A Framework for Dynamic Node-Scheduling of Two-Sided Blocked Matrix Computations

Lars Karlsson and Bo Kågström

Department of Computing Science and HPC2N, Umeå University,
S-901 87 Umeå, Sweden, {larsk,bokg}@cs.umu.se

Abstract. Blocked matrix computations are characterized by a high utilization of floating point units. Limiting factors of distributed computation are communication overhead and spurious synchronization. Load balance is often achieved by a 2D block cyclic layout (BCL). To reduce communication overhead and synchronization, node programs are often rearranged into less obvious, more complex flows. A dynamic approach for scheduling node programs promise to alleviate the down sides of rearrangements while maintaining most of their up sides. We present some new ideas, which, coupled with established ideas, result in a complete dynamic node-scheduling framework design. A two-sided model algorithm with applications in solving the non-symmetric eigenvalue problem is dramatically improved by the framework. We demonstrate that the priority-based scheduling is critical for achieving the improvements.

Key words: Distributed, blocked matrix computations, two-sided algorithms, dynamic scheduling, multicore.

1 Introduction

Blocked matrix algorithms usually utilize a good portion of the machine's peak performance [5]. The main limiting factors on distributed memory (DM) systems, possibly with SMP-style multicore nodes, are communication overhead and spurious synchronization, both between and within nodes. Typically, the performance is increased by changing the execution order and using asynchronous communication.

For one-sided matrix computations like the Cholesky, LU, and QR factorizations it is possible to find efficient static execution orders for 2D block cyclic data layouts (e.g., see [3, 7]). However, two-sided matrix computations have more complex data dependencies and efficient schedules are much harder to find [1]. This complexity can be managed by a dynamic execution approach. In this contribution, we describe the design of an efficient dynamic node-scheduling framework, targeting two-sided blocked matrix computations.

We introduce a two-sided model algorithm which appears as the computationally dominant part in algorithms for computing Schur-like decompositions [2, 8, 6]. A straightforward parallelization yields a non-scalable algorithm on 2D block cyclic data layouts due to a suboptimal execution order rather than load

imbalance. We show that the execution can be guided towards interesting orders, dramatically increasing the practical scalability.

2 Framework Design

Our framework design consists of five major parts:

1. A programming interface to express programs in a familiar sequential style.
2. A representation of tasks and dependencies.
3. A method by which approximate dependencies are deduced from programmer annotations of read and write accesses to matrix blocks.
4. A scheduling mechanism by which the next task to execute is selected from the set of ready tasks.
5. Finally, a method to express asynchronous communication with support for communicating matrix blocks without tags similar to BLACS [4].

The tasks of the node program are fed into the framework in any correct sequential order and are given priorities. The programmer specifies, for each task, the set of matrix blocks it needs to read and/or write. By a simple algorithm, requiring only a constant amount of storage per matrix block, approximate dependencies for the newly added task are deduced in time proportional to the number of matrix blocks accessed by the task.

Once the dependency graph has been built, tasks are scheduled onto the cores of the node by a binary heap priority queue. By assigning priorities to the tasks, the programmer suggests an execution order.

Communication is dynamically reordered by exploiting automatic assignment of tags to messages. We will present an MPI-based implementation of our novel generic communication framework.

3 Two-Sided Model Algorithm

Our two-sided model Algorithm 1 has applications in solving non-symmetric eigenvalue problems. Examples include the chasing procedure of the QR and QZ algorithms [2, 8]. In the model algorithm, f stands for the bulge chasing part

Algorithm 1 ModelAlgorithm(N_b, n_b)

Require: Let $N = N_b n_b$ where n_b is the block size and N_b the number of blocks and assume that n_b is a multiple of 2. The input matrix A has size $N \times N$.

1: **for** $i = 1$ to $N - n_b$ step $\frac{n_b}{2}$ **do**
2: Let $a = i$ and $b = i + n_b - 1$ for ease of notation.
3: $Q = f(A(a:b, a:b))$ *% Compute orthogonal Q*
4: $A(1:a-1, a:b) = A(1:a-1, a:b) * Q$ *% Apply Q from the right*
5: $A(a:b, b+1:N) = Q^T * A(a:b, b+1:N)$ *% Apply Q^T from the left*
6: **end for**

of the QR algorithm. The model algorithm is parallelized for DM by partitioning the complete block row and column updates into smaller block operations. Consider the block row update (line 5 in Algorithm 1). The A submatrix is split vertically at block boundaries. In its local form the update turns to a series of $n_b \times n_b$ block updates. In the cross-border form the blocks are further split horizontally at the block boundary and computed in parallel.

4 A Dual-Wavefront Algorithm

A dual-wavefront algorithm is obtained by setting priorities as a function of the (i, j) coordinates of the top left element in the submatrix of A associated with the task. The priorities are

$$\begin{cases} 0 & \text{for } f\text{-tasks,} \\ i+j & \text{for left updates,} \\ i+j+.5N & \text{for right updates.} \end{cases}$$

The task with the lowest priority executes first. The progress of the algorithm

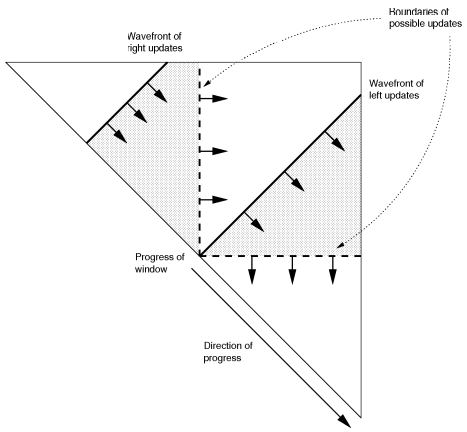

Fig. 1. Two wavefronts corresponding to left and right updates with some separation. The gray areas are the parts of the matrix which will be updated even if no f-task is executed.

and the two wavefronts are sketched in Figure 1 and this pattern has been verified by analyses of real execution traces. The wavefronts are suggested by the priorities and the scheduler softly guides the execution towards this pattern.

5 Results

Figure 2 summarizes our measurements. The three topmost solid curves are from the dual-wavefront algorithm while the three other curves are from the straight-

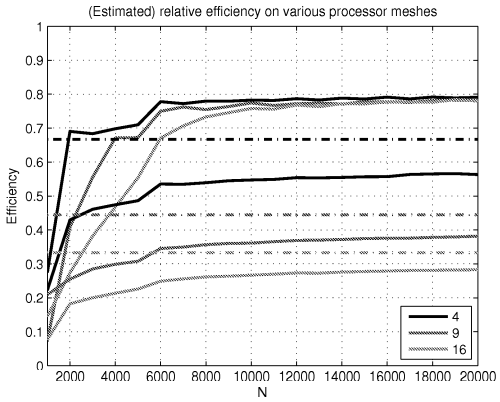

Fig. 2. Comparison between the wavefront and sequential settings.

forward implementation. The dashed curves are theoretical limits on the latter. The dual-wavefront algorithm has clearly better practical scalability compared to the straightforward implementation. The results show that priority-based scheduling is critical for this algorithm.

References

1. B. Adlerborn, B. Kågström, and D. Kressner. Parallel Variants of the Multishift QZ Algorithm with Advanced Deflation Techniques . In B. Kågström et al., editor, *Applied Parallel Computing: State of the Art in Scientific Computing, PARA 2006*, Lecture Notes in Computer Science, LNCS 4699, pages 117–126. Springer, 2007.
2. K. Braman, R. Byers, and R. Mathias. The Multishift QR Algorithm. Part I: Maintaining Well-Focused Shifts and Level 3 Performance. *SIAM Journal on Matrix Analysis and Applications*, 23:929–947, 2001.
3. A. Cleary, J. Dongarra, A. Petitet, and R. C. Whaley. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. `http://www.netlib.org/benchmark/hpl/index.html`, 2004.
4. J. Dongarra and R. C. Whaley. A User's Guide to the BLACS v1.0. Technical Report CS-95-281, University of Tennessee at Knoxville, March 1995. LAPACK Working Note 94.
5. G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.
6. R. Granat, D. Kressner, and B. Kågström. Parallel Eigenvalue Reordering in Real Schur Forms. *Concurrency and Computation: Practice and Experience*, submitted 2007. LAPACK Working Note 192.
7. F. Gustavson, L. Karlsson, and B. Kågström. Distributed SBP Cholesky Factorization Algorithms with Near-Optimal Scheduling. *ACM Transactions on Mathematical Software*, submitted 2007.
8. B. Kågström and D. Kressner. Multishift Variants of the QZ Algorithm with Aggressive Early Deflation. *SIAM Journal on Matrix Analysis and Applications*, 29:199–227, 2006.