Efficient Parallel Implementation of Classical Gram-Schmidt Orthogonalization Using Matrix Multiplication

Takuya Yokozawa, Daisuke Takahashi, Taisuke Boku and Mitsuhisa Sato

Graduate School of Systems and Information Engineering, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573, Japan
yokozawa@hpcs.cs.tsukuba.ac.jp
{daisuke,taisuke,msato}@cs.tsukuba.ac.jp

Extended Abstract

The Gram-Schmidt orthogonalization process is one of the fundamental algorithms in linear algebra that implements the QR decomposition of a matrix into the factorization $A = QR$. Efficient Gram-Schmidt orthogonalization algorithms have been investigated thoroughly [1–3]. Two basic computational variants of the Gram-Schmidt process exist: the classical Gram-Schmidt (CGS) algorithm and the modified Gram-Schmidt (MGS) algorithm [4]. The modified Gram-Schmidt (MGS) algorithm is often selected for practical application because it is much more stable than the CGS algorithm. However, the MGS algorithm cannot be expressed by Level-2 BLAS, and so parallel implementation requires additional communication [5].

On the other hand, the CGS algorithm can be expressed by Level-2 BLAS and is suitable for parallelization. Moreover, the CGS orthogonalization with the DGKS correction [1] is one of the most efficient ways to perform the orthogonalization process.

We present herein an efficient parallel implementation of the CGS orthogonalization using matrix multiplication. The CGS orthogonalization of a matrix can be changed into a matrix multiplication. The CGS orthogonalization can also be extended with matrix multiplication into a recursive formulation. The recursion leads to automatic variable blocking [6].

A recursive CGS algorithm to perform the QR decomposition is shown in Fig. 1. Let the matrix A be denoted as $A = (\boldsymbol{a_1}\boldsymbol{a_2}\cdots\boldsymbol{a_n})$ and the matrix Q be denoted as $Q = (\boldsymbol{q_1}\boldsymbol{q_2}\cdots\boldsymbol{q_n})$. Here, NB, S and $\boldsymbol{w}$ are the blocking size, the work matrix and the work vector, respectively. The function `recursive_CGS`$(A,\ Q,\ 0,\ n)$ performs the orthogonalization process of matrix A. We parallelized the recursive CGS algorithm using a column-wise distribution [3].

In order to evaluate the proposed recursive CGS algorithm, we compared its performance to that of the proposed recursive CGS algorithm and a naive implementation of the CGS algorithm using Level-2 BLAS. The CGS orthogonalization processes were performed on double-precision real data. A 32-node Xeon PC cluster (Irwindale 3 GHz, 12 K uops L1 instruction cache, 16 KB L1

```
recursive_CGS(A, Q, k, m)
  if (m <= NB) then
    q_k = q_k/||q_k||
    do j = k + 1, k + m
      do i = j + 1, j + m
        GEMV(Q^T_{j,i}, a_i, w);
        GEMV(Q_{j,i}, w, q_i);
        q_i = q_i/||q_i||
      end do
    end do
  else
    recursive_CGS(A, Q, k, m/2);
    GEMM(Q^T_{k, k+m/2}, A_{m/2+1,m}, S);
    GEMM(S, Q_{k, k+m/2}, Q_{m/2+1,m});
    recursive_CGS(A, Q, k + m/2, m/2);
  end if
end
```

Fig. 1. Recursive classical Gram-Schmidt algorithm in the QR decomposition

data cache, 2 MB L2 cache, 1 GB DDR2-400 SDRAM main memory per node, Linux 2.6.16-1smp) was used. The nodes on the PC cluster are interconnected through a 1000Base-T Gigabit Ethernet switch. LAM/MPI 7.1.1 was used as a communication library., and Intel MKL 8.1 was used as a BLAS library. The compiler used was gcc 4.0.2, and the optimization option was specified as "`-O3`". All programs were run in 64-bit mode.

For $n = 10000$, the proposed recursive CGS algorithm runs approximately 1.37 times faster than the naive implementation of the CGS algorithm using Level-2 BLAS. As a result of cache blocking, the performance of the proposed recursive CGS algorithm remains high, even for the larger problem size.

Note that on a 32-node Xeon 3.0 GHz PC cluster, a performance of over 55 GFLOPS was realized for a size of $n = 40000$.

References

1. Daniel, J., Gragg, W.B., Kaufman, L., Stewart, G.W.: Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. Math. Comput. **30** (1976) 772–795
2. Vanderstraeten, D.: A parallel block Gram-Schmidt algorithm with controlled loss of orthogonality. In: Proc. Ninth SIAM Conference on Parallel Processing for Scientific Computing. (1999)
3. Katagiri, T.: Performance evaluation of parallel Gram-Schmidt re-orthogonalization methods. In: Proc. 5th International Meeting on High Performance Computing for Computational Science (VECPAR 2002). Volume 2565 of Lecture Notes in Computer Science., Springer-Verlag (2003) 302–314
4. Björck, Å.: Numerical Methods for Least Squares Problems. SIAM Press, Philadelphia, PA (1996)
5. Dongarra, J.J., Duff, I.S., Sorensen, D.C., van der Vorst, H.A.: Numerical Linear Algebra for High-Performance Computers. SIAM Press, Philadelphia, PA (1998)
6. Elmroth, E., Gustavson, F.G.: Applying recursion to serial and parallel QR factorization leads to better performance. IBM J. Res. Develop. **44** (2000) 605–624