Implementation and Numerical Techniques for One EFlop/s HPL-AI Benchmark on Fugaku

Shuhei Kudo, Keigo Nitadori, Takuya Ina, and Toshiyuki Imamura*
Center for Computational Science, RIKEN

11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems
Outline

- **Fugaku and Mixed-precision computing**
  - Fugaku system configurations
  - Mixed-precision on scientific code and benchmark

- **HPL-AI**
  - The rule defined in 2019
  - Preliminary analysis of the benchmark matrix
  - Numerical and Implementation techniques

- **Numerical experiments on Fugaku**
  - Performance analysis
  - Ever largest benchmark result in May 2020
A new Japanese flagship supercomputer successor to K
- the official launch available for public service is 2021
- targeting traditional HPC and AI

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 benchmark tests
- HPL-AI to demonstrate capability for mixed-precision computation
  → Indirectly, new AI workload

<table>
<thead>
<tr>
<th></th>
<th>K</th>
<th>Fugaku</th>
<th>Summit</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP64</td>
<td>10.62PF</td>
<td>537PF</td>
<td>200PF</td>
</tr>
<tr>
<td>FP32</td>
<td>10.62PF</td>
<td>1.07EF</td>
<td>400PF</td>
</tr>
<tr>
<td>FP16</td>
<td>--</td>
<td>2.15EF</td>
<td>3.3EF</td>
</tr>
<tr>
<td>INT8</td>
<td>--</td>
<td>4.30EO</td>
<td>--</td>
</tr>
</tbody>
</table>
For the linear solver

- Haidar et al. examined the feasibility of FP16 for the mixed-precision solver for well-conditioned matrices (ScalA17, SC18).
- Yamaguchi et al. Finite Element solver with FP21-32-64 (WACCPD/SC19)
- Idomura et al. SSOR preconditioner using FP16 (SC20)

Applications, Not only FP but also INT

- Neural Net Trained to Recognize Extreme Weather Patterns (GBP@SC18)
- Modified CoMet algorithm on TC’s to find fundamental biology (GBP@SC18)

Other potential mixed-precision studies, ...

- For example, Mukunoki et al. Ozaki scheme (JCAM), to emulate FP64 by TensorCores (ISC20)

Of course, we can not stop listing up more works in DNL.
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 one Exa Flop/s**
  - Achieved World record in floating point benchmarking

- Potential of Mixed-precision computing FP16-32-64, etc.
- Indirect examination to AI workload
- Learn of HP-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
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 $\frac{2}{3}n^3 + O(n^2)$ 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] 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 $U$.

  (FP64) $D \leftarrow \text{diag}(A)$, the diagonal part of $A$.
  
  (FP64) $x \leftarrow D^{-1}b$, the initial guess.
  
  (FP32&16) Factorize $A = LU$ using the mixed-precision algorithm.

  **while** the backward-error (HPL-harness) > 16 **do**

  (FP64) $r \leftarrow b - Ax$, the residual.
  
  (FP64) Solve $LUd = r$.
  
  (FP64) $x \leftarrow x + d$.

  **end while**

- **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
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)

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 $\Rightarrow$ Even FP16 is much harder.

HGEMM $\Leftarrow$ A blocked summation approach in FP16 to avoid satiation by accumulating the block result converted to FP32
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, thereby extending the overlapping part with a tiny overhead and latency.
Limited access during the installation phase of Fugaku.

1. **Micro Fugaku and Very Early access program** (Dec 2019 to March 2020)
2. **Early access program III:**
   - April to mid May 2020, up to 27,648 nodes.
3. **Open for privileged users**
   - May 8 and 9 and 14 and 15, up to 126,720 nodes.

- Run the code with **2.0GHz (normal mode)** in these periods.
- **4MPI processes per node →** 12threads per 1process
- **Fujitsu compiler and Fujitsu MPI environment**
- Matrix footprint equal to 23.7GiB (large) or 5.9GiB(tiny).

<table>
<thead>
<tr>
<th>Node</th>
<th>2D-Shape</th>
<th>Peak(DP)</th>
<th>Peak(HP)</th>
<th>N(tiny)</th>
<th>N(large)</th>
<th>NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>432</td>
<td>24x18</td>
<td>1.33PF</td>
<td>5.31PF</td>
<td>829,440</td>
<td>1,658,880</td>
<td>288/320/576/640</td>
</tr>
<tr>
<td>1,728</td>
<td>48x36</td>
<td>5.31PF</td>
<td>21.23PF</td>
<td>1,658,880</td>
<td>3,317,760</td>
<td>↓</td>
</tr>
<tr>
<td>6,912</td>
<td>96x72</td>
<td>21.23PF</td>
<td>84.93PF</td>
<td>3,317,760</td>
<td>6,635,520</td>
<td>↓</td>
</tr>
<tr>
<td>27,648</td>
<td>192x144</td>
<td>84.93PF</td>
<td>339.74PF</td>
<td>6,635,520</td>
<td>13,271,040</td>
<td>↓</td>
</tr>
</tbody>
</table>
Experiments on smaller cases (NB=576)

- Two matrix footprint cases
  - Both: >91% of the peak
  - Efficiencies between large and tiny settings → only 4%

- Suggestion
  - Even if it is a small case, the tiny setting is enough to confirm the performance of Fugaku, comparing the HPL result on Fugaku (~80%).

<table>
<thead>
<tr>
<th>Node</th>
<th>T (sec)</th>
<th>Pflop/s</th>
<th>Efct %</th>
</tr>
</thead>
<tbody>
<tr>
<td>432</td>
<td>78.6</td>
<td>4.83</td>
<td>91.13</td>
</tr>
<tr>
<td>1728</td>
<td>156.7</td>
<td>19.42</td>
<td>91.48</td>
</tr>
<tr>
<td>6912</td>
<td>312.6</td>
<td>77.88</td>
<td>91.70</td>
</tr>
<tr>
<td>27648</td>
<td>625.0</td>
<td>311.6</td>
<td>91.73</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>T (sec)</th>
<th>Pflop/s</th>
<th>Efct %</th>
</tr>
</thead>
<tbody>
<tr>
<td>432</td>
<td>603</td>
<td>5.05</td>
<td>95.12</td>
</tr>
<tr>
<td>1728</td>
<td>1203</td>
<td>20.24</td>
<td>95.32</td>
</tr>
<tr>
<td>6912</td>
<td>2403</td>
<td>81.07</td>
<td>95.45</td>
</tr>
<tr>
<td>27648</td>
<td>4807</td>
<td>324.1</td>
<td>95.40</td>
</tr>
</tbody>
</table>

https://www.top500.org/lists/top500/list/2020/06/
Preliminary Benchmark result

- Benchmarked on a limited part of Fugaku (2.0GHz normal mode)

<table>
<thead>
<tr>
<th>Node</th>
<th>Shape</th>
<th>N(tiny)</th>
<th>N(large)</th>
<th>Block size</th>
</tr>
</thead>
<tbody>
<tr>
<td>6,912</td>
<td>96x72</td>
<td>3,317,760</td>
<td>6,635,520</td>
<td>288/320/576/640</td>
</tr>
</tbody>
</table>

77.88PFlop/s 91.70 %

81.07PFlop/s 95.45 %

- We can read time breakdown:
  - HGEMM: 82~97%
  - WAIT (load imbalance) 2~18%
  - Depending on the place on PE grid

Block LU

Top500 #1 record at June 2020

- **1.421 EHFLop/s** (approximately 91% of the theoretical peak).
- 126,720 nodes of the non-boosted Fugaku system (5/6 of the full system)
- The target matrix size is 13,516,800, and the computing time required was 1,158 sec.
- Scored more than 2.5 times of that on the Summit system.

- Ever fastest benchmark result, and
- Big epoch in the HPC community
- The world’s first achievement to exceed the wall of exascale in a floating-point arithmetic benchmark.
Analysis and Discussions

- **Performance**
  - Well optimized. 91.27% in total \(\leftarrow\leftarrow\) HGEMM performs \(>95\%\) of the peak
  - Load-imbalance = 2–18% (in the case of the tiny setting)
    - Mostly caused by data dependency of the LU decomposition.
    - **Broadcast operations** \(\rightarrow\) offloaded onto the TofuD functionality.
  - **Performance bottleneck**
    - A64FX has an on-die communication control unit \(\rightarrow\) simple arch. and extremely low latency
    - for 8B put latency is \(0.49-0.54[\mu s]\), and 1MiB throughput attains \(6.35[\text{GB/s}]\).

- **On the other hand, case of the Summit code**
  - Not sure for implementation of the partial pivoting: ours has no pivoting
  - Tensor-Core’s computational performance is extremely high,
  - However, latency between NIF and GPU memory is large.

\(\rightarrow\) Reasons for the large score gap between Summit and Fugaku.
We investigated three numerical and four implementation techniques of the HPL-AI benchmark

- Reported what we have learned, and the real tests on the supercomputer Fugaku.
- Preliminary numerical analysis and implementation works, and we never seen numerical instability and inaccurate results.

Our HPL-AI implementation achieved >1Eflops

- demonstrated the world’s first Exa-flops computing, 1.421 EFlop/s.
- We intended to make our HPL-AI code open source soon.
- Only focused on performance (speed or flop/s), but other metrics are also significant. We should analyze and report them very shortly.

We will update the result using the full Fugaku system in SC week.
Acknowledgement

- We sincerely express our grateful acknowledgment to
  - Prof. Satoshi Matsuoka, Director R-CCS,
  - The Flagship 2020 project,
  - Dr. Yutaka Ishikawa, Dr. Mitsuhisa Sato, Dr. Fumiyoshi Shoji, and Fujitsu company.

- The results obtained on the evaluation environment in the trial phase do not guarantee the performance, power and other attributes of the supercomputer Fugaku at the start of its operation.