Minimizing Synchronization in Parallel Nested Loops

Reza Rafreh
Centre for Business, Information Technology and Enterprise
Waikato Institute of Technology
Hamilton, New Zealand
Reza.Rafeh@wintec.ac.nz

Mohammad Hossein Roosta
Department of Computer Engineering,
Islamic Azad University, Malayer Branch,
Malayer, Iran
roosta_nh@yahoo.com

ABSTRACT

Although, computer system architecture and the throughput enhances continuously, the need for high computational speed and power in many scientific applications grows every day. As a result, implementation of parallel applications has gained more attention. Since nested loops are the most time-consuming parts of most programs, we propose a method for scheduling uniform nested loops to processors based on the equation of a straight line which includes the maximum possible number of dependence vectors. Experimental results show that the proposed method imposes a lower communication between processors compared with similar methods.

Keywords: Uniform Nested loops, Scheduling, Chaining, Communication.

1. INTRODUCTION

Parallel processing is a form of computing in which many instructions are executed simultaneously. The process of parallelization in general consists of three steps as follows (Sinnen, 2007):

- Decomposing the application into tasks.
- Analysing the dependencies between the decomposed tasks.
- Scheduling tasks into the target parallel or distributed system.

Since the most time-consuming parts of scientific and engineering sequential programs are nested loops and recursive procedures, their optimization plays an important role in reducing the execution time of the program. For parallelizing nested loops, at first, the iterations should be broken down into tasks. In the second step, dependencies between tasks must be discovered. Dependencies prevent parallel execution of tasks efficiently. Generally, there are two types of dependencies:

- Data dependencies caused by data transfer between tasks. For example: \( X = Y \); \( Z = X \);  
- Control dependencies caused by the order of instructions that are logically related to each other. For example:

  \[
  \text{IF cond THEN s1 ELSE s2}
  \]

There are two types of nested loops based on dependencies:

- DOALL loops: Nested loops with no dependency.
- DOACROSS loops: Nested loops with dependencies which are divided into two categories:
  - Uniform: A loop in which the pattern of dependencies remains constant during its execution
  - Non-uniform: A loop in which the pattern of dependencies may have variations during execution.

In this paper we propose a novel approach to reduce the communication in uniform nested loops.

2. METHODOLOGY

Loop iteration space: Iteration space of any n-dimensional nested loop can be mapped into a finite discrete n-dimensional Cartesian space \( J = \{ (j_1, \ldots, j_n) \in \mathbb{N}^n | l_r \leq j_r \leq u_r, 1 \leq r \leq n \} \), in which each point represents an iteration of the nested loop. The body of each loop may include a set of instructions.

Loop dependencies: Statements \( S_1 \) in iteration \( J_1 \) is dependent to statement \( S_2 \) in iteration \( J_2 \) if and only the following conditions hold:

1. \( J_1 < J_2 \) or \( J_1 = J_2 \) and an execution path exists from \( J_1 \) to \( J_2 \)
2. both statements access a same memory location
3. at least one access modifies the memory content

Dependency vectors: In a n-dimensional nested loop existing dependency between two iterations \( J_1 \) and \( J_2 \) (if any) is represented by a dependence vector \( d \) of length \( n \):

\[
d(J_1, J_2) = (j_{21} - j_{11}, j_{22} - j_{12}, \ldots, j_{2n} - j_{1n})
\]

The size of dependence vector, which is always greater than zero, shows the number of iterations between two dependent iterations. If the size of all dependence vectors remains fixed during the execution of a loop, the loop is known as a uniform loop.
**Example 1:** The dependence vectors of the two dimensional nested loop in Figure 1 are: $d_1 = (1, 3), d_2 = (2, 2), d_3 = (4, 1)$ and are shown in Figure 2.

```plaintext
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        S2: A[i][j] = B[i - 2][j + 2] + 2;
        S3: A[i][j] = A[i - 4][j - 1];
    }
}
```

**Figure 1.** A two Dimensional nested loop with three dependence vectors

**Figure 2.** Dependence vectors of nested loops in Figure 1

The communication vector ($d_c$): Communication vector is a vector that incurs the largest amount of communication. In most cases the communication vector is the dependence vector with the smallest absolute coordinate values.

Our proposed algorithm which is referred as the Best Straight Line Scheduling (BSLS), uses the concept of chains which was introduced in Chain Pattern Scheduling (CPS) algorithm. The idea of BSLS is to minimize the communication cost between dependent iterations by assigning them to a single processor. In other words, when a point $J_s = (x_s, y_s)$ sends data to points in set $DP = \{ J_1, \ldots, J_m \}$ because of dependence vectors $d_1, \ldots, d_m$, by mapping $J_s$ and all points of set $DP$ to a single processor, the communication incurred by vectors $d_1, \ldots, d_m$ is eliminated. However it is impossible to map all such points to a processor. If we map the point $J_s$, with maximum number of possible points from $DP$ to a processor, the data locality will be increased. In the proposed approach, the points assigned to a processor are determined by a straight line which passes through the maximum number of points in $DP$. This line is referred to as the best straight line.

In this algorithm, for each pair of points in $DP$, the equation of the line passes through these two points is determined. Then, the number of other points lie on this line is calculated. The line on which the maximum number of points lies will be the best straight line. The points of $DP$ lie on the best straight line along with $J_s$ are assigned to a single processor.

Since the dependence vectors remain unchanged during the execution of the loop, the equation of the straight line can be used to chain all points in the iteration space. In a two-dimensional space, assuming the equation of the straight line be $y = mx + r$, the equation of chains are are: $C_i = \{ j \in J | j = m \times x + r_i, m \in R \}$ where $r_i$ is the offset of the $i$th chain with the best straight line.

**Example 2:** We consider a system with 4 processor and a two-dimensional uniform nested loop with the following dependence vectors: $d_1 = (1, 3), d_2 = (2, 2), d_3 = (4, 1)$ and the communication vector $d_c = d_3$. The following equations are candidates for being the equation of the best straight line:

1) $y = -x + 4 \quad \{ (1,3), (2,2) \}$
2) $y = -2/3 x + 8 \quad \{ (1,3), (4,1) \}$
3) $y = -1/2 x + 9 \quad \{ (2,2), (4,1) \}$

Each line passes two points of set $DP$, thus, each of them can only eliminate two dependence vectors. In Figure 3 the equation $y = -x + 4$ is being considered for the best straight line and chains are formed in parallel with this line. In this way, the communication cost between $d_1$ and $d_2$ is eliminated. Chains are assigned to processors in a round robin fashion shown in the figure in different colors.

**Figure 3.** Chaining of points in an index space of a two dimensional nested loop with four dependence vectors and four processors using the proposed algorithm

### 3. RESULTS

To evaluate the proposed approach in terms of the communication cost, we compare it with CSP. In this experiment, we examine the uniform nested loop given in Example 1 which includes four dependence vectors. The index space ranges from $10 \times 10$ to $1000 \times 1000$. Results have been depicted in Figures 4 and 5 with 4 and 5 processors, respectively. Since in this experiment each line passes two points (see Example 5), the worst case of the proposed algorithm happens. However, the proposed algorithm works better than CSP when number of processors is 4. Note that in diagrams, the total column shows all dependencies in the iteration space.

### 4. REFERENCES


Lim, A. W., Cheong, G. I., & Lam, M. S. (1999). An affine partitioning algorithm to maximize parallelism and minimize communication.


**Figure 4.** Experimental results, BSLS versus CPS, on four processors

**Figure 5.** Experimental results, BSLS versus CPS, on five processors