Debugger for Multi-level Hybrid Parallel Programs on Heterogeneous Accelerator Cluster Architectures – Survey and Challenges

Shamjith K V, Mangala N, Prahlada Rao BB, Sarat Chandra Babu N
Hybrid Computing Group, Centre for Development of Advanced Computing, Bangalore, India
{shamjithkv, mangala, prahladab, sarat}@cdac.in

Abstract – **The need to debug hybrid parallel programs on heterogeneous accelerator clusters opens a new set of challenges for concurrently managing the processes and threads at node and accelerator levels. Currently, there exist open source debuggers for traditional HPC clusters, which support debugging of multi-node parallel programs. At present, debugging at the accelerator level is handled through language-specific debuggers provided by the device manufacturers. It is desired that a standards based debugger for hybrid parallel programs should support multiple-languages, multiple devices, ease of use, scalability, and portability. However, currently there is no open-source standardized debugger for hybrid parallel programs on heterogeneous multi-accelerator clusters. The authors in this paper survey the existing debugger solutions for multi-level hybrid parallel programs. The authors in this paper bring out the challenges involved in developing a standardized open source debugger based on GDB, for heterogeneous accelerator clusters and present the features of a debugger on heterogeneous multi- accelerator clusters.**

Keywords – high performance debugger; heterogeneous accelerator cluster; multi-level debugging; hybrid parallel program; GDB; LLDB; HPC; GPU; MPI; OpenMP; OpenCL; CUDA

I. INTRODUCTION

The petascale computing systems[1], like Tianhe-2[2] of National University of Defense Technology, China, Titan[3] of Oak Ridge National Laboratory, USA and Sequoia[4] of Lawrence Livermore National Laboratory, USA, contain general purpose graphics processing units (GPGPU) which offer vast SIMD parallelism. Petascale applications [5] spawn thousands of processes and threads on these hybrid systems to solve large scientific problems. The distribution of computation and data among the processes and threads, along with the communication among them, makes it challenging to design and develop High Performance Computing (HPC) applications. Ensuring the correctness of the HPC application is very important to deliver proper and efficient solution to scientific problems. High performance debugging [6, 7] is always an active area among the HPC researchers, due to the important role of debugger software in assisting HPC application developers. Hence, it is very important to have an open source debugger for the benefit of the academic and research communities.

The hybrid HPC applications are written using multiple paradigms such as MPI, OpenMP and OpenCL, and execute on the multicore CPUs as well as the accelerator cores in the heterogeneous accelerator cluster. A debugger for such applications should be capable of handling the coordinated execution of huge number of processes and threads across CPUs and accelerators.

The authors in this paper present their research findings and challenges, in an attempt to develop an open source debugger for heterogeneous architectures [8, 9] under the Hybrid Computing project [10]. The authors also discuss the relevance and approach for extending the popular sequential debuggers, GDB and LLDB, as hybrid program debuggers.

The paper is organized as follows: In Section II, the authors present the procedure of debugging hybrid multi-level parallel programs on heterogeneous accelerator cluster; Section III presents the survey of existing High Performance Debugging solutions. Section IV presents the status of high performance debugging. In Section V the authors discuss about the open source debugger for heterogeneous accelerator clusters. The Section VI describes about extending GDB to accelerator level debugging. The challenges of using GDB for OpenCL debugging are captured in VII. In Section VII, workarounds for multi-level hybrid parallel program debugging are explained; followed by conclusion in Section IX.

II. DEBUGGING MULTI-LEVEL HYBRID PROGRAMS

A. Multi-level Hybrid Parallel Programming

A hybrid parallel program [11, 12] shall have a hierarchical multi-granularity parallelism at - i) node/core level and ii) accelerator level. Fig. 1 depicts a typical multi-level hybrid parallel program running on different nodes and in turn on different compute elements in the node. It contains multiple programing paradigms which execute on heterogeneous compute elements. Hence, the hybrid parallel programs have several hundreds of threads executing simultaneously on multiple cores; and each thread will be working with a different variable. In such a program, the code that runs on CPUs is termed as Host Code, and the one that runs on accelerators is termed as Device Code. Accelerators require target specific program paradigms such as OpenCL/ CUDA [13,14] to execute the parallel section (referred to as *kernel*) on the devices. Shared memory paradigms like OpenMP[15] facilitate the utilization of processor cores on a single node, and usually Message Passing Interface (MPI) [16] plays a key role in coordinating the computation and communication among

978-1-4673-9286-0/15/$31.00 ©2015 IEEE

the concurrent execution of multiple processes within and across multiple computing nodes in a HPC cluster.

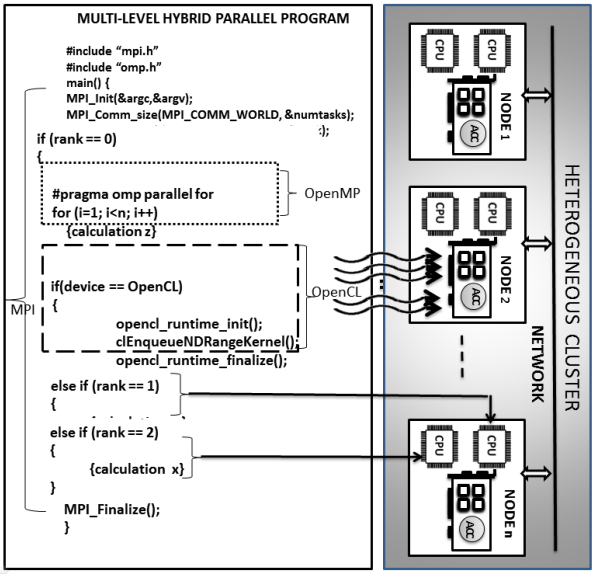

Figure 1. Multi-level Hybrid Parallel Program Mapping onto Heterogeneous Accelerator Cluster

The GPU programming follows a push-pull model, where the data meant for *kernel* execution will be pushed to the memory accessible by the accelerators for feeding to the computing cores inside the accelerator device. And, upon completion of *kernel* computation the output data is pulled from the GPU device memory to the CPU host memory area where the concurrent processes have access. These accelerator specific data transfers are carried out by the Application Programming Interfaces (APIs) exposed by the accelerator device vendors (such as NVIDIA, AMD), or the programming paradigms supported on specific GPU devices.

B. *Challenges of Debugging Multi-level Hybrid Parallel Programs on Heterogeneous Accelerator Cluster*

Debugging a multi-level hybrid parallel application is a challenging task as the conventional debugging operations used in sequential applications should also be available to each parallel thread of execution [17]; and the most effective way is to have debugger software that allows the programmer to debug at the scale of the thread [18].

In order to give a better understanding of a debugger for heterogeneous accelerator cluster, the authors start by describing traditional cluster program debugger and then extend the discussion to debugging multi-level hybrid parallel program on a heterogeneous accelerator cluster.

The typical approach for debugging traditional CPU-only cluster parallel programs, that have many processes executing in parallel, is to have a separate debugger instance attach to each processes that needs to be debugged [19]. A separate debug agent will coordinate the debugging instructions from a unified interface to the user(s) and the debugging instances.

In the Fig. 2, P1 to P4 are the processes participating in the parallel program, on a computing cluster. If the user wishes to debug two processes P1 and P2, then separate instances of the debugger (G1 and G2) are attached to them, and the debug agent co-ordinates the commands instructed by the user through the debugging command server. This is the conventional approach for a multi-process parallel application debugging on a cluster [20].

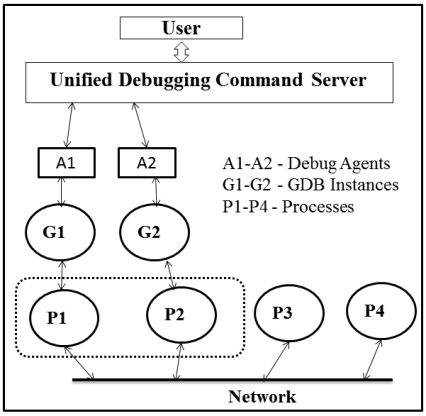

Figure 2. An Abstract View of Various Components in Debugging Concurrent Processes in a CPU-only cluster

For a hybrid program on accelerator cluster, the debugger should be extended further to debug the threads which execute on the accelerators. Fig. 3 shows the processes and threads executing on a heterogeneous multi-accelerator cluster. Once the *kernel* function starts to execute in the GPU cores, the debugger should be able to attach to a desired thread (of execution) and step through the instructions of the *kernel* function.

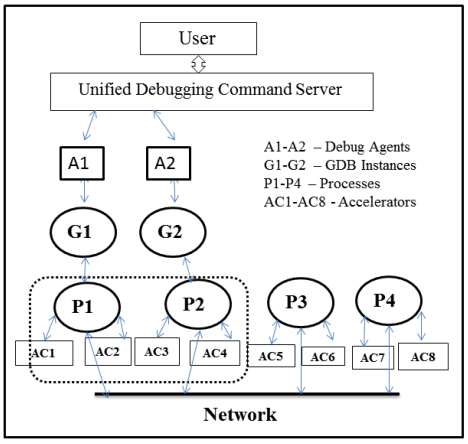

Figure 3. An Abstract View of Debugging Multi-level Hybrid Parallel Program on a Heterogeneous Accelerator Cluster

978-1-4673-9286-0/15/$31.00 ©2015 IEEE

III. SURVEY OF DEBUGGERS

While there have been significant efforts in developing debuggers for cluster level massively parallel programs [21, 22] and while most of the accelerator card manufacturers are providing debuggers to debug GPGPUs [23,24], there are no open source debuggers for a multi-level hybrid parallel program on a heterogeneous accelerator cluster. Table I gives a detailed comparison of the features of various debuggers. It can be inferred from Table I, that there are no single debugger to debug hybrid parallel programs on a heterogeneous multi-accelerator cluster.

TABLE I. COMPARISON OF DEBUGGERS FOR CLUSTERS

Tool	Debugging support for CPU	Debugging support for Accelerator	Extendibility	Company
GDB [23]	Host code, OpenMP, MPI	Does not support either CUDA or OpenCL	Can be extended to new devices	Open Source – GNU Project
DDT [24]	Host code, OpenMP, MPI	Supports CUDA but not OpenCL	The base debugger being GDB, DDT can be extended to architectures supported by GDB	Allinea
TotalView [25]	Host code, OpenMP, MPI	Supports CUDA but not OpenCL	The backend debugger is proprietary	Rogue Wave
CUDA-GDB [28]	Host code, OpenMP, MPI	Supports CUDA but not OpenCL	Extended from GDB	Open Source maintained by NVDIA
gDebugger [30]	Host code	Supports OpenCL No CUDA support	Proprietary of Graphics Remedy	Graphics Remedy (AMD)
CodeXL [31]	Only Host Code	Supports OpenCL only on AMD No CUDA support	Proprietary of AMD	AMD
PGDBG [32]	Host Code, OpenMP, MPI, Support CUDA-x86	Does not support either CUDA or OpenCL	Proprietary of PGI	PGI
NSIGHT [33]	Host code, OpenMP, MPI	Supports CUDA but not OpenCL	Proprietary of NVIDIA	NVIDIA
DIViA [34]	Host Code, OpenMP, MPI	Does not support either CUDA or OpenCL	The backend being GDB, it can be extended to architectures supported by GDB	C-DAC

IV. DEBUGGING STANDARDS

A. High Performance Debugging Forum (HPDF) Recommendations

A lot of work has been done in 1990s and early 2000s in the areas of high performance debugging and its standards. In 1998, High Performance Debugging Forum constituted by open source community experts brought the standards [6] for Higher Performance Debugger (HPD). But, there has been no effort in the last decade to address the standardisation of high performance debugging involving accelerators.

The goals of HPD standards were to ensure, parallel debuggers satisfy the basic requirements of persons who develop applications for HPC systems, parallel debuggers be usable by those application developers, in the sense of easy to learn and easy to use, and parallel debuggers be consistent across platforms, so that users of one standard-conforming debugger can switch to another with little or no effort. The HPD standard addresses programs which are typically in parallel, which are written in one or more high level languages (FORTRAN, C and C++) and the programs which are meant for multiple nodes and architectures. This standard caters to both shared and distributed memory programming approaches. The debugging forum published their efforts on 'High Performance Debugging Standards Effort', capturing the standards guidelines for implementing High Performance Debuggers [6].

The HPD Forum sponsored by the Parallel Tools Consortium established three general goals for parallel and distributed debuggers. The HPDF recommendations are shown in Box 1.

- Satisfy basic debugging requirements of high performance computing application developers;
- Be usable – in the sense of easy to learn and easy to use – by these application developers;
- Be consistent across any platforms, so that users of one standard-conforming debugger can switch to another with little or no effort.

Box 1. HPDF† Recommendations

B. Common Debugging Information Format

To allow the source level debugging of hybrid program applications on heterogeneous clusters, the debugging information must be embedded in the binaries. This allows the low level program debugger like, GDB (GNU Debugger) or LLDB (LLVM Debugger), to interpret and decipher the program information. The information embedded must be conforming to the open standards, so as the commonly available binary utilities can carry out the symbol interpretations. DWARF[36] debugging symbol

† High Performance Debugging Forum

978-1-4673-9286-0/15/$31.00 ©2015 IEEE

standard is the most widely used and preferred debugging information format. Conforming the debugging information of hybrid programs, especially the program codes meant for heterogeneous accelerators, to DWARF debugging symbol would help the easy source level debugging of hybrid applications. Several researchers are working to address each of the goals with respect to multi-level hybrid parallel program debugging [24, 37, 38]. Our efforts are towards enhancing the widely accepted open source debugger to achieve the above goals for hybrid parallel programs on heterogeneous accelerator cluster.

V. Open Source Debugger For Heterogeneous Accelerator Cluster

With the emergence of OpenCL standard[13] after MPI and OpenMP, the hybrid parallel programs on latest heterogeneous multi-accelerator cluster use a combination of MPI, openMP, and OpenCL to have portability across vendor architectures. In Table 1, of section III, presently the accelerator manufactures are extending debugging support for their respective accelerator devices. Research organizations are trying to develop a comprehensive debugger for hybrid (MPI, OpenMP, OpenCL) parallel programs on heterogeneous multi-accelerator clusters.

Developing a complete debugger from scratch for a heterogeneous multi-accelerator cluster would be time consuming and requires huge effort. Instead, it would be better to extend an existing open source debugger to support the requirements of new generation HPC architectures to facilitate OpenCL *kernel* debugging. GDB and LLDB [39] are the most popular open source debuggers for sequential code debugging. These debuggers can be used to debug concurrent multi-process applications; by spawning multiple instances of the serial debugger at cluster nodes in a co-ordinated fashion to take on multitudes of processes of a parallel multi-process application[7]. Open source debuggers can also be extended to recognize new accelerator devices by adding support for new device architectures [40, 41].

In a hybrid parallel program the concurrent CPU processes can be debugged by attaching an instance of sequential debugger (say GDB) to each process. In accelerators the execution happens through multiple lightweight threads on tiny computing cores. A single host process in the application takes care of getting the computation done on an accelerator device. To carryout debugging on programs running on accelerators, the debugger must be able to control the individual or group of thread(s) of executions on the computing cores of an accelerator. In order to reduce the number of instances of sequential debugger spawned for debugging, it would be better to use the same process debugger to debug programs running on the accelerator device. Due to the difference in compute device architectures, the GDB used for debugging host process may not be capable of debugging the accelerator threads. Hence, the GNU Debugger need to be adapted to include debugging of accelerator devices.

Multi-level Hybrid Parallel Program Debugger - Features

The complexity of debuggers will increase in the case of multi-level hybrid parallel programs, due to the multiple processes and threads executing over heterogeneous hardware architectures, in addition to the conventional complexities in parallel programs, such as dead-lock, race conditions and improper communications. There have been no efforts in the last decade for addressing the standardization of High Performance Debugging tools for the accelerator devices. The hardware vendors have built their own proprietary debugging solutions for catering the specific platform and architecture level requirements.

The authors bring out the need to uphold the goals proposed by the HPDF towards realising the hybrid program debugger for heterogeneous multi-accelerator clusters. The design of a standardized open source debugger for heterogeneous accelerator cluster should support the following features, Portability, Scalability, Repeatability and Adaptability.

Portability: The debugger must provide a portable interface to accommodate varied hardware and software preferences of the application developers to visually carry out the debugging activity, irrespective of the underlying heterogeneity in operating systems and window managers. The efforts are being carried out, in such way that the interface part of the debugger will be portable across the multiple operating systems and window managers. Portable user interface toolkits are being used to ensure the portability of the debugger interfaces across multiple systems.

Scalability: Parallel applications exploiting the heterogeneous accelerators based HPC clusters scale to hundreds of threads and processes. Hence, the debugging software employed on these applications, must cater to scalability of the applications. The debugger must be able to intelligently group the set of processes and threads that are meant to be analysed for correctness. It should also issue respective instructions to proceed execution in a controlled environment. In the multi-level hybrid program debugger, processes and threads which are part of the hybrid HPC application being debugged can be grouped, to carryout debug activity for a selected set, and the respective debugging operations are applied to the running processes and threads without hampering the application execution. The number of processes and threads debugged in group can be scaled based on the requirements of the application developer.

Repeatability: The interface and result of the operations must stay consistent across multiple platforms being used, both in the case of the client and execution device platforms, irrespective of the operating systems and accelerators being used in the heterogeneous clusters.

978-1-4673-9286-0/15/$31.00 ©2015 IEEE

Adaptability: Parallel programs can be written in multiple high level languages, like C/C++, FORTRAN. The hybrid program debuggers must adapt to the source level debugging of applications written in these high level languages, along with the accelerator specific parallel programming paradigms. HPC Systems and accelerator vendors also have their own debugging tools, which are specific to their architecture. Those tools may not be available in the open source domain. In such a scenario, the hybrid program debugger for heterogeneous cluster must be able to cater such requirements also. There shall be provisions to seamlessly plug-in third party debuggers to debug the vendor specific codes, so that the debugger can adapt to the hybrid programs of future proprietary accelerators and their programming paradigms.

VI. Extending GDB to Accelerator Level Debugging

The key aspects of porting GDB to a new architecture in terms of - providing ways to read the executable file for the new architecture, description of the Abstract Binary Interface, description of the physical architecture, and operations to access the debuggee program being executed on the new architecture[42].

Debugging a program running on a specific accelerator, GDB needs to understand the respective device architecture and the debugging symbols embedded by the compiler into the executable. To control the execution of the debuggee *kernel* the debugger must have access to the system calls and hardware support(such as trap flag). The accelerator vendors should provide hardware support and API's which can be used by GDB to extend the debugging to accelerator threads.

To carry out source level debugging, GDB must understand the instructions and source code being executed on target accelerator hardware. Functional units of a GNU debugger are shown in the Fig 4. Major sections are, user interface section, symbol handling section, and target system handling section. The user interface section consists of interface modules for interacting with users. The symbol handling section composed of an interpreter for understanding debugging information, a unit for managing the symbol table, a section for parsing and understanding the programming language source code, and a type and value printing module. The target system handling section contains units to control program execution, analysing the stack frames, and unit for physical target manipulation. In this regard, compiler support must be enabled to generate the debugging symbols for the accelerator device hardware. Conforming to a common standard for the debugging symbols [43] would help the interpretation of device *kernel* source code and easier debugging on various accelerator devices.

Defining the target operations, within the GDB, for an accelerator device is essential for inspecting the runtime behaviour and instrumenting the instructions for controlling the execution of device *kernel*(s) on an accelerator device.

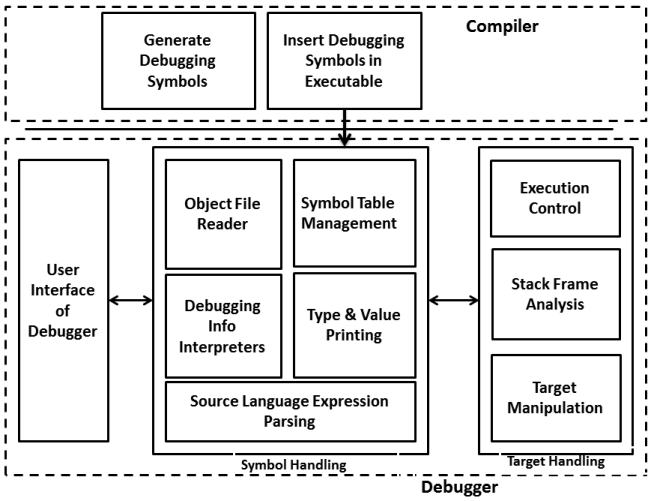

Figure 4. Functional Units of a GNU Debugger

A. Current Scenario in Debugging on Accelerators

Currently available accelerator device architectures are vendor specific, such as NVIDIA and AMD have their own proprietary tools and programming paradigms for their hardware devices. NVIDIA has extended the GDB to support debugging of CUDA *kernels* on their devices with CUDA-GDB. Analysis of the CUDA-GDB code[44] reveals that it uses the CUDA debugger API's at the backend. Hence, target device operations, controls and interpreting the debug symbols are completely abstracted by the CUDA debugger APIs, as shown in the Fig. 5. Thus, this approach does not permit the development of a debugger, which supports OpenCL.

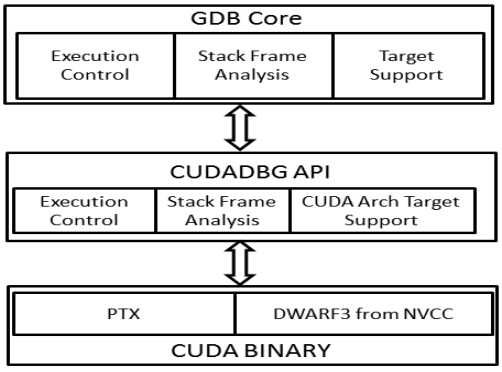

Figure 5. GNU Debugger extension for CUDA

CodeXL is a proprietary tool from AMD, and does not depend on GDB. CodeXL allows debugging of OpenCL *kernel* on AMD devices alone. It will not debug the same OpenCL *kernel* application on a NVIDA GPGPU device. It also does not work on a multi-accelerator environment.

978-1-4673-9286-0/15/$31.00 ©2015 IEEE

VII. Challenges with GDB to Support OpenCL on Accelerators

The primary challenges in achieving debugging of OpenCL based multi-level hybrid programs on multi-accelerator cluster with GDB are given below.

A. Challenges for GDB due to heterogeneous accelerator devices

Most of the accelerators are vendor specific devices. The challenges for GDB to debug applications executing on heterogeneous accelerator devices are the following:

1) Unavailability of information on executable binary

GPGPU vendors follow their own binary file format for the program code execution on the accelerator devices. The binary representation of the OpenCL *kernel*, that is meant for the accelerator's computing cores are generated by the just-in-time compilers along with the device drivers provided by the concerned hardware vendors. This gives complete control for vendors of the accelerator hardware to restrict and relax certain features to paradigms of their interest. This imposes lock on options for analysing the binary code by the third party tools. Also, it limits the scope of performance optimization of OpenCL *kernels* on vendor specific accelerators. The binary format required for the devices must be made open so that the binary file reading utilities understand them.

2) Lack of details on device physical architecture

GPGPU accelerator device vendors protect the hardware internals to get an edge in the competitive market. But this limits lot of opportunities in making the hardware adaptable to multiple programming paradigms. The information provided on the architecture of most accelerators does not reflect the actual internal computing cores and memory architecture of the accelerator device. It is important for the debugging tools to understand the architecture of the device, to perform code analysis and controlling its execution. Lack of such information poses a greater challenge for implementing device architecture module for the GDB to support OpenCL debugging on a vendor specific hardware device. In a multi-accelerator cluster scenario, if the GDB is used for debugging a process which is a part of the OpenCL based parallel hybrid application, the code executing on the vendor specific accelerator will run without the control of GDB, due to the lack of information available on the accelerator device architecture.

3) Little or no debugging operations information available

GPGPU accelerator hardware vendors conceal the information on hardware level instructions that can be potentially harnessed to implement the debugging tools for applications running on an accelerator device. Knowledge of specific instructions available for inspecting the behaviour of a running instance of a program on an accelerator device is very important to analyse and find out the potential areas of errors or bugs creeping in an OpenCL *kernel* program. The information on instructions to control the execution of the *kernel* program, to keep check on the data flow, and also to read values being handled by the variables involved in the application algorithm are necessary to realise the debugging at runtime. This information is currently not available for open source community.

4) No common standard for debugging information

Most of the accelerator vendors embed the debugging symbols for their accelerator code for execution, using their own proprietary compilers and proprietary debugging symbol standard. This restricts the options of debugging, codes running on an accelerator, only to the vendor specific system tools. Unavailability of libraries or APIs to decipher information from such debugging symbols is a challenge for the developers, who wish to extend GDB for the new architecture. The uniformity for representing debugging symbols across multiple accelerator devices will ease the effort required by the developer to adapt the GDB for newer architectures, as well as to debug the code at source level, especially in OpenCL based hybrid parallel applications.

B. Challenges for GDB wrt multi-accelerator architecture

Apart from the implementation specific challenges, there are some more challenges that get added to a GDB based debugging solution, when it is considered for debugging OpenCL based hybrid parallel applications on multi-accelerator clusters.

1) *Multiplicity of Processes and Threads*

When an OpenCL based hybrid parallel application is launched for debugging on a multi-accelerator heterogeneous cluster, the number of processes and threads co-ordinated by a debugging tool will be extremely high. It would nearly impossible to put all the outputs of interactions and execution of threads and processes to be shown on the single interface. This creates greater challenges for grouping and selectively handling the communication and execution of desired, processes and threads, as per the user demands, in real-time.

2) *Localizing the error across multiple threads of execution*

In order to selectively show the desired group of processes and threads of execution in a hybrid parallel program, the potential areas of errors or bugs need to be analyzed well by the application user. The intelligent grouping of processes can be realized with the stack trace analysis of the previous run. But, there are many challenges in realizing the method for pin-pointing probable area for debugging, in a heterogeneous cluster environment. The stack trace must be able to capture the specific device level execution and process or thread identity information, so that the area of error in the code during the previous execution can be tracked down. The core specific thread trace information are not directly available from most the accelerator devices, which again limits the capability of the debugger in localizing potential areas of bugs.

978-1-4673-9286-0/15/$31.00 ©2015 IEEE

3) *Minimizing the number of GDB instances*

Due to the multitudes of processes and threads involved in an OpenCL based hybrid parallel application, any increase in the number of non-contributing processes for the application, will create a performance overhead. Methodologies need to be adopted to decide the optimum number of debugger instances to handle the debugging activities. This shall be accomplished by intelligently localizing the potential area and selecting the specific group of processes and threads for the purpose of debugging.

VIII. Workarounds for Multi-Level Hybrid Program Debugging on Heterogeneous Cluster

In this section the authors briefly present some of the alternate solutions possible for debugging OpenCL based multi-level hybrid programs on heterogeneous accelerator clusters, till the vendor specific restrictions are solved. The approaches given below may not consider the performance aspects when compared to the debugging of the OpenCL kernel on real accelerators cores.

A. Debugging OpenCL Kernel Threads on Host

The details of proprietary architecture, instructions, binary and debug symbols are not revealed for open source community usage. In order to carry out debugging of the processes that run codes on accelerator devices, the respective codes shall be made to run on CPU(s) available to achieve the same functional operations. OpenCL drivers for CPUs can be deployed and enabled to compile the functional code to run on CPUs.

The OpenCL specific drivers for CPUs help the OpenCL *kernel* for an accelerator device, in a hybrid parallel application, to run on the CPU(s) without actually modifying its source code. OpenCL *kernels* can be debugged if they are made to run on the CPUs, instead of the GPGPUs, using the GDB itself. Considering the vastly differing architecture of CPUs and Accelerators, it is impractical to run thousands of threads on the host without suffering a massive performance set back. The above approach may not be an apt one for debugging OpenCL kernels in a performance point of view.

B. Exploiting Device Emulators

Emulators are available to provide a virtualized accelerator device, even if the actual hardware is not physically attached to the system. These emulators allow execution of the device specific codes, in a similar fashion as in the physical hardware. Obviously, it comes at a price of performance overhead. Emulators also capture information for analyzing and debugging the program execution. This help in debugging functional errors in the application, even though it may not exactly represent the dynamic behaviour of the application when it runs on an actual device. Attaching such emulators to the debugging framework for the hybrid applications, can help to debug the device specific codes in the heterogeneous clusters.

The OpenCL Emulator-Debugger (ocl-emu) is an open source project from AMD that allows compilation and debugging of OpenCL kernels as C++ procedures.

C. Post Mortem Analysis

This method involves identifying the potential variable(s), for which the value(s) need to be captured and analysed as shown in Fig. 6. Once the variable is identified, instrumentation of the OpenCL kernel code as well as the host code for sampling the value of the variable under consideration is carried out, without hampering the application algorithm. During the execution of the OpenCL kernel code these value(s) of the variable(s) under analysis are captured and stored. These values are analysed separately upon the termination of the code execution on the target device. Even though this method cannot be treated as a direct replacement of real-time debugging, this would help application developer analyse the potential erroneous values crept in the application during the course of execution.

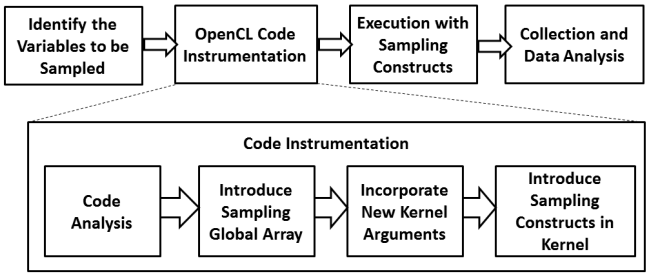

Figure 6. Post Mortem Analysis Approach For Debugging Programs

IX. Suggestions for Heterogeneous Cluster Debugger

With the workarounds suggested in Section VIII, an open source debugger solution can be provided for multi-level hybrid parallel programs. However, developing a standardized high performance debugger for heterogeneous multi-accelerator cluster architectures has several engineering issues in scalability and usability. The multi-level HPD should be - scalable, capable of coordinating large number of processes and threads, and provide a rich user interface. The debugger shall be designed to debug applications spanning across large number of cores, supported by different accelerators. As the number of processes & threads increases, debugger should control the execution of processes and coordinating the debugging operations, without overloading the CPU & Memory. A rich user interface is required to allow the user to carryout interactive debugging on processes and threads at a large scale. Co-ordinating several concurrent debugging windows at the interface client systems and processes execution is essential for easy debugging. Provisioning the processes and threads in groups for selective debugging is essential for handling the scale. Presenting the debugging information of large number of processes and threads, without much effort

978-1-4673-9286-0/15/$31.00 ©2015 IEEE

on the user end, is another key feature for high performance debuggers.

X. CONCLUSION

Petascale systems use multiple heterogeneous accelerators to improve the performance to power ratio. Currently there is no single debugger that supports different types of accelerators. It is useful to have a generic heterogeneous accelerator cluster debugger, conforming to the extended HPDF standards for multi-level hybrid programs on heterogeneous multi-accelerator clusters. Considering that the OpenCL is evolving as a standard, it is useful to develop a MPI-OpenMP-OpenCL application debugger that can work on heterogeneous accelerator cluster architectures. The authors presented the challenges of extending open source debugger for hybrid parallel programs on heterogeneous accelerator clusters and suggested the possible workarounds to solve the problem till the vendor specific restrictions are solved. Co-ordinated efforts from the open source community researchers & hardware vendors are necessary to bring out a unified debugger framework for enabling easy development of parallel hybrid applications.

REFERENCES

[1] Gropp, W.D., "Software for Petascale Computing Systems," Computing in Science & Engineering , vol.11, no.5, pp.17,21, Sept.-Oct. 2009, doi: 10.1109/MCSE.2009.148

[2] W. Xue, et al., "Enabling and Scaling a Global Shallow-Water Atmospheric Model on Tianhe-2," Parallel and Distributed Processing Symposium, 2014 IEEE 28th International , vol., no., pp.745,754, 19-23 May 2014 doi: 10.1109/IPDPS.2014.82

[3] Rogers, Jim, "Power Efficiency and Performance with ORNL's Cray XK7 Titan," High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion: , vol., no., pp.1040,1050, 10-16 Nov. 2012, doi: 10.1109/SC.Companion.2012.126

[4] H. Jina, et al., "High performance computing using MPI and OpenMP on multi-core parallel systems", Parallel Computing, Vol 37, Issue 9, Pages 499-652 (September 2011)

[5] B. Carnes, et al., "Science at LLNL with IBM Blue Gene/Q," IBM Journal of Research and Development , vol.57, no.1/2, pp.11:1,11:18, Jan.-March 2013, doi: 10.1147/JRD.2012.2233371

[6] Parallel Tools Consortium, HPD (High Performance Debugging) Version 1 Standard: Command Interface for Parallel Debuggers, Sep. 1998.

[7] S. M. Balle et al., "Extending a traditional debugger to debug massively parallel applications", Journal of Parallel and Distributed Computing , Vol 64, Issue 5, May 2004, pp 617 - 628

[8] V. V. Kindratenko, et al., "GPU Clusters for High-Performance Computing", IEEE International Conference on Cluster Computing and Workshops, 2009. CLUSTER '09

[9] M. Showerman, et al., "QP: A Heterogeneous Multi-Accelerator Cluster", LCI International Conference on High Performance Clustered Computing, March 2009

[10] Deepika, H.V.; Mangala, N.; Babu, N.S.C., "Extendable framework for monitoring heterogeneous multi-accelerator HPC cluster," Computing for Sustainable Global Development (INDIACom), 2014 International Conference on , vol., no., pp.244,249, 5-7 March 2014, doi: 10.1109/IndiaCom.2014.6828136

[11] Chao-Tung Yangm, et al., "Hybrid Parallel Programming on GPU Clusters", International Symposium on Parallel and Distributed Processing with Applications, 2010

[12] Sylvain Contassot-Vivier, Stéphane Vialle, Jens Gustedt, "Development methodologies for GPU and cluster of GPUs", Chapman & Hall, Ed. 2013

[13] Aaftab Munshi, "The OpenCL Specification", Version 1.1, Document Revision 48, June 2010

[14] "NVIDIA CUDA - compute unified device architecture : Programming Guide", 2011, NVIDIA Corporation,

[15] Chapman, B.; Jost, G.; van der Pas, R , "Using OpenMP:Portable Shared Memory Parallel Programming", MIT Press, 2007

[16] Gropp, W.; Lusk, E., "The MPI communication library: its design and a portable implementation," Scalable Parallel Libraries Conference, 1993., Proceedings of the , vol., no., pp.160,165, 6-8 Oct 1993, doi: 10.1109/SPLC.1993.365571

[17] Debugging Massively Parallel Applications: Overview, Intel Debugger User's and Reference Guide for Intel C++ Composer XE 2013, Intel

[18] D.C. Arnold, et al., "Stack Trace Analysis for Large Scale Debugging," Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International , vol., no., pp.1,10, 26-30 March 2007, doi: 10.1109/IPDPS.2007.370254

[19] Vern Paxson , Prof Anderson, "A Survey of Support For Implementing Debuggers" (1990), (Accessed on 17 October 2014)

[20] Wei Cui; Weiguo Wu; Yixin Zhao; Leiqiang Zhang, "Research and Design of a Parallel Debugging Interface in a New Parallel Debugger Framework," Grid and Cooperative Computing, 2008. GCC '08. Seventh International Conference on , vol., no., pp.70,74, 24-26 Oct. 2008, doi: 10.1109/GCC.2008.54

[21] Comparison of Debuggers, Wikipedia Article, (Accessed on 29 May 2014)

[22] S. M. Balle et al., "Extending a traditional debugger to debug massively parallel applications", Journal of Parallel and Distributed Computing , Vol 64, Issue 5, May 2004, pp 617 - 628

[23] Debugging with GDB: the GNU Source-Level Debugger for GDB (GDB) Version 7.8.0.20141017

[24] Ian Lumb, "Petascale Debugging via Allinea DDT for IBM Blue Gene /P and IBM Blue Gene /QAllinea DDT", ALCF L2P Workshop, May 23, 2012

[25] Ed Hinkel, "Debugging with TotalView on the Blue Gene Q", MIRA Performance Boot Camp, May 21-23, 2013

[26] Lindholm, E.; Nickolls, J.; Oberman, S.; Montrym, J., "NVIDIA Tesla: A Unified Graphics and Computing Architecture," Micro, IEEE , vol.28, no.2, pp.39,55, March-April 2008, doi: 10.1109/MM.2008.31

[27] Tomasz Boinski, Pawel C, Pawel R, Michal W, "Parallel Processing in CUDA/OpenCL Laboratories", Jan 2014,

[28] CUDA-GDB, the NVIDIA CUDA debugger for Linux and Mac OS, NVIDIA Documentation (Accessed on 18 October 2014)

[29] Taylor, R.; Xiaoming Li, "A Micro-benchmark Suite for AMD GPUs," Parallel Processing Workshops (ICPPW), 2010 39th International Conference on , vol., no., pp.387,396, 13-16 Sept. 2010, doi: 10.1109/ICPPW.2010.59

[30] gDebugger CL, Graphics Remedy Documentation (Accessed on 18 October 2014)

[31] CodeXL – Powerful Debugging, Profiling & Analysis, AMD Developer Documentation (Accessed on 17 October 2014)

[32] PGDBG Graphical Symbolic Debugger, Portland Group Documentation (Accessed on 17 October 2014)

[33] NVIDIA Nsight, NVIDIA Documentation (Accessed on 17 October 2014)

[34] DIViA- Portable Parallel Debugging Environment, CDAC Documentation (Accessed on 17 October 2014)

[35] P.K. Sinha, et al., "Current state and future trends in high performance computing and communications (HPCC) research in India," Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of , vol., no., pp.217,220, 26-28 May 2004, doi:10.1109/FTDCS.2004.1316619

[36] DWARF standards, Dwarf Standards Repository(Accessed on 17 October 2014)

978-1-4673-9286-0/15/$31.00 ©2015 IEEE

[37] Dong H Ahn, Bronis R de Supinski, Ignacio Laguna, Gregory L Lee, Ben Liblit, Barton P Miller, Martin Schulz, "Scalable temporal order analysis for large scale debugging", Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Nov 2009

[38] Ignacio Laguna, Todd Gamblin, Bronis R de Supinski, Saurabh Bagchi, Greg Bronevetsky, Dong H Anh, Martin Schulz, Barry Rountree, "Large scale debugging of parallel tasks with AutomaDeD", SC'11 International Conference for High Performance Computing, Networking, Storage and Analysis

[39] The LLDB Debugger, LLDB Documentation (Accessed on 17 October 2014)

[40] GDB Wiki, Internals Adding-a-New-Target, Sourceware Documentation, (Accessed on 17 October 2014)

[41] LLDB Debugger architecture, LLDB Architecture Documentation, (Accessed on 18 October 2014)

[42] Jeremy Bennett , "Howto: Porting the GNU Debugger, Practical Experience with the OpenRISC 1000 Architecture", Embecosm , November 2008

[43] Stoppe, J.; Wille, R.; Drechsler, R., "Data extraction from SystemC designs using debug symbols and the SystemC API," VLSI (ISVLSI), 2013 IEEE Computer Society Annual Symposium on , vol., no., pp.26,31, 5-7 Aug. 2013, doi: 10.1109/ISVLSI.2013.6654618

[44] CUDA-GDB, Source Repository of NVIDIA cuda-gdb, (Accessed on 18 October 2014)

978-1-4673-9286-0/15/$31.00 ©2015 IEEE