HDF5-UDF

User-Defined Functions in HDF5 using Lua, Python and C/C++

Lucas C. Villa Real
IBM Research
lucasvr@br.ibm.com

Motivation: can we do better than this?

```
1 def main()
2     h5 = h5py.File(”file.h5”, ”r+”)
3     a, b = h5[“A”], h5[“B”]
4     c = a + b
5     h5.create_dataset(“C”, a.shape, a.dtype, data=c)
```

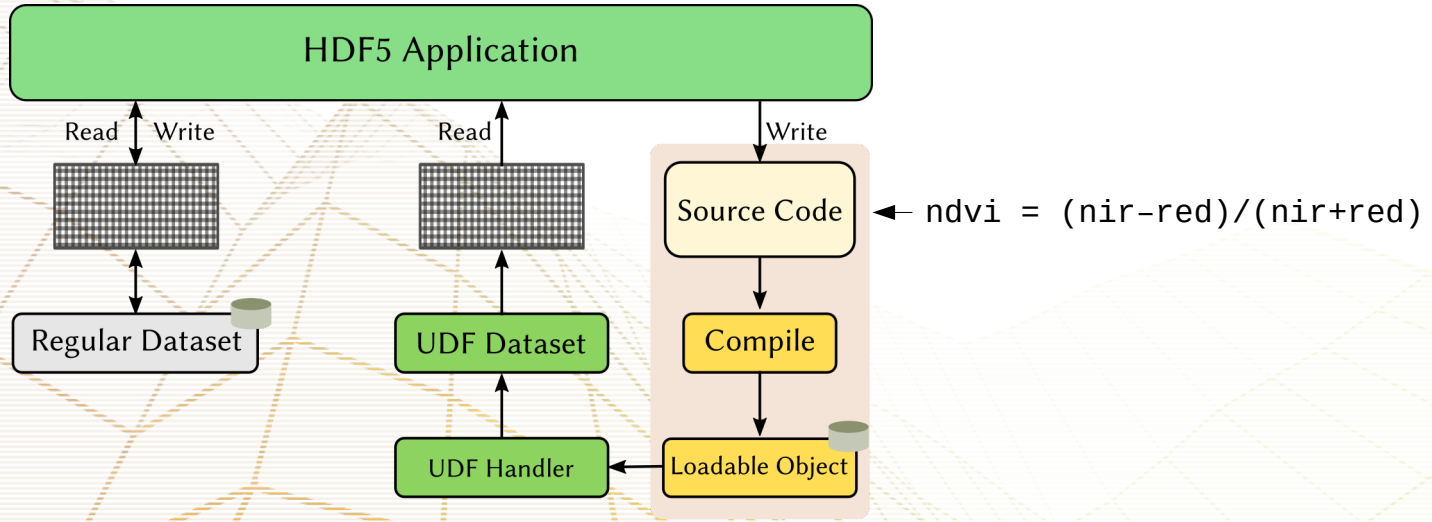
“C” is barely a combination of “A” and “B”

Motivation: can we do better than this?

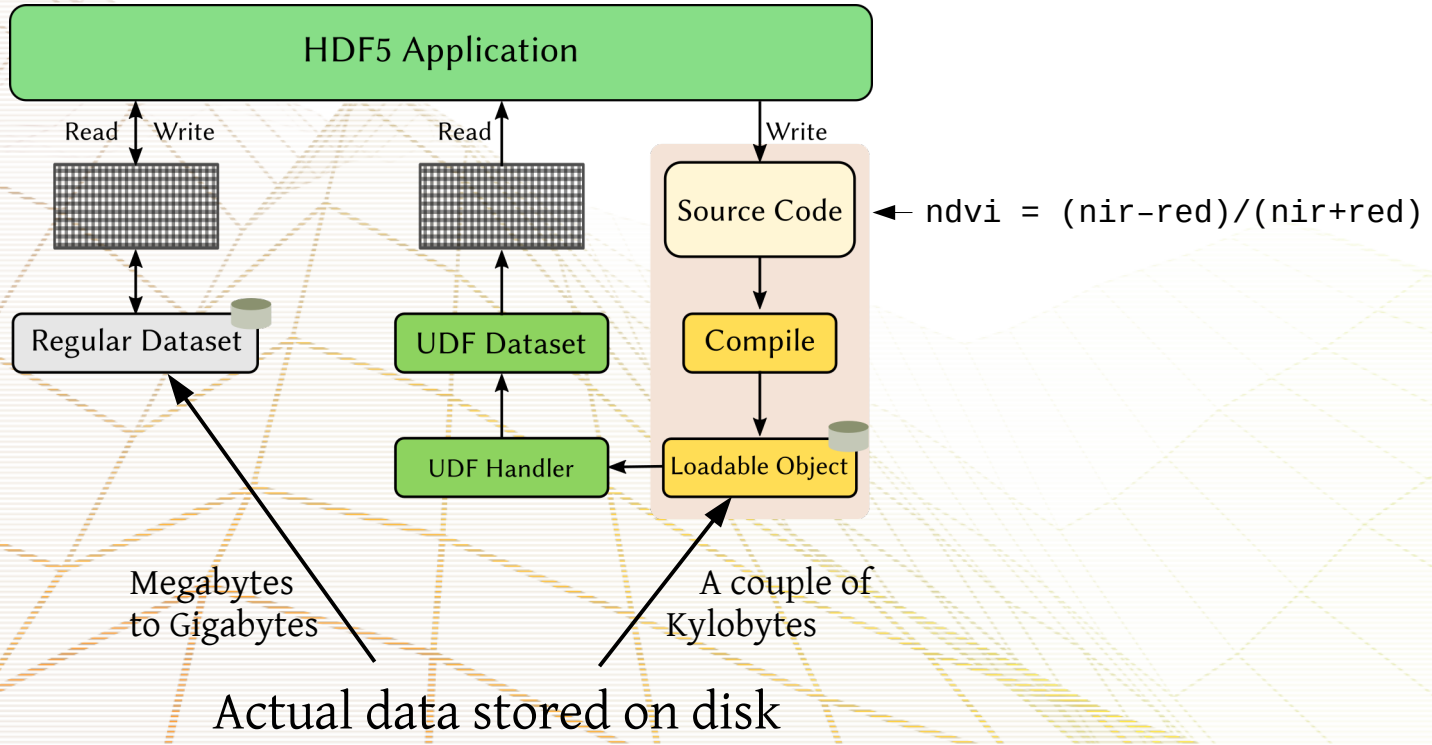
```
1 def main()
2       h5 = h5py.File(”landsat.h5”, ”r+”)
3       nir, red = h5[“nir_band”], h5[“red_band”]
4       ndvi = (nir – red) / (nir + red)
5       h5.create_dataset(“ndvi”, nir.shape, nir.dtype, data=ndvi)
```

“ndvi” is a combination of “red_band” and “nir_band”

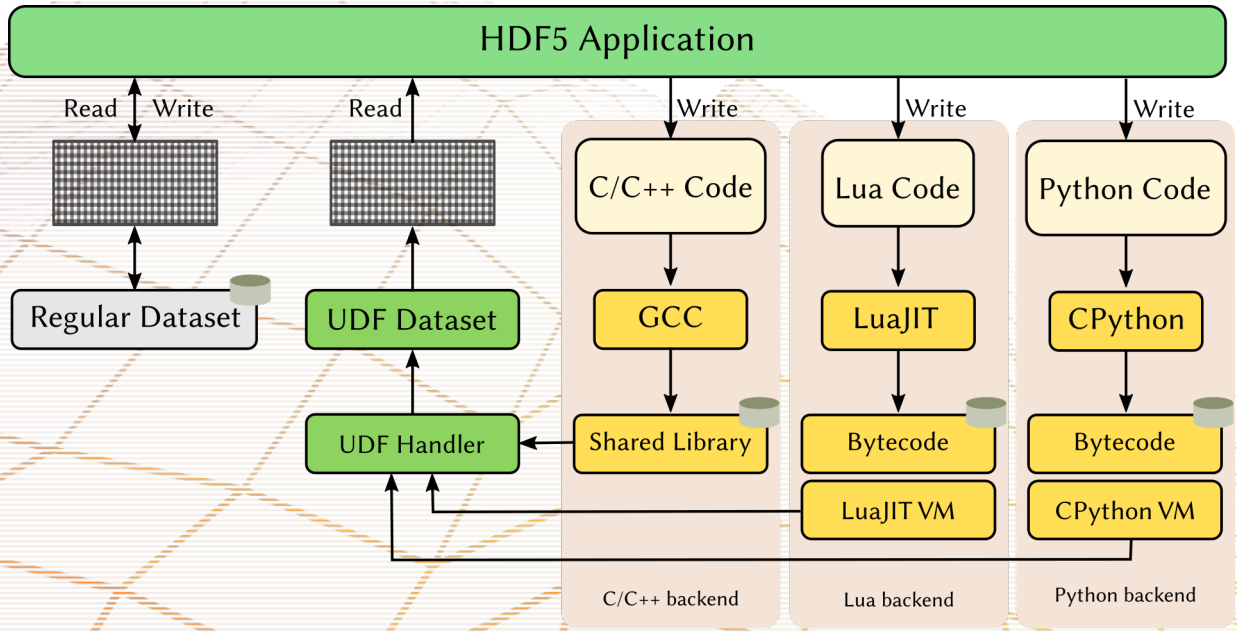
Procedural generation of datasets



Procedural generation of datasets



User-Defined Functions for HDF5



HDF5 Filter API

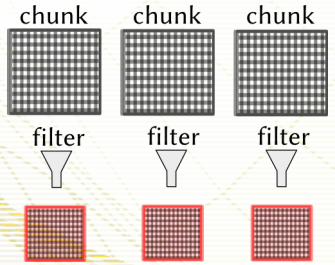
- Compress: “write” path
- Decompress: “read” path

H5Dwrite

- Stores bytecode + metadata

One big chunk

- Chunk size == Dataset size



Three languages, one API

```
Python: def dynamic_dataset()
C/C++:  void dynamic_dataset()
Lua:    function dynamic_dataset()
```

```
lib.getData("DatasetName") → pointer to data (FFI)
```

```
lib.getDims("DatasetName") → array
[1024, 768]
```

```
lib.getType("DatasetName") → string
int16, int32, int64, uint16, uint32, uint64, float, double
```

Python

```
def dynamic_dataset()
    a = lib.getData("A")
    b = lib.getData("B")
    c = lib.getData("VirtualDataset")
    n = lib.getDims("A")[0] * lib.getDims("A")[1]

    for i in range(n):
        c[i] = a[i] + b[i]
```

```
# hdf5-udf file.h5 udf.py VirtualDataset:1024x768:float
```

Python

```python
def dynamic_dataset()
    a = lib.getData("A")
    b = lib.getData("B")
    c = lib.getData("VirtualDataset")
    n = lib.getDims("A")[0] * lib.getDims("A")[1]

    for i in range(n):
        c[i] = a[i] + b[i]
```

```
# hdf5-udf file.h5 udf.py VirtualDataset:1024x768:float
```

```
UDF dataset header = {
 "backend": "Python",
 "bytecode_size": 879,
 "input_datasets": ["A", "B"],
 "output_dataset": "VirtualDataset",
 "output_datatype": "float",
 "output_resolution": [1024, 768]
}
```

Python

```
def dynamic_dataset()
    a = lib.getData("A")
    b = lib.getData("B")
    c = lib.getData("VirtualDataset")
    n = lib.getDims("A")[0] * lib.getDims("A")[1]

    for i in range(n):
        c[i] = a[i] + b[i]
```

```
# hdf5-udf file.h5 udf.py ~~VirtualDataset:1024x768:float~~
```

```
UDF dataset header = {
 "backend": "Python",
 "bytecode_size": 879,
 "input_datasets": ["A", "B"],
 "output_dataset": "VirtualDataset",
 "output_datatype": "float",
 "output_resolution": [1024, 768]
}
```

C/C++

```
extern “C” void dynamic_dataset() {
    auto a = lib.getData<float>(“A”);
    auto b = lib.getData<float>(“B”);
    auto c = lib.getData<float>(“VirtualDataset”);
    auto n = lib.getDims(“A”)[0] * lib.getDims(“A”)[1];

    for (auto i=0, i<n; ++i)
        c[i] = a[i] + b[i];
}
```

```
# hdf5-udf file.h5 udf.cpp
```

Lua

```lua
function dynamic_dataset()
    local a = lib.getData("A")
    local b = lib.getData("B")
    local c = lib.getData("VirtualDataset")
    local n = lib.getDims("A")[1] * lib.getDims("A")[2]

    for i=0, n-1 do
        c[i] = a[i] + b[i]
    end
end
```

```
# hdf5-udf file.h5 udf.lua
```

Interface with sensors & the Web

```
extern “C” void dynamic_dataset() {
    auto host = gethostbyname(“remote.address.com”)
    auto sockfd = socket(…);
    connect(sockfd, …);
    …
    auto dataset = lib.getData(“IoT_Sensor”);
    auto size = lib.getSize(“IoT_Sensor”)[0];
    read(sockfd, dataset, size);
}
```

- Provide real-time readings from weather radars
- Expose data from remote files as HDF5 datasets
- Hide REST-based services behind HDF5

One UDF, many output datasets

Wind chill, Wind speed, Heat index

```
function dynamic_dataset()
    -- Input datasets
    local t2m  = lib.getData("T2")
    local psfc = lib.getData("PSFC")
    local q2m  = lib.getData("Q2")
    local u10m = lib.getData("U10")
    local v10m = lib.getData("V10")
    -- Output datasets
    local wsp        = lib.getData("wsp")
    local wind_chill = lib.getData("wind_chill")
    local heat_index = lib.getData("heat_index")
    -- Auxiliar variables
    local nx, ny, time = 99, 99, 1
    local CELKEL       = 273.15
    local Rv           = 461.5
    local e_const      = 611
    local epsln        = 0.622
    local L            = 2.5e6
    for i = 0, nx*ny*time do
        wsp[i] = math.sqrt(math.pow(u10m[i], 2) + math.pow(v10m[i], 2))
        -- 10-meter wind speed in mph
        local wsp_mph = wsp[i] * 2.23694
        ---- 2-m temperature in F
        local T2F = (t2m[i] - CELKEL) * (9/5) + 32
        -- Wind Chill in F
        if ((T2F <= 50) and (wsp_mph >= 3)) then
            wind_chill[i] = 35.74 + (0.6215 * T2F) - (35.75 * math.pow(wsp_mph, 0.16)) + (0.4275 * T2F * math.pow(wsp_mph, 0.16))
        else
            wind_chill[i] = T2F
        end
        -- Heat Index in F: first compute 2-m vapor pressure, 2-m saturation vapor pressure, and 2-m RH
        local e2m = q2m[i] * psfc[i] / (epsln + q2m[i])
        local es2m = e_const * math.exp((L/Rv) * ((1/CELKEL) - (1/t2m[i])))
        local rh2m = (e2m/es2m) * 100
        if (rh2m > 100) then
            rh2m = 100
        end
        if ((T2F >= 80) and (rh2m >= 40)) then
            heat_index[i] = -42.379 + (2.04901523 * T2F) + (10.14333127 * rh2m)
            - (0.22475541 * T2F * rh2m) - (6.83783 * 1.e-3 * (math.pow(T2F, 2)))
            - (5.481717 * 1.e-2 * (math.pow(rh2m, 2))) + (1.22874 * 1.e-3 * (math.pow(T2F, 2)) * rh2m)
            + (8.5282 * 1.e-4 * T2F * (math.pow(rh2m, 2))) - (1.99 * 1.e-6 * (math.pow(T2F, 2)) * math.pow(rh2m, 2))
        else
            heat_index[i] = T2F
        end
    end
end
```

One UDF, many output datasets

Wind chill, Wind speed, Heat index

```
function dynamic_dataset()
    -- Input datasets
    local t2m  = lib.getData("T2")
    local psfc = lib.getData("PSFC")
    local q2m  = lib.getData("Q2")
    local u10m = lib.getData("U10")
    local v10m = lib.getData("V10")
    -- Output datasets
    local wsp        = lib.getData("wsp")
    local wind_chill = lib.getData("wind_chill")
    local heat_index = lib.getData("heat_index")
    -- Auxiliar variables
    local nx, ny, time = 99, 99, 1
    local CELKEL       = 273.15
    local Rv           = 461.5
    local e_const      = 611
    local epsln        = 0.622
    local L            = 2.5e6
    for i = 0, nx*ny*time do
        wsp[i] = math.sqrt(math.pow(u10m[i], 2) + math.pow(v10m[i], 2))
        -- 10-meter wind speed in mph
        local wsp_mph = wsp[i] * 2.23694
        ---- 2-m temperature in F
        local T2F = (t2m[i] - CELKEL) * (9/5) + 32
        -- Wind Chill in F
        if ((T2F <= 50) and (wsp_mph >= 3)) then
            wind_chill[i] = 35.74 + (0.6215 * T2F) - (35.75 * math.pow(wsp_mph, 0.16)) + (0.4275 * T2F *
        else
            wind_chill[i] = T2F
        end
        -- Heat Index in F: first compute 2-m vapor pressure, 2-m saturation vapor pressure, and 2-m RH
        local e2m = q2m[i] * psfc[i] / (epsln + q2m[i])
        local es2m = e_const * math.exp((L/Rv) * ((1/CELKEL) - (1/t2m[i])))
        local rh2m = (e2m/es2m) * 100
        if (rh2m > 100) then
            rh2m = 100
        end
        if ((T2F >= 80) and (rh2m >= 40)) then
            heat_index[i] = -42.379 + (2.04901523 * T2F) + (10.14333127 * rh2m)
            - (0.22475541 * T2F * rh2m) - (6.83783 * 1.e-3 * (math.pow(T2F, 2)))
            - (5.481717 * 1.e-2 * (math.pow(rh2m, 2))) + (1.22874 * 1.e-3 * (math.pow(T2F, 2)) * rh2m)
            + (8.5282 * 1.e-4 * T2F * (math.pow(rh2m, 2))) - (1.99 * 1.e-6 * (math.pow(T2F, 2)) * math.p
        else
            heat_index[i] = T2F
        end
    end
end
```

```
wsp dataset header = {
 "backend": "LuaJIT",
 "bytecode_size": 1671,
 "input_datasets": ["T2","PSFC","Q2","U10","V10"],
 "scratch_datasets": ["wind_chill","heat_index"],
 "output_dataset": "wsp",
 "output_datatype": "float",
 "output_resolution": [1, 99, 99]
}
```

```
wind_chill dataset header = {
 "backend": "LuaJIT",
 "bytecode_size": 1671,
 "input_datasets": ["T2","PSFC","Q2","U10","V10"],
 "scratch_datasets": ["wsp","heat_index"],
 "output_dataset": "wind_chill",
 "output_datatype": "float",
 "output_resolution": [1, 99, 99]
}
```

```
heat_index dataset header = {
 "backend": "LuaJIT",
 "bytecode_size": 1671,
 "input_datasets": ["T2","PSFC","Q2","U10","V10"],
 "scratch_datasets": ["wsp","wind_chill"],
 "output_dataset": "heat_index",
 "output_datatype": "float",
 "output_resolution": [1, 99, 99]
}
```

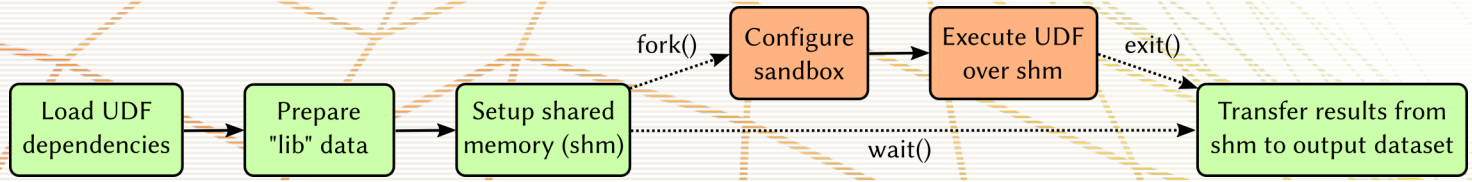
Would you trust third-party UDFs?

Seccomp

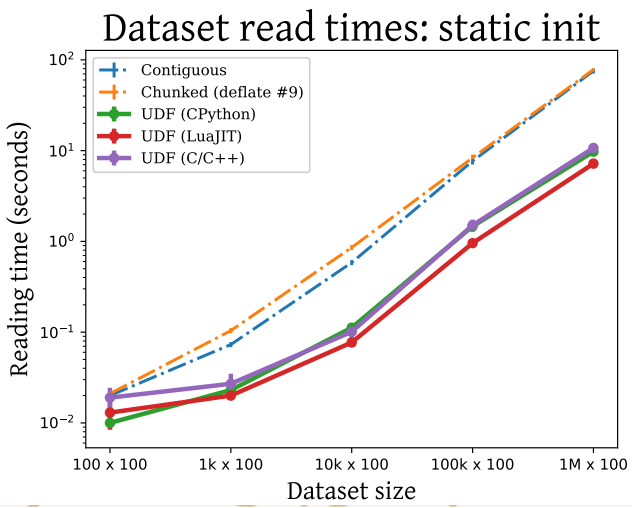
- Defines a list of system calls an UDF can use
- Possible to filter based on arguments, too (caveat: strings are not supported)
- Kills the UDF if it violates the rules

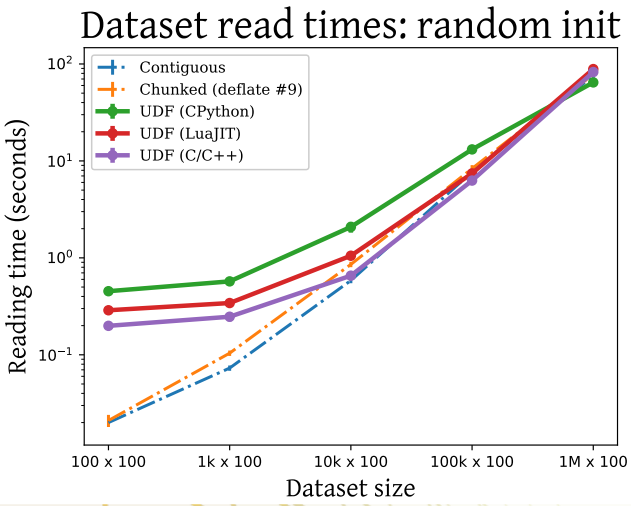
Syscall-Intercept Library

- Intercepts allowed system calls
- Enables filtering based on strings (such as paths)



Performance: we're doing good!





Have fun with HDF5-UDF!

Try it out and check the examples at:
https://github.com/lucasvr/hdf5-udf

HDF5-UDF

User-Defined Functions in HDF5 using Lua, Python and C/C++

Lucas C. Villa Real
IBM Research
lucasvr@br.ibm.com