HDF5 (Hierarchical Data Format v5)
PPP 07

Gabriela Nečasová

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
inecasova@fit.vut.cz

Úvod

- **Úkoly:** pracujeme s funkcemi **H5***
  - Sériové I/O a zápis skalární hodnoty do datasetuVytvoření souboru následované individuálními zápisy
  - Zápis matice do datasetu pomocí kolektivního I/O
  - Čtení několika matic z různých datasetů a skupin

- Kde počítat?
  - Barbora/Karolina/Merlin – možno vypracovat celé cvičení
  - Poznámka: potřebujeme 16 procesů, na Merlinovi je potřeba --oversubscribe

- Cvičení jsou dobrovolná, ovšem mohou výrazně usnadnit řešení projektu ☺
- **Nápověda**:
  - Přednáška PPP 7
  - HDF5 dokumentace: https://support.hdfgroup.org/documentation/
    - **HDF5 File**
    - **HDF5 Groups**
    - **HDF5 Datasets**
  - Open MPI dokumentace: https://www-lb.open-mpi.org/doc/v4.1/
    - Seznam všech MPI rutin (MPI_Init,...)

Úvod

- Připojení na server?
  - `ssh [merlin|barbora|karolina]`
- Ověření, že je k dospozici MPI:
  - `mpicc --help`

- Pokud pracujete na Barboře/Karolině:
  - Karolina
    - `$ salloc -A DD-24-108 -p qcpu_exp -N 1 --ntasks-per-node 128 -t 01:00:00 --x11`
  - Barbora
    - `$ salloc -A DD-24-108 -p qcpu_exp -N 1 --ntasks-per-node 36 -t 01:00:00 --x11`
  - `ml OpenMPI`

- Zobrazení info o MPI instalaci:
  - `ompi_info`

Překlad a spuštění

- 1. možnost: cmake:
  - `$ cmake -Bbuild -S.`
  - `$ cmake --build build --config Release`
- 2. možnost: mpicc:
  - `mpic++ hdf5.cpp -o one`
- Aplikace má 1 parametr – číslo úlohy (zde 1):
  - `mpirun -np 8 ./hdf5 1`

- **Samopopisná hierarchická struktura**
- **Analogie: MATLAB *.mat soubory**

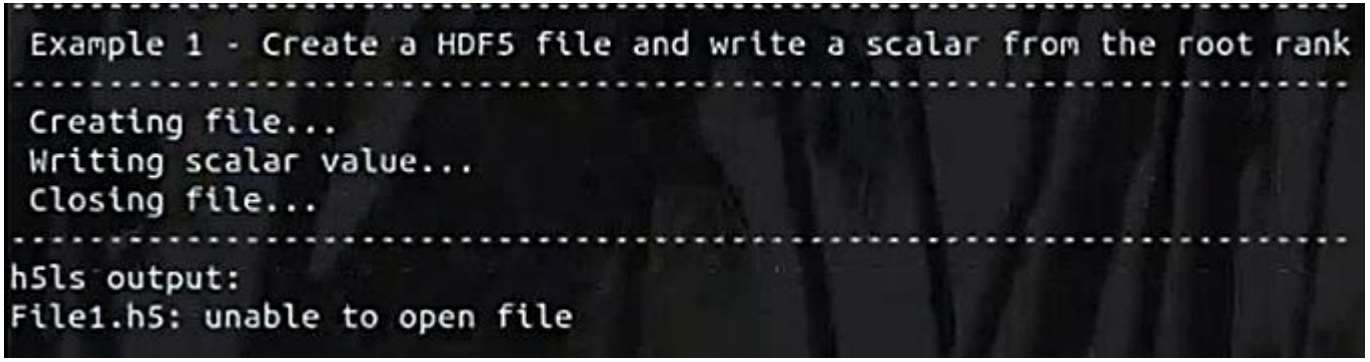
• H5Fcreate (H5Fopen)	create (open) File
○ H5Screate_simple/H5Screate	create dataSpace
– H5Dcreate (H5Dopen)	create (open) Dataset
• H5Dread, H5Dwrite	access Dataset
– H5Dclose	close Dataset
○ H5Sclose	close dataSpace
• H5Fclose	close File

• DataSpaces:	H5Sselect_hyperslab (Partial I/O), H5Sselect_elements (Partial I/O)
• DataTypes:	H5Tcreate, H5Tcommit, H5Tclose, H5Tequal, H5Tget_native_type
• Groups:	H5Gcreate, H5Gopen, H5Gclose
• Attributes:	H5Acreate, H5Aopen_name, H5Aclose, H5Aread, H5Awrite
• Property lists:	H5Pcreate, H5Pclose, H5Pset_chunk, H5Pset_deflate

Základní funkce

- /// Execute a given command in the shell and return the output.
- **std::string exec**(const std::string_view cmd);
  - → Pro zjišťování informací o HDF souborech

- **Pozor**: V HDF je nutné správně uzavírat soubory, datasety, skupiny… protože pokud dojde k přerušení výpočtu, tak se může stát, tak dojde k porušení souboru a ten je pak nečitelný.



ÚKOL 1: SÉRIOVÉ I/O A ZÁPIS SKALÁRNÍ HODNOTY DO DATASETU

1: Sériové I/O a zápis skalární hodnoty do datasetu

Cílem tohoto příkladu je vyzkoušet si tvorbu a uzavření souboru (`H5Fcreate`, `H5Fclose`) a zápis z jednoho ranku do vybraného datasetu. Se souborem bude pracovat pouze root rank! Zadání se nachází pod sekcí `case 1:` ve funkci `main`.

1. Nejprve si deklarujte objekt HDF5 souboru (`hid_t`).
2. Vytvořte soubor se jménem uloženým v konstantě `fileName`. Použijte takový flag, který zajistí přepis souboru, pokud již existuje.
3. Vytvořte `filespace` a `memspace` popisující tvar skalární proměnné (velikost [1]).
4. Zapište hodnotu `value` do datasetu.
5. Uzavřete dataset a soubor.
6. Přeložte soubor.
7. Spusťte výslednou binárku a prohlédněte si obsah souboru:

```
$ mpiexec -np 8 ./hdf5 1
```

8. Prostudujte obsah souboru pomocí cmd rutin `h5ls` a `h5dump`. Vyzkoušejte si různé parametry.

Pouze root: v souboru vytvořit data set o velikosti 1 integeru a do něj zapsat 1 číslo

1: Sériové I/O a zápis skalární hodnoty do datasetu

- // 1. Declare an HDF5 file: **hid_t file;**
- // 2. Create a file with write permission. Use such a flag that overrides existing file.
- // The list of flags is in the header file called H5Fpublic.h
- **H5Fcreate() // flags = H5F_ACC_TRUNC, fcpl (creation property) = fapl (access property) = H5P_DEFAULT**
- // 3. Create file and memory spaces. We will only write a single value.
- **HDScreate_simple() // dims = dimenze datasetu, maxdims = max velikost dimenzí, zde nullptr**
- // 4. Create a dataset of a size [1] and int datatype.
- // The list of predefined datatypes can be found in H5Tpublic.h
- **H5Dcreate() // type_id = H5T_NATIVE_INT, space_id=filespace, ostatní H5T_DEFAULT**
- //5. Write value into the dataset.
- **H5Dwrite() // id datasetu, dat. typ, tvar dat, plist = H5P_DEFAULT.**
  // Zapisuji jednu hodnotu, takže namespace = filespace
- // 6. Close dataset. **H5Dclose()**
- // 7. Close memspace. **H5Sclose()**
- // 8. Close filespace. **H5Sclose ()**
- // 9. Close file **H5Fclose()**

1. Vytvoření souboru
2. Vytvoření datasetu a zavření datasetu
3. Zápis do datasetu

```
hid_t H5Fcreate (const char * filename, unsigned flags, hid_t fcpl_id, hid_tfapl_id)

herr_t H5Dwrite (hid_t dset_id, hid_t mem_type_id, hid_t mem_space_id, hid_t file_space_id,
hid_t dxpl_id, const void *   buf)

hid_t H5Screate_simple(int rank, const hsize_t dims[],  hsize_t maxdims[])
hid_t H5Dcreate(hid_t loc_id, const char * name, hid_t type_id, hid_t space_id, hid_t dcpl_id)
herr_t H5Dwrite(hid_t dset_id, hid_t mem_type_id ,hid_t mem_space_id, hid_t file_space_id, hid_t
dxpl_id, const void * buf)
herr_t H5Sclose(hid_t space_id)
herr_t H5Dclose(hid_t dset_id)
herr_t H5Fclose (hid_t file_id)
```

1: Sériové I/O a zápis skalární hodnoty do datasetu

```
---------------------------------------------------------------------
h5ls output:
---------------------------------------------------------------------
/                        Group
/Dataset-1               Dataset {1}

---------------------------------------------------------------------
h5dump output:
HDF5 "File1.h5" {
DATASET "Dataset-1" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   STORAGE_LAYOUT {
      CONTIGUOUS
      SIZE 4
      OFFSET 2048
   }
   FILTERS {
      NONE
   }
   FILLVALUE {
      FILL_TIME H5D_FILL_TIME_IFSET
      VALUE  H5D_FILL_VALUE_DEFAULT
   }
   ALLOCATION_TIME {
      H5D_ALLOC_TIME_LATE
   }
   DATA {
   (0): 128
   }
}
}
```

Hlavička 2KB

V datasetu je jedno číslo: 128

- **h5ls** – zjistím co je v souboru – 2 dataset o velikosti 1
- **h5dupm** – zobrazí objekty v HDF5 souboru

- **Co když chci zapsat 10 hodnot?**
- constexpr int value[10] = {1,2,3,4,5,6,7,8,9,10};
- constexpr hsize_t rank = 1;
- constexpr hsize_t size = 10;

1: Sériové I/O a zápis skalární hodnoty do datasetu

- **Co když chci data zapsat jako matici 2x5?**
- Co když chci zapsat 10 hodnot?
- constexpr int value[10] = {1,2,3,4,5,6,7,8,9,10};
- constexpr hsize_t rank = 2;
- constexpr hsize_t size[2] = {2, 5};
- constexpr hsize_t memDims = 10;

- const hid_t filespace = H5Screate_simple(rank, size, nullptr);
- const hid_t memspace = H5Screate_simple(1, &memDims, nullptr);

```
h5ls output:
/                        Group
/Dataset-1               Dataset {2, 5}

-----------------------------------------
h5dump output:
HDF5 "File1.h5" {
DATASET "Dataset-1" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SIMPLE { ( 2, 5 ) / ( 2, 5 ) }
   STORAGE_LAYOUT {
      CONTIGUOUS
      SIZE 40
      OFFSET 2048
   }
   FILTERS {
      NONE
   }
   FILLVALUE {
      FILL_TIME H5D_FILL_TIME_IFSET
      VALUE  H5D_FILL_VALUE_DEFAULT
   }
   ALLOCATION_TIME {
      H5D_ALLOC_TIME_LATE
   }
   DATA {
   (0,0): 1, 2, 3, 4, 5,
   (1,0): 6, 7, 8, 9, 10
   }
}
}
```

ÚKOL 2: ZÁPIS MATICE DO DATASETU POMOCÍ KOLEKTIVNÍHO I/O

2: Zápis matice do datasetu pomocí kolektivního I/O

- Máme matici 16x4 distribuovaná po řádcích
- Globální matice gMatrix
- Lokální matice lMatrix
- **Každý rank** má lRows a nCols a chceme to **zapsat do datasetu**
- Začátek je stejný jako v minulém příkladu

2: Zápis matice do datasetu pomocí kolektivního I/O

Mějme matici o velikosti 16 × 4 prvky typu `int`. Tato matice je distribuovaná mezi ranky po blocích řádků. Vaším cílem je tuto matici zapsat do jednoho datasetu uvnitř HDF5 souboru. Zadání se nachází pod sekcí `case 2:` ve funkci `main`.

1. Nejprve si deklarujte objekt HDF5 souboru (`hid_t`).
2. Vytvořte file access property list a zapněte MPI-IO (`H5Pcreate, H5Pset_fapl_mpio`).
3. Vytvořte soubor se jménem uloženým v konstatně `fileName`. Použijte takový flag, který zajistí přepis souboru, pokud již existuje.
4. Vytvořte `filespace` a `memspace` popisující celkovou velikost datasetu a jeho část v paměti daného ranku.
5. Pomocí hyperslabu vyberte část datasetu, do které budete zapisovat (`H5Sselect_hyperslab`).
6. Vytvořte XFER property list a zapněte kolektivní IO (`H5Pset_dxpl_mpio`).
7. Zapište hodnotu části matice do datasetu.
8. Uzavřete property listy, dataset a soubor.
9. Přeložte soubor.
10. Spusťte výslednou binárku a prohlédněte si obsah souboru:

    ```
    $ mpiexec -np 8 ./hdf5 2
    ```

11. Prostudujte obsah souboru pomocí cmd rutin `h5ls` a `h5dump`. Vyzkoušejte si různé parametry.

2: Zápis matice do datasetu pomocí kolektivního I/O

- // 1. Declare an HDF5 file. **hid_t file;**
- **// 2. Create a property list to open the file using MPI-IO in the MPI_COMM_WORLD communicator.**
- **H5_DLL herr_t H5Pset_fapl_mpio (hid_t fapl_id, MPI_Comm comm, MPI_Info info)**
- // 3. Create a file called (filename) with write permission. Use such a flag that overrides existing file.
- // The list of flags is in the header file called H5Fpublic.h
- **H5Fcreate()**
- // 4. Close file access list.
- **H5Pclose()**
- // 5. Create file space - a 2D matrix [nRows][nCols] - globální matice
- // Create mem space - a 2D matrix [lRows][nCols] mapped on 1D array lMatrix.
- **const hid_t filespace = H5Screate_simple()**
- **const hid_t memspace = H5Screate_simple()**
- // 6. Create a dataset. The name is store in datasetname, datatype is int. All other parameters are default.
- **H5Dcreate()**
- **// 7. Select a hyperslab to write a local submatrix into the dataset.**
- **herr_t H5Sselect_hyperslab() // z dataspacu vyberu jen část**
  - **hid_t space_id, // ze kterého datového prostoru data beru**
  - **H5S_seloper_t op, // H5S_SELECT_SET**
  - **const hsize_t start[], // Kde začínáme? Podle toho který jsme rank.**
  - **const hsize_t stride[], // nullptr**
  - **const hsize_t count[], // velikost dané části: lRows, nCols**
  - **const hsize_t block[] // máme jen jeden blok – jeden balík řádků, takže: nullptr**
- **// 8. Create XFER property list and set Collective IO.**
- **hid_t H5Pcreate (hid_t cls_id)**
- **H5_DLL herr_t H5Pset_dxpl_mpio(hid_t dxpl_id, H5FD_mpio_xfer_t xfer_mode) // mode = H5DF_MPIO_COLLECTIVE**
- // 9. Write data into the dataset.
  herr_t H5Dwrite(hid_t dset_id, hid_t mem_type_id ,hid_t mem_space_id, hid_t file_space_id, hid_t dxpl_id, const void * buf)
  - **dxpl_id = xferPList**

- // 10. Close XREF property list. **H5Pclose()**
- // 11. Close dataset. **H5Dclose()**
- // 12. Close memspace and filespace. **2x H5Sclose()**

- Vytvořím soubor
- Potom v něm vyrobím dataset
- Pak přidám další věci

ÚKOL 3: ČTENÍ NĚKOLIKA MATIC Z RŮZNÝCH DATASETŮ A SKUPIN

3: Čtení několika matic z různých datasetů a skupin

- Představme si unit test – potřebujeme nějaká data, na kterých vyzkoušíme náš kód
- Máme **Matrix-File.h5**
  - Matice A, B
  - Matice C – referenční výsledek
- Musíme načíst data
- Spočítat matematickou operaci
- Porovnat výsledky

- struct **Dim2D**: nRows, nCols, konstruktory
  - nElements(), toString(), toArray()
- define DEBUG – zapnutí debug režimu

```
gnecasov@login2:~/PPP_bk/lab7$ h5ls -r Matrix-File.h5
/                        Group
/Inputs                  Group
/Inputs/Matrix-A         Dataset {16, 32}
/Inputs/Matrix-B         Dataset {32, 16}
/Outputs                 Group
/Outputs/Matrix-C-ref    Dataset {16, 32}
```

3: Čtení několika matic z různých datasetů a skupin

```
gnecasov@login2:~/PPP_bk/lab7$ h5dump -r Matrix-File.h5
HDF5 "Matrix-File.h5" {
GROUP "/" {
   GROUP "Inputs" {
      DATASET "Matrix-A" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 16, 32 ) / ( 16, 32 ) }
         DATA {
         (0,0): 0.287648, 0.918922, 0.209915, 0.207157, 0.376689, 0.512666,
         (0,6): 0.543695, 0.492667, 0.570892, 0.176661, 0.640559, 0.425483,
         (0,12): 0.528611, 0.0849048, 0.439923, 0.505086, 0.0219298,
         (0,17): 0.126099, 0.626955, 0.229306, 0.941231, 0.746182, 0.174118,
         (0,23): 0.492313, 0.772656, 0.0127556, 0.427062, 0.10532, 0.566871,
```

```
DATASET "Matrix-B" {
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( 32, 16 ) / ( 32, 16 ) }
   DATA {
   (0,0): 6.89435, 3.41677, 8.97771, 4.93299, 7.19264, 1.24144,
   (0,6): 5.58298, 2.39176, 5.45222, 9.50395, 6.78492, 6.92659,
   (0,12): 2.62089, 3.78792, 4.50614, 8.4327,
```

```
GROUP "Outputs" {
   DATASET "Matrix-C-ref" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 16, 32 ) / ( 16, 32 ) }
      DATA {
      (0,0): 1.98314, 0.463642, 0.387155, 0.0945843, 3.33386, 4.30534,
      (0,6): 0.642404, 2.02198, 0.686376, 1.01067, 6.0814, 1.09088,
      (0,12): 5.23254, 0.297004, 0.917338, 3.363, 0.213453, 0.785217,
```

3: Čtení několika matic z různých datasetů a skupin

Tento příklad slouží jako ukázka typického unit testu výpočetního kernelu. Naším cílem je otestovat správnost výpočtu Hadamardova násobení

$$C = A \cdot B'; \quad \forall i, j : c[i][j] = a[i][j] * b[j][i] \tag{5.1}$$

Ve vstupním souboru naleznete 2 skupiny "Inputs"a "Outpus". První skupina obsahuje matici A a B zatímco druhá skupina referenční hodnotu C. Všechny matice jsou obdélníkové s hranou o velikosti mocniny 2. Matici A a C je třeba distribuovat po blocích řádků, matici B po blocích sloupců.
Zadání se nachází pod sekcí `case 3:` ve funkci `main`.

1. Nejprve otevřete soubor v režimu pouze pro čtení. Inspirujte se předchozím příkladem.
2. Nyní otevřete vstupní a výstupní HDF5 skupinu.
3. Následně otevřete vstupní datasety s maticemi `A`, `B` a `Cref`.
4. Nyní musíme zjistit velikost matic a spočítat jejich distribuci.
   a) Nejprve si napíšeme pomocnou lambda funkci, která nám přečte počet dimenzí daného datasetu a uloží je do objektu typu `Dim2D`. Zde je třeba nejprve získat dataspace `H5Dget_space` a následně vhodnou funkcí přečíst velikosti dimenzí `H5Sget_simple_extent_dims`. Přepis výsledku do objektu `Dim2D` zajistíte konverzí struktury na pole `Dim2D::toArray()`.
   b) Následně použijte tuto lambda funkci na zjištění celkových velikostí daných matic.
   c) Dle počtu ranků v komunikátoru spočtěte lokální velikosti matic A, B a C.
5. Následně přečteme požadované části datasetů do připravených polí jednotlivých ranků. Pro tento účel si napíšeme novou lambda funkci, která má jako parametry: id datasetu, pozici slabu, velikost slabu a velikost celého datasetu.
6. Následně načteme dané datasety.
7. Nyní necháme proběhnout výpočet. Pokud byla data distribuována správně, měla by bát maximální absolutní chyba rovna 0.
8. Na závěr uzavřete property listy, dataset a soubor.
9. Přeložte soubor v normálním nebo debug módu (vypíše obsah matic).
10. Spusťte výslednou binárku a prohlédněte si obsah souboru:

```
$ mpiexec -np 8 ./hdf5 3
```

3: Čtení několika matic z různých datasetů a skupin

- **// 1. Create a property list to open the HDF5 file using MPI-IO in the MPI_COMM_WORLD communicator.**
-
- **// 2. Open a file called (filename) with read-only permission.**
- // The list of flags is in the header file called H5Fpublic.h
-
- **// 3. Close file access list.**
-
- **// 4. Open HDF5 groups with input and output matrices.**
- H5Gopen(hid_t loc_id, const char* name, hid_t gapl_id)
- H5Gopen(hid_t loc_id, const char* name, hid_t gapl_id)
- **// 5. Open HDF5 datasets with input and output matrices.**
- HDopen()
- HDopen()
- **// 6. Write a lambda function to read the dataset size. The routine takes dataset ID and returns Dims2D.**
- // i. Get the dataspace from the dataset using H5Dget_space.
- // ii. Read dataset sizes using H5Sget_simple_extent_dims.
- // The rank of the dataspace is 2 (2D matrices) and the dimensions are stored in Row, Col manner (row major)
- // You can use a conversion dims.toArray() to pass the structure as an array.
- // iii. Close the dataspace.
- auto getDims = [](hid_t dataset) -> Dim2
- {
- Dim2 dims{}
-
- return dims;
- };// end of getDims

3: Čtení několika matic z různých datasetů a skupin

```
// 7. Get global matrix dimension sizes.
    const Dim2 gDimsA /* =  ? */;
    const Dim2 gDimsB /* =  ? */;
    const Dim2 gDimsC /* =  ? */;

    // 8. Calculate local matrix dimension sizes. A, C and Cref are distributed by rows, B by columns.
    const Dim2 lDimsA /* =  ? */;
    const Dim2 lDimsB /* =  ? */;
    const Dim2 lDimsC /* =  ? */;

    // Print out dimension sizes
    mpiPrintf(MPI_ROOT_RANK, "  - Number of ranks: %d\n", mpiGetCommSize(MPI_COMM_WORLD));
    mpiPrintf(MPI_ROOT_RANK, "  - Matrix A global and local size [%llu, %llu] / [%llu, %llu]\n",
        gDimsA.nRows(), gDimsA.nCols(), lDimsA.nRows(), lDimsA.nCols());
    mpiPrintf(MPI_ROOT_RANK, "  - Matrix B global and local size [%llu, %llu] / [%llu, %llu]\n",
        gDimsB.nRows(), gDimsB.nCols(), lDimsB.nRows(), lDimsB.nCols());
    mpiPrintf(MPI_ROOT_RANK, "  - Matrix C global and local size [%llu, %llu] / [%llu, %llu]\n",
        gDimsC.nRows(), gDimsC.nCols(), lDimsC.nRows(), lDimsC.nCols());

    // Allocate memory for local arrays
    matrixA.resize(lDimsA.nElements());
    matrixB.resize(lDimsB.nElements());
    matrixC.resize(lDimsC.nElements());
    matrixCref.resize(lDimsC.nElements());
```

3: Čtení několika matic z různých datasetů a skupin

```
// 9. Write a lambda function to read a particular slab at particular ranks.
    //  i.  Create a 2D filespace, then select the part of the dataset you want to read.
    //  ii. Create a memspace where to read the data.
    //  iii. Enable collective MPI-IO.
    //  iv.  Read the slab.
    //  v.  Close property list, memspace and filespace.
    auto readSlab = [](hid_t dataset,
                const Dim2& slabStart, const Dim2& slabSize, const Dim2& datasetSize,
                double* data) -> void
    {
    };// end of readSlab

    // 10. Read parts of the matrices A, B and Cref.
    mpiPrintf(MPI_ROOT_RANK, "  Reading matrix A...\n");

    mpiPrintf(MPI_ROOT_RANK, "  Reading matrix B...\n");

    mpiPrintf(MPI_ROOT_RANK, "  Reading matrix Cref...\n");

    mpiPrintf(MPI_ROOT_RANK, "  Calculating C = A ° B' ...\n");

    // Calculate the Hadamard product C = A ° B'
    std::fill(matrixC.begin(), matrixC.end(), 0.0);

    for (hsize_t row{}; row < lDimsC.nRows(); row++)
    {
      for (hsize_t col{}; col < lDimsC.nCols(); col++)
      {
        matrixC[row * lDimsC.nCols() + col] = matrixA[row * lDimsA.nCols() + col] * matrixB[col * lDimsB.nCols() + row];
      }
    }
```

3: Čtení několika matic z různých datasetů a skupin

```
h5ls output:
/                        Group
/Inputs                  Group
/Inputs/Matrix-A         Dataset {16, 32}
/Inputs/Matrix-B         Dataset {32, 16}
/Outputs                 Group
/Outputs/Matrix-C-ref    Dataset {16, 32}

----------------------------------------------------------------------
h5dump output (dataset data omitted, enable by using -d parameter):
HDF5 "Matrix-File.h5" {
GROUP "/" {
   GROUP "Inputs" {
      DATASET "Matrix-A" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 16, 32 ) / ( 16, 32 ) }
      }
      DATASET "Matrix-B" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 32, 16 ) / ( 32, 16 ) }
      }
   }
   GROUP "Outputs" {
      DATASET "Matrix-C-ref" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 16, 32 ) / ( 16, 32 ) }
      }
   }
}
}
```

```
Opening file...
Reading dimension sizes...
 - Number of ranks: 8
 - Matrix A global and local size [16, 32] / [2, 32]
 - Matrix B global and local size [32, 16] / [32, 2]
 - Matrix C global and local size [16, 32] / [2, 32]
Reading matrix A...
Reading matrix B...
Reading matrix Cref...
Calculating C = A ° B' ...
Verification C == Cref ...
Maximum abs error = 0.000000e+00
```

Děkuji vám za pozornost!