Optimal Bit-Level Processor Arrays for Matrix Multiplication

Chien-Wei Li and Benjamin W. Wah *
Coordinated Science Laboratory, University of Illinois, Urbana-Champaign,
Urbana, IL 61801 {cwli,wah}@manip.crhc.uiuc.edu

Abstract

Uniform recurrent algorithms belong to a fundamental class of operations used in almost every scientific and engineering computation. An important example of these algorithms is matrix multiplication. In this paper, we present optimal designs of bit-level processor arrays for matrix multiplication, and analyze trade-offs between bit-level and word-level designs in terms of execution time and chip area. We formulate the design as an optimization problem and search for optimal designs using the General Parameter Method developed by Ganapathy and Wah. We show that previous bit-level processor arrays for matrix multiplication are two special cases of our more general designs, and prove that one of the previous designs is optimal in execution time. Finally, we show that the fastest bit-level design is only $O(\log p)$ faster than than the fastest word-level design, instead of $O(p)$ as claimed in the previous work of Shang and Wah, where p is the number of bits per word.

1 Introduction

Processor arrays speed up the execution of regular computations, like matrix multiplication, by exploiting the large amount of parallelism among loop iterations and uniform or uniformized inter-iteration dependence relations, and by using many regularly connected processing elements (PEs) and fast inter-processor communication. In a word-level processor array, communication between PEs is in words, which requires each PE to gather all the bits of a result before sending them to another PE. In contrast, a bit-level PE can send part of its result bits to other PEs without waiting for all result bits to be generated. Hence, by utilizing parallelism at the bit level, a bit-level processor array will be faster than a word-level processor array.

Previously, Shang and Wah [3] have proposed two bit-level processor arrays for matrix multiplication. Based on their design method, they can only show that their designs are feasible but not optimal. This paper applies another design method called General Parameter Method (GPM), which was developed by Ganapathy and Wah [1] to find optimal word-level processor arrays. We extend GPM to design optimal bit-level processor arrays for matrix multiplication under various design objectives and constraints. In addition to showing that

* This research was supported by National Science Foundation Grant MIP 92-18715.

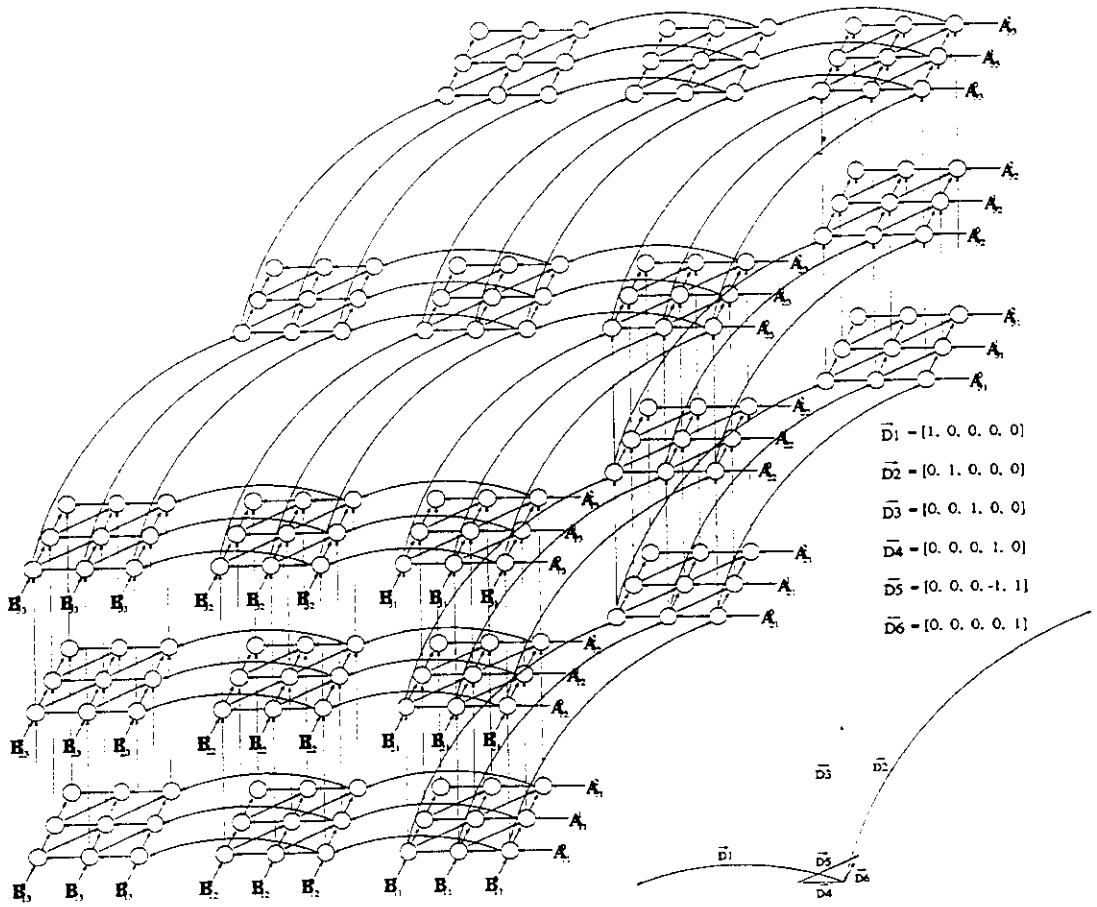

Figure 1: 5-D dependence graph for bit-level matrix multiplication of two 3-by-3 matrices with 3-bit numbers

one of previous designs by Shang and Wah [3] is optimal in terms of execution time, we also present alternative designs with different trade-offs between execution time and chip area.

This paper is organized as follows. Section 2 discusses the problem of designing optimal bit-level processor arrays for matrix multiplication. Section 3 formulates the problem of designing bit-level processor-arrays. Section 4 shows four alternative designs and compares them with word-level designs.

2 Bit-Level Matrix Multiplication

Figure 1 shows the data flows among bit-level matrix-multiplication operations, which was obtained by expanding a uniformized word-level dependence graph. Each word-level node representing a word-by-word multiplication and addition is expanded into a set of bit-level nodes that represent the bit-by-bit operations implementing the shift-and-add multiplication algorithm. By this expansion, each word-level arc corresponds to a set of bit-level arcs.

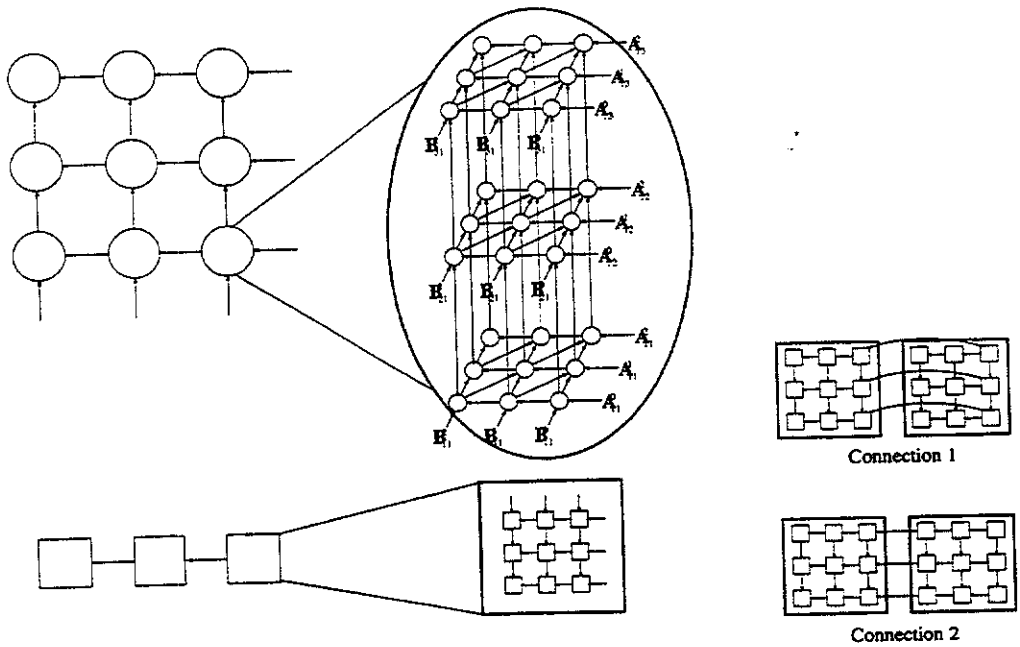

Figure 2: Decomposition of the 5-D dependence graph in Figure 1 into 2-D and 3-D dependence graphs

This expansion process results in a non-uniform dependence graph. For instance, data token A can move into a node along either $\vec{d}_1$ or $\vec{d}_4$, and data token B along either $\vec{d}_2$ or $\vec{d}_6$. Since two nodes of the dependence graph mapped to the same PE may take their input data from different directions, we need two control signals and the associated control circuits to direct each PE to get their inputs from the appropriate links at the right time. Although this practice is common in word-level designs, it is deemed unacceptable in bit-level design as each bit-level PE implements very simple bit-level operations.

To deal with this problem, we decompose the 5-D dependence graph in Figure 1 into a uniform 2-D dependence graph at the word level and a uniform 3-D dependence graph at the bit level. This decomposition is illustrated in Figure 2. We then design a processor array for each dependence graph separately, and expand each "PE" designed for the 2-D dependence graph into a processor array designed for the 3-D dependence graph. Since the dependence relations between two nodes in the 2-D dependence graph are "Read-After-Read" relations with respect to inputs A and B, two sets of PEs corresponding to two nodes in the 2-D dependence graph can be connected in two different ways (see Figure 2). The first structure can pipeline input data faster but have longer wires as compared to the second structure. Using this scheme, every bit-level PE will have the same structure without specific control signals and circuits. For each dependence graph, we apply GPM to find alternative designs with different trade-offs.

3 Problem Formulation

Suppose the goal is to find a bit-level design with the minimum time for multiplying two N-by-N matrices. The design can be formulated into the following two optimization problems corresponding to the 2-D and 3-D dependence graphs, respectively. The corresponding parameters in GPM are the periods (t_i), displacements ($\vec{k}_i$) and spacing parameters ($\vec{s}$). For the 2-D dependence graph, we have

$$\begin{array}{lll} \min & T_{comp} = 1 + (N-1)(t_1 + t_2) & \\ \text{s.t.} & t_i \geq 1 & \text{for } i, j = 1, 2 \text{ and } i \neq j \\ & t_i \geq |\vec{k}_i| \geq 0 & \\ & \vec{s}_{i,j} = \vec{k}_j - t_j(\vec{k}_i / t_i) & \text{no input data conflict for } \vec{s}_{i,j} \end{array}$$

For the 3-D dependence graph, we have

$$\begin{array}{lll} \min & T_{comp} = 1 + (p-1)(2t_1 + t_2) + (N-1)t_3 & \\ \text{s.t.} & t_i \geq 1 \text{ for } i = 1, 2, 3 & t_4 = t_1 + t_2 \\ & t_i \geq |\vec{k}_i| \geq 0 \text{ for } i = 1, 2, 3 & \vec{k}_4 = \vec{k}_1 + \vec{k}_2 \\ & \vec{s}_{i,j} = \vec{k}_j - t_j(\vec{k}_i / t_i) \text{ for } i \neq j & \text{no input data conflict in } \vec{s}_{i,j} \end{array}$$

For the 2-D dependence graph, both 1-D and 2-D processor arrays are considered. In contrast, we only consider 2-D processor arrays for the 3-D dependence graph, as 1-D bit-level designs are very inefficient.

By adding another constraint

$$\#PE \leq \text{Upper Bound}$$

and by tightening the upper bound gradually, we are able to obtain different trade-off between T_{comp} and $\#PE$.

Other design objectives and constraints can be considered, including number of PEs ($\#PE$), number of latches ($\#BUF$), input load time (T_{load}), output drain time (T_{drain}), and completion time ($T_c = T_{load} + T_{comp} + T_{drain}$). As each bit-level PE is of the complexity of a latch, we must estimate the cost of a chip using both $\#PE$ and $\#BUF$.

4 Results

Table 1 shows two possible solutions for the 2-D and 3-D dependence graphs. $\mathcal{D}^1_{3D}$ has the minimum $T_c = T_{comp} = N + 3p - 3$ and needs p^2 PEs and $p(p-1)$ latches. For solutions

Table 1: Alternative designs implementing the 2-D and 3-D dependence graphs

$\mathcal{D}^1_{2D}$	$\mathcal{D}^2_{2D}$	$\mathcal{D}^3_{2D}$
$t[1,2] = [1,1]$	$t[1,2] = [1,1]$	$t[1,2] = [1,1]$
$\vec{k}_1 = [1,0]$	$\vec{k}_1 = [1,0]$	$\vec{k}_1 = [1,0]$
$\vec{k}_2 = [0,1]$	$\vec{k}_2 = [-1,0]$	$\vec{k}_2 = [0,0]$

$\mathcal{D}^1_{3D}$	$\mathcal{D}^2_{3D}$
$t[1,2,3,4] = [1,1,1,2]$	$t[1,2,3,4] = [1,1,1,2]$
$\vec{k}_1 = [1,0]$	$\vec{k}_1 = [1,0]$
$\vec{k}_2 = [-1,1]$	$\vec{k}_2 = [-1,0]$
$\vec{k}_3 = [0,0]$	$\vec{k}_3 = [0,1]$
$\vec{k}_4 = [0,1]$	$\vec{k}_4 = [0,0]$

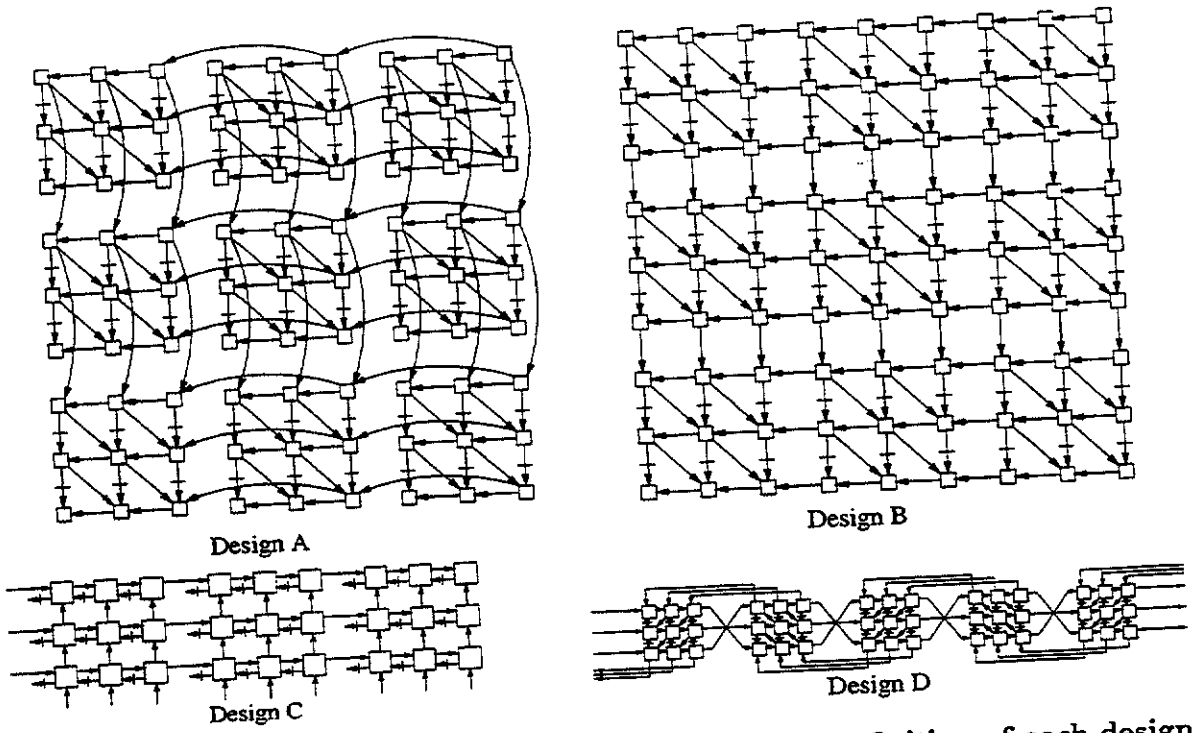

Figure 3: Four alternative designs. See Table 2 for the definition of each design. Designs A and B were first proposed by Shang and Wah [3].

with stationary inputs, $\mathcal{D}^2_{3D}$ has $T_{preload} = T_{drain} = p$ and the minimum $T_{comp} = N + 3p - 3$ and $T_c = T_{preload} + T_{comp} + T_{drain} = N + 5p - 3$, and needs Np PEs.

By combining these solutions using different connections, and by excluding combinations that result in conflicts in input data, we get four alternative designs. Figure 3 illustrates these alternatives for $N = 3$ and $p = 3$, and Table 2 shows the definition of each design.

T_{comp} and T_c of the combined solution depend on T_{comp} and T_c of $\mathcal{D}^i_{3D}$, the time for input data to pipeline through $\mathcal{D}^i_{3D}$, and the structure of $\mathcal{D}^j_{2D}$. Obviously, Design A is optimal in both T_{comp} and T_c because $\mathcal{D}^1_{3D}$ is time-optimal and its input data are pipelined in the fastest way. Table 2 summarizes T_c, $\#PE$ and $\#BUF$ for the four designs.

It is worthwhile to note that for Designs A, B and D, $T_c \times \#PE = O(N^3p^2)$, which is the total number of nodes in the 5-D dependence graph. This means that these three designs achieve the maximum efficiency asymptotically.

In comparing our designs with word-level processor arrays for matrix multiplication, we note that the latter can have either $O(N)$ PEs in order to have $T_c = O(N^2)$, or $O(N^2)$ PEs with $T_c = O(N)$. Each word-level PE can use carry-save adders to implement its word-level multiplication, which requires $O(p \log p)$ space and $O(\log p)$ execution time [2]. On the whole, a word-level processor array may need either $O(Np \log p)$ space to get $O(N^2 \log p)$ execution time, or $O(N^2 p \log p)$ space to get $O(N \log p)$ time. Hence, the fastest bit-level design is $O(\log p)$ faster than the fastest word-level design. Figure 4 shows the complexities of the alternative bit-level design as compared to the alternative word-level designs W_{2D} and W_{1D} in terms of T_c, $\#PE$, and $\#BUF$.

Table 2: Cost-performance of four proposed designs. See Table 1 for the definitions of the 2-D and 3-D parts; see Figure 2 for connection type.

Design	2-D Part	3-D Part	Connection	T_c	$\#PE$	$\#BUF$	I/O
A	$\mathcal{D}^1_{2D}$	$\mathcal{D}^1_{3D}$	1	$3N+3p-5$	N^2p^2	$N^2p(p-1)$	$2N$
B	$\mathcal{D}^1_{2D}$	$\mathcal{D}^1_{3D}$	2	$3Np-2$	N^2p^2	$N^2p(p-1)$	$2N$
C	$\mathcal{D}^2_{2D}$	$\mathcal{D}^2_{3D}$	2	$3Np+N+p-2$	N^2p	N^2p	N
D	$\mathcal{D}^2_{2D}$	$\mathcal{D}^2_{3D}$	2	$2N^2-N+3p-3$	$(2N-1)p^2$	$4Np(N-p)$	p

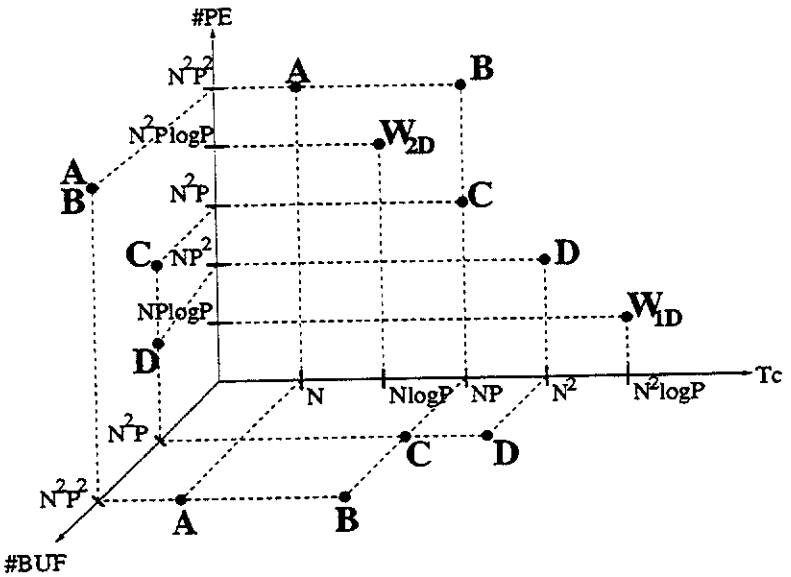

Figure 4: Asymptotical cost-performance trade-offs of alternative designs

References

[1] Kumar N. Ganapathy. *Mapping Regular Recursive Algorithms to Fine-Grained Processor Arrays.* PhD thesis, University of Illinois at Urbana-Champaign, 1994.

[2] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 1990.

[3] Weijia Shang and Benjamin W. Wah. Dependence analysis and architecture design for bit-level algorithms. In *Proc. Int'l Conf. on Parallel Processing*, volume 1, pages 30–38. CRC Press, 1993.