RIKEN's Effort for Sustainable Linear Algebra Computations

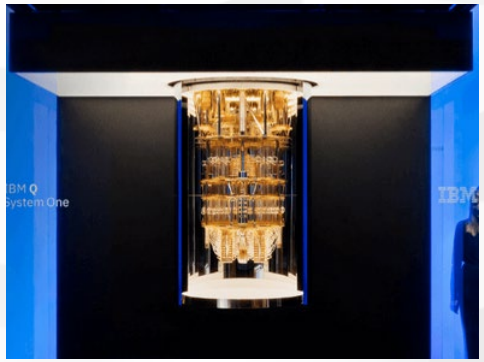
(Mainly mixed-precision computing on Fugaku and emerging systems, as post-Fugaku)

Toshiyuki Imamura, Daichi Mukunoki
(imamura.toshiyuki@rikenjp)
Thanks to Shuhei Kudo, Keigo Nitadori, and
Takuya Ina, Center for Computational Science, RIKEN

SIAM LA21, 17-21 May. 2021 @Zoom virtual/New Orleans

Outline

- **As a mixed-precision computing platform**
  - Fugaku system configurations
  - Mixed-precision on scientific code and benchmark at R-CCS
- **2.0 EXA flop/s benchmarking on HPL-AI (AI=Accelerator Introspection, AI≠Artificial Intelligent)**
  - Numerical experiments on Fugaku
    - Performance analysis, and Ever largest benchmark result in Nov 2020



?

?

Supercomputer Fugaku

- **A new Japanese flagship supercomputer successor to K**
  - officially launched since March 2021.
  - targeting traditional HPC, AI, and anywhere in the Society 5.0 fields
  - 30~40MW range in operation
- **CPU: A64FX (158,976 in total)**

  ARMv8.2+SVE(512bit x 2pipes) +clock(2.0/2.2GHz) and HBM2(32GB, 1TB/s)

  48cores(+2/4 assistant cores)/CPU
- **Interconnect: TofuD**
  - the network interface embedded in the A64FX microprocessor
  - injection bandwidth per node equaling 40.8 GB/s
  - low-latency communication with hardware-offloaded communication capabilities

- **Through tough benchmark tests**
  - We want to demonstrate capability for mixed-precision computation

  → Indirectly, new AI workload

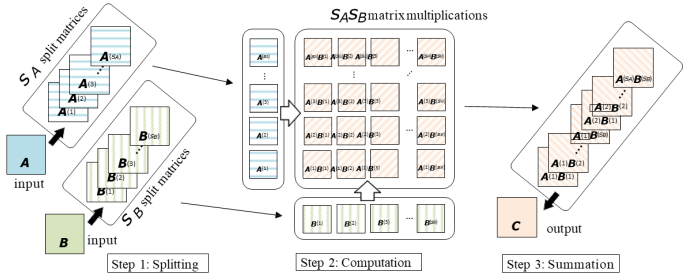
	K	Fugaku	Summit
FP64	10.62PF	537PF	200PF
FP32	10.62PF	1.07EF	400PF
FP16	--	2.15EF	3.3EF
INT8	--	4.30EO	--

Achievements in RIKEN, up to now

- **We have learned or claimed the necessity of higher precision due to concern of huge computing results**
- **Computational speed vs Computational accuracy**
  - generally considered to be conflicting.
- **The diversity and enhancement of hardware and the high productivity of software**
  - Allows users to choose the precision within the requirement of appropriate computational accuracy. These may provide us with enormous changes in scientific and technical computing, whereas it has been dominated by double-precision calculation for a long time.

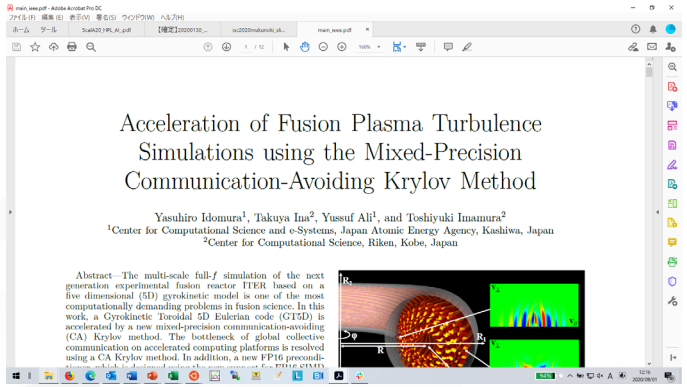
1. establishment of higher precision software by massively-and-high-performance low-precision computing units (ISC20)
2. algorithmic advancement of lower-precision units in scientific computing like HPL-AI benchmark (ScalA20/SC20, top500)
3. idea of minimal-precision computing (JLESC, CRE19).

Daichi Mukunoki, Katsuhisa Ozaki, Takeshi Ogita, Toshiyuki Imamura: DGEMM using Tensor Cores, and Its Accurate and Reproducible Versions, ISC High Performance 2020, Lecture Notes in Computer Science, Vol. 12151, pp. 230-248, Jun. 2020



$C = AB$ by Ozaki scheme (original version, not for Tensor Cores)

- **Step1 Splitting** – an input matrix into several split matrices
- **Step2 Computation** – computation of all-to-all multiplications of split matrices
- **Step3 Summation** – summation of the above multiplication results



Acceleration of Fusion Plasma Turbulence Simulations using the Mixed-Precision Communication-Avoiding Krylov Method

Yasuhiro Idomura[1], Takuya Ina[2], Yussuf Ali[1], and Toshiyuki Imamura[2]
[1]Center for Computational Science and e-Systems, Japan Atomic Energy Agency, Kashiwa, Japan
[2]Center for Computational Science, Riken, Kobe, Japan

Abstract—The multi-scale full-f simulation of the next generation experimental fusion reactor ITER based on a five dimensional (5D) gyrokinetic model is one of the most computationally demanding problems in fusion science. In this work, a Gyrokinetic Toroidal 5D Eulerian code (GT5D) is accelerated by a new mixed-precision communication-avoiding (CA) Krylov method. The bottleneck of global collective communication on accelerated computing platforms is resolved using a CA Krylov method. In addition, a new FP16 precondi-

Yasuhiro Idomura, Takuya Ina, Yussuf Ali, Toshiyuki Imamura: Acceleration of fusion plasma turbulence simulations using the mixed-precision communication-avoiding krylov method. SC 2020: 93

Shuhei Kudo, Keigo Nitadori, Takuya Ina, Toshiyuki Imamura: Implementation and Numerical Techniques for One EFlop/s HPL-AI Benchmark on Fugaku. ScalA@SC 2020: 69-76

HPL-AI, our Motivation and Main contributions

- **Almost full system size and speed benchmark on Fugaku**
  - 126,720 nodes (5/6 system), and 91%~>95% of the peak
- **Performance beyond two Exa Flop/s**
  - Updated World record in floating point benchmarking

- **Potential of Mixed-precision computing FP16-32-64, etc.**
- **Indirect examination to AI workload**
- **Learn mixed-precision benchmark via the HPL-AI benchmark**
  - We learned a lot via preliminary analysis
  - Several techniques (numerical and implementation):
    - Less iteration by single-iteration iterative refinement
    - Numerical stability by Scaling and reordering
    - Reduction of memory footprint
    - HW-supported asynchronous communication

HPL-AI and the HPL-AI matrix

- **The HPL-AI rule (Nov. 2019)**

  Solving $Ax = b$ by

  - the LU factorization of a matrix, and
  - the iterative refinement method (IR).

  LU: $A = LU$

  - Must consist of $2/3n^3 + O(n^2)$ flops.
  - **Not critical on accuracy/precision at this stage**

  IR: $x^{(n+1)} = x^{(n)} + A^{-1}(b - Ax^{(n)})$

  - solves the system of linear equations **with 64bit-precision accuracy**
  - tweak the IR algorithm without restriction, but IR should use the LU factors.

  - The HPL-harness must be less than 16 until 49 iterations

$$\frac{\|Ax - b\|_\infty}{\|A\|_\infty \|x\|_\infty + \|b\|_\infty} \times (n \cdot \varepsilon)^{-1} < 16$$

- **HPL matrix vs. HPL-AI matrix**

  [HPL] a randomly generated matrix.

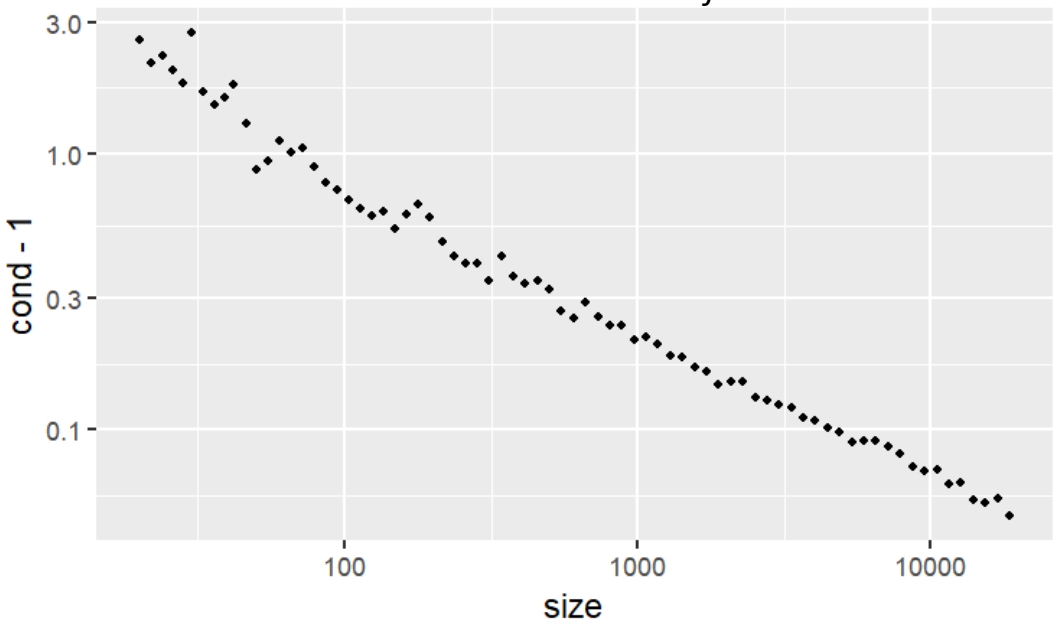
  On the other hand,

  [HPL-AI] diag = an absolute sum of off-diagonals :: weakly diagonally dominant

  Diagonal: O(*n*), off-diagonal: O(1)

  → Not necessary pivoting!

Estimated conditioned numbers of HPL-AI matrices



Significant techniques in Numeric

- **Three-precision mixed computing**

Require: An $n \times n$ matrix A, a vector b.
Ensure: The solution of the linear equations x, the LU factors L and U.

preprocess
- (FP64) $D \leftarrow \mathrm{diag}(A)$, the diagonal part of A.
- (FP64) $x \leftarrow D^{-1}b$, the initial guess.

LU decomp.
- (FP32&16) Factorize $A = LU$ using the mixed-precision algorithm.

while the backward-error (HPL-harness) > 16 **do** (IR iter.)
- (FP64) $r \leftarrow b - Ax$, the residual.
- (FP64) Solve $LUd = r$.
- (FP64) $x \leftarrow x + d$.

end while

IEE754 half precision (binary16/FP16)

Sign 1bit Frac. 10bit

Exp. 5bit

$O(n) + O(1) \approx O(n)$ when $n > 2^{10} = 1024$

- **Two notes on FP16 implementation**
  - [Scaling]
    - Value range on FP16 arithmetic is quite narrow, thus, we should avoid over/under-flows by appropriate scaling
  - [Ordering]
    - To prevent meaningless operations due to rounding, the order of operations should be devised

Scaling in LU factorization

$$A_{i,j}^{(m)} := s^{-1}\left(-\sum_{k=1}^{m}(sL_i^{(k)})U_j^{(k)}\right) + A_{i,j}^{(0)}$$

$$s = n^{\frac{3}{4}},\ \|sL_i^{(k)}\|_{\max} \approx O(n^{-1/4})$$

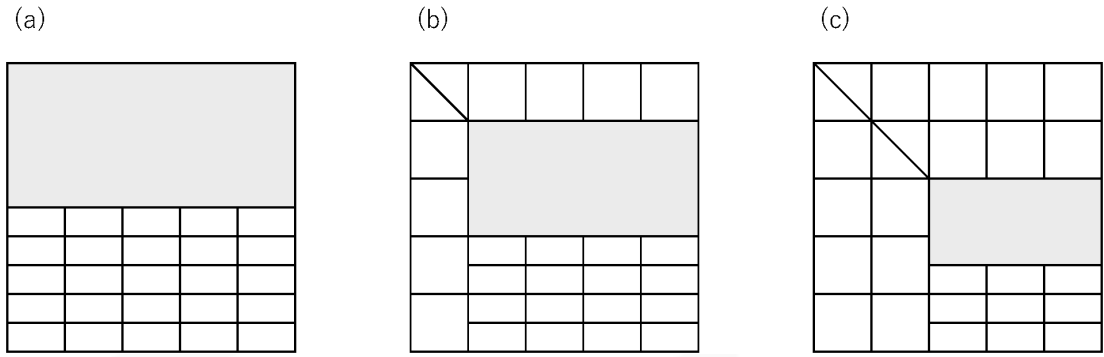
Basic idea of our ordering in LU factorization

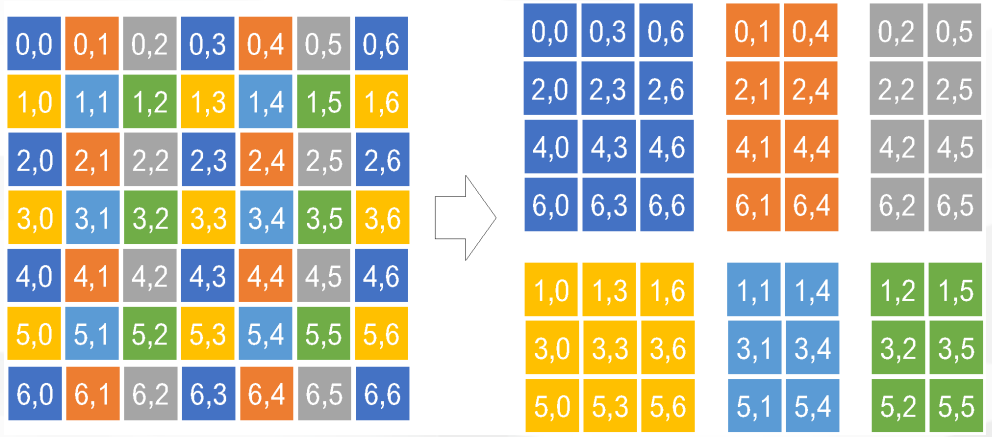
$$A_{i,j}^{(m)} := A_{i,j}^{(0)} - \sum_{k=1}^{m} L_i^{(k)} U_j^{(k)}$$

$$\Rightarrow\quad A_{i,j}^{(m)} := \left(-\sum_{k=1}^{m} L_i^{(k)} U_j^{(k)}\right) + A_{ij}^{(0)}$$

Significant techniques in Implementation

- **On-the-fly technique**
  - The matrix elements are generated in the on-the-fly manner during the LU and IR phases to **avoid the memory consumption of the FP64 matrix, and Lazy initialization as well.**
- **Double-decker layout for multi-data formats**
  - The memory regions of the **FP16 and FP32** matrices are laid out in an **overlapping** fashion
- **MPI parallelization with HW-offloaded communications**
  - The patterns of computation and communication are the same as in HPL, except for the non-obligatory pivoting.
  - Most of the communication is **asynchronously offloaded to TofuD (uTofu function),** thereby extending the overlapping part with a tiny overhead and latency.





Top500 #1 : updated in November 2020

- **2.0 EHFLop/s (approximately 93% of the theoretical peak).**
  - 158,976 nodes of the boosted full-Fugaku system
  - The target matrix size is 16,957,440, and the computing time required was 1,621 sec.
  - Scored more than 3.6 times compared to that on the Summit system at #2 spot.

- **Ever fastest benchmark result, and big update in the HPC community**
  - The world's first achievement **two** exascale in a floating-point arithmetic benchmark.

```
done MPI_Init_thread, provided = 2
#MPI_Init_thread: Thu Oct  1 08:18:45 2020
!REMAP FOR 414x1 RACKS
jobid=1638621
n=16957440 b=320 r=552 c=288
2dbc lazy rdma full pack nocheck noskiplu
numasize=1 numamap=CONT2D nbuf=3
epoch_size = 2119680
#BEGIN: Thu Oct  1 08:18:46 2020
!epoch 0/8: elapsed=0.000118, 0.000000 Pflops (estimate)
!epoch 1/8: elapsed=525.338805, 2042.522532 Pflops (estimate)
!epoch 2/8: elapsed=921.380582, 2039.728144 Pflops (estimate)
!epoch 3/8: elapsed=1206.298329, 2036.928902 Pflops (estimate)
!epoch 4/8: elapsed=1398.540852, 2033.866710 Pflops (estimate)
!epoch 5/8: elapsed=1516.588164, 2030.456576 Pflops (estimate)
!epoch 6/8: elapsed=1578.735426, 2026.939616 Pflops (estimate)
!epoch 7/8: elapsed=1603.524786, 2023.321290 Pflops (estimate)
# iterative refinement: step=  0, residual=4.3445890820578020e-04
hpl-harness=153719.691976
# iterative refinement: step=  1, residual=2.3068849523470399e-11
hpl-harness=0.008161
#END__: Thu Oct  1 08:45:57 2020
1621.837122590 sec. 2004390930.516057014 GFlop/s resid =
2.306884952347040e-11 hpl-harness = 0.008161425
2004.390931 Pflops, 102.605268 %
```

Summary in mixed-precision computing on Fugaku

- **Investigated techniques via acceleration of several real-application code.**
  - Esp. solving linear equation to reduce memory traffics
  - Non-standard FP formats have been also investigated, or, emulation of higher precision arithmetic by lower-precision arithmetic are feasible on manycore or GPU.
- **Highlight of our research in HPL-AI**
  - 1.4Exa flop/s at ISC20 (June 2020), and **2.0Exa flop/s at SC20 (Nov.2020).**
  - Demonstrated the world's first Exa-flops computing in HPL-AI.
  - FP64+FP32+FP16, especially HGEMM(FP16) and non-pivoting
- **Energy issue:**
  - HPL and HPL-AI show similar tendency: Approximately **28-29MW**.

We will release HPL-AI code for Fujitsu platforms and partly Intel x86_64 systems.

Further Tech part:

S. Kudo, K. Nitadori, T. Ina and T. Imamura, "Implementation and Numerical Techniques for One EFlop/s HPL-AI Benchmark on Fugaku," Proc. ScalA 2020, doi: 10.1109/ScalA51936.2020.00014.

Power & energy analysis:

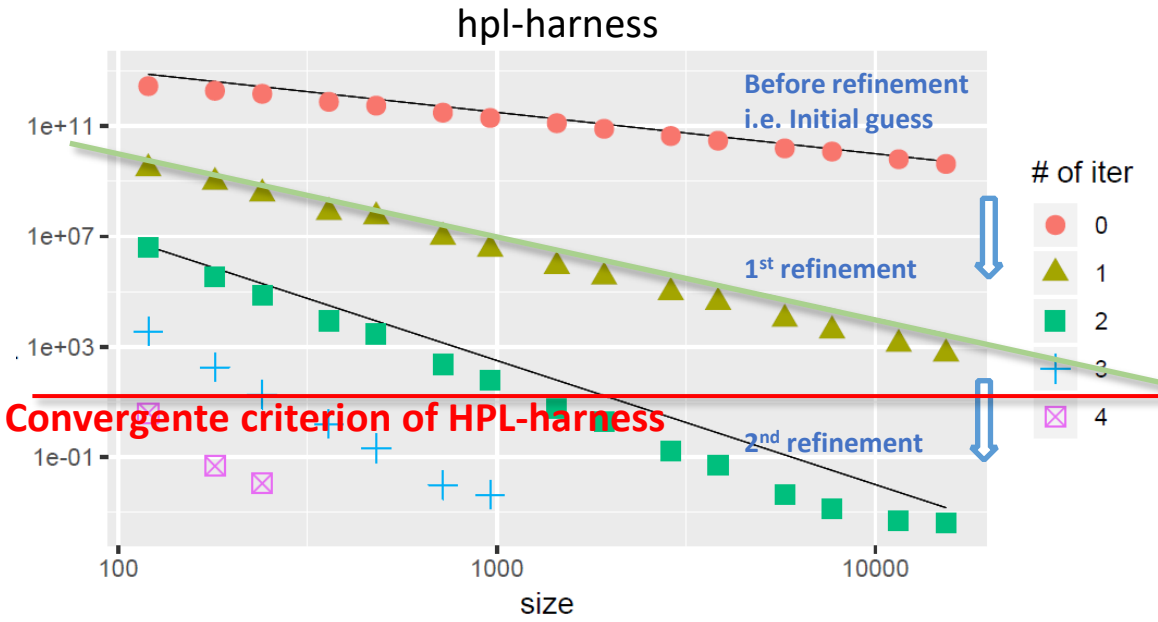
Kuroda et. al, Evaluation of Power Consumption of MLPerf HPC by Comparison HPL-AI on the Supercomputer Fugaku, 3rd R-CCS international Symposium poster presentation

Significant techniques in Numeric

- **SIIR (Single-Iteration for IR)**

  Good guess $x_0 := D^{-1}b$ enables to skip IR iterations, if quick factorize (approx.) as $A \approx L_J U_J = ID$.

  Only one iteration is enough, if n>100,000.



- **Lazy initialization (ordering the initial values come last="(Σ) first")**

  The large gap between the initial elements and the lower part like

  $$\{\text{diag}\} = O(n), \{\text{others}\} = O(1) \text{ and } \{\text{lower}\} \to O(1/n)$$

  causes information drops in summation when n>2^{24} in FP32 → FP16 is much^2 harder.

  HGEMM ← A blocked summation approach in FP16 to avoid satiation by accumulating the block result converted to FP32

$$A_{I,J}^{(n/b-1)} = \begin{cases} \text{lu}\left(A_{I,I}^{(0)} - \left(\sum_{k=1}^{I-1} A_{I,k}^{(k)} A_{k,J}^{(k)}\right)\right), & \text{if } I = J \\ \left(B_{I,J}^{(I)}\right)^{-1}\left(A_{I,J}^{(0)} - \left(\sum_{k=1}^{I-1} A_{I,k}^{(k)} A_{k,J}^{(k)}\right)\right), & \text{if } I < J \\ \left(C_{I,J}^{(J)}\right)^{-1}\left(A_{I,J}^{(0)} - \left(\sum_{k=1}^{J-1} A_{I,k}^{(k)} A_{k,J}^{(k)}\right)\right), & \text{otherwi} \end{cases}$$

$$B_{I,J}^{(I)} = \text{triu}\left(A_{I,I}^{(I)}\right), \text{ and } C_{I,J}^{(J)} = \left(\text{stril}\left(A_{J,J}^{(J)}\right) + I_b\right)$$

Improvement from June to Nov.

```
!REMAP FOR 330 RACKS
jobid=567937
n=13516800 b=320 r=1056 c=480
2dbc lazy rdma full pack
numasize=4 numamap=CONT2D nbuf=3
epoch_size = 1689600
#BEGIN: Fri May 15 08:03:02 2020
!epoch 0/8: elapsed=0.000321, 0.000000 Pflops (estimate)
!epoch 1/8: elapsed=370.133893, 1468.210388 Pflops (estimate)
!epoch 2/8: elapsed=650.033814, 1464.253280 Pflops (estimate)
!epoch 3/8: elapsed=851.582777, 1461.317344 Pflops (estimate)
!epoch 4/8: elapsed=988.277782, 1457.670705 Pflops (estimate)
!epoch 5/8: elapsed=1072.277294, 1454.437530 Pflops (estimate)
!epoch 6/8: elapsed=1117.625903, 1450.088542 Pflops (estimate)
!epoch 7/8: elapsed=1143.143441, 1437.409844 Pflops (estimate)
# iterative refinement: step=  0, residual=5.0168606685474515e-04
hpl-harness=222677.730166
# iterative refinement: step=  1, residual=1.5618974863462753e-11
hpl-harness=0.006931
#END__: Fri May 15 08:22:50 2020
1158.483898010 sec. 1421151817.927741528 GFlop/s resid =
1.561897486346275e-11 hpl-harness = 0.006931424
1421.151818 Pflops, 91.267070 %
```

```
done MPI_Init_thread, provided = 2
#MPI_Init_thread: Thu Oct  1 08:18:45 2020
!REMAP FOR 414x1 RACKS
jobid=1638621
n=16957440 b=320 r=552 c=288
2dbc lazy rdma full pack nocheck noskiplu
numasize=1 numamap=CONT2D nbuf=3
epoch_size = 2119680
#BEGIN: Thu Oct  1 08:18:46 2020
!epoch 0/8: elapsed=0.000118, 0.000000 Pflops (estimate)
!epoch 1/8: elapsed=525.338805, 2042.522532 Pflops (estimate)
!epoch 2/8: elapsed=921.380582, 2039.728144 Pflops (estimate)
!epoch 3/8: elapsed=1206.298329, 2036.928902 Pflops (estimate)
!epoch 4/8: elapsed=1398.540852, 2033.866710 Pflops (estimate)
!epoch 5/8: elapsed=1516.588164, 2030.456576 Pflops (estimate)
!epoch 6/8: elapsed=1578.735426, 2026.939616 Pflops (estimate)
!epoch 7/8: elapsed=1603.524786, 2023.321290 Pflops (estimate)
# iterative refinement: step=  0, residual=4.3445890820578020e-04
hpl-harness=153719.691976
# iterative refinement: step=  1, residual=2.3068849523470399e-11
hpl-harness=0.008161
#END__: Thu Oct  1 08:45:57 2020
1621.837122590 sec. 2004390930.516057014 GFlop/s resid =
2.306884952347040e-11 hpl-harness = 0.008161425
2004.390931 Pflops, 102.605268 %
```