High-Performance Batched Computations

Innovative Computing Laboratory
University of Tennessee, Knoxville

In collaboration with:
LLNL, Livermore, CA, USA
University of Manchester, Manchester, UK
University of Paris-Sud, France

May 18–19, 2016

Azzam Haidar
Outline

• Motivation

• Current approaches and challenges

• MAGMA Batched computations
  – Algorithmic basics
  – Design and optimizations for batched computations
  – LU, QR, and Cholesky
  – Performance results
  – Energy efficiency

• MAGMA Batched computations for variable sizes

• Future direction
MAGMA Batched Computations
MAGMA Batched Computations

```c
main (int argc, char *argv[]) {
    int nthreads, tid;

    /* Fork a team of threads giving them their own copies of variables */
    #pragma omp parallel private(nthreads)
    { /* All threads join master thread and disband */
        printf("Batched computation from thread = %d\n", tid);
        nthreads = omp_get_num_threads();
        tid = omp_get_thread_num();
        printf("Number of threads = %d\n", nthreads);
        printf("Obtain thread number\n");
    }

    if (tid == 0)
        printf("Only master thread does this\n");

    #pragma omp parallel private(nthreads, tid)
    { /* Fork a team of threads giving them their own copies of variables */
        printf("Obtain thread number\n");
        in_threads, tid;
    }

    int main (int argc, char *argv[])
```

Batched computation
MAGMA Batched Computations

We present here a feasibility design study, the idea is to target the new high-end technologies.

Our goals:

• Develop a high-performance numerical library for batched linear algebra subroutines tuned for performance and energy efficiency on modern processor architectures, CPU, GPU, Phi

• Define modular interfaces that allow code replacement techniques [to provide the developers of applications, compilers, and runtime systems with the option of expressing new, application-specific batched computations ]

• Propose template design and code auto generation for performance portability
MAGMA Batched Computations

We present here a feasibility design study, the idea is to target the new high-end technologies.

Key observations and current situation:

• There is a **lack of HPC linear algebra software for small problems especially for GPU**

• **CPU**: this can be done easily using existing software infrastructure

  1. naïve way: what we call Non-batched design

  2. **better way**: Batched design
MAGMA Batched Computations CPU

1. Non-batched computation
   loop over the matrices one by one and compute either:

   - sequentially wasting all the other cores, and attaining very poor performance
   - Or using multithread (note that for small matrices there is not enough work for all cores so expect low efficiency as well as threads contention can affect the performance)

   for (i=0; i<batchcount; i++)
2. Batched computation

Loop over the matrices and assign a matrix to each core working on it sequentially and independently.

- Since matrices are very small, all the n_cores matrices will fit into L2 cache thus we do not increase L2 cache misses while performing in parallel n_cores computations reaching the best of each core.
MAGMA Batched Computations

We present here a feasibility design study, the idea is to target the new high-end technologies.

Key observations and current situation:

• There is a lack of HPC linear algebra software for small problems especially for GPU

• CPU: this can be done easily using existing software infrastructure

• MIC: similarly to CPU but, however, today it requires optimizing BLAS routines
MAGMA Batched Computations MIC

Programming model: auto-generation and auto-tuning:

- Many parameter need to be considered, such as:
  - Number of registers, cache size, vectorization AVX2, preferctching, intrinsic versus simple loop, etc...
  - Data Access Optimizations and Loop Transformation Techniques
  - Register Data Reuse and Locality
  - Effect of the Multi-threading
  - Effect of the NUMA-socket and Memory Location

Simply

Need performance analysis (theoretical and tools)
Many code designs, and each may need tuning
MAGMA Batched Computations MIC

Programming model: auto-generation and auto-tuning:

```c
void matmulblk_v3(int M, int N, int K,
    double alpha, const double* restrict A, int lda,
    const double* restrict B, int ldb,
    double beta, double* C, int ldc,
    int localtid, int threadsgroup)
{
    int y, i, j, k, l;
    int Ecol, col, row;
    //local matrix pointers specific to thread
    double myA, myB, myC, myW, myV;
    double myW, myC, myV;
    int rowstart, colstart, rowstride, colstride;
    __m512d va, vb, vc, v1, v2, v3, v4, v5, v6, v7;
    __m512d vy, v1, v2, v3, v4, v5, v6, v7;
    rowstart = localtid % 2;
    colstart = localtid / 2;
    rowstride = threadsgroup % 1 ? 1 : 2;
    colstride = threadsgroup % 1 ? 1 : threadsgroup / 2;
    for(Ecol=colstart;Ecol<N;Ecol=BLK_K+colstride)
    { myB=v0+Ecol;
      myC=0+col;
      for(rorow=rowstart:BLK_M;row<row+BLK_M;rowstr)
      {
        myA=0+row;
        cc=myC+row;
        #pragma unroll
        for(int l=0; l < BLK_K; l++)
            __m512d va;
        #pragma unroll
        for(int m=0; m < BLK_N; m++)
            __m512d vy;
        //initialize the 6x6 BLK of product Matrix VY
        vy0=0+0+0+0;
        vy1=0+0+0+0;
        vy2=0+0+0+0;
        vy3=0+0+0+0;
        vy4=0+0+0+0;
        vy5=0+0+0+0;
        vy6=0+0+0+0;
        vy7=0+0+0+0;
        #pragma unroll
        for(c0=col;c0<K; c0=BLK_K)
        {
            aa = myA+4+col;
            bb = myB+col;
            va = __m512d load_p4(vy0+0+0+0);
            vb = __m512d load_p4(vy1+0+0+0);
            vc = __m512d load_p4(vy2+0+0+0);
            v1 = __m512d load_p4(vy3+0+0+0);
            v2 = __m512d load_p4(vy4+0+0+0);
            v3 = __m512d load_p4(vy5+0+0+0);
            v4 = __m512d load_p4(vy6+0+0+0);
            v5 = __m512d load_p4(vy7+0+0+0);
            #pragma unroll
            for(int i=0; i < BLK_K; i++)
                __m512d va;
            #pragma unroll
            for(int i=0; i < BLK_N; i++)
                __m512d vy;
        }
    }
}
```

MAGMA Batched Computations  MIC

Programming model: auto-generation and auto-tuning:
MAGMA Batched Computations MIC

Programming model: auto-generation and auto-tuning:
MAGMA Batched Computations MIC

Programming model: auto-generation and auto-tuning:
We present here a feasibility design study, the idea is to target the new high-end technologies.

Key observations and current situation:

- There is a lack of HPC linear algebra software for small problems especially for GPU
- **CPU**: this can be done easily using existing software infrastructure
- **MIC**: Similarly to CPU but, today it requires optimizing BLAS routines
- **GPU**: are efficient for large data parallel computations, and therefore have often been used in combination with CPUs, where the CPU handles the memory bound and difficult tasks to be parallelized while the GPU is used for data intensive tasks

- **What programming model is best for small problems?**
Algorithmic basics

- **Linear solver** \(Ax=b\) follow the LAPACK-style algorithmic design

- Two distinctive phases
  - panel factorization: latency-bound workload
  - trailing matrix update: compute-bound operation

---

References:

1. A. Haidar, S. Tomov, P. Luszczek, and J. Dongarra. 
   *MAGMA Embedded: Towards a Dense Linear Algebra Library for Energy Efficient Extreme Computing*
   IEEE High Performance Extreme Computing Conference IEEE-HPEC 2015, Waltham, MA USA.
   **Best paper award**

2. A. Haidar, T. Dong, S. Tomov, P. Luszczek, and J. Dongarra.
   *A Framework for Batched and GPU-resident Factorization Algorithms Applied to Block Householder Transformations*
   International Supercomputing Conference IEEE-ISC 2015, Frankfurt, Germany.

K. Kabir, A. Haidar, S. Tomov, and J. Dongarra
/*On the Design, Development and Analysis of Optimized Matrix-Vector Multiplication Routines for coprocessors.*/
International SuperComputing Conference IEEE-ISC 2015, Frankfurt, Germany

   *Batched Matrix Computations on Hardware Accelerators Based on GPUs.*
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.
MAGMA Batched Computations

Classical strategies design

- For large problems performance is driven by the update operations, e.g., Level 3 BLAS (GEMM)

Challenges

- For batched small matrices it is more complicated and requires both phases to be efficient

Proposition

- **Rethink and Redesign both phases** in a tuned efficient way
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.
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**
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.

- Optimized kernel
- Using shared memory
- Left v.s. right looking
- Autotuned
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.
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

Graph showing performance metrics for fixed size batched dpotrf (kernel-1), batchCount = 3000, 1 K40c GPU.
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.
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
Anatomy of Optimizing an Algorithm: Performance Analysis and Kernels
Design of the LU Factorization
Consider the LU factorization

Batched dgetrf count = 2000

- Magma v1: classic blocked algorithm
- CuBLAS

- 2x8-core Intel Xeon E5-2670 Sandy Bridge socket
- NVIDIA Kepler K40 GPU
Profile and trace to find bottlenecks

- swap kernel 60%
- gemm kernel 15%
Profile and trace to find bottlenecks

classical swap:

How does the swap work?
Profile and trace to find bottlenecks

Classic swap:

Parallel swap:
MAGMA Batched Computations

- Magma v2: parallel swap
- Magma v1: classic blocked algorithm
- CuBLAS

Batched dgetrf count = 2000

- 2x8-core Intel Xeon E5-2670 Sandy Bridge socket
- NVIDIA Kepler K40 GPU

Matrix size vs. Gflops/s graph with data points for different GPU types and matrix sizes.
Panel factorization classic dgetf2:

Bottlenecks:
- \( nb \) large: panel get slower
  -- very bad performance.
- \( nb \) small: panel get faster but the update is not anymore efficient since dealing with gemm’s of small sizes
  -- very bad performance.
- trade-off? No effect, since we are talking about small size.

Proposition:
- We propose to develop two layers blocking: a recursive and nested blocking technique that block also the panel.
MAGMA Batched Computations GPU

Two-layers blocking:

(a) Recursive nested blocking fashion.

(b) Classical blocking fashion.
MAGMA Batched Computations

Panel factorization
classical dgetf2:

Recursive blocking of
dgetf2:
MAGMA Batched Computations

Batched dgetrf count = 2000

- Magma v3: recursive blocking
- Magma v2: parallel swap
- Magma v1: classic blocked algorithm
- CuBLAS

- 2x8-core Intel Xeon E5-2670 Sandy Bridge socket
- NVIDIA Kepler K40 GPU
MAGMA Batched Computations

- Batched dgetrf count = 2000
- Magma v3: recursive blocking
- Magma v2: parallel swap
- Magma v1: classic blocked algorithm
- CuBLAS

Try to tune and optimize batched dgemm
MAGMA Batched Computations

Batched dgetrf count = 2000

-  Magma v4: streamed/batched gemm
-  Magma v3: recursive blocking
-  Magma v2: parallel swap
-  Magma v1: classic blocked algorithm
-  CuBLAS

Matrix size vs. Gflops/s

-  2x8-core Intel Xeon E5-2670 Sandy Bridge socket
-  NVIDIA Kepler K40 GPU
MAGMA Batched Computations Comparison to CPUs

![Graph showing performance comparison between different computing systems for batched dgetrf 2000 operation. The x-axis represents matrix size, and the y-axis represents GFlops/s. The graph includes lines for GPU: Magma, GPU: CUBLAS, CPU v2: 16 parallel facto using sequential MKL, and CPU v1: each matrix uses MKL multithread_16. The x-axis ranges from 0 to 1000, and the y-axis ranges from 0 to 320. The note indicates that 'Higher is better' and specifies the hardware configurations used: 2x8-core Intel Xeon E5-2670, Sandy Bridge socket, and NVIDIA Kepler K40 GPU.]
MAGMA Batched Computations
Comparison to CPUs

Batched dgeqrf count = 2000

- GPU: Magma
- GPU: CUBLAS
- CPU v2: 16 parallel facto using sequential MKL
- CPU v1: each matrix uses MKL multithread_16

Higher is better

- 2x8-core Intel Xeon E5-2670  Sandy Bridge socket
- NVIDIA Kepler K40 GPU
MAGMA Batched Computations
Comparison to CPUs

Higher is better

Batched dpotrf count = 2000

- GPU: Magma
- CPU v2: 16 parallel facto using sequential MKL
- CPU v1: each matrix uses MKL multithread_16

Matrix size vs. GFlops/s graph

- 2x8-core Intel Xeon E5-2670 Sandy Bridge socket
- NVIDIA Kepler K40 GPU
Energy efficiency GPU

*dgeqrf of 1000 batched matrices of size 1024x1024*

- 2x8-core Intel Xeon E5-2670 Sandy Bridge socket
- NVIDIA Kepler K40 GPU

CPU does not include DRAM power
NVIDIA Jetson TK1
# Tegra K1 Main Specs

<table>
<thead>
<tr>
<th>GPU</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Kepler</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>192</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>ARM Cortex A15 r3</td>
</tr>
<tr>
<td>Cores</td>
<td>4-plus-1</td>
</tr>
<tr>
<td>Frequency</td>
<td>2.3 GHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>DDR3L and LPDDR3</td>
</tr>
<tr>
<td>Size</td>
<td>8 GiB max (40 bit extension)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Display</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD</td>
<td>3840x2160</td>
</tr>
<tr>
<td>HDMI</td>
<td>4K (UltraHD, 4096x2160)</td>
</tr>
<tr>
<td>Manufacturing Process</td>
<td>28 nm</td>
</tr>
</tbody>
</table>
QR Factorization Highlights

- Perfect tool for imperfect data
- Works for over- and under-determined systems
- Deals with rank deficient matrices
  - Rank-revealing QR (RRQR)
  - Better stability than partial pivoting LU on some
- Parallelizes well
  - Numerical stability allows code optimization
- Reduces cost for SVD
  - For certain matrix shapes
Final Performance

Kepler ~ 100 Gflop/s
ARM ~ 1 Gflop/s
MAGMA Batched Computations

Summary

- Batched computation can give a boost in performance for problems with very small sizes.
- Traditional algorithmic design might not be the best direction.
  - We need a new way of thinking.
  - Revisit and redesign algorithm to take advantage of the hardware specifics.
- Performance modeling can help analyzing algorithm and their implementation, for example:
  - An optimized GPU function cannot be efficient for all kinds of computation, it depends on the context used for.
  - Small computations are delicate and require specific kernels (building block or fused).
  - Low level API is required to avoid overhead and context switching.
Future Directions

• Extended functionality
  – Variable sizes (work in progress)
  – Mixed-precision techniques
  – Sparse direct multifrontal solvers & preconditioners
  – Applications

• Further tuning
  – autotuning

• GPU-only algorithms and implementations

• MAGMA Embedded
References


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