# Parallel Implementation of a Recursive Blocked Algorithm for Classical Gram-Schmidt Orthogonalization

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

## 1 Introduction

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 parallel algorithms have been investigated thoroughly [1–4]. Two basic computational variants of the Gram-Schmidt process exist: the classical Gram-Schmidt (CGS) algorithm and the modified Gram-Schmidt (MGS) algorithm. The 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 communications [2].

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 [5] and the Iterated-CGS (ICGS) method [3] are more efficient ways to perform the orthogonalization process.

In this paper, we propose a parallel implementation of a recursive blocked algorithm for classical Gram-Schmidt orthogonalization.

## 2 Classical Gram-Schmidt Orthogonalization

Let $A = (\boldsymbol{a_1 a_2} \cdots \boldsymbol{a_n})$ be a set of $n$ vectors of size $m$. The classical Gram-Schmidt orthogonalization algorithm for the matrix $A$ is shown in Fig. 1. Here, $(\boldsymbol{q_i}, \boldsymbol{a_j})$ denotes the inner product of $\boldsymbol{q_i}$ and $\boldsymbol{a_j}$, and $||\boldsymbol{q_j}||$ denotes the Euclidean norm of $\boldsymbol{q_j}$. If the orthogonalization of $\boldsymbol{q_1}, \boldsymbol{q_2}, \cdots, \boldsymbol{q_{j-1}}$ has already computed, the inner products with $\boldsymbol{a_j}, \boldsymbol{a_{j+1}}, \cdots, \boldsymbol{a_n}$ has no dependency. Thus, in the computation of $\boldsymbol{q_1}, \boldsymbol{q_2}, \cdots, \boldsymbol{q_n}$, we can perform the inner product part and the vector operation part independently.

The computation for multiple inner products can be performed with a matrix multiplication, and the computation for multiple vector operations can be also performed with a matrix-vector multiplication.

```
do $j = 1, n$
    $\boldsymbol{q_j} = \boldsymbol{a_j}$
    do $i = 1, j-1$
        $\boldsymbol{q_j} = \boldsymbol{q_j} - (\boldsymbol{q_i}, \boldsymbol{a_j})\boldsymbol{q_i}$
    end do
    $\boldsymbol{q_j} = \boldsymbol{q_j} / ||\boldsymbol{q_j}||$
end do
```

**Fig. 1.** Classical Gram-Schmidt orthogonalization algorithm

```
begin RBCGS$(A, Q, n, s, h)$
    if $(h <= \mathrm{NB})$ then
        $\boldsymbol{q_s} = \boldsymbol{q_s} / ||\boldsymbol{q_s}||$
        do $i = s+1, s+h$
            $\boldsymbol{w} = Q^t_{s,\,i}\,\boldsymbol{a_i}$
            $\boldsymbol{q_i} = \boldsymbol{q_i} - Q_{s,\,i}\,\boldsymbol{w}$
            $\boldsymbol{q_i} = \boldsymbol{q_i} / ||\boldsymbol{q_i}||$
        end do
    else
        RBCGS$(A, Q, n, s, h/2)$;
        $S = Q^t_{s,\,s+h/2-1} A_{s,\,s+h/2-1}$
        $Q_{s+h/2,\,s+h-1} = Q_{s+h/2,\,s+h-1} - Q_{s,\,s+h/2-1} S$
        RBCGS$(A, Q, n, s+h/2, h/2)$;
    end if
end
```

**Fig. 2.** Recursive blocked classical Gram-Schmidt algorithm

## 2.1 Recursive Blocked CGS Algorithm

The CGS orthogonalization using matrix multiplication can be extended into a recursive formulation. The recursion leads to automatic variable blocking [6].

A recursive blocked CGS (RBCGS) algorithm is shown in Fig. 2. Here, NB, $S$ and $\boldsymbol{w}$ are the blocking size, the work matrix and the work vector, respectively. Note that $Q_{i,j}$ denotes $(\boldsymbol{q_i}\,\boldsymbol{q_{i+1}} \cdots \boldsymbol{q_j})$. The function RBCGS($A$, $Q$, $n$, 1, NB) performs the orthogonalization process of matrix $A$. The computational space of the RBCGS orthogonalization is shown in Fig. 3. The RBCGS orthogonalization is performed in order of I to VII.

## 2.2 Parallelization

We parallelized the recursive CGS algorithm using a row-wise distribution. In the row-wise distribution, the whole elements of the vector $\boldsymbol{a_j}$ and the vectors $\boldsymbol{q_1}, \boldsymbol{q_2}, \cdots, \boldsymbol{q_{j-1}}$ are distributed to each processor.

In the row-wise distribution, "Global sum" operations are necessary to compute the inner product of $\boldsymbol{q_i}$ and $\boldsymbol{a_j}$.

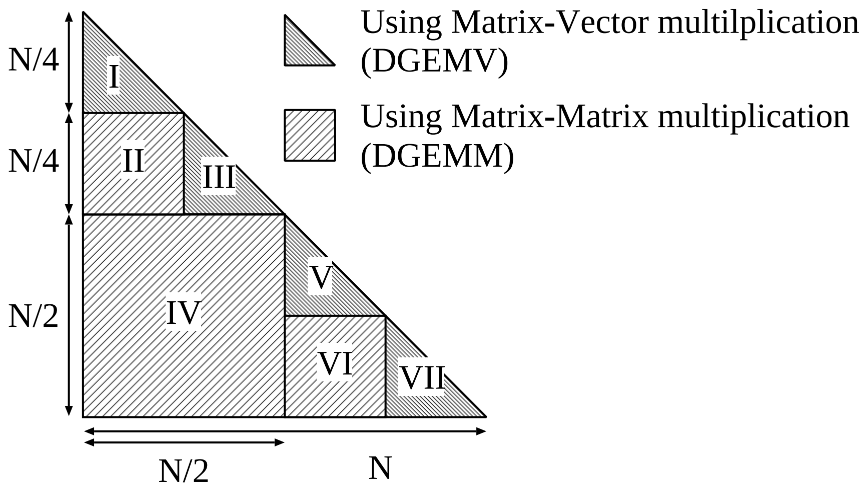
Using Matrix-Vector multilplication (DGEMV)
Using Matrix-Matrix multiplication (DGEMM)
N/4
N/4
N/2
I
II
III
IV
V
VI
VII
N/2
N

**Fig. 3.** Computational space of recursive blocked classical Gram-Schmidt orthogonalization

## 3 Experimental Results

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 data cache, 2 MB L2 cache, 1 GB DDR2-400 SDRAM main memory per node, Linux 2.6.20-1smp) was used. The nodes on the PC cluster are interconnected through a 1000Base-T Gigabit Ethernet switch. OpenMPI 1.2.4 was used as a communication library, and Goto BLAS r1.19 was used as a BLAS library. The compiler used was Intel C Compiler 10.0, and the optimization option was specified as "`-O3 -xP`". All programs were run in 64-bit mode.

Fig. 4 compares the proposed recursive blocked CGS (RBCGS) algorithm and naive implementation of CGS in terms of their GFLOPS. For $n = 40000$, the proposed recursive blocked CGS algorithm runs approximately 5.36 times faster than the naive implementation of the CGS algorithm using Level-2 BLAS. As a result of multi-level blocking, the performance of the proposed recursive blocked CGS algorithm remains high, even for the larger problem size.

Note that on a 32-node Xeon 3 GHz PC cluster, a performance of over 122 GFLOPS was realized for a size of $n = 40000$.

## 4 Conclusion

In this paper, we propose the parallel implementation of a recursive blocked algorithm for classical Gram-Schmidt orthogonalization. The CGS orthogonal-

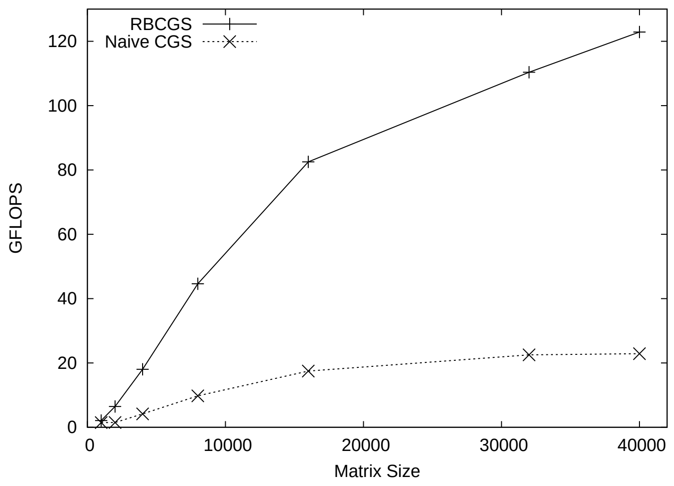

**Fig. 4.** Performance comparison of parallel implementation of recursive blocked CGS and naive implementation using Level-2 BLAS on 32-node Xeon 3 GHz PC cluster

ization using matrix multiplication can be extended into a recursive formulation. We showed that a multi-level blocking of the CGS orthogonalization improves performance effectively. We parallelized the recursive CGS algorithm using a row-wise distribution. The experimental results show the proposed recursive blocked CGS algorithm runs approximately 5.36 times faster than the naive implementation of the CGS algorithm using Level-2 BLAS for $n = 40000$ on a 32-node Xeon 3 GHz PC cluster.

## References

1. 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)
2. Vanderstraeten, D.: A stable and efficient parallel block Gram-Schmidt algorithm. In: Proc. 5th International Euro-Par Conference (Euro-Par 1999). Volume 1685 of Lecture Notes in Computer Science., Springer-Verlag (1999) 1128–1135
3. Lingen, F.J.: Efficient Gram-Schmidt orthonormalisation on parallel computers. Communications in Numerical Methods in Engineering **16** (2000) 57–66
4. 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
5. 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
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