

УДК 004.4'233

DOI: 10.53403/9785951505309_2022_27_1_168

Новые возможности отладчика параллельных программ PD

Приводится описание новых возможностей отладчика параллельных программ PD: неинтерактивной отладки, поддержки графических ускорителей, коммуникационной схемы «дерево». Описаны изменения в графическом интерфейсе пользователя.

А. Б. Киселёв, С. Н. Киселёв

Введение

В РФЯЦ-ВНИИЭФ ведется разработка отладчика параллельных программ PD (Parallel Debugger) [1], который входит в состав дистрибутивов системного программного обеспечения (ПО) супер-ЭВМ со встроенными средствами защиты информации от несанкционированного доступа [2] и защищенной операционной системы «Араמיד» [3].

В результате активного использования отладчика PD разработчиками программных комплексов математического моделирования физических процессов и пакета программ инженерных расчетов ЛОГОС появились пожелания доработать некоторые программные компоненты отладчика, а также обеспечить возможность отладки программ на графических ускорителях. Со времени выхода в печать статьи [1] авторы заменили в PD транспортную подсистему передачи информации – создали коммуникационную схему *дерево*, которая позволила значительно уменьшить время передачи информации и, следовательно, время реакции отладчика, реализовали новый программный компонент PD – неинтерактивный отладчик, обеспечили возможность отладки программ на графических ускорителях фирмы NVIDIA. Данная статья посвящена описанию новых свойств и возможностей, реализованных в последней версии отладчика PD.

Коммуникационная схема «дерево»

В ранних версиях отладчика PD была реализована схема, в которой каждый программный агент PD связан с сервером команд и служебных сообщений, входящим в состав программы графического интерфейса пользователя. Такая схема проста в реализации и хорошо работает, пока с сервером отладчика PD взаимодействуют до сотен программных агентов. Как только к серверу подключаются тысячи агентов, данная схема становится причиной общего замедления процесса отладки: значительная нагрузка на коммуникационную подсистему инструментального сервера, на котором запущена программа графического интерфейса отладчика PD, препятствует прохождению команд к программным агентам отладчика и служебной информации от программных агентов.

Для распределения нагрузки по узлам вычислительной системы в отладчике PD была реализована коммуникационная схема *дерево*, в которой с программой графического интерфейса (верхняя точка рис. 1) связаны, например, три программных агента, каждый из которых, в свою очередь, соединен еще с четырьмя программными агентами и т. д.

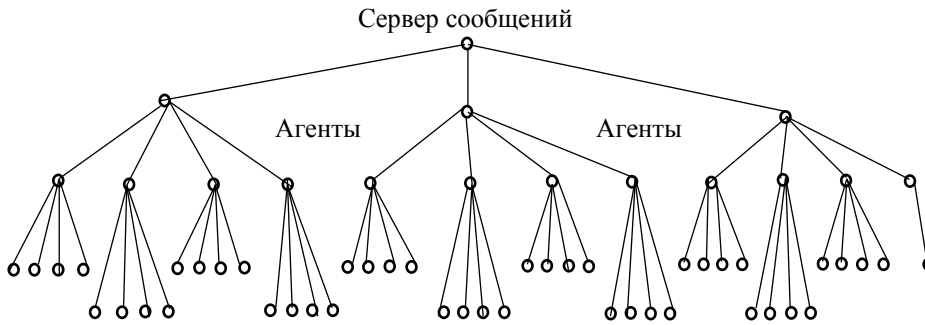


Рис. 1. Пример коммуникационной схемы *дерево*

Сервер и программный агент могут иметь разное число связей, так как для регулирования количества связей используются разные конфигурационные параметры.

Новая коммуникационная схема позволила уменьшить время формирования связей между программными агентами PD и сервером команд в процессе создания отладочной сессии, а также время передачи команд и сообщений в процессе отладки программы. В табл. 1 приведено время формирования коммуникационного дерева в зависимости от количества программных агентов в момент создания отладочной сессии.

Таблица 1

Время формирования коммуникационного дерева в начале отладочной сессии

Количество агентов	500	1000	1500	2500	3000	4400	6000	8000	10000
Время, мс	19	42	74	119	141	206	343	417	564

В табл. 2 приведены замеры времени фактического выполнения строки программы параллельным отладчиком PD. Время включает передачу информации (команды *выполнить шаг* в формате машинно-ориентированного интерфейса MI) программным агентам, выполнение команды *выполнить шаг* отладчиком GDB [1] и передачу информации с результатом выполнения команды серверу.

Таблица 2

Общее время выполнения строки программы

Количество процессов	500	1000	2000	3000	4000	6000	8000	10000
Время, с	0,14	0,29	0,94	1,39	3,25	4,56	5,54	8,83

Авторы полагают, что в будущем время передачи можно будет уменьшить, изменив применяемый в отладчике PD формат команд и служебных сообщений, – формируя так называемые групповые команды, содержащие, например, команду и список агентов, которым она предназначена. Таким же способом можно сократить размер и количество служебных сообщений, передаваемых от программных агентов PD серверу команд. В целом реализация коммуникационной схемы *дерево* позволила сократить общее время выполнения строки программы более чем в 2,5 раза.

Неинтерактивная отладка программы

Новая возможность, появившаяся в последней версии параллельного отладчика PD, – неинтерактивная (offline) отладка – позволяет отлаживать программу без участия пользователя. Такая отладка может быть полезной, когда требуется отладить ресурсоемкую программу, для которой на вычислительной системе нет свободных вычислительных ресурсов, а в отсутствие разработчика они могут появиться.

Для offline-отладки пользователь должен сформировать пакетное задание, указав название исполняемого файла, количество узлов/процессоров, время выполнения и т. д. Кроме того, необходимо ввести конфигурацию профилирования, название журнального файла, метрики точек прерывания/наблюдения, названия глобальных программных переменных и ранг наблюдаемого MPI-процесса. По мере ввода информация о точках прерывания, наблюдения и программных переменных помещается в общую таблицу графического окна ввода параметров offline-сессии, показанную на рис. 2. Из этой таблицы пользователь может удалить строки, а также добавить информацию, указав файл, в котором она находится.

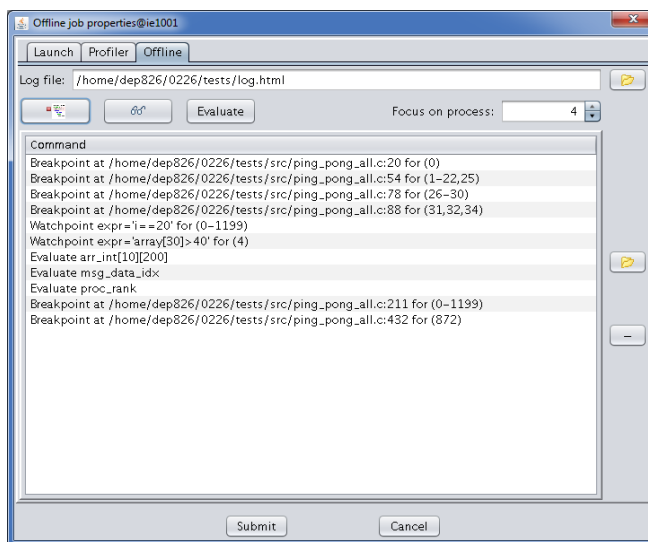


Рис. 2. Графическое окно ввода параметров offline-сессии отладки

После щелчка мышью по кнопке *Submit* программа графического интерфейса пользователя запускает offline-отладчик, которому она передает атрибуты пакетного задания, конфигурацию профилирования и метрики отладочной сессии. После этого offline-отладчик в отдельных программных потоках запускает обработчик служебных сообщений*, программу контроля завершения задания и обработчик стандартной выдачи и диагностики процессов отлаживаемой программы.

В соответствии со схемой инициализации отладочного задания, которая подробно описана в [1], агенты отладчика PD запускаются системой пакетной обработки заданий на узлах вычислительной системы. Между сервером сообщений и программными агентами отладчика формируется коммуникационное дерево для передачи команд/служебных сообщений. Сервер сообщений посылает агентам PD команду – *exec-run-start* (запустить программу и остановиться на точке входа

* Служебные сообщения содержат информацию о достижении точки прерывания/наблюдения, поступлении программного сигнала, завершении программного потока, останове программы и т. п.

в программу). Все программные агенты выполняют ее с помощью отладчика GDB, а потом оповещают сервер команд о готовности к отладке. После этого offline-отладчик устанавливает точки прерывания/наблюдения и продолжает выполнение программы.

Если в контролируемом процессе сработала точка прерывания/наблюдения, то offline-отладчик с помощью соответствующих MI-команд GDB считывает значения локальных переменных процедуры/функции, указанных глобальных переменных, а также данные о программных потоках и стеке программы. Эту информацию offline-отладчик записывает в журнальный файл HTML-формата. Затем он продолжает выполнение прерванного процесса.

Точки прерывания, которые срабатывают в других процессах, игнорируются. Однако если какой-либо процесс получает программный сигнал, то offline-отладчик всегда считывает и записывает в журнальный файл всю перечисленную выше информацию.

По завершении программы offline-отладчик записывает в журнальный файл профилировочную информацию, а затем информацию из файла стандартной выдачи и диагностики. После этого offline-отладчик завершает свою работу.

Формат журнального файла offline-отладчика

Журнальный файл, который создает offline-отладчик, логически разбит на три части. Первая часть («Messages») содержит таблицу (рис. 3) с информацией о событиях, которые произошли в процессе отладки (старт программы, срабатывание точек прерывания/наблюдения и т. д.).

Time	Processes	Type	Message
00:00:00	0-23		Launching job 141899.ce100 at Fri Feb 01 10:09:03 GMT+03:00 2019
00:00:00	0-23		Startup complete
00:00:01	0-23	■	Add breakpoint at /home/dep826/0226/tests/fort/mod.f90:25
00:00:01	0-23	■	Add breakpoint at /home/dep826/0226/tests/fort/mod.f90:38
00:00:02	0	💡	Process stopped at breakpoint in prom() at /home/dep826/0226/tests/fort/mod.f90:24 ▼Stack Stack arguments #3 _start() #2 __libc_start_main() at /lib64/libc.so.6 #1 main() #0 prom() at /home/dep826/0226/tests/fort/mod.f90:24 ▼Locals ▼Threads ▼Evaluate ▼Summary

Рис. 3. Пример таблицы «Messages»

При срабатывании точки прерывания или получении программного сигнала *offline-отладчик* записывает в таблицу строку с названием исходного файла и номером строки программы, в которой произошло прерывание. Рядом записывается информация о стеке (*Stack*), локальных переменных процедуры/функции (*Locals*), программных потоках (*Threads*), наблюдаемых глобальных переменных (*Evaluate*) и резюме (*Summary*), скрытые за символом ▼. Кликнув мышью по данному символу, можно раскрыть запись (в таблице на рис. 3 раскрыта запись *Stack*). Пункт *Summary* содержит сводную информацию о срабатывании точек прерывания/наблюдения в процессах программы в момент возникновения прерывания или срабатывании точки наблюдения у контролируемого процесса в течение некоторого короткого промежутка времени.

Во вторую часть журнального файла («Profiles») записывается информация профилирования программы, которая содержит данные, получаемые с помощью свободно распространяемых профилировщиков Google Performance Tools [4] и MpiP [5] (табл. 3). Если пользователь не расставил «галочки» на вкладке *Profiles* при создании *offline-отладки*, то часть «Profiles» в журнальном файле будет отсутствовать.

Таблица 3

Пример профиля использования кучи (*heap-use profile*) в журнальном файле *offline-отладчика* [5]

Function	Full path	Objs	Objs %	In-use space	In-use space %	Allocated objs	Allocated objs %	Allocated objs space	Allocated objs space %
strdup	/usr/lib64/libc-2.12.so	0	0,0	0	0,0	1	0,3	30	0,0
??*	/usr/lib64/libstdc++.so.6.0.13	1	25,0	30	0,4	3	0,9	89	0,0
ProfileHandler::Init	/home/dep826/0226/PD/gperftools-2.7/src/profile-handler.cc	1	25,0	64	0,8	1	0,3	64	0,0

Примечание: знаки «??» означают, что профилировщик не смог определить название функции. Появляются в профиле при использовании программных библиотек без отладочной информации.

Последняя, третья часть журнального файла содержит информацию из стандартного вывода и диагностики процессов программы. Диагностическая информация выделяется красным цветом.

Поддержка CUDA

В последней версии отладчика PD реализована возможность отлаживать программу на GPU (Graphics Processing Unit). Собственно отладка программы на GPU выполняется проприетарным отладчиком GDB фирмы NVIDIA (NVIDIA-GDB), который входит в состав CUDA SDK [6]. Для формирования и обработки нестандартных MI-команд и служебных сообщений NVIDIA-GDB, управления отладкой CUDA-программы были доработаны сервер команд и графический интерфейс отладчика PD.

Надо отметить, что в составе CUDA SDK имеется отладчик *nsight*, который позволяет отлаживать программы, использующие графические ускорители фирмы NVIDIA, но *nsight* не обеспечивает отладку MPI-программ на вычислительных системах. Отлаживать MPI-программы позволяют только отладчики Allinea DDT [7] и TotalView, но это зарубежные коммерческие продукты, которые большинству отечественных программистов недоступны.

MI-команды для NVIDIA-GDB

В процессе отладки программы на компьютерах семейства Intel x86 или, например, «Эльбрус» отладчик PD посылает стандартные MI-команды и принимает стандартные служебные сообщения – ответы. Однако если программа выполняется в GPU, то NVIDIA-GDB выдает информацию в формате, отличном от описанного в руководстве GNU GDB. Приведем ее пример:

```
CudaFocus={device="0",sm="0",warp="0",lane="0",kernel="0",grid="1",
           blockIdx="(0,0,0)",threadIdx="(255,0,0)"}
```

Данное сообщение свидетельствует, что код отлаживаемой программы выполняется на GPU с индексом 0, индекс блока (0, 0, 0) и индекс нити (255, 0, 0) и т. д. Дальнейшее управление отладкой может выполняться только с помощью команд с приставкой `-cuda`:

```
-cuda-focus-switch, -cuda-info-kernels, -cuda-info-blocks,
-cuda-info-threads и др.
```

Для выполнения переключений между нитями в графическом интерфейсе отладчика PD значения метрик `device`, `warp`, `kernel`, `blockIdx` и `threadIdx` записываются в характеристику процесса, а приведенные выше команды посылаются NVIDIA-GDB до тех пор, пока программный код использует GPU.

Рассмотрим этапы переключения отладчика на программную нить с индексом (53,0,0). Графический интерфейс посылает NVIDIA-GDB MI-команду

```
-cuda-focus-switch kernel 0 block (0,0,0) thread (53,0,0)
```

Ответ отладчика NVIDIA-GDB с результатом ее выполнения может содержать, например, такую информацию:

```
CudaFocus={device="0",sm="0",warp="1",lane="21",kernel="0",grid="1",
           blockIdx="(0,0,0)",threadIdx="(53,0,0)",frame={addr="0x0000000000dbd720",
           func="bitreverse",args=[{name="data",value="0x7fffce600000"}]},
           file="bitreverse.cu",fullname="/home/dep826/kiselev_ab/CUDA/bitreverse.cu",
           line="12.
```

Соответствующий программный класс сервера команд обрабатывает информацию об успешном переключении на нить (53,0,0), а затем графический интерфейс отладчика PD выполняет смену индексов в переключателях ядра, блока и нити.

Графические компоненты для отладки программы на GPU

Переключатель ядра, блока и программной нити. Новая версия отладчика PD позволяет переключать ядро, блок или нить в режиме отладки программных потоков (рис. 4). Переключатель появляется в графическом интерфейсе отладчика PD тогда, когда отладчик в служебном сообщении получает блок `CudaFocus={...}`, т. е. код программы выполняется на GPU.

Чтобы пользователь мог указывать в переключателях доступные в данный момент индексы, при каждой остановке программы отладчик PD считывает из GPU граничные значения индексов `kernel`, `block` и `thread`. Затем отладчик конфигурирует переключатели индексов так, чтобы пользователь мог устанавливать только действующие значения индексов. Операция переключения на ядро, блок и нить выполняется после клика мышью по кнопке «Ok».



Рис. 4. Переключатели индексов ядра, блока и нити для отладки программы на GPU

Переключатель нити автоматически переключается на «живую» нить после завершения отлаживаемой нити.

Информация о программных нитях. При выполнении кода программы на GPU в графическом интерфейсе отладчика PD на закладке *Thread* появляется таблица «CUDA Thread», содержащая список GPU-нитей (рис. 5).

Thread	Function	State	CUDA Thread
1	poll	stopped	simpleMultiGPU.cu:57
2	accept4	stopped	simpleMultiGPU.cu:58
3	poll	stopped	simpleMultiGPU.cu:57
4	poll	stopped	simpleMultiGPU.cu:58

Tooltip text:
 /home/dep826/kiselev_ab/samples/0_Simple/simpleMultiGPU/simpleMultiGPU.cu:57
 32 CUDA threads:
 <<<(0,0,0)(160,0,0)>>> ... <<<(0,0,0)(191,0,0)>>>

Рис. 5. Пример таблицы «CUDA Thread» с всплывающей подсказкой

Если пользователю нужен диапазон индексов *blockIdx* и *threadIdx*, то они отображаются во всплывающей подсказке, которая появляется при удержании указателя мыши над таблицей «CUDA Thread». Подсказка содержит полный путь к исходному файлу, номер исходной строки (числа 57 и 58 на рис. 5), общее количество программных нитей (32), а также их начальный и конечный индексы (строка с обрамлением из символов «<<» и «>>»).

Графический интерфейс отладчика PD скрывает таблицу «CUDA Thread», если процесс более не использует GPU.

Информация о GPU. вкладка *GPU Devices* (рис. 6) также появляется при использовании программой GPU. На вкладке отображается сводная информация о доступных/используемых процессами устройствах.

Attribute name	Value
▼ Ranks 0-180	Have device
▼ GV100GL-A	2 device
IDs	0,1
Compute capability	sm_70
Number of SMs	80
Warps per SM	64
Lanes per Warp	32
Registers per Lane	256
Ranks 181-800	No device

Рис. 6. Пример содержимого вкладки *GPU Devices*

Так, например, на рис. 6 показано, что процессам с рангами от 0 по 180 доступны по два устройства с указанными характеристиками, остальные процессы не используют GPU, о чем свидетельствует фраза «No device».

Заключение

Параллельный отладчик успешно применяется в РФЯЦ-ВНИИЭФ для отладки программных комплексов моделирования физических процессов и пакета программ инженерных расчетов ЛОГОС. Он входит в дистрибутив системного ПО супер-ЭВМ со встроенными средствами защиты информации от несанкционированного доступа, а также в состав защищенной операционной системы «Арамид».

По своим возможностям отладчик PD очень близок к Allinea DDT и TotalView [8], но в отличие от них у него нет лицензионных ограничений: PD позволяет отлаживать и профилировать любое количество процессов параллельной Си- или Фортран-программы на любом количестве процессоров. Кроме того, последняя версия отладчика PD качественно отличается от предыдущих: за счет создания другой транспортной подсистемы передачи информации уменьшилось время реакции на управляющее воздействие, реализован offline-отладчик, обеспечена отладка CUDA-программ.

Список литературы

1. Киселёв А. Б., Киселёв С. Н., Семёнов Г. П. Отладчик параллельных программ для кластеров на базе ОС Linux // Вопросы атомной науки и техники. Сер. Матем. моделирование физ. процессов. 2018. Вып. 2. С. 72–80.
2. Кульнев Д. В., Модянов Р. В., Петрик А. Н. Защищенная ОС // Открытые системы. СУБД. 2015. № 4.
3. Петрик А. Н. Защищенная операционная система «Арамид» для супер-ЭВМ // Национальный суперкомпьютерный форум (НСКФ-2019). Тез. докл. (26–29 ноября), 2019. С. 3 [Электронный ресурс]. – http://2019.nscf.ru/TesisAll/01_Systemnoe_i_promezhytochnoe_PO/040_PetrikAN.pdf.
4. Google Performance Tools [Electronic resource] – <http://code.google.com/p/google-perftools>.
5. Профилировщик MpiP [Электронный ресурс] – <http://mpip.sourceforge.net>.
6. Документация CUDA SDK. [Электронный ресурс] – <http://docs.nvidia.com/cuda/eula/index.html>.
7. The Distributed Debugging Tool [Electronic resource] – ресурс: <http://www.allinea.com>.
8. TotalView [Electronic resource] – <http://www.totalviewtech.com>.

New Capabilities of an Interactive Parallel Programm Debugger (PD)

A. B. Kiselev, S. N. Kiselev

The paper describes new capabilities of a parallel program debugger (PD), such as non-interactive debugging, support of graphics accelerators and a «tree»-like communication scheme. Modifications to the graphical user's interface are described.