Motivation
Integer arithmetic is available on most hardware architectures. FPGA is usually more capable in integer operations and might not have floating-point number arithmetic units. New application-specific integrated circuits (ASICs) for deep learning inference are also made in mind using mostly integer arithmetic in quantized neural networks. This motivated this study to look at the fundamental numerical linear algebra operation: Gaussian elimination (LU factorization) with partial pivoting using integer neural arithmetic (INNA) solver linear system Ax = b. The goal is to have a low accuracy but fast solution and factorization for later preconditioned iterative refinement in mixed precision algorithm.

Number Representations

<table>
<thead>
<tr>
<th>Format</th>
<th>Range</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>2^31 - 1</td>
<td>±1</td>
</tr>
<tr>
<td>Double-Precision (FP64)</td>
<td>2^52 - 1</td>
<td>±1</td>
</tr>
<tr>
<td>Single-Precision (FP32)</td>
<td>2^23 - 1</td>
<td>±1</td>
</tr>
<tr>
<td>Half-Precision (FP16)</td>
<td>2^11 - 1</td>
<td>±1</td>
</tr>
<tr>
<td>Bitwidth</td>
<td>0 to 32</td>
<td>1/2</td>
</tr>
<tr>
<td>INT16</td>
<td>0 to 65535</td>
<td>1/2</td>
</tr>
<tr>
<td>INT32</td>
<td>0 to 4294967295</td>
<td>1/2</td>
</tr>
</tbody>
</table>

Related Mixed Precision Work
Heister et al. (1) achieved 4x speed up by utilizing the half-precision tensor core from NVIDIA Volta architecture for solving linear system. The matrix multiplication operation in tensor core is accumulating in single precision so the result is better than zero half precision in previous architectures. Carson and Higham (2) provided error analysis for varying the precision in different part of the mixed precision algorithm as well as the condition number of the input matrix A. Higham et al. (3) proposed a method to deal with the limited range of half precision format (FP16).

HPL-AI Mixed Precision Benchmark
The HPL-AI benchmark seeks to highlight the emerging convergence of high-performance computing (HPC) and artificial intelligence (AI) workloads. While traditional HPC focused on simulation runs for modeling phenomena in physics, chemistry, biology, and so on, the mathematical models that drive these computations require, for the most part, double precision accuracy. On the other hand, the machine learning methods that fuel advances in AI achieve desired results at 32-bit and even lower floating-point precision formats. This lesser demand for accuracy fueled a resurgence of interest in new hardware platforms that deliver a mix of unprecedented performance levels and energy savings to achieve the classification and recognition fidelity afforded by higher-accuracy formats.

Proposed Algorithm
Input: A, b, m by n matrix A in double precision, n vectors b in double precision. Convert A into proposed fixed-point representation. Convert b into proposed fixed-point representation. Scale the columns. Flip the pivot indices. Compute P' = L'U'. The input matrix A is then converted into P'Ap. LU Factorization with Integer Arithmetic
The output and output matrices of this algorithm are still in double precision to be comparable with the reference factorization from LAPACK. The input integer r determines how many bits we are actually using (32) while converting A into integer format. Because the matrix would grow during the factorization and we do not dynamically scale, it might hit the range and overflow at some point. To avoid this, we first scale the input into [-2^2, 2^2]. The higher r is, the more we will have from the range. But less accurate the input matrix will be after being converted into [-2^2].

The computation inside the loop is mainly 32-bit integer arithmetic. Line 9 requires 64-bit integer division but only once per column. The scaling at line 10 will remain in 32-bit because the joint has larger magnitude then other elements in the column. The update in line 11 is 32-bit integer multiply but we only need the high 32 bits in 64 bits result. Other than mentioned lines, the algorithm mimics the standard LU factorization with partial pivoting. Partial pivoting also controls the growth rate to be at most 2. Otherwise the overflow could happen easily.

Conclusion and Future Work
We have demonstrated that it is feasible to use 32-bit integer arithmetic perform LU factorization with partial pivoting and achieve better accuracy than single precision. The main issue for this approach is the possibility of overflow during factorization. To tackle the problem, we would like to continue the research on following directions:

- Dynamically scale the column during factorization: While finding the pivot, if it’s too close to overflow, we can further scale down the column and remaining factorization. It can also be done in parallel factorization and do the check once for each column.
- Other representation formats. There are other less common number representation like blocked floating-point numbers. Although they might not have native hardware support, they could be cast as new better in the algorithm.
- Error analysis. We would like to perform error analysis to have a better understanding about behavior of algorithm and incorporate the findings to improve the algorithm.
- Smaller data size including int 32 and int 16. The goal of this project is to find a fast solver using smaller dataset. The computational complexity of factorization is (O(n^3)) while the latter refinement is (O(n^2)). So the factorization is critical to overall performance of mixed-precision solver. Smaller data size might give more potential for speedup while the desired accuracy can still be obtained with preconditioned GMRES or other refinement schemes.

REFERENCES

Numerical Results
The figure shows the unscaled backward error $\|r\|_2 / \|b\|_2$, input matrix size. The algorithm is implemented in MATLAB R2021b. Each element of the matrix is generated from random normal distribution uniform(1,3). The results from single and double precision LU factorization are also reported as reference. While 32-bit, overflow occurs and the algorithm failed. The higher r is, the similar range input matrix A will be normalized into. There will be more room from overflowing so it could work with larger matrix. However, the backward error grows with a since the input is truncated more. Nevertheless, when $r > 10$ it is still using 32-bit/128 bits and the result is still better than single precision which is using 23 mantissa bits.

The research was supported by the Illinois Computing Project (I-182015-19C), a collaborative effort of the Center of Energy Office of Science within the National Nuclear Security Administration, under contract DE-AC02-06CH11357, and IT Bandfellow $44003515042$.