УДК 004.4'233

ОТЛАДЧИК ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ДЛЯ КЛАСТЕРОВ НА БАЗЕ ОС LINUX

А. Б. Киселёв, С. Н. Киселёв, Г. П. Семёнов
(ФГУП "РФЯЦ-ВНИИЭФ", г. Саров Нижегородской области)

Дается описание отладчика параллельных программ, написанных на языках программирования Си, Си++ или Фортран и выполняемых на вычислительных системах с распределенной памятью. Описаны программные компоненты отладчика, схема их взаимодействия. Представлены возможности графического интерфейса пользователя, приведена схема выполнения встроенного в отладчик профилирования.

Ключевые слова: многопроцессорная вычислительная система, интерактивная отладка, отладка параллельной программы.

Введение

Появление многопроцессорных вычислительных систем (МВС), высокая сложность разрабатываемых для них параллельных программ стали предпосылками для создания отладчиков параллельных программ. Для отладки на МВС программисты могут использовать как коммерческие, так и свободно распространяемые отладчики. Коммерческие отладчики — это TotalView [1], Allinea DDT (Distributed Debugging Tool) [2], PGDBG Graphical Symbolic Debugger [3]. 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 [4], который за несколько лет развития приобрел необходимый набор возможностей для отладки параллельной программы. Отечественные программные средства отладки — отладчик параллельных программ PDB [5], разработанный и использовавшийся в РФЯЦ-ВНИИЭФ, GEPARD [6] (СО РАН), диалоговый отладчик программ, написанных на непроцедурном языке НОРМА [7] (ИПМ им. М. В. Келдыша РАН), судя по отсутствию новых исходных кодов и публикаций, не развиваются.

Описанный в данной статье отладчик PD (Parallel Debugger) заполняет свободное место в ряду средств отладки параллельных программ, разрабатываемых для отечественных МВС. 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 MS Visual Studio, Eclipse, IDEA и Allinea DDT. В настоящее время отладчик PD 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 используются свободно распространяемые программные компоненты SwingX [14], JHDF5 [15], Jzy3D [16], RSyntaxTextArea [17] и OpenGL [18].

Программные компоненты отладчика

Отладчик PD состоит из программы графического интерфейса пользователя, сервера сообщений и агента. В качестве базового отладчика используется GDB [8]. Программные компоненты и схема их взаимодействия показаны на рис. 1.

Графический интерфейс и сервер сообщений выполняются в разных потоках одной программы. Вариант отладки на МВС подразумевает, что они запускаются пользователем на инструментальном сервере МВС. Сервер сообщений посылает программным агентам MI-команды (команды машинно-ориентированного интерфейса GDB) [8], а программные агенты, в свою очередь, пересылают их отладчикам GDB. Информацию о результатах выполнения команд отладчики GDB записывают в стандартный вывод, из которого программные агенты ее считывают и пересылают серверу сообщений. Кроме того, программные агенты контролируют стандартный ввод и диагностику процессов программы.

В ходе выполнения программы отладчики GDB формируют асинхронные сообщения, которые не связаны с MI-командами, 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 не мог ее пропустить.

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

Программа графического интерфейса пользователя и сервер сообщений. Управление отладкой осуществляется посредством программы графического интерфейса. Она обеспечивает отображение списка названий исходных файлов, значений переменных программы, стандартной выдачи и диагностики процессов, их состояний, загрузку и сохранение параметров сессии — кодировки, шрифта и его размера, точек останова и наблюдения и т. д.

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

Программный агент. Программный агент реализован для передачи MI-команд отладчикам GDB, сообщений с результатами их выполнения, стандартной выдачи и диагностики отлаживаемого процесса. Его взаимодействие с отладчиками GDB осуществляется посредством псевдотерми-

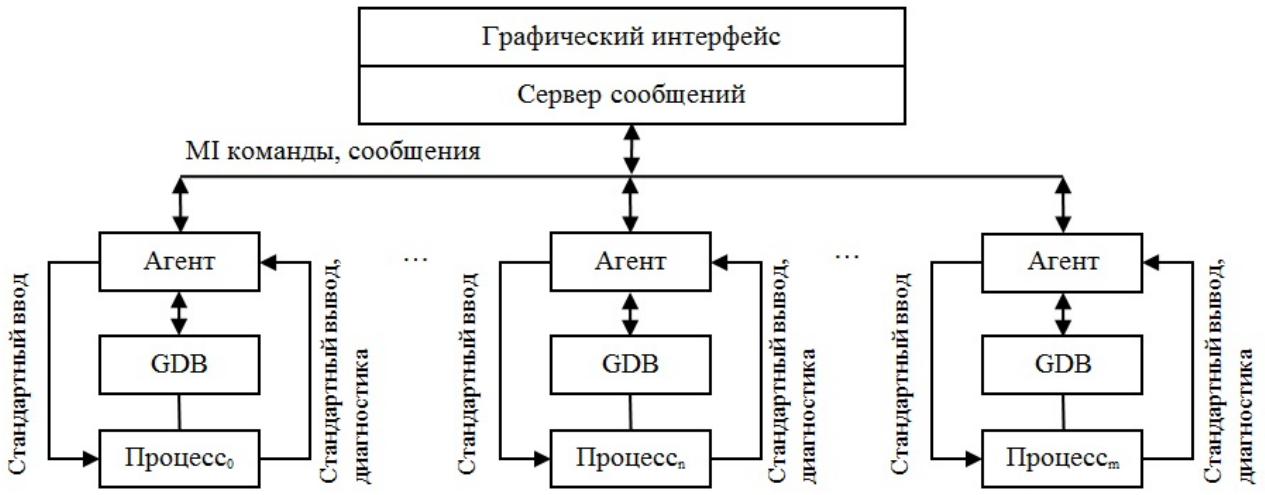

Рис. 1. Схема отладчика PD

налов*. Процесс отлаживаемой программы также использует псевдотерминалы для ввода/вывода информации.

Важной функцией программного агента является обработка результатов профилирования программы.

Базовый отладчик GDB. Отладчик GDB обеспечивает отладку процессов программы. Управление отладчиком GDB осуществляется посредством команд его машинно-ориентированного интерфейса (MI).

В отладчике PD может использоваться одна из модифицированных авторами статьи версий отладчика GDB — 7.11.1 или 7.12.1. Внесенные в GDB исправления позволили повысить надежность его функционирования, благодаря им пользователь обеспечен информацией о модулях Фортран-программы, функциях и процедурах, типах указателей на массивы и многом другом. Для отладки может быть использована версия отладчика GDB без модификаций, но в этом случае перечисленные виды информации отображаться не будут.

Опции компилятора

Для отладки программы ее необходимо скомпилировать с ключом `-g` и лучше использовать минимальный уровень оптимизации `(-O0)` или вовсе обойтись без нее, поскольку при включенной оптимизации компилятор может переставить инструкции так, что при выполнении программа будет "скакать" по строкам исходного текста. Кроме того, вместо ключа `-fomit-frame-pointer` лучше применять `-fno-omit-frame-pointer`, который разрешает использование регистра указателя стека процессора. В случае 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 для компилятора требуется указать ключ `-debug all`. Компилятор GNU Фортран не включает информацию о модулях в исполняемый файл Фортран-программы, поэтому при отладке скомпилированной им программы в отладчике PD вкладка *Fortran modules* не отображается.

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. В качестве примера приведем 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 -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
```

*Псевдотерминал — эмулятор терминала в ОС Unix/Linux (PTY), псевдоустройство, используемое для взаимодействия пользователя с локальным или удаленным компьютером.

В верхней части примера находятся строки, содержащие служебные директивы, с помощью которых осуществляется постановка задания в очередь, удаление и другие действия (`submitjob, canceljob, signaljob, regexpjob` и `showqueue`). Регулярное выражение `\D+(\d+)` позволяет из сообщения о постановке задания в очередь (например, *Submitted batch job 123456*) выделить идентификатор задания (123456). Вместо служебных выражений `:JOBID, :SIGNAL, :COMMENT, :NODES` и т. д. будут подставлены значения, которые укажет пользователь в графическом окне ввода атрибутов задания.

Утилита `srun` запустит на вычислительном узле МВС программный агент отладчика PD, а название исполняемого файла отлаживаемой программы и ее аргументы будут переданы в него как параметры.

Если на МВС пользователя функционирует система пакетной обработки заданий, которая неизвестна отладчику PD, администратор отладчика PD может достаточно легко сформировать собственный шаблон задания.

Алгоритм отладчика

Для отладки параллельной программы на МВС 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 указывает путь к каталогу, где находятся исходные файлы параллельной программы, название ее исполняемого файла, входные параметры программы, количество требуемых для ее 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 директивы в шаблоне задания заменяются введенными пользователем значениями, в задание добавляется переменная окружения с IP-адресом и сетевым портом компьютера, на котором выполняется сервер сообщений. Задание передается системе пакетной обработки заданий.

После старта задания на вычислительных узлах первыми запускаются программные агенты. Каждый программный агент создает 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 MI-команда запуска исполняемого файла `-exec-run --start`. Программные агенты передают данную команду отладчикам GDB. От них агенты принимают информацию о запуске и останове программы, которую передают серверу сообщений. Сервер сообщений преобразует ее во внутреннее представление и передает 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 ее выполнение. После появления исходного текста программы в графическом интерфейсе отладчика пользователь может расставлять точки прерывания и наблюдения, просматривать и изменять переменные программы, выполнять программу по шагам. Любое перечисленное действие порождает MI-команду, которая с помощью сервера сообщений отсылается программным агентам, отладчикам GDB, а затем результаты ее 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 сообщений, который сформирует MI-команду `-exec-next`. Сервер сообщений пошлет ее отладчикам GDB процессов 0—2 (для простоты 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 сообщений информация будет передана программе графического интерфейса, которая в графическом окне изменит состояния процессов 0—2 на *выполняется*. После выполнения MI-команды `-exec-next` GDB пришлет серверу 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 сообщений преобразует текстовую информацию ответов в двоичную форму и передает данные программе графического интерфейса, которая изменяет состояния процессов на *остановлен*, отображает в графическом окне содержимое файла /home/test/hello3.c и выделяет цветом 48-ю строку. Кроме того, для обновления информации о программных переменных, кадрах стека и потоках первому ответившему отладчику GDB посылаются MI-команды `-thread-info`, `-thread-list-ids`, `-stack-list-locals`, `stack-list-frames` и `-stack-list-arguments`. Остальным GDB-отладчикам перечисленные MI-команды не посылаются, поскольку в графическом интерфейсе 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 сессии отладки, а затем удаляет задание. При повторном запуске отладчика параметры отладочной сессии восстанавливаются автоматически.

Профилирование и обработка результатов

Профилирование отлаживаемой программы в отладчике PD может осуществляться с помощью Google Performance Tools (GPT) и mpiP. Профилировщик GPT собирает информацию об эффективности использования процессора и оперативной памяти, а mpiP — MPI-метрики программы. Результаты профилирования записываются в файлы, которые по запросу пользователя обрабатывает программный агент.

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)

16384:  8388608 [ 16384:  8388608] @ 0x00403d1b 0x00400e35 0x347641d994 0x00400cb9
128:  131072 [ 128:  131072 ] @ 0x004040f5 0x00400e35 0x347641d994 0x00400cb9        (2)
1:  69 [ 1:  69 ] @ 0x347c09b801 0x347c09c305 0x347c09c4b2
```

```
MAPPED_LIBRARIES:
00400000-00405000 r-xp 00000000 00:00 6611789              /home/test/hello3
3475400000-347541c000 r-xp 00000000 00:00 5658721          /lib64/ld-2.5.so          (3)
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) содержатся числа, означающие 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 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 следует искать информацию. Значение 0x00403d1b находится в диапазоне адресов от 00 400 000 до 00 405 000, оно принадлежит адресному пространству файла /home/test/hello3. С помощью утилиты `addr2line`, учитывая адрес и название файла, программный агент определяет названия выполняемой функции и исходного файла, 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.

Графический интерфейс пользователя

Отладчик PD позволяет создавать и завершать сессию отладки программы на вычислительных узлах МВС, а также 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 обеспечиваются:

- отображение списка названий исходных файлов программы и содержащихся в них программных функций;
- логическое деление процессов программы на отдельные множества для их раздельной отладки;
- сравнение программных переменных в процессах и программных потоках;
- отображение значений элементов массивов как в цифровом, так и в графическом виде, запись массивов в файлы;
- отображение результатов профилирования программы, запись результатов в файлы;
- установку сочетания "горячих клавиш", отображение исходного текста отлаживаемой программы в соответствии с пользовательскими настройками;
- автоматическое сохранение и восстановление параметров отладочной сессии.

Графическое окно отладчика. Окно графического интерфейса параллельного отладчика вызывается с помощью сценария интерпретатора `shell pdx`. Для создания отладочной сессии пользователь вызывает *окно ввода атрибутов* отладочного задания или окно создания отладочной сессии на локальном компьютере. Например, если пользователь выбрал отладку программы на МВС, 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, значения локальных переменных, состояния процессов, потоков и т. д.

На рис. 2 (см. также цветную вкладку) показано графическое окно отладчика PD в начале отладки четырех (0—3) процессов MPI программы ping-pong (тест коммуникационной подсистемы). Строка 12 подсвечена зеленым цветом, она будет выполнена на следующем шаге. Левее номера строки 33 находится пиктограмма, а строка выделена красным цветом, так как связана с точкой прерывания.

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.

Из рис. 2 видно, что процессы программы разбиты на два подмножества — группы *All* и *User defined*0. Группа *All* формируется отладчиком PD автоматически и содержит информацию обо всех отлаживаемых процессах программы. Четыре процесса группы *All* остановлены, выполняемых и завершенных процессов нет. При отладке группы процессов можно указать ранг процесса, информацию о переменных которого отладчик PD должен отображать в графическом окне. Так, в группе *All* указан второй процесс.

Группа *User defined*0 сформирована пользователем, в нее входит процесс с рангом 0, эта группа для отладки не выбрана.

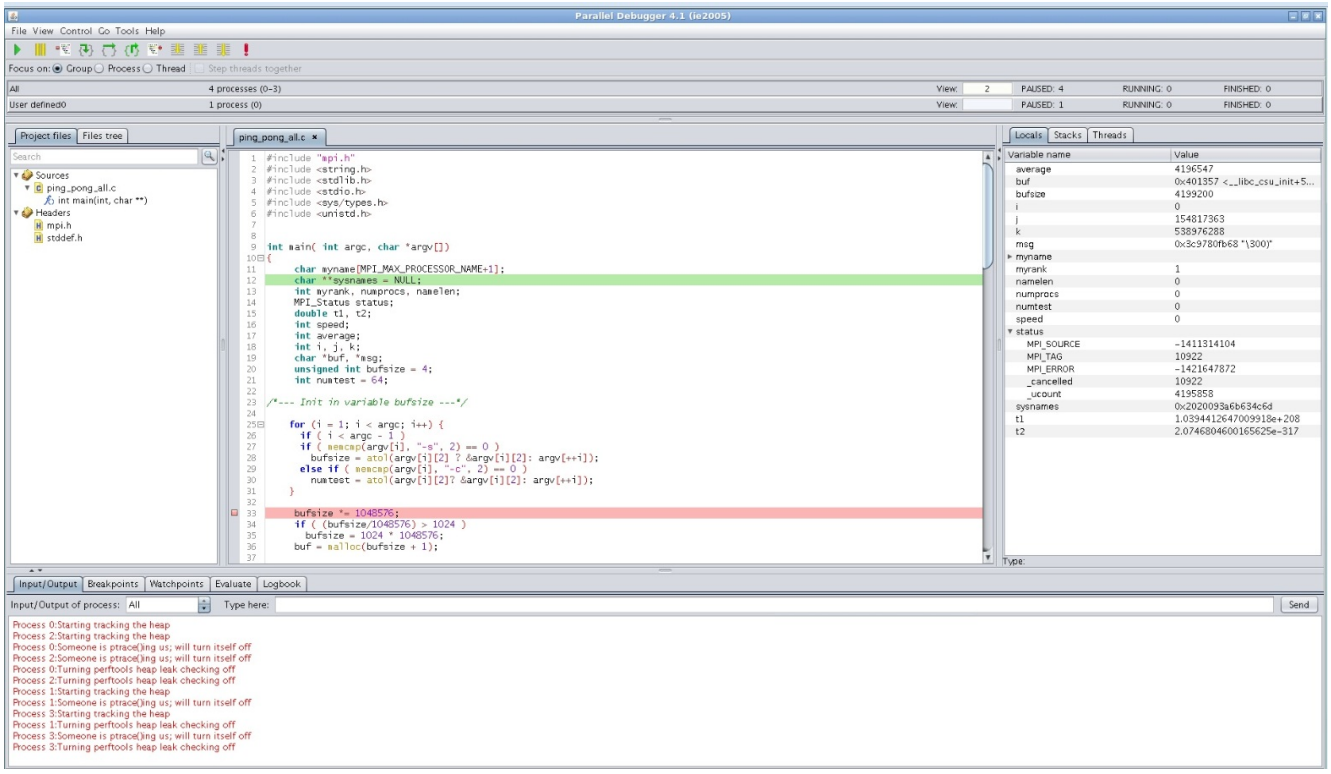

Рис. 2. Графическое окно отладчика PD

Графическое окно просмотра многомерных массивов. Для просмотра значений элементов многомерных массивов реализовано окно (рис. 3), в котором пользователь может фильтровать значения, увидеть статистические данные — количество значений типов ±`nan`, ±`inf`, чисел меньших, больших или равных нулю и т. д.

Кроме того, 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). Данное окно позволяет представить значения одно- и двумерных массивов в виде трехмерного изображения, которое формируется с помощью программной библиотеки OpenGL.

Графическое окно просмотра результатов профилирования памяти. Результаты профилирования памяти отображаются в окне, пример которого приведен на рис. 4. Значения, представленные в примере, связаны со строками исходного текста, поскольку выбран режим *Lines*. Послед-

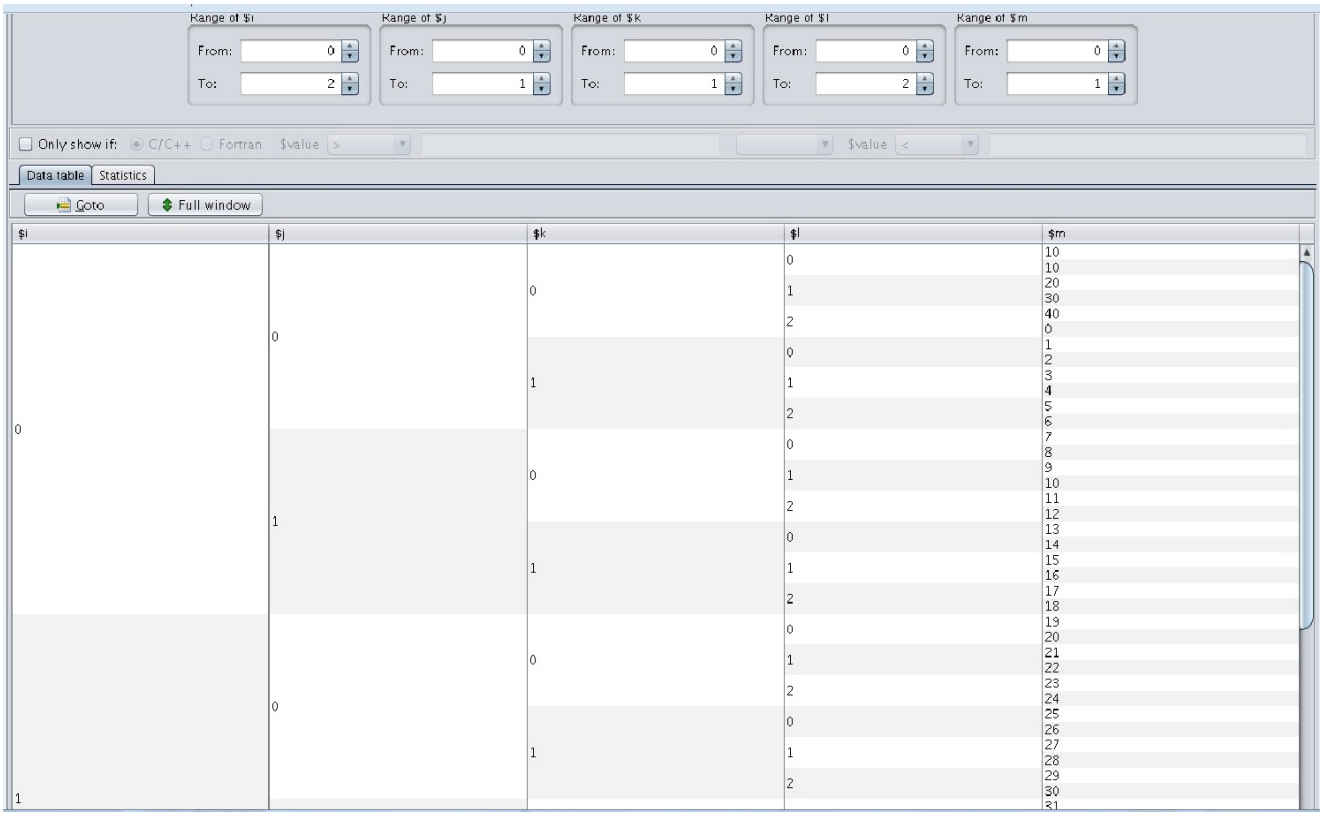

Рис. 3. Графическое окно просмотра многомерных массивов

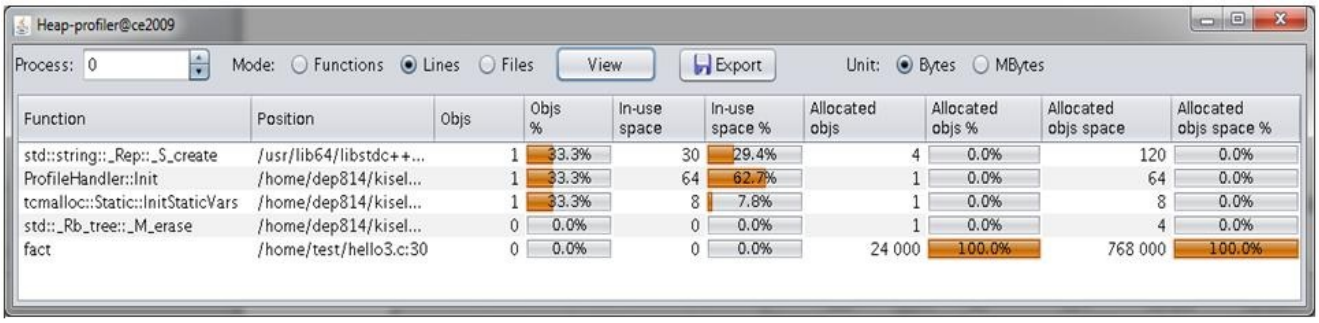

Рис. 4. Графическое окно просмотра результатов профилирования памяти

няя строка таблицы свидетельствует о том, что в тридцатой строке файла hello3.c было создано 24 000 программных объектов, т. е. 100 % их общего количества, но к моменту завершения программы вся занятая ими память была освобождена, о чем свидетельствует 0 в столбце *In-use space*.

Заключение

По своим возможностям отладчик PD очень близок к TotalView и Allinea DDT, но в отличие от них у него нет лицензионных ограничений: он позволяет отлаживать и профилировать одновременно любое количество параллельных программ без учета числа используемых ими процессоров.

Параллельный отладчик успешно применяется в РФЯЦ-ВНИИЭФ для отладки программных комплексов моделирования физических процессов.

Отладчик PD входит в дистрибутив системного программного обеспечения суперЭВМ со встроенными средствами защиты информации от несанкционированного доступа.

Список литературы

1. TotalView. http://www.totalviewtech.com.
2. The Distributed Debugging Tool. http://www.allinea.com.
3. PGDBG Graphical Symbolic Debugger. http://www.pgroup.com.
4. Eclipse Parallel Tools Platform. http://eclipse.org/ptp.
5. *Фёдоров В. К., Киселёв С. Н.* Отладчик параллельных приложений (PDB) // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2013. Вып. 3. С. 65—71.
6. *Малышкин В. Э., Романенко А. А.* Отладчик параллельных программ для мультикомпьютера // Автометрия. 2003. Т. 39, № 3. С. 109—114.
7. *Бугеря А. Б., Колударов П. И., Кулешова М. И.* Диалоговый интерфейс для отладки параллельных программ // Научный сервис в сети Интернет: Тр. Всерос. науч. конф. М.: Изд-во МГУ, 2006. С. 86—88.
8. GDB. http://www.gnu.org/software/gdb.
9. Open PBS/Torque. http://www.adaptivecomputing.com/products/open-source/torque.
10. SLURM. http://www.llnl.gov/linux/slurm.
11. *Киселёв А. Б, Киселёв С. Н.* Система пакетной обработки заданий JAM // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2009. Вып. 4. С. 60—66.
12. Google Performance Tools. http://goog-perftools.sourceforge.net.
13. Профилировщик mpiP. http://mpip.sourceforge.net.
14. SwingX. http://www.jdesktop.org/swingx.
15. JHDF5. http://www.hdfgroup.org.
16. Jzy3D. http://www.jzy3d.org.
17. RSyntaxTextArea. http://www.sorceforge.org/rsyntaxtextarea.
18. OpenGL. http://www.opengl.org.

Статья поступила в редакцию 24.10.17.

A PARALLEL DEBUGGER FOR CLUSTERS ON THE BASE OF OS LINUX / A. B. Kiselev, S. N. Kiselev, G. P. Semenov (FSUE "RFNC-VNIIEF", Sarov, N. Novgorod region).

The paper presents a debugger for parallel programs in C/C++, or FORTRAN, which are executed on distributed-memory 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.

Keywords: multiprocessor computing system, interactive debugging, parallel program debugging.