Cyclic Staggered Scheme: A Loop Allocation Policy for DOACROSS Loops

A.R. Hurson, K. Kavi, and J.T. Lim

Abstract—Within the scope of the multithreaded dataflow, the problem of scheduling/allocation of DOACROSS loops has been discussed and it was shown that the so-called *staggered allocation* offers higher performance and resource utilization than other schemes described in the literature. The staggered scheme, however, produces an unbalanced load among processors. This paper introduces an extension to the staggered scheme—*cyclic staggered scheme*—that produces a more balanced distribution of iterations among processors. The cyclic staggered scheme is simulated and its performance improvement is analyzed.

Index Terms—Loop allocation, DOACROSS loop, multithreaded dataflow organization, scheduling and load balancing, control-flow multiprocessor organization, simulation.

1 Introduction

In a traditional multiprocessor organization, the basis of control-flow processing is extended to allow more than one execution thread to be active at an instance. However, architects of such an organization must address the loss in processor efficiency due to two fundamental issues: memory latencies and synchronization overhead. The *dataflow* model of computation was proposed as an alternative to the conventional control-flow model of computation. It explicitly addresses the issue of programmability as well as memory latency and synchronization. Theoretically, in a dataflow machine, maximal concurrency can be exploited, constrained only by the availability of hardware resources.

There are basically three types of loops; sequential loops, vector loops (DOALL), and loops of intermediate parallelism (DOACROSS) [1]. For a DOALL loop, all N iterations can be executed concurrently. Sequential loops have zero percent parallelism and would not gain any improvement if executed in a multiprocessor. Hence, loops of intermediate parallelism are of the greatest interest, since these can be scheduled or distributed in various ways to achieve speedup in a multiprocessor environment. The DOACROSS loop model proposed in [1] was aimed to model the execution of sequential loops, vector loops, and loops of intermediate parallelism by considering control and data dependencies.

This paper considers DOACROSS loops which have a *lexically-backward dependence* (*LBD*) that cannot be eliminated by reordering the statements. A new loop scheduling that maximizes the utilization of processors while achieving significant speedup is examined.

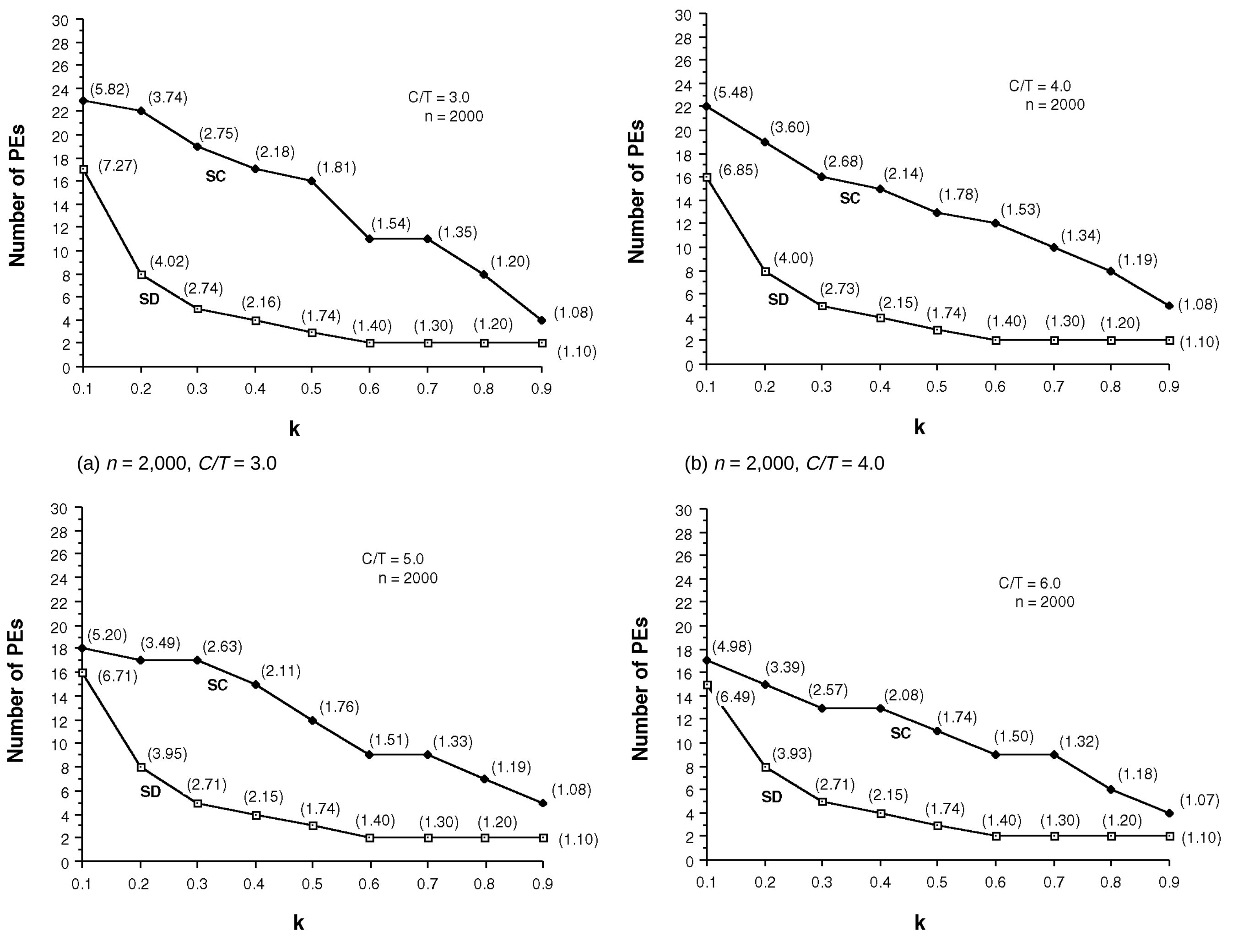
2 Staggered Distribution Scheme

The Staggered distribution originally developed for multithreaded dataflow multiprocessors [4], [7] uses heuristics to distribute the loop iterations unevenly among processors to mask the delay caused by data dependencies as well as inter-PE communication. If all iterations start at the same time and each processor is assigned

• *A.R. Hurson and J.T. Lim are with the Computer Science and Engineering Department, The Pennsylvania State University, University Park, PA 16802. E-mail: hurson@cse.psu.edu.*
• *K. Kavi is with the Computer Science and Engineering Department, The University of Texas at Arlington, Arlington, TX 76019.*

Manuscript received 4 Dec. 1994; revised 8 Nov. 1995.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number 104104.0.

0018-9340/98/$10.00 © 1998 IEEE



(a) *n* = 2,000, *C/T* = 3.0

(b) *n* = 2,000, *C/T* = 4.0

(c) *n* = 2,000, *C/T* = 5.0

(d) *n* = 2,000, *C/T* = 6.0

Fig. 1. Number of PEs to attain maximum speedup.

the same number of iterations, then all processors would take the same amount of time to finish executing the $T(S_1, S_s)$-d portion of the loop—independent portion of each loop. But each processor PE_i ($1 < i \le P$), has to wait for processor PE_{i-1} to finish executing the d portion and send the partial results (synchronization message) to allow processor PE_i to continue execution. This creates delays due to the *LBD* and communication.

To achieve higher performance and resource utilization, the loop iterations are then distributed according to the following policy: The iterations assigned to PE_i succeed the iterations assigned to PE_{i-1} with PE_i having m more iteration nodes assigned to it than PE_{i-1}. The delay caused by iterations assigned to PE_{i-1} will be equal to d per iteration plus the communication cost C. This delay will be masked out by the $T(S_1, S_s) - d$ portion of the additional iterations (m) assigned to PE_i. As a result, instead of waiting for the message to arrive, the additional number of iterations in each chunk relative to the previous chunk keeps the processor busy. Incremental distribution of the iterations among processors is hence determined by:

$$m_i = \left\lceil \left(n_{i-1} * d + C\right) \big/ \left(T\left(S_1, S_s\right) - d\right) \right\rceil \tag{1}$$

where n_{i-1} is the number of iterations allocated to PE_{i-1}, $T(S_1, S_s)$ is the execution time of one iteration, d is the delay, and C is the inter-processor communication cost. The number of iterations n_i allocated to PE_i would be:

$$n_i = n_{i-1} + m_i = \left\lceil \left(n_{i-1} * T\left(S_1, S_s\right) + C\right) \big/ \left(T\left(S_1, S_s\right) - d\right) \right\rceil \tag{2}$$

The distribution is performed by expanding (2) to determine the value of n_1—number of iterations assigned to the first processor. The n_1 value is used to calculate n_i ($1 < i \le P$). The n_i^s are then fine tuned for better resource utilization. This scheme automatically controls and determines the maximum number of processors (*maxpe*) required for efficient execution of the loop based on the physical characteristics of the loop and the underlying machine architecture—i.e., higher resource utilization. The maxpe can be determined by expanding (1) and (2), and considering the n_1 and n. The synchronization overhead is only $(P - 1) * C$, which is significantly less than the synchronization overhead incurred by cyclic scheduling and pre-synchronized scheduling. Staggered scheme, however, distributes an unbalanced load among processors, with the last processor receiving the largest number of iterations. To remedy this problem and to be able to handle variations in iteration execution times, a modification of this scheme is required and is presented in the next subsection.

2.1 Cyclic Staggered Distribution

As mentioned earlier, the *staggered* scheme determines the maximum number of processors (*maxpe*) required for each loop. If the

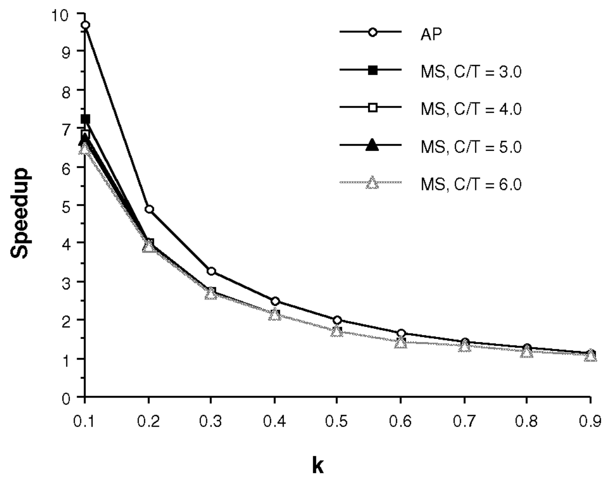


Fig. 2. Maximum speedup (MS) using Staggered distribution.

number of processors available P is less than *maxpe*, the initial distribution for P would be the same as the first P processors of *maxpe*. The remaining iterations n_r are then redistributed among the P processors starting from the first processor, utilizing the *staggered* concept according to (3).

$$\begin{aligned} n_i &= \left\lceil \left(n_{i-1} * T(S_1,S_s) + C - n_p * T(S_1,S_s)\right) \Big/ \left(T(S_1,S_s) - d\right) \right\rceil \\ &= \left\lceil \left(\left(n_{i-1} - n_p\right)T(S_1,S_s) + C\right) \Big/ \left(T(S_1,S_s) - d\right) \right\rceil \end{aligned} \tag{3}$$

where n_p is the number of iterations previously allocated to processor PE_i. The extended scheme results in a more balanced load and improved speedup than the original staggered scheme on P processors. We refer to this scheme as CS1. In case of variations in iteration execution time, this scheme can be utilized dynamically during run-time to account for the difference between the worst case iteration execution time and the actual execution time in determining the distribution for the second and subsequent passes. As an alternative, one iteration is assigned to the first processor in the first pass, and subsequent iterations are assigned to the next processors using (2). After the first pass, (3) is applied. We call this scheme CS2.

The number of iterations assigned to a processor at each scheduling step for cyclic, static chunking (SC), staggered distribution (SD), and cyclic staggered (first version (CS1) and second version (CS2)) has been simulated [6]. It was shown that the two cyclic staggered versions offer more even distribution than the staggered scheme. CS2 distributes iterations more evenly since it assigns smaller chunks per scheduling step than CS1. CS1 and CS2 incur more communication cost, however, they offer an overall better execution time.

2.2 Cyclic Staggered for Control-Flow Environment

The cyclic staggered scheme can be adapted to a control-flow environment. In order to get the same behavior for a control-flow environment the loop would have to be separated into two loops [6]. The first loop would be the instructions that are involved in the $T(S_1, S_s) - d$ portion of the loop and the second would be the instructions involved in the d portion.

3 Simulation Results

Effectiveness of the Staggered scheme has been simulated and compared against those of static chunking and cyclic scheduling [4]. Dynamic scheduling schemes, such as GSS and Factoring, have been proposed for vector loops, therefore, they cannot be used as a means to evaluate the staggered scheme. Furthermore, our studies have shown that static chunking performs better than the aforementioned dynamic schemes. We also did not consider presynchronized scheduling [5], since the best-case performance of this scheme would be equivalent to cyclic scheduling.

Our test-bed includes a representative loop with the execution time of $T(S_1, S_s) = 50$ and loops 3, 5, 11, 13, and 19 of the Livermore Loops, which have cross-iteration dependencies [3]. In our simulation:

TABLE 1
Speedup of Staggered Distribution Relative to Static Chunking Su(SC) and Cyclic Scheduling Su(CYC) for the Livermore Loops with *C/E* = 30

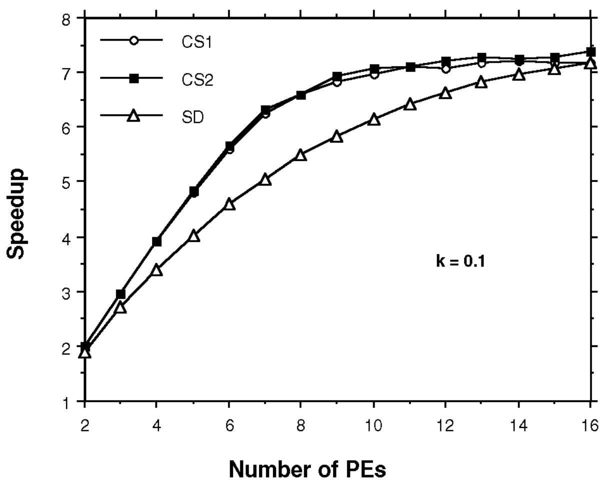
			PE = 4		PE = 8	
LOOP #	k	C/T	Su (SC)	Su (CYC)	Su (SC)	Su (CYC)
3	0.25	3.75	1.20	10.72	1.21 (7)	13.10 (7)
5	0.30	3.00	1.21	8.22	1.16 (6)	9.35 (6)
11	0.25	3.75	1.21	10.50	1.21 (7)	12.18 (7)
13	0.05	0.71	1.07	2.82	1.14	5.05
19(1)	0.33	3.33	1.24	7.53	1.34 (4)	7.53 (4)
19(2)	0.27	2.73	1.23	6.86	1.28 (5)	6.93 (5)

Actual number of PEs used by Staggered Distribution in parentheses.

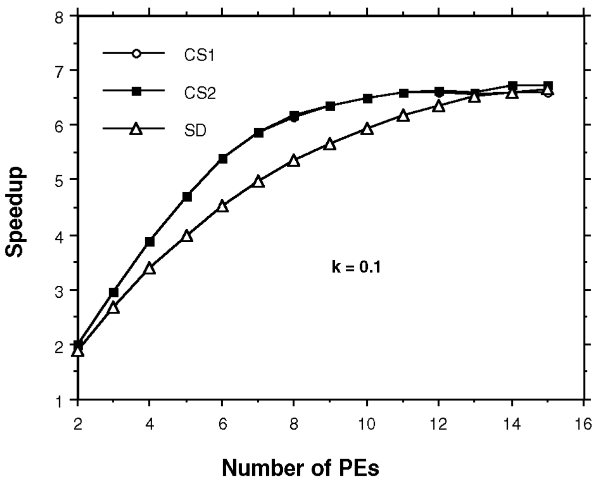
TABLE 2
Speedup of Staggered Distribution Relative to Static Chunking Su(SC) and Cyclic Scheduling Su(CYC) for the Livermore Loops with *C/E* = 107

			PE = 4		PE = 8	
LOOP #	k	C/T	Su (SC)	Su (CYC)	Su (SC)	Su (CYC)
3	0.25	13.38	1.22	34.80	1.25 (6)	39.00 (6)
5	0.30	10.70	1.22	26.42	1.19 (5)	28.18 (5)
11	0.25	13.38	1.24	32.63	1.33 (6)	34.17 (6)
13	0.05	2.55	1.07	9.50	1.16	16.31
19(1)	0.33	11.89	1.37 (3)	19.36 (3)	1.99 (3)	19.36 (3)
19(2)	0.27	9.73	1.31 (3)	17.39 (3)	1.83 (3)	17.39 (3)

Actual number of PEs used by Staggered Distribution in parentheses.

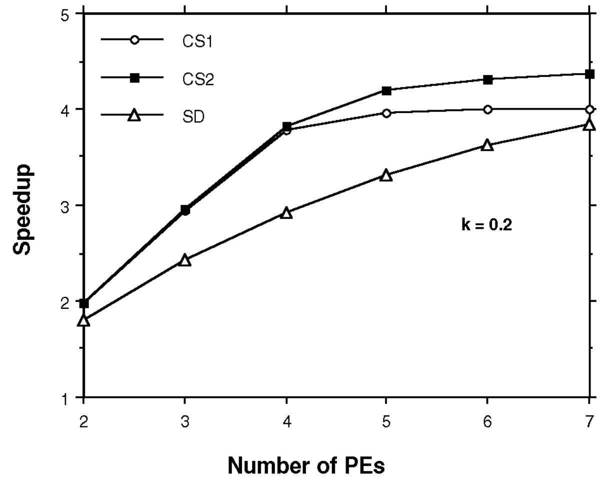


(a) $C/T = 3.0$

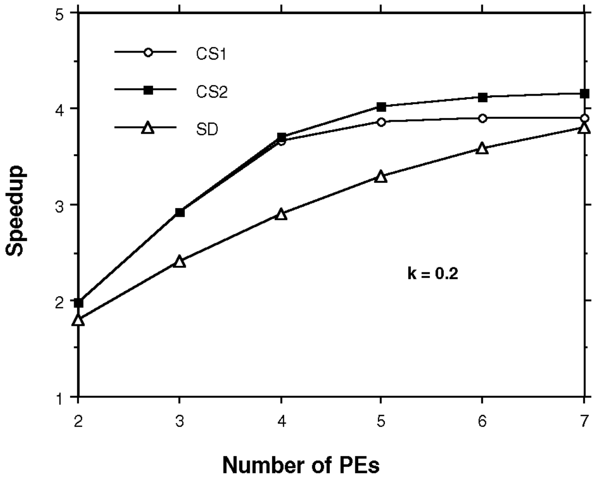


(b) $C/T = 5.0$

Fig. 3. Comparative analysis of the staggered schemes, $k = 0.1$.



(a) $C/T = 3.0$



(b) $C/T = 5.0$

Fig. 4. Comparative analysis of the staggered schemes, $k = 0.2$.

1) The inter-PE communication delays are varied based on the ratio of *communication time to iteration execution time* ($C/T(S_1, S_s)$).
2) Delays due to LBD are computed for various k values, where $k = d/T(S_1, S_s)$.

We also computed the average parallelism (AP), which is the ratio of the total execution time to the critical-path length:

$$AP = \left(n * T(S_1, S_s)\right) / \left(T(S_1, S_s) + d(n-1)\right) \qquad (4)$$

The number of PEs required to attain maximum speedup for both Staggered distribution (SD) and Static chunking (SC) has been simulated and analyzed. Fig. 1 shows the results for $T(S_1, S_s) = 50$, $n = 2{,}000$, and k varying from 0.1 to 0.9. In general, irrespective of the values of n, $C/T(S_1, S_s)$, and k, the staggered approach uses fewer processors to obtain significant speedup, even though the common effect of delays due to the lexically backward dependency (LBD) and inter-processor communication tends to reduce resource utilization. As can be seen from Fig. 1, for k from 0.1 to 0.4, the staggered approach offers a greater speedup, and for k from 0.5 to 0.9, it achieves almost the same speedup factor. The static chunking scheme distributes the iterations evenly among the processing elements without any consideration for delays. Each processor, except the first, is idle for some period of time, which makes static chunking an inefficient scheme. From Figs. 1a and 1d, it can be concluded that the speedup achieved by SD at $k = 0.1$ decreases from 7.27 to 6.49, while, for SC, it decreases from 5.82 to 4.98. Aside from the fact that SD attains a significantly higher speedup, the speedup for SD decreased by only 10.7 percent, while, for SC, speedup decreased by 14.4 percent when the communication cost was doubled. On the other hand, for $k = 0.2$, the speedup for SD decreased by only 2.24 percent, but, for SC, speedup decreased by 9.4 percent. Both schemes however, require fewer PEs to attain maximum speedup as the $C/T(S_1, S_s)$ ratio increases.

As expected, Cyclic scheduling (CYC) is ineffective if the communication cost is significant. For $C/T(S_1, S_s)$ greater than or equal to 1.0, CYC did not produce any speedup. Hence, we ran simulations for lower $C/T(S_1, S_s)$ ratios. Our simulation results showed that, as the $C/T(S_1, S_s)$ ratio increases, the speedups realized by the CYC scheme decreases faster in comparison with the SD scheme. The speedup achieved by SD at $k = 0.1$ decreases from 8.14 to 8.00, while for CYC, it decreases from 3.33 to 1.67. Aside from the fact that CYC attains a significantly lower speedup, the speedup for SD decreased by only 1.7 percent, while, for CYC, speedup decreased by 50 percent when the communication cost was almost doubled. The speedups for SD from $k = 0.2$ and up remained constant, as the

$C/T(S_1, S_s)$ ratio increases, while for CYC the speedup falls rapidly, dropping to less than two for $C/T(S_1, S_s) = 0.5$ and less than one when $k = 0.6$. Finally, the maximum speedups attained by CYC for $C/T(S_1, S_s) = 1.0$ and up are all less than one. This means that the loops can obtain better performance if they were executed serially in one PE. The number of PEs required to realize maximum speedup for CYC drops to two independent of k for $C/T(S_1, S_s) \geq 0.5$. This is due to the fact that for $C/T(S_1, S_s) = 0.5$, after two iterations, the communication delay would be equivalent to the execution time of one iteration $T(S_1, S_s)$. Therefore, the third and fourth iterations can be executed in the same two processors without any additional delay. The cycle will be repeated for every pair of iterations—using more processors does not affect the performance.

Fig. 2 depicts the average parallelism (*AP*) and the maximum speedup (*MS*) for $n = 2{,}000$. In general, regardless of the values of n, $C/T(S_1, S_s)$, and k, the SD scheme in the presence of inter-PE communication, offers a maximum speedup close to the average parallelism.

Tables 1 and 2 show the speedup of SD over SC and CYC when the Livermore Loops were simulated. Timing values and interprocessor communication used in the simulation were based upon instruction and communication times for the nCUBE 3200 and 6400 [2]. Loop 19 consists of two loops. Therefore, we tested each loop separately (19(1) and 19(2)). The number of iterations for each loop were based on the specification of each loop. Loops 3, 5, and 13 were simulated for $n = 1{,}000$, Loop 11 with $n = 500$, and Loops 19(1 and 2) with $n = 100$. Although the number of iterations for Loop 11 can reach a maximum of 1,000, we felt that 500 would give us a different perspective from Loop 3, since they both have the same value of k. There was not much speedup for Loop 13, since it had a negligible delay. For Loops 3, 5, 11, and 19(1 and 2) when $PE = 8$, the SD scheme utilized fewer PEs than the available number of PEs. This agrees with the results shown earlier in Fig. 1 that SD offers better resource utilization. Furthermore, the number of PEs required also decreases as the communication cost increases.

Effectiveness of the Cyclic Staggered scheme, first version (CS1) and second version (CS2), was simulated and compared against the original Staggered scheme (SD). The speed-up factor has been used as a measure of the evaluation. Our test-bed includes the same representative loop with an execution time of $T(S_1, S_s) = 50$ and $n = 2{,}000$. The speedup attained was calculated by varying the k, the interprocessor communication cost, and the number of the processors (Figs. 3 and 4). As can be seen, both cyclic staggered distribution schemes performed better than SD regardless of the values of n, $C/T(S_1, S_s)$, and k, especially when the number of PEs was halfway between two and *maxpe*-1. This is due to the fact that under such circumstances, both schemes have a higher number of remaining iterations n_r for redistribution, which results in a more balanced load. Also, since the number of PEs is greater than two, the remaining iterations can be distributed to more PEs, again resulting in a more balanced load, hence better speedup. As expected, the speedups start to converge as the number of PEs approaches the *maxpe*, since these schemes produce a distribution similar to SD. This is more evident in the case of CS1. Finally, both CS1 and CS2 schemes attained an almost linear speedup for smaller number of PEs, even with delays due to LBD and communication cost. We showed that SD offers better resource utilization, since it attains better speedup than cyclic scheduling and static chunking, utilizing fewer number of PEs. We also showed that the maximum speedup for SD in the presence of inter-PE communication, is very close to the average parallelism, which is the maximum speedup possible for a particular loop. Since CS1 and CS2 outperform SD, we can conclude that CS1 and CS2 come even closer to the maximum speedup possible for a particular loop. However, these advantages are made possible if the number of PEs available is less than *maxpe*.

4 Summary and Future Directions

A distribution scheme for DOACROSS loops—*Cyclic Staggered* distribution with its two variations—has been introduced. It uses the same concepts as our previous strategy, *Staggered* distribution—to distribute the loop iterations, albeit unevenly, among processors in order to mask out the delay caused by data dependencies as well as inter-PE communication. This approach offers a more balanced load and better speedup. Effectiveness of this new scheme relative to our previous *Staggered* scheme has been reported, based on simulation and execution on an nCUBE 2 multiprocessor [6]. *Cyclic Staggered* scheme attains better speedup than *Staggered*, when the number of PEs is less than *maxpe* (number of processors needed to attain optimum speedup). It also produces an almost linear speedup for a small number of PEs, even in the presence of LBDs and inter-PE communication.

The success of multithreading depends on how quickly context switching can be supported. This is only possible if threads are resident in fast memories, such as cache. The sizes of cache are usually small, hence the number of active threads and, thus, the amount of latency that can be tolerated is limited. The generality of dataflow scheduling makes it difficult to execute a logically related set of threads through the processor pipeline, thereby removing any opportunity to utilize registers across thread boundaries. Relegating the responsibilities of scheduling and storage management to the compiler alleviates this problem to some extent. Appropriate means of directing scheduling based on some global-level understanding of program execution will be crucial to the success of future dataflow architectures. We are currently investigating strategies based on cyclic staggered approaches to enhance locality in dataflow architectures, for the purpose of using cache memories.

Acknowledgments

This work has been supported in part by the U.S. National Science Foundation under Grants MIP-9622836 and MIP-9622593.

References

[1] R. Cytron, "DOACROSS: Beyond Vectorization for Multiprocessors," *Proc. Int'l Conf. Parallel Processing*, pp. 836-844, 1986.
[2] T.H. Dunigan, "Performance of the Intel iPSC/860 and Ncube 6400 Hypercubes," *Parallel Computing*, vol. 17, pp. 1,285-1,302, 1991.
[3] J.T. Feo, "An Analysis of the Computational and Parallel Complexity of the Livermore Loops," *Parallel Computing*, vol. 7, pp. 163-185, 1988.
[4] A.R. Hurson, J.T. Lim, K. Kavi, and B. Shirazi, "Loop Allocation Scheme for Multithreaded Dataflow Computers," *Proc. Eighth Int'l Parallel Processing Symp.*, pp. 316-322, 1994.
[5] V.P. Krothapalli and P. Sadayappan, "Dynamic Scheduling of DOACROSS Loops for Multiprocessors," *Proc. Parbase-90: Int'l Conf. Databases and Parallel Architectures*, pp. 66-75, 1990.
[6] J.T. Lim, A.R. Hurson, K.M. Kavi, and B. Lee, "A Loop Allocation Policy for DOACROSS Loops," *Proc. Symp. Parallel and Distributed Processing*, pp. 240-249, 1996.
[7] J.T. Lim, A.R. Hurson, B. Lee, and B. Shirazi, "Staggered Distribution: A Loop Allocation Scheme for Dataflow Multiprocessor Systems," *Proc. Fourth Symp. Frontiers of Massively Parallel Computation*, pp. 310-317, 1992.
[8] C.D. Polychronopoulus and U. Banerjee, "Processor Allocation for Horizontal and Vertical Parallelism and Related Speedup Bounds," *IEEE Trans. Computers*, vol. 36, no. 4, pp. 410-420, Apr. 1987.