User Manual for Ultra Data-Oriented Parallel Fractional Hot-Deck Imputation (UP-FHDI)

Yicheng Yang and In Ho Cho

1. INTRODUCTION

UP-FHDI is a general-purpose, distributional assumption-free imputation program for handling multivariate missing data by filling each missing item with multiple observed values without resorting to artificially created values. UP-FHDI can be segmented into four stages: cell construction, estimation of joint probability, hot-deck imputation, and variance estimation. This manual provides a comprehensive illustration of how UP-FHDI can be deployed, complied, and performed for big incomplete data curing.

This manual is organized as follows: Section 2 explicitly illustrates available options of input configurations and proper setups regarding users' demands. Section 3 provides descriptions and interpretations of outputs. We demonstrate the execution of UP-FHDI step by step in Section 4. Finally, Section 5 elaborates example applications of UP-FHDI to different types of incomplete datasets.

We share developed codes, examples, and documents with GPL-2 (Cho, 2021). For users' benefit, various synthetic and real-world incomplete datasets are publicly accessible in IEEE DataPort (Yang, Kim, & Cho, 2021). If users require more information about UP-FHDI, please refer to full details in (Yang, Kwon, Kim, & Cho, 2021) and (Yang, Kim, & Cho, 2020).

2. CONFIGURATIONS OF INPUT FILES

Let **y**, **r**, **z** denote incomplete raw data, response indicator matrix, and imputation cells, respectively. Let D be the total data volume of **y** and **r**. Input file requires nineteen arguments in total.

1. *i_option_ultra* (default: 1)

[0] – Perform P-FHDI

[1] – Perform UP-FHDI

2. *i_option_read_data* (default: 1)

Method	Mechanism	Option	Required Input Files	Scenario
P-FHDI	POSIX IO	[0]	*input.txt*	Small D
	POSIX IO	[1]	*input.txt*; *daty.txt* (**y**); *datr.txt* (**r**)	Large D
UP-FHDI	POSIX IO	[0]	*input.txt*; *daty.txt* (**y**); *datr.txt* (**r**)	Small D
	MPI IO	[1]	*input.txt*; *datr_column_binary.bin* (**r**); *daty_column_binary.bin* (**y**); *daty_row_binary.bin* (**y**)	Big D

Configurations of inputs are set up in *input.txt*. POSIX IO is strictly restricted by the available memory per processor, and MPI IO does not have any limitations on D. Note that binary files for MPI IO require strict data distribution for ease of efficient data import. The data distributions in "*daty_column_binary.bin*" and "*daty_row_binary.bin*" must be column-oriented and row-oriented, respectively. Data distribution in "*datr_column_binary.bin*" must be column-oriented. Notably, users must ensure input file names to match the exact names listed above.

3. *i_option_perform* (default: 1)

[1] – Perform all stages using the automatic **z**

[2] – Perform cell construction only

[3] – Perform estimation of cell probability only

[4] – Perform all stages using a user-defined **z**

P-FHDI supports all available options, but UP-FHDI only allows the option [1]. Note that P-FHDI must set *i_user_defined_datz* $= 1$ to adopt a user-defined **z** if *i_option_perform* $= 4$.

4. *nrow*

The number of instances in **y**

5. *ncol*

The number of variables in **y**

6. *i_option_imputation* (default: 2)

[1] − Perform the fully-efficient fractional imputation (FEFI)

[2] − Perform the fractional hot-deck imputation (FHDI)

P-FHDI supports all available options, but UP-FHDI only allows the option [2].

7. *i_option_variance* (default: 1)

[0] − Skip variance estimation

[1] − Perform variance estimation

8. *i_option_merge* (default: 0)

[0] − Turn on the fixed seed for reproducible results

[1] − Turn on the standard random seed generator

9. *i_donor* (default: 5)

Adopt a user-defined integer as the number of donors used to fill in each missing item

The past literature recommends five as the default value after detailed case studies.

10. *i_user_defined_datz* (default: 0)

[0] − Adopt the automatic **z**

[1] − Adopt a user-defined **z**

UP-FHDI does not allow the option [1]. P-FHDI must set *i_user_defined_datz* $= 1$ to adopt a user-defined **z** in case of *i_option_perform* $= 4$.

11. *i_option_collaspsing* (default: 4)

Activate the sure independent screening (SIS) by a user-defined number of selected variables

i_option_collaspsing > 0 activates SIS to reduce all variables to a user-defined number of selected variables. Otherwise, SIS is turned off by setting *i_option_collaspsing* $= 0$.

12. *i_option_SIS_type* (default: 3)

[1] − SIS based on an intersection of simple correlations

[2] − SIS based on a union of simple correlations

[3] − SIS based on a global ranking of simple correlations

i_option_SIS_type will switch different methods for SIS and is meaningful only if *i_option_collaspsing* > 0.

13. *top_correlation* (default: 1000)

Number of top-ranking variables based on simple correlations to be used for the extraction of selected variables in SIS

An optimal *top_correlation* $= \min(top\_correlation, ncol)$ will be the least value that guarantees extraction of *i_option_collaspsing* selected variables.

14. *i_option_cellmake* (default: 2)

$[1] -$ Adopt the cell collapsing method to generate artificial donors in cell construction

$[2] -$ Adopt the k-nearest neighbor (KNN) method to find deficient donors in cell construction

UP-FHDI does not allow the option $[1]$.

15. *i_option_var_type* (default: 2)

$[1] -$ Adopt the Jackknife variance estimation

$[2] -$ Adopt the linearized variance estimation

P-FHDI does not allow the option $[2]$. This parameter is meaningful only if *i_option_variance* $=$ 1.

16. *memory* (default: 8)

Available memory in gigabyte per MPI task $=$ Total memory in gigabyte per node $/$ Number of MPI tasks per node.

17. *category* (default: 3)

A vector indicates the number of total categories per variable. The maximum number of categories is 35 due to nine integers $(1-9)$ and twenty-six alphabet letters $(a-z)$.

18. *NonCollapsible_categorical* (default: 0)

A vector indicates non-collapsible categorical variables. Zero indicates a variable is collapsible categorical, or continuous. One indicates a variable is non-collapsible and categorical.

19. *weight* (default: 1)

Sampling weight for each instance of $\mathbf{y}$.

3. EXPLANATIONS OF OUTPUT FILES

Overall, UP-FHDI generates six output files that cover the most outputs of the R package FHDI. These files include results of cell construction, hot-deck imputation, and variance estimation of UP-FHDI, respectively. The data type of all UP-FHDI outputs is double. The length of a double value in a binary file is eight bytes. Note that data distributions in all binary output files are row-oriented. Let n and p denote the number of instances and the number of variables of raw input data $\mathbf{y}$.

1) Results of cell construction

Cell construction categorizes raw input data into imputation cells, consisting of unique observed patterns and unique missing patterns.

- *datz_binary.bin*: Imputation cells in the size of n row and p columns are included.
- *uox_binary.bin*: Unique observed patterns with p columns are included. Considering the total size of this binary file and the number of columns, users can compute the number of rows of unique observed patterns.
- *mox_binary*.bin: Unique missing patterns with p columns are included. Considering the total size of this binary file and the number of columns, users can compute the number of rows of unique missing patterns.

2) Results of hot-deck imputation
   Hot-deck imputation fills in missing items using selected donors with fractional weights and thus creates a single complete dataset.
   - *fmat_FHDI_binary.bin*: Imputation results with $4 + p$ columns, consisting of instance ID, sampling weights, FEFI fractional weights, FHDI fractional weights, and fractionally imputed data. Considering the total size of this binary file and the number of columns, users can compute the number of rows of imputation results.
   - *final_daty_binary.bin*: A single complete data in the size of n row and p columns.
3) Results of variance estimation

   Estimate variance of each variable using the user-defined method. The standard error can be computed by the square of variance.

   - *summary.data.txt*: mean estimates and variance estimates

If P-FHDI is enabled, it generates three TXT output files, including results from hot-deck imputation and variance estimation.

1) Results of hot-deck imputation
   - *fimp.data.txt*: Imputation results consisting of instance ID, donor ID, sampling weights, FHDI fractional weights, and fractionally imputed data.
   - *simp.data.txt*: A single complete data in the same dimensions with raw input data
2) Results of variance estimation
   - *summary.data.txt*: mean estimates and variance estimates

4. EXECUTION OF UP-FHDI

Users must run UP-FHDI on high-performance computing (HPC) facilities, e.g., Condo 2017 (Iowa State University, 2021) and Stamepde2 (Texas Advanced Computing Center, 2021). HPC aggregates powerful computing that deliveries much higher performance than stand-alone computers. We illustrate how to run UP-FHDI on different platforms as follows.

4.1 Run on Condo Cluster

Condo Cluster is a free HPC platform open to all Iowa State University faculties and their groups, consisting of 134 nodes, 128 GB of memory, and 2.5 TB of local disk. Illustration on how to access Condo is openly accessible in (Sarkar, 2021a). Please refer (Iowa State University,

2021) for full details about Condo Cluster. Execution of UP-FHDI on Condo consists of three steps.

1) Load required modules
   Intel compiler must be loaded to compile a C++ program by issuing:
   ```
   module load intel/18.3
   ```
   where 18.3 represents the recommended Intel compiler version on Condo.
2) Compile UP-FHDI program
   One can run the command:
   ```
   mpiicc –o main_MPI main_MPI.cpp
   ```
   where mpiicc can be invoked to compile an MPI program written in C++ if the Intel compiler was pre-loaded. As a result, the main function of UP-FHDI (main_MPI.cpp) is compiled to an exact executable file (main_MPI).
3) Create and submit a job script

   To submit a parallel-computing task to an HPC system, users need to prepare a job script for the job management system (i.e., Slurm). A job script contains setup information for the batch system followed by commands to be executed. Slurm Job Script Generator in (Iowa State University, 2017) can automatically generate Condo job scripts regarding users' computing resource requirements. An example job script (named as *run.sbatch*) on Condo should contain basic information below:

   ```
   #!/bin/bash

   #SBATCH --job-name=UP-FHDI          # Job name

   #SBATCH --nodes=1                   # Total number of nodes

   #SBATCH --ntasks=16                 # Total number of MPI tasks

   #SBATCH --time=00:10:00             # Maximum run time (hh:mm:ss)

   mpirun -np 16 ./main_MPI            # Launch MPI code
   ```

   The first line of a job script must specify the interpreter that will parse non-Slurm commands by issuing #!/bin/bash. Users can use #SBATCH directives to request computing resources, and these directives should precede all shell commands. Users can reserve computing resources by specifying the requested number of nodes, the number of MPI tasks per node, and a maximum run time. An appropriate request of necessary computing resources is of importance rather than requesting as much resource as possible since the Slurm will have an easier time finding a slot for your job. Overall, this example requires one compute node with sixteen MPI tasks and a ten-minute maximum running time. Finally, one can launch a single application by mpirun. Note that the number of requested processors (-np) must be consistent with the total number of MPI tasks (--ntasks). Then users can submit the created job script into the queue by issuing:

   ```
   sbatch run.sbatch
   ```

4.2 Run on Stampede2

Stampede2 is a flagship HPC facility at the University of Texas at Austin open to global researchers with awarded allocations. It hosts 4200 KNL compute nodes and 1736 SKX compute nodes. Each KNL node and SKX node have 96 GB and 192 GB of memory, respectively. Illustration on how to access Stampede2 is openly accessible in (Sarkar, 2021b). Please refer (Texas Advanced Computing Center, 2021) for more details about Stampede2. Execution of UP-FHDI on Stampede2 consists of two steps.

1) Compile UP-FHDI program
   Stampede2 automatically loads the recommended Intel compiler when users log in. Run the command:
   ```
   mpicxx –o main_MPI main_MPI.cpp
   ```
   where `mpicxx` is invoked to compile an MPI program written in C++. As a result, the main function of UP-FHDI (`main_MPI.cpp`) is compiled to an exact executable file (`main_MPI`).
2) Create and submit a job script
   Job script format varies between different HPC facilities. Stampede2 provides various example job scripts in (Texas Advanced Computing Center, 2021) for MPI jobs in the different Slurm partitions. An example job script (named as *run.sbatch*) on Stampede2 should contain basic information below:

   ```
   #!/bin/bash

   #SBATCH –J UP-FHDI        # Job name

   #SBATCH –o UP-FHDI.o%j    # Name of the output file

   #SBATCH –e UP-FHDI.e%j    # Name of the error file

   #SBATCH –p skx-normal     # Queue (partition) name

   #SBATCH –N 1              # Total number of nodes

   #SBATCH –n 24             # Total number of MPI tasks

   #SBATCH –t 00:10:00       # Maximum run time (hh:mm:ss)

   ibrun ./main_MPI          # Launch MPI code
   ```

   Job management system (i.e., Slurm) writes all console outputs to the file "UP-FHDI.o%j" and error messages to the file "UP-FHDI.e%j", where %j represents the numeric job ID. The directive "–p" specifies the used Slurm partition for your job. For example, "skx-normal" refers to SKX compute nodes and "normal" refers to KNL compute nodes. Please see (Texas Advanced Computing Center, 2021) for more currently available partitions on Stampede2. Overall, this example requires one SKX compute node with twenty-four MPI tasks and a ten-minute maximum running time. Finally, one can launch a single application by ibrun. Then users can submit the created job script into the queue by issuing:

```
sbatch run.sbatch
```

5. APPLICATIONS OF UP-FHDI TO BIG INCOMPLETE DATA

To maximize users' benefits, we provide example applications of UP-FHDI to different types of big incomplete data, i.e., big data with numerous instances (so-called big-n), big data with many variables (so-called big-p), and ultra data (concurrently big-n and big-p). We upload various incomplete datasets in IEEE DataPort (Yang, Kim, et al., 2021) for users to practice using UP-FHDI.

5.1 Big-n Data Application on Condo

This input data has one million instances, four variables, and a 30% missing rate. The application enables POSIX IO to import data and P-FHDI for data curing such that it requires a single input file (*input.txt*) and a job script (*run.sbatch*). The input file contains configurations, **y** (raw data), and **r** (response indicator matrix) below:

```
INPUT INFORMATION
i_option_read_data          0
i_option_ultra              0
i_option_perform            1
nrow                        1000000
ncol                        4
i_option_imputation         2
i_option_variance           1
i_option_merge              0
i_donor                     5
i_user_defined_datz         0
i_option_collapsing         0
i_option_SIS_type           3
top_correlation             1000
i_option_cellmake           1
i_option_var_type           1
memory                      8
END INPUT INFORMATION


daty
      0            0            2.517        -0.105061
   1.45522      0.522759       1.56393       -1.2951
  -0.799204     1.93609        2.6653        0.00410365
      ⋮            ⋮              ⋮              ⋮


datr
```

```
        0            0            1            1
        1            1            1            1
        1            1            1            1
        ⋮            ⋮            ⋮            ⋮


category
        3            3            3            3


NonCollapsible_categorical
        0            0            0            0


weight
1
1
1
⋮
END DATA INPUT
```

This input data is classified as a big-n data since the number of instances (*nrow* $= 1000000$) is much greater than the number of variables (*ncol* $= 4$). P-FHDI (*i_option_ultra* $= 0$) is particularly potent towards big-n data since the number of MPI tasks for UP-FHDI must be less than the number of variables, i.e., 4. Due to small data volume, the configurations enable POSIX IO (*i_option_read_data* $= 0$) to import all required data from a single TXT file. We perform all procedures of P-FHDI (*i_option_perform* = 1) including fractional hot-deck imputation (*i_option_imputation* $= 2$) and Jackknife variance estimation (*i_option_variance* $= 1$ & *i_option_var_type* $= 1$). P-FHDI allows the cell collapsing method (*i_option_cellmake* = 1) to generate artificial donors, and SIS is disabled (*i_option_collapsing* $= 0$) for big data with a few variables. The available memory per MPI task (*memory*) $= 128$ GB$/\ 16 = 8$ GB considering 128 GB memory of a Condo compute node. *NonCollapsible_categorical* are set to be zeros because all variables are continuous. We leave the rest of the parameters as default values. The job script file contains information on computing resources below:

```
#!/bin/bash

#SBATCH --job-name=UP-FHDI          # Job name

#SBATCH --nodes=2                   # Total number of nodes

#SBATCH --ntasks=32                 # Total number of MPI tasks

#SBATCH --time=10:00:00             # Maximum run time (hh:mm:ss)

mpirun -np 32 ./main_MPI            # Launch MPI code
```

Overall, this application requires two compute nodes with thirty-two MPI tasks and a ten-hour maximum running time. Users must ensure the input file, the job script, and UP-FHDI codes are in the same directory of HPC facilities. Users can download all required files in (Yang, Kim, et al., 2021). This input data is named Synthetic Data 2, belonging to the big-n category. Notably, we adopt a different naming convention for input files of Synthetic Data 2 in (Yang, Kim, et al., 2021). Users must modify input file names to match the exact names above.

Following procedures in Section 4.1, users need to load recommended Intel complier by:

```
module load intel/18.3
```

One can compile the main function of UP-FHDI by:

```
mpiicc –o main_MPI main_MPI.cpp
```

As a result, an exact executable file (main_MPI) will be generated. Finally, users can submit the created job script to launch the MPI application by:

```
sbatch run.sbatch
```

P-FHDI generates three output files. Imputation results are included in a TXT file (*fimp.data.txt*) below:

```
ID      FID     WT      FWT     y1          y2          y3          y4
1       1       1       0.2     1.74781     2.00483     2.517       -0.105061
1       2       1       0.2     1.63866     0.810845    2.517       -0.105061
1       3       1       0.2     0.680109    2.2576      2.517       -0.105061
1       4       1       0.2     2.61151     3.6176      2.517       -0.105061
1       5       1       0.2     1.29574     1.84862     2.517       -0.105061
2       1       1       1       1.45522     0.522759    1.56393     -1.2951
3       1       1       1       -0.799204   1.93609     2.6653      0.00410365
⋮       ⋮       ⋮       ⋮       ⋮           ⋮           ⋮           ⋮
```

Note that ID, FID, WT, FWT denotes instance ID, donor ID, sampling weights, and fractional weights, respectively. A single complete data is included in a TXT file (*simp.data.txt*) below:

```
1.59477       2.1079        2.517         -0.105061
1.45522       0.522759      1.56393       -1.2951
-0.799204     1.93609       2.6653        0.00410365
⋮             ⋮             ⋮             ⋮
```

The TXT file (*summary.data.txt*) contains mean estimates and variance estimates below:

```
Mean estimates
0.998926        2.00016         2.00045         -0.000442508
Variance results
1.3753e-06      1.1484e-06      2.05412e-06     1.97582e-06
s_op_imputation:
2
i_option_merge:
```

```
0
M:
5
```

In addition to mean and variance estimates, this file prints several important configurations for validations. Note that “s_op_imputation” is equivalent to “*i_option_imputation*” and M represents “*i_donor*” in configurations.

5.2 Big-*p* Data Application on Condo

This input data has a thousand instances, 10,000 variables, and a 30% missing rate. The application enables POSIX IO to import data and UP-FHDI for data curing such that it requires three input files and a job script (*run.sbatch*). The first input file (*input.txt*) contains basic configurations below:

```
INPUT INFORMATION
i_option_read_data                          0
i_option_ultra                              1
i_option_perform                            1
nrow                                        1000
ncol                                        10000
i_option_imputation                         2
i_option_variance                           1
i_option_merge                              0
i_donor                                     5
i_user_defined_datz                         0
i_option_collapsing                         4
i_option_SIS_type                           3
top_correlation                             1000
i_option_cellmake                           2
i_option_var_type                           2
memory                                      8
END INPUT INFORMATION


category
       3            3            3            3            ⋯


NonCollapsible_categorical
       0            0            0            0            ⋯


weight
1
1
1
⋮
```

```
END DATA INPUT
```

This input data is classified as a big-p data since the number of variables ($ncol = 10000$) is much greater than the number of instances ($nrow = 1000$). UP-FHDI ($i\_option\_ultra = 1$) is activated since it is very efficient to process high-dimensional data. This example enables POSIX IO ($i\_option\_read\_data = 0$) to import basic configurations from *input.txt*, raw data **y** from *daty.txt*, and response indicator matrix **r** from *datr.txt*, respectively. This option is often used for UP-FHDI with small data volume. We perform all procedures of UP-FHDI ($i\_option\_perform = 1$) including fractional hot-deck imputation ($i\_option\_imputation = 2$) and linearized variance estimation ($i\_option\_variance = 1$ & $i\_option\_var\_type = 2$). UP-FHDI only supports the KNN method ($i\_option\_cellmake = 2$) to find deficient donors. SIS is adopted to reduce high dimensions to four selected variables ($i\_option\_collapsing = 4$) based on a global ranking of simple correlations ($i\_option\_SIS\_type = 3$). The available memory per MPI task ($memory$) = 128 GB/ $16 = 8$ GB considering 128 GB memory of a Condo compute node. *NonCollapsible_categorical* are set to be zeros because all variables are continuous. We leave the rest of the parameters as default values. The second input file (*daty.txt*) contains raw data **y** below**:**

```
daty
      0          1.733795841    2.77845516    0.351834951     ⋯
 1.869890653     2.778282048    6.141678817   2.480226273     ⋯
  2.5180421      2.36430633     5.149706382        0          ⋯
      ⋮               ⋮              ⋮             ⋮           ⋯
```

The third input file (*datr.txt*) contains response indicator matrix **r** below**:**

```
datr
      0               1              1             1          ⋯
      1               1              1             1          ⋯
      1               1              1             0          ⋯
      ⋮               ⋮              ⋮             ⋮           ⋯
```

The job script file contains information on computing resources below:

```
#!/bin/bash
#SBATCH --job-name=UP-FHDI   # Job name
#SBATCH --nodes=1            # Total number of nodes
#SBATCH --ntasks=16          # Total number of MPI tasks
#SBATCH --time=00:20:00      # Maximum run time (hh:mm:ss)
mpirun -np 16 ./main_MPI     # Launch MPI code
```

Overall, this application requires a compute node with sixteen MPI tasks and a twenty-minute maximum running time. Users must ensure three input files, the job script, and UP-FHDI codes in the same directory of HPC facilities. Users can download all required files in (Yang, Kim, et al., 2021). This input data is named Synthetic Data 4, belonging to the ultra category. Notably, we adopt a different naming convention for input files of Synthetic Data 4. Users must modify input file names to match the exact names above.

Following procedures in Section 4.1, users need to load the recommended Intel complier by:

```
module load intel/18.3
```

One can compile the main function of UP-FHDI by:

```
mpiicc –o main_MPI main_MPI.cpp
```

As a result, an exact executable file (main_MPI) will be generated. Finally, users can submit the created job script to launch the MPI application by:

```
sbatch run.sbatch
```

UP-FHDI generates six output files. Imputation cells, unique observed patterns, and unique missing patterns are included in *datz_binary.bin*, *uox_binary.bin*, and *mox_binary.bin*, respectively. *fmat_FHDI_binary.bin* and *final_daty_binary.bin* contain imputation results and a single complete data, respectively. We will not exhibit the contents of these binary files since they are not human-readable. The TXT file (*summary.data.txt*) contains mean estimates and variance estimates below:

```
Mean estimates
    0.92344         1.95462         1.86083         -0.0904484        ⋯
Linearized variance results
  0.00102272      0.00106363      0.0019027       0.00198615        ⋯
```

5.3 Ultra Data Application on Stampede2

This input data has 0.1 million instances, 10000 variables, and a 30% missing rate. The application adopts MPI IO to import data and UP-FHDI for data curing such that it requires four input files and a job script (*run.sbatch*). The first input file (*input.txt*) contains basic configurations below:

```
INPUT INFORMATION
i_option_read_data                    1
i_option_ultra                        1
i_option_perform                      1
nrow                                  100000
ncol                                  10000
i_option_imputation                   2
i_option_variance                     1
i_option_merge                        0
i_donor                               5
i_user_defined_datz                   0
i_option_collapsing                   4
```

```
i_option_SIS_type                    3
top_correlation                      1000
i_option_cellmake                    2
i_option_var_type                    2
memory                               8
END INPUT INFORMATION


category
        3             3             3             3             ⋯


NonCollapsible_categorical
        0             0             0             0             ⋯


weight
1
1
1
⋮
END DATA INPUT
```

This input data is classified as a ultra data since the number of instances (*nrow* $= 100000$) and the number of variables (*ncol* $= 10000$) are both very large. We adopt UP-FHDI (*i_option_ultra* $= 1$) because it is capable of processing ultra datasets efficiently. This example enables MPI IO (*i_option_read_data* $= 1$) to import basic configurations from a TXT file (*input.txt*), raw data **y** from two binary files (*daty_column_binary.bin* and *daty_row_binary.bin*), and response indicator matrix **r** from a binary file (*datr_column_binary.bin*), respectively. This option is often used for UP-FHDI with a large data volume. We perform all procedures of UP-FHDI (*i_option_perform* $= 1$) including fractional hot-deck imputation (*i_option_imputation* $= 2$) and linearized variance estimation (*i_option_variance* $= 1$ & *i_option_var_type* $= 2$). UP-FHDI only supports the KNN method (*i_option_cellmake* $= 2$) to find deficient donors. SIS is adopted to reduce high dimensions to four selected variables (*i_option_collapsing* $= 4$) based on a global ranking of simple correlations (*i_option_SIS_type* $= 3$). The available memory per MPI task (*memory*) $= 192$ GB$/ 24 = 8$ GB considering 192 GB memory of a SKX node on Stampede2. *NonCollapsible_categorical* are set to be zeros because all variables are continuous. We leave the rest of the parameters as default values. We will not exhibit the other three binary input files (i.e., *daty_column_binary.bin*, *daty_row_binary.bin*, and *datr_column_binary.bin*) since they are not human-readable. The job script file contains information on computing resources below:

```
#!/bin/bash

#SBATCH –J UP-FHDI        # Job name

#SBATCH –o UP-FHDI.o%j # Name of the output file
```

```
#SBATCH –e UP-FHDI.e%j  # Name of the error file
#SBATCH –p skx-normal   # Queue (partition) name
#SBATCH –N 2            # Total number of nodes
#SBATCH –n 48           # Total number of MPI tasks
#SBATCH –t 15:00:00     # Maximum run time (hh:mm:ss)
module load ooops       #Load optimal overloaded IO protection system
set_io_param_batch $SLURM_JOBID 0 high # Set up configurations of OOOPS
ibrun ./main_MPI        # Launch MPI code
```

Overall, this example requires two SKX nodes with forty-eight MPI tasks and a fifteen-hour maximum running time. Notably, a single user's intensive and simultaneous IO running on a few nodes can even overload the metadata server and degrade global file system performance. To resolve this issue, we adopt the optimal overload IO protection system (OOOPS) to automatically detect and throttle the IO workload. Users can adjust configurations of OOOPS by issuing `set_io_param_batch`. The variable "`$SLURM_JOBID`" represents the numeric job ID and `0` represents the Stampede2 SCRATCH file system. The SCRATCH file system manages IO activities of MPI applications. Throttling IO has four levels, from the least to the most: low, medium, high. Users only need to adjust levels of throttling IO regarding their needs. The status "`high`" is usually enough to resolve most of the issues.

Users must ensure four input files, the job script, and UP-FHDI codes in the same directory of HPC facilities. Users can download all required files in (Yang, Kim, et al., 2021). This input data is named Synthetic Data 3, belonging to the ultra category. Notably, we adopt a different naming convention for input files of Synthetic Data 3. Users must modify input file names to match the exact names above. Besides, we highly recommend using Globus to transfer big input files between the local computer and HPC facilities.

Following procedures in Section 4.2, users can compile the main function of UP-FHDI by:

```
mpicxx –o main_MPI main_MPI.cpp
```

As a result, an exact executable file (`main_MPI`) will be generated. Finally, users can submit the created job script to launch the MPI application by:

```
sbatch run.sbatch
```

UP-FHDI generates six output files. Imputation cells, unique observed patterns, and unique missing patterns are included in *datz_binary.bin*, *uox_binary.bin*, and *mox_binary.bin*, respectively. *fmat_FHDI_binary.bin* and *final_daty_binary.bin* contain imputation results and a single complete data, respectively. We will not exhibit the contents of these binary files since they are not human-readable. The TXT file (*summary.data.txt*) contains mean estimates and variance estimates below:

```
 Mean estimates
     0.993402            3.10881            4.10978            1.0672            ···
 Linearized variance results
   9.96415e-06        2.68932e-05        3.61617e-05        3.27215e-05          ···
```

REFERENCE

Cho, I. H. (2021). Data-driven, computational science and engineering platforms repository. Retrieved from https://sites.google.com/site/ichoddcse2017/home/type-of-trainings/ultra-data-oriented-parallel-fhdi-up-fhdi

Iowa State University. (2017). Slurm job script generator for Condo. Retrieved from https://www.hpc.iastate.edu/guides/condo-2017/slurm-job-script-generator-for-condo

Iowa State University. (2021). Condo2017: Iowa state university high-performance computing cluster system. Retrieved from https://www.hpc.iastate.edu/guides/condo-2017

Sarkar, T. (2021a). Manual for accessing Condo 2017. Retrieved from https://iastate.app.box.com/folder/153144496277

Sarkar, T. (2021b). Manual for accessing Stampede2. Retrieved from https://iastate.app.box.com/folder/153144496277

Texas Advanced Computing Center. (2021). Stampede2 user guide. Retrieved from https://portal.tacc.utexas.edu/user-guides/stampede2

Yang, Y., Kim, J., & Cho, I. H. (2020). Parallel fractional hot deck imputation and variance estimation for big incomplete data curing. *IEEE Transactions on Knowledge and Data Engineering*. https://doi.org/10.1109/tkde.2020.3029146

Yang, Y., Kim, J. K., & Cho, I. H. (2021). Incomplete big datasets for ultra data-oriented parallel fractional hot-deck imputation. Retrieved from https://ieee-dataport.org/open-access/incomplete-big-datasets-ultra-data-oriented-parallel-fractional-hot-deck-imputation

Yang, Y., Kwon, Y., Kim, J. K., & Cho, I. H. (2021). Ultra data-oriented parallel fractional hot-deck imputation with efficient linearized variance estimation. *IEEE Transactions on Knowledge and Data Engineering*.