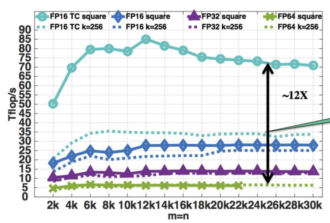
Harnessing GPU's Tensor Cores Fast FP16 Arithmetic to Speedup Mixed-Precision Iterative Refinement Solvers and Achieve 74 Gflops/Watt on Nvidia V100

Azzam Haidar, Ahmad Abdelfattah, Stanimire Tomov, and Jack Dongarra

Abstract: Recent in-hardware GPU acceleration of half precision arithmetic (FP16) -- motivated by various machine learning (ML) and artificial intelligence (AI) applications -- has reinvigorated a great interest in the mixed-precision iterative refinement technique. The technique is based on use of low precision arithmetic to accelerate the general HPC problem of solving Ax = b, where A is a large dense matrix, and the solution is needed in FP64 accuracy. While being a well known technique, its successful modification, software development, and adjustment to match architecture specifics, is challenging. For current manycore GPUs the challenges range from efficient parallelization to scaling, and using the FP16 arithmetic. Here, we address these challenges by showing how to algorithmically modify, develop high-performance implementations, and in general, how to use the FP16 arithmetic to significantly accelerate, as well as make more energy efficient, FP64-precision Ax = b solvers. One can reproduce our results as the developments will be made available through the MAGMA library. We quantify in practice the performance, and limitations of the approach stressing on the use of the Volta V100 Tensor Cores that provide additional FP16 performance boost.

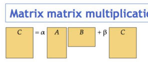
Motivation: Leverage FP16 in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100



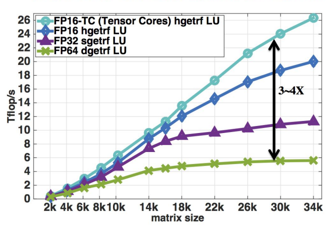
- dgemm achieve about 6.4 Tflop/s
- sgemm achieve about 14 Tflop/s
- hgemm achieve about 27 Tflop/s
- Tensor cores gemm reach about 85 Tflop/s

Matrix matrix multiplication GEMM

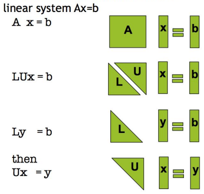


Approach: 1) Develop *Ax=b* solver in FP16

Study of the LU factorization algorithm on Nvidia V100



- LU factorization is used to solve a linear system Ax=b



Approach: 2) Iterative refinement

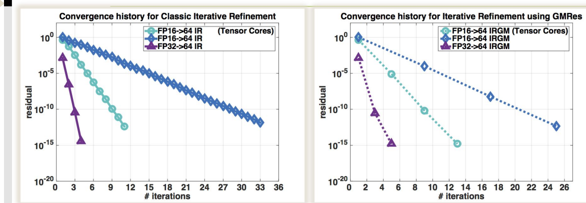
Idea: use lower precision to compute the expensive flops (LU $O(n^3)$) and then iteratively refine the solution in order to achieve the FP64 arithmetic

```
Iterative refinement for dense systems, Ax = b, can work this way.
L U = lu(A)                                                                     lower precision    O(n³)
x = U\(L\b)                                                                     lower precision    O(n²)
r = b – Ax                                                                      FP64 precision     O(n²)

WHILE || r || not small enough
    1. find a correction "z" to adjust x that satisfy Az=r
       solving Az=r could be done by either:
         ➢ z = U\(L\r)                                 Classical Iterative Refinement         lower precision    O(n²)
         ➢ GMRes preconditioned by the LU to solve Az=r  Iterative Refinement using GMRes     lower precision    O(n²)
    2. x = x + z                                                                FP64 precision     O(n¹)
    3. r = b – Ax                                                               FP64 precision     O(n²)
END
```

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

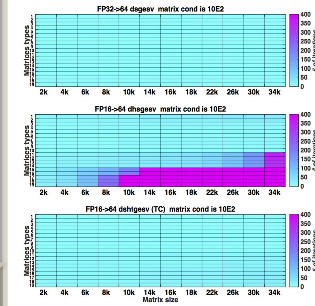
Numerical behavior of FP16 on V100



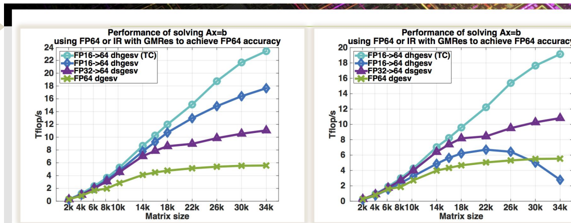
- Convergence history of the iterative refinement solver to achieve FP64 solution accuracy.
- Left graph shows the classical iterative refinement method.
- Right graph shows the iterative refinement using GMRes.
- Problem generated with an arithmetic distribution of the singular values $\sigma_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$
- and where condition number of the matrix is 10^2.
- The FP32->64 algorithm converge as expected by theory and is able to achieve the FP64 solution accuracy in about 3-5 iterations for this types of matrices.
- The FP16->64 algorithm requires more iterations because of the lower precision factorization.
- The FP16->64 (Tensor Cores) outperforms the FP16->64 because the accumulation during the FP16-TC GEMM (Schur update) use 32-bit.
- Iterative Refinement using GMRes (IRGM) is always preferred in order to achieve FP64 accuracy.

- Convergence history of the iterative refinement solver to achieve FP64 solution accuracy for the three algorithms:
  - FP32->FP64
  - FP16->FP64
  - FP16->64 (Tensor Cores)
- Analysis performed on 19 types of matrices [1] (vertical view) for each of the algorithms. Types are ordered by increased difficulty.
- Graphs also illustrate the effect of the matrix sizes on the convergence of each algorithm (horizontal view).
- The use of the FP16-64 Tensor Cores accelerates most of the problems since it requires small number of iterations.
- The FP16-64 is able to cope with many problem but may fail for other, more difficult problems.
- The FP32->64 work as expected by theory for all the problem considered.

Results illustrate that different type of problems can be accelerated by the usage of the FP16 arithmetic.



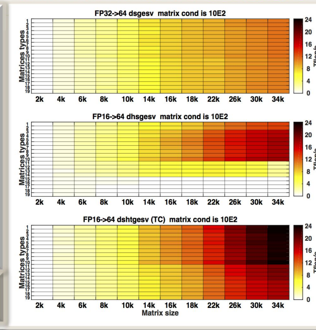
Performance results on V100



- solving Ax = b using FP64 LU
- solving Ax = b using FP32 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 Tensor Cores LU and iterative refinement to achieve FP64 accuracy
- Using FP16-TC solver achieves about 4X speedup over the FP64 solver and provides solution in FP64 accuracy.
- The FP16-TC solver requires less iterations than FP16 classic and thus provide better performance.
- For more difficult problem (right), the number of iterations of FP16 classic increases thus the performance drop down for that the performance of the FP16 classic drop for difficult matrices (right figure).

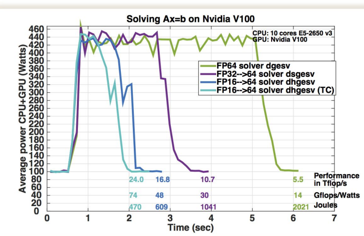
- Performance (reflecting time to solution) of the iterative refinement solver to achieve FP64 solution accuracy for the three algorithms:
  - FP32->FP64
  - FP16->FP64
  - FP16->64 (Tensor Cores)
- Analysis performed on 19 types of matrices [1] (vertical view) for each of the algorithms. Types are ordered by increased difficulty.
- Graphs also illustrate the effect of the matrix sizes on the performance of each algorithm (horizontal view).
- The use of the FP16-64 Tensor Cores accelerates most of the problems and provides up to 4X speedup over the FP64 solver (24 Tflop/s versus 4.7 Tflop/s).
- The FP16-64 is able to accelerate many problem reaching about 3X speedup (16 Tflop/s versus 4.7 Tflop/s) but may fail to accelerate other problems.
- The FP32->64 work as expected and provide around 2X speedup (10 Tflop/s versus 4.7 Tflop/s).

Results illustrate that different type of problems can be accelerated up to 4X by the usage of the FP16 or 2X otherwise, using the FP32 arithmetic.



Energy efficiency on V100

Power awareness



Matrices generated with positive λ and arithmetic distribution of its singular values $\sigma_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and where its condition number is equal to 10^2.

Mixed precision techniques can provide a large gain in energy efficiency

- Power consumption of the FP64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 5.5 Tflop/s and requires about 2021 joules providing about 14 Gflops/Watts.
- Power consumption of the mixed precision FP32→64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 10.7 Tflop/s and requires about 1041 joules providing about 30 Gflops/Watts.
- Power consumption of the mixed precision FP16→64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 16.8 Tflop/s and requires about 609 joules providing about 48 Gflops/Watts.
- Power consumption of the mixed precision FP16→64 TC algorithm using Tensor Cores to solve Ax=b for a matrix of size 34K, it achieve 24 Tflop/s and requires about 470 joules providing about 74 Gflops/Watts.

Conclusion:

- We accelerated the solution of linear system Ax = b solver using hardware-accelerated FP16 arithmetic on GPUs;
- We introduced a framework for exploiting mixed-precision FP16-FP32/FP64 iterative refinement solvers and describe the path to draw high-performance and energy-aware GPU implementations;
- Our technique shows that a number of problems can be accelerated up to 4X by the usage of the FP16 or 2X otherwise, using the FP32 arithmetic.
- We studied the energy-efficiency of our approach that showed incredible energy savings, 5X energy savings compared to the FP64 implementation.
- We illustrated a technique to use V100 Tensor Cores that achieves FP64 accuracy at a highly efficient/accelerated performance equating to 74 FP64 Gflops/Watt and 24 FP64 Tflops/s.

Acknowledgement: This work was supported by the Exascale Computing Project, a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This work was also partially supported by the National Science Foundation under Grant OAC-1740250 and NVIDIA.

REFERENCES:
[1] A. Haidar, P. Wu, S. Tomov, J. Dongarra, **Investigating Half Precision Arithmetic to Accelerate Dense Linear System Solvers**, SC-17, ScalA17: 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ACM, Denver, Colorado, November 12-17, 2017.
[2] A. Haidar, P. Wu, S. Tomov, J. Dongarra, **Harnessing GPU's Tensor Cores Fast FP16 Arithmetic to Speedup Mixed-Precision Iterative Refinement Solvers**, https://arxiv.org/