Offloading Integer GMRES Method to Accelerators

Yijie YU
yuyijie@g.ecc.u-tokyo.ac.jp
The University of Tokyo
Kashiwa, Chiba, Japan

Toshihiro Hanawa*
hanawa@cc.u-tokyo.ac.jp
The University of Tokyo
Kashiwa, Chiba, Japan

Acknowledgment:
This research was partially supported by JSPS 18H03246, 19H04122. The authors also thank Intel DevCloud for the FPGA experiment environment.

Introduction

- **Problems:** Over-allocation of bits
- **Solutions:** Transprecision

-Computing with appropriately lower precision

- **Pros:** Simultaneous calculations increase
- **Attentions:** Confirm the required precision !

Examples of using mixed precision

Benchmark of supercomputer

HPL Only 64 bits precision

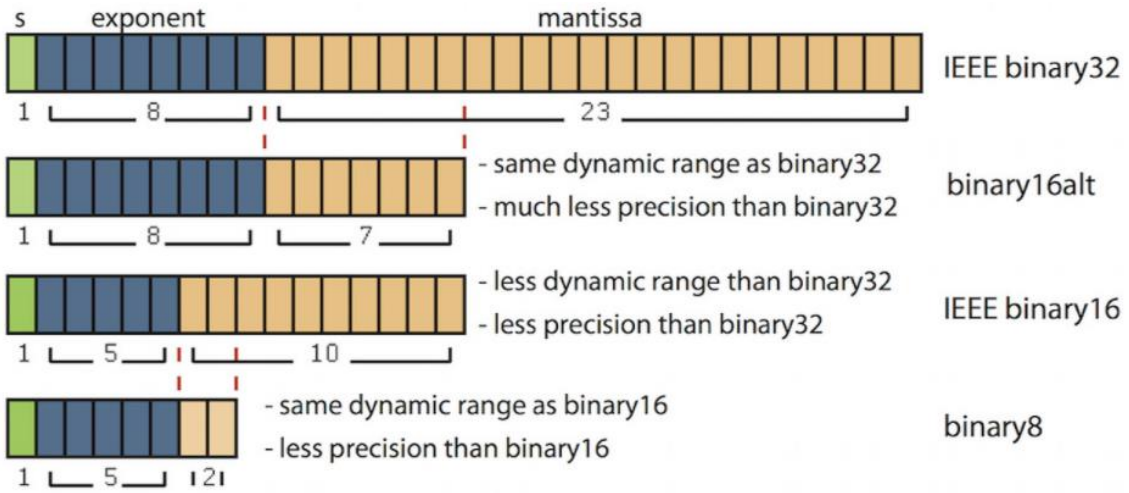
HPL-AI Support lower precision

HPL-AI in Fugaku

Three precision types (FP16,FP32,FP64)

↓

One EFlop/s !



NVIDIA A100 Tensor Core GPU Performance Specs

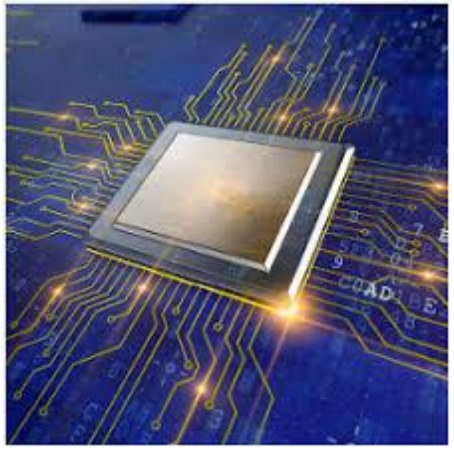
Peak FP64[1]	9.7 TFLOPS
Peak FP64 Tensor Core[1]	19.5 TFLOPS
Peak FP32[1]	19.5 TFLOPS
Peak FP16[1]	78 TFLOPS
Peak BF16[1]	39 TFLOPS
Peak TF32 Tensor Core[1]	156 TFLOPS | 312 TFLOPS[2]
Peak FP16 Tensor Core[1]	312 TFLOPS | 624 TFLOPS[2]
Peak BF16 Tensor Core[1]	312 TFLOPS | 624 TFLOPS[2]
Peak INT8 Tensor Core[1]	624 TOPS | 1,248 TOPS[2]
Peak INT4 Tensor Core[1]	1,248 TOPS | 2,496 TOPS[2]

1 - Peak rates are based on GPU Boost Clock.
2 - Effective TFLOPS / TOPS using the new Sparsity feature

- **Motivation**

-Using transprecision in large-scale math problems through offloading to accelerators ⟶ improve performance

- **One of test workload: integer-GMRES Algorithm**
- **Target platform: Intel oneAPI & OpenCL**



Offloading Principal

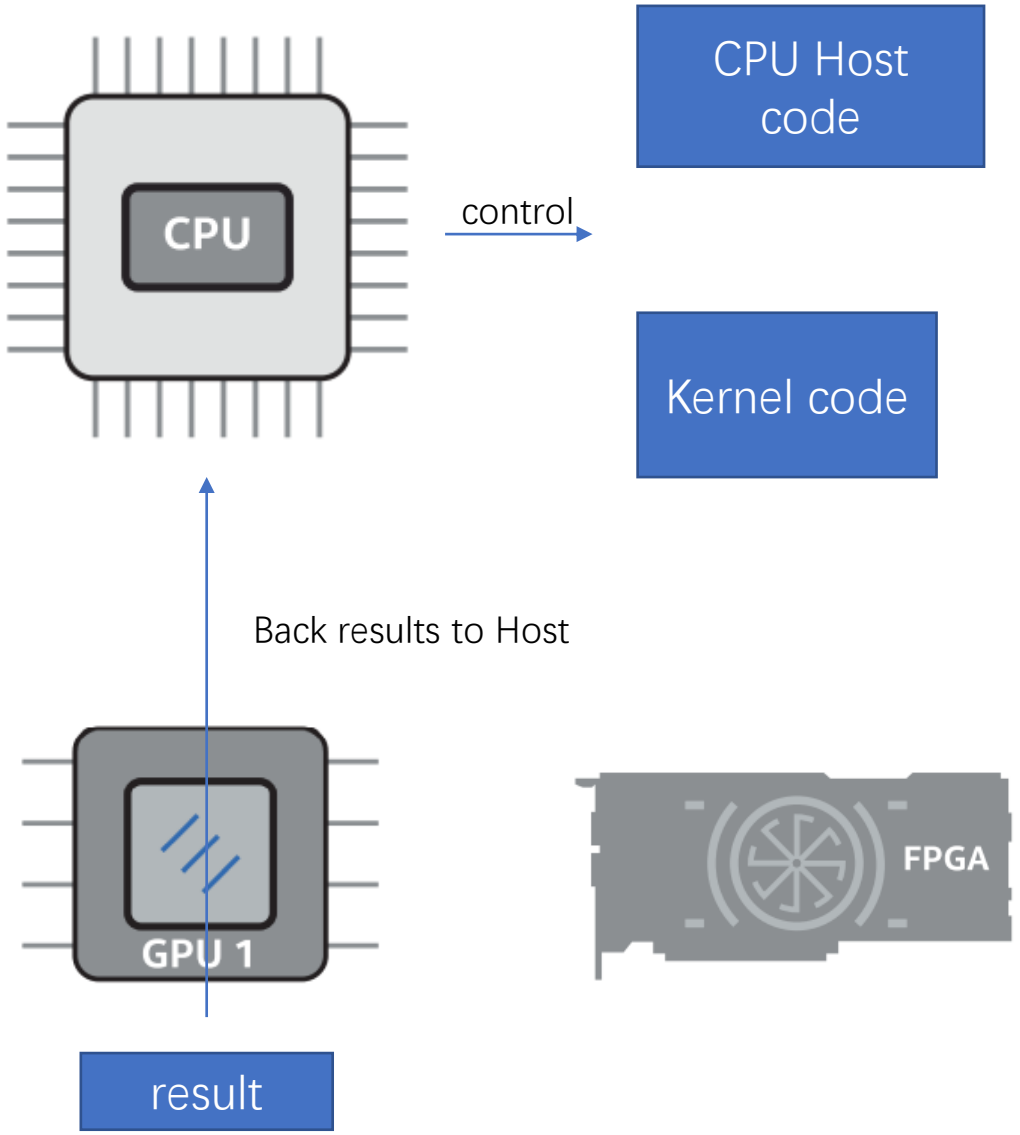
- **Offloading purpose**

-Overcome limitation of CPU-only systems

-Accelerators deliver better performance in HPC systems

- **Pros of Accelerators**
- GPUs: parallelism of graphics processing units
- FPGAs: reconfigurability

& optimize arithmetic units and data types



Intel oneAPI

-Based on modern C++ and SYCL
-DPC++: Support one source code

INT-GMRES Algorithm

- **GMRES Introduction**

-Krylov subspace method

- **why integer calculation is introduced?**

-FP arithmetic are power consuming

-FPGA can support light-weight integer arithmetic

- **Kernels of the GMRES**

-Inner product, Norm and Matrix Vector Multiplication

- **Int-GMRES expectations**

-Provides almost the same performance as norm GMRES

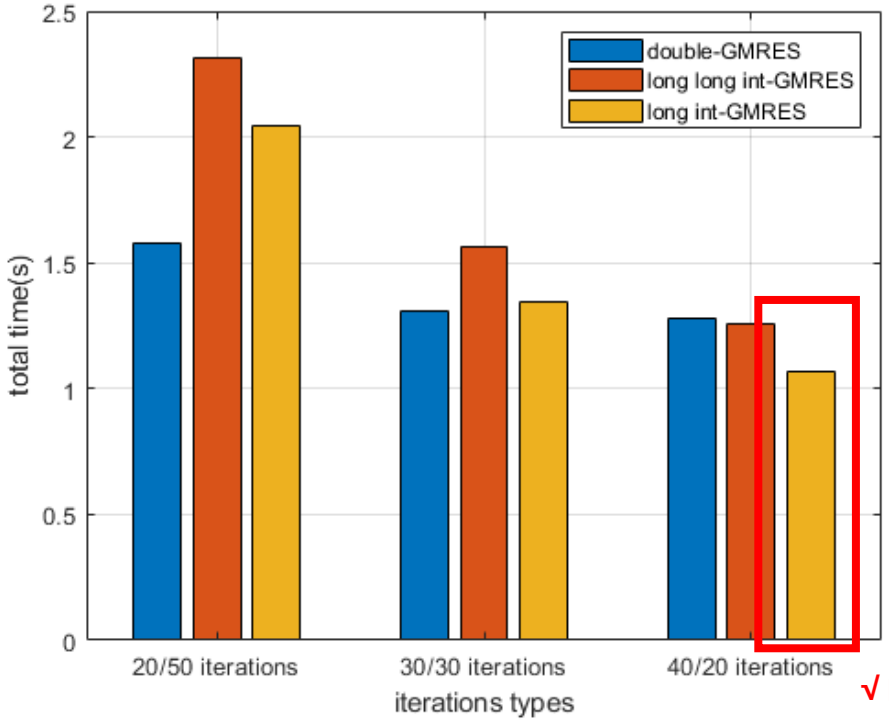
1. Compute $\boldsymbol{r}_0 = \boldsymbol{b}^{(k)} - \bar{\boldsymbol{A}}^{(k)}\boldsymbol{x}^{(k)}$, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$ // (FP)
2. Cast $\boldsymbol{v}_1$ to $\bar{\boldsymbol{v}}_1$
3. $\bar{\boldsymbol{g}} = (1, 0, \ldots, 0)^\top$
4. For j=1, 2, …, m
5. Compute $\bar{\boldsymbol{w}}_{j+1} = \bar{\boldsymbol{A}}^{(k)}\bar{\boldsymbol{v}}_j$ // (INT)
6. For $i = 1, \ldots, j$
7. $\bar{h}_{i,j} = (\bar{\boldsymbol{w}}_{j+1}, \bar{\boldsymbol{v}}_i)$ // (INT)
8. $\bar{\boldsymbol{w}}_{j+1} = \bar{\boldsymbol{w}}_{j+1} - \bar{h}_{i,j}\bar{\boldsymbol{v}}_i$ // (INT)
9. Endfor
10. $\bar{h}_{j+1,j} = \|\bar{\boldsymbol{w}}_{j+1}\|$ // (INT)
11. $\bar{\boldsymbol{v}}_{j+1} = \bar{\boldsymbol{w}}_{j+1}/\bar{h}_{j+1,j}$ // (INT)
12. For $i = 1, \ldots, j-1$
13. $\begin{pmatrix} \bar{h}_{i,j} \\ \bar{h}_{i+1,j} \end{pmatrix} = \begin{pmatrix} \bar{c}_i & \bar{s}_i \\ -\bar{s}_i & \bar{c}_i \end{pmatrix} \begin{pmatrix} \bar{h}_{i,j} \\ \bar{h}_{i+1,j} \end{pmatrix}$ // (INT)
14. Endfor
15. $\bar{t}_{mp} = \sqrt{\bar{h}_{j,j}^2 + \bar{h}_{j+1,j}^2}$ // (INT)
16. $\bar{c}_j = \frac{\bar{h}_{j,j}}{\bar{t}_{mp}}$, $\bar{s}_j = \frac{\bar{h}_{j+1,j}}{\bar{t}_{mp}}$ // (INT)
17. $\bar{g}_j = \bar{c}_j * \bar{g}_j$, $\bar{g}_{j+1} = -\bar{s}_j * \bar{g}_j$ // (INT)
18. $\bar{h}_{j,j} = \bar{t}_{mp}$
19. $\bar{h}_{j+1,j} = 0$ // (INT)
20. Endfor
21. Cast $\bar{\boldsymbol{g}}$ to $\boldsymbol{g}$, and $\bar{\boldsymbol{v}}_i$ to $\boldsymbol{v}_i$
22. $\boldsymbol{y} = \|\boldsymbol{r}_0\|\boldsymbol{H}_m^{-1}\boldsymbol{g}$ // (FP)
23. $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k)} + \sum_{i=1}^{j} y_i \boldsymbol{v}_i$ // (FP)

[2] Takeshi Iwashita, Kengo Suzuki, and Takeshi Fukaya. 2020. An Integer ArithmeticBased Sparse Linear Solver Using a GMRES Method and Iterative Refinement. In 2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA). IEEE, 1–8

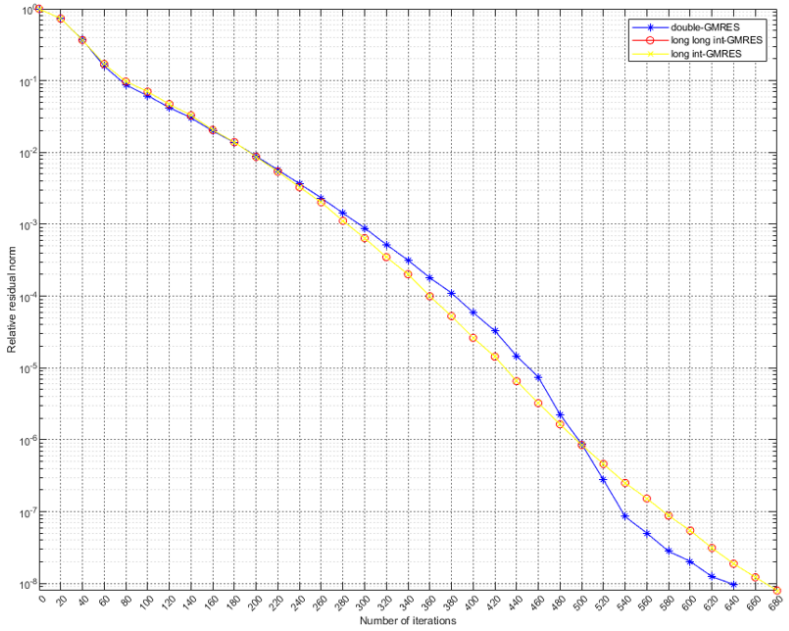
Host CPU Experiment: Original precision & Modified precision

Environment: Oakbridge-CX system

Item	Description
CPU	Intel® Xeon Platinum 8280 (code name: Cascade Lake) x 2
Memory	2,933 RDIMM 16 GB x 12 = 192 GB
Network	Intel® Omni-Path Host Fabric Interface (HFI) 12.5 GB/s
SSD (CX2560 M5)	1.6TB(NVMe) Read:3.20 GB/s, Write:1.32GB/s



40/20 iterations



√ Best performance

Modified long int type (less bits)

Fewer single iterations times (Less redundant computing)

→ **reduce computation time**

Porting GMRES Kernel Code

- **Experimental objectives**

-Porting kernel to OpenCL and oneAPI

-Complete the host code

Kernel part

-Assign threads (work items)

-Parallel for nested loop in GMRES

-Back results to the CPU host

Inner product kernel in OpenCL

```
/*d_inner_product(&V[i*N], &V[(j+1)*N], &H[j*(istep+1)+i], N);*/
float temp = 0;
temp += V[i*N+id] * V[(iteration+1)*N+id];
H[iteration*(istep+1)+i] = temp;
barrier(CLK_LOCAL_MEM_FENCE);

/*d_vec_minus_cons_vec(&V[(j+1)*N], &V[i*N], &H[j*(istep+1)+i], N);*/
V[(iteration+1)*N+id] -= H[iteration*(istep+1)+i] * V[i*N+id];
barrier(CLK_LOCAL_MEM_FENCE);
```

oneAPI DPC++ matrix multiplication kernel

```
  // Submit command group to queue to multiply matrices: c = a * b
  q.submit([&](auto &h) {
    // Read from a and b, write to c
    accessor a(a_buf, h, read_only);
    accessor b(b_buf, h, read_only);
    accessor c(c_buf, h, write_only);

    int width_a = a_buf.get_range()[1];

    // Execute kernel.
    h.parallel_for(range(M, P), [=](auto index) {
      // Get global position in Y direction.
      int row = index[0];
      // Get global position in X direction.
      int col = index[1];

      float sum = 0.0f;

      // Compute the result of one element of c
      for (int i = 0; i < width_a; i++) {
        sum += a[row][i] * b[i][col];
      }

      c[index] = sum;
    });
  });
} catch (sycl::exception const &e) {
  cout << "An exception is caught while multiplying matrices.\n";
  terminate();
}
```

Future work & Reference

Future work

- Continue to optimize the code to ensure that we can offload the kernel to the FPGA hardware by using oneAPI tool
- Improving transprecision algorithms to reduce unnecessary computation
- Developing engines that allow arbitrary and multiple length precision depending on tasks

Reference

[1] JA Hittinger, PG Lindstrom, H Bhatia, PT Bremer, DM Copeland, KK Chand, AL Fox, GS Lloyd, H Menon, GD Morrison, et al. 2019. Variable Precision Computing. Technical Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

[2] Takeshi Iwashita, Kengo Suzuki, and Takeshi Fukaya. 2020. An Integer ArithmeticBased Sparse Linear Solver Using a GMRES Method and Iterative Refinement. In 2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA). IEEE, 1–8