University of Groningen

Improving Utility of GPU in Accelerating Industrial Applications With User-Centered Automatic Code Translation

Yang, Po; Dong, Feng; Codreanu, Valeriu; Williams, David; Roerdink, Jos B. T. M.; Liu, Baoquan; Anvari-Moghaddam, Amjad; Min, Geyong

Published in:
IEEE transactions on industrial informatics

DOI:
[10.1109/TII.2017.2731362](https://doi.org/10.1109/TII.2017.2731362)

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2018

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):

Yang, P., Dong, F., Codreanu, V., Williams, D., Roerdink, J. B. T. M., Liu, B., Anvari-Moghaddam, A., & Min, G. (2018). Improving Utility of GPU in Accelerating Industrial Applications With User-Centered Automatic Code Translation. *IEEE transactions on industrial informatics*, 14(4), 1347-1360.
<https://doi.org/10.1109/TII.2017.2731362>

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Improving Utility of GPU in Accelerating Industrial Applications With User-Centered Automatic Code Translation

Po Yang¹, Member, IEEE, Feng Dong, Valeriu Codreanu, David Williams, Jos B. T. M. Roerdink², Senior Member, IEEE, Baoquan Liu, Amjad Anvari-Moghaddam³, Member, IEEE, and Geyong Min, Member, IEEE

Abstract—Small to medium enterprises (SMEs), particularly those whose business is focused on developing innovative products, are limited by a major bottleneck in the speed of computation in many applications. The recent developments in GPUs have been the marked increase in their versatility in many computational areas. But due to the lack of specialist GPU programming skills, the explosion of GPU power has not been fully utilized in general SME applications by inexperienced users. Also, the existing automatic CPU-to-GPU code translators are mainly designed for research purposes with poor user interface design and are hard to use. Little attention has been paid to the applicability, usability, and learnability of these tools for normal users. In this paper, we present an online automated CPU-to-GPU source translation system (GPSME) for inexperienced users to utilize the GPU capability in accelerating general SME applications. This system designs and implements a directive programming model with a new kernel generation

scheme and memory management hierarchy to optimize its performance. A web service interface is designed for inexperienced users to easily and flexibly invoke the automatic resource translator. Our experiments with nonexpert GPU users in four SMEs reflect that a GPSME system can efficiently accelerate real-world applications with at least 4× and have a better applicability, usability, and learnability than the existing automatic CPU-to-GPU source translators.

Index Terms—Automatic translation, graphics processing unit (GPU), parallel computing, usability.

I. INTRODUCTION

SMEs, particularly those whose business is focused on developing innovative products, are subject to many pressures in maintaining and growing their market share and ensuring that their products remain competitive in an age of rapid technological change. In many high-tech fields, users are experiencing a huge growth in data, with increases in quantity, resolution, variety, etc., while the work often presents significant time constraints on the associated data processing. This leads to a continual upward pressure on computational resources and, indeed, the speed of computation is now a major bottleneck that dramatically limits the applicability of available technology in many applications in SMEs.

The major challenges in many high-tech applications in SMEs relate to a huge growth in data processing requirements through increases in quantity, resolution, variety, etc., demanded by general applications. Parallel computing techniques [1] have gained wide popularity among researchers and developers to overcome these challenges. The concept of parallel computing is to split large problems into small components and distributing them among multiple processors. The conventional parallel computing takes place using multicore CPUs or via distributed, grid, high-performance computers. The recently remarkable rise in the performance of graphics processing unit (GPU) [2] offers a very attractive alternative, which can handle many demanding tasks by only harnessing local computing resource in low-cost computer platforms.

The most important development in GPUs in recent years has been the marked increase in their versatility. Their capabilities are now much more widely applicable and they have become used in many computational areas—this is known as General Purpose GPU programming (GPGPU) [3]. OpenCL [4] and

Manuscript received September 30, 2016; revised May 15, 2017; accepted June 24, 2017. Date of publication July 24, 2017; date of current version April 3, 2018. This work was supported in part by the European Commission and the Engineering and Physical Sciences Research Council within the iManageCancer Project under Grant 611140, in part by the MyHealthAvatar Project under Grant 600929, and in part by the MyLifeHub Project under Grant EP/L023830. Paper no. TII-17-0238. (Corresponding author: Po Yang.)

P. Yang is with the Department of Computer Science, Liverpool John Moores University, Liverpool L3 5UA, U.K. (e-mail: poyangcn@gmail.com).

F. Dong is with the University of Bedfordshire, Luton LU1 3JU, U.K. (e-mail: feng.dong@beds.ac.uk).

V. Codreanu is with SURFSara, Amsterdam 1098 XG, The Netherlands (e-mail: v.b.codreanu@gmail.com).

D. Williams is with the Faculteit Wiskunde en Natuurwetenschappen, Rijksuniversiteit Groningen, Groningen 9712 CP, The Netherlands (e-mail: david@david-williams.info).

J. B. T. M. Roerdink is with the Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, Groningen 9712 CP, The Netherlands (e-mail: j.b.t.m.roerdink@rug.nl).

B. Liu was with the Department of Computer Science and Technology, University of Bedfordshire, Luton LU1 3JU, U.K. He is now with Huawei UK Research Centre, Cambridge CB4 0FY, U.K. (e-mail: Baoquan.liu@huawei.com).

A. Anvari-Moghaddam is with the Department of Energy Technology, Aalborg University, Aalborg 9100, Denmark (e-mail: aam@et.aau.dk).

G. Min is with the Department of Mathematics and Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter EX4 4QF, U.K. (e-mail: G.Min@exeter.ac.uk).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TII.2017.2731362

NVIDIA's CUDA [5] are two mainly widespread GPU parallel programming languages designed to help users manage GPU utilization. If the capacities of the GPU are harnessed properly, the achieved speedup can be significant. But the parallelization of the CPU code for execution on GPUs is not light and handy to general users. This process requires an in-depth knowledge of the complex underlying GPU architecture and the GPU memory optimization schemes. These skills are still in short supply to nonexpert GPU users. It is desirable to have a cost-effective approach that enables inexperienced users to easily utilize GPU technology for accelerating their applications.

An automatic CPU-to-GPU source translation technique can be a candidate to make GPU technology more accessible to the inexperienced user. To date, numerous automatic CPU-to-GPU source parallelization translation tools [9]–[27], including algorithmic skeleton based [14]–[16], polyhedral model based [9]–[13], or directive based [17]–[23], have been developed for academic and commercial use. While their acceleration is promising, utilizing them by normal users in general real-world applications is still challenging. Many tools are originally for research purposes with a nonavailability of public access and a limited applicability of supporting different algorithm structures. Simultaneously, the usability and learnability of these tools are not prospective, since their attentions are mostly on improving speedup performance rather than making them more accessible to general users.

Also, the diverse types of algorithms and loops in general applications pose significant challenges toward the use of these tools. So, no existing CPU-to-GPU source translation tools are reported in the literature to provide an outstanding solution for nonexpert GPU users with reasonable acceleration, wide applicability, good usability, and well learnability.

In this paper, we propose a web service based automated CPU-to-GPU source translation system (GPSME) for inexperienced users to utilize GPU capability in accelerating general SME applications. We design and implement a directive-based programming model that is capable of carrying out semiautomatic CPU-to-GPU source-to-source translation on moderately priced standard GPU cards and off-the-shelf GPU clusters. A web service based interface is particularly designed for inexperienced users to easily and flexibly invoke the automatic resource translator. Our experiments with nonexpert users from four SMEs reflect that a GPSME system can efficiently accelerate general real-world applications with at least $4\times$, and also have an improved applicability, usability and learnability than existing CPU-to-GPU source translation tools. The main contributions of this paper are as follows.

- 1) A comprehensive requirement analysis of inexperienced users for utilizing GPU technology in general SMEs applications is given. It is beneficial to improve the accessibility and the applicability of existing automatic CPU-to-GPU source translators in real-world applications.
- 2) A web service based automated CPU-to-GPU source translation system, GPSME, is presented and implemented. This tool introduces a new kernel generation scheme and a memory management hierarchy to optimize its performance.

- 3) A thorough performance evaluation of the GPSME system with general SMEs applications has been carried out. The results suggest that the proposed tool can effectively and efficiently accelerate general real-world applications, and has improved applicability, usability, and learnability over existing automatic CPU-to-GPU source translation tools [23]–[30].

The rest of the paper is organized as follows. Section II reviews notable automatic CPU-to-GPU source translators. Section III analyzes the general requirement of inexperienced GPU users. Section IV presents the design and implementation of the GPSME system. Section V shows the experimental validation results. Section VI gives a conclusion and brief details of future work.

II. RELATED WORK

A polyhedral model [9], [10] for performing loop transformations has been the basis of early attempts for automatic optimization and parallelization of CPU programs. With the emergence of GPUs, the polyhedral model is adopted to develop efficient CPU-to-GPU source translators, such as Pluto [11], R-Stream [12], Par4All [13], and PPCG [24]. They translate source code with affine loop structures by performing dependence analysis and loop transformations. These tools normally require little or no input from the users, and have a promising acceleration performance; but they have some drawbacks on applicability and adaptability. R-Stream supports C-to-CUDA compilation but is not publicly available yet. Pluto automatically generates CUDA kernel code; but the CUDA host code has to be written manually by users. Par4All compiler is a public available tool supporting automatic integrated compilation of applications for hybrid architectures including GPUs. Yet some restrictions and code restructuring might be required for reaching a promising performance.

Algorithmic skeleton based tools adopt an idea of generating an efficient target code by a specific algorithm class. Examples of such tools are SkePU [14], SkelCL [15], and Bones [16]. Each algorithm skeleton is coded as a template of specific algorithm class on target architecture. These tools have highly optimized library implementations for classes of algorithms instead of individual algorithm, as a result of dramatic acceleration. Both algorithmic skeleton and polyhedral model based tools have a well usability since they do not require users having deep GPU knowledge to identify parallel region and memory transfer in CPU code. Yet, their applicability is relatively narrow and highly sensitive to the characteristics and data structure of CPU algorithms. This shortcoming limits their wide acceptances by general users.

For allowing automatic CPU-to-GPU translators to be more applicable, directive-based source translators [17]–[23] became popular. By using these tools for generating the target GPU code, users only need to provide some basic annotations about parallelism exploitation and also annotations that deal with data transfer. CUDA-lite [17] introduces some directives to improve the memory hierarchy of CUDA by directly inserting the directives into the CUDA code. hiCUDA [18] provides a set of

TABLE I
COMPARISON OF PROPERTIES OF TYPICAL AUTOMATIC PARALLELIZATION SOURCE TRANSLATION TOOLS

	Acceleration	Applicability	Usability	Adaptability
PoCC [10]	1×–10×	8 benchmarks	C-to-optimized C	Open source, programming in Linux
Pluto [11]	2×–12×	13 benchmarks	OpenMP to C	Open source, programming in Linux
R-Stream [12]	>10×	Matrix multiplication, Gauss Seidal	C-to-C (Binary)	Nonpublic available source
Par4All [13]	2×–128×	6 benchmarks	C-to-CUDA/OpenCL	Open source, programming in Linux
SkePu [14]	10×	7 skeletons (map, mapArray, reduce, etc.)	C-to-CUDA/OpenCL	Nonpublic available source
HMPP [25]	–10×	Multiple types of loops	C-to-CUDA/OpenCL	Commercial product
Bones [16]	1×–13×	8 benchmarks	C-to-CUDA/OpenCL	Open source, programming in Linux
CUDA-lite [17]	2×–17×	MRI-FHD, TPACF	CUDA-to- optimized CUDA	Nonpublic available source
hiCUDA [18]	–18×	9 benchmarks, MCML	C to CUDA	Open source, programming in Linux
MINT [19]	10×–16×	Stencil computing	C to CUDA	Open source, programming in Linux
OpenMPC [20]	–50×	JACOBI, SPMUL	OpenMP-to-CUDA	Nonpublic available source
PGI [21]	10×	Multiple types of loops	Fortran, C, C++ to CUDA	Commercial product
PPCG [24]	1×–100×	30 benchmarks	C to CUDA	Open source, programming in Linux

TABLE II
DETAILED INFORMATION OF INEXPERIENCED GPU USERS FROM INDUSTRY

	Language	Product Area	Problem
IME	C++	Image forgery detection	Time-consuming task in detecting suspicious and altered parts of the image or video.
B3C	C++	Virtual physiological human	Many VPH applications are computationally demanding.
ROTA	C++	Augmented reality book	Image processing speed in real-time AR books.
AnSmart	C++	Eye tracking	Medical image analysis in diagnosing eye diseases.

pragmas mapping to typical CUDA operations for programmers. CUDA code generated from hiCUDA is optimized by operating global memory and transformations to leverage the complex memory hierarchy. But a prerequisite of hiCUDA is that users have to understand sufficient GPU knowledge for specifying the threads and thread blocks. OpenMPC [20] project proposes a Cetus compiler framework for translating standard OpenMP shared-memory programs into CUDA-based GPGPU programs. Despite the significant speedup of OpenMPC, its adoption was slowed by a manual revision of input source as OpenMP programs. Similarly, PGI compiler [21] accelerates applications written in C++ by adding standard OpenACC [22] directives; but its pragmas are far too complex, and the GPGPU code it outputs is almost unreadable. Besides, MINT [19] is a very easy-to-use C-to-CUDA source translator containing only five types of pragmas. It is designed for speeding up only stencil computations on NVIDIA GPUs. This translator accepts C source input with some intuitive MINT directives to generate highly optimized CUDA C, which may produce performance gains of up to 10×. So, directive-based tools have a better applicability in dealing with complex CPU algorithms due to their flexibility of adding annotations in the CPU source code. Yet, their usability is not very good, since users have to identify the parallelization region and manage the complex memory hierarchy by themselves. Also, hard-learning directives and unreadable output code in the tools increase the difficulties for inexperienced users to harness them.

III. REQUIREMENT ANALYSIS

This section identifies and analyzes general expectations of inexperienced GPU users on an automatic CPU-to-GPU source translator for accelerating their applications. The participated

users are from four companies in the EU funded project GPSME [29]: ImageMetry Ltd (IME) [30], Biocomputing Competence Centre (B3C) [31], Rotasoft [32] and AnSmart [33]. Their products involve a wide applicable area including image forgery detection, augmented reality book, and virtual physiological human. Table II illustrates the applicable area of each company and the problem they face. For collecting a general requirement of their nonexpert GPU users, IME Ltd communicated with the other three companies and collected their feedbacks in three months through emails or project meetings. Inexperienced GPU users have some common objectives such as:

- 1) no need on having an in-depth understanding of GPUs;
- 2) full or semiautomatic CPU-to-GPU source translation;
- 3) support C++ programming language;
- 4) support either CUDA or OpenCL;
- 5) efficient speedup performance and no accuracy loss;
- 6) source code protection;
- 7) report the system process and error diagnostics.

Regarding the above general objectives, it appears that existing CPU-to-GPU source translators in Table I hardly satisfy the full needs. Nonexpert GPU users expect a system that enables them to quickly take advantage of current GPU capability to effectively and economically speed up their products. An explicit requirement analysis of their expectations on this system is given as below.

- 1) *Acceleration*: They expect their general CPU applications to be accelerated significantly on moderate hardware platforms. Nonexpert GPU users are more interested in actual time saved in their applications instead of a high speedup ratio of GPU over CPU. Existing CPU-to-GPU translators focus more on the improvement of their speedup ratio for reflecting their parallelization efficiency. Their acceleration results are

mostly achieved by running simple C code samples though a high-level GPU hardware. Their utilization in practical applications cannot reach and can even decrease the performance since some indispensable CPU source code revisions are required. So, the acceleration capability of the GPSME system need be evaluated by practical applications and not only the sample code for the parallel region.

- 2) *Applicability*: They look forward to a system with wide applicability, which can solve time-consuming problems in various types of products. Among the existing CPU-to-GPU translators, algorithm skeleton based tools like Bones [16] has limited classes so they cannot support the applications with complex or diverse loop types. Directive-based tools like OpenMP [20], hiCuda [18] and PGI [21] have wide applicability guaranteed by flexible usage of standard pragmas. But the learning of these pragmas becomes a hard task for nonexpert users. There has to be a tradeoff of these tools between applicability and directive complexity. The directives of the GPSME system have to be simple but enable supporting all types of algorithms skeleton and loop patterns from their general applications.
- 3) *Usability*: Inexperienced GPU users have strong demands on usability of the GPSME system. First, the input and output languages are essentially to support C/C++ and CUDA/ OpenCL. Second, they suggest using a web service interface to achieve a cross platform (Windows and Linux) usage of code translation. A user file management system with source code protection scheme is required for this interface. Among existing CPU-to-GPU source translators, most of their interfaces are C-to-CUDA based command line tools under Linux. A system with better usability for nonexpert GPU users is expected.
- 4) *Adaptability*: The easy-to-learn nature of the tool is paramount to inexperienced GPU users. GPU technology and programming skills are hard to grasp. The existing CPU-to-GPU source translators still need users to study the usage of the directives. In fact, the simplicity of the directives is crucial to the adaptability of the system. This paper aimed to design simple, flexible, and efficient directives.

IV. GPSME SYSTEM OVERVIEW

A GPSME system is a deliverable of the GPSME [29] EU funded project. The significance of this tool distinguishing from other CPU-to-GPU source translators is that relatively much attention has been paid to the usability and adaptability. Also, driven by the potential difficulties of having a local GPSME installation and the increased visibility of the system, access to GPSME system through a remote web server is more suggested. The main system architecture of the GPSME system includes two components: GPSME web interface and GPSME core library. The communication between these two components is handled through web service. Though the web interface, inexperienced GPU users can upload their CPU source files; execute

Fig. 1. GPSME web interface.

translations by invoking the core library; and download the generated GPU source files.

A. GPSME Web Interface

GPSME web interface is designed and implemented using the ExtJS framework for providing appearance and user experience close to desktop applications. The web application of GPSME includes file explorer, rich text editor, interactive help system, and tabbed information windows. Also, it encrypts users' code to protect their intelligent properties and facilitates a translation of users' source code, not needing anything related to the system locally installed on the users' machines. The GPSME web interface is presented in Fig. 1.

The typical usages of the GPSME web application are as follows.

- 1) The user creates an account with his/her details.
- 2) The user uploads C/C++ source file and necessary header files.
- 3) The user selects the desired output type and initiates the code translation process.
- 4) It takes a few seconds for the GPSME web server to process the user files. If success, the results can be retrieved under the "Processed files" tab.

The user does not run the system locally on a Linux machine but will be instead making use of this remote web server.

- 1) If CPU source code has an external dependence then it must be presented or installed on the remote web server (in addition to user's local machine). Users need to contact the server administrator to get the dependencies installed if so required.
- 2) When uploading a C++ file for parallelization, users must also upload any of their own headers on which the C++ file is dependent. It is assumed that these belong in the same directory as the C++ source file, so users avoid deep paths in their `#include` statements.

B. GPSME Core Library

The design of GPSME core library is inspired by MINT [19], which is developed on Linux by using the ROSE compiler

Fig. 2. Structure of GPSME core library.

framework [34] at Lawrence Livermore National Laboratory. ROSE provides an open-source compiler with API to develop customized source-to-source translators for performing code transformations, analyses, and optimizations. The structure of GPSME core library is similar to MINT [19] but with some extended components. The system structure and translation flow of implementing the core library is illustrated in Fig. 2.

The input of GPSME system is a set of C/C++ source files annotated with GPSME directives. Once source files are read, the ROSE frontend constructs an abstract syntax tree (AST). The core library traverses the AST and queries parallel regions containing data parallel for loops. The Identifier is responsive to identify and classify loop pattern, variables, and device information from the AST. Regarding the identifications, the analyzer and the optimizer investigate the possibility of using predefined approaches in the GPSME system for AST transformations. The translator adopts similar rules in MINT baseline translator to perform transformations on the AST. But it enhances the functionalities of MINT, whose details are presented in Section I. GPSME core library supports both CUDA and OpenCL code by unparsing the transformed AST. Due to the similar code generation procedure between CUDA and OpenCL, this paper mainly discusses the generation of the CUDA code using the GPSME core library.

C. Interaction

The interaction between web application and core library is designed and implemented by utilizing the jQuery Framework and Java Servlet. RESTful web services are deployed to Tomcat 7 servlet container for handling the AJAX requests including file upload, edit, and execute. Fig. 3 illustrates an interactive workflow between web interface and core library at server.

Fig. 3. Interaction between GPSME web interface and core library.

In Fig. 3, users first register and upload their source files through GPSME web interface. They receive a private key for decrypting uploaded source files in their emails. The public key was generated and stored in server database for encrypting uploaded source files. Users need a correct private key to display, edit, save, or translate the source files. In code process, an AJAX request will be sent to the REST service end point and invoke the GPSME core library to execute command line tools. Once execution finished, the outputted files and log files are collected and returned to GPSME web interface. User can view the diagnosis and error reporting information and download the CUDA or OpenCL source files. Meanwhile, some sample codes and tutorials are provided in web interface for users to learn and experience.

D. GPSME Directives

In MINT [19], five types of different directives are employed for the following three main tasks:

- 1) identification of parallel region;
- 2) memory management; and
- 3) kernel generation.

The MINT *for* directives in MINT is the most important since it identifies a parallel *for* loop nest and helps guide optimization and generation of kernel code. The MINT *copy* directives help users manage the separate host and device memory space. However, the utilization of MINT in general applications faces the following three challenges.

- 1) The MINT directives indicating parallel regions and kernel regions are too simple to use for complicated algorithms. The directive *parallel* marking a parallel region must be located immediately behind the directive *copy*. It cannot handle algorithms in which users need to insert source code between these two directives.
- 2) The *copy* directives combining memory allocation and data transfer is too extensive to support high-level storage setup and management. Particularly, in practical

TABLE III
LISTING OF GPSME DIRECTIVES (SOME ARE INHERITED FROM AND MINT [19])

	Directives	Descriptions
Basic pragma	Parallel (MINT)	To identify a region generating a kernel function
	For (MINT)	To mark the succeeding “ <i>for</i> ” loop for GPU acceleration
	Single (MINT)	To indicate serial regions in the GPSME
	Parallel region	To identify a parallel region containing parallel work
Memory management	Copy (MINT)	To express the declaration, allocation, data transfers between the host and device
	CopyByTexture	To create a CUDA texture on a device, and bind or unbind with 2-D data
	CopyMalloc1DArray	To create a CUDA array on a device, associating it with a CUDA texture on the device
	CopyMemcpy2D	To create a CUDA <i>cudaMemcpy2D</i> function to copy a matrix between CPU and GPU memory
	CopyMemcpy2DTo Array	To create a CUDA function <i>cudaMemcpy2DToArray</i> to copy data between CPU and GPU memory
	CopyBindTexture	To bind the created texture memory to a CUDA global array
	Copy2DArrayTo1DArray	To convert the array with different dimensions on the CPU memory buffer
Kernel generation	For, nest, tile, chunksize (MINT)	To generate CUDA kernel for each parallel “ <i>for</i> ” loop with given thread blocks and threads.
	Initialization	To define a one dimensional array for storing the data in a sliding window
	Transfer	To transform the code of putting the data in a sliding window into a local variable within a “ <i>for</i> ” loop
	Remain	To transform the operations on a sliding window from the CPU algorithm to the GPU kernel.
	Assign	To assign the new data to the relevant GPU buffer with the correct index.

applications, it requires separating the operations of memory allocation and data transfer to allow the reuse of the allocated memory for data transfer.

- 3) The MINT kernel generation directives only support stencil computing. Many algorithm skeletons in general applications are beyond stencil computing, which hardly copied by MINT kernel.

For overcoming the limits of MINT directives, the design of GPSME directives consider the practical requirements mentioned in Section II. The detailed information of GPSME directives are reported in our early work on the GSWO programming model [41]. First, the basic GPSME directives are inherited from MINT, as shown in Table III. The primary extension of GPSME directives is that its memory management directives have an enhanced hierarchy. GPSME introduces a set of memory management pragmas to control GPU memory allocation, CPU-to-GPU memory transfer, and CPU memory conversion, respectively. It also provides pragmas to allow for the use of texture memory (in addition to the use of global memory). These new pragmas bring the flexibility and effectiveness to memory management that is needed in general applications for image processing.

Also, GPSME introduces a set of newly defined kernel generation directives. These were designed for sliding window operations based image processing applications by following a typical procedure, which contains *initiation*, *transfer*, *remain*, and *assign*. They are simple and can be applied to all types of parallelizable operations in sliding windows. Our experiments showed that using GPSME directives provides a significant improvement in usability and productivity when compared with other CPU-to-GPU translators.

The final improvement of GPSME directives is that it extends the directive *parallel* of MINT into two directives *parallel* and *parallel region* to distinguish the kernel region from the parallel region. The *parallel region* indicates the start of a parallel region containing the CPU source code for parallelization, whereas the *parallel* marks a loop for generating a GPU kernel function. This

extension is highly similar to the directives *parallel* and *kernels* in the OpenACC standard, but is less complicated and more easy to use by nonexpert GPU users. With these two directives, the GPSME system can support more complicated algorithmic structures than MINT.

V. KEY ENHANCEMENTS

The GPSME system has some functional extensions to MINT [19], including C++ and multfile support, preliminary OpenCL output, and a user-friendly interface. Besides, some key enhancements on improving its acceleration capability and applicability were also implemented.

A. Supporting Triangular Loops

In earlier work [35], the GPSME system with original MINT kernel was applied to the PolyBench [23] benchmarking suite in order to assess the resulting performance increase. During this process, the lack of support for *triangular loops* was identified as an inhibiting factor of the autoparallelization process. MINT kernel was still able to process the outermost loop but this yielded significantly lower performance than that was theoretically obtainable.

An example of such a problematic triangular loop is given in Algorithm 1, and the iteration space is depicted visually in Fig. 4. Note how the initial value of $j2$ in the inner loop is dependent on the current value of $j1$ in the outer loop. Mint was still able to process the input code but generated a noncompilable output that attempted to make use of variables prior to their declaration.

GPSME have, therefore, implemented a triangular loop support as an extension to the MINT. We define a rectangular iteration space over the full range of values which $j1$ and $j2$ can assume, and then overlay a grid of CUDA thread blocks. The CUDA kernel checks whether the current iteration does indeed fall within the triangular part of the iteration space, and skips execution if this test fails.

Fig. 4. Iteration space of the two-level covariance loop from [35].

Algorithm 1: Example of Problematic Triangular Loop.

```

1: #pragma GPSME copy(data, toDevice, M, N)
2: #pragma GPSME copy(mean, toDevice, M)
3: #pragma GPSME copy(symmat, toDevice, M, N)
4: #pragma GPSME parallel
5: {
6:     ... //Some code omitted for brevity
7:     #pragma GPSME for nest(2) tile(16, 16)
8:     for (j1 = 0; j1 < M; j1++)
9:     {
10:        for (j2 = j1; j2 < M; j2++)
11:        {
12:            ... //Some code omitted for brevity
13:        }
14:    }
15: }
16: #pragma GPSME copy(symmat, fromDevice, M, N)

```

With this in mind, a thread block can be categorized as being in one of three states with respect to the number of threads which need to execute.

- 1) *Full*: All threads are part of the triangular iteration space and must be executed. No processing capability is wasted in this scenario.
- 2) *Empty*: None of the threads are part of the triangular iteration space. All threads will fail the membership test implemented in the kernel and return immediately.
- 3) *Half-full*: In this case, the running time of the thread block is determined by the threads that do need to run. Threads that do not need to run must still wait upon those that do, and this represents some wasted processing capability.

This approach has yielded a performance increase greater than that was obtained from OpenACC, and more than 30 times greater than that was obtained from the original Mint. The proportion of half-full blocks decreases as the problem size increases, which limits the impact of the GPU’s relatively poor performance in the presence of divergent operations. The GPSME system has evaluated the triangular loop support on a

sample CPU code for the computation of colon centerlines to be used for the purpose of virtual endoscopy. When running on the CPU, the provided code took approximately 5 min to fully process a CT dataset with a resolution of $512 \times 512 \times 512$ voxels. This code was optimized prior to GPUification such that it ran in 48 s on a single core. The GPSME system was then able to achieve a threefold speed increase bringing the execution time down to 17 s, which was comparable to the 15 s taken by the *four-core* CPU version. A manual implementation was able to further decrease the runtime to only 1.2 s [36].

While the GPSME system was effective in parallelizing this application, the main problem was the large memory usage. Our manual implementation was able to be much smarter about the allocation and copying of memory which led to a significant speed increase. But this manual implementation did take several days of work, whereas the autoparallized version only took 1 h to add the directives and perform some minor debugging.

B. Single-Dimensional Versus Multidimensional Arrays

Another improvement to the GPSME system is that it adds more optimization opportunities when applied to code that use multidimensional arrays. The optimizations are in terms of better register reuse, as well as better shared memory usage. This assumption was evaluated on some of the Polybench tests [23]. For the 2MM and SYR2K tests, a further 25% performance increase is obtained when using two-dimensional (2-D) addressing instead of the default flattened array addressing. The changes from single-dimensional to multidimensional array accesses were done in a manual manner, as in Polybench all tests are written with flattened array accesses. However, with extra hints from the programmer, the GPSME system should be able to treat the single-dimensional arrays as multidimensional ones. An interesting observation is that when faced with the same 2-D arrays in the 2MM and SYR2K tests, the OpenACC compiler reports more than two times worse performance. The reasons for this are not currently clear and will be the subject of some future investigation.

C. Kernel Generation Scheme for SWO

Considering the limitation of the MINT kernel generation, another kernel generation scheme in the GPSME system is designed into our early work (GSWO model [41]) to orchestrate the GPU kernel code generation. Fig. 5 shows an example workflow of the kernel generation scheme for median filter. We have designed new “single” pragmas for kernel code generation, four of which are defined below.

- 1) *Single Initialization* generates the CUDA kernel code that defines a 1-D array with size $I \times J$ for storing the data in the sliding window.
- 2) *Single Transfer* generates the CUDA kernel code to transfer the data of the sliding window into a 1-D array defined in the *Single Initialization* directive.
- 3) *Single Remain* generates the CUDA kernel code that corresponds to the operations on the sliding window.
- 4) *Single Assign* generates the CUDA kernel code that copies the processed data in the sliding window to the relevant GPU buffer obtained via the thread and block IDs.

Fig. 5. Working flow of Kernel Generation Pragmas from [41].

VI. PERFORMANCE EVALUATION

The GPSME system is evaluated on the general applications of four companies from the GPSME project [29]. The evaluation methodology used in this paper is based on the measurement of acceleration ratio between GPU and CPU performance without losing the original algorithm's accuracy. The baseline is the performance of the original CPU code running on conventional hardware without using multithreading. The evaluation platforms were as follows:

- 1) Intel Core i7-2670QM CPU and NVIDIA GeForce GT 540M;
- 2) Intel Core i7-3770K CPU and NVIDIA GeForce GTX 690;
- 3) Intel Core i3-2.1 GHz CPU and NVIDIA GeForce GT 520M;
- 4) Intel Core i7-3.4 GHz CPU and NVIDIA GeForce GTX 680M.

All GPU implementations used NVIDIA GPU SDK version 4.1. OpenMP programs were compiled using Visual Studio 2008, and all computation used double precision.

A. Acceleration Performance

Acceleration is the most important indicator reflecting the performance of the GPSME system. In order to evaluate this, we have compared the execution performance of the original

CPU code, the GPSME system generated GPU code, and the manually generated GPU code. The original CPU codes were provided by inexperienced GPU users from their general applications. They were revised by the users in order to be processed by the GPSME system. The system-generated the machine-generated GPU codes. To better represent the performance of the generated GPU code, we have also performed the CPU-to-GPU code conversion manually. The acceleration performance is shown in Table IV.

SCS: The application from SCS is centerline extraction for a given 3-D model. The code from SCS is based on C++, and also calls VTK functions for centerline extraction. The performance gain from the use of the system is given in Table IV. It appears that the GPSME system can accelerate the application from B3C up to 2–3 times on average, dependent on the GPU devices used.

IME: The application from IME is to produce a camera fingerprint by applying denoising methods to a set of images that are known to come from a given camera. The sample code from IME is based on C++, and aims at implementing a 3×3 median filter for denoising. The image resolution is 3648×2736 . The number of images is 39. The algorithm splits the images into a number region of interests, which can be processed in parallel. Table V shows the performance gain from the use of the GPSME system. It shows that the accuracy of the IME application delivered by the GPU implementation and the CPU implementation is exactly the same, which means that the machine-generated GPU implementation does not negatively impact the camera fingerprint application. Second, it appears that on average the GPSME system accelerates the performance with up to 3–4 times. If only considering the acceleration of the kernel region, the speedup performance can achieve execution times of up to six times shorter than the original CPU application. This proves that the GPU kernel implementation is certainly capable of speeding up the CPU code in the parallel regions.

ROTA: The application from Rotasoft uses the highly viewpoint-invariant ASIFT algorithm for feature extraction in augmented reality applications. Rotasoft has successfully evaluated the ASIFT implementations on their own dataset. The matching accuracy of the GPU implementation is almost the same as the original CPU implementation. After using the GPSME system, the performance of the application is greatly increased, as shown in Table VI. It appears that the GPSME system accelerates the whole application up to $6 \times$ times for a lower grade system, and up to $13.6 \times$ for a high-performance system.

AnSmart: The application from AnSmart is to use morphological filter to detect the eye's region position in a video. While OpenCV provides some functions to detect the position of eye regions, the performance is limited by a variety of issues, such as the lighting, the head position, other noise, etc. Therefore, AnSmart develops some own morphological filter based algorithms to segment the eye regions from videos. The morphological filter relies on the repeated use of dilation and erosion operations on a binary image. However, the repeated use of dilation and erosion is a quite time-consuming task, particularly for high-resolution image with large sizes of window kernels. The GPSME system is capable of successfully accelerating the performance of mor-

TABLE IV
ACCELERATION PERFORMANCE OF GENERAL APPLICATIONS FROM INDUSTRY

Companies	Applications	Platform	GPSME Speedup	Manual Speedup	Auto-GPU running time
SCS: Centerline extraction	Small 3D Model	A	2.11	4.56	78 ms
		B	2.34	4.68	56 ms
	Big 3D model	A	3.14	24.38	448 ms
		B	3.25	24.68	128 ms
IME: Centerline extraction	Denosing filter kernel region for single image	A	6.17	13.25	8.32 s
		B	4.23	7.23	3.91 s
	Whole program for single image	A	4.29	10.56	19.7 s
		B	3.40	7.12	8.28 s
	Denosing filter kernel region for image sequence	A	6.21	13.54	5 m 25 s
		B	4.35	7.56	2 m 34 s
	Whole program for image sequence	A	4.31	11.23	11 m 47 s
		B	3.78	7.34	4 m 48 s
RotaSoft: Feature extraction in augmented reality	ASFIT algorithm for feature extraction	C	4.76	5.56	14.6 s
		D	8.09	13.6	3.2 s
AnSmart: Eye recognition with morphological filtering	People Eye 1 (1285 × 751)	A	2.91	5.23	847 ms
	People Eye 2 (1279 × 721)	A	3.72	6.65	780 ms
	People Eye 3 (640 × 480)	A	2.68	6.12	458 ms

TABLE V
DOCUMENT ANALYSIS CODE EVALUATION BY GPSME

System A (s)			System B (s)			Details
CPU	GPU	Times	CPU	GPU	Times	
0.26	1.0	0.26	1.3	1.6	0.81	3 × 3 dilation
1.19	1.2	0.92	4.3	1.9	2.22	5 × 5 dilation
3.69	1.2	2.92	18.2	2.3	7.87	9 × 9 dilation
0.24	1.03	0.24	1.5	2.3	0.68	3 × 3 erosion
1.10	1.0	1.07	5.7	2.1	2.73	5 × 5 erosion
3.35	1.1	2.87	16.8	2.5	6.62	9 × 9 erosion

TABLE VI
BLUR MOMENT INVARIANT EVALUATION BY GPSME

Photos Size	CPU	CPU (no OpenCV)	GPU Manual	GPU by GPSME	Speedup Ratio
1000 × 1000	70.2 s	69.7 s	22.56 s	23.44 s	2.99
2000 × 2000	287.1 s	285.5 s	90.6 s	96.16 s	2.98
3000 × 3000	652.1 s	647.4 s	207.5 s	218.0 s	2.987

TABLE VII
PRNU ESTIMATION IN VIDEO SIGNALS EVALUATION

Photos Size	CPU	CPU (No OpenCV)	GPU Manual	GPU by GPSME	Speedup Ratio
1000 × 1000	0.344 s	0.110 s	0.143 s	0.082 s	4.19
2000 × 2000	1.348 s	0.434 s	0.257 s	0.213 s	6.32
3000 × 3000	2.988 s	0.967 s	0.495 s	0.451 s	6.625
4000 × 4000	5.252 s	1.691 s	0.821 s	0.729 s	7.204
5000 × 5000	8.21 s	2.624 s	1.192 s	1.104 s	7.436
6000 × 6000	31.43 s	9.177 s	3.892 s	3.760 s	8.36

phological filters. The results are shown in Table VII. It appears that the GPSME system can effectively speed up the morphological filtering for the AnSmart eye detection algorithm with up to three times. The times of repeating the operations of dilation and

erosion will impact the acceleration performance of the GPSME system. If the times for the dilation and erosion operations are increased, the acceleration performance will be better. Another issue is that the window size of the running kernel could impact the acceleration performance. Due to the image resolution, a 9×9 morphological kernel is used in this case. If the window size increases, the speedup ratio is enhanced significantly. Oppositely, for small window size kernel, the performance of the GPSME system is close to that of the CPU implementation.

To sum up, the acceleration ability of the GPSME system is outstanding. With necessary code revisions on original CPU applications, the GPSME system successfully semiautomatically converts C/C++ code into either CUDA or OpenCL code. On average their applications can be sped up to 3–4× times, even up to over ten times for a high-grade GPU system. Considering that the targeted applications are all real-world programs, the overall acceleration is very good.

B. Applicability

Applicability is another important factor for the GPSME system. We expect the system to be applicable to a wide range of industrial applications. To this end, we have evaluated the GPSME system in a variety of application scenarios.

Document segmentation: Large-scale document digitalization is a popular topic for many libraries and museums in recent years. It involves a significant amount of document layout analysis, region segmentation, and text line segmentation. For large-scale document digitalization, this is a time-consuming task due to the amount of newspapers, magazines, and other documents required to be scanned at high resolution on a daily basis. We have used dilation and erosion algorithms to process some sample newspaper documents images from IMPACT, which is the most successful large-scale document digitalization project in the last ten years. The processed newspaper document images are set to be evaluated by a region segmentation method. The image resolution is 3595×5194 . The GPSME system is

Fig. 6. Performance evaluations of the SWO image filters.

capable of processing C++ code, and the results are shown in Table V.

The evaluation results in Table V show that for dilation or erosion operators with size over 5×5 , the GPSME system can speed up the application performance up to 1–3 times. For dilation or erosion operator on less than 5×5 subwindows, the GPU performance is even slower than the CPU performance. This phenomenon implies that if the dilation or erosion operator is less than 5×5 subwindows, the benefit of GPU acceleration is cancelled out by the introduced overheads (e.g., data transmission between CPU and GPU) and by other commitment introduced on the CPU side, e.g., the extra CPU code employed for the purpose of processing the pragmas, etc.

Sliding window image filter: Sliding window operation is a very popular technique in image processing. Typically, sliding window operation repeatedly applies an image filter to a predefined small-size subwindow that is shifted across a target image. This operation involves high computing complexity if the image filter contains many loops or iterations with high floating-point arithmetic intensity. This particular structure fits very well with the GPU data parallel programming model. The IME users have implemented several statistic measurements for image filter algorithms. Ten typical SWO image operators were selected as benchmarks. A high-resolution image is evaluated by these SWO image operators with different sizes of sliding windows. The size of the evaluated sliding window is respectively given as 3×3 , 5×5 , 7×7 , and 9×9 . The resolution of the evaluated image is 3325×4765 . The baseline is the performance of the original CPU code running on conventional hardware without using multithreads. It compares the speedup ratio of the GPSME-generated CUDA, MINT-generated CUDA, and OpenMP over this baseline. For the simplicity, here, we only demonstrate the speedup ratio of the above ten benchmarks with sliding window 5×5 . The results are shown in Fig. 6.

Blur moment invariants: Blur moment invariants are widely used in digital image processing. They are functional invariant with respect to blur. These blur invariants are employed by IME to identify near-duplicated regions in a digital image. This is carried out in a few main steps:

- 1) tiling the image with overlapping blocks;
- 2) moment blur invariants representation of the overlapped blocks;
- 3) principal component transformation;
- 4) K-d tree representation;
- 5) blocks and neighbors analyses (matching); and
- 6) near-duplication map creation.

The image is tiled by overlapping blocks of $R \times R$ pixels. Blocks slide by one pixel along the image from the upper left corner right and down to the lower right corner. The total number of overlapped blocks for an image of $M \times N$ pixels is $(M - R + 1) \times (N - R + 1)$. For instance, an image with the size of 2000×2000 with blocks of size 16×16 will produce 3,940,225 overlapped blocks. The moment blur invariants representation for each block is computed separately making the run time of the method too expensive. Thus, this is the part that we can accelerate using the GPSME system. The experimental results are shown in Table VI.

PRNU estimation in video signals: PRNU stands for photo response nonuniformity and it is the key information estimated from the video signals enabling us to provide image and video ballistics services. Having a video signal consisting of thousands of frames, PRNU is estimated separately for each frame, this being very computationally expensive. An essential step in estimating PRNU is denoising the image in every JPEG block (compressed block) separately. Moreover, in every block we need to compute the residual of the image and its denoised version. This should be done in thousands of frames for an HD video. For example, a 1280×720 video of 10 min length having 30 frames per second generates 4,320,000 blocks that should be analyzed separately. Thus, the need for GPU acceleration is obvious. The experimental results are shown in Table VII. From Table VII, it appears that the revised CPU application by removing the use of the OpenCV library brings a significant improvement over the original CPU code (three times faster). The machine-generated GPU code can speed up the original CPU application about 6–8 $\times$. The GPSME system is therefore well suited for dealing with this application.

Support vector machine (SVM): In order to further evaluate the applicability of the GPSME system, we have chosen to test it on a different class of application, this time from the field of machine learning. We have decided on an application for handwritten digit recognition, and we chose the SVM as the learning algorithm. Although the accuracy of the SVM is good for a multitude of classification tasks, its execution time tends to be very high, especially for large datasets comprised of large feature sets. We have applied the GPSME system in two key stages of the SVM execution: the generation of the kernel matrices and the actual SVM training. The datasets used were the standard MNIST and the Indian Bangla digit dataset. Both datasets are comprised of around 10 000 training examples, each example being described by a feature space with 784 dimensions. The experimental results clearly outline the effectiveness of the system, being highly close in terms of performance to the highly optimized CUBLAS-based GPU-LibSVM implementation, and faster than the OpenMP and OpenACC implementations. By having a fast GPSME-based implementation, we can run

TABLE VIII
SVM EVALUATION BY GPSME

SVM Implementation	Accuracy [%]	Standard deviation [%]	Time [s]	Dataset/ Feature
OpenMP	97.34	0.45	117.1	Bangla/Pixel features
LibSVM	96.70	n/a	60.5	
GPU-LibSVM	96.70	n/a	10.5	
PGI	97.34	0.45	36.3	
GPSME	97.34	0.45	17.4	
OpenMP	97.65	0.18	136.1	MNIST/Pixel features
LibSVM	97.17	n/a	35.5	
GPU-LibSVM	97.17	n/a	7.8	
PGI	97.65	0.18	43.8	
GPSME	97.65	0.18	17.4	

several simulations for parameter tuning, pushing further also the accuracy results. The results are shown in [Table VIII](#).

C. Usability and Adaptability

The usability of the GPSME system mainly lies in the friendliness of GPSME web interface. The general users have used the GPSME web interface to upload, convert their C/C++ source code, and download the machine-generated CUDA or OpenCL code. The evaluation procedure involves the test of the server functions, user friendliness, efficiency, and accuracy. Most of the essential functions stated in the user requirements have been achieved by providing the server service. This includes the transfer of source codes for analysis, converting CPU source code for GPU processing, running performance diagnostics with the system, validation of converted source codes, and creating reports/logs. In addition, the sample files can be accessed in the web interface of the GPSME system after user logs in; a reminder message for the private key automatically occurs when users log in for their first time; users can add pragma by either keying in or using a dialogue box. The efficiency of the GPSME system is good. The processing time of running the system for each operation is less than 5 s, which is acceptable by all nonexpert GPU users from industry.

The adaptability of the GPSME system indicates how easily and efficiently is for novices to learn how to use the GPSME system. GPU programming requires a steep learning curve for novices.

The GPSME system features a great potential in bringing a cost-effective solution for accessing GPU power. The evaluation of the adaptability involves four parts, including the understanding of loop patterns, algorithmic skeletons, and pragmas and warning messages. In summary, the adaptability of the GPSME system is good. While the understanding of the kernel generation pragmas is still hard to new users, the loop pattern and algorithm skeleton appear to be easy to understand by users. Also, the use of warning messages is well received by users. We also designed a questionnaire to collect feedbacks from nonexpert GPU users after evaluating the GPSME system. The results are shown in [Table IX](#).

D. Competitively

In order to know how the GPSME system behaves compared to other CPU-to-GPU translators, we attempt to use MINT,

TABLE IX
LEARNING AND USING GPSME SYSTEM BY INEXPERIENCED GPU USERS

	IME	B3C	AnSmart	Rotasoft
Understand loop pattern	Easy	Easy	Easy	Easy
Understand basic pragma	Easy	Easy	Easy	Easy
Understand advance pragma	Moderate	Moderate	Moderate	Moderate
Web interface user friendly	Yes	Yes	Yes	Yes
File-editor easy to use	Yes	Yes	Yes	Yes
Running sufficiently fast	Yes	Yes	Yes	Yes
Error and warning reporting	Satisfied	Satisfied	Satisfied	Satisfied
Code protection	Satisfied	Satisfied	Satisfied	Satisfied
Easy to learn	Yes	Yes	Yes	Yes

Bones, Par4All, OpenACC, and OpenMP to evaluate some sample codes. We identify a number of typical directive based source translators and compare their performance in [Table X](#).

OpenACC and PGI are both commercial GPU programming tools with stable applicability but not outstanding acceleration performance in practical applications. CUDA-lite introduces some directives to improve memory hierarchy of CUDA, but it cannot directly support C++.

hiCuda can optimize the CUDA code by dealing with global memory and transformations to leverage the complex memory hierarchy. But it requires users to have some GPU programming experience. Compared to hiCuda, MINT is an easy-use CPU-to-GPU source translator containing only five types of pragmas. It is designed for accelerating stencil computations on the NVIDIA GPU. This translator accepts the input of C source with some intuitive MINT directives, and then generates CUDA C with a speedup performance of up to 10 $\times$.

The following issues have been observed regarding these existing CPU-to-GPU translators.

- 1) Applications written in C++ cannot be processed by most of the above tools. Bones and Par4All do not accept the C++ language as an input source, so they cannot process the given applications. Meanwhile, Bones is an algorithm skeleton based tool with a limited applicability.
- 2) Second, while MINT and OpenMP can be extended to support C++ language, it is indispensable to rewrite the original CPU code as an acceptable input for each tool. The actions of removing the use of external library and breaking up the variable dependencies in the parallelized regions are required.

E. Other Issues

The security requirement aims to protect the source code of the general users. The current GPSME web interface provides a user registration system to access the system. It provides the registered users with private keys to view their source code. In general, the security scheme can satisfy the user requirement to protect their code. One issue that needs some further attention is that the user password and private key are currently stored into the cookie of the browsers unless users delete the cookies. The users can also delete their uploaded files. If they do not delete their files, these files are encrypted to store on server for 30 days. After 30 days, the files will be deleted so users have to upload the files again if they need.

TABLE X
COMPARISON OF PROPERTIES OF OTHER CPU-TO-GPU TOOLS

	hiCUDA	PGI	MINT	CUDA-lite	GPSME
Language support	C-to-CUDA	C/Fortran-to-CUDA	C-to-CUDA	CUDA-to-CUDA	C/C++-to-CUDA/OpenCL
Easy use of directives	Complex	Very complex	Easy	Easy	Easy
Applicability	Good	Outstanding	Limited	Good	Outstanding
Speedup performance	Good	Good	Outstanding	Good	Good
Optimization option	Use of shared memory	No particular one	Shared memory and loop aggregation	Improved memory hierarchy	Improved memory hierarchy (use CUDA Texture)

VII. CONCLUSION AND FUTURE WORK

In this paper, we introduced a web service based CPU-to-GPU source translation system, the GPSME system for general applications. This system enables inexperienced GPU users to take advantage of current GPU capability without having the need for a deep understanding of GPUs. The architecture of this system is inspired by an advanced programming model, MINT, but with some practical extensions and improvements. The functionality of GPSME is more generic, with better flexibility and applicability for improving the productivity of practical applications than conventional automatic CPU-to-GPU programming models with purely research purpose. The experimental results prove that this tool has an improved efficiency and generality in a variety of real-world applications. We show that it enables nonexpert GPU users to flexibly and effectively use automatic CPU-to-GPU code translation. This allows them to gain great speed performance and help the advance of each application domain by allowing for advanced computing models with high complexity. However, the limitation of this GPSME system is that its kernel generation directives are only benefit to the SWO or stencil computing based applications. The future work will consider introducing new directives to solve this problem. Meanwhile, it is expected to be compatible with the existing research tools to optimize the GPU performance of this tool.

REFERENCES

- [1] S. W. Keckler, W. J. Dally, B. Khailany, and M. Garland, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sep. 2011.
- [2] W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar. 2010.
- [3] GPGPU, "General-purpose computation on graphics hardware," Nov. 2013. [Online]. Available: <http://gpgpu.org/>
- [4] OpenCL, "Open computing language (OpenCL)," Nov. 2013. [Online]. Available: <http://www.khronos.org/opencl/>
- [5] CUDA, "NVIDIA, 2007. NVIDIA CUDA programming guide v1.1.," Nov. 2013. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf
- [6] J. Enmyren and C. K. Kessler, "SkePU: A multi-backend skeleton programming library for multi-GPU systems," in *Proc. 4th Int. Workshop High-Level Parallel Program. Appl.*, Baltimore, MD, USA, 2010, pp. 5–14.
- [7] M.M. Baskaran, U. Bondhugula, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, "A compiler framework for optimization of affine loop nests for GPGPUs," in *Proc. Int Conf. Supercomput.*, Island of Kos, Greece, 2008, pp. 225–234.
- [8] A. Leung *et al.*, "A mapping path for multi-GPGPU accelerated computers from a portable high level programming abstraction," in *Proc. 3rd Workshop Gen.-Purpose Comput. Graph. Process. Units*, Pittsburgh, PA, USA, 2010, pp. 51–61.
- [9] U. Bondhugula, U. Hartono, A. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," *ACM SIGPLAN Notices*, vol. 43, no. 6, pp. 101–113, Jun. 2008.
- [10] "PoCC: The polyhedral compiler collection version 1.1," 2012. [Online]. Available: <http://www.cse.ohio-state.edu/pouchet/software/pocc/>
- [11] PLuTo, "A polyhedral automatic parallelizer and locality optimizer for multicores," Dec. 2013. [Online]. Available: <http://pluto-compiler.sourceforge.net>
- [12] A. Leung *et al.*, "A mapping path for multi-GPGPU accelerated computers from a portable high level programming abstraction," in *Proc. 3rd Int. Workshop Gen.-Purpose Comput. Graph. Process. Units*, 2010, pp. 51–61.
- [13] HPC Project, "Par4all automatic parallelization," Oct. 2011. [Online]. Available: <http://www.par4all.org>
- [14] J. Enmyren and C. K. Kessler, "SkePU: A multi-backend skeleton programming library for multi-GPU systems," in *Proc. 4th Int. Workshop High-Level Parallel Program. Appl.*, Baltimore, MD, USA, 2010, pp. 5–14.
- [15] S. Sato and H. Iwasaki, "A skeletal parallel framework with fusion optimizer for GPGPU programming," in *Proceeding of 7th Asian Symposium Programming Languages and Systems* (Lecture Notes on Computer Science, vol. 5904). 2009, pp. 79–94.
- [16] C. Nugteren and H. Corporaal, "Introducing 'bones': A parallelizing source-to-source compiler based on algorithmic skeletons," in *Proc. 5th Annu. Workshop Gen. Purpose Process. Graph. Process. Units*, ACM, New York, NY, USA, 2012, pp. 1–10.
- [17] S. Z. Ueng, M. Lathara, S. S. Baghsorkhi, and W. W. Hwu, "CUDA-lite: Reducing GPU programming complexity," in *Proceeding International Workshop Languages and Compilers for Parallel Computing*. Berlin, Heidelberg: Springer, 2008, pp. 1–15.
- [18] T. Han and T. Abdelrahman, "hiCUDA: High-Level GPGPU programming," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 1, pp. 78–90, Jan. 2011.
- [19] D. Unat, X. Cai, and S. B. Baden, "Mint: Realizing CUDA performance in 3D stencil methods with annotated C," in *Proc. 11th Int. Conf. Supercomput.*, Tucson, AZ, USA, 2011, pp. 214–224.
- [20] S.Y. Lee, S. J. Min, and R. Eigenmann, "OpenMP to GPGPU: A compiler framework for automatic translation and optimization," in *Proc. Int. Conf. Principles Pract. Parallel Program.*, 2009, pp. 101–110.
- [21] The Portland Group, "PGI Fortran and C accelerator programming model," Jun. 2009. [Online]. Available: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.0.pdf
- [22] The OpenACC Standard, "The OpenACC™ application programming interface," Nov. 2011. [Online]. Available: http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf
- [23] L.-N. Pouchet, "PolyBench: The polyhedral benchmark suite," Nov. 2011. [Online]. Available: <http://www.cse.ohio-state.edu/~pouchet/software/polybench/>
- [24] S. Verdoolaege, J. C. Juega, A. Cohen, J. I. Gomez, C. Tenllado, and F. Cathoor, "Polyhedral parallel code generation for CUDA," *ACM Trans. Architecture Code Optim.*, vol. 9, no. 4, Jan. 2013, Art. no. 54.
- [25] HMPP Workbench, "Directive-based multi-language and multi-target hybrid programming model," 2010. [Online]. Available: <http://www.caps-enterprise.com/hmpp.html>
- [26] J. C. Linford, J. Michalakos, M. Vachharajani, and A. Sandu, "Automatic generation of multicore chemical kernels," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 1, pp. 119–131, Jan. 2011.
- [27] J. Kurzak, S. Tomov, and J. Dongarra, "Autotuning GEMM kernels for the Fermi GPU," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 11, pp. 2045–2057, Nov. 2012.

- [28] Y. P. Zhang and F. Mueller, "Autogeneration and autotuning of 3D stencil codes on homogeneous and heterogeneous GPU clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 3, pp. 417–427, Mar. 2013.
- [29] "GPSME: A general toolkit for "GPUtilisation" in SME applications," Oct. 2013. [Online]. Available: www.gp-sme.co.uk
- [30] IME, Image Forgery Detection, Ltd., Nov. 2013. [Online]. Available : <http://www.imagemetry.com/>
- [31] Biocomputing Competence Centre (B3C), Bologna, Italy, Nov. 2013. [Online]. Available: <http://www.b3c.it/>
- [32] RotaSoft, Ltd., Ankara, Turkey, Nov. 2013. [Online]. Available: <http://www.rotasoft.com.tr/>
- [33] AnSmart, Ltd., Wembley, U.K., Nov. 2013. [Online]. Available: <http://www.ansmart.co.uk/>
- [34] ROSE, "ROSE compiler infrastructure," Nov. 2012. [Online]. Available: <http://rosecompiler.org/>
- [35] D. Williams *et al.*, "Evaluation of autoparallelization toolkits for commodity graphics hardware," in *Proceeding of 10th International Conference on Parallel Processing and Applied Mathematics*. Warsaw, Poland: Springer, 2013.
- [36] D. Williams *et al.*, "Accelerating colonic polyp detection using commodity graphics hardware," in *Proc. Int. Conf. Comput. Med. Appl.*, Sousse, Tunisia, 2013, pp. 1–6.
- [37] M. A. Orgun and L. Xue, "From predefined consistency to user-centered emergent consistency in real-time collaborative editing systems," *IEEE Trans. Syst. Man, Cybern. A, Syst. Humans*, vol. 36, no. 6, pp. 1063–1073, Oct. 2006.
- [38] M. C. Dorneich, "A system design framework-driven implementation of a learning collaboratory," *IEEE Trans. Syst. Man, Cybern. A, Syst. Humans*, vol. 32, no. 2, pp. 200–213, Nov. 2002.
- [39] W. P. Brinkman, R. Haakma, and D. G. Bouwhuis, "Component-specific usability testing," *IEEE Trans. Syst. Man, Cybern. A, Syst. Humans*, vol. 38, no. 5, pp. 1143–1155, Aug. 2008.
- [40] B. Liu *et al.*, "Parallel centerline extraction on the GPU," *Comput. Graph.*, vol. 41, pp. 72–83, 2014.
- [41] P. Yang *et al.*, "GSWO: A programming model for GPU-enabled parallelization of sliding window operations in image processing," *Signal Process. Image Commun.*, vol. 47, pp. 332–345, 2016.

Po Yang (M'13) received the B.Sc. degree in computer science from Wuhan University, Wuhan, China, in 2004, the M.Sc. degree in computer science from the University of Bristol, Bristol, U.K., in 2006, and the Ph.D. degree in electronic engineering from the University of Staffordshire, Stoke-on-Trent, U.K., in 2010.

He is currently a Senior Lecturer in the Department of Computing Science, Liverpool John Moores University, Liverpool, U.K. He has a wide range of research interests, including Internet of

Things, RFID and indoor localization, pervasive health, image processing, GPU, and parallel computing. He holds a tracking of high-quality publications and research experiences. He has published more than 50 papers.

Feng Dong received the B.Sc., M.Sc., and Ph.D. (computer science) degrees from Zhejiang University, Hangzhou, China.

He is currently a Professor of visual computing. He joined the Research Centre for Computer Graphics and Visualisation, Brunel University, Middlesex, U.K., in 2007. He became a member of academic staff in the State Key Laboratory of CAD and Computer Graphics, China, a leading computer graphics laboratory. His recent work has also developed new areas in vi-

sual analytics, pattern recognition, image-based rendering, and figure animation. His current research interests include computer graphics, medical visualization, and image processing.

Valeriu Codreanu received the Ph.D. degree in electrical engineering from the Polytechnic University of Bucharest, Bucharest, Romania, in 2011.

He held postdoctoral positions at the University of Groningen (2011–2014) and the Eindhoven University of Technology (2014–2015). He is currently an HPC consultant at SURF-sara, Amsterdam, The Netherlands, the Dutch national supercomputing center. His research interests include theory, architecture, and programming of high-performance and energy-efficient computing systems.

David Williams received the Ph.D. degree from City University, London, U.K., in 2007, for research on virtual endoscopy.

He spent a short period of time doing similar work in the medical visualization industry. He was a Postdoctoral Researcher in the GPSME project, which is developing a general toolkit for "GPUtilisation" in SME applications. The idea is to make the power of GPU hardware more easily accessible to developers by automatically translating C/C++ code into OpenCL or CUDA. This is done using compiler directives similar in principle to OpenMP.

Jos B. T. M. Roerdink (SM'03) received the Ph.D. degree in theoretical physics from the University of Utrecht, Utrecht, The Netherlands, in 1983.

From 1983 to 1985, he held a postdoctoral position at the University of California, San Diego, then he joined the Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, where his work was focused on image processing and tomographic reconstruction. He was an Associate Professor in 1992 and a Full

Professor in 2003 at the Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, Groningen, The Netherlands, where he currently holds a chair in the research group Scientific Visualization and Computer Graphics. He also has a joint appointment with the University Medical Center Groningen.

Baoquan Liu received the Ph.D. degree in computer graphics from the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2007.

He is currently a Senior Research Fellow of computer graphics in the Department of Computer Science and Technology, University of Bedfordshire, Luton, U.K. His research interests are computer graphics, GPU computing, and visualization.

Amjad Anvari-Moghaddam (S'10–M'14) received the Ph.D. degree (Hons.) in energy systems engineering from the University of Tehran, Tehran, Iran, in 2015.

He is currently a Postdoctoral Fellow in the Department of Energy Technology, Aalborg University, Aalborg, Denmark. His research interests include smart microgrids, optimal control and management, and integrated energy systems. He has authored and coauthored 6 book chapters and more than 60 technical papers in energy systems operation and control.

Geyong Min (M'03) received the B.Sc. degree from the Huazhong University of Science and Technology, Wuhan, China, in 1995, and the Ph.D. degree from the University of Glasgow, Glasgow, U.K., in 2003, all in computing science.

He is currently a Professor of high performance computing and networking in the Department of Mathematics and Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, U.K. His current research interests include future Internet, computer networks, wireless communications, multimedia systems, information security, high performance computing, ubiquitous computing, and modeling and performance engineering.