High Performance Design of Batched Tensor Computations: Performance Analysis, Modeling, Tuning and Optimization

Azzam Haidar
Ahmad Abdelfattah, Jack Dongarra, Stan Tomov

MAGMA team @ ICL . UTK . EDU

MAGMA: Batched, Tensor, Deep Learning, Embedded, LA
Some of my thoughts and observations

- Reproducibility and reliability
- Design: library v.s. paper
- Design: standard v.s. interleaved
- Methodology, Performance Model and Performance Counter Analysis
- Small sizes results
Reproducibility and reliability

• Working on MAGMA customized kernel for Deep Learning and tensor contraction
• How a 2X speedup can be faster than 10X
• Some of my observations on benchmark reporting
• I am not going to talk about accuracy reproducibility (Jim cover it) but rather I am going to talk benchmark reproducibility
Reproducibility and reliability

![Graph showing performance trend](image)

```
initialize (A, B, C);
start_timer
call mydgemm_batched
end_timer
```
Reproducibility and reliability

- Intel Xeon E5-2650 v3 (Haswell) 20 cores

- initialize \((A, B, C)\);
- start_timer
- call mydgemm_batched
- end_timer

```
initialize (A, B, C);
start_timer
call mydgemm_batched
end_timer
```
Reproducibility and reliability

Intel Xeon E5-2650 v3 (Haswell) 20 cores

initialize (A, B, C);
flush cache large data
start_timer
call mydgemm_batched
end_timer
Reproducibility and reliability

Matrix Size

Gflop/s

Intel Xeon E5-2650 v3 (Haswell) 20 cores

initialize \((A, B, C)\);
flush cache large data
start_timer
for \((i=0; i<\text{maxiter}; i++)\)
    call mydgemm\_batched
end_timer
Reproducibility and reliability

Intel Xeon Phi 7250 KNL 68 cores

Gflop/s

Matrix Size

initialize (A, B, C);
flush cache large data
start_timer
for (i=0; i<maxiter; i++)
call mydgemm_batched
end_timer
Reproducibility and reliability

The SCALAPACK SVD story
LAPACK: it performs an QR(A) then SVD on $R=U\Sigma V^T$ then $U=Q^*U$
Unfortunately Scalapack do not perform this and so comparing
tall-skinny SVD against Scalapack is always a win-big.
However a very simple 3 lines of codes can fix the issue
pdgeqrf
pdgesvd
pdormqr
Some of my thoughts and observations

- Reproducibility and reliability
- **Design: library v.s. paper**
- Design: standard v.s. interleaved
- Methodology, Performance Model and Performance Counter Analysis
- Small sizes results
Design: library v.s. paper

- Library need to follow standard interface
  - Which might add many overhead in particular for small matrices
- Library have to be used by developers, applications
- Library have to be generic and accommodate at least most practical cases
- A library (.a .so) cannot be 1 GB, so template instantiation should be limited
- Library have to reliable and robust
- Maybe reproducible accuracy
Design: library v.s. paper

• A library should check argument error
• A Library should also check for numerical error and need to be conform with the standard (e.g., Cholesky, LU, and QR need to check for: diag error, singularity, overflow, underflow etc.)
• Papers, posters, proposals:
  • I have seen code without any checking,
  • Even sometimes without accuracy verification, or with self made error checking
  • Most of the time compiled for every size for every run,
  • Work only on a very particular case
  • That’s fine when dealing with particular application but cannot be adopted in a library
Design: library v.s. paper

Matrix Size

Nvidia P100

- LU instantiation for every size
- LU generic any size

Gflop/s $2n^3/3$
Design: library v.s. paper

Matrix Size

Nvidia P100

- LU instantiation for every size
- LU generic any size
- what we would like to have
Design: library v.s. paper

Nvidia P100

- Magma tensor dgemm predefined size at compile time
- Magma batched dgemm generic small
- cuBLAS v8.0

Matrix Size

Gflop/s

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
Some of my thoughts and observations

- Reproducibility and reliability
- Design: library v.s. paper
- Design: standard v.s. interleaved
- Methodology, Performance Model and Performance Counter Analysis
- Small sizes results
Design: standard v.s. interleaved

• Interleaved is good but:
  • gemm might work
  • Cholesky, is easy to implement but when the matrix is larger than 16/32 the matrices might not fit into the reg/sm of the SMX, and thus the matrix is going to be reloaded at every update.
  • LU what is going to happen when every matrix has different pivot
  • How to handle variable sizes (gemm, lu, QR LA)
  • Iterative solvers working on different set of batched matrices might converge while the other set still iterating.
  • Multifrontal solver where some data is runtime created and fill in occur

• Standard format showed very good performance and efficiency, but sure effort from analyzing, to modeling, to design to tuning is needed to reach this
Some of my thoughts and observations

- Reproducibility and reliability
- Design: library v.s. paper
- Design: standard v.s. interleaved
- Methodology, Performance Model and Performance Counter Analysis
- Small sizes results
GPU Optimization Summary

- **Hardware concepts**
  - CUDA core
  - Warp
  - Half-warp
  - Register file
  - Shared memory
  - Atomics
  - Shuffles
  - SMX

- **Software concepts**
  - Stream
  - Thread block
  - Kernel
  - Inlining
  - Intrinsics

- **Algorithmic concepts**
  - Blocking
  - Recursive blocking
  - Kernel replacement
  - Out-of-place operations
Batched Computations

Classical strategies design

- For large problems the strategy is to prioritize the data-intensive operations to be executed by the accelerator and keep the memory-bound ones for the CPUs since the hierarchical caches are more appropriate to handle it.

Challenges

- **Cannot be used** here since matrices are very small and communication becomes expensive.

Proposition

- Develop a GPU-only implementation.
Batched Computations

Classical strategies design

• For large problems performance is driven by the Level 3 BLAS (GEMM)

Challenges

• For batched small matrices it is more complicated

Proposition

• Rethink and Redesign both phases in a tuned efficient way
Batched Computations

Key observations and current situation:

Classical strategies design

- A recommended way of writing efficient GPU kernels is to use the whole GPU’s shared memory, registers/TB – load it with data and reuse that data in computations as much as possible.

Challenges

- Our study and experience shows that this procedure provides very good performance for classical GPU kernels but is not that appealing for batched algorithm for different reasons.
Batched Computations

Challenges

• Completely **saturating the shared memory** per SMX can decrease the performance of memory bound operations, since only one thread-block will be mapped to that SMX at a time (**low occupancy**)

• Due to the **limited parallelism** in the small matrices, the number of threads used in the thread block will be limited, resulting in **low occupancy**, and subsequently poor core utilization

• **Shared memory is small** (48KB/SMX) to fit the whole panel

• The panel involves **Non-GPU friendly operations**:
  • Vectors column (find the max, scale, norm, reduction)
  • Row interchanges (swap)
  • Small number of vectors (apply)

Proposition: **custom design per operations type**
Batched Computations

Performance metrics analysis

- A recommended way of writing efficient GPU kernels is to use the whole GPU’s shared memory, registers/TB – load it with data and reuse that data in computations as much as possible.

![Graph showing Gflop/s vs. Values of m for different batch sizes (nb=2, nb=4, nb=8, nb=10).](image)

- Optimized kernel
- Using sm/rg
- Left v.s. right looking
- Autotuned
Batched Computations

Performance metrics analysis

- A recommended way of writing efficient GPU kernels is to use the whole GPU’s shared memory, registers/TB – load it with data and reuse that data in computations as much as possible.
Batched Computations

Performance metrics analysis

- A recommended way of writing efficient GPU kernels is to use the whole GPU’s shared memory, registers/TB – load it with data and reuse that data in computations as much as possible.

We should focus on the performance analysis and the design of a kernel

- optimized kernel
- using shared memory
- left v.s. right looking
- autotuned
Batched Computations

Performance metrics analysis

- A recommended way of writing efficient GPU kernels is to use the whole GPU’s shared memory, registers/TB – load it with data and reuse that data in computations as much as possible.

![Graphs showing performance metrics for batched dpotrf kernels](image-url)

fixed size batched dpotrf (kernel-1), batchCount = 3000, 1 K40c GPU

fixed size batched dpotrf (kernel-2), batchCount = 3000, 1 K40c GPU
Methodology, Performance Model and Performance Counter Analysis

\[ P_{\text{max}} = \frac{F}{T_{\text{min}}} \]

- Flops for the computation
- Fastest time to solution

For square matrices

\[ F \approx 2n^3, \quad T_{\text{min}} = \min_T (T_{\text{Read}(A,B,C)} + T_{\text{Compute}(C)} + T_{\text{Write}(C)}) \]

- Need to read/write \(4n^2\) elements, i.e., \(32n^2\) Bytes in DP
  => if max bandwidth is \(B\), we can take \(T_{\text{min}} = 32n^2 / B\) in DP. Thus,

\[ P_{\text{max}} = \frac{2n^3 B}{32n^2} = \frac{nB}{16} \text{ in DP.} \]

- With ECC on, peak on B on a K40c is \(\approx 180\) GB/s, so when \(n=16\) for example, we expect theoretical max performance of \(180\) Gflop/s in DP
Methodology, Performance Model and Performance Counter Analysis

- **Achieved Occupancy**
  - Matrix Size: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
  - Occupancy values for Nvidia K40, Our design, MAGMA K40, Cublas K40, and Rocache design

- **Global Memory Load Efficiency (%)**
  - Matrix Size: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
  - Efficiency values for Nvidia K40, Our design, MAGMA K40, Cublas K40, and Rocache design

- **Shared Memory Load Throughput TB/s**
  - Matrix Size: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
  - Throughput values for Nvidia K40, Our design, MAGMA K40, Cublas K40, and Rocache design

- **Gflop/s**
  - Matrix Size: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
  - Gflop/s values for Nvidia K40 / Intel Xeon E5-2650 v3 (Haswell) 10 cores, Our design, MAGMA K40, Cublas K40, Rocache design, MKL+openMP on CPU, and Roofline bound
Methodology, Performance Model and Performance Counter Analysis
Methodology, Performance Model and Performance Counter Analysis

Nvidia P100

Matrix Size

Gflop/s

magma  - red

cublas  - blue

mkl+openMP - black

upper bound

Tegra ARM

Matrix Size

Gflop/s

magma  - blue

openblas - red

ijk_loop - purple

ikj_loop - gray

upper bound
Methodology, Performance Model and Performance Counter Analysis

Nvidia P100 batchcount 40K

Magma Cholesky DPOTRF

Matrix Size

Gflop/s $n^3/3$
Methodology, Performance Model and Performance Counter Analysis

Nvidia P100 batchcount 40K

Matrix Size

Gflop/s $n^3/3$

Magma LU DGETRF
Methodology, Performance Model and Performance Counter Analysis

Nvidia P100 batchcount 40K

Gflop/s $2n^3$

- Magma matrix inversion DGETRI (LU+solve)
- Gauss Jordan Inversion

Matrix Size

ICL
THE UNIVERSITY OF TENNESSEE KNOXVILLE
Future trending direction

- Extended functionality and variable sizes
- Customized batched routines for Deep Learning
- FP16 batched routines coming soon
- Sparse components SpDMM, SpMM, SpMV, etc
- Introducing interleaved format
- More Applications specific design
- MAGMA Embedded
- MAGMA DL

- I would encourage a framework for accuracy and performance benchmarking
Collaborators and Support

MAGMA team
http://icl.cs.utk.edu/magma

PLASMA team
http://icl.cs.utk.edu/plasma

Collaborating partners
University of Tennessee, Knoxville
Lawrence Livermore National Laboratory, Livermore, CA
University of California, Berkeley
University of Colorado, Denver
INRIA, France (StarPU team)
KAUST, Saudi Arabia