A Load Balancing Strategy for Iterated Parallel Loop Scheduling

Salvatore Orlando[1] and Raffaele Perego[2]

[1] Dip. di Mat. Appl. e Inf., Università di Venezia, e-mail: orlando@unive.it
[2] Istituto CNUCE–CNR, Pisa, e-mail: r.perego@cnuce.cnr.it

Abstract. An efficient template for the implementation on distributed-memory multiprocessors of *iterated parallel loops*, i.e. parallel loops nested in a sequential loop, is presented. The template is explicitly designed to smooth *unbalanced processor workloads* deriving from loops whose iterations are characterized by highly varying execution times. Experiments conducted shows performance gains w.r.t. HPF-like language supports.

1 Introduction

The effective exploitation of parallel loops is the key to achieve considerable speedups in many scientific compute-intensive applications. Data parallel languages such as HPF rely on user-provided directives to derive the distribution of data and on the *owner computes rule* [4] to determine static schedules of parallel loops from the provided data distribution. The choice of the best data distribution (*blocking* or *cyclic*) is currently up to programmers. Note that, for some non–uniform problems, it is impossible to find a good distribution scheme if both a static data distribution and a static scheduling policy are adopted. In fact, one may need a blocking data distribution strategy to exploit locality, but also a cyclic data distribution to balance the workload among the processors.

Our template [3] overcomes this strong limitation of data parallel languages. It relies upon a simple blocking data distribution to exploit locality, and combines static and dynamic policies to obtain well-balanced iteration schedules.

2 The technique

Our template adopts, as far as possible, a static scheduling policy, and introduces dynamic scheduling and data movements among processors only if a load imbalance occurs. Accessed arrays, which may be *aligned* to an abstract topology, are partitioned into P contiguous blocks, and each block is statically assigned to one of the P processors. Iterations operating on each partition are subdivided into *chunks* and assigned to the various processors according to the owner computes rule. Each processor has a local queue Q to store its own chunks. At the beginning, and until a load imbalance has been detected, chunks are statically scheduled by fetching them from Q. The dynamic part of the scheduling policy, which can be classified as a *receiver initiated technique* [2] and starts when a processor understands that queue Q is *becoming empty*, requires both chunks and data to be moved. The main features of our techniques are: **a)** a

processor load evaluation based on local information only. Each processor p_i can estimate its current *workload* simply by counting the chunks still stored in Q; **b)** an effective *prefetching policy* which reduces overheads of chunk migration by overlapping computation and communication. It is driven by a *Threshold* parameter: a processor asks a partner for a new chunk only if its local load is less than *Threshold*. Similarly, the partner process serves the requests only if its own local load is greater than *Threshold*; **c)** a simple local *round-robin* policy to select the next partner to be asked for a further chunk; **d)** an asynchronous *coherence protocol* to migrate updated data from the processor that actually performed the chunk to the processor that is the owner of the corresponding data partition. A *full/empty-like* [1] technique is used to avoid processing invalid data. When processor p_i sends a chunk b_j to p_{i_k}, it sets a flag indicating that the data to be modified by b_j are no longer valid. The next time p_i needs to access the same data, it checks the flag and, if the flag is still set, waits for the updated data from node p_{i_k}. Such control of data coherence does not require barrier synchronizations at the end of each execution of the parallel loop.
As a consequence of the accurate implementation, the template performances are better than or comparable with those obtained with a semantically equivalent CRAFT-Fortran program which adopts a *cyclic* distribution of the data sets. It is worth pointing out that the benchmark we used to validate the technique does not take advantage of data locality deriving from a *blocking* distribution.

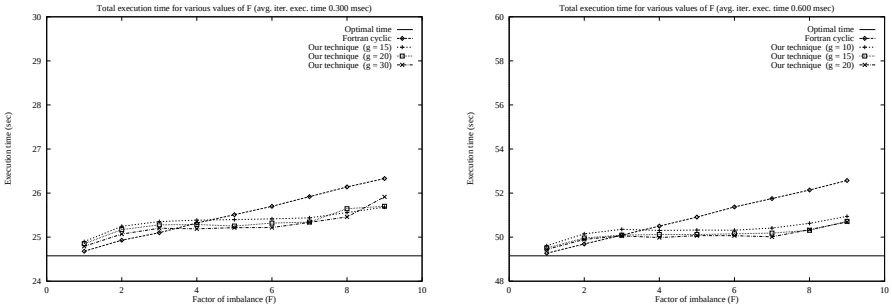

Fig.1. Execution times obtained on a 64 node Cray-T3D for problems characterized by different *factor of imbalance* F. Each curve was obtained by choosing different sizes for chunks. The figure compares the template times with optimal and CRAFT-Fortran execution times. Plots refer to an average iteration execution time of 0.3 and 0.6 *msec*.

References

1. R. Alverson et al. The Tera computer system. In *Proc. of ACM ICS '90*, 1990.
2. V. Kumar et al. Scalable Load Balancing Techniques for Parallel Computers. *Journal of Parallel and Distributed Computing*, 22:60–79, 1994.
3. S. Orlando and R. Perego. A Template for Non–Uniform Parallel Loops Based on Dynamic Scheduling and Prefetching Techniques. In *Proc. of ACM ICS '96*, 1996, to appear.
4. H.S. Zima and B.M. Chapman. Compiling for Distributed-Memory Systems. *Proc. of the IEEE*, pages 264–287, February 1993.

This article was processed using the LaTeX macro package with LLNCS style