \geq El \tag{104}$$

$$(EU^B E^{-1})(EJ_B) \leq Eu \tag{105}$$

Substituting from 96 to 99 gives the required form of 100 and 101.

Finally by the repeated application of theorem 2, and using Observation 1 it can be shown that the first s iterators of J_o are load balanced for 100 and 101.

This analysis gives the following algorithm to reorder the iterators.

5.1 Algorithm 3

1. For $\ell \in 1, \ldots, s$
2. For $k \in i_\ell$ to 2 step -1
3. Interchange rows k and $k-1$ of U, L, l, u
4. Interchange columns k and $k-1$ of U, L, and rows k , $k-1$ of J
5. End For
6. End For

To illustrate this algorithm consider the matrix form of the program given in figure 4 after transforming to invariant form.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \geq \begin{bmatrix} 1 \\ n-1 \\ 1 \end{bmatrix} \tag{106}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \leq \begin{bmatrix} n \\ 2n+1 \\ n \end{bmatrix} \tag{107}$$

There is only one loop j to move out, i.e. $s = 1, i = 2, j_i = j_2 = j$. On interchanging rows we have

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \geq \begin{bmatrix} n-1 \\ 1 \\ 1 \end{bmatrix} \tag{108}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ -5 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \leq \begin{bmatrix} 2n+1 \\ n \\ n \end{bmatrix} \tag{109}$$

Interchanging columns of U, L and rows of J gives

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{bmatrix} \begin{bmatrix} j \\ i \\ k \end{bmatrix} \geq \begin{bmatrix} n-1 \\ 1 \\ 1 \end{bmatrix} \tag{110}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -5 & 1 \end{bmatrix} \begin{bmatrix} j \\ i \\ k \end{bmatrix} \leq \begin{bmatrix} 2n+1 \\ n \\ n \end{bmatrix} \tag{111}$$

which finally gives the following program:

```
DOALL j = n-1,2*n+1
 DOALL i = 1,n
  DOALL k in 1+4*i , n+5*i
   a[i,j+i] = Min(c[i,k+2*j]*d[-j,k+2*j])
```

Figure 8: Load Balanced Loop Outermost

This program has a load balanced loop which is outermost. Each iteration of j will have exactly the same amount of work to perform and all that is now required is for the n+3 iterations to be divided amongst the processors.

6 Review

The properties of unimodular transformations for doubly nested loops with constant bounds have been covered by [1]. It has also shown that existing transformations such as the wavefront method [4] can be described in terms of unimodular transformations [3].

Program transformations have mainly been focused on revealing program parallelism, however in [5] unimodular transformations are used as a mechanism to describe distribution of loops across distributed processors. They study the effect of unimodular transformations by giving a measure of parallelism, load imbalance and volume of communication which are again restricted to the two-dimensional rectangular loop case. The unimodular transformations used within our paper are related to loop skewing [9] and loop interchange[10].

The polytope and related notation is based upon the work of [8]. The main concern of his thesis was the unification of the systolic framework based on uniform recurrences with data dependency analysis. Although the polytope notation was developed quite extensively, it was used chiefly to find a legal ordering vector within the polytope so as to maintain program data dependencies.

The work presented in this paper forms part of the general mapping of array computation to distributed memory architectures. In [6] load balancing for more general programs including serial loops and conditionals and arbitrary nesting is presented. Matrix transformations to minimise non-local access by alignment, data partitioning and data re-use are also presented where the interaction between load balancing and communication overhead is investigated.

7 Conclusion

This paper has addressed the use of unimodular transformations to load balance a particular sub-class of loop structures and to reorder them so as the appropriate iterators are outermost.

By viewing the iteration space of such loops as a polytope it has been possible to determine an invariance condition to ensure perfect load balancing.

The existence conditions for a load balancing transformation have been derived. If any one loop in a loop nest satisfies this condition an algorithms to perform the transformation in polynomial time has been presented.

If more than one load balanced loop is to be partitioned and scheduled over the processors, an algorithm has been derived which simultaneously transforms all of them without increasing the complexity of the scheme.

It is desirable to have partitioned loops outermost. The surprising result that this is always possible for load balanced loops has been shown. A simple unimodular transformation and hence an algorithm that moves such loops outermost has been given.

This paper has shown that load balancing, for a sub-class of programs, can be achieved by application of unimodular transformations. Further work will no doubt apply such transformations for other issues in compiling for multiprocessors.

Acknowledgement

The authors would like to thank Chris Kirkham for his many helpful suggestions during the preparation of this report.

References

[1] Banerjee U.,**Unimodular Transformations of Double Loops**, Proc. of 3rd Workshop on Programming Languages and Compilers for Parallel Computing, Irvine CA, August 1990.

[2] Callahan D. and Kennedy K.,**Compiling Programs for Distributed Memory Multiprocessors**, Journal of Supercomputing, Vol. 2 No. 2, pp 151-207, 1988.

[3] Dowling, M. **Optimal Code Parallelization using Unimodular Transformations**, Parallel Computing Vol. 16, pp 157-171, 1990.

[4] Lamport L., **The Parallel Execution of DO loops**, CACM Vol. 17 No. 2, Feb 1974.

[5] Kulkarni D., Kumar K.G., Basu A., and Paulraj A.,**Loop Partitioning for Distributed Memory Multiprocessors as Unimodular Transformations**, Proc. of ACM International Conference on Supercomputing, June, 1991.

[6] O'Boyle M.F.P., **Program and Data Transformations for Efficient Execution on Distributed Memory Architectures**, PhD thesis, Department of Computer Science, University of Manchester, January 1992.

[7] Ramanujam J. and Sadyappan P.,**Tiling of Iteration Spaces for Multicomputers**, Proc. of International Conference on Parallel Processing, Vol. 2, pp 179 -186, 1990.

[8] Ribas, H.B.,**Automatic Generation of Systolic Programs from Nested Loops**, Carnegie-Mellon Tech. Rep. CMU-CS-90-143, June 1990.

[9] Wolfe M., **Loop Skewing: The Wavefront Method Revisited**, International Journal of Parallel Programming, Vol. 15, No. 4 pp 279-294, August 1986.

[10] Wolfe M., **Massive Parallelism through Program Restructuring**, 3rd Symposium on the Frontiers of Massively Parallel Computation, pp 407-415, October 1990.