DOI: 10.15514/ISPRAS–2020–32(4)–7

Отладчик параллельных программ для ОС Linux

А.Б. Киселев, ORCID: 0000-0002-6124-2359 <abkiselev@vniief.ru>
С.Н. Киселев, ORCID: 0000-0002-4236-6516 <snkiselev@vniief.ru>
Российский федеральный ядерный центр –
Всероссийский научно-исследовательский институт экспериментальной физики,
607188, Россия, Нижегородская область, г. Саров, пр. Мира, 37

Аннотация. В статье представлен отладчик параллельных программ, написанных на языке программирования Си/Си++ или Фортране, которые предназначены для выполнения на высокопроизводительных вычислительных системах. В работе раскрывается схема взаимодействия компонентов отладчика параллельных программ, представлен алгоритм обработки результатов профилирования программы с помощью встроенных средств профилирования. Описаны возможности графического интерфейса пользователя и отладчика в целом. В статье рассказано о развитии отладчика параллельных программ, в частности о реализации коммуникационной древовидной схемы соединения его компонентов между собой, о режиме неинтерактивной отладки, о поддержке графических ускорителей корпорации Nvidia.

Ключевые слова: отладка параллельной программы; параллельный отладчик; CUDA

Для цитирования: Киселев А.Б., Киселев С.Н. Отладчик параллельных программ для ОС Linux. Труды ИСП РАН, том 32, вып. 4, 2020 г., стр. 97-114. DOI: 10.15514/ISPRAS–2020–32(3)–7

A debugger of parallel programs for OS Linux

A.B. Kiselev, ORCID: 0000-0002-6124-2359 <abkiselev@vniief.ru>
S.N. Kiselev, ORCID: 0000-0002-4236-6516 <snkiselev@vniief.ru>
Russian Federal Nuclear Center – All-Russian Research Institute of Experimental Physics,
37 Mira st., Sarov, Nizhny Novgorod region, 607188, Russia

Abstract. The paper presents a debugger for parallel programs in C/C++, or FORTRAN, which are executed in high-performance computers. The debugger's program components and mechanism of their interaction are described. The graphic user's interface capabilities are discussed and the profiling procedure using built-in profiling tools is described. The paper contains of the description of the new parallel debugger capabilities such as a communication treelike scheme of his components connection, and a non-interactive debugging mode, and the support of Nvidia's graphic accelerators. Currently, the debugger provides launching of debug jobs in the systems of batch processing of jobs such as Open PBS / Torque, SLURM, and CSP JAM but it can be configured for other systems. The PD debugger allows to debug program processes and threads, manage breakpoints and watchpoints, logically divide program processes into subsets, manage them, change and view variables, and profile the debugged program using the free Google Performance Tools and mpiP. The PD debugger is written in the Java programming language, intended for debugging programs on Unix / Linux operating systems, and it uses free software components such as SwingX, JHDF5, Jzy3D, RSyntaxTextArea, and OpenGL.

Keywords: parallel program debugging; parallel debugger; CUDA

For citation: Kiselev A.B., Kiselev S.N. A debugger of parallel programs for OS Linux. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 4, 2020. pp. 97–114 (in Russian). DOI: 10.15514/ISPRAS–2020–32(4)–7

1. Введение

Отладка параллельного математического программного комплекса, предназначенного для выполнения на высокопроизводительной вычислительной системе (ВС), – важный этап разработки программного обеспечения, требующий применения специального программного средства – отладчика.

Для отладки на ВС программисты могут использовать как коммерческие, так и свободно распространяемые отладчики. Коммерческие отладчики – это TotalView [1] и Allinea DDT [2]. Их лицензии ограничивают пользователей определенным количеством одновременно отлаживаемых процессов. Свободно распространяемые отладчики не имеют таких ограничений. Среди зарубежных разработок выделяется свободно распространяемый отладчик проекта Eclipse Parallel Tools Platform [3] (PTP), который за несколько лет развития получил необходимый набор возможностей для отладки параллельной программы. Отечественные программные средства отладки – отладчик параллельных программ PDB [4], разработанный и использовавшийся в РФЯЦ-ВНИИЭФ, GEPARD [5] (СО РАН), диалоговый отладчик программ, написанных на непроцедурном языке НОРМА [6] (ИПМ им. М.В. Келдыша, РАН), – судя по отсутствию новых публикаций и исходных кодов, не развиваются.

Описанный в данной статье отладчик PD (parallel debugger) [7] заполняет свободное место в ряду средств отладки параллельных программ, разрабатываемых для отечественных ВС. Отладчик PD обеспечивает отладку программ на Си/Си++ или Фортране. В отладчике PD используются усовершенствованные авторами статьи версии GNU debugger (GDB) [8]. Графический интерфейс отладчика похож на графический интерфейс Allinea DDT, его можно настроить на сочетание *горячих клавиш* и цветовое оформление исходного текста отладчиков Microsoft Visual Studio, Eclipse, IDEA и Allinea DDT. В настоящее время отладчик обеспечивает запуск отладочных заданий в системах пакетной обработки заданий Open PBS/Torque [9], SLURM [10] и СПО JAM [11], но может быть настроен и на другие системы.

Отладчик PD позволяет отлаживать процессы и потоки программы, управлять точками прерывания и наблюдения, логически делить процессы программы на подмножества, управлять ими, изменять и просматривать переменные, а также выполнять профилирование отлаживаемой программы с использованием свободно распространяемых профилировщиков Google Performance Tools [12] и mpiP [13].

Отладчик PD написан на языке программирования Java, предназначен для отладки программ в ОС Unix/Linux, в нем применяются такие свободно распространяемые программные компоненты, как SwingX [14], JHDF5 [15], Jzy3D [16], RSyntaxTextArea [17] и OpenGL [18].

2. Программные компоненты отладчика

Отладчик PD состоит из программы графического интерфейса пользователя, сервера команд и сообщений, агента (рис. 1). В качестве базового отладчика используется GDB. Графический интерфейс и сервер команд и сообщений выполняются в разных потоках одной программы. Вариант отладки на ВС подразумевает, что они запускаются пользователем на инструментальном сервере ВС. Сервер команд и сообщений посылает программным агентам MI-команды [8] (команды машинно-ориентированного текстового интерфейса GDB), а программные агенты, в свою очередь, пересылают их отладчикам GDB. Информацию о результатах выполнения команд (сообщения) отладчики GDB передают программным агентам, которые пересылают её серверу команд и сообщений. Программные агенты не только пересылают информацию, но и контролируют стандартные потоки процессов программы.

В ходе выполнения программы отладчики GDB формируют асинхронные сообщения, которые не связаны с MI-командой, потому что вызваны, например, срабатыванием точки прерывания или наблюдения. Такие сообщения обрабатываются отдельно, а содержащаяся в

них информация отображается во всплывающем графическом окне, чтобы пользователь её не пропустил.

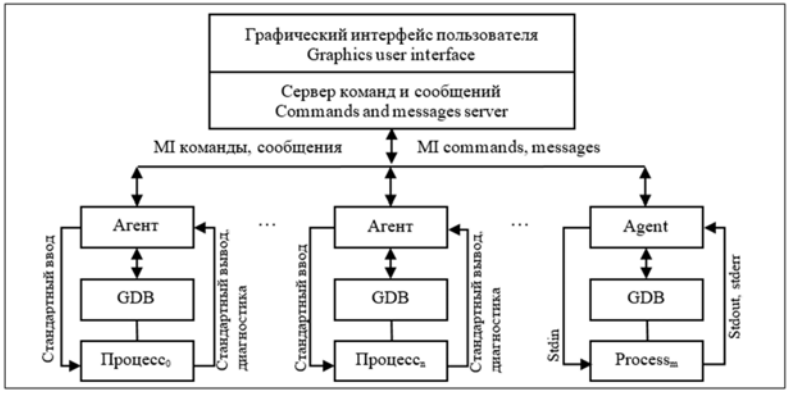

Рис. 1. Схема отладчика PD
Fig. 1. PD debugger schematic

Программные агенты, отладчики GDB и процессы параллельной программы запускаются на вычислительных узлах ВС.

2.1 Программа графического интерфейса пользователя, сервер команд и сообщений

Управление отладкой осуществляется посредством программы графического интерфейса, которая обеспечивает отображение списка названий исходных файлов, значений переменных программы, стандартной выдачи и диагностики процессов, их состояний, загрузку и сохранение параметров сессии - кодировка, шрифт и его размер, точки останова, наблюдения и т.д.

Сервер команд и сообщений выполняется в отдельном программном потоке, он необходим для связи с программными агентами. Сервер формирует текстовые команды машинно-ориентированного командного интерфейса GDB, посылает команды программным агентам, обрабатывает результаты их выполнения и передает их программе графического интерфейса.

2.2 Программный агент

Программный агент реализован для передачи MI-команд отладчику GDB, информации на стандартный ввод отлаживаемого процесса, сообщений с результатами их выполнения, стандартной выдачи и диагностики отлаживаемого процесса серверу команд и сообщений. Взаимодействие программного агента с отладчиком GDB осуществляется посредством псевдотерминалов[1]. Процесс отлаживаемой программы также использует псевдотерминалы для ввода/вывода информации.

Важной функцией программного агента является обработка результатов профилирования программы.

[1] Псевдотерминал – эмулятор терминала в ОС Unix/Linux (PTY), псевдоустройство, используемое для взаимодействия пользователя с локальным или удаленным компьютером.

2.3 Базовый отладчик GDB

Отладчик GDB обеспечивает отладку процессов программы. Управление отладчиком GDB осуществляется с помощью команд его текстового машинно-ориентированного интерфейса (MI).

В отладчике PD используется одна из модифицированных авторами статьи версий GDB – 7.11.1 или 7.12.1. Внесенные в отладчик GDB исправления позволили повысить надежность его функционирования, благодаря им пользователь обеспечен информацией о модулях Фортран-программы, функциях или процедурах, о типах указателей на массивы и многом другом. Для отладки программы, например, на графических ускорителях может быть использована версия отладчика GDB без модификаций[2], но в этом случае перечисленная выше информация отображаться не будет.

3. Опции компилятора

Для отладки программы её необходимо скомпилировать с ключом *-g* и использовать минимальный уровень оптимизации (*-O0*) или вовсе отключить её, поскольку при включенной оптимизации компилятор может переставить инструкции так, что при выполнении программа будет *скакать* по строкам исходного текста. Кроме этого, вместо ключа *-fomit-frame-pointer* лучше применять ключ *-fno-omit-frame-pointer*, который разрешает использование регистра указателя стека процессора.

В случае использования компилятора фирмы Intel не рекомендуется использовать ключ *-ax*, действие которого не позволяет отладчику GDB получить нужную информацию из стека программы. Некоторые компиляторы этой фирмы по умолчанию не добавляют в объектный файл программы расширенную отладочную информацию, поэтому компилятору требуется указать ключ *-debug all*. Компилятор GNU Фортран не включает информацию о модулях в исполняемый файл Фортран-программы, поэтому при отладке скомпилированной им программы вкладка *Fortran modules* в отладчике PD не отображается.

4. Шаблон задания

Отладчик PD позволяет выполнять запуск отладочных заданий в трех системах пакетной обработки заданий – СПО JAM, Open PBS/Torque и SLURM. Каждой системе соответствует отдельный файл с шаблоном задания. Ниже в качестве примера приведен файл с шаблоном задания для SLURM:

```
#!/bin/bash
# submitjob=sbatch
# canceljob=scancel :JOBID
# signaljob=sbatch --signal=:SIGNAL :JOBID
# regexpjob=\D+(\d+)
# showqueue=squeue
#SBATCH -U :COMMENT
#SBATCH -t :WALLTIME
#SBATCH -N :NODES --ntasks-per-node=:PPN
#SBATCH -o dbg.o%j -e dbg.e%j
#SBATCH -J debug
export :ENV
srun ${PD_HOME}/bin/agent :EXEC :ARGS
```

В верхней части примера находятся строки, содержащие служебные директивы (submitjob, canceljob, signaljob, regexpjob и showqueue), с помощью которых осуществляется передача

2 Разработчики GDB реализовали в версии 9.1 аналогичные MI-команды – -symbol-info-functions, -symbol-info-modules, -symbol-info-module-functions и -symbol-info-module-variables. Эти команды будут использованы в следующей версии отладчика PD.

задания SLURM, удаление и другие действия. Регулярное выражение \D+(\d+) позволяет из сообщения о постановке задания в очередь SLURM (например, *Submitted batch job 123456*) выделить идентификатор задания (123456). При взаимодействии с системой пакетной обработки заданий вместо служебных выражений :JOBID, :SIGNAL, :COMMENT, :NODES и т.д. отладчик PD подставит значения, которые пользователь укажет в графическом окне ввода атрибутов задания.

Утилита *srun* сначала запустит на вычислительных узлах ВС программные агенты отладчика PD, а они в свою очередь вызовут исполняемый файл программы, используя параметры - название исполняемого файла отлаживаемой программы (:EXEC) и её аргументы (:ARGS).

Если на ВС функционирует система пакетной обработки заданий, которая не поддерживается отладчиком PD, то администратор достаточно легко сможет сформировать для неё шаблон задания, пользуясь руководством администратора отладчика PD и редактором шаблона задания, который можно вызвать из графического окна отладчика.

5. Алгоритм отладчика

Для отладки параллельной программы на ВС пользователь должен с помощью графического интерфейса отладчика PD сформировать задание и передать его системе пакетной обработки заданий. Для этого в отладчике PD реализовано отдельное графическое окно, в котором пользователь должен указать название исполняемого файла программы, каталог с её исходными файлами, если файлы были перемещены в него после компиляции и сборки, входные параметры программы, количество требующихся для её выполнения вычислительных узлов и процессоров, переменные окружения и другие атрибуты.

После подтверждения ввода информации пользователем программа графического интерфейса считывает содержимое файла с шаблоном задания. Служебные выражения в шаблоне задания заменяются введенными пользователем значениями, в задание добавляется переменная окружения с IP-адресом и сетевым портом компьютера, на котором выполняется сервер команд и сообщений. Затем задание передаётся системе пакетной обработки заданий.

После старта задания на вычислительных узлах первыми запускаются программные агенты. Каждый программный агент создаёт псевдотерминалы для взаимодействия с отладчиком GDB и процессом программы, а также посылает на указанный IP-адрес сообщение о готовности к работе. Отладчик GDB запускает исполняемый файл программы.

Определив, что все программные агенты запущены, программа графического интерфейса с помощью MI-команд *-file-list-exec-source-files* и *-info-modules* получает от отладчика GDB информацию об исходных файлах программы, процедурах/функциях и модулях. Информация обрабатывается и отображается в графическом окне отладчика PD. Далее всем программным агентам посылается команда запуска исполняемого файла *-exec-run --start*. Программные агенты передают её отладчикам GDB. От них агенты принимают информацию о запуске и останове программы, которую пересылают серверу команд и сообщений. Сервер команд и сообщений преобразует её во внутреннее представление и передаёт программе графического интерфейса.

Сообщение с результатом выполнения MI-команды *-exec-run --start* содержит номер строки и название исходного файла программы. Они используются в графическом интерфейсе отладчика для отображения исходного текста и подсветки строки программы, на которой было приостановлено её выполнение. После появления исходного текста программы в графическом интерфейсе отладчика пользователь может расставлять точки прерывания и наблюдения, просматривать и изменять программные переменные, выполнять программу по шагам. Любое перечисленное действие порождает MI-команду, которая с помощью сервера команд и сообщений отсылается программным агентам, отладчикам GDB, а затем результаты её выполнения, пройдя программные компоненты в обратном порядке, передаются программе графического интерфейса.

Поясним вышесказанное на примере отладки трех процессов. Клик мышью по кнопке *Step over* в графическом окне отладчика вызовет обращение к подпрограмме-обработчику данного действия. В ней будет выполнено обращение к серверу команд и сообщений, который сформирует MI-команду *-exec-next*. Сервер пошлет её отладчикам GDB процессов 0-2 (для простоты понимания участие программных агентов опускаем):

```
Process 0 172.17.133.29 ->MI_COMMAND 37-exec-next
Process 1 172.17.133.30 ->MI_COMMAND 37-exec-next
Process 2 172.17.133.31 ->MI_COMMAND 37-exec-next
```

и получит от них ответы, что процессы выполняются:

```
Process 0 172.17.133.29 ->MI_OUTPUT 37^ running
Process 1 172.17.133.30 ->MI_OUTPUT 37^ running
Process 2 172.17.133.31 ->MI_OUTPUT 37^ running
```

Полученная сервером команд и сообщений информация будет передана программе графического интерфейса, которая в графическом окне изменит состояния процессов на *выполняется*. После завершения MI-команды *-exec-next* GDB пришлет серверу команд и сообщений информацию об останове процессов:

```
Process 0 172.17.133.29->MI_OUTPUT *stopped, reason="end-stepping-range",
frame={addr="0x000000000040065f", func="main", args=[{name="argc",
value="3"}], file="hello3.c", fullname="/home/test/hello3.c", line="48"},
thread-id="1", stopped-threads="all", core="1"
Process 1 172.17.133.30 ->MI_OUTPUT *stopped, reason="end-stepping-
range", frame={addr="0x000000000040065f", func="main",
args=[{name="argccc", value="3"}], file="hello3.c",
fullname="/home/test/hello3.c",line="48"}, thread-id="1", stopped-
threads="all", core="0"
Process 2 172.17.133.31 ->MI_OUTPUT *stopped, reason="end-stepping-
range", frame={addr="0x000000000040065f", func="main",
args=[{name="argccc",value="3"}], file="hello3.c",
fullname="/home/test/hello3.c", line="48"}, thread-id="1", stopped-
threads="all",core="1"
```

Сервер команд и сообщений преобразует текстовую информацию ответов в двоичную форму и передаст данные программе графического интерфейса, которая изменит состояния процессов на *остановлен*, отобразит в графическом окне содержимое файла /home/test/hello3.c и выделит цветом 48-ю строку. Кроме этого, для обновления информации о программных переменных, кадрах стека и потоках первому ответившему отладчику GDB будут посланы MI-команды *-thread-info*, *-thread-list-ids*, *-stack-list-locals*, *-stack-list-frames* и *-stack-list-arguments*. Остальным GDB-отладчикам перечисленные MI-команды посланы не будут, поскольку в графическом интерфейсе отладчика PD нет возможности отображать данные всех процессов одновременно. Для установки точек прерывания и наблюдения используются MI-команды *-break-insert* и *-break-watch*.

Перед завершением сессии отладки отладчик PD всегда записывает информацию об установленных точках наблюдения, прерывания, наблюдаемых переменных программы, настройках графического окна в файл с параметрами отладочной сессии, а затем удаляет задание из системы пакетной обработки заданий. При повторном запуске отладчика PD параметры отладочной сессии автоматически восстанавливаются.

6. Профилирование и обработка результатов

Профилирование отлаживаемой программы в отладчике PD осуществляется с помощью Google Performance Tools (GPT) и mpiP. GPT собирает информацию об эффективности

использования процессора и оперативной памяти, а профилировщик mpiP MPI[3]-метрики программы. Результаты профилирования записываются в файлы, которые по запросу пользователя обрабатывает программный агент и пересылает серверу команд и сообщений.

Ниже приведен фрагмент содержимого файла с информацией об использовании памяти и вкратце описан алгоритм его разбора программным агентом.

```
heap profile: 49675: 51865101 [ 49755: 85442155] @ heapprofile       (1)
6384: 8388608 [ 16384: 8388608] @ 0x00403d1b 0x00400e35              (2)
0x347641d994 0x00400cb9
128:  131072  [ 128:   131072 ] @ 0x004040f5 0x00400e35
0x347641d994 0x00400cb9
1:    69      [   1:   69     ] @ 0x347c09b801 0x347c09c305
0x347c09c4b2
MAPPED_LIBRARIES:                                                    (3)
00400000-00405000         r-xp 00000000 00:00 6611789
/home/test/hello3
3475400000-347541c000     r-xp 00000000 00:00 5658721
/lib64/ld-2.5.so
7fc9fc72f000-7fc9fc739000 r-xp 00000000 00:00 24259788
/lib/libunwind.so.8.0.1
```

Информация в файле логически разделена на три секции. В секции (1) находится заголовок, который содержит общее количество программных объектов, использующих память к моменту завершения программы, количество байтов занимаемой памяти, а также интегральные значения, относящиеся ко всем созданным программным объектам. Значения секции (1) используются для вычисления процентных соотношений.

В секции (2) находится информация об использовании памяти с детализацией по строкам исходного текста программы, в которых выделяется память. В левой части (2) находятся числа, в которых зафиксировано количество использующих память программных объектов (к моменту завершения программы), а также количество байтов используемой ими памяти в целом.

Секция (3) содержит информацию о библиотеках, которые использовались в процессе работы программы. В левой части находятся диапазоны адресов, которые связаны с программной библиотекой или исполняемым файлом программы, значения сдвига от начала файла и т.д. Правый столбец содержит названия файлов библиотек или исполняемых файлов.

Обработка файла с результатами профилирования начинается с первой строки секции (2), из которой программный агент считывает первое шестнадцатеричное число после амперсанда. В примере это 0x00403d1b. Число 0x00403d1b – это адрес вызова программной функции, который находится в одном из диапазонов адресов секции (3). По диапазону программный агент находит название файла, в котором следуют искать информацию. Так, значение 0x00403d1b находится в диапазоне адресов от 00400000 до 00405000, оно принадлежит адресному пространству файла /home/test/hello3. С помощью утилиты *addr2line*, учитывая адрес и название файла, программный агент определяет названия исходного файла и функции, а также номер строки исходного текста программы.

Если адрес соответствует динамической библиотеке, то из адреса секции (1) вычитаются адрес текстовой секции файла динамической библиотеки и смещение текстовой секции относительно начала файла, которые вычисляются с помощью утилиты *objdump*. Полученный адрес используется при определении названий функции, исходного файла и номера строки исходного текста программы.

По запросу пользователя программный агент может суммировать информацию, относящуюся к одной и той же исходной строке, функции или файлу.

[3] MPI (message passing interface) – программный интерфейс обмена данными между процессами параллельной программы.

7. Графический интерфейс пользователя

Отладчик PD позволяет отлаживать программу на ВС, на рабочем (локальном) компьютере пользователя, запускать и останавливать процессы программы, расставлять точки прерывания и наблюдения, просматривать и изменять программные переменные; он отображает состояния процессов и потоков, их стандартный вывод и диагностику.

С помощью графического интерфейса обеспечивается:

- отображение списка названий исходных файлов программы и содержащихся в них программных функций;
- логическое деление процессов программы на подмножества для их раздельной отладки;
- сравнение программных переменных в процессах и программных потоках;
- отображение значений элементов массивов, представление значений одно- и двумерных массивов в виде геометрической фигуры, запись значений массивов в файл;
- отображение результатов профилирования программы, запись результатов в файл;
- установка сочетания *горячих клавиш*, отображение исходного текста отлаживаемой программы в соответствии с пользовательскими настройками;
- поиск информации в исходных файлах программы;
- автоматическое сохранение и восстановление параметров отладочной сессии.

Графическое окно параллельного отладчика вызывается с помощью сценария интерпретатора shell *pdx*. Для создания отладочной сессии пользователь вызывает графическое окно ввода атрибутов отладочного задания или окно создания отладочной сессии на локальном компьютере. Например, если пользователь выбрал отладку программы на ВС, то в графическом окне кроме исполняемого файла и аргументов программы он должен указать количество вычислительных узлов и процессоров, время выполнения отладки и передать атрибуты задания системе пакетной обработки заданий. При успешном старте отладочной сессии в окне графического интерфейса появляется содержимое исходного файла программы, отображаются значения локальных переменных, состояния процессов, программных потоков и т.д.

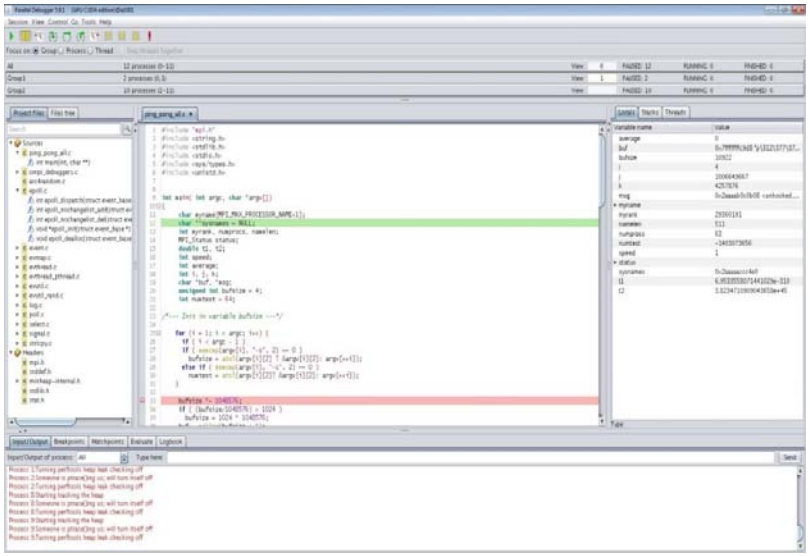

Рис. 2. Графическое окно отладчика PD
Fig. 2. PD debugger graphics window

7.1 Графическое окно отладчика

На рис. 2 показано графическое окно отладчика PD в начале отладки двенадцати процессов MPI-программы ping-pong (тест коммуникационной подсистемы ВС). Строка номер 12 подсвечена зеленым цветом, так как будет выполнена на следующем шаге. В строке 33 установлена точка прерывания: левее номера строки отображается соответствующая пиктограмма, а сама строка выделена красным цветом.

Слева от текста программы показан список исходных файлов и содержащихся в них функций.

Справа от него отображены значения локальных переменных функции main(). Внизу на вкладке *Input/Output* видны сообщения GPT – *Starting tracking the heap* и *Someone is ptrace()ing us; will turn itself off* с номерами процессов, от которых они были получены. На этой вкладке отображаются стандартный вывод и диагностика программы. На рис. 2 показано, что процессы программы разбиты на три подмножества - группы *All* (ранги[4] 0-11), *Group1* (ранги 0 и 1) и *Group2* (ранги 2-11). Группа *All* формируется отладчиком PD автоматически и содержит информацию обо всех отлаживаемых процессах программы. Группы *Group1* и *Group2* сформированы пользователем. *Group1* выбрана для отладки. Двенадцать процессов группы *All* остановлены, выполняющихся и завершившихся процессов нет. В группах *Group1* и *Group2* остановлены два и десять процессов соответственно.

При отладке группы процессов можно указать ранг процесса, информацию о переменных которого отладчик PD должен отображать в графическом окне. Так, в группе *All* указан шестой, а в *Group1* первый ранг.

7.2 Графическое окно просмотра многомерных массивов

Для просмотра значений элементов многомерных массивов реализовано графическое окно (рис. 3), в котором пользователь может отфильтровать значения, указав отображаемый диапазон, увидеть статистические данные – количество ±nan, ±inf, чисел меньше, больше или равных нулю и т.д.

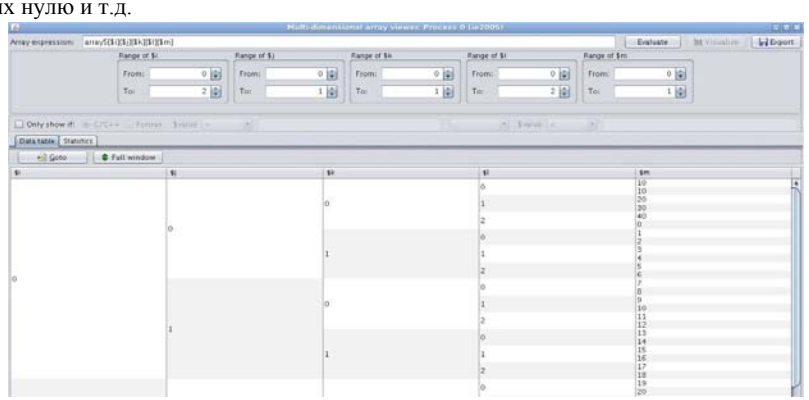

Рис. 3. Графическое окно просмотра значений многомерных массивов
Fig. 3. The graphical window for viewing values of multidimensional arrays

Кроме этого, пользователь имеет возможность записать информацию в файл в формате HDF5 (hierarchical data format – формат файла иерархической структуры) или CSV (comma-separated value – значения, разделенные запятыми). С помощью программного интерфейса OpenGL можно отобразить массив в форму плоской или объемной геометрической фигуры и оценить значения. Например, на рис. 4 изображена фигура, в которой можно визуально обнаружить точки, далеко выходящие за рамки некоторого диапазона.

[4] Ранг – номер MPI-процесса.

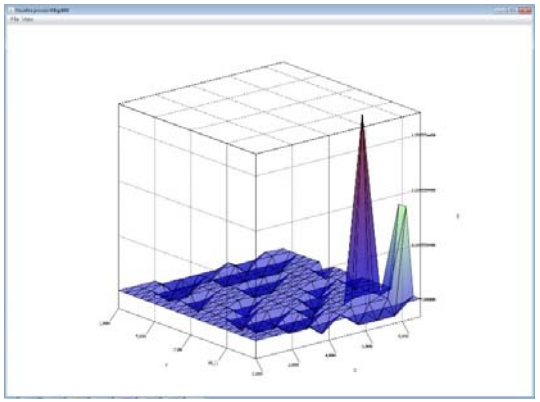

Рис. 4. Графическое представление значений многомерного массива
Fig. 4. A graphic representation of multidimensional array's values

7.3 Графическое окно просмотра результатов профилирования памяти

Результаты профилирования памяти отображаются в графическом окне, пример которого приведен на рис. 5. Значения, представленные в примере, связаны со строками исходного текста, поскольку выбран режим *Lines*. Последняя строка таблицы свидетельствует о том, что в тридцатой строке файла hello3.c было создано 24000 программных объектов, то есть 100% их общего количества, но к моменту завершения программы вся занятая ими память была освобождена, о чем свидетельствует 0 в столбце *In-use space*.

Рис. 5. Графическое окно с результатами профилирования программной памяти
Fig. 5. The graphic window with a profile of a program memory

8. Развитие отладчика PD

В процессе использования отладчика PD возникла необходимость доработать его программные компоненты. В результате в отладчике PD были реализованы коммуникационная схема *дерево*, позволившая качественно уменьшить время передачи информации, и новый программный компонент – не интерактивный отладчик. Кроме этого, была обеспечена отладка программ на графических ускорителях корпорации Nvidia.

8.1 Коммуникационная схема дерево

В ранних версиях отладчика PD была реализована схема, в которой каждый программный агент был непосредственно связан с сервером команд и сообщений. Такая схема проста в реализации и хорошо работает, пока с сервером взаимодействуют сотни программных агентов. Как только к серверу подключены тысячи агентов, она становится причиной общего замедления процесса отладки: значительная нагрузка на коммуникационную подсистему инструментального сервера, на котором запущена программа графического интерфейса отладчика PD, препятствует прохождению команд/сообщений к/от программных агентов.

Для распределения нагрузки по узлам ВС в отладчике PD была реализована коммуникационная схема *дерево*, в которой с программой графического интерфейса (верхняя

точка на рис. 6) связаны, например, три программных агента, которые в свою очередь соединены еще с четырьмя программными агентами и т.д. Новая коммуникационная схема позволила уменьшить время формирования соединений между программными агентами PD и сервером команд и сообщений в процессе создания отладочной сессии, а также время передачи информации в процессе отладки программы. В табл. 1 приведено время формирования коммуникационной схемы *дерево* в зависимости от количества программных агентов. В табл. 2 приведены замеры времени фактического выполнения строки программы параллельным отладчиком PD. Время включает передачу MI-команд *выполнить шаг* программным агентам, её обработку отладчиками GDB и передачу результатов серверу команд и сообщений.

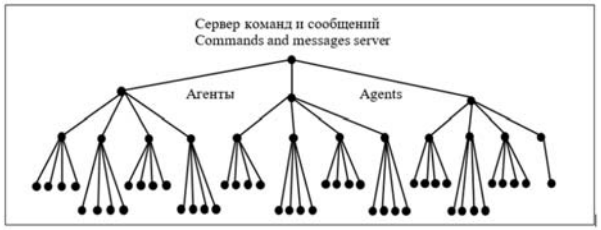

Рис. 6. Пример коммуникационной схемы дерево
Fig. 6. An example of a treelike communication scheme

Табл. 1. Время формирования коммуникационного дерева
Table 1. Formation time of a communication tree

Количество агентов (тысячи)	0,5	1	1,5	2,5	3	4,4	6	8	10
Время (миллисекунды)	19	42	74	119	141	206	343	417	564

Табл. 2. Общее время выполнения строки программы
Table 2. Total execution time of a program line

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

На время прохождения команды *выполнить шаг* большое влияние оказывает количество связей у программных агентов. Так, при исследовании этой метрики коммуникационного дерева было замечено, что вслед за увеличением количества связей с единицы до восьми время прохождения команды *выполнить шаг* уменьшается, но по мере дальнейшего роста числа связей оно начинает увеличиваться.

Увеличивая количество связей сервера команд с агентами, можно уменьшить длину пути, который проходят команды, например, при отладке небольшого количества процессов параллельной программы (идеально, если их агенты непосредственно связанны с сервером команд). Однако при отладке всех процессов программы суммарный путь, который пройдут все команды, не уменьшится. Кроме этого, каждая связь потребляет аппаратно-программные ресурсы компьютера, поэтому нет смысла делать значение этой метрики большим, в связи с чем было решено, что у сервера команд будет 32 связи. У программных агентов должно быть не более 8 связей[5]. Представленные в табл. 1 и 2 значения получены с использованием этих параметров коммуникационного дерева.

В целом реализация коммуникационной схемы *дерево* позволила качественно сократить общее время выполнения строки программы. Например, операция *выполнить шаг* при отладке четырех тысяч процессов выполняется более чем в 2,5 раза быстрее.

[5] Параметры коммуникационного дерева содержатся в текстовом конфигурационном файле отладчика PD. При необходимости их можно изменить.

Авторы полагают, что время передачи можно будет уменьшить, изменив формат команд и сообщений: формировать так называемые групповые команды, содержащие, например, команду и список агентов, которым она предназначается, передавать серверу ответ только от наблюдаемого процесса. Реализовав это, можно будет сократить количество информации, передаваемой между сервером команд и сообщений и программными агентами PD.

8.2 Неинтерактивная отладка программы

Новая возможность, реализованная в параллельном отладчике PD, - неинтерактивная (offline) отладка, позволяет отлаживать программу без участия пользователя. Такая отладка может быть применена тогда, когда требуется отладить ресурсоёмкую программу, для которой на ВС нет свободных вычислительных ресурсов, а в отсутствие разработчика они могут появиться.

Для offline-отладки пользователь должен сформировать в графическом интерфейсе отладчика PD пакетное задание, указав название исполняемого файла, количество узлов/процессоров, время выполнения и т.д. Кроме этого ввести, если требуется, 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. 7. Из этой таблицы пользователь может удалить строки, а также добавить информацию из файла.

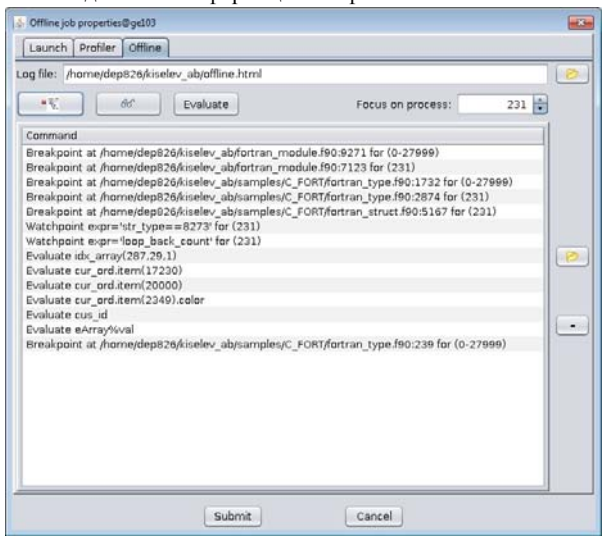

Рис. 7. Графическое окно ввода параметров offline-сессии отладки
Fig. 7. The graphical window for entering parameters of offline debugging session

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

В соответствии со схемой инициализации отладочного задания, которая подробно описана в разд. 5, агенты отладчика PD запускаются системой пакетной обработки заданий на узлах

ВС, между сервером команд и сообщений и программными агентами формируется коммуникационное *дерево*. Сервер команд и сообщений посылает агентам PD команду *-exec-run --start* (запустить программу и остановиться на точке входа в программу). Все программные агенты с помощью отладчика GDB выполняют её, а потом оповещают сервер 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 считывает значения локальных переменных процедуры/функции, значения указанных пользователем глобальных переменных, 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 этого процесса.

По завершении программы offline-отладчик записывает в журнальный файл результаты её профилирования и данные, считанные из временного файла стандартной выдачи и диагностики. После этого он завершается.

8.2.1 Формат журнального файла

Журнальный файл разбит на три части. Первая часть (Messages) содержит таблицу (табл. 3) с информацией о событиях, которые произошли в процессе отладки (старт программы, установка точек прерывания/наблюдения и т.д.).

При срабатывании точки прерывания или получении программного сигнала offline-отладчик записывает в таблицу информацию, содержащую название исходного файла и номер строки программы, в которой произошло прерывание. Рядом записываются данные о стеке (Stack), локальных переменных процедуры/функции (Locals), программных потоках (Threads), наблюдаемых глобальных переменных (Evaluate) и резюме (Summary), скрытые за символом ▼.

Табл. 3 Пример таблицы Messages
Table 3 A sample Messages *table*

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

Кликнув мышью по данному символу, можно раскрыть запись (в табл. 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 (табл. 4), которая содержит данные, получаемые с помощью профилировщиков GPT и mpiP. Если пользователь не расставил галочки на вкладке *Profiles* при создании offline-сессии, то секции Profiles в журнальном файле не будет.

Табл. 4. Пример информации использования процессора [12]
Table 4. An example of CPU-use information [12]

Function	Call	Call %	Full path
cycles	201813	97.2	/home/dep826/0226/tests/c/threads.c
doSomething	5898	2.8	/home/dep826/0226/tests/c/threads.c

Третья, последняя часть (Output) журнального файла, содержит информацию из стандартного вывода и диагностики процессов программы. Диагностическая информация выделяется красным цветом.

8.3 Поддержка CUDA

В отладчике PD была реализована возможность отлаживать программу на GPU (graphics processing unit). Собственно отладка программы на GPU выполняется проприетарным отладчиком GDB корпорации Nvidia (Nvidia-GDB), который входит в состав CUDA SDK [19]. Для формирования нестандартных MI-команд Nvidia-GDB, разбора сообщений и управления отладкой CUDA-программы были доработаны графический интерфейс отладчика PD и сервер команд и сообщений.

Надо отметить, что в составе CUDA SDK имеется отладчик nsight, который позволяет отлаживать программы, использующие графические ускорители Nvidia, но nsight не обеспечивает отладку MPI-программ на ВС. Кроме него, только отладчики TotalView и Allinea DDT обеспечивают отладку MPI- и CUDA-программ, но они большинству отечественных программистов недоступны.

8.4 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 сообщения – ответы. Однако в момент выполнения программы в GPU Nvidia-GDB выдаёт специализированную информацию, например:

```
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 в GPU с индексом 0, индекс блока (0,0,0) и 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 т.д.

Для выполнения переключений и отображения информации о GPU в графическом интерфейсе отладчика PD значения метрик device, warp, kernel, blockIdx и threadIdx записываются в характеристику процесса, а команды с префиксом -cuda посылаются Nvidia-GDB до тех пор, пока программный код использует GPU. Рассмотрим этапы переключения отладчика PD на программную нить с индексом (53,0,0). Графический интерфейс PD посылает Nvidia-GDB MI-команду

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

Ответ отладчика Nvidia-GDB с результатом её выполнения может, например, содержать 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/CUDA/bitreverse.cu", line="12"
```

Соответствующий программный класс сервера команд и сообщений обрабатывает информацию об успешном переключении фокуса, а затем графический интерфейс отладчика PD выполняет смену индексов в переключателе (рис. 8).

Рис. 8 Переключатель индексов ядра, блока и нити при отладке программы на GPU
Fig. 8 The switch for changing of a kernel, a block and a thread indexes at the debugging on GPU

8.5 Графические компоненты для отладки программы на GPU

8.5.1 Переключатель ядра, блока и программной нити

Переключатель (рис. 8) позволяет переключаться на ядро, блок или нить в режиме отладки программных потоков. Он появляется в графическом интерфейсе отладчика PD только тогда, когда код программы выполняется в GPU. Чтобы пользователь мог использовать доступные на момент отладки индексы, при каждом останове программы отладчик PD считывает из GPU граничные значения индексов Kernel, Block и Thread, а затем переконфигурирует переключатель. Операция переключения на ядро, блок и нить выполняется после клика мышью по кнопке Ok.

В случае завершения отлаживаемой нити отладчик автоматически переключается на *живую* нить.

8.5.2 Информация о программных нитях

При выполнении кода программы в GPU в графическом интерфейсе отладчика PD на закладке *Thread* появляется таблица *CUDA Thread*, содержащая список GPU-нитей (рис. 9).

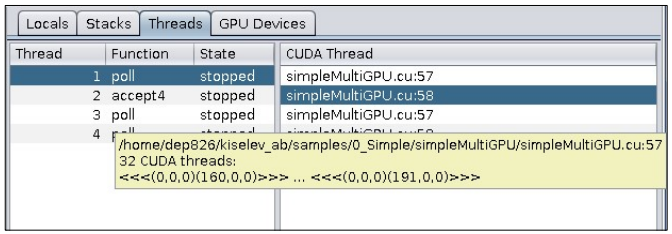

Рис.9. Пример таблицы CUDA Thread *и всплывающей подсказки*
Fig. 9. An example of the CUDA Thread *table and a tooltip*

Если пользователю важен диапазон индексов blockIdx и threadIdx, то достаточно некоторое время удерживать указатель мыши над таблицей *CUDA Thread*, чтобы появилась всплывающая подсказка с интересующей его информацией. Подсказка содержит полный путь к исходному файлу, номер исходной строки (цифра 57), общее количество программных нитей (32), а также их начальный и конечный индексы (строка с символами меньше и больше).

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

8.5.3 Информация о GPU

На вкладке *GPU Devices* (рис. 10) отображается сводная информация о доступных/используемых GPU-устройствах. Так, например, на рисунке показано, что процессам с рангами 0-180 доступны по два устройства с указанными характеристиками, остальные процессы не используют GPU, об этом свидетельствует фраза *No device*.

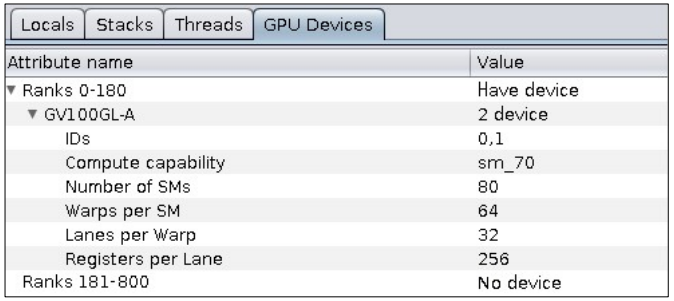

Рис. 10. Пример содержимого таблицы GPU Devices
Fig. 10. An example of contents of the GPU Devices *table*

9. Заключение

Параллельный отладчик применяется в РФЯЦ-ВНИИЭФ для отладки программных комплексов моделирования физических процессов. Он входит в дистрибутивы защищенных ОС *системное программное обеспечение супер-ЭВМ со встроенными средствами защиты информации от несанкционированного доступа* [20] и *Арамид* [21] и в составе этих ОС используется на вычислительных системах ФГУП РФЯЦ-ВНИИЭФ.

По своим возможностям отладчик PD очень близок к Allinea DDT и TotalView, но в отличие от них у него нет лицензионных ограничений: PD позволяет отлаживать и профилировать любое количество процессов параллельной программы на любом количестве процессоров.

Отладчик PD написан на *чистой* Яве, поэтому по сравнению, например, с Eclipse PTP он использует гораздо меньше оперативной и дисковой памяти компьютера. Кроме этого, обладая достаточно большим опытом использования Eclipse PTP, авторы статьи уверенно заявляют, что пользовательский интерфейс отладчика PD более удобный.

Отладчик PD функционирует не только на компьютерах фирмы Intel, но и на отечественной платформе *Эльбрус*.

Последняя версия отладчика PD имеет качественные отличия от предыдущих версий за счет реализации древовидной схемы передачи информации, в ней появился новый программный компонент – offline-отладчик, обеспечена отладка CUDA-программ.

Список литературы / References

[1]. TotalView. URL: https://totalview.io/free-trial, 12.03.2020.
[2]. Distributed Debugging Tool. URL: https://www.arm.com/products/development-tools/server-and-hpc/forge/ddt, 12.03.2020.
[3]. Eclipse Parallel Tools Platform. URL: http://eclipse.org/ptp, 12.03.2020.
[4]. Федоров В.К., Киселев С.Н. Отладчик параллельных приложений (PDB). Вопросы атомной науки и техники. Серия «Математическое моделирование физических процессов», вып. 3, 2013 г., стр. 65-71. // Fedorov V.K, Kiselev S.N. A debugger of parallel applications (PDB). Voprosy Atomnoy Nauki i Tekhniki. Series «Mathematical modelling of physical processes», issue 3, 2013, pp.65-71 (in Russian).
[5]. Malyshkin V. E, Romanenko A.A. GEPARD – General Parallel Debugger for MVS-1000/M. Lecture Notes in Computer Science, vol. 2763, 2003, pp. 519-523.
[6]. Андрианов А.Н., Базаров С.Б., Бугеря А.Б., Колударов П.И., Набоко И.М. Применение отладчика параллельных программ при решении задачи о фокусировке ударных и взрывных волн на

многопроцессорных ЭВМ. Препринты ИПМ им. М. В. Келдыша, № 50, 2004 г., 15 стр. / Andrianov A.N., Bazarov S.B., Bugerya A.B., Koludarov P.I., Naboko I.M. Application of parallel programs debugger for multiCPU modeling of shock and blast waves focusing. Keldysh Institute preprints, № 50, 2004, 15 p. (in Russian).

[7]. Киселев А.Б., Киселев С.Н., Семенов Г.П. Отладчик параллельных программ для кластеров на базе ОС Linux. Вопросы атомной науки и техники. Серия «Математическое моделирование физических процессов», 2018 г., вып. 2, стр. 72-80. // Kiselev A.B., Kiselev S.N. A parallel debugger for clusters on the base of OS Linux. Voprosy Atomnoy Nauki i Tekhniki. Series «Mathematical modelling of physical processes», issue 2, 2018, pp. 72-80 (in Russian).

[8]. GDB. URL: http://www.gnu.org/software/gdb, 12.03.2020.

[9]. Torque. URL: http://www.adaptivecomputing.com/products/open-source/torque, 12.03.2020.

[10]. SLURM. URL: https://slurm.schedmd.com/documentation.html, 12.03.2020.

[11]. Киселев А.Б, Киселев С.Н. Система пакетной обработки заданий JAM. Вопросы атомной науки и техники. Серия «Математическое моделирование физических процессов», 2009 г., вып. 4, стр. 60-66. / Kiselev A.B., Kiselev S.N. JAM batch jobs system. Voprosy Atomnoy Nauki i Tekhniki. Series «Mathematical modelling of physical processes», issue 4, 2009, pp. 60-66 (in Russian).

[12]. Google Performance Tools. URL: https://github.com/gperftools/gperftools, 12.03.2020.

[13]. mpiP: Lightweight, Scalable MPI Profiling. URL: http://mpip.sourceforge.net, 12.03.2020.

[14]. SwingX. URL: https://github.com/arotenberg/swingx, 12.03.2020.

[15]. JHDF5. URL: http://www.hdfgroup.org, 12.03.2020.

[16]. Jzy3D. URL: http://www.jzy3d.org, 12.03.2020.

[17]. RSyntaxTextArea. URL: http://www.sorceforge.org/rsyntaxtextarea, 12.03.2020.

[18]. OpenGL. URL: https://www.opengl.org/sdk/libs/, 12.03.2020.

[19]. CUDA Toolkit Documentation. URL: http://docs.nvidia.com/cuda/eula/index.html, 12.03.2020.

[20]. Кульнев Д.В., Модянов Р.В., Петрик А.Н. Защищенная ОС. Открытые системы. СУБД, № 4, 2015 г. / Kulnev D.V., Modjanov R. V, Petrik A.N. Secured OS. Open systems. DBMS, № 4. 2015 (in Russian).

[21]. Петрик А.Н. Защищенная операционная система Арамид для супер-ЭВМ. Сборник тезисов докладов Национального суперкомпьютерного форума (НСКФ-2019), 2919 г. / Petrik A.N. Secured OS Aramid for the super-computer. In Proc. of the National Supercomputer Forum (NSCF-2019), 2019 (in Russian).

Информация об авторах / Information about authors

Алексей Борисович КИСЕЛЕВ, кандидат физико-математических наук, начальник лаборатории разработки системного программного обеспечения. Сфера научных интересов: системное программное обеспечение высокопроизводительных вычислительных систем, облачные вычисления, методы защиты информации.

Aleksey Borisovich KISELEV, Ph.D in physics and mathematics, head of the laboratory for system software development. Research interests: system software for high-performance computing systems, cloud computing, information security methods.

Сергей Николаевич КИСЕЛЕВ, старший научный сотрудник. Сфера научных интересов: системное программное обеспечение высокопроизводительных вычислительных систем, облачные вычисления, методы защиты информации в распределенных вычислительных системах.

Sergey Nikolaevich KISELEV, senior researcher. His research interests include system software for high-performance computing systems, cloud computing, methods of information's protection in distributed computing systems.