Реализация параллельного ввода-вывода в DVM-системе*

В.А. Бахтин[1,2], В.А. Крюков[1,2], Н.В. Поддерюгина[1], М.Н. Притула[1]

ФГУ ФИЦ ИПМ им. М.В. Келдыша РАН[1], МГУ им. М.В. Ломоносова[2]

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 гетерогенных вычислительных кластеров, в узлах которых в качестве вычислительных устройств наряду с универсальными многоядерными процессорами могут использоваться ускорители (графические процессоры или сопроцессоры Intel Xeon Phi). В статье будут рассмотрены новые возможности параллельного ввода-вывода, которые были реализованы в DVM-системе в последнее время. Использование разработанной подсистемы ввода-вывода позволяет существенно ускорить выполнение DVMH-приложений за счет совмещения выполнения операций ввода-вывода с вычислениями.

Ключевые слова: автоматизация разработки параллельных программ, DVM-система, параллельный ввод-вывод, контрольная точка, Фортран, Си.

1. Введение

Существует широкий класс задач, решаемых на современных вычислительных системах, которые требуют размещения 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. Для задач с моделируемым временем необходимо через некоторые промежутки времени запоминать текущее состояние некоторых массивов. Эти данные могут использоваться, например, в подсистемах визуализации.
4) Вычисления с массивами на внешней памяти. Если некоторые массивы не помещаются в оперативной памяти (рабочие внешние массивы), то организуется следующий процесс выполнения:
   - 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).

Все эти обмены осуществляются и в последовательной программе, выполняемой на одном процессоре. Для кластерных систем оптимальное управление обменами значительно усложняется по следующим причинам:

- используется распределенная оперативная память,
- используется распределенная внешняя память,
- существует множество различных файловых систем параллельного ввода-вывода со своими библиотеками поддержки.

* Работа поддержана грантами РФФИ № 16-07-01014 и 16-07-01067.

Поэтому «ручное» управление параллельным вводом-выводом является серьезной проблемой, как с точки зрения эффективности, так и с точки зрения мобильности параллельной программы.

В то же время, неэффективное управление параллельным вводом-выводом сводит на нет любые оптимизации параллельного выполнения задач с интенсивным вводом-выводом. В статье рассматриваются новые возможности для управления параллельным вводом-выводом, которые были реализованы в DVM-системе.

2. Обзор существующих решений

В настоящее время нет общего широко используемого интерфейса параллельного ввода-вывода. Для различных технологий параллельного программирования, существуют свои решения, например, MPI-IO[1] или OpenMP-IO[2]. В параллельных программах используются специализированные библиотеки ввода-вывода: HPF5[3], pNetCDF[4], предназначенные для работы с научными данными различных форматов.

Отсутствие общего подхода и стремление к достижению максимальной производительности ввода-вывода привело к тому, что разработчики новых языков программирования, например, UPC[5] (Unified Parallel C), Chapel[6], XcalableMP[7] вынуждены разрабатывать собственные средства для параллельного ввода-вывода.

2.1 MPI-IO

MPI представляет программисту набор функций MPI-IO - интерфейс для осуществления параллельного ввода-вывода.

Для осуществления ввода-вывода в MPI-IO предусмотрены 3 группы функций (по типу позиционирования в файле):

1) Функции, работающие с указанием точного смещения в файле, передаваемого в качестве параметра. Примером такой функции является MPI_File_write_at.

2) Функции, работающие с индивидуальными файловыми указателями. Для каждого MPI-процесса текущее смещение в файле увеличивается на размер блока после выполнения операции чтения/записи. Примером такой функции является MPI_File_write.

3) Функции, работающие с общим файловым указателем. MPI поддерживает общее для всех процессов смещение, которое меняется после завершения каждой операции ввода-вывода любого MPI-процесса. Примером такой функции является MPI_File_write_shared.

Все функции ввода-вывода в MPI-IO можно разделить на блокирующие (завершаются после выполнения соответствующей операции ввода-вывода) и неблокирующие (позволяют совместить вычисления и ввод-вывод), а также индивидуальные и коллективные (выполняемые всеми MPI-процессами).

Одним из достоинств MPI-IO является возможность работы с представлением файлов (file views) - описанием способа отображения частей файла в логическое представление файла в MPI.

Модель MPI-IO поддерживает ослабленную модель консистентности, возлагающую ответственность за создание согласованной версии файла на приложение.

К недостакам MPI-IO можно отнести их низкий уровень.

Параллельный ввод-вывод с использованием MPI-IO далеко не всегда обеспечивает кросплатформенность. Для организации эффективного ввода-вывода с помощь MPI-IO важно учитывать, как влияют на время работы такие параметры как: тип используемой функции ввода-вывода, размер файла, размер блока записи, представление файла на диске. Для разных платформ эти решения могут отличаться. Средства MPI-IO не всегда хорошо согласованы/настроены с файловыми системами параллельных компьютеров [8].

2.2 OpenMP-IO

В настоящее время для приложений, использующих OpenMP, возможности ввода-вывода существенно ограничены. Возможны два подхода.

Для того, чтобы поддерживать согласованность результирующего файла, операции чтения и записи должны выполняться вне параллельных областей. В случае, если несколько потоков обращаются к одному файлу, доступ к дескриптору файла (общему для всех потоков) должен быть защищен, например, с использованием механизма критических секций.

Другой подход - позволить каждому потоку использовать отдельный файл, чтобы избежать ошибок типа race condition или синхронизаций при доступе к общему дескриптору файла. Хотя такой подход часто приводит к более высокой производительности, чем описанный ранее, он имеет три основных недостатка. Во-первых, он требует дополнительных (достаточно дорогостоящих) шагов пред-и пост-обработки, для того чтобы создать необходимое количество входных файлов и объединить выходные файлы, записанные разными потоками. Во-вторых, такой сценарий работы трудно поддерживать в случае, когда число потоков может меняться динамически во время выполнения программы. В-третьих, управление большим количеством файлов часто является узким местом для сервера метаданных параллельной файловой системы. Последнее может стать актуальным в ближайшее время, как количество ядер современных микропроцессоров, как ожидается, вырастет до сотен или даже тысяч.

Для повышения производительности операций ввода-вывода в OpenMP приложениях разработан новый интерфейс параллельного ввода-вывода, который обеспечивает доступ нескольких потоков к одному файлу без необходимости явного использования операций синхронизации[2]. Разработанные спецификации очень близки к спецификациям MPI-IO. Основное отличие от MPI-IO - отсутствие неблокирующих операций ввода-вывода, а также отсутствие поддержки работы с производными типами данных (их нет в OpenMP).

Успешное внедрение OpenMP на мультипроцессорах, повсеместное внедрение SMP-кластеров (узлами которых являются мультипроцессоры), привело к тому, что все шире начал использоваться гибридный подход, когда программа представляет собой систему взаимодействующих MPI-процессов, а каждый процесс программируется на OpenMP.

Такой подход имеет преимущества с точки зрения упрощения программирования в том случае, когда в программе есть два уровня параллелизма - параллелизм между подзадачами и параллелизм внутри подзадач. Такая ситуация возникает, например, при использовании многообластных (многоблочных) методов решения вычислительных задач. Основной недостаток этого подхода очевиден - программисту надо знать и уметь использовать две разные модели параллелизма и разные инструментальные средства.

Важный вопрос - совместимость интерфейса MPI-IO и OpenMP-IO. На данный момент при разработке гибридных приложений в модели MPI/OpenMP приходится выбирать между двумя конкурирующими спецификациями. В зависимости от того, используется ли один файл для каждого процесса или существует один файл на несколько процессов, в приложении могут использоваться подпрограммы ввода-вывода OpenMP-IO или MPI-IO, но не оба одновременно. Для повышения производительности операций ввода-вывода на каждом узле требуется использование подпрограмм ввода-вывода OpenMP-IO в самой библиотеке MPI.

Таким образом, существующие решения MPI-IO и OpenMP-IO являются слишком низкоуровневыми и сложны для прикладных программистов, знакомых с разрабатываемой ими математической моделью, но далеких от тонкостей реализации/настройки параллельных файловых систем. Для достижения максимальной производительности ввода-вывода требуется расширение интерфейсов традиционных файловых систем, но такое расширение должно быть максимально простым и удобным для большинства программистов.

3. Реализация параллельного ввода-вывода в DVM-системе

В настоящее время осуществляется перенос компилятора с языка C-DVMH[9] на одну из промышленно используемых компиляторных платформ (CLANG+LLVM), поддерживающую современные стандарты языков Си и Си++. В связи с таким переходом многие решения, принятые в компиляторе были пересмотрены, в том числе была реализована новая подсистема ввода-вывода.

3.1 Последовательный синхронный ввод-вывод в компиляторе C-DVMH

В полном соответствии со стандартом C99 были реализованы следующие операции: 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.

Последовательный синхронный ввод-вывод был реализован следующим образом:

1) Файл открывается только одним процессом из текущей многопроцессорной системы. Назовем его процессом ВВ/ВЫВ.
2) Все операции над файлом производит только процесс ВВ/ВЫВ.
3) При запросе ВВ/ВЫВ распределенного массива т.к. данные распределены и их объем таков, что не позволяет сконцентрировать их в одном процессе, чтение (или запись) делается порциями около 100МБ, после (перед) которого (которой) делается рассылка (сбор) данных на (с) процессы (процессов) ими владеющие (владеющих).
4) Если операция предполагает запись пользовательских переменных или возврат значения, то они рассылаются процессом ВВ/ВЫВ.
5) Если в процессе выполнения операции возникла ошибка и стандарт предписывает установку errno, то его значение также рассылается процессом ВВ/ВЫВ.
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 (remove, rename, tmpnam) выполняются процессом ВВ/ВЫВ текущей многопроцессорной системы с рассылкой результатов выполнения на остальные процессы.

3.2 Параллельный синхронный ввод-вывод в компиляторе C-DVMH

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

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, читается (или записывается) только локальная часть распределенного массива. Тем самым работа с такими файлами является достаточно удобным средством параллельного ввода-вывода распределенных данных, однако требует совпадения распределений при записи файлов и при их последующем чтении.

Параллельный файл открывается каждым процессом и привязывается к текущей многопроцессорной системе. Почти все операции над параллельными файлами выполняются так же, как и над обычными файлами, за исключением операций fread, fwrite, fputs. Эти операции являются потенциально тяжелыми и предполагают заранее известный объем чтения/записи. Для них имеется параллельная реализация, состоящая в следующем:

1) Синхронизировать на файловую систему все операции с процесса ВВ/ВЫВ (т.к. все остальные операции выполняет только он). Для синхронизации содержимого файла в памяти с содержимым на диске используется системный вызов fdatasync в Linux или _commit в Windows.
2) Передать позицию в файле от процесса ВВ/ВЫВ 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 ВВ/ВЫВ 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).

3.3 Асинхронный параллельный ввод-вывод в компиляторе C-DVMH

Для реализации асихнронного ввода-вывода в компиляторе C-DVMH был добавлен новый режим для фунций fopen и freopen: асинхронный файл(**s**).

Лишь незначительная часть стандартных функций ВВ/ВЫВ не возвращают значения и не устанавливают errno. Это rewind и clearerr.

Поэтому 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 данных в файл, или чтение из файла в переданные переменные данных.

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 генерирует вызов невозвращающего значения варианта реализации.

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

Все операции над одним файлом выполняются последовательно. 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.

Отметим, что операция будет синхронной, если ее возвращаемое значение используется в программе или она должна устанавливать errno в случае ошибки.

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 обработки асинхронных операций каждым процессом заводится задаваемое параметром количество нитей, формирующих пул.

Если файл открывается в асинхронном нелокальном режиме, и реализация MPI поддерживает параллельную работу, то создается также дубликат коммуникатора текущей многопроцессорной системы, который будет использоваться для коммуникаций, связанных с этим файлом.

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

3.4 Параллельный ввод-вывод в компиляторе Fortran-DVMH

Для организации ввода-вывода данных в программе на языке Fortran-DVMH используются операторы стандарта Фортран 95.

Существующая версия языка Fortran-DVMH допускала только ограниченную форму этих операторов для распределённых массивов:

- Список ввода-вывода должен состоять только из одного имени распределённого массива и не может содержать других объектов ввода- вывода.
- В операторах ввода-вывода по формату допускается только формат, задаваемый ‘*’.
- Список управляющей информации не должен содержать параметры ERR, END и IOSTAT.
- В списке управляющей информации допускается использование только размноженных переменных.
- Не разрешается 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 не совместима с последовательной программой на Фортране 95. Данные, записанные одной программой, не могут быть прочитаны другой, вследствие разницы длин записей[9].

Учитывая данную особенность, переход на использование средств Си для ввода-вывода распределенных массивов из Fortran-DVMH программ не вносит никаких новых проблем.

В системе поддержки выполнения DVMH-программ на базе реализованных для языка C-DVMH функций параллельного ввода-вывода, были созданы новые функции, которые являются своего рода переходниками между операторами ввода-вывода языка Фортран и функциями ввода-вывода для языка C-DVMH.

Для задания режима выполнения ввода-вывода в язык Fortran-DVMH добавлена 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).

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=16000)
        DOUBLE PRECISION 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 языке Fortran-DVMH

4. Исследование эффективности реализованной подсистемы ввода-вывода

Для исследования эффективности реализованной подсистемы ввода-вывода 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. Замеряется время выполнения всей программы и время ввода-вывода.

Параметр **mode** функции fopen задает режим выполнения ввода-вывода: '**s**' - асинхронный; '**l**' - локальный (каждый процесс пишет в свой файл); '**p**' - параллельный (все процессы пишут в один и тот же файл).

Для выполнения операций ввода-вывода система поддержки выполнения DVMH-программ создает служебные нити. Все операции с одним файлом сериализуются. Операции с разными файлами могут выполняться одновременно. Количество нитей, используемых для ввода-

вывода, задается при помощи переменной окружения DVMH_IO_THREAD_COUNT (по умолчанию значение равно 5).

```
/* Jacobi-2 program */
#include <math.h>
#include <stdio.h>
#define Max(a, b) ((a) > (b) ? (a) : (b))
#define L 32000
#define ITMAX 100
#define mode "wbsl"
int i, j, it;
float eps;
float MAXEPS = 0.5f;
/* 2D arrays block distributed along 2 dimensions */
#pragma dvm array distribute[block][block]
float A[L][L];
#pragma dvm array align([i][j] with A[i][j])
float B[L][L];
int main(int an, char **as)
{
    double startt, endt;
    double ioTime = 0;
#ifdef _DVMH
    dvmh_barrier();
    startt = dvmh_wtime();
#else
    startt = 0;
#endif
    FILE *cp = fopen("jac_%02d.dat", mode);
    /* iteration loop */
    for (it = 1; it <= ITMAX; it++)
    {
        double t1 = dvmh_wtime();
        if (it % 10 == 0) {
            fclose(cp);
            cp = fopen("jac_%02d.dat", mode);
            fwrite(B, sizeof(float) * L, L, cp);
        }
        ioTime += dvmh_wtime() - t1;
        eps = 0;
        /* Parallel loop with base array A */
        /* calculating maximum in variable eps */
        #pragma dvm parallel([i][j] on A[i][j]) reduction(max(eps))
        for (i = 1; i < L - 1; i++)
            for (j = 1; j < L - 1; j++)
            {
                float tmp = fabs(B[i][j] - A[i][j]);
                eps = Max(tmp, eps);
                A[i][j] = B[i][j];
            }
        /* Parallel loop with base array B and */
        /* with prior updating shadow elements of array A */
        #pragma dvm parallel([i][j] on B[i][j]) shadow_renew(A)
        for (i = 1; i < L - 1; i++)
            for (j = 1; j < L - 1; j++)
                B[i][j]=(A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i + 1][j])/4.0f;
        printf(" IT = %4i   EPS = %14.7E\n", it, eps);
        if (eps < MAXEPS) break;
    }
    fclose(cp);
#ifdef _DVMH
```

```
    dvmh_barrier();
    endt = dvmh_wtime();
#else
    endt = 0;
#endif
    printf(" Jacobi2D Benchmark Completed.\n");
    printf(" Size            =    %6d x %6d\n", L, L);
    printf(" Iterations      =       %12d\n", ITMAX);
    printf(" Time in seconds =       %12.2lf\n", endt - startt);
    printf(" I/O time        =       %12.2lf\n", ioTime);
    printf(" END OF Jacobi2D Benchmark\n");
    return 0;
}
```

Рис. 2. Алгоритм Якоби на языке C-DVMH

На рис. 3 показаны времена выполнения C-DVMH-версии алгоритма 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 последовательный (OLD), параллельный (PARALLEL), локальный (LOCAL) и асинхронный/локальный режимы (ASYNCHRONOUS).

Использование локального режима для данного эксперимента позволило почти в 2 раза сократить время ввода-вывода (в этом случае каждый процесс сохраняет ½ 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. Использование буферизации существенно упрощает использование асинхронного ввода-вывода. Прикладной программист может удалить, перераспределить, модифицировать массив, не дожидаясь завершения выполнения операции вывода массива в файл.

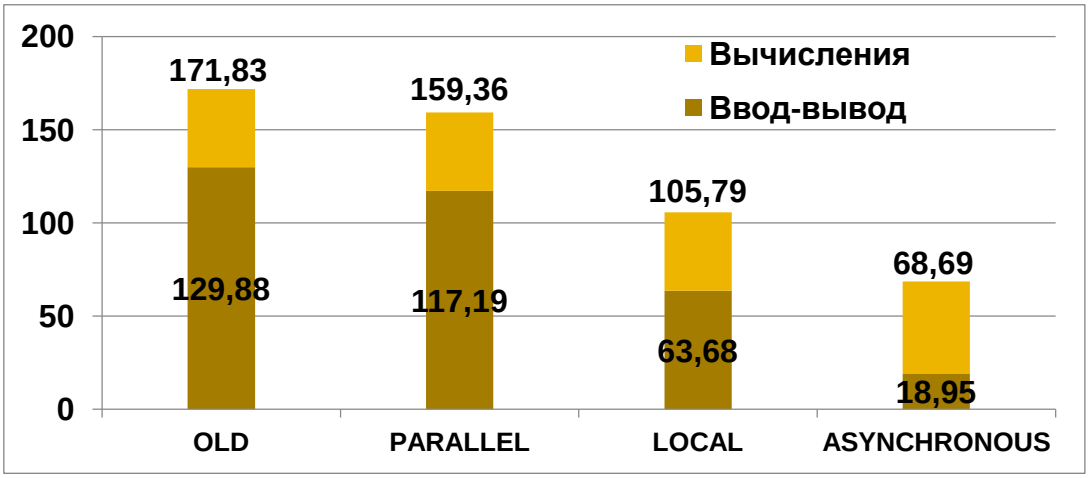

Рис. 3. Времена выполнения C-DVMH программы Якоби на суперкомпьютере «Ломоносов»

4. Заключение

В системе автоматизации разработки параллельных программ (DVM-системе) разработаны средства параллельного ввода-вывода, обеспечивающие эффективное использование возможностей файловых систем современных кластеров.

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).

Использование разработанных средств параллельного ввода-вывода в тестовых программах позволило существенно сократить время, требуемое для выполнения операций ввода-вывода.

Разработанные средства асинхронного 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.

Литература

1. Message Passing Interface Forum: MPI-2.2: Extensions to the Message Passing Interface (September 2009). URL: http://www.mpi-forum.org (дата обращения: 14.06.2016).
2. Kshitij Mehta, Edgar Gabriel, Barbara Chapman. Specification and Performance Evaluation of Parallel I/O Interfaces for OpenMP. OpenMP in a Heterogeneous World. Lecture Notes in Computer Science Volume 7312, 2012, pp. 1-14.
3. Parallel netCDF: A Parallel I/O Library for NetCDF File Access. URL: https://trac.mcs.anl.gov/projects/parallel-netcdf (дата обращения: 14.06.2016).
4. Group, H.D.F.: HDF5 Reference Manual. (November 2011) Release 1.8.8, National Center for Supercomputing Application (NCSA), University of Illinois at Urbana-Champaign. URL: https://www.hdfgroup.org/HDF5/doc/PSandPDF/HDF5_RefManual.PDF (дата обращения: 14.06.2016).
5. El-Ghazawi, T., Cantonnet, F., Saha, P., Thakur, R., Ross, R., Bonachea, D.:UPC-IO: A Parallel I/O API for UPC. V1.0. URL: http://www.gwu.edu/~upc/docs/UPC-IOv1.0.pdf (дата обращения: 14.06.2016).
6. Rafael Larrosa, Rafael Asenjo, Angeles Navarro, Bradford L. Chamberlain. A First Implementation of Parallel IO in Chapel for Block Data Distribution. ParCo 2011, September 2011. URL: http://chapel.cray.com/publications/ParCo-Larrosa.pdf (дата обращения: 14.06.2016).
7. XcalableMP Specification Working Group. XcalableMP. Language Specification. Version 1.2.1. URL: http://www.xcalablemp.org/download/spec/xmp-spec-1.2.1.pdf (дата обращения: 14.06.2016).
8. Кульдин С.П. Ввод-вывод на высокопроизводительных кластерных системах. URL: ftp://ftp.keldysh.ru/K_student/Diploms/dipl-Kuldin.pdf (дата обращения: 14.06.2016).
9. Язык C-DVMH. C-DVMH компилятор. Компиляция, выполнение и отладка CDVMH-программ. URL: http://dvm-system.org/static_data/docs/CDVMH-reference-ru.pdf (дата обращения: 14.06.2016).
10. 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 (дата обращения: 14.06.2016).

Implementation of parallel I/O in DVM-system

V.A. Bakhtin[1,2], V.A. Krukov[1,2], N.V. Podderyugina[1], M.N. Pritula[1]

Keldysh Institute of Applied Mathematics[1], Lomonosov Moscow State University[2]

DVM-system was designed to create parallel programs of scientific-technical calculations in C-DVMH and Fortran-DVMH languages. These languages use the same model of parallel programming (DVMH-model) and are the extensions of standard C and Fortran languages by parallelism specifications, implemented as compiler directives. DVMH-model allows to create efficient parallel programs for heterogeneous computational clusters, which nodes use as computing devices not only universal multi-core processors but also can use attached accelerators (GPUs or Intel Xeon Phi coprocessors). This article discusses new possibilities of parallel I/O, which were implemented in the DVM-system recently. Using the developed I/O subsystem can significantly speed up DVMH-applications by combining input-output operations with computations.

Keywords: automation the development of parallel programs, DVM-system, parallel I/O, checkpoint, Fortran, C.

References

1. Message Passing Interface Forum: MPI-2.2: Extensions to the Message Passing Interface (September 2009). URL: http://www.mpi-forum.org (accessed:14.06.2016).
2. Kshitij Mehta, Edgar Gabriel, Barbara Chapman. Specification and Performance Evaluation of Parallel I/O Interfaces for OpenMP. OpenMP in a Heterogeneous World. Lecture Notes in Computer Science Volume 7312, 2012, pp. 1-14.
3. Parallel netCDF: A Parallel I/O Library for NetCDF File Access. URL: https://trac.mcs.anl.gov/projects/parallel-netcdf (accessed:14.06.2016).
4. Group, H.D.F.: HDF5 Reference Manual. (November 2011) Release 1.8.8, National Center for Supercomputing Application (NCSA), University of Illinois at Urbana-Champaign. URL: https://www.hdfgroup.org/HDF5/doc/PSandPDF/HDF5_RefManual.PDF (accessed:14.06.2016).
5. El-Ghazawi, T., Cantonnet, F., Saha, P., Thakur, R., Ross, R., Bonachea, D.:UPC-IO: A Parallel I/O API for UPC. V1.0. URL: http://www.gwu.edu/~upc/docs/UPC-IOv1.0.pdf (accessed:14.06.2016).
6. Rafael Larrosa, Rafael Asenjo, Angeles Navarro, Bradford L. Chamberlain. A First Implementation of Parallel IO in Chapel for Block Data Distribution. ParCo 2011, September 2011. URL: http://chapel.cray.com/publications/ParCo-Larrosa.pdf (accessed:14.06.2016).
7. XcalableMP Specification Working Group. XcalableMP. Language Specification. Version 1.2.1. URL: http://www.xcalablemp.org/download/spec/xmp-spec-1.2.1.pdf (accessed:14.06.2016).
8. Kuldin S.P. Vvod-vyvod na vysokoproizvoditel'nyh klasternyh sistemah [Input-output on high-performance cluster systems]. URL: ftp://ftp.keldysh.ru/K_student/Diploms/dipl-Kuldin.pdf (accessed:14.06.2016).
9. Jazyk C-DVMH. C-DVMH kompiljator. Kompiljacija, vypolnenie i otladka CDVMH-programm [C-DVMH language. C-DVMH compiler. The compilation, execution and debugging of CDVMH-programs]. URL: http://dvm-system.org/static_data/docs/CDVMH-reference-ru.pdf (accessed:14.06.2016).
10. Jazyk Fortran-DVMH. Fortran-DVMH kompiljator. Kompiljacija, vypolnenie i otladka DVMH-programm [Fortran-DVMH language. Fortran-DVMH compiler. The compilation, execution and

debugging of DVMH-programs]. URL: http://dvm-system.org/static_data/docs/FDVMH-user-guide-ru.pdf (accessed:14.06.2016).