Параллельный ввод-вывод и контрольные точки в DVM-системе

В.Ф. Алексахин[1], В.А. Бахтин[1,2], О.Ф. Жукова[1], В.А. Крюков[1,2], М.Ю. Кузнецов[1], М.Н. Притула[1], О.А. Савицкая[1]

1 ИПМ им. М.В. Келдыша РАН
2 МГУ им. М.В. Ломоносова

Аннотация. DVM-система предназначена для разработки параллельных программ научно-технических расчетов на языках C-DVMH и Fortran-DVMH. Эти языки используют единую модель параллельного программирования (DVMH-модель) и являются расширением стандартных языков Си и Фортран спецификациями параллелизма, оформленными в виде директив компилятору. DVMH-модель позволяет создавать эффективные параллельные программы для гибридных вычислительных кластеров. В статье представлены новые возможности для работы с контрольными точками, которые основаны на использовании параллельного ввода-вывода, реализованного в DVM-системе.

Ключевые слова: автоматизация разработки параллельных программ, DVM-система, ускоритель, ГПУ, Фортран, Си, MPI, OpenMP, OpenACC, DVMH, ввод-вывод, контрольная точка

Parallel I/O and control points in DVM system

V.F. Aleksakhin[1],V.A. Bakhtin[1,2], O.F. Zhukova[1], V.A. Krukov[1,2], M.U. Kuznetsov[1], M.N. Pritula[1], O.A. Savitskaya[1]

1 Keldysh Institute of Applied Mathematics
2 Lomonosov Moscow State University

Abstract. DVM-system is designed for the development of parallel programs of scientific and technical calculations in C-DVMH and Fortran-DVMH languages. These languages use a single parallel programming model (DVMH model) and are extensions of the standard C and Fortran languages with parallelism specifications, written in the form of directives to the compiler. The DVMH model makes it possible to create efficient parallel programs for heterogeneous computing clusters. The article presents new features of DVM system for working with control points, which are based on the use of parallel I/O.

Keywords: automation of development of parallel programs, DVM-system, accelerator, GPU, Fortran, C, MPI, OpenMP, OpenACC, DVMH, I/O, control point

1. Введение

Множество задач, решаемых сегодня на современных вычислительных системах, требуют размещения большого объема данных на внешней памяти (дисках) и интенсивного обмена данными между дисками и оперативной памятью. Функционально эти обмены можно разделить на следующие группы:

1. Явные операторы ввода-вывода, которые необходимы для ввода начальных данных и вывода окончательных результатов.
2. Контрольные точки. Для задач с большим временем выполнения необходимо периодически запоминать состояние задачи. Если выполнение задачи прервано по каким-либо причинам, то ее выполнение можно продолжить с последнего запомненного состояния.
3. Временные шаги. В некоторых задачах требуется запоминать текущее состояние массивов через определенные промежутки времени. Например, эти данные могут использоваться в подсистемах визуализации.
4. Вычисления с массивами на внешней памяти. Если некоторые массивы не помещаются в оперативной памяти (рабочие внешние массивы), то процесс выполнения может быть организован следующим образом:
   - Внешние массивы размещаются на файлах (дисках).
   - Для каждого внешнего массива в оперативной памяти распределяется буфер (или несколько буферов).
   - Использование и модификация внешнего массива осуществляется порциями, равными размеру буфера. Очередная порция считывается в буфер, обрабатывается и записывается на диск (если необходимо).

Разные группы операций требуют использования различных подходов к их реализации. Например, для контрольных точек прежде всего необходимо сократить время чтения/записи контрольной точки, а количество файлов, формат этих файлов отходят на второй план. При записи данных для их последующей визуализации, наоборот, в первую очередь необходимо сохранить данные в требуемом формате. Для вычислений с массивами на внешней памяти можно реализовать режим, в котором очередная порция данных из файла будет читаться во время использования ранее прочитанной порции данных, что позволит существенно сократить время выполнения программы.

Стремление к достижению максимальной производительности ввода-вывода и отсутствие общего широко используемого интерфейса параллельного ввода-вывода привело к тому, что разработчики новых языков программирования, например, UPC (Unified Parallel C), Chapel, XcalableMP вынуждены разрабатывать собственные средства для параллельного ввода-вывода. Такие средства были реализованы и в DVM-системе.

Следует отметить, что в последнее время контрольные точки все чаще стали использоваться прикладными программистами:

- при отладке параллельных программ (например, для обнаружения ошибки, которая происходит после нескольких часов счета программы, можно сохранить контрольную точку перед возникновением ошибки и многократно стартовать программу с данной точки под контролем отладчика);
- при запуске программы на суперкомпьютере с системой очередей, если максимальный квант времени, выделяемый системой очередей, меньше времени, необходимого для расчета задачи;
- для разработки надежных, отказоустойчивых программ.

Последняя проблема становится все более актуальной. Современные суперкомпьютеры состоят из десятков тысяч узлов, и каждый год это число увеличивается[1]. Одновременно с этим повышается и вероятность выхода из строя отдельных частей системы или даже всей системы полностью[2]. Это требует искать новые подходы к реализации контрольных точек.

В данной статье представлены возможности DVM-системы[3], которые могут быть востребованы при разработке отказоустойчивых параллельных программ с интенсивным вводом-выводом. Статья построена следующим образом. В главе 2 рассматриваются различные режимы параллельного ввода-вывода, реализованные для C-DVMH[4] программ. В главе 3 представлены директивы для организации параллельного ввода-вывода в языке Fortran-DVMH[5]. В главе 4 описаны новые возможности DVM-системы для работы с контрольными точками. В главе 5 показан пример использования данных возможностей, выполнен анализ эффективности реализованной подсистемы ввода-вывода.

2. Параллельный ввод-вывод в языке C-DVMH

DVM-система поддерживает различные режимы ввода-вывода:

1. Последовательный синхронный ввод-вывод.
2. Параллельный синхронный ввод-вывод:
   a. в локальный файл;
   b. в параллельный файл.
3. Асинхронный параллельный ввод-вывод.

В языке C-DVMH в полном соответствии со стандартом C99 реализованы следующие операции ввода-вывода:
remove, rename, tmpfile, tmpnam, fclose, fflush, fopen, freopen, setbuf, setvbuf, fgetc, fgets, fputc, fputs, getc, getchar, gets, putc, putchar, puts, ungetc, fread, fwrite, fgetpos, fseek, fsetpos, ftell, rewind, clearerr, feof, ferror.

Остановимся на некоторых деталях реализации различных режимов ввода-вывода.

Последовательный синхронный ввод-вывод выполняется следующим образом:

1. Файл открывается только одним процессом из текущей многопроцессорной системы (процессом ввода-вывода).
2. Все операции над файлом производит только процесс ввода-вывода.
3. При вводе/выводе распределенного массива чтение или запись делается порциями около 100МБ, после которой делается рассылка или сбор данных на процессы (с процессов) ими владеющие (владеющих).
4. Если операция предполагает запись пользовательских переменных или возврат значения, то они рассылаются процессом ввода-вывода.
5. Если в процессе выполнения операции возникла ошибка и стандарт предписывает установку errno, то его значение также рассылается процессом ввода-вывода.
6. Все операции над файлом допустимо выполнять только коллективно всеми процессами той многопроцессорной системы, которой был создан этот файл.
7. Операции не над файловыми дескрипторами (rename, remove, tmpnam) выполняются процессом ввода-вывода текущей многопроцессорной системы с рассылкой результатов выполнения на остальные процессы.

При реализации **параллельного синхронного ввода-вывода** были добавлены 2 новых режима для функций fopen и freopen: локальный файл и параллельный файл.

Локальный файл открывается каждым процессом независимо. Все операции над локальными файлами обрабатываются каждым процессом независимо и никаких коммуникаций не вызывают.

При чтении (или записи) распределенного массива из **локального файла**, каждый процесс читает (или записывает) только локальную часть распределенного массива. Современные параллельные файловые системы ориентированы на эффективную работу с большим количеством файлов. Таким образом, работа с локальными файлами является достаточно удобным средством параллельного ввода-вывода распределенных данных, выполняется эффективно, но требует совпадения распределения данных при записи и при последующем чтении. DVMH-программа[6], которая работает с локальными файлами, должна запускаться на той же процессорной решетке, что и DVMH-программа, которая записала эти данные.

Основная идея работы с **параллельными файлами** заключается в следующем. Пусть у нас есть файл размером 4 гигабайта, в котором хранится двухмерный распределенный массив. Пусть DVMH-программа была запущена на 4-х процессах. В этом случае каждый процесс отвечает за чтение ¼ части файла. Такое чтение может выполняться параллельно. 1-ый процесс читает 1-ый Гбайт файла, 2-ой процесс читает 2-ой Гбайт файла и.т.д. Прочитав свою часть файла, процесс может разослать эти данные остальным процессам. Если массив в программе распределен по столбцам (на каждом процессоре находится

некоторое подмножество столбцов распределенного массива), то процесс оставит ¼ часть прочитанной информации себе, а ¾ части прочитанных данных разошлет остальным процессам(«чужие» столбцы), а если массив распределен по строкам, то в этом случае никакая рассылка данных не требуется, т.к. каждый процесс будет читать только «свои» строки массива.

Таким образом, параллельный файл открывается каждым процессом многопроцессорной системы. Все операции над параллельными файлами выполняются так же, как и над обычными файлами, за исключением операций fread, fwrite, fputs. Рассмотрим алгоритм реализации данных операций:

1. Синхронизировать на файловую систему все операции с процесса ввода-вывода (т.к. все остальные операции выполняет только он). Для синхронизации содержимого файла в памяти с содержимым на диске используется системный вызов _commit в Windows или fdatasync в Linux.
2. Передать позицию в файле от процесса ввода-вывода всем остальным процессам текущей многопроцессорной системы.
3. Каждый процесс устанавливает позицию в файле в ту точку, с которой он будет читать (или записывать).
4. Каждый процесс выполняет свою часть работы.
    a. Если запрошена операция чтения в размноженную переменную, то после чтения части производится объединение прочитанных участков (операция типа Allgather).
    b. Если запрошена операция с распределенным массивом, то каждый процесс выполняет сбор нужных ему данных со всех других процессов перед записью либо выполняет рассылку после чтения своей части файла (операция типа Alltoall).
5. Все процессы синхронизируют свои изменения на файловую систему.
6. Процесс ввода-вывода устанавливает позицию за последним участком чтения (или записи).

Такой режим сохраняет содержимое файла таким же, как если бы он записывался последовательной программой. А также позволяет читать файлы, записанные программой, работавшей на любом количестве процессов (в т.ч. исходной последовательной).

Для реализации **асинхронного ввода-вывода** в каждом процессе создаются дополнительные нити ввода-вывода. Все операции над одним файлом выполняются последовательно одной нитью. Операции над разными файлами могут выполняться параллельно разными нитями. Асинхронный режим ввода-вывода основан на использовании дополнительной памяти. Во время записи распределенного массива в файл, создается копия этого массива в оперативной памяти центрального процессора, создается задача по записи массива-копии в файл, одна из нитей ввода-вывода начинает выполнять эту задачу, после этого осуществляется возврат управления в программу пользователя. Далее ввод-вывод и вычисления могут выполняться

одновременно – программа устанавливает новые значения распределенного массива, а в файл пишутся сохраненные значения из массива-копии. Такое одновременное выполнение возможно лишь для функций ввода-вывода, которые не возвращают значения и не устанавливают errno (переменную, которая хранит целочисленный код последней ошибки). Если в программе используется результат операции ввода-вывода, например, количество элементов, которые были успешно записаны в файл, то это приводит к ожиданию завершения операции.

Следует отметить, что лишь незначительная часть стандартных функций ввода-вывода не возвращают значения и не устанавливают errno. Это функции rewind и clearerr. Но в большинстве пользовательских программ эти значения не используются. Например, вместо оператора n_elem = fwrite(B, sizeof(double), L*L, f) используется оператор fwrite(B, sizeof(double), L*L, f).

Для реализации асинхронного ввода-вывода в системе поддержки были заведены дополнительные варианты функций, которые не возвращают значения: fprintf, fscanf, printf, scanf, vfprintf, vfscanf, vprintf, vscanf, fgets, fputc, fputs, gets, putc, putchar, puts, ungetc, fread, fwrite, fseek. Также они не устанавливают errno, а значит результатом их работы является только лишь или запись переданных в функцию данных в файл, или чтение этих данных из файла. Также была введена оптимизация в компиляторе C-DVMH, которая для этого набора функций распознает: используется ли их возвращаемое значение и в случае если значение не используется, то генерируется вызов невозвращающего значения варианта реализации.

Для реализации асинхронного ввода-вывода в компиляторе C-DVMH был добавлен новый режим для фунций fopen и freopen: асинхронный файл.

Асинхронными могут быть как обычные файлы (нелокальные и непараллельные), так и локальные или параллельные (одновременно локальным и параллельным файл быть не может).

Как уже отмечалось, операции над разными файлами могут перекрываться (выполняться разными нитями ввода-вывода). Любая недопускающая асинхронного выполнения операция приводит к ожиданию всех асинхронных операций над соответствующим файлом перед началом выполнения синхронной операции.

Также некоторые операции (те, что требуют коммуникаций) не могут быть асинхронными, если реализация MPI не обеспечивает параллельной работы с ней (требуется режим MPI_THREAD_MULTIPLE).

Для сериализации асинхронных операций над каждым файлом используется механизм зависимых задач и параллельного выполнения графа задач.

Таким образом, DVM-система поддерживает несколько различных режимов ввода-вывода. Режим ввода-вывода задается при открытии файла. Для этого у функций fopen и freopen был расширен набор режимов открытия файла.

При добавлении литеры "l"или "L" к режиму открытия файла, каждый процессор открывает свой локальный файл и все операции производятся каждым процессором независимо от других. При открытии в этом режиме в имени файла можно использовать конструкцию "%d" для задания различных имен различными процессорами.

Следующий оператор:

FILE *f = fopen("out_%04d.txt", "wl");

приведет к открытию N файлов на запись (где N – кол-во процессоров в текущей многопроцессорной системе) с именами out_0000.txt, out_0001.txt, out_0002.txt, out_0003.txt, ..

Удаление всей группы файлов может быть осуществлено при помощи специальной функции:

int dvmh_remove_local(const char *filename);

При добавлении литеры "p" или "P" к режиму открытия файла все процессоры осуществляют параллельный ввод/вывод в глобальный файл.

Добавление литеры "s" или "S" включает режим асинхронного ввода-вывода.

В таблице 1 указаны переменные окружения, которые влияют на эффективность выполнения ввода-вывода.

Таблица 1. Переменные окружения, влияющие на выполнение ввода-вывода

DVMH_PARALLEL_IO_THRES	положительно целое число, задающее минимальный размер вводимых или выводимых данных (в байтах), который РТС будет пытаться вводить или выводить параллельно. По умолчанию значение равно104857600, т.е. 100 Мбайт.
DVMH_IO_BUF_SIZE	положительное целое число, задающее максимальный размер буфера ввода/вывода при его выполнении через процессор ввода/вывода. Значение по умолчанию равно104857600, т.е. 100 Мбайт.
DVMH_IO_THREAD_COUNT	неотрицательное целое число, задающее кол-во нитей ввода/вывода, выполняющих асинхронные операции ввода/вывода. Указание нуля приведет к отключению асинхронного ввода/вывода. Значение по умолчанию равно 5.

3. Параллельный ввод-вывод в языке Fortran-DVMH

До реализации подсистемы параллельного ввода-вывода для C-DVMH программ, ввод-вывод в программах на языке Fortran-DVMH осуществлялся последовательно и синхронно. Все операции с файлом выполнял специальный процесс ввода-вывода, который осуществлял необходимую рассылку прочитанных данных и сбор необходимой для записи информации со всех процессов. Для организации ввода-вывода использовались операторы стандарта Фортран 95.

Для ввода-вывода распределённых массивов в языке Fortran-DVMH существовали серьезные ограничения:

- Список ввода-вывода должен был состоять только из одного имени распределённого массива и не мог содержать других объектов ввода-вывода.
- В операторах ввода-вывода по формату допускался только формат, задаваемый '*'.
- Список управляющей информации не должен был содержать параметры ERR, END и IOSTAT.
- В списке управляющей информации допускалось использование только размноженных переменных.
- Не разрешалось использовать операторы ввода-вывода распределённых массивов в параллельном цикле.

Следует отметить, что программа на языке Fortran-DVMH, выполняющая бесформатный ввод-вывод распределенных массивов, в общем случае была не совместима с последовательной программой на Фортране 95. Данные, которые были записаны одной программой, не могли быть прочитаны другой программой, вследствие разницы длин записей. Например, последовательная программа могла записать весь массив при помощи одной операции, а параллельная программа накапливала данные в специальном буфере ввода-вывода, который мог сбрасываться в файл несколько раз (по мере накопления информации с различных процессов).

Учитывая данную особенность, переход в компиляторе Fortran-DVMH на использование средств параллельного ввода-вывода, реализованных для C-DVMH программ, не внес никаких дополнительных проблем.

В системе поддержки выполнения DVMH-программ были созданы новые функции, которые являются своего рода переходниками между операторами ввода-вывода языка Фортран и функциями ввода-вывода для языка C-DVMH. Разработана новая версия компилятора с языка Fortran-DVMH, которая использует данные функции-переходники.

Для задания режима выполнения ввода-вывода в язык Fortran-DVMH введена новая директива:

```
!DVM$ IO_MODE ([PARALLEL]
```

[[,]LOCAL]
[[,]ASYNC])

Данная директива может быть указана перед оператором открытия файла и управляет выполнением всех последующих операций ввода-вывода в этот файл (unit). Если данная директива не указана перед оператором открытия файла, то ввод-вывод осуществляется по старой схеме (через процессор ввода-вывода). Спецификации LOCAL, PARALLEL и ASYNC соответствуют режимам, которые были рассмотрены в предыдущей главе.

На рис. 1 представлен фрагмент программы, в котором показаны новые возможности языка Fortran-DVMH: задание режима ввода-вывода; работа с секциями распределенных массивов; использование нескольких распределенных массивов в одном операторе ввода-вывода; поддержка управляющих параметров типа ERR, END, которые задают оператор в программе, на который необходимо перейти в случае возникновения ошибки или достижения конца файла.

```
          PARAMETER (L=1000)
          FLOAT A(L,L), B(L,L)
!DVM$     DISTRIBUTE ( BLOCK, BLOCK) :: A
!DVM$     ALIGN B(I,J) WITH A(I,J)
          …
!DVM$     IO_MODE (LOCAL,ASYNC)
          OPEN(4, ACCESS='STREAM', FILE = 'DATA.DAT', ERR=44)
          …
          WRITE(4) A(2:L-1,2:L-1),B
          …
          CLOSE(4)
          …
44:       PRINT *, 'ERROR HAPPENED! PROGRAM TERMINATES'
          STOP
```

Рис. 1. Ввод-вывод в прогрмме на языке Fortran-DVMH

4. Механизм работы с контрольными точками

Описанные в предыдущих разделах режимы параллельного ввода-вывода позволяют реализовать эффективную работу с контрольными точками в DVMH-программах. Например, использование локальных файлов позволяет каждому процессу быстро сохранить/прочитать свою часть распределенного массива; использование асинхронного ввода-вывода позволяет писать контрольную точку одновременно с выполнением вычислительных операторов программы.

Однако использование данных операторов ввода-вывода требует серьезного усложнения текста исходной программы, добавления множества новых операторов при работе с контрольными точками:

- проверить наличие файла с данными;
- проверить режим, в котором файл был записан;

- для локального файла проверить число процессоров и процессорную решетку, которая была использована при записи файла;
- проверить корректность прочитанных данных. Например, совпадение размера массива, который был сохранен, и размера массива, который был создан после перезапуска программы;
- реализовать механизм чередования файлов. Например, писать данные по очереди в файлы file1.dat и file2.dat, чтобы в случае сбоя при сохранении очередной контрольной точки, хотя бы один из файлов имел корректное значение;
- реализовать совмещение записи контрольной точки и выполнения программы;
- и многое другое.

Программист вынужден повторять описанную выше логику работы с контрольными точками в каждой DVMH-программе. Чтобы упростить работу с контрольными точками, в DVM-системе были разработаны новые спецификации параллелизма.

Директива определения контрольной точки:

!DVM$ CP_CREATE cp-name, VARLIST(cp-var-list), FILES(filenames) [, mode]

cp-name ::= string-expr
cp-var ::= subarray | variable
filenames ::= filename-list | filename-array
filename ::= string-expr
filename-array ::= string-array
mode ::= PARALLEL | LOCAL

В директиве указывается имя контрольной точки (cp_name), в аргументе VARLIST – данные для сохранения и чтения, в аргументе FILENAMES – файлы, которые будут использоваться, а также режим использования файлов (mode). Режим может быть локальным или параллельным. Асинхронность контрольной точки указывается непосредственно в самом операторе сохранения. Если режим не указан, то по умолчанию используется параллельный.

Пример использования директивы CP_CREATE:

```
INTEGER I
INTEGER, DIMENSION(1:N) ARR
!DVM$ DISTRIBUTE ARR(BLOCK)
!DVM$ CP_CREATE CP1, VARLIST(I, ARR), FILES('file1.dat', 'file2.dat'), LOCAL
```

В данном примере объявлена контрольная точка с именем CP1. Для неё заданы файлы, переменные и локальный режим открытия.

Для **сохранения контрольной точки** используется директива CP_SAVE, в которой достаточно указать имя контрольной точки, а также опционально указать асинхронность выполнения данной операции.

Синтаксис директивы имеет следующий вид:

```
!DVM$ CP_SAVE cp-name [, ASYNC]
```

Например, при выполнении инструкции:

```
!DVM$ CP_SAVE CP1, ASYNC
```

и предположении, что контрольная точка CP1 была определена, как указано выше, в файл file1.dat или file2.dat будут записаны скалярная переменная I и распределённый массив ARR. Файл будет выбран в зависимости от того, в какой из них последний раз была выполнена успешная запись. Режим открытия файлов – локальный, поэтому в него будет записана только локальная часть массива ARR.

Для **загрузки контрольной точки** используется директива CP_LOAD. В ней указывается только имя контрольной точки. Синтаксис директивы следующий:

```
!DVM$ CP_LOAD cp-name
```

Например, при выполнении инструкции:

```
!DVM$ CP_LOAD CP1
```

и предположении, что контрольная точка CP1 была определена, как указано ранее, из файла file1.dat, либо из файла file2.dat будут загружены скалярная переменная I и локальная часть массива ARR. Последний корректно записанный файл будет выбран автоматически.

Для **ожидания окончания сохранения асинхронной контрольной точки** используется директива CP_WAIT:

```
!DVM$ CP_WAIT cp-name, STATUS(status-var)
status-var ::= int-variable
```

Например, при выполнении инструкции:

```
!DVM$ CP_WAIT CP1, STATUS(st)
```

и предположении, что контрольная точка CP1 была определена, как указано выше, программа дожидается окончания асинхронной записи в файлы file1.dat и file2.dat, если они была, и закроет их. В переменную st, переданную в

качестве аргумента параметра STATUS будет записано одно из следующих значений: 0 – асинхронная запись успешно завершилась; отличное от нуля значение, если произошла какая-то ошибка.

Разработана экспериментальная версия DVM-системы, которая поддерживает данные спецификации для работы с контрольными точками.

5. Апробация подхода

Для исследования эффективности реализованной подсистемы ввода-вывода, а также механизма работы с контрольными точками, было разработано несколько тестовых программ. Одна из них, реализующая итерационный алгоритм Якоби, представлена на рис. 2. В данном тесте моделируется запись контрольной точки (распределенного массива B), которая выполняется каждые 10 итераций.

```
      PROGRAM JAC2D
      PARAMETER (L=32000, ITMAX=100)
      REAL A(L, L), B(L, L), EPS
      INTEGER ST
!     arrays A and B  with block distribution
!DVM$ DISTRIBUTE(BLOCK, BLOCK) :: A
!DVM$ ALIGN B(I, J) WITH A(I, J)
!DVM$ CP_CREATE CP1,VARLIST(B),FILES('file1.dat','file2.dat'),LOCAL
      MAXEPS = 0.5
      DO IT = 1, ITMAX
       EPS = 0.
       IF (MOD(IT,10) .EQ. 0) THEN
!DVM$    CP_SAVE CP1, ASYNC
         CONTINUE
       ENDIF
!      variable EPS is used for calculation of maximum value
!DVM$  PARALLEL(J, I) ON A(I, J), REDUCTION(MAX(EPS))
       DO J = 2, L - 1
          DO I = 2, L - 1
            EPS = MAX(EPS, ABS(B(I, J) - A(I, J)))
            A(I, J) = B(I, J)
          ENDDO
       ENDDO
!      Copying shadow elements of array A from
!      neighbouring processors before loop execution
!DVM$  PARALLEL(J, I) ON B(I, J), SHADOW_RENEW(A)
       DO J = 2, L - 1
         DO I = 2, L - 1
           B(I, J) = (A(I, J-1) + A(I-1, J) + A(I+1, J) + A(I, J+1)) / 4.
         ENDDO
       ENDDO
       PRINT 200, IT, EPS
200    FORMAT (' IT = ', I4, '   EPS = ', E14.7)
      ENDDO
!DVM$ CP_WAIT CP1, STATUS(ST)
      END
```

Рис. 2. Фрагмент итерационного метода Якоби на языке Fortran-DVMH

На рис. 3 показаны времена выполнения программы Якоби на 1 узле суперкомпьютера "Ломоносов" при использовании различных режимов ввода-вывода. При проведении данного эксперимента на узле запускались 2 MPI-

процесса по 6 нитей, тестировались параллельный (PARALLEL), локальный (LOCAL) и асинхронный/локальный режимы (ASYNCHRONOUS) сохранения контрольной точки. Для сравнения дано время сохранения массива, используя старую схему через процессор ввода-вывода (OLD).

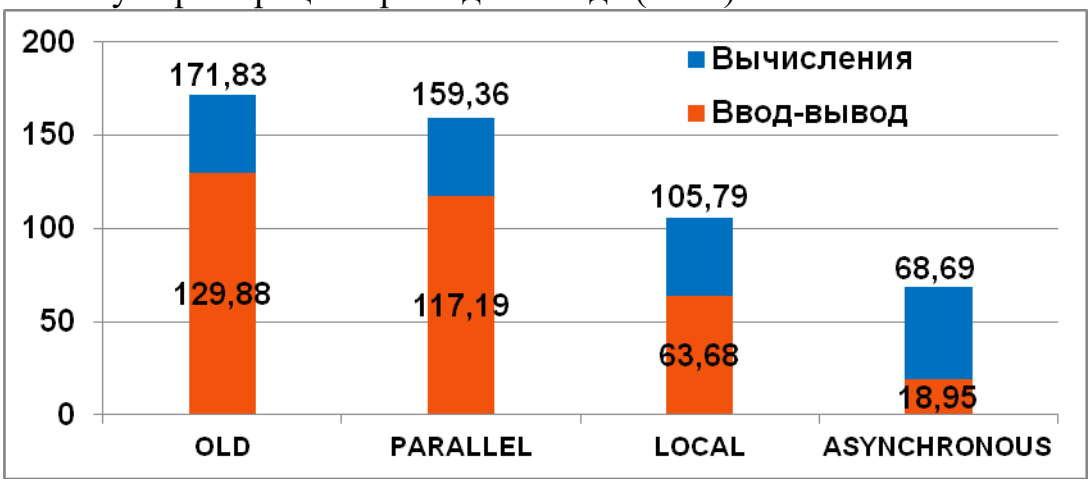

Рис. 3. Времена выполнения программы Якоби на суперкомпьютере “Ломоносов”

Использование локального режима для данного эксперимента позволило почти в 2 раза сократить время сохранения контрольной точки (в этом случае каждый процесс пишет свою часть массива в файл) по сравнению с последовательным режимом (в этом случае один процесс ввода-вывода сохраняет весь массив в файл, плюс требуется время для пересылки части массива от одного процесса процессу ввода-вывода). Применение асинхронного режима сокращает почти в 7 раз время выполнения ввода-вывода со 128,99 секунд до 18,95 секунд за счет совмещения выполнения ввода-вывода и вычислений (за время выполнения очередных 10 итераций алгоритма, предыдущая контрольная точка успевает записаться в файл). Все асинхронные записи буферизуются. Для данного эксперимента время выполнения 10 операций буферизации массива B (копирования память-память) составляет почти 18 секунд.

Заключение

Неэффективное управление вводом-выводом сводит на нет любые оптимизации параллельного выполнения задач с интенсивным вводом-выводом. Представленные в статье новые возможности позволяют повысить эффективность выполнения ввода-вывода для DVMH-программ.

Одно из преимуществ разработанных средств параллельного ввода-вывода – это простота использования. В своей программе прикладной программист использует привычные ему операторы ввода-вывода последовательного языка программирования (Си или Фортран), переключение между режимами осуществляется изменением всего одного параметра функций fopen и freopen (для языка C-DVMH) или при помощи директивы IO_MODE (для языка Fortran-DVMH).

Разработанные средства ввода-вывода являются универсальными – могут использоваться и для сохранения контрольных точек, и для сохранения данных, полученных на очередном временном шаге алгоритма (например, для визуализации), и для вычислений с массивами на внешней памяти.

В настоящее время ведется разработка новой версии системы SAPFOR[7,8], которая автоматизирует процесс разработки параллельных программ в DVMH-модели. В данной системе будет реализован режим автоматической расстановки контрольных точек. Система автоматически будет определять переменные, которые необходимо сохранить/восстановить в данной точке и генерировать необходимые директивы.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проекты 19-07-00889-а, 19-07-01101-а и 20-01-00631-а.

Литература

1. TOP500 supercomputers — URL: https://www.top500.org/ .
2. Бондаренко А.А., Якобовский М.В. Обеспечение отказоустойчивости высокопроизводительных вычислений с помощью локальных контрольных точек // Вестн. ЮУрГУ. Сер. Выч. матем. информ., 2014, том 3, выпуск 3. — С. 20-36 .
3. Система автоматизации разработки параллельных программ (DVM-система). — URL: http://dvm-system.org .
4. Язык C-DVMH. C-DVMH компилятор. Компиляция, выполнение и отладка CDVMH-программ. — URL: http://dvm-system.org/static_data/docs/CDVMH-reference-ru.pdf .
5. Язык Fortran-DVMH. Fortran-DVMH компилятор. Компиляция, выполнение и отладка DVMH-программ. — URL: http://dvm-system.org/static_data/docs/FDVMH-user-guide-ru.pdf .
6. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами // Вестник Южно-Уральского государственного университета, серия "Математическое моделирование и программирование", №18 (277), выпуск 12. — Челябинск: Издательский центр ЮУрГУ, 2012. — С. 82-92 .
7. Клинов М.С., Крюков В.А. Автоматическое распараллеливание Фортран-программ. Отображение на кластер // Вестник Нижегородского университета им. Н.И. Лобачевского, № 2, 2009. — С. 128–134 .
8. Катаев Н.А., Колганов А.С., Смирнов А.А. Поддержка интерактивности в системе САПФОР // Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции (18-23 сентября 2017 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2017. — С. 243-249. — URL: http://keldysh.ru/abrau/2017/57.pdf doi:10.20948/abrau-2017-57 .

References

1. TOP500 supercomputers — URL: https://www.top500.org/ .
2. Bondarenko A.A., Yakobovskii M.V. Obespechenie otkazoustojchivosti vysokoproizvoditelnyh vychislenij s pomoshchyu lokalnyh kontrolnyh tochek // Vestn. YUUrGU. Ser. Vych. matem. inform., 2014, tom 3, vypusk 3. — P. 20–36.
3. System for automating the development of parallel programs (DVM-system). — URL: http://dvm-system.org .
4. C-DVMH language, C-DVMH compiler, compilation, execution and debugging of DVMH programs. — URL: http://dvm-system.org/static_data/docs/CDVMH-reference-en.pdf .
5. Fortran DVMH langauge, Fortran DVMH compiler, compilation, execution and debugging of DVMH programs. — URL: http://dvm-system.org/static_data/docs/FDVMH-user-guide-en.pdf .
6. Bakhtin V.A., Klinov M.S., Kriukov V.A., Podderiugina N.V., Pritula M.N., Sazanov Iu.L. Rasshirenie DVM-modeli parallelnogo programmirovaniia dlia klasterov s geterogennymi uzlami // Vestnik Iuzhno-Uralskogo gosudarstvennogo universiteta, seriia "Matematicheskoe modelirovanie i programmirovanie", №18 (277), vypusk 12. — Cheliabinsk: Izdatelskii tsentr IuUrGU, 2012. — P. 82-92 .
7. Klinov M.S., Kriukov V.A. Avtomaticheskoe rasparallelivanie Fortran-programm. Otobrazhenie na klaster // Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo, № 2, 2009. — P. 128–134 .
8. Kataev N.A., Kolganov A.S., Smirnov A.S. Podderzhka interaktivnosti v sisteme SAPFOR // Nauchnyi servis v seti Internet: trudy XIX Vserossiiskoi nauchnoi konferentsii (18-23 sentiabria 2017 g., g. Novorossiisk). — M.: IPM im. M.V.Keldysha, 2017. — P. 243-249. — URL: http://keldysh.ru/abrau/2017/57.pdf doi:10.20948/abrau-2017-57 .