CEUR-WS.org/Vol-2534/31_short_paper.pdf

Framework-based Approach to Implementation of High-Performance Image Processing Library

Evgeny V. Rusin

[1] Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia, rev@ooi.sscc.ru

Abstract. Framework-based approach to the implementation of high-performance image processing library is suggested. The implementation of prototype libraries for doing processing on computational cluster and GPU is described.

Keywords: remote sensing, image processing, high-performance computing, parallel computing, computational cluster, GPGPU, library of subprograms, framework.

1 Introduction

In recent years, the characteristics of remote sensing of the Earth from space are, on the one hand, the increase in spatial resolution of satellite images, and on the other, the use of hyper spectral survey with a large number of spectral bands. The multispectral data composed of the images obtained from several spectral channels can be hundreds of megabytes in size. While the data from relatively small areas can be handled by the average desktop computers, the solution of large-scale and global tasks of geodata analysis demands the approaches based on high-performance technologies [1] such as parallel, cluster, and distributed computing or calculations on GPU or Intel Xeon Phi. Today, there are a large number of the libraries of image processing subroutines [2-5], some of which support the use of modern facilities of high-performance computing. All of them, however, have one limitation which is significant from our point of view, the lack of extensibility, i.e. impossibility to introduce additional algorithms to the library; the set of the algorithms provided by a typical library is limited to the set implemented by the developer. This circumstance strongly limits the possibility of using existing libraries in the research projects aimed at the creation of new methods and algorithms for satellite data processing. In the present work, we discuss the "framework" approach to the architecture of image processing library allowing to avoid the limitation.

2 Framework Approach

Extensibility is one of the main advantages of the program systems based on framework architecture: image processing library constructed in the form of a framework consists of generic subroutines each of which implements its family of processing algorithms (for example, the family of "pixel-to-pixel" algorithms); such a generic subroutine contains the code common for all the algorithms of the family. The specific algorithm of the family is implemented by the class/subroutine called by the generic subroutine and does, for example, calculation of value of one pixel of result. By parametrizing the corresponding generic subroutine with different classes/subroutines of specific algorithms, a user gets ready implementations of processing algorithms of the family. And, if the interface between the generic subroutine and class/subroutine of a specific algorithm is known (published), then the user of the library can create specific classes/subroutines extending the library. An important issue in the design of such a system is performance, because the overhead of growth of the abstraction level of calculations should not be significant.

Based on the reasoning above, two prototype libraries for high-performance image processing were created.

3 ParImProLib Library

ParImProLib [6] is the library for doing image processing on the cluster multiprocessor systems. When designing the library, the following methodological principles and caused by them technical solutions were used:

1. Maximum hiding the use of parallelism from library users. The program using the library should look like average "sequential" program: all the code dealing with parallel environment (initialization/deinitialization of the environment, node self-identification among all executing nodes, internodal transfers and synchronization) should be encapsulated in the library. At the same time, full hiding parallelism from user would result in potential inefficiency of the program systems based on the composition of algorithms (algorithm A requires one way of parallelization; B,

Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

other; and the composition of A and B, the third, perhaps the same as the first or second). Besides, a user, as a rule, has some general thoughts on how to parallelize an algorithm efficiently; for example, if the algorithm to parallelize is the composition of three filters with the kernel of 5×5 pixels, then the best parallelization is cutting an image onto strips with six-pixel overlapping. Therefore, to provide the most efficient parallelization of an algorithm, the library allows user to specify how the image will be represented on the set of processing nodes:

– Full copy of the image on each processing node.

– Cutting image on horizontal non-overlapping strips: each node stores "its" strip for processing.

– Cutting on horizontal strips overlapping in given number of pixels.

2. Framework architecture which allows both to avoid duplication of the program code implementing parallel algorithms and to provide extensibility of the library.

3. The code of the library has to minimize the inevitable overhead due to the growth of the abstraction level of the computational model. For this purpose, C++ language was chosen for implementation which provides the possibility of efficient generalization by means of the templates' mechanism.

4. Portability of the library to the wide range of the cluster computers which is provided by the choice of C++ and MPI (the standard of parallel programming for the computers with distributed memory) as development tools.

Basic elements of the library are the following classes:

`Image` is the class implementing the abstraction of image. As one of its responsibilities, it (transparently for user) guarantees that the distributed representation of image data on the set of processing nodes is consistent and up-to-date.

`NeighborhoodManipulator` is the class implementing the abstraction of the manipulator with the neighborhood of pixel. Allows to implement processing algorithms in isolation from concrete parameters of the image (the physical sizes, representation on the processing nodes and so forth), in the terms of the neighborhood of pixel being processed. The use of this abstraction provides the extensibility of the library by means of the creation of new processing functions compatible with the library. The following fragment of the C++ code shows **`SobelFilter`** class, implementation of the well-known Sobel filter which is compatible with the library:

```
1. class SobelFilter {
2. public:
3.    unsigned LeftMargin()   const { return 1; }
4.    unsigned RightMargin()  const { return 1; }
5.    unsigned TopMargin()    const { return 1; }
6.    unsigned BottomMargin() const { return 1; }
7.
8.    unsigned operator()(const NeighborhoodManipulator& n) {
9.       return abs(
10.              1 * n.PixelRelativeToCurrent(-1, -1)
11.            + 2 * n.PixelRelativeToCurrent(-1,  0)
12.            + 1 * n.PixelRelativeToCurrent(-1,  1)
13.            - 1 * n.PixelRelativeToCurrent( 1, -1)
14.            - 2 * n.PixelRelativeToCurrent( 1,  0)
15.            - 1 * n.PixelRelativeToCurrent( 1,  1))
16.           + abs(
17.              1 * n.PixelRelativeToCurrent(-1, -1)
18.            + 2 * n.PixelRelativeToCurrent( 0, -1)
19.            + 1 * n.PixelRelativeToCurrent( 1, -1)
20.            - 1 * n.PixelRelativeToCurrent(-1,  1)
21.            - 2 * n.PixelRelativeToCurrent( 0,  1)
22.            - 1 * n.PixelRelativeToCurrent( 1,  1)));
23.   }
24. };
```

– Lines 3-6 define the size of the neighborhood affecting the result at a pixel; the library needs it to calculate the result at the pixels on the margins of image and on the margins of strips for "distributed" representation of image. For Sobel filter, all four sizes are ones as the filter accounts only the closest neighbors.

– Lines 8-23 describe the filter in the terms of the processing of single pixel. The parameter of this operation is the neighborhood manipulator which, by means of the **`PixelRelativeToCurrent(i, j)`** method, provides the access to the neighbor pixels (that is, displaced by **`i`** pixels horizontally and **`j`** pixels vertically from the pixel being processed).

The following fragment of code shows how the filter can be applied to image:

```
1. Image im("image.jpg", PartitioningInfo pi(CutWithOverlap, 1));
2. im.DoNeighborhoodToPixelOperation(SobelFilter());
```

– Line 1 reads the image from disk file and distributes it among processing nodes as strips overlapping in one pixel.

– Line 2 applies to the image generic neighborhood-to-pixel operation parameterized with the object of the class implementing Sobel filter. The generic operation, simultaneously for all strips, will call processing operator consequently for each pixel.

– Generally speaking, the overlapping of strips specified in line 1 is not strictly obligatory as the code of line 2, if necessary, will load to each node the missing data from neighbor nodes. However, it does make sense as the code optimization which the user of the library can perform, having knowledge what operations he/she wants to apply.

Distinctive and important point in using and extending the library is that a user does not need to know the MPI model and to understand the details of internodal exchange in a cluster. The given approach showed its efficiency:

1. The performance of library-compatible implementation of the algorithm for circle structure detection on aerospace images [7] is about 10 percent less compared to the implementation "from scratch" (that is in pure C++ and MPI, with the direct access to image data). At the same time, the creation itself of parallel program with the use of the developed framework is much simpler than the one "from scratch".

2. The efficiency of library-compatible implementation of the algorithm for circle structure detection is about 95% when executing on 8 nodes of NKS-30T+GPU cluster of the Siberian Supercomputer Center (SSCC) hosted in ICM&MG SB RAS [8].

4 SSCCIPGPU Library

In recent years, there is a great practical interest to the use of modern graphic processors (Graphics Processing Unit, GPU) as the general-purpose calculator. Generally speaking, GPU is oriented to the efficient solution of the tasks of computer graphics, in particular it contains the hardware functions allowing to do mass calculations (same operations over the large volume of data) efficiently (with the productivity of hundreds of gigaflops). These opportunities allow to use GPU in the tasks which are unrelated to visualization but also based on mass calculations, for example in image processing and analysis. In a number of practical problems, GPU calculations provided 70-fold acceleration compared to CPU calculations, which corresponds to the performance typical for supercomputers. The concept of general-purpose calculations in GPU also received the support of GPU vendors (e.g. CUDA technology from NVIDIA) which makes available the creation of GPU programs in high level languages without the knowledge of coprocessor architecture. This fact, as well as the low cost of modern GPUs, make them popular equipment of the modern supercomputer centers; thus, the main computing power at the moment of the SSCC SB RAS is the hybrid cluster NKS-30T+GPU which includes 40 nodes, each is equipped with three GPU NVIDIA Tesla M 2090 with Fermi architecture (compute capability 2.0), 512 kernels, and 6 Gbytes of GDDR5 memory. All of this makes important the development of software for the GPU involvement in the processing and the analysis of remote sensing data.

The principles on which the development of SSCCIPGPU [9] library was based have much in common with the above-stated principles on which the earlier ParImProLib library was created. We will add only that:

1. C++ and CUDA was chosen as the development tools, which made the library portable and allowed to implement efficiently the extensible architecture of framework.

2. The multilevel hierarchy of GPU memory makes desirable the possibility to choose how to store images and algorithm parameters: in global, textural, or constant memory of GPU.

3. The complexity of the programming model of graphic processor makes desirable the possibility to choose between the use of synchronous and asynchronous CUDA API, and also to choose how to distribute calculations between several GPUs of single computing node (multi-GPU processing).

The following fragment of the code shows the **`SobelFilter_GPU`** class, GPU implementation of the Sobel filter:

```
class SobelFilter_GPU {
public:
   __host__ unsigned LeftMargin()   const { return 1; }
   __host__ unsigned RightMargin()  const { return 1; }
   __host__ unsigned TopMargin()    const { return 1; }
   __host__ unsigned BottomMargin() const { return 1; }

   __host__ ParamsForDevice ExportParamsForDevice() const { return ParamsForDevice(); }

   __device__ unsigned operator()(const NeiborhoodManupulator& n, const byte* /*params*/) {
      return abs(
              1 * n.PixelRelativeToCurrent(-1, -1)
            + 2 * n.PixelRelativeToCurrent(-1,  0)
            + 1 * n.PixelRelativeToCurrent(-1,  1)
            - 1 * n.PixelRelativeToCurrent( 1, -1)
            - 2 * n.PixelRelativeToCurrent( 1,  0)
            - 1 * n.PixelRelativeToCurrent( 1,  1))
           + abs(
              1 * n.PixelRelativeToCurrent(-1, -1)
            + 2 * n.PixelRelativeToCurrent( 0, -1)
            + 1 * n.PixelRelativeToCurrent( 1, -1)
            - 1 * n.PixelRelativeToCurrent(-1,  1)
            - 2 * n.PixelRelativeToCurrent( 0,  1)
            - 1 * n.PixelRelativeToCurrent( 1,  1)));
   }
};
```

– The implementation of the class for GPU is very similar to the corresponding implementation for cluster with the difference of the use of CUDA directives (`__host__` for the code for CPU; `__device__`, for GPU).

– Line 7 defines the `ExportParamsForDevice` method providing the serialization (translation to block of memory) algorithm parameters. The method is called by the framework to when copying algorithm parameters to GPU memory. In the case of the Sobel filter, the implementation of the method is trivial as the filter has no parameters.

– Line 10, to the described above neighborhood manipulator parameter of the operator of single pixel processing, adds `params` parameter which is a serialized form of algorithm parameters (memory block) copied by a framework to GPU memory.

– The implementation of the operator of processing (line 10-25), as a rule, deserializes (restores) algorithm parameters from `params` memory block (the step is absent for the trivial Sobel filter) and performs the processing of single pixel using the manipulator of the neighborhood.

The following fragment of code shows how the filter can be applied to image:

```
1. Image im("image.jpg");
2. im.DoNeighborhoodToPixelOperation<Global1D, Synchronous>(SobelFilter_GPU());
```

which is very similar to the corresponding code for cluster, except that line 2 specifies the "one-dimensional" access to image data in global GPU memory and the use of synchronous CUDA API for processing. Again, we specially note that the user of the library does not need to have deep knowledge in GPU programming to use and extend the library.

Similar approach was also used for the implementation of multi-GPU processing. Here, the user has to specify the strategy of parallelization of calculation between several graphic accelerators (using OS multi-threading versus switching active GPUs in single OS thread context).

The following tells about the efficiency of the approach:

1. The performance of the library-compatible implementation of the algorithm for circle structure detection is the same as the one of the implementation "from scratch" (that is in pure C++ and CUDA). At the same time, the creation itself of parallel program with the use of the developed framework is much simpler than the one "from scratch".

2. The library-compatible implementation of the algorithm for circle structure detection, when running on single GPU NVIDIA Tesla M 2090 of NKS-30T+GPU cluster, is about 90 times faster than similar CPU implementation running on single Intel Xeon X5670 (2.93 GHz) processor of the cluster.

5 Conclusion

The results obtained demonstrate that the framework approach to the implementation of high-performance image processing libraries is promising. Currently, the created prototype libraries are used when developing an experimental framework of high-performance image processing SSCCIP [10], a part of an experimental cloud framework [11]. In further plans of the authors is the propagation of the approach to the image processing on Intel Xeon Phi processors with which the new NKS-1P cluster of SSCC SB RAS is equipped.

Acknowledgements. This work was conducted within the framework of the budget project 0315-2016-0003 for ICM&MG SB RAS with the supercomputer facilities provided by Siberian Supercomputer Center SB RAS.

References

[1] Buchnev A., Pyatkin V., Rusin E.V. Software Technologies for Processing of Earth Remote Sensing Data // Pattern Recognition and Image Analysis. 2013. Vol. 23. No. 4. P. 474-480.

[2] OpenCV. https://opencv.org.

[3] TensorFlow. https://www.tensorflow.org.

[4] Computer Vision Toolbox. https://www.mathworks.com/help/vision.

[5] Intel® Integrated Performance Primitives (Intel® IPP). https://software.intel.com/intel-ipp.

[6] Rusin E.V. Object-Oriented Parallel Image Processing Library // Parallel Computing Technologies. PaCT 2009. Lecture Notes in Computer Science. 2009. Vol. 5698. P. 344-349.

[7] Alekseev A.S, Pyatkin V.P., Salov G.I. Crater Detection in Aero-space Imagery Using Simple Nonparametric Statistical Tests // Lecture Notes in Computer Science. 1993. Vol 179. P. 793-799.

[8] SSCC SB RAS. http://www.sscc.icmmg.nsc.ru/hardware.html.

[9] Rusin E.V. Tehnologii obrabotki dannyh distancionnogo zondirovanija Zemli na gibridnom klastere NKS-30T+GPU [Technologies for processing Earth remote sensing data on the NKS-30T+GPU hybrid cluster] // Interekspo Geo-Sibir'. 2016. Vol. 4. No. 1. P. 46-49.

[10] Rusin E.V. SSCCIP – A Framework for Building Distributed High-Performance Image Processing Technologies // Parallel Computing Technologies. PaCT 2011. Lecture Notes in Computer Science. 2011. Vol. 6873. P. 467-472.

[11] Buchnev A., Kim P., Pyatkin V., Pyatkin F., Rusin E. Framework of cloud web services for processing remote sensing data // E3S Web of Conferences. 2019. Vol. 75. Paper 03001.