HPC: Where We Are Today And A Look Into The Future

Jack Dongarra
University of Tennessee, Oak Ridge National Laboratory, and University of Manchester
Confessions of an Accidental Benchmarker

- Appendix B of the Linpack Users’ Guide
  - Designed to help users extrapolate execution time for Linpack software package
- First benchmark report from 1977;
  - Cray 1 to DEC PDP-10
Started 42 Years Ago
Have seen a Factor of $10^9$ - From 14 Mflop/s to 442 Pflop/s

- In the late 70’s the fastest computer ran LINPACK at 14 Mflop/s
- Today with HPL we are at 442 Pflop/s
  - Roughly 10 orders of magnitude in performance
  - About 7 orders of magnitude increase in the number of processors
  - Plus many algorithmic improvements

It's important to remember this began in late 70’s when floating point operations were expensive compared to data movement and other operations.
Top500/HPL Benchmark Timeline

- 1974 LINPACK library (vector computers)
- 1977 LINPACK 100 Benchmark (Fortran only)
- 1986 LINPACK 1000 Benchmark (assembly allowed)
- **1988 First Gflop/s system, NEC SX-2**
- 1991 LINPACK Table 3 (HPL rules defined)
- **1993 Top500 starts (LANL CM-5 #1, 60 Gflop/s)**
- 1994 All Top500 systems over 1 Gflop/s
- **1997 First Tglop/s system ASCI Red @ SNL**
- 2001 China has its first computer on Top500
- 2002 #1 Earth Simulator @ JAMSTEC 5 X faster
- 2005 All Top500 systems over 1 Tglop/s
- **2008 First Pflop/s system Roadrunner @ LANL**
- 2010 Tianhe-1A @ Tianjin (2 Pflop/s) 1st time China #1
- 2011 K computer @ RIKEN (8 Pflop/s)
- 2012 Sequoia @ LLNL (1.5 M cores)
- 2014 Tianhe-2 @ NUDT (33 Pflop/s)
- 2016 China and US have = number of systems
- 2016 TaihuLight @ Wuxi (93 Pflop/s, >10 M cores)
- 2018 Summit @ ORNL (122 Pflop/s)
- **2019 All TOP500 systems over 1Pflop/s**
- 2020 Fugaku @ RIKEN (442 Pflop/s)
- 2020 Of the Top500: China=212 & US=113
- **2021 First Exascale System?**
State of Supercomputing in 2021

• Pflops (> $10^{15}$ Flop/s) computing fully established with all 500 systems.

• Three technologies / architectures.
  • Commodity (e.g. Intel)
  • Commodity + accelerator (e.g. GPUs) (162 systems; 153 NVIDIA, 3 Intel Phi + 6 other)
  • Lightweight cores (e.g. IBM BG, Xeon Phi, TaihuLight, ARM (4 system))

• China: Top consumer and producer overall (but not for the TOP50 (yet))

• Interest in supercomputing is now worldwide, and growing in many new markets (~50% of Top500 computers are in industry).

• Intel processors largest share, 92% followed by AMD, 4%.

• Exascale ($10^{18}$ Flop/s) projects exist in many countries and regions.
PERFORMANCE DEVELOPMENT OF HPC OVER THE LAST 28 YEARS FROM THE TOP500

In 1993 the #1 system was the Thinking Machine CM-5 with 1024 processors at Los Alamos National Laboratory.

My Laptop: 166 Gflop/s

SUM

N=1

N=500
# Systems on the Top500 Over the Past 27 Years

<table>
<thead>
<tr>
<th>Top500 List (No. of times)</th>
<th>Computer</th>
<th>HPL $r_{\text{max}}$ (Tflop/s)</th>
<th>Procs/Cores</th>
<th>Matrix Size</th>
<th>Hours To BM</th>
<th>MW</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/93 (1)</td>
<td>TMC CM-5/1024 (DOE LANL)</td>
<td>.060</td>
<td>1,024</td>
<td>52,224</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>11/93 (1)</td>
<td>Fujitsu Numerical Wind Tunnel (Nat. Aerospace Lab of Japan)</td>
<td>.124</td>
<td>140</td>
<td>31,920</td>
<td>0.1</td>
<td>1.</td>
</tr>
<tr>
<td>6/94 (1)</td>
<td>Intel XP/S140 (DOE SNL)</td>
<td>.143</td>
<td>3,680</td>
<td>55,700</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>11/94–11/95 (3)</td>
<td>Fujitsu Numerical Wind Tunnel (Nat. Aerospace Lab of Japan)</td>
<td>.170</td>
<td>140</td>
<td>42,000</td>
<td>0.1</td>
<td>1.</td>
</tr>
<tr>
<td>6/96 (1)</td>
<td>Hitachi SR2201/1024 (Univ. of Tokyo)</td>
<td>.220</td>
<td>1,024</td>
<td>138,240</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>11/96 (1)</td>
<td>Hitachi CP-PACS/2048 (Univ of Tsukuba)</td>
<td>.368</td>
<td>2,048</td>
<td>103,680</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>6/97–6/00 (7)</td>
<td>Intel ASCI Red (DOE SNL)</td>
<td>2.38</td>
<td>9,632</td>
<td>362,880</td>
<td>3.7</td>
<td>.85</td>
</tr>
<tr>
<td>11/00–11/01 (3)</td>
<td>IBM ASCI White, SP Power3 375 MHz (DOE LLNL)</td>
<td>7.23</td>
<td>8,192</td>
<td>518,096</td>
<td>3.6</td>
<td></td>
</tr>
<tr>
<td>6/02–6/04 (5)</td>
<td>NEC Earth-Simulator (JAMSTEC)</td>
<td>35.9</td>
<td>5,120</td>
<td>1,000,000</td>
<td>5.2</td>
<td>6.4</td>
</tr>
<tr>
<td>11/04–11/07 (7)</td>
<td>IBM BlueGene/L (DOE LLNL)</td>
<td>478.</td>
<td>212,992</td>
<td>1,000,000</td>
<td>0.4</td>
<td>1.4</td>
</tr>
<tr>
<td>6/08–6/09 (3)</td>
<td>IBM Roadrunner -PowerXCell 8i 3.2 Ghz (DOE LANL)</td>
<td>1,105.</td>
<td>129,600</td>
<td>2,329,599</td>
<td>2.1</td>
<td>2.3</td>
</tr>
<tr>
<td>11/09–6/10 (2)</td>
<td>Cray Jaguar - XT5-HE 2.6 GHz (DOE ORNL)</td>
<td>1,759.</td>
<td>224,162</td>
<td>5,474,272</td>
<td>17</td>
<td>6.9</td>
</tr>
<tr>
<td>11/10 (1)</td>
<td>NUDT Tianhe-1A, X5670 2.93Ghz NVIDIA (NSC Tianjin)</td>
<td>2,566.</td>
<td>186,368</td>
<td>3,600,000</td>
<td>3.4</td>
<td>4.0</td>
</tr>
<tr>
<td>6/11–11/11 (2)</td>
<td>Fujitsu K computer, SPARC64 VIIIfx (RIKEN)</td>
<td>10,510.</td>
<td>705,024</td>
<td>11,870,208</td>
<td>29</td>
<td>9.9</td>
</tr>
<tr>
<td>6/12 (1)</td>
<td>IBM Sequoia BlueGene/Q (DOE LLNL)</td>
<td>16,324.</td>
<td>1,572,864</td>
<td>12,681,215</td>
<td>23</td>
<td>7.9</td>
</tr>
<tr>
<td>11/12 (1)</td>
<td>Cray XK7 Titan AMD + NVIDIA Kepler (DOE ORNL)</td>
<td>17,590.</td>
<td>560,640</td>
<td>4,423,680</td>
<td>0.9</td>
<td>8.2</td>
</tr>
<tr>
<td>6/13–11/15 (6)</td>
<td>NUDT Tianhe-2 Intel IvyBridge + Xeon Phi (NCSS Guangzhou)</td>
<td>33,862.</td>
<td>3,120,000</td>
<td>9,960,000</td>
<td>5.4</td>
<td>17.8</td>
</tr>
<tr>
<td>6/16–11/17 (4)</td>
<td>Sunway TaihuLight System (NSCC Wuxi)</td>
<td>93,014.</td>
<td>10,549,600</td>
<td>12,288,000</td>
<td>3.7</td>
<td>15.4</td>
</tr>
<tr>
<td>6/18–11/19 (4)</td>
<td>IBM Summit Power9 + Nvidia Volta (DOE ORNL)</td>
<td>148,600</td>
<td>2,414,592</td>
<td>16,473,600</td>
<td>3.3</td>
<td>10.1</td>
</tr>
<tr>
<td>6/20–?</td>
<td>Fujitsu Fugaku ARM A64FX (RIKEN)</td>
<td>442,010</td>
<td>7,630,828</td>
<td>21,288,960</td>
<td>4.4</td>
<td>29.9</td>
</tr>
</tbody>
</table>
November 2020: The TOP 10 Systems
(42% of the Total Performance in Top 10; 50% in Top20)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer</th>
<th>Country</th>
<th>Cores</th>
<th>Rmax [Pflops]</th>
<th>% of Peak</th>
<th>Power [MW]</th>
<th>GFlops/Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RIKEN Center for Computational Science</td>
<td>Fugaku, ARM A64FX (48C, 2.2 GHz), Tofu D Interconnect</td>
<td>Japan</td>
<td>7,299,072</td>
<td>442</td>
<td>82</td>
<td>29.9</td>
<td>14.8</td>
</tr>
<tr>
<td>2</td>
<td>DOE / OS Oak Ridge Nat Lab</td>
<td>Summit, IBM Power 9 (22C, 3.0 GHz), NVIDIA GV100 (80C), Mellonox EDR</td>
<td>USA</td>
<td>2,397,824</td>
<td>149</td>
<td>74</td>
<td>10.1</td>
<td>14.7</td>
</tr>
<tr>
<td>3</td>
<td>DOE / NNSA L Livermore Nat Lab</td>
<td>Sierra, IBM Power 9 (22C, 3.1 GHz), NVIDIA GV100 (80C), Mellonox EDR</td>
<td>USA</td>
<td>1,572,480</td>
<td>94.6</td>
<td>75</td>
<td>7.44</td>
<td>12.7</td>
</tr>
<tr>
<td>4</td>
<td>National Super Computer Center in Wuxi</td>
<td>Sunway TaihuLight, SW26010 (260C), Custom</td>
<td>China</td>
<td>10,649,000</td>
<td>93.0</td>
<td>74</td>
<td>15.4</td>
<td>6.05</td>
</tr>
<tr>
<td>5</td>
<td>NVIDIA Corporation</td>
<td>Selene NVIDIA DGX A100, AMD EPYC 7742 (64C, 2.25GHz), NVIDIA A100 (108C), Mellanox HDR</td>
<td>USA</td>
<td>555,520</td>
<td>63.4</td>
<td>80</td>
<td>18.4</td>
<td>3.32</td>
</tr>
<tr>
<td>6</td>
<td>National Super Computer Center in Guangzhou</td>
<td>Tianhe-2A NUDT, Xeon (12C) + MATRIX-2000 (128C), Custom</td>
<td>China</td>
<td>4,981,760</td>
<td>61.4</td>
<td>61</td>
<td>18.5</td>
<td>3.32</td>
</tr>
<tr>
<td>7</td>
<td>JUWELS Booster Module</td>
<td>Bull Sequana XH2000, AMD EPYC 7402 (24C, 2.86GHz), NVIDIA A100 (108C), Mellonox HDR InfiniBand/ParTec ParaStation ClusterSuite</td>
<td>Italy</td>
<td>448,280</td>
<td>44.1</td>
<td>62</td>
<td>1.76</td>
<td>25.0</td>
</tr>
<tr>
<td>8</td>
<td>Eni S.p.A in Italy</td>
<td>HPC5, Dell EMC PowerEdge C4140, Xeon (24C, 2.1 GHz) + NVIDIA V100 (80C), Mellonox HDR</td>
<td>Italy</td>
<td>669,760</td>
<td>35.5</td>
<td>69</td>
<td>2.25</td>
<td>15.8</td>
</tr>
<tr>
<td>9</td>
<td>Texas Advanced Computing Center / U of Texas</td>
<td>Frontera, Dell C6420, Xeon Platinum, 8280 (28C, 2.7 GHz), Mellonox HDR</td>
<td>USA</td>
<td>448,448</td>
<td>23.5</td>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Saudi Aramco</td>
<td>Cray CS-Storm, Xeon Gold 6248 (20C, 2.5GHz), NVIDIA Tesla V100 SXM2 (80C), InfiniBand HDR</td>
<td>USA</td>
<td>672,520</td>
<td>22.4</td>
<td>61</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Today’s HPC Environment for Numerical Libraries

• Highly parallel
  – Distributed memory
  – MPI + Open-MP programming model

• Heterogeneous
  – Commodity processors + GPU accelerators

• Simple loop level parallelism too limiting in terms of performance

• Communication between parts very expensive compared to floating point ops

• Comparison of operation counts may not reflect time to solution

• Floating point hardware at 64, 32, and 16 bit levels
Over the Past 50 Years Evolving SW and Alg Tracking Hardware Developments

<table>
<thead>
<tr>
<th></th>
<th>Software/Algorithms follow hardware evolution in time</th>
</tr>
</thead>
</table>
| **EISPACK (1970’s)** (Translation of Algol to F66) | Rely on  
  - Fortran, but row oriented |
| **LINPACK (1980’s)** (Vector operations) | Rely on  
  - Level-1 BLAS operations  
  - Column oriented |
| **LAPACK (1990’s)** (Blocking, cache friendly) | Rely on  
  - Level-3 BLAS operations |
| **ScaLAPACK (2000’s)** (Distributed Memory) | Rely on  
  - PBLAS Mess Passing |
| **PLASMA / MAGMA (2010’s)** (Many-core friendly & GPUs) | Rely on  
  - DAG/scheduler  
  - block data layout |
| **SLATE (2020’s)** (DM and Heterogeneous arch) | Rely on  
  - Tasking DAG scheduling  
  - Tiling, but tiles can come from anywhere  
  - Heterogeneous HW, Batched dispatch |
SLATE: Software for Linear Algebra Targeting Exascale

- Multicore, distributed, accelerator based, dense linear algebra library
  - Target large HPC machines
  - BLAS: matrix multiply \((C = AB)\), etc.
  - Linear systems \((Ax = b)\)
    - LU, Cholesky, symmetric indefinite
  - Least squares \((Ax \approx b)\)
    - QR, LQ
  - Eigenvalue \((Ax = \lambda x)\)
  - SVD \((A = U\Sigma V^H)\)

- Modern replacement for Sca/LAPACK
  - Explicit multi-threading (OpenMP) using MPI
  - C++ 17
The Curse of Pivoting: Data Movement Issues on Heterogeneous Architectures

- For solving $Ax=b$, for a general matrix $A$, row pivoting is critical for numerical stability, without pivoting the algorithm is unstable*.

- However pivoting adds overhead (testing and data movement)

- For each column: need to find the maximum element and swap rows

- Significant overhead on a heterogeneous architecture

- Data movement cripples the performance.
Ax = b, using LU Decomposition

Pivoting needed to:
- Avoid dividing by zero or a small number
  - Degeneracy
- Avoid growth in the elements
  - Loss of accuracy

\[ A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad A = \begin{pmatrix} 10^{-16} & 1 \\ 1 & 0 \end{pmatrix} \]

- 16 nodes of Summit (4 × 8 grid)
- Spectrum MPI, ESSL
Algorithmic Improvements:
Avoid Pivoting; Minimize Communication & Synchronization

Goal: Transform A into a matrix that would be sufficiently “random” so that, with a probability close to 1, pivoting is not needed.

Think of the transformation that randomizes the elements in the matrix to precondition the matrix.

- To solve $Ax = b$:
  - Compute $A_r = U^TAV$, with $U$ and $V$ structured butterfly matrices
  - Factorize $A_r$ without pivoting (GENP)
  - Solve $A_r y = U^Tb$ and then Solve $x = Vy$

- $U$ and $V$ are Recursive Butterfly Matrices
  - Randomization is cheap ($O(n)$ operations)
  - GENP is fast (can take advantage of the GPU)
  - Accuracy is in practice similar to GEPP, but...

A butterfly matrix is defined as any $n$-by-$n$ matrix of the form:

$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} R & S \\ R & -S \end{pmatrix}$$

where $R$ and $S$ are random diagonal matrices.

In the simple case $U$ and $V$ have the same structure as $B$.
d=2 applies the following transformation

- 16 nodes of Summit (4 × 8 grid)
- Spectrum MPI, ESSL
Accuracy of RBT-LU (depth 2)

\[ \text{Error} = \frac{\| b - Ax \|_1}{\| A \|_1 \| x \|_1} \]

- LAPACK/ScaLAPACK/SLATE
- Other matrices

\[ d=2 \text{ applies the following transformation} \]

\[ n = 16384 \]
Butterfly transformation for $d=14$

RBT performance

- $d_{gesv\_-hopiv}$
- $d_{gesv\_rbt \ - \ d=2}$
- $d_{gesv\_rbt \ - \ d=14}$
- $d_{gesv}$

- 16 nodes of Summit ($4 \times 8$ grid)
- Spectrum MPI, ESSL
gesv (with partial pivoting) error

\[ \text{Error} = \frac{\| b - Ax \|_1}{\| A \|_1 \| x \|_1} \]

\[ \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 1 \\
-1 & 1 & 0 & 0 & 0 & 0 & 1 \\
-1 & -1 & 1 & 0 & 0 & 0 & 1 \\
-1 & -1 & -1 & 1 & 0 & 0 & 1 \\
-1 & -1 & -1 & -1 & 1 & 0 & 1 \\
-1 & -1 & -1 & -1 & -1 & 1 & 1 \\
-1 & -1 & -1 & -1 & -1 & -1 & 1
\end{bmatrix} \]

gfpp matrix causes LU w/pivoting to fail with exponential growth in matrix elements.

Random butterfly transformation for \( d=14 \)

\( n = 16384 \)
Rethinking Algorithms

- Traditionally, we use a strong coupling between the precision formats used for arithmetic operations and storing data.
- The arithmetic operations are in effect free.
- Data access should be as cheap as possible, reduced precision.

- Communication is the bottleneck!
Rethinking Algorithms

- Traditionally, we use a strong coupling between the precision formats used for arithmetic operations and storing data.
- The arithmetic operations are in effect free.
- Data access should be as cheap as possible, reduced precision.
My interest in mixed precision began with my dissertation...

- *Improving the Accuracy of Computed Matrix Eigenvalues*
  - Compute the eigenvalues and eigenvectors in low precision then improve selected values/vectors to higher precision for $O(n^2)$ ops using the matrix decomposition

- Extended to singular values, 1983
- Algorithm in TOMS 710, 1992
IBM’s Cell Processor - 2004

- **9 Cores**
  - Power PC at 3.2 GHz
  - 8 SPEs

- **204.8 Gflop/s peak!**
  - The catch is that this is for 32 bit fl pt; (Single Precision SP)
  - 64 bit fl pt peak at 14.6 Gflop/s
    - 14 times slower that SP; factor of 2 because of DP and 7 because of latency issues

The SPEs were fully IEEE-754 compliant in double precision. In single precision, they only implement round-towards-zero, denormalized numbers are flushed to zero and NaNs are treated like normal numbers.
Mixed Precision Idea Goes Something Like This...

- Exploit 32 bit floating point as much as possible.
  - Especially for the bulk of the computation
- Correct or update the solution with selective use of 64 bit floating point to provide a refined results
- Intuitively:
  - Compute a 32 bit result,
  - Calculate a correction to 32 bit result using selected higher precision and,
  - Perform the update of the 32 bit results with the correction using high precision.
Iterative refinement for dense systems, \(Ax = b\), can work this way.

\[
\begin{align*}
L U &= lu(A) \\
x &= U\backslash(L\backslash b) \\
r &= b - Ax \text{ (with original } A) \\
\end{align*}
\]

FP32 precision \(O(n^3)\)  
FP32 precision \(O(n^2)\)  
FP64 precision \(O(n^2)\)

\[
\begin{align*}
&\text{WHILE } ||r|| \text{ not small enough} \\
&1. \text{ find a correction }"z"\text{ to adjust }x\text{ that satisfy }Az=r \\
&2. \quad x = x + z \\
&3. \quad r = b - Ax \text{ (with original } A) \\
&\end{align*}
\]

FP32 precision \(O(n^2)\)  
FP64 precision \(O(n^1)\)  
FP64 precision \(O(n^2)\)

**Idea:** use low precision to compute the expensive flops (\(LU \quad O(n^3)\)) and then iteratively refine (\(O(n^2)\)) the solution in order to achieve the FP64 arithmetic.

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.

Need a copy of the original matrix to compute residual (\(r\)) and matrix cannot be too badly conditioned.

Leveraging Half Precision

Idea: use low precision to compute the expensive flops (LU $O(n^3)$) and then iteratively refine ($O(n^2)$) the solution in order to achieve the FP64 arithmetic.

Iterative refinement for dense systems, $Ax = b$, can work this way.

1. Find a correction "z" to adjust $x$ that satisfy $Az = r$.
   - Classical Iterative Refinement
   - GMRes preconditioned by the LU to solve $Az = r$ Iterative Refinement

2. $x = x + z$

3. $r = b - Ax$ (with original $A$)

WHILE $|| r ||$ not small enough

E. Carson & N. Higham showed can solve the inner problem with iterative method and not infect the solution.

- 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.
- Need the original matrix to compute residual ($r$) and matrix cannot be too badly conditioned.

Intriguing Potential

• Exploit lower precision as much as possible
  ▪ Payoff in performance
    • Faster floating point
    • Less data to move

• Automatically switch between SP and DP to match the desired accuracy
  ▪ Compute solution in SP and then a correction to the solution in DP

• Potential for GPU, FPGA, special purpose processors
  ▪ Use as little precision as you can get away with and improve the accuracy

• Linear systems and Eigenvalue, optimization problems, where Newton’s method is used.

\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]

\[ x_{i+1} - x_i = -\frac{f(x_i)}{f'(x_i)} \]

\[ z = -A(b - Ax) \]
Many fields are beginning to adopt machine learning to augment modeling and simulation methods

- Climate
- Biology
- Drug Design
- Epidemiology
- Materials
- Cosmology
- High-Energy Physics
Today’s Floating-Point Precision Arithmetic

<table>
<thead>
<tr>
<th>Type</th>
<th>Bits</th>
<th>Range</th>
<th>$u = 2^t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>fp128 quad</td>
<td>128</td>
<td>$10^{4932}$</td>
<td>$2^{-113} \approx 1 \times 10^{-34}$</td>
</tr>
<tr>
<td>fp64 double</td>
<td>64</td>
<td>$10^{308}$</td>
<td>$2^{-53} \approx 1 \times 10^{-16}$</td>
</tr>
<tr>
<td>fp32 single</td>
<td>32</td>
<td>$10^{38}$</td>
<td>$2^{-24} \approx 6 \times 10^{-8}$</td>
</tr>
<tr>
<td>fp16 half</td>
<td>16</td>
<td>$10^{5}$</td>
<td>$2^{-11} \approx 5 \times 10^{-4}$</td>
</tr>
<tr>
<td>bfloat16 half</td>
<td>16</td>
<td>$10^{38}$</td>
<td>$2^{-8} \approx 4 \times 10^{-3}$</td>
</tr>
</tbody>
</table>

Half precision increasingly supported by hardware
- Present: NVIDIA Pascal, Volta, and Ampere GPUs, AMD Radeon Instint M125 GPU, Google TPU, ARM NEON, and Fujitsu A64FX ARM
- Near future: IBM AI chips, Intel Xeon Cooper Lake and Intel Nervana Neural Network
Problem generated with an arithmetic distribution of the singular values and positive eigenvalues. Note that solving $Ax = b$ using either FP64 arithmetic solution using either the $s_i = 1 - \left( \frac{i-1}{n-1} \right) (1 - \frac{1}{\text{cond}})$ and positive eigenvalues.

- solving $Ax = b$ using FP64 LU
- solving $Ax = b$ using FP16 Tensor Cores LU and iterative refinement to achieve FP64 accuracy
- solving $Ax = b$ using BF16 Tensor Cores LU and iterative refinement to achieve FP64 accuracy
- solving $Ax = b$ using TF32 Tensor Cores LU and iterative refinement to achieve FP64 accuracy

Results obtained using CUDA 11.0 and A100 GPU.
Leveraging Half Precision in HPC

**Use Mixed Precision algorithms**

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

- Achieve higher performance \(\rightarrow\) faster time to solution
- Reduce power consumption by decreasing the execution time \(\rightarrow\) **Energy Savings !!!**
Problem generated with an arithmetic distribution of the singular values and positive eigenvalues.

Mixed precision techniques can provide a large gain in energy efficiency:

- Power consumption for a matrix of size 40K
- The **FP64 algorithm** achieve 5.3 Tflop/s providing about **21 Gflops/Watts**.
- The **FP32→64 algorithm** achieve 10 Tflop/s providing about **40 Gflops/Watts**.
- The **FP16→64 TC algorithm using Tensor Cores** achieve 22 Tflop/s providing about **94 Gflops/Watts**.

**Results obtained using CUDA 10.2 and GV100 GPU.**
The Take Away

- HPC Constantly Changing
  - Scalar
  - Vector
  - Distributed
  - Accelerated
  - Mixed precision

- Data movement critical for performance.
- Algorithm / Software advances follows hardware
  - And there is “plenty of room at the top”
  - “There's life in the old dog yet”