SysPar: A Software Package for Systolising and Parallelising Nested Loop Algorithms

Jingling Xue
School of Electrical and Electronic Engineering
Nanyang Technological University
Singapore 2263

Abstract
Systolic array design consists in the distribution of iterations to both space and time in such a way that the distribution specifies a systolic array. Parallelising compilation consists in the identification of loops whose different iterations may run simultaneously on different processors, either independently or in an overlapped fashion. Both use various matrix transformations such as loop interchange and reversal to detect and exploit parallelism in nested loop algorithms. This paper describes a software package called SysPar for systolising and parallelising nested loop algorithms by unifying these various transformations as non-singular (unimodular or non-unimodular) transformations.

1 Introduction
By combining the advantages of the approaches to systolic array design and parallelling compilation, progress has been made recently in detecting and exploiting parallelism in nested loop algorithms [2, 5, 8, 21, 23, 24]. This leads to an approach that combines the mathematical rigour in the matrix transformation model of systolic array design with the generality of the parallelising compiler approach.

Fig. 1 depicts the three common steps in systolic array design and parallelling compilation. In both areas, one starts with a source program — which is often a sequential loop nest and produces a target program — a loop nest with explicit parallelism.

Source Loop Nest

Dependence Analysis

Transformation Selection

Loop Rewriting

Target Loop Nest

Figure 1: The process of systolic array design and parallelising compilation.

In the systolic array design approach, algorithms under consideration are nested loops with uniform dependence structures [16, 17]. These algorithms have the characteristic that their dependences can be represented by integer vectors, known as the dependence vectors, which can be obtained by simple algebraic calculation. This class includes many important linear algebra codes on dense matrices, such as LU-decomposition and Gaussian-Jordan elimination method. The source and the target programs are related by one non-singular matrix transformation, known as space-time mapping. Optimal mappings can be found by either linear and integer programming (LIP) [17, 19] or a systematic enumeration of a polynomial search space [9]. A mapping provides all necessary information for fabrication of a VLSI processor array. The issue of loop rewriting was usually not addressed except in the transformational approach for unimodular transformations [12].

In the parallelising compiler approach, a more general class of loop nests is considered. To detect and further represent the dependences in this class, a lot of sophisticated dependence analysis techniques have been proposed [4, 14, 22], and some new ones are in the process of being developed. In most existing parallelising compilers, the source and target programs are related by a sequence of individual transformations like loop interchange on pairs of loops. Since the search space of transformations has potentially infinite instantiations, deciding a priori the optimal combination of transformations is expensive. Usually, the compiler applies a transformation, one step at a time, if it contributes to a particular goal. Based on the matrix transformation model used in systolic array design, a recent approach is to combine this sequence of individual transformations into one (compound) unimodular transformation. The rewriting of loop nests for simple transformations like loop interchange and reversal is straightforward. A general method for unimodular transformations also exists [2, 21].

In [24], we presented an algorithm for the automatic rewriting of loop nests under any non-singular (unimodular or non-unimodular) transformations. This algorithm allows us to unify the various transformations in systolic array design and parallelising compilation as one (compound) non-singular transformation that maximises some objective function, while satisfying a set of design constraints. This motivates us to develop this experimental software package, called SysPar, for systolising and parallelising nested loop algorithms.

2 Design Methodology
There have been many efforts towards computer-aided synthesis of systolic arrays by space-time mapping methods. Examples of such design systems include ADVIS [15], HIPI [3], PRESAGE [20] and AL-
The non-singular loop transformation problem is to rewrite the source loop nest into another loop nest, i.e., the target loop nest, such that any two iterations \( I \) and \( J \) in the source loop nest are executed in the lexicographical order of \( TI \) and \( TJ \). The target loop nest has the form:

\[
\text{DO } I' = \max([L_{i,1}', \ldots, L_{i,n}']), \\
\text{min}([U_{i,1}', \ldots, U_{i,n}'])),
\]

\[
\ldots
\]

\[
\text{DO } I'' = \max([L_{i,1}'', \ldots, L_{i,n}''], \\
\text{min}([U_{i,1}'', \ldots, U_{i,n}''])),
\]

loop-body

The rewriting of the original loop body, loop-body, into loop-body’ is straightforward. We note that \( I = T^{-1}I' \) is a set of equations expressing the original loop indices in terms of the new ones. A simple algebraic substitution is all that is required. To rewrite loop bounds and strides, we use the algorithm reported in [24], which is sketched as follows:

**Something wrong here!!!**

1. Calculate the \( L_{k,i} \) and \( U_{k,i} \) using the Fourier-Motzkin elimination algorithm or Reatir’s FIP
2. Calculate the Hermite normal form \( \Delta \) of \( T \)
3. Set \( \text{step}_k' = \Delta_{k,k} \)
4. Let \( \Delta_{k-1} \) be the \( k \times (k-1) \) submatrix at the upper-left corner of \( \Delta \).

\[
\delta_{k,i} = \left( -\frac{\det \left[ \Delta_{k-1} \right]}{\Delta_{1,1} \cdots \Delta_{1,k-1} \Delta_{k,k}} \right) \% \Delta_{k,k}
\]

where \( \% \) is the modulo operator: \( x \% y = r \) iff \( x = qy + r \), where \( x, y, q, r \in \mathbb{Z} \) and \( 0 \leq r < |y| \).

The first step can be done in parallel with the last three steps. Furthermore, the computations of all offsets \( \delta_{k,i} \) can be carried out simultaneously. If the loop transformation is unimodal, the last three steps can be omitted. In this case, \( \Delta \) is the identity matrix. So \( \text{step}_k' = 1 \) and \( \delta_{k,i} = 0 \) for all \( k \) and \( i \).

### 2.2 Systolic Array Design

The **systolic array** is a paradigm that exploits the strength of VLSI and is particularly suitable for implementing many locally recursive algorithms (e.g., DSP and scientific computation). Quite a number of methods have been proposed to map algorithms to processor arrays. For details, we refer to [16, 17] and the references therein. The loop transformation \( T \) can be further divided into two components:

\[
T = \begin{bmatrix}
\lambda \\
\sigma
\end{bmatrix}
\]

where \( \lambda \in \mathbb{Z} \) is the **scheduling vector** and \( \sigma \in \mathbb{Z}^{n \times (n-1)} \) the **allocation matrix**. The computation indexed by \( I \) is executed at time \( t \lambda \) at processor \( I \).

SysPar supports two approaches to systolic array design: the **transformational approach** and the **geometrical approach**. In the transformational approach, one
views the process of systolic array design as a process of loop transformation. The target loop nest completely specifies a systolic array [12], where the first loop represents time and the remaining $n-1$ loops the processor allocation. In the geometrical approach, one designs systolic arrays by scheduling and mapping the dependence graph of the source loop nest in space-time. The graph in space-time represents a systolic array. Geometrically, the iteration space is sliced by two types of hyperplanes: the temporal hyperplanes with the normal $\lambda$ and the spatial hyperplanes with the normal $u$ satisfying $\sigma u = 0$. A temporal hyperplane contains all the iterations that are scheduled at the same time step. A spatial hyperplane contains all the iterations that are allocated to the same processor. The right null vector $u$ of $\sigma$ is known as the projection vector. The name is to signify the fact that a systolic array is simply the projection of the dependence graph along the projection vector.

2.3 Parallelising Compilation

Parallelising compilation consists in the identification of the loops whose iterations may run simultaneously on different processors, either independently or in a overlapped fashion. In addition to the sequential loop DO, SysPar creates two more types of loops: DOALL and DOACROSS. In a DOALL loop, cross-iteration dependences do not exist and thus all iterations can be executed in parallel and in any order. However, the processes executing the iterations of a DOACROSS loop may be synchronised. A DO loop can be legally transformed into a DOALL loop iff its does not contain a loop-carried dependence. In contrast, every DO loop can be transformed into a DOACROSS loop if all dependences are satisfied by proper synchronisation. See [21, 23] and the references therein for details.

3 System Overview

SysPar is an experimental prototype system for the scheduling and mapping of nested loop algorithms with emphasis on the importance of a graphics interface based on dependence graphs. The target loop nests produced by SysPar can be implemented either as processor arrays or executed on parallel machines. The schematic structure of SysPar is shown in Fig. 2. The components enclosed in the dashed lines are for interactive scheduling and mapping of dependence graphs into systolic arrays.

A typical SysPar session for systolic array design goes as follows. The designer enters a set of nested loops in the text-editor window. SysPar checks the input to see whether there are any syntactical errors and whether the loop nest is uniform. It then calculates the dependence vectors and builds and displays the dependence graph in the 2-D or 3-D graphics interface window (Fig. 3). The window has a pull-down menu. The designer can now invoke the commands from the menu to schedule and map the dependence graph in space-time. The designer can do the scheduling and projection in either order. For any projection vector selected, SysPar displays the corresponding processor layout in the space plane and prints out some useful statistics about the array (e.g., the processor count). Once both the scheduling and projection vectors have been selected, the dependence graph scheduled and mapped in space-time will be displayed. Useful information such as the computation time and the buffers is available in the text-editor window. Finally, the target loop nest specifying the systolic array is automatically derived.

A typical SysPar session for parallelising compilation goes along the left path of Fig. 2. The dependences must be detected and represented by more sophisticated dependence analysis techniques. Here, the concept of projection vector does not apply. In addition, the manual scheduling may be difficult because the dependences may be irregular and non-uniform. The optimal transformations have to be found in an automatic manner with some possible user intervention.

Figure 2: Schematic structure of SysPar.

Figure 3: The 3-D graphical user interface.

SysPar is written in C and is currently implemented on PC486 computers.

As a simple SysPar application example, we consider the design of systolic arrays for matrix product. Matrix product is the multiplication of two square matrices $A$ and $B$ of size $m \times m$ to form a matrix $C$. Two space-time mappings are used for illustration: one describes S.Y. Kung's array and the other describes H.T. Kung's array. Fig. 4 shows the source loop nest, the two space-time mappings and the corresponding target loop nests. Fig. 5 depicts the dependence graph of the source loop nest and the processor layouts of the two arrays.

4 Conclusions and Future Work

SysPar is not intended to be a production quality system. Its purpose is rather to demonstrate that
the various transformations in systolic array design and parallelising compilation can be unified as one non-singular loop transformations.

SysPar is still in the process of being developed. From a number of experiments, we draw the following conclusions and describe some future work.

- SysPar can quickly display the processor layout, given a projection vector. After several attempts, the designer can usually find the optimal designs (say, with the smallest number of processors or some prescribed channel connections).

- At present, the designer manually finds the optimal scheduling vector based on the dependence graph. This may be hard for the dependence graphs with irregular boundaries. An interesting extension would be to provide a method that can find the optimal solutions automatically or semi-automatically. This method can be used in combination with the manual scheduling.

- To be more useful in systolic array design, SysPar has to be extended to address the following issues: the array partitioning [6], the synthesis of lower dimensional arrays [10, 13, 25], and the synthesis of control signals [26].

- SysPar supports non-unimodular transformations of loop nests. The usefulness of non-unimodular transformations is well understood in systolic array design. However, it remains to be studied how non-unimodular transformations can contribute to the two important goals in parallelising compilation: parallelism and the efficient use of the memory hierarchy.

- At present, SysPar can only perform the dependence analysis for uniform loop nests. To handle non-uniform loop nests, the techniques in [4] and their recent extensions [11] will be used. The implementation of the component "Auto-Select Transformations" in Fig. 2 will be based on the algorithms of [21] with extensions to deal with non-unimodular loop transformations.

Acknowledgements

Thanks to Neo Sae Kiong and Ne Choon Wei for drawing some pictures in this paper. The research is supported by ARP grant no. RP59/93.

References


Figure 5: S.Y. Kung' and H.T. Kung' systolic arrays.


