Automatic Generation of Full-Set Batched BLAS

Yusuke Hirota[1] (yusuke.hirota@riken.jp)
Daichi Mukunoki[1,2] (daichi.mukunoki@riken.jp)
Toshiyuki Imamura[1] (imamura.toshiyuki@riken.jp)
1) RIKEN Advanced Institute for Computational Science
2) Tokyo Woman's Christian University

1. Introduction

- **Batched Basic Linear Algebra Subroutine (batched BLAS)**: a new BLAS interface which computes multiple independent BLAS operations as a single subroutine [1]
- On many-core processors, a small size problem may not utilize the computation power of all the cores. Batched BLAS is a solution to utilize many cores effectively
- Some of high-demanded batched BLAS routines (mostly level-3 operations) have been implemented for CPU/XeonPhi [2] and GPUs [3][4][5], but a full set of the BLAS routines (including level-1/2/3 routines) has not been provided yet
- In this study, we propose **an efficient development method to develop a full set of batched BLAS routines using automatic code generation** with some existing standard BLAS implementation such as Intel MKL
- This is **the first implementation of the level 1-2-3 full-set variable size Batched BLAS (vbatched, Intel MKL style)** as far as we know

Interfaces of standard & batched DGEMM

Argument	Description	Standard BLAS	Batched BLAS: NVIDIA cuBLAS	Batched BLAS: MAGMA BLAS Batched	Batched BLAS: MAGMA BLAS VBatched	Batched BLAS: Intel MKL
HANDLE	context handler	--	cublasHandle_t	--	--	--
TRANSA	op (A)	char	char	char	char*	char*
TRANSB	op (B)	char	char	char	char*	char*
M	Rows of op(A)/C	int	int	int	int*	int*
N	Columns of op(B)/C	int	int	int	int*	int*
K	Columns of op(A)/rows of op(B)	int	int	int	int*	int*
ALPHA	alpha	double	double*	double	double*	double*
A	input matrix A	double*	double**	double**	double**	double**
LDA	leading dimension of A	int	int	int	int*	int*
B	input matrix B	double*	double**	double**	double**	double**
LDB	Leading dimension of B	int	int	int	int*	int*
BETA	beta	double	double*	double	double*	double*
C	input/output matrix C	double*	double**	double**	double**	double**
LDC	leading dimension of C	int	int	int	int*	int*
BATCHCOUNT	number of operations	--	--	int	int	--
QUEUE	queue to execute in	--	--	magma_queue_t	--	--
BATCH_OPTS	batched style (fixed or variable)	--	--	--	enum	--
INFO	error handling	--	--	--	int*	--
GROUP_COUNT	number of groups	--	--	--	--	int
GROUP_SIZES	number of operations per group	--	--	--	--	int*

*This table was created referencing [2] and [3]

Our current implementation supports Intel MKL style variable size batched interface

2. Implementation

- Batched BLAS source files are generated by our automatic code generator implemented in Python based on (1) routine definition, (2) cost definition, and (3) scheduling template files
  - ➢ Batched BLAS API (subroutine name, arguments, etc.) can be modified easily by modifying the BLAS routine definition file
  - ➢ Scheduling strategy for batched tasks can be modified depending on the target architecture by modifying the scheduling template
- Our current implementation was generated from Intel MKL's standard BLAS implementation and supports Intel MKL style variable size batched interface

● Cost definition and scheduling

- Cost of each BLAS call is estimated by its number of FLOPs
- BLAS operations are allocated to threads by **a greedy scheduling** (see the below figure)

1. Evaluate the cost of all BLAS operations
2. Allocate the BLAS operation which has the largest cost in unassigned ones to a thread whose total cost is smallest in all threads
3. Repeat 2. until all BLAS operations are assigned

● Automatic code generation

```
void,cblas_dgemm
CBLAS_LAYOUT,layout,a,CBLAS_TRANSPOSE,transa,g,CBLAS_TRANSPOSE,transb,g,int,m,g,int,n,g,int,k,g,double,alpha,g, double *,a,l,int,lda,g,double *,b,l,int,ldb,g,double,beta,g,double *,c,l,int,ldc,g
get_cost_n1n2n3,m,n,k
...
```

BLAS routine definition
batched_blas_data.csv

BLAS cost definition
batched_blas_cost.c

Scheduling template
batched_blas_schedule.c

IN →

Code generation script (Python)
$ python batched_blas.py batched_blas_data.csv

OUT →

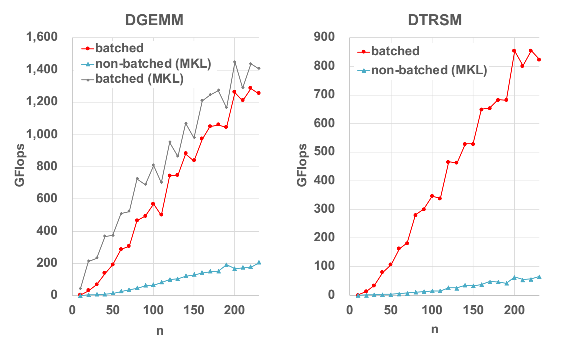
Batched BLAS source files
- Makefile
- cblas_caxpy_batch.c
- cblas_ccopy_batch.c
- cblas_dgemm_batch.c
- ...

```
void cblas_dgemm_batch(const CBLAS_LAYOUT layout,
        const CBLAS_TRANSPOSE* transa, const CBLAS_TRANSPOSE* transb,
        const int* m, const int* n, const int* k, const double* alpha, const double ** a, const int* lda,
        const double ** b, const int* ldb, const double* beta, double ** c, const int* ldc,
        const int group_count, const int *group_size)
```

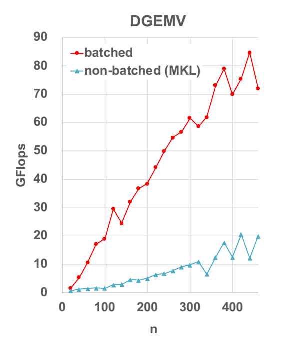
3. Performance Evaluation

- We compared the performance of our batched BLAS routines generated from Intel MKL using our method with non-batched MKL routines (Intel MKL 17.0.2)
- Target platform: **Intel Xeon Phi 7210** (Knights Landing, 1.3GHz, 64 cores, 64 threads), MCDRAM was used in flat-mode (numactrl --membind=1)
- Batch count: 1000, group count: 1, problem size: m=n=k (GEMM & TRSM), m=n (GEMV), n (AXPY & DOT)
- Scalar values (alpha & beta) are randomly generated but constant within a group
- Leading dimensions are randomly decided (e.g. m <= lda <= 1.5*m) but constant within a group
- Matrices for batched computations are allocated on memory sequentially

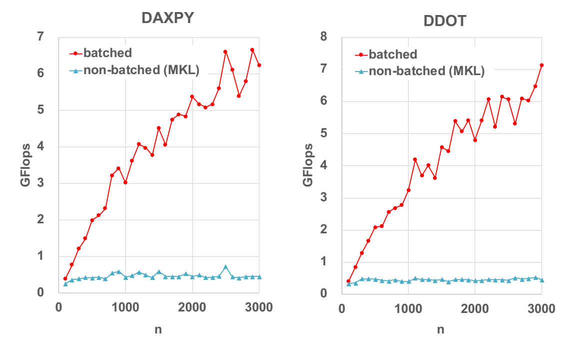
Level-3 routine



Level-2 routine



Level-1 routine



4. Conclusion and Future Work

- The first implementation of the level 1-2-3 full-set variable size Batched (vbatched) BLAS
- An efficient development method to generate a full set of batched BLAS routines using automatic code generation with some existing standard BLAS implementation
- Our evaluation demonstrated that the auto-generated batched BLAS routines achieved competitive performance with standard BLAS
- Our results suggest that such an automatic generation would be an effective method to develop batched BLAS routines for future architectures
- There is still plenty of room for improvement in batch scheduling
- We plan to utilize this study for helping the development of Batched BLAS on our next generation supercomputers

References:
[1] J. Dongarra et al., "The Design and Performance of Batched BLAS on Modern High-Performance Computing Systems", ICCS2017, 2017
[2] Intel MKL Team, "Compact Batched BLAS", http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/Batched-BLAS-2017/talk17-costa.pdf, 2017
[3] A. Abdelfattah, "Performance, Design, and Autotuning of Batched GEMM for GPUs", ISC2016, 2016
[4] University of Tennessee, "MAGMA", http://icl.eecs.utk.edu/magma/
[5] NVIDIA, "CUBLAS LIBRARY User Guide", DU-06702-001_v9.1, 2018, http://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf

Acknowledgement:
This study is supported by the FLAGSHIP 2020 project.