Asynchronous Dynamic Load Balancing of Tiles*

Tung Nguyen† Michelle Mills Strout† Larry Carter† Jeanne Ferrante †

Many scientific computations have work-intensive kernels consisting of nested loops with a regular stencil of data dependences. For such loop nests, tiling [3] is a well-known compiler optimization that can help achieve efficient parallelism. For a loop nest of depth K, tiling partitions the K-dimensional space of iterations into regular size and shape chunks of work. These *tiles* are allocated to different processors, giving parallelism and locality. In this paper, we consider the simple but common case of a doubly nested loop with a regular stencil of dependences in both dimensions. These dependences give rise to tile dependences, which require synchronization and communication of data between tiles. We address the specific case where a tile needs data computed in the tile to its left and the tile below it to execute. Execution of tiles proceeds in a wavefront fashion: each processor will be at least one row ahead of its neighbor to the right. Rebalancing can prevent much larger imbalances that would otherwise introduce idle time.

Our target architecture is a distributed-memory computer such as a Cray T3E, IBM SP, or network of workstations. Initially, each processor is allocated the same fixed number of columns of tiles in block distribution. Each processor executes its allotted tiles a row at a time. If each processor has the same workload and computing capacity, then this allocation should be efficient. However, if the workloads or compute power vary, then dynamically adjusting the allocation of columns to achieve a better load balance is desirable. Varying processor loads can arise because of an evolving irregular application or because of contention in a non-dedicated system.

We present the Left-Right Handoff Protocol, a dynamic load balancing algorithm which does not require centralized control. This protocol can be described as a series of asynchronous "races" between pairs of contiguous processors. In alternate rounds, a processor alternates between racing against its left neighbor and its right neighbor. Each processor executes a fixed percentage of its workload, then signals its neighbor it has reached its "checkpoint". If the neighbor receives this signal before reaching its own checkpoint, it may decide to hand off some of its work to the sender, assuming it determines that this will increase the performance of the both processors. To actually hand off work, the processor sends a message that includes the amount of work being transferred, and the data that the receiver will need to perform the work. A key feature of this algorithm is that the processor reaching the checkpoint first can (without waiting for a reply) complete its row. Assuming the imbalance isn't too large, the reply will be available at the row's end, and neither processor will have any idle time. We note, however, that the protocol involves more overhead than a static schedule, since extra rounds of commuication are involved.

There is an enormous amount of literature on dynamic load balancing [5]. Our scheme uses a local rescheduler; this has advantage over methods using a global scheduler, such as [1], which can require a dedicated processor and global synchronization, and may not scale well to a large number of processors. The Left-Right Handoff Protocol is an "asynchronous deterministic iterative diffusion" method, where the work can "diffuse" over time from processors with high "concentrations" of work to less busy neighbors [4]. Asynchronous methods have the potential to be faster than synchronous protocols; however, the rate of convergence and the optimal communication schedule are not theoretically known [4]. Experimental results are therefore useful in determining the practicality of asynchronous schemes.

Our experiments used a simple SOR (Successive Over Relaxation) on the Cray T3E using 8 processors. Performance was measured in three scenarios: (1) Load is distributed equally among

*This work was supported in part by NSF REU grant CCR-9504150.

†Computer Science and Engineering Department, University of California at San Diego

processors. (2) Load is synthetically introduced by requiring the i-th processor to execute each tile i times. (This simulates either increasingly heavy contention, or increasingly more iterations needed, on the higher-numbered processors.) (3) Load is increased for the low number processors by requiring processor i to execute each tile $9-i$ times. In each scenario, our protocol was compared against the non-load balancing version of the application. The results are given in the table below.

	No Load		Increasing Load		Decreasing Load	
Tile Size	Static	Dynamic	Static	Dynamic	Static	Dynamic
5×5	0.53 sec	0.87	4.31	3.51	4.03	3.27
10×10	0.47	0.57	3.63	2.16	3.62	2.02
20×20	0.46	0.50	3.54	1.87	3.53	1.77
50×50	0.47	0.52	3.60	2.13	3.58	2.95
100×100	0.53	0.52	4.09	3.76	3.75	3.76

TABLE 1

Execution time in seconds, assuming three different workloads.

For (1), we did not expect our protocol to perform better than the static version, but hoped it would give competitive results. The experiments confirmed our hope, except for the smallest tile sizes (which entail the most extra overhead since there are more rows of tiles). Surprisingly, for the largest tile size, dynamic load balancing performed better, despite its extra overhead! It turns out that for all tile sizes, both processor 1 and 8 end up with more than one eighth of the work, compensating for the fact that they need to spend less time communicating (since each has only one neighbor). Our technique provides a method of overcoming the non-linear pipeline delays introduced by early-completing processors observed by [2].

For (2), load balancing significantly improved performance. On average, each processor 13.5% of its time waiting for communication. This compares with a 44.3% waiting time for the static protocol. The Left-Right Handoff Protocol performed best for tile sizes between 10 and 50. Smaller tiles had more overhead, and larger tiles result in slower convergence, since a larger fraction of the computation is in the (wider) initial rows. Also note that for processor P to hand off work to its left neighbor, processor P-1, the amount of work transferred is limited to the work finished by P when it receives the checkpoint message from P-1. Thus, if there is a large imbalance, it may take several rounds to achieve the right balance.

For (3), our protocol also performed best for tile sizes between 10 and 40. Convergence occurred more swiftly than in case (2) because a handoffs from a processor to its right neighbor are not limited as in the previous case.

In summary, these preliminary experiments show the Left-Right Handoff Protocol was successful in achieving load balance for uneven or unpredicted loads, while incurring only a modest overhead.

Acknowledgement: We gratefully acknowledge the guidance of Professor Fran Berman in her CSE 260 class.

References

[1] B. S. Siegell and P. Steenkiste, *Automatic generation of parallel programs with dynamic load balancing*, IEEE Symp. on High Performance Dist. Computing (Aug. 1994).

[2] R. F. Van der Wijngaart, S. R.. Sarukkai, and P. Mehra, *The Effect of Interrupts on Software Pipeline Execution on Message-passing Architectures*, International Conference on Supercomputing (May, 1996).

[3] M. J. Wolfe, *High Performance Compilers for Parallel Computing*. Addison-Wesley (1996).

[4] C.-Z. Xu and F. C.M. Lau, *Iterative dynamic load balancing in multicomputers*. J. of Operational Research Society, 45(7):786–796 (1994).

[5] C.-Z. Xu and F. C.M. Lau, *Load Balancing in Parallel Computers*. Kluwer, 1997.