Improving Utility of GPU in Accelerating Industrial Applications with User-centred Automatic Code Translation

***Abstract*—SMEs (Small and medium-sized enterprises), particularly those whose business is focused on developing innovative produces, are limited by a major bottleneck on 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 (Graphics processing units) programming skills, the explosion of GPU power has not been fully utilized in general SME applications by inexperienced users. Also, existing automatic CPU-to-GPU code translators are mainly designed for research purposes with poor user interface design and hard-to-use. Little attentions have 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 GPU capability in accelerating general SME applications. This system designs and implements a directive programming model with new kernel generation scheme and memory management hierarchy to optimize its performance. A web-service based interface is designed for inexperienced users to easily and flexibly invoke the automatic resource translator. Our experiments with non-expert GPU users in 4 SMEs reflect that GPSME system can efficiently accelerate real-world applications with at least 4x and have a better applicability, usability and learnability than existing automatic CPU-to-GPU source translators.**

Index Terms*— Usability, Parallel Computing, GPU, Automatic Translation

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, in resolution, in variety, etc., while the work often present 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, in resolution, in variety etc. demanded by general applications. Parallel computing techniques [1] have gained wide popularity among researchers and developers to overcome these challenges. Many computing tasks exhibit a parallel nature and are hence suitable for parallel computing. The concept of parallel computing is to split large problems into small components and distributing them among multiple processors. Conventional parallel computing takes place using multi-core CPUs or via distributed, grid, high performance computers. The remarkable rise in performance of Graphics Processing Unit (GPU) [2] in recent years 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 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 speed-up can be significant. But the parallelization of 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 relatively short supply to non-expert GPU users. It is highly desirable to have a cost-effective approach that enables inexperienced users to easily utilise GPU technology for accelerating their general applications.

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-word applications is still challenging. Many tools are originally for research purposes with a non-availability 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

Table 1. Comparison of properties of typical automatic parallelization source translation tools

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

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 towards the use of these tools. So there are no existing CPU-to-GPU source translation tools reported in literature to provide an outstanding solution for non-expert GPU users with reasonable acceleration, wide applicability, good usability and well learnability. The motivation of this work is to seek out a solution to satisfy the above requirements.

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 semi-automatic 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 non-expert users from 4 SMEs reflect that GPSME system can efficiently accelerate general real-world applications with at least 4x; and also have an improved applicability, usability and learnability than existing CPU-to-GPU source translation tools. The main contributions of this paper are below:

- A comprehensive requirement analysis of inexperienced users for utilizing GPU technology in general SMEs applications is given. It is benefit to improve the accessibility and the applicability of existing automatic CPU-to-GPU source translators in real-world applications.
- 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.
- A thorough performance evaluation of 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 have 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 2 reviews notable automatic CPU-to-GPU source translators. Section 3 analyses the general requirement of inexperienced GPU users. Section 4 presents the design and implementation of GPSME system. Section 5 shows the experimental validation results. Section 6 gives a conclusion and future work.

II. RELATED WORK

A large amount of research has been dedicated to automatic converting CPU code to GPU code. This section reviews existing typical automatic parallelization source translators regarding acceleration, applicability, usability and adaptability. 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 dependency 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 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.
Algorithmic skeleton and polyhedral model based tools both 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 the purpose of allowing automatic CPU-to-GPU translators to be more applicable, directive-based source translators [17-23] became popular. By using these tools for generating 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

Table.2. Detailed information of inexperienced GPU users from industry

	Langauge	*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	Imge processing speed in real time AR books.
AnSmart	C++	Eye Tracking	Medical image analysis in diagnosising eye diseases.

CUDA by directly inserting the directives into the CUDA code. hiCUDA [18] provides a set of 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 (since PGI is designed as a compiler instead of a source-to-source translator). Besides, MINT [19] is a very easy-to-use C-to-CUDA source translator containing only five types of pragmas. It is designed for speedup stencil computations on NVIDIA GPUs only. 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 10x. 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. However, 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 analyses 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.2 illustrates the applicable area of each company and the problem they face. For collecting a general requirement of their non-expert 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:

- No need on having an in-depth understanding of GPUs
- Full or semi-automatic CPU-to-GPU source translation
- Support C++ programming language
- Support either CUDA or OpenCL
- Efficient speedup performance and no accuracy loss
- Source code protection
- Report the system process and error diagnostics

Regarding the above general objectives, it appears that existing CPU-to-GPU source translators in Table.1 hardly satisfies the full needs.

Non-expert GPU users expect a system that enables them to quickly take advantage of current GPU capability to effectively and economically speed up their products. In terms of this goal, an explicit requirement analysis of their expectations on this system is given as below:

- **Acceleration:** They expect their general CPU applications to be accelerated significantly on moderate hardware platforms. Non-expert GPU users are more interested in actual time saved in their applications instead of a high speed-up ratio of GPU over CPU. However, existing CPU-to-GPU translators focus more on the improvement of their speed-up ratio for reflecting their parallel 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 in this paper need be evaluated by practical applications, and not only the sample code for the parallel region.
- **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 understanding and learning of these pragmas become hard tasks for non-expert users. There has to be a trade-off of these tools between applicability and directive complexity. The directives of GPSME system have to be simple but enable supporting all types of algorithms skeleton and loop patterns from their general applications.
- **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 interface are C-to-CUDA based command line tools under Linux. A system with better usability for non-expert GPU users is expected.

- **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

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 a web-service. Though the web-interface, inexperienced GPU users can upload their CPU source files; execute 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 in Figure.1.

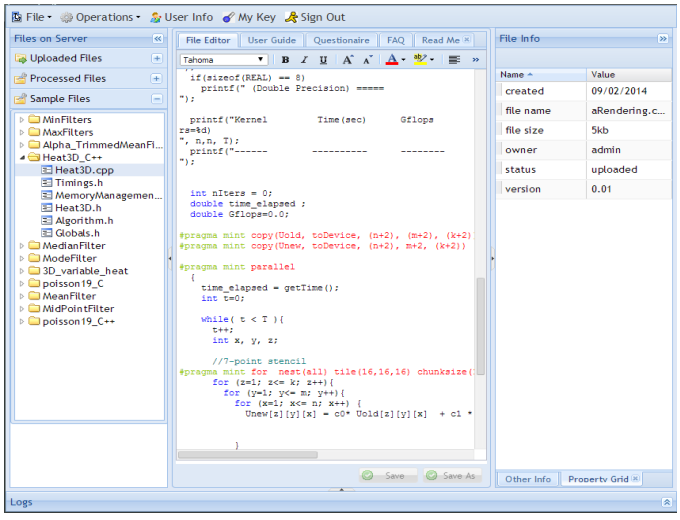

Fig. 1. GPSME web interface

The typical usage of the GPSME web application is as follows:
• The user creates an account with his/her details.
• The user uploads C/C++ source file and necessary header files.
• The user selects the desired output type and initiates the code translation process.
• 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 this remote web server.
• If CPU source code has an external dependency then it must be presented or installed on the remote webserver (in addition to user's local machine). Users need to contact the server administrator to get the dependencies installed if so required.
• When uploading a C++ file for parallelisation 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 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 Figure.2.

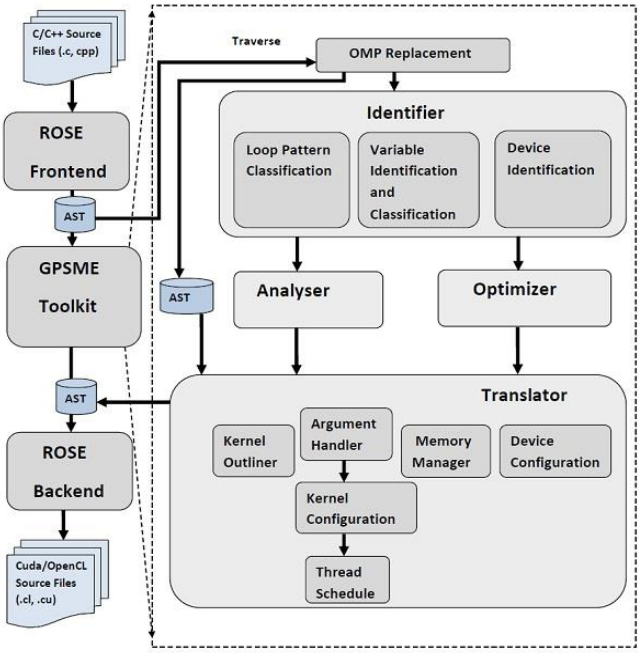

Fig. 2. Structure of GPSME core library

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 *Analyser* and the *Optimizer* investigate the possibility of using predefined approaches in GPSME system for

Table 3. 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 "**for**" 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 2D data
	CopyMalloc1DArray		To create a CUDA array on a device, associating it with a CUDA texture on the device
	CopyMemcopy2D		To create a CUDA *cudamemcpy2D* function to copy a matrix between CPU and GPU memory
	CopyMemcopy2DToArray		To create a CUDA function *cudaMemcpy2DToArray* 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 parapllel "**for**" loop with given thread blocks and threads.
	Initialisation		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 "**For**" loop
	Remain		To transform the operations on a sliding window from CPU algorithm to the GPU kernel.
	Assign		To assign the new data to the relevant GPU buffer with the correct index.

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 CUDA code using the GPSME core library.

C. Interaction

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

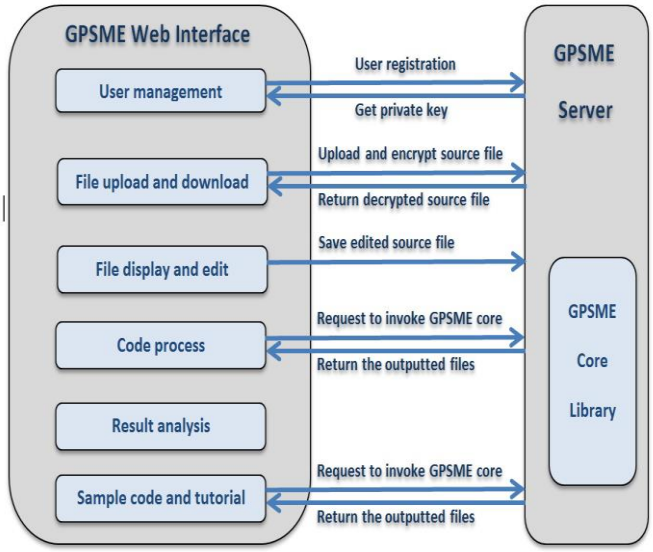

Fig. 3. Interaction between GPSME web interface and core library

In Figure.3, users firstly register and upload their source files though 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 key in 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 three main tasks: a) Identification of parallel region. b) Memory management. c) 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 to three challenges below:

- 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.
- The *copy* directives combining memory allocation and data transfer is too extensive to support high level storage setup and management. Particularly, in practical applications, it requires separating the operations of memory allocation and data transfer to allow the reuse of the allocated memory for data transfer.
- The MINT kernel generation directives only support stencil computing. Many algorithm skeletons in general applications are beyond stencil computing, which hardly copy with by MINT kernel.

For overcoming the limits to MINT directives, the design of GPSME directives consider the practical requirements in section 2. The detailed information of GPSME directives are reported in our early work on GSWO programming model [37]. First, the basic GPSME directives are inherited from MINT as shown in Table.3. 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 non-expert 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 multi-file 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], 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 auto-parallelization process. MINT kernel was still able to process the outermost loop but this yielded significantly lower performance that that which was theoretically obtainable.

An example of such a problematic triangular loop is shown in Table.4, and the iteration space is depicted visually in Figure.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 non-compilable output which attempted to make use of variables prior to their declaration.

GPSME have therefore implemented 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.

Table.4. Example of problematic triangular loop

Algorithm

```
1: #pragma mint copy(data,toDevice, M, N)
2:  #pragma mint copy(mean,toDevice, M)
3:  #pragma mint copy(symmat,toDevice, M, N)
4:  #pragma mint parallel
5: {
6:         ... //Some code omitted for brevity
7:     #pragma mint 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 mint 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:

- **Full:** All threads are part of the triangular iteration space and must be executed. No processing capability is wasted in this scenario.
- **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.
- **Half-full:** In this case the running time of the thread block is determined by the threads which do need to run. Threads which do not need to run must still wait upon those that do, and this represents some wasted processing capability.

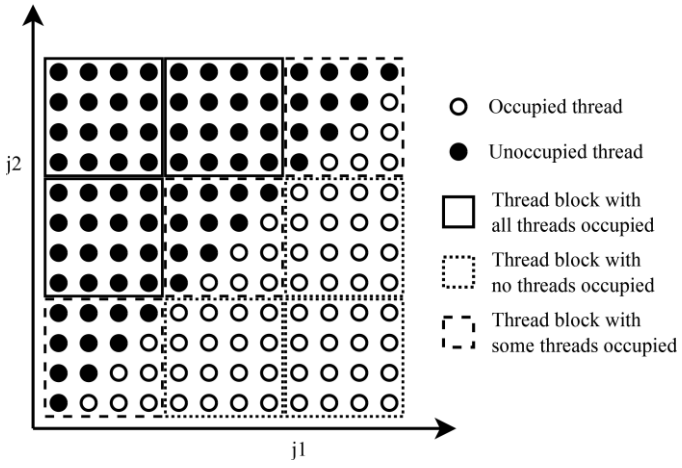

Fig. 4. Iteration space of the two-level covariance loop

This approach has yielded a performance increase greater than that which was obtained from OpenACC, and more than 30 times greater than that which 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.

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 five minutes to fully process a CT dataset with a resolution of 512x512x512 voxels. This code was optimised prior to GPUification such that it ran in 48 seconds on a single core. The GPSME system was then able to achieve a three-fold speed increase bringing the execution time down to 17 seconds, which was comparable to the 15 seconds taken by the *four-core* CPU version. A manual implementation was able to further decrease the runtime to only 1.2 seconds [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. However, this manual implementation did take several days of work, whereas the auto-parallized version only took one hour to add the directives and perform some minor debugging.

B. Single-dimensional vs Multi-dimensional Arrays

Another improvement to the GPSME system is that it adds more optimization opportunities when applied to code that use multi-dimensional 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 [26]. For the 2MM and SYR2K tests, a further 25% performance increase is obtained when using two-dimensional addressing instead of the default flattened array addressing.

The changes from single-dimensional to multi-dimensional 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 multi-dimensional ones. An interesting observation is that when faced with the same two-dimensional 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 MINT kernel generation, another kernel generation scheme in GPSME system is designed into our early work (GSWO model [37]) to orchestrate the GPU kernel code generation. Figure.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.

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

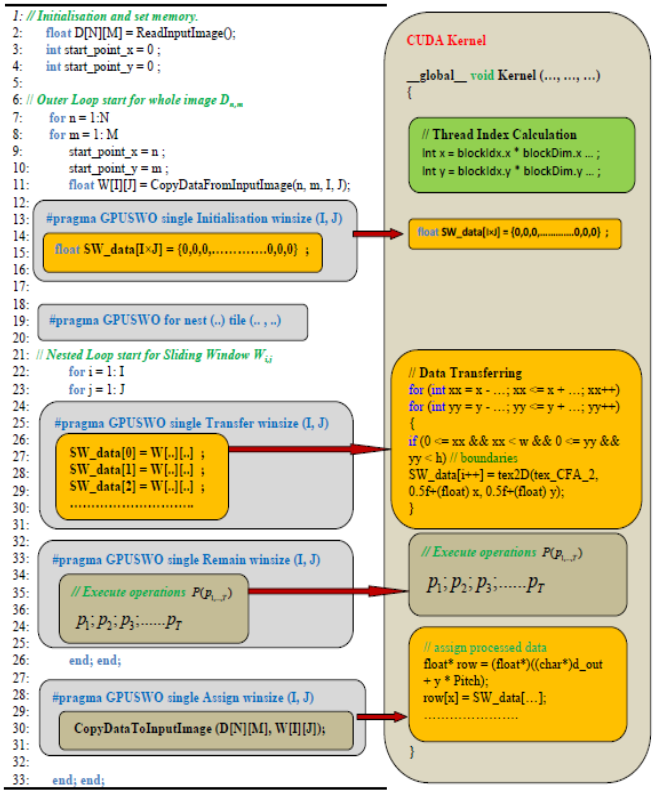

Figure. 5. Working flow of Kernel Generation Pragmas

VI. PERFORMANCE EVALUATION

In this section, the effectiveness of 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 multi-threading. The evaluation platforms were: (a) Intel Core i7-2670QM CPU and NVIDIA GeForce GT 540M; (b) Intel Core i7-3770K CPU and NVIDIA GeForce GTX 690; (c) Intel Core i3-2.1GHz CPU and NVIDIA GeForce GT 520M; (d) Intel Core i7-3.4GHz 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. 4.

SCS: The application from SCS is centerline extraction for a given 3D 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 shown in Table.4. 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 de-noising methods to a set of images that are known to come from a given camera. The sample code from IME are based on C++, and aims at implementing a 3×3 median filter for de-noising. The image resolution is 3648 × 2736. The number of images is 39. The algorithm splits the images into a number Region of Interests (ROIs), which can be processed in parallel. Table 5 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. Secondly, 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 6 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 GPSME system, the performance of the application is greatly increased, as shown in Table. 6. It appears that GPSME system accelerates the whole application up to 6x times for a lower grade system, and up to 13.6x 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 morphological filters. The results are as shown in Table.7. It appears that the GPSME system can effectively speed up the morphological filtering for the AnSmart eye detection algorithm with up to 3 times. The times of repeating the operations of dilation and erosion will impact the acceleration performance of 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 GPSME system is close to that of the CPU implementation.

Table 4. Acceleration performance of general applications from industry

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

To sum up, the acceleration ability of the GPSME system is outstanding. With necessary code revisions on original CPU applications, GPSME system successfully semi-automatically converts C/C++ code into either CUDA or OpenCL code. On average their applications can be sped up to 3-4x times, even up to over 10 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 digitalisation 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 digitalisation, 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 digitalisation project in the last 10 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 capable of processing C++ code, and the results are shown in Table 5.

Table 5. Document Analysis code evaluation by GPSME

System A (seconds)			System B (seconds)			
CPU	**GPU**	**Times**	**CPU**	**GPU**	**Times**	**Details**
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

The evaluation results in Table 5 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 sub-windows, 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 sub-windows, 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 sub-window 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 date 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 by using different size of sliding windows. The size of the evaluated sliding window is respectively given as 3×3, 5×5, 7×7, 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 multi-threads. 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 Figure 7.

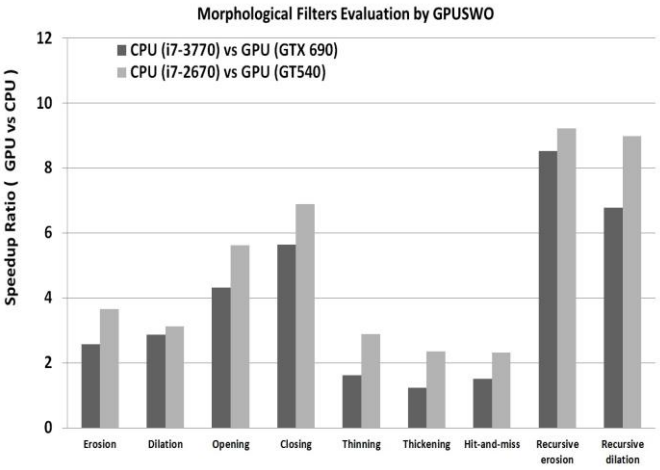

Figure. 6. Performance evaluations of the SWO image filters

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 neighbours analyses (matching), 6. Near-duplication map creation. The image is tiled by overlapping blocks of R × 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 × N pixels is (M – R + 1) × (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 6.

Table 6. Blur moment invariant evaluation by GPSME

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

PRNU estimation in video signals: PRNU stands for photo response nonuniformity (PRNU) 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 de-noising the image in every JPEG block (compressed block) separately. Moreover, in every block we need to compute the residual of the image and its de-noised version. This should be done in thousands of frames for an HD video. For example, a 1280×720 video of 10 minute length having 30 frames per second generates 4.320.000 blocks that should be analysed separately. Thus, the need for GPU acceleration is obvious. The experimental results are shown in Table 7. From Table 7, 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 6-8x. The GPSME system is therefore well suited for dealing with this application.

Table 7. PRNU estimation in video signals evaluation

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

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 support vector machine (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 was the standard MNIST and the Indian Bangla digit dataset. Both datasets are comprised of around 10000 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 several simulations for parameter tuning, pushing further also the accuracy results. The results are shown in Table.8.

Table 8. 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	

C. Usability and Adaptability

The usability of 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 seconds, which is acceptable by all non-expert 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, 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 non-expert GPU users after evaluating the GPSME system. The results are shown in Table. 9

Table 9. 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

Table 10. Comparision of properties of other CPU-to-GPU tools

	hiCUDA	*PGI*	*MINT*	*CUDA-lite*	*GPSME*
Language support	C-to-CUDA	C/Fortan-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
Optimisation option	Use of shared memory	No particular one	Shared memory and loop aggregation	Improved memory hierarchy	Improved memory hierarachy (use CUDA Texture)

D. Competitiveness

In order to know how the GPSME system behaves compared to other CPU-to-GPU translators, we attempt to use MINT, 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 10.

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 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 10x.

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

• 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.

• Secondly, 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.

In the GPSME system, we use our existing GSWO model [41] to determine the size of block and thread. The selection of block and thread size here is based on the pragmas: nest, tile and chunksize. They are used for indicating the depth of for-loop parallelization within a loop nest, specifying how the iteration space of a loop nest is to be subdivided into tiles, and aggregating logical threads into a single CUDA thread, respectively. The size of a CUDA thread block in the GPSME project is the same as in MINT: threads (tx/cx, ty/cy,tz/cz). But the impact of selected block and thread size on acceleration in GSWO model is not as significant as that in MINT. The kernel generator in MINT makes all of the parameters in the function argument become kernel call parameters and makes all memory references through device memory.

There are a few minor limitations on memory use in GPSME project. In the GSWO model, the memory management pragmas are not simple for a non-expert to understand and use correctly, though they can be successful with a little care. Finally, no optimizations of the CUDA kernels in the GSWO model are considered in this GPSME. Hence, traditional optimization methods that use shared memory or improve memory bandwidth cannot be used directly. We will investigate using shared memory to improve kernel acceleration in future work.

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 non-expert 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, Issue 5, pp7-17, Sep, 2011.
2. W.J. Dally, "The GPU Computing Era". *IEEE Micro.* Vol 30, Issue 2, pp56-69, Mar, 2010.
3. GPGPU. (Nov. 2013), "General-Purpose Computation on Graphics Hardware." Available [Online]: http://gpgpu.org/.
4. OpenCL. (Nov. 2013), "Open Computing Language (OpenCL)." Available [Online]: http://www.khronos.org/opencl/.
5. CUDA. (Nov. 2013), "NVIDIA, 2007. NVIDIA CUDA Programming Guide v1.1." Available [Online]: 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 on High-Level Parallel Programming and Applications* (HLPP-2010), Baltimore, Maryland, USA. ACM, pp. 5-14, 2010.
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", *Proc. Int'l Conf. Supercomputing*, NewYork, USA. ACM, pp. 225-234, 2008.
8. A. Leung, N. Vasilache, B. Meister, M. Baskaran, D. Wohlford, C. Bastoul, and R. Lethin, "A mapping path for multi-gpgpu accelerated computers from a portable high level programming abstraction", *in Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units* (GPGPU '10), New York, NY, USA, ACM, pp. 51–61, 2010.
9. U. Bondhugula, U. Hartono, A. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer", *ACM SIGPLAN Not. Vol 43, Issue* 6, pp.101–113, June, 2008.
10. PoCC 2012. PoCC: the polyhedral compiler collection version 1.1. http://www.cse.ohio-state.edu/ pouchet/-software/pocc/
11. Pluto, (Dec, 2013), "A polyhedral automatic parallelizer and locality optimizer for multicores", Available [Online]: http://pluto-compiler.sourceforge.net
12. A. Leung, N. Vasilache, B. Meister, M. Baskaran, D. Wohlford, C. Bastoul, and R. Lethin, "A mapping path for multi-GPGPU accelerated computers from a portable high level programming abstraction," *In Proc. 3rd International Workshop on General-Purpose Computation on Graphics Processing Units*, pp. 51-61, 2010.
13. HPC Project, (Oct, 2011), "Par4all automatic parallelization," Available [Online]: 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 on High-Level Parallel Programming and Applications* (HLPP-2010), Baltimore, Maryland, USA. ACM, pp. 5-14, 2010.
15. S. Sato and H. Iwasaki, "A skeletal parallel framework with fusion optimizer for GPGPU programming", *Programming Languages and Systems, Lecture Notes on Computer Science*, vol 5904, pp 79-94, 2009.
16. C. Nugteren and H. Corporaal. "Introducing 'Bones': A Parallelizing Source-to-Source Compiler Based on Algorithmic Skeletons." In GPGPU-5: 5th Workshop on General Purpose Processing on Graphics Processing Units. ACM, 2012.
17. S.Z., Ueng, M. Lathara, S.S. Baghsorkhi, and W. W. Hwu, "CUDA-lite: Reducing GPU Programming Complexity ", *Proc. Int'l Workshop Languages and Compilers for Parallel Computing*, Berlin, Heidelberg. Springer, pp. 1-15. 2008.
18. T. Han and T. Abdelrahman, "hiCUDA: High-Level GPGPU Programming", *IEEE Trans. Parallel and Distributed Systems*, 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 ICS '11: International Conference on Supercomputing*, New York, NY, USA, ACM, pp. 214-224, 2011.
20. S.Y. Lee, S. J. Min, and R. Eigenmann, "OpenMP to GPGPU: A compiler framework for automatic translation and optimization", *PPoPP 2009, International Conference on Principles and Practice of Parallel Programming*, pp. 101-110, 2009.
21. The Portland Group, (June. 2009), "PGI Fortran and C Accelerator Programming Model ", Available [Online]: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.0.pdf
22. The OpenACC Standard, "The OpenACC™ Application Programming Interface", Available [Online]: http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf, November 2011.
23. L.-N. Pouchet. (Nov. 2011), "PolyBench: The Polyhedral Benchmark Suite." Available [Online]: http://www.cse.ohio-state.edu/~pouchet/software/polybench/
24. S. Verdoolaege, J. C. Juega, A. Cohen, J. I. Gomez, C. Tenllado, and F. Catthoor, "Polyhedral parallel code generation for CUDA", ACM Transactions on Architecture and Code Optimization, vol.9, issue.4, Jan, 2013.
25. HMPP 2010. "HMPP workbench: directive-based multi-language and multi-target hybrid programming model." Available [Online]: http://www.caps-entreprise.com/hmpp.html
26. J. C. Linford, J. Michalakes, M. Vachharajani, and A. Sandu, "Automatic Generation of Multicore Chemical Kernels", *IEEE Trans. Parallel and Distributed Systems*, 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 and Distributed Systems*, 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 and Distributed Systems*, vol 24, no.3, pp. 417-427, Mar, 2013.
29. GPSME. (Oct, 2013), "A General Toolkit for "GPUtilisation" in SME Applications", Available [Online]: www.gp-sme.co.uk
30. IME. (Nov. 2013), Image Forgery Detection, Ltd. Available [Online]: http://www.imagemetry.com/
31. B3C. (Nov. 2013), Biocomputing Competence Centre. Available [Online]: http://www.b3c.it/
32. RotaSoft. (Nov. 2013), RotaSoft, Ltd. Available [Online]: http://www.rotasoft.com.tr/
33. AnSmart. (Nov. 2013), AnSmart, Ltd. Available [Online]: http://www.ansmart.co.uk/
34. ROSE. (Nov, 2012),"ROSE compiler infrastructure" Available [Online]: http://rosecompiler.org/
35. D.Williams, V.Codreanu, P.Yang, B.Q.Liu, F. Dong, B. Yasar, B. Mahdian, A. Chiarini, X. Zhao, and J. B.T.M. Roerdink. "*Evaluation of autoparallelization toolkits for commodity graphics hardware*", In 10th International Conference on Parallel Processing and Applied Mathematics. Warsaw, Poland. Springer, 2013.
36. D. Williams, V. Codreanu, J. B.T.M. Roerdink, P. Yang, B.Q. Liu, F. Dong, and A. Chiarini. "*Accelerating Colonic Polyp Detection Using Commodity Graphics Hardware*", In Proceedings of the International Conference on Computer Medical Applications. Sousse, Tunisia, pages 1–6, 2013.
37. M. A. Orgun, and L. Xue, "From Predefined Consistency to User-Centered Emergent Consistency in Real-time Collaborative Editing Systems", *IEEE Trans. Systems, Man, and Cybernetics: Part A: System and 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. Systems, Man, and Cybernetics: Part A: System and 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. Systems, Man, and Cybernetics: Part A: System and Humans*, vol 38, no.5, pp.1143-1155, August, 2008.
40. B. Liu, A.C Telea, J. BTM. Roerdink, G. J. Clapworthy, D. Williams, P. Yang, F. Dong, V. Codreanu, and A. Chiarini. 2014. ”*Parallel centerline extraction on the GPU”*, Computers & Graphics, 41 :72-83
41. P. Yang, G. Clapworthy, F. Dong, V. Codreanu, D. Williams, B. Liu, J. BTM. Roerdink, and Z. Deng. 2016. “GSWO: A programming model for GPU-enabled parallelization of sliding window operations in image processing,” *SIGNAL PROCESSING-IMAGE COMMUNICATION*, 47: :332-345