Parallel Raster-based Geocomputation Operators (PaRGO) Version 2 - User Manual

1. Introduction

Parallel Raster-based Geocomputation Operators (PaRGO) is a C++ parallel programming framework for raster-based geocomputation, featuring support for easy implementation of various geocomputation algorithms in a serial style but with parallel performance on different parallel platforms. In PaRGO Version 2, we have improved its load-balancing performance using the idea of the spatial computational domain. The main features of PaRGO (or PaRGO V2) are:

1. support for one parallel program running on different parallel platforms/models: MPI and MPI+OpenMP for CPU in Beowulf and SMP clusters, and CUDA for GPU.
2. support for implementation of raster-based geocomputation algorithms with different characteristics: local, focal, zonal, and global.
3. enable flexible, practical, and effective load-balancing strategy in multiple modes: the intensity ratio mode, the estimate function mode, and the preliminary experiment mode, for uniform or nonuniform data and computation spatial distribution.

Usage on Windows is documented below.

2. Installation on Windows

2.1 MSVC

PaRGO is a C++ project. Please install `Visual Studio 2010 (VS2010) or later versions`.

To choose a suitable version: any newly released version of `VS` that compatible with GDAL is fine if you only want to run existing algorithms in PaRGO. `Visual Studio 2010` is specifically recommended if you intend to develop new algorithms with PaRGO, since it is the last version that supports the MPI Cluster Debugger.

2.2 CMake

Any newly released version of CMake is fine. You can download the installer from CMake Official Website.

2.3 GDAL

GDAL `1.x`, `2.x`, and `3.x` are all supported.

Take GDAL 2.4.4 as an example:

1. From http://download.gisinternals.com/release.php download the
    - release-1900-x64-gdal-2-4-4-mapserver-7-4-3.zip
    - release-1900-x64-gdal-2-4-4-mapserver-7-4-3-libs.zip

2. Unzip them into **the same folder** (e.g., C:\lib\gdal\2-4-4-vs2015x6)
3. Modify system environmental variables: (replace "C:\lib\gdal\2-4-4-vs2015x64" with your path)
    - `GDAL_ROOT` = `C:\lib\gdal\2-4-4-vs2015x64`
    - `GDAL_DATA` = `C:\lib\gdal\2-4-4-vs2015x64\bin\gdal-data`
    - `GDAL_PATHS` = `C:\lib\gdal\2-4-4-vs2015x64\bin; C:\lib\gdal\2-4-4-vs2015x64\bin\gdal\apps; C:\lib\gdal\2-4-4-vs2015x64\bin\proj\apps; C:\lib\gdal\2-4-4-vs2015x64\bin\curl;`
    - Add `%GDAL_PATHS%` to `PATH`
    - For GDAL 3.x, an additional variable should be added: `PROJ_LIB` = `C:\lib\gdal\3-3-3-vs2019x64\bin\proj\share`

To test if the GDAL is installed successfully, open the command line (CMD) and type:

```
gdalinfo
```

The CMD would print the usages of `gdalinfo` if success.

2.4 MS-MPI

Download `MS-MPI v6 or later versions` from

GitHub (v10 or later) or

Microsoft Archived Websites (v8.1).

To choose a suitable version: note that `VS2010` supports up to `MS-MPI v8.x`. So it is recommended to install `v8.1` if you want to develop new algorithms with PaRGO.

Then:

1. Install the `msmpisdk.msi` and `MSMpiSetup.exe` to a designated path. Take the default path as an example.
2. Add those paths to system environment variables:
    - `MSMPI_BIN` = `C:\Program Files\Microsoft MPI\Bin\`
    - `MSMPI_INC` = `C:\Program Files (x86)\Microsoft SDKs\MPI\Include\`
    - `MSMPI_LIB32` = `C:\Program Files (x86)\Microsoft SDKs\MPI\Lib\x86\`

- `MSMPI_LIB64=C:\Program Files (x86)\Microsoft SDKs\MPI\Lib\x64\`

To test if the MS-MPI is installed successfully, open the command line and type:

```
mpiexec
```

The CMD would print the usages of `mpiexec` if success.

2.5 (Satisfied) OpenMP

OpenMP is included in Visual Studio by default.

2.6 (Optional) CUDA

Depending on specific computing platforms, PaRGO has dependency on CUDA.

3. Build

After installation, you can download the PaRGO project from GitHub using `git clone` or just download the zip file.

Enter the root directory of the PaRGO project, and build it from the command line in the following steps.

1. Create the build directory

   ```
   cd ..
   mkdir build
   cd build
   ```

2. Compile the project

   If you are using VS2010, use

   ```
   cmake .. -G "Visual Studio 10 2010 Win64" ../PaRGO
   ```

   If you are using VS2015, use

```
cmake .. -G "Visual Studio 14 2015 Win64" ../PaRGO
```

It can also be like:

```
cmake .. -G "Visual Studio 11 2012 Win64" ../PaRGO
cmake .. -G "Visual Studio 12 2013 Win64" ../PaRGO
cmake .. -G "Visual Studio 15 2017 Win64" ../PaRGO
cmake .. -G "Visual Studio 16 2019" ../PaRGO
```

Optional CMake argument:

`-DINSTALL_PREFIX=.....`: By default, the install directory is /path/to/source/bin, which can also be specified by adding this argument.

`-DUSE_MPI_DEBUGGER=1`: Please add this argument if you want to use the MPI Cluster Debugger in VS2010, which will be enabled by making each program in separate folders.

The arguments can be specified as follows:

```
cmake .. -G "Visual Studio 10 2010 Win64" ../PaRGO -
DUSE_MPI_DEBUGGER=1 -DINSTALL_PREFIX=D:/compile/bin/pargo
```

3. Build the project
    a. using VS GUI:

        Run `ALL_BUILD` and `INSTALL` in the "Solution explorer -> CMakePredefinedTargets"

    b. using command line:

        Build in the visual studio command line prompt. You can find it in the Windows startup menu, in the Visual Studio folder. Note to choose the x64 version if your Windows is 64-bit.

        For VS2010, the folder is named "Microsoft Visual studio 2010". It also can be found in the path like `"C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Microsoft Visual Studio 2010\Visual Studio Tools\Visual Studio x64 Win64 command prompt (2010).lnk"`

        For VS2015 (and later), the folder is named "Visual Studio 2015". The corresponding path is like `C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Visual Studio 2015\Visual Studio Tools\Developer Command Prompt for VS2015.lnk`

In the command prompt, `cd` to the "build" directory and use

```
msbuild ALL_BUILD.vcxproj /p:Configuration=Release
msbuild INSTALL.vcxproj /p:Configuration=Release
```

Reference: CMake and Visual Studio | Cognitive Waves (wordpress.com)

4. Run

In the PaRGO root directory, run the demos: (`%cd%` will be replaced by the current path when execution)

```
mpiexec -n 4 ..\build\apps\demo\Release\demo1_reclassify.exe -input %cd%\data\dem.tif -output %cd%\out\temp.tif

mpiexec -n 4 ..\build\apps\demo\Release\demo2_slope.exe -elev %cd%\data\dem.tif -nbr %cd%\neighbor\moore.nbr -slp %cd%\out\temp.tif
```

Also, run the `full_test.bat` in the PaRGO root directory to check if everything is OK.

Note the "\PaRGO\vs2010\build\apps\spatial\`Release`\fcm.exe" has significant better performance than the `Debug` path.

5. Development Based on PaRGO & MPI

PaRGO encapsulates lots of parallel details (MPI functions) to provide serial programming experience. If you are unfamiliar with MPI and parallel programming, you will find it relatively easy to use PaRGO. If you are familiar with MPI, you will also find it convenient to make parallel program.

Take the `Reclassify` algorithm in `\PaRGO\apps\demo\demo1` as an example, here we explain how to develop your algorithm based on PaRGO.

5.1 Create a new program

1. create program files:
    - `reclassifyOperator.h` and `reclassifyOperator.cpp`: where you write your algorithm.
    - `demo1_reclassify.cpp`: where you write the main() function

   in a proper folder (i.e., `\PaRGO\apps\demo\demo1`).
2. modify the `CMakeLists.txt` of the created folder (`\PaRGO\apps\demo\CMkaeLists.txt`) and that of its parent folder (`\PaRGO\apps\CMakeLists.txt`). For a new algorithm, replace the "demo"-related words in the following code.

```
FILE(GLOB DEMO1FILES ./demo1/*.cpp)
SET(DEMO1FILES ${DEMO1FILES} ${GPRO_SRCS})
ADD_EXECUTABLE(demo1_reclassify ${DEMO1FILES})

SET(DEMO_TARGETS demo1_reclassify)
```

3. re-compile the PaRGO project using the `cmake` command in Section 3.2.

Now you can open Visual Studio and start to program in the PaRGO way!

5.2 Write a local geocomputation algorithm

To write a simple local algorithm, you don't have to know any parallel programming knowledge. Please refer to the `reclassify` algorithm in `\PaRGO\apps\demo\demo1` for a simple local algorithm.

5.3 Write a focal geocomputation algorithm

The main difference between local and focal algorithms is that the focal ones require neighborhood calculation. Please refer to the `slope` algorithm in `\PaRGO\apps\demo\demo2` as a simple focal algorithm.

5.4 MPI basics

For more complex algorithms, some MPI knowledge is necessary.

MPI is the Message Passing Interface. By using MPI functions, multiple processes are assigned with each others' computation tasks, and communicate with each other to exchange intermediate results. Algorithms in PaRGO almost all include the following mostly used functions. While it's fine to copy some of the functions from an existing algorithm in PaRGO (e.g., the `reclassify` algorithm) to write a simple local geocomputation algorithm, it's better to understand some basic functions before coding.

1. `MPI_Init(int* argc, char*** argv)`

   Initialize the calling MPI process's execution environment.

2. `MPI_Finalize()`

   Terminate the calling MPI process's execution environment.

3. `MPI_Comm_size(MPI_Comm comm, int *size)`

   Retrieve the total number of processes available.

4. `MPI_Comm_rank(MPI_Comm comm, int *rank)`

   Retrieve the rank of the calling process.

5. `MPI_Barrier(MPI_Comm comm)`

   Block the calling process until all processes reached a barrier.

6. `MPI_Wtime()`

   Return high-resolution elapsed time (second).

Some inter-process communication functions may be necessary when an algorithm needs to exchange intermediate result between processes.

- `MPI_Bcast`, `MPI_Send`, `MPI_Receive`: MPI Broadcast and Collective Communication · MPI Tutorial](https://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/)
- `MPI_Reduce`, `MPI_Allreduce`: MPI Reduce and Allreduce · MPI Tutorial
- `MPI_Allgather`: MPI Scatter, Gather, and Allgather · MPI Tutorial

Full API document please see MPI Reference - Message Passing Interface | Microsoft Docs.

Reference materials: Tutorials · MPI Tutorial

5.5 Load-balancing

The greatest upgradation of PaRGO V2 is support for load-balancing. In PaRGO V1, the only way to allocate the computational tasks is to divide the input raster layer (i.e., data domain) into multiple equal parts, known as the "Equal-Area" load-balancing strategy. The load-balancing strategy proposed in PaRGO V2 is based on the concept of the spatial computational domain, which is a raster layer with computational intensity. The crux of the proposed strategy is to equally divide the spatial computational domain so that each part has the same summed computational intensity.

To utilize the proposed load-balancing strategy, a `ComputeLayer` instance should be initialized. A `ComputeLayer` has the same extent as, while typically has coarser resolution than the input `RasterLayer`. Three modes to fill the spatial computational domain are provided in PaRGO V2.

1. Intensity ratio mode.

   Set the computational intensity for **empty** (e.g., NoData) and **non-empty** cells in the `RasterLayer`.

2. Estimate function mode.

   Use the `Transformation` class to estimate the computational intensity for every `ComputeLayer` cell.

3. Preliminary experiment mode.

   Firstly, record the execution time of the algorithm in a rough run, and write to a TIFF file.

   Then, read the TIFF file of recorded time as the `ComputeLayer`.

See the `\PaRGO\apps\spatial\fcm` and `\PaRGO\apps\spatial\idw` for detailed usages.

5.6 Debug

Set your algorithm as the `startup project` in VS, and use the `Debug` mode to try out your program in **serial** for the first time. If it goes without error, you can start to try it in **parallel**. Take the `demo1_reclassify` algorithm running with 4 processes as an example, the property settings should be like the following.

MPI Cluster Debugger

It is recommended to debug using MPI Cluster Debugger for better error locating, in which you can see the outputs of every processes in separate windows. To configure the MPI Cluster Debugger, right click your program in the solution explore and open **properties -> Configuration Properties -> Debug -> MPI Cluster Debugger**.

- Run Environment: `localhost/4`
- Application Parameter: `-input D:\data\dem.tif -output D:\data\output.tif`

Two optional properties can specify the path and version of MPI and GDAL:

- MPIExec Command: `"C:\Program Files\Microsoft MPI\Bin\mpiexec.exe"`
- MPIExec Arguments: `-env PATH C:\lib\gdal\2-4-4-vs2015x64\bin`

Local Windows Debugger

If you only want to use one window for outputting, you can choose the `Local Windows Debugger`, and set:

- Command: `"C:\Program Files\Microsoft MPI\Bin\mpiexec.exe"`
- Command Arguments: `-n 4 "$(TargetPath)"`

5.7 Run

Programs compiled in the `Release` mode would have significantly better performance than the `Debug` mode. Switch to the `Release` mode in your VS, right click your project and `Build` it, and executable files would be generated at path like `C:\src\PaRGO\vs2010\build\apps\spatial\Release\fcm.exe`. You can run it through command line or just in VS.

6. Usage of Current Operators in PaRGO

---

you can refer to the `full_test.bat` for usage of some operators.

Direct execution of current operators will gets you their usages, like:

```
C:\src\PaRGO\vs2010\PaRGO>..\build\apps\morphology\Release\slope
.exe
FAILURE: Too few arguments to run this program.

 Usage: slope -elev <elevation grid file> -nbr <neighbor
definition file> -slp <output slope file> [-mtd <algorithm>]
The available algorithm for slope are:
         FD: (default) Third-order finite difference weighted by
reciprocal of squared distance
         FFD: Frame finite difference
         MD: Maximum downslope
         SD: Simple difference
         SFD: Second-order finite difference
         TFD: Third-order finite difference
         TFDW: Third-order finite difference weighted by
reciprocal of distance

 Or use the Simple Usage: slope <elevation grid file> <neighbor
definition file> <output slope file> [<algorithm>]

Example.1. slope -elev /path/to/elev.tif -nbr /path/to/moore.nbr
-slp /path/to/slp.tif
Example.2. slope -elev /path/to/elev.tif -nbr /path/to/moore.nbr
-slp /path/to/slp.tif -mtd SD
Example.3. slope /path/to/elev.tif /path/to/moore.nbr
/path/to/slp.tif
Example.4. slope /path/to/elev.tif /path/to/moore.nbr
/path/to/slp.tif TFD
```

7. Common Problems

1. Fatal error when running operators of PaRGO.

   Error message like

```
Aborting: mpi appplication on DESKTOP-XXXXXX is unable to
connect to the smpd manager on (null):60490 error 1722

job aborted:
[ranks] message

[1] fatal error
Fatal error in MPI_Init: Other MPI error, error stack:
MPI_Init(argc_p=0x00000045F051F548,
argv_p=0x00000045F051F550) failed
RPC XXXXXX (errno 1722)
```

may be due to wrong configuration of MPI, or using different MPI versions for compiling and running. This reason may also cause building errors in Visual Studio like `cannot open source file "mpi.h"` and `error LNK2019: unresolve external symbol...` . To solve this problem:

Firstly check if the system environment variables regarding MPI are set correctly (see Section 2.4 MS-MPI). Do not mix the MPIs from different distributions. Especially, do not use the MPI from Microsoft HPC pack.

You can check the configuration of your program in VS to see if the MPI configurations are matched. Right click the project in the "Solution Explorer" and click "properties":

- In **Configuration Properties -> C/C++ -> General -> Additional Include Directories**, append `C:\Program Files (x86)\Microsoft SDKs\MPI\Include`.
- In **Configuration Properties -> Linker ->**
    - **General -> Additional Library Directories**, append `C:\Program Files (x86)\Microsoft SDKs\MPI\Lib\x86`
    - **Input -> Additional Dependencies**, append `msmpi.lib`

Secondly check if the compiling and running environments matches. The same MPI versions should be used for both compile (Section 3.2) and running (Section 4). The same VS versions should be used for both compile (section 3.2) and build (Section 3.3).