УДК 519.67; 532.546

ПРИМЕНЕНИЕ MPI-IO: РЕАЛИЗАЦИЯ МОДЕЛИ ШАХТНОЙ ВЕНТИЛЯЦИИ

Б. Л. Петушкеев[1]

Высокая вычислительная мощность кластеров позволяет не только решать текущие задачи быстрее, но и ставить другие, ранее недоступные. Однако это требует изменения подхода к программированию по причине возрастающих объемов результирующих данных. MPI, являясь стандартом де-факто для вычислительных систем с распределенной памятью, с момента публикации второй версии предоставляет средства параллельного ввода/вывода. Часто при проектировании приложений для решения больших задач данной возможностью пренебрегают. В работе представлены результаты практического внедрения MPI-IO в контексте моделирования процесса фильтрации газа через пористую среду шахтных выработок. Рассматриваются вопросы масштабирования программы и производительности файловой подсистемы, а также описаны предпринятые действия по оптимизации последовательного и параллельного кодов. Все работы проводились на базе кластера СКИФ-Cyberia МВЦ ТГУ. Работа выполнена при финансовой поддержке РФФИ (код проекта 08–08–12029-офи).

Ключевые слова: высокопроизводительные вычисления, гидродинамика, фильтрация, параллельный ввод/вывод, MPI-IO, масштабируемость, управление данными.

1. Мотивация и описание проблемы. В Томском государственном университете (ТГУ) на протяжении нескольких лет ведутся исследовательские работы по созданию эффективных моделей разнообразных процессов, протекающих в выработанных пространствах угольных шахт. Такие области образуются после прохождения угледобывающего комбайна, вслед за которым кровля шахты без соответствующей опоры обваливается. Внимание к выработкам вызвано тем, что они представляют собой потенциально взрывоопасные участки с локально повышенным содержанием метана, ответственным за все крупные техногенные катастрофы. Посредством системы шлюзов, скважин и подачи свежего воздуха организуется проветривание выработки, но возникает вопрос, где эффективнее располагать элементы вентиляции, чтобы учесть особенности движения метана в пористом пространстве.

Предыдущая версия этого приложения [1] обеспечивает возможность моделирования пространств протяженностью до нескольких километров на сетке с числом ячеек порядка нескольких миллионов, в то время как требуется увеличить эти показатели по крайней мере на порядок. Расчеты проводятся на кластере СКИФ-Cyberia МВЦ ТГУ [2] с момента введения последнего в строй. Проблема заключается в том, что, реализуя доменную декомпозицию расчетной области, исследователь вынужден хранить на мастер-процессе весь объем данных, обеспечивая ведомых процессов необходимой информацией, и собирать, а затем записывать обработанные результаты на диск, или же организовывать ввод/вывод (I/O) каждым процессом, а также корректный обмен в ходе расчета. Однако первый подход страдает дефицитом оперативной памяти, но прост в реализации и поэтому наиболее часто используется, а второй — последовательными медленными операциями I/O. К параллельной стратегии оперирования с файлами приводят следующие аргументы:

1) последовательный I/O в больших задачах становится узким местом;

2) для работы с глобальным файлом (слияние, извлечение и т.д.) требуется трудоемкий пост- и препроцессинг;

3) средства API (Application Programming Interface), предоставляемые производителями параллельных файловых систем, часто (или как правило) не переносимы; это касается неблокирующих функций I/O, функций предварительного размещения файла, установки режима доступа к файлу (атомарный или неатомарный) и т.д.;

4) современные файловые системы позволяют достичь отличной производительности и масштабируемости на операциях с параллельным I/O.

[1] Томский государственный университет, физико-технический факультет, просп. Ленина, 36, 634045, Томск; аспирант, e-mail: petushkeev@ftf.tsu.ru

Большинство реализаций MPI используют ROMIO (проект Argonne National Laboratory) [3] или его производные [4], имеющие прослойку ADIO (Abstract Device Interface for I/O) [5], написанную и оптимизированную под определенную файловую систему и, таким образом, обеспечивающую переносимость (рис. 1). MPI-IO предоставляет такие возможности, как чтение/запись непоследовательных блоков данных, неблокирующие операции I/O, коллективные операции I/O, HPF-стиль представления распределенного массива.

Вообще говоря, есть три причины осуществления операций I/O в ходе численного эксперимента. Во-первых, следует задать сетку и начальные данные, во-вторых, по мере продвижения расчетов расставлять контрольные точки для возможности продолжить расчет, одновременно можно проанализировать и (или) визуализировать текущие результаты. Однако сохранять данные на локальный диск нельзя, иначе сбой на вычислительном узле приведет к ситуации, когда расчет придется начинать сначала. Некоторые приложения также используют файлы для хранения рабочих данных в процессе счета. В-третьих, требуется записать полученные результаты в единый файл. MPI-IO позволяет делать подобные операции гораздо эффективнее [6, 7], что и демонстрируется в настоящей работе.

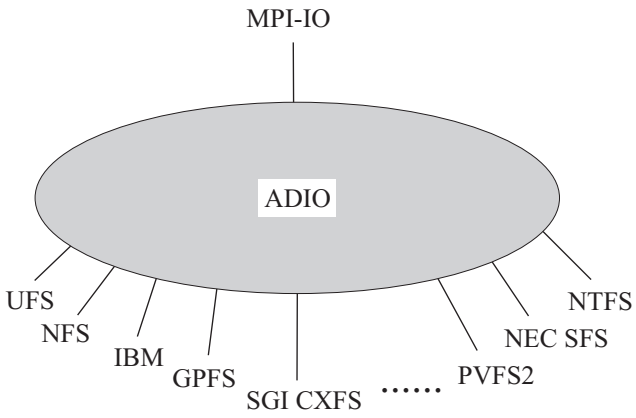

Рис. 1. MPI-IO реализован над программным уровнем ADIO

Далее в статье описываются используемая математическая модель (раздел 2), в применении к которой была разработана параллельная программа, и работы по оптимизации приложения (раздел 3). В разделе 4 проводим проверку целесообразности использования MPI-IO и демонстрируем потенциал разработанной программы. В заключительной части кратко подводятся итоги работы и делается ряд выводов.

2. Характеристика используемой модели. Физико-математическая постановка задачи базируется на рассмотрении трехмерного пространства, заполненного обрушенными горными породами. Вводится величина пористости $\varepsilon = \dfrac{V - V_T}{V}$, где V — объем, занимаемый двухфазной средой, V_T — объем, занимаемый твердой фазой. Специфика математической модели, описывающей движение газа в среде обрушенных пород, состоит в том, что необходимо учитывать воздействие этих пород на динамику газа. К таким особенностям относятся, во-первых, фактор уменьшения проходных сечений и, следовательно, объема, занимаемого газом в пористой среде, по сравнению с проходными сечениями и объемами в свободном пространстве. Во-вторых, существенное воздействие на течение газа оказывает сила сопротивления со стороны частиц породы. Такая сила обусловлена вязким сопротивлением (размеры пор много меньше характерных размеров течения) и может быть учтена только в среднем как сила объемного сопротивления. Вместе с тем газ, совершая работу против сил сопротивления пористой среды, диссипирует свою кинетическую энергию, поэтому соотношение, отвечающее за баланс энергии, должно учитывать указанные силы сопротивления.

Система уравнений в интегральной форме принимает следующий вид:

уравнение неразрывности: $\dfrac{\partial}{\partial t}\int\limits_\Omega \varepsilon\rho\, d\Omega + \oint\limits_\Sigma \varepsilon\rho(\boldsymbol{q}\cdot p\boldsymbol{n})\, d\Sigma = 0,$

уравнение движения: $\dfrac{\partial}{\partial t}\int\limits_\Omega \varepsilon\rho\boldsymbol{q}\, d\Omega + \oint\limits_\Sigma \varepsilon\left[p\boldsymbol{n} + (\boldsymbol{q}\cdot\boldsymbol{n})\boldsymbol{q}\rho\right] d\Sigma = \int\limits_\Omega \boldsymbol{F}_c\, d\Omega + \int\limits_\Omega p\ \mathrm{grad}\ \varepsilon\, d\Omega + \int\limits_\Omega \varepsilon\rho\boldsymbol{g}\, d\Omega,$

уравнение энергии: $\dfrac{\partial}{\partial t}\int\limits_\Omega \varepsilon\rho\left(e + \dfrac{|\boldsymbol{q}|^2}{2}\right) d\Omega + \oint\limits_\Sigma \varepsilon\rho(\boldsymbol{q}\cdot\boldsymbol{n})\left[e + \dfrac{p}{\rho} + \dfrac{|\boldsymbol{q}|^2}{2}\right] d\Sigma - \int\limits_\Omega \varepsilon\rho(\boldsymbol{g}\cdot\boldsymbol{q})\, d\Omega = 0,$

уравнение переноса в форме уравнения неразрывности для метана: $\dfrac{\partial}{\partial t}\int\limits_\Omega \varepsilon c\rho\, d\Omega + \oint\limits_\Sigma \varepsilon c\rho(\boldsymbol{q}\cdot\boldsymbol{n})\, d\Sigma = 0$, где $\boldsymbol{q}$ — вектор скорости, $\boldsymbol{g}$ — ускорение силы тяжести, $\boldsymbol{F}_c$ — вектор объемных сил сопротивления, c — концентрация метана. Слагаемое $\int\limits_\Omega p\ \mathrm{grad}\ \varepsilon\, d\Omega$ в правой части уравнения движения появляется в резуль-

тате учета импульса давления со стороны пористой среды, слагаемое $\int\limits_\Omega \varepsilon\rho\boldsymbol{g}\,d\Omega$ учитывает силу тяжести. В уравнении энергии член $\int\limits_\Omega \varepsilon\rho(\boldsymbol{g}\cdot\boldsymbol{q})\,d\Omega$ отвечает за влияние силы тяжести на энергию объема двухфазной среды. Для замыкания системы уравнений необходимо добавить уравнение состояния $\epsilon = \dfrac{1}{k-1}\dfrac{p}{\rho}$, $k = \dfrac{C_{p\mathrm{M}}\,c + C_{p\mathrm{V}}(1-c)}{C_{v\mathrm{M}}\,c + C_{v\mathrm{V}}(1-c)}$, где индекс "V" относится к воздуху, "M" означает метан, C_p и C_v — удельные теплоемкости при постоянном давлении и объеме соответственно.

Представленная система уравнений рассчитывается методом Годунова с использованием процедуры ускорения [8], позволяющей значительно сократить время расчета. Источником метана считается верхний свод кровли выработки, напряженность в котором вызывает образование трещин. Моделью предусмотрена возможность задания источников и стоков газа, таким образом определяя конфигурацию процесса вентилирования. Искомыми параметрами являются вектор скорости, концентрация метана, давление и плотность газа, а также, весьма показательными в плане превышения ПДК и создания взрывоопасной ситуации, величины концентрации метана в сечениях стоков газа. Реализация численного метода хорошо себя зарекомендовала на ряде задач о распространении взрывной волны и других задачах механики движения газа.

Разрешение модели по всем трем координатам корректируется в соответствии с имеющимися вычислительными мощностями и габаритами моделируемой области. В число начальных данных, задаваемых пользователем, входит также ряд горно-геологических и горнотехнических параметров.

3. Трудности оптимизации. Оптимизация программного кода на однопроцессорной машине, конечно, имеет важное значение и прямым образом сказывается на качестве параллельной программы, позволяя уменьшить и время расчета и, что тоже актуально, итоговую стоимость расчетов. Сегодня доступен ряд средств от компаний-производителей процессоров для анализа работы программы на предмет выявления "горячих" мест в коде с учетом архитектуры CPU. В нашем случае оптимизация проводилась для процессоров Intel Xeon с использованием соответствующего компилятора Intel C++ версии 10.1 и утилиты Intel VTune Performance Analyzer.

Спецификой написанной программы является отсутствие кусков кода, на которых эффективным решением могло стать использование агрессивных и довольно простых методик оптимизации, таких как раскрутка циклов или векторизация. Поэтому на первом этапе проведена работа по ручному переписыванию отдельных частей кода (особенно циклов), устранению ошибок предсказания переходов на ветвлениях и т.д., к тому же некоторые часто вызываемые функции переопределялись как встроенные [9]. Получить 50%-й эффект в короткие сроки помог VTune, заострив внимание лишь на конкретных проблемных точках.

Таблица 1

Время расчета тестовых прогонов программы на одном процессоре

Оптимизация	Время CPU, с	Ускорение
отсутствует	790	1
+ ручное переписывание частей кода	398	1.9
+ ключи компилятора	224	3.5
+ IPO	166	4.8
+ IPO + PGO	153	5.1

Дальнейшая действия заключались в работе с компилятором, а именно подбор ключей для автоматической оптимизации и указания компилятору использовать встроенные анализаторы: IPO (Interprocedural Optimization) и PGO (Profile-Guided Optimizations). Отчеты, полученные от HLO (High-Level Optimizations), не представляли интереса и во внимание не принимались. В табл. 1 приведены результирующие величины ускорения и времени выполнения тестового расчета после каждого этапа оптимизации. К тому же, во избежание проблем с управлением большими массивами, код аккуратно перенесен с 32-битной платформы на 64-битную.

Для распараллеливания применена блочная декомпозиция расчетной области в трех измерениях, хорошо подходящая для реализации численных методов. Конфигурация декомпозиции соответствует сгенерированной при запуске программы декартовой топологии процессов, зависящей от размерности области и количества используемых процессоров. Главный цикл организован стандартно и включает в себя обмен

информацией в "затененных" областях, синхронизацию ряда параметров и проверку условия окончания расчета.

Балансировка загрузки между процессорами осуществляется исходя из разбиения расчетных подобластей на внутренние и внешние, своей гранью выходящие на внешнюю границу глобальной области. Происходит это на этапе декомпозиции и в дальнейшем не пересматривается. Пересылки между процессами заменены на персистентные (в литературе встречаются как отложенные), дающие, как показал опыт на других программах, некоторый положительный эффект. К тому же все производные MPI-типы данных определяются один раз и используются до окончания расчета. В данной работе реализован двухпроходной алгоритм, на втором этапе которого требуется пересылать лишь часть из требуемых параметров. Для минимизации объема пересылок сначала использовался дополнительный массив из структур с меньшим количеством полей, но затем был определен новый MPI-тип, описывающий только нужные величины. Такой подход позволил еще немного улучшить итоговые результаты.

4. Тестирование. Авторы ROMIO предложили следующую классификацию использования приложением MPI-IO, отражающую все большее количество передаваемой в нижележащие компоненты MPI информации, представленную в табл. 2.

Таблица 2

level-0:	независимое обращение к одному непрерывному блоку данных (Unix style);
level-1:	level-0, но вызовы I/O заменены на коллективные;
level-2:	процесс создает MPI-тип для описания шаблона доступа к данным в памяти, определяет файловый вид (шаблон доступа к файлу) и независимо вызывает I/O функции;
level-3:	level-2, но вызовы I/O заменены на коллективные.

Воспользовавшись более высоким уровнем, можно получить больший выигрыш в быстродействии за счет выполняемой ROMIO оптимизации доступа [11–13]; например, level-2 задействует механизм data sieving (механизм просеивания информации), а level-3 — так называемый двухпроходной алгоритм. Главная цель указанных оптимизаций: минимизировать эффект высокой латентности I/O, уменьшив количество запросов к файловой системе. Data sieving достигает этого, читая информацию большими блоками (даже если данные лежат фрагментарно), затем копируя действительно нужные в пользовательский буфер, а запись идет в режиме прочитать–изменить–записать, одновременно требуя блокировки части файла. Двухпроходной алгоритм работает только на коллективных I/O, что позволяет проанализировать и скоординировать запросы от разных процессов. Выделяют фазу I/O и фазу коммуникации: при чтении процесс сначала читает отведенную ему часть файла (возможно задействуя data sieving), затем пересылает часть из данных тем, кому они действительно нужны. Запись производится в обратном порядке. Вместе с тем, предоставляя MPI-IO максимум информации о запросе (level-3), ROMIO может получить дополнительные преимущества от кэширования и упреждающей выборки, выполняемой самой файловой системой.

Для того чтобы показать производительность файловой системы в разных режимах, были проведены эксперименты на операциях чтения/записи трехмерного массива вещественных чисел double в/из файла общим размером 1 Гб. Массив на сетку процессов распределен блочно в форме (block, block, block), часто используемой в научных приложениях. Следует отметить также тот факт, что, несмотря на последовательный без перекрытий шаблон доступа к данным с точки зрения приложения, доступ к файлу может стать нерегулярным (рис. 2). Обычно в реальности это приводит к деградации производительности из-за генерирования большого количества вызовов функций I/O к маленьким последовательным участкам данных.

Во всех экспериментах использовались значения по умолчанию для параметров filestripping и размеров внутренних буферов ROMIO, варьировать которые можно с помощью функции MPI_Info_set. Целевая платформа кластера СКИФ-Cyberia использует параллельное хранилище ReadyStorage ActiveScale Cluster, основанное на файловой системе PanFS компании Panasas, Inc., с заявленной производительностью на уровне 700 MB/s. Эта специализированная файловая система удовлетворяет специфическому требованию параллельных вычислений: обеспечивает эффективный параллельный доступ к хранилищу данных каждому узлу кластера. К тому же PanFS отлично масштабируется, сохраняя высокую производительность при увеличении количества вычислительных узлов. В настоящее время существуют лишь три файловые системы, которые могут эффективно использоваться на больших системах размером в сотни и тысячи узлов — это Lustre компании Sun, PanFS компании Panasas (установлена на всей линейке

кластеров “СКИФ”) и GPFS от IBM (BlueGene/P на факультете вычислительной математики и кибернетики МГУ, а также МВС-100К в Суперкомпьютерном центре РАН). Существуют и некоторые другие масштабируемые решения. Однако они проигрывают по параметру отказоустойчивости. PanFS и Lustre относятся к классу архитектуры, основанной на объектно-ориентированных устройствах хранения (OSD, Object-based Storage Device), GPFS использует комбинированную. Идея объектной архитектуры заключается в том, что исходные файлы преобразуются в набор объектов, находящихся на разных физических устройствах; в дальнейшем вычислительный узел, имея “карту” размещения данных, работает напрямую с устройствами в отличие от традиционных хранилищ, где узкое место — файловый сервер. Таким образом, если файловая система распределяет информацию по нескольким OSD-устройствам, то рост числа операций I/O и пропускной способности ожидается линейный. К тому же и с увеличением числа клиентов должна проявиться независимость хранилища от сервера метаданных, поскольку работа по управлению блоками данных перенесена на сами устройства хранения.

Отличительной особенностью MPI является использование производных типов данных не только для описания шаблона данных в памяти, но и для определения файлового вида (fileview): суммы компонентов из смещения в файле, базового типа (etype) и файлового типа (filetype). При открытии файла начальный файловый вид — это весь файл, но затем он переопределяется посредством функции MPI_File_set_view и дальнейший доступ происходит только в указанную для каждого процесса область файла. Производные типы данных обладают хорошей гибкостью и позволяют описывать практически произвольный шаблон, причем компактно. Таким образом, алгоритм работы с файлом в нотации MPI сводится к следующим пунктам:

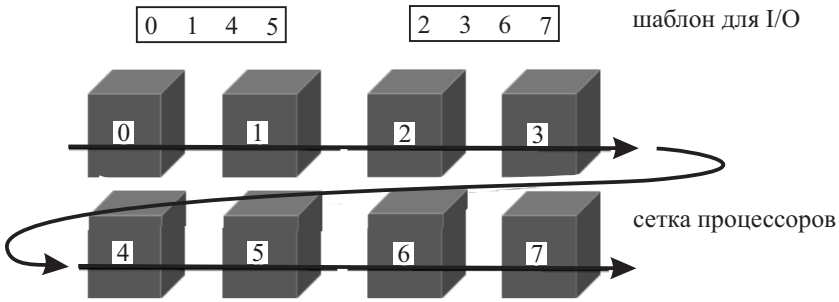

Рис. 2. Пример отображения сетки процессов на I/O узлы

— создание производного типа данных (или ограничиться базовым типом);
— MPI_File_open(... , &fhandle), открытие(создание) файла;
— MPI_File_set_view(fhandle, ... , filetype, ...), на уровнях I/O level-2 и level-3;
— MPI_File_read(fhandle, ... , buffer, ...);
— MPI_File_close(&fhandle).

В режимах level-2 и level-3 операции записи/чтения выполняются вызовом одной функции один раз.

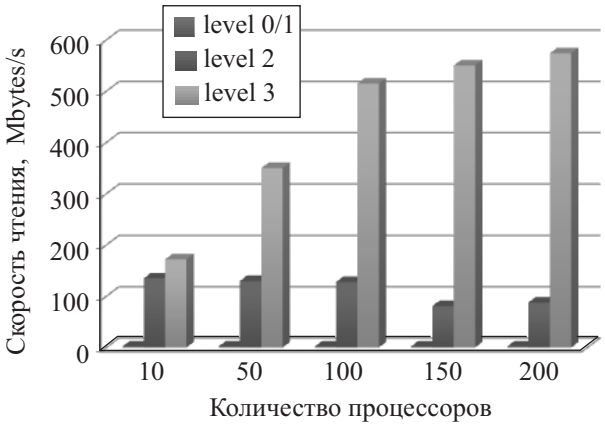

Рис. 3. Производительность файловой системы при чтении (массив $512 \times 512 \times 512$ doubles)

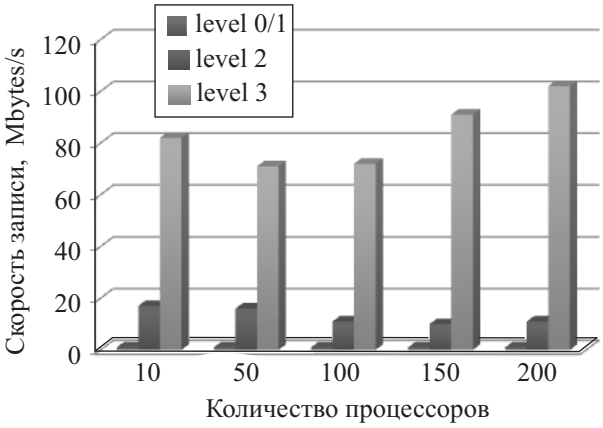

Рис. 4. Производительность файловой системы при записи (массив $512 \times 512 \times 512$ doubles)

На рис. 3 и 4 представлены графики пропускной способности в зависимости как от уровня I/O вызовов, так и от количества задействованных процессоров (процессор рассматриваем одноядерный). Пропускная способность рассчитывалась как значение объема переданных данных, отнесенное к максимальному времени чтения/записи любого из процессов. Отметим чрезвычайно низкую производительность level-0 и level-1 (менее 2 MB/s) вследствие, в основном, того, что вызовы level-0/1 транслируются в

большое количество операций чтения/записи. Level-1 для распределенного массива, как в нашем случае, просто вызывает соответствующие независимые функции I/O, поэтому на диаграммах они объединены и показывают одинаково плохие результаты (однако, например, файловая система Lustre отлично себя показывает при level-0 I/O благодаря кэшированию, упреждающей выборке и записи из буфера по заполнении, но level-1 рекомендуют избегать из-за конфликтов при блокировке [6]). Level-2 демонстрирует существенно более высокие показатели относительно level-0/1, выполняя операции чтения в 80–130 раз быстрее, а операции записи — в 10–17 раз. Однако при этом с ростом числа процессоров быстродействие падает, связано это с особенностью работы механизма data sieving, из-за чего при чтении требуются дополнительные обращения к файловой системе; к тому же для операции записи необходима блокировка части файла и, как следствие, увеличение количества конфликтных ситуаций за владение ресурсом. Последний недостаток успешно устраняется на следующем уровне level-3: в этом случае рост относительно level-2 при числе процессов 200 для операций чтения составляет 8 раз, записи — 12 раз (авторы [6] тоже отмечают хорошие результаты у Lustre на уровнях level-2 и level-3). Следует отметить, что величины пропускной способности еще не вышли на точку насыщения и будут выше при большем количестве клиентов.

Испытания на масштабируемость итоговой программы проводились на сетке с количеством ячеек, равным 24 млн. Это число подбиралось так, чтобы расчет для сравнения можно было провести на малом количестве процессоров без свопинга (в такой конфигурации суммарно требуется более 18 Гб оперативной памяти). В ходе работы программы запись в файл не велась, но через некоторые промежутки времени выдавались текущие данные. Итоговый график ускорения приведен на рис. 5 в единицах относительно времени счета на 25 процессорах. Результаты тестов показывают практически линейное ускорение с ростом количества процессоров. При 1000 процессорах объем расчетов на временно́м шаге невелик и на конечную величину ускорения сильно влияет межпроцессный обмен, при соответствующем количестве узлов уменьшение наклона кривой не наблюдается. При 400 процессорах заметен прогиб; при этой конфигурации сложилась ситуация, когда объем пересылок с процесса в направлении оси z больше, чем в другом (ось x, разбиения по оси y не было во всех случаях), но последовательность обменов организована как x–y–z. В остальных тестах ситуация обратная: объем пересылок по оси x всегда превалировал. На этом вопросе внимание не заострялось — была выявлена только закономерность. Вполне вероятно, что описанная картина связана с буферизацией MPI на первоначальном обмене и на ускорении выполнения пересылки большего массива.

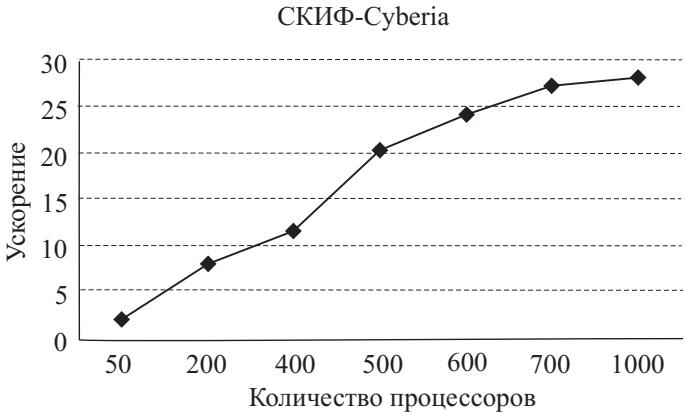

Рис. 5. Ускорение относительно времени расчета на 25 процессорах (1CPU=1core), 24 млн. узлов

5. Заключение. Проведенные работы по оптимизации и совершенствованию параллельного приложения, более полно задействующего потенциал вычислительной системы, показали пригодность программы к проведению моделирования задач гидродинамики на платформах с гораздо бóльшим количеством процессоров, чем сегодня доступно для наших исследований. Данная работа не нацеливалась на сравнение различных высокоэффективных файловых систем, но в ней продемонстрированы возможности MPI-IO, игнорирование которых на больших задачах приведет к дисбалансу между временем счета и временем операций I/O. Внедрение MPI-IO устраняет подобные недостатки, облегчает оперирование и координацию информации между процессами и в итоге позволило реализовать действительно высокомасштабируемое приложение [14, 15].

Большой размер файлов на контрольных точках и объем результирующих данных (для использованной конфигурации каждое сохранение порядка 1.5 Гб) требует внимательного отношения к вопросам хранения, обработки, а также их визуализации. В данной работе указанные моменты не рассматривались, но надеемся сделать это в следующих работах.

СПИСОК ЛИТЕРАТУРЫ

1. *Васенин И.М., Петушкеев Б.Л.* Опыт применения параллельных вычислений к проблематике шахтной вентиляции // IV Сибирская школа-семинар по параллельным и высокопроизводительным вычислениям. Томск,

2007. 116–122.
2. Сайт МВЦ ТГУ (http://www.skif.tsu.ru).
3. ROMIO. A high-performance, portable MPI-IO implementation (http://www.mcs.anl.gov/romio).
4. *Yu H.* On developing BlueGene/L MPI-IO with high performance (http://w3.ibm.com/ibm/presentations).
5. *Thakur R., Gropp W., Lusk E.* An abstract-device interface for implementing portable parallel-I/O interfaces // Proc. of the 6th Symposium on the Frontiers of Massively Parallel Computation. 1996. 180–187.
6. *Kimpe D., Lani A., Quintino T., Vandewalle S., Poedts S., Deconinck H.* A Study of real world I/O performance in parallel scientific computing // PARA 2006, LNCS 4699. 2007. 871–881.
7. *Borrill J., Oliker L., Shalf J., Shan H.* Investigation of leading HPC I/O performance using a scientific-application derived benchmark // Int. Conf. for High-Performance Computing Networking Storage. Reno, 2007.
8. *Васенин И.М., Петушкеев Б.Л.* Задача об ускорении решения совместных уравнений газовой динамики и переноса // Известия ВУЗов. Физика. 2007. № 9/2. 274–281.
9. Intel Architecture Software Developer's Manual. Volume 3: System Programming (http://www.intel.com/products/processor/manuals/index.htm).
10. Message Passing Interface Forum, MPI-2: Extensions to the Message-Passing Interface, July 1997 (http://www.mpi-forum.org/docs/docs.html).
11. *Thakur R., Gropp W., Lusk E.* Optimizing noncontiguous access in MPI-IO // Parallel Computing. 2002. **28**. 83–105.
12. *Thakur R., Gropp W., Lusk E.* A case for using MPI's derived datatypes to improve I/O performance // Proc. of SC98: High Performance Networking and Computing. San Jose, 1998.
13. *Thakur R., Gropp W., Lusk E.* Data sieving and collective I/O in ROMIO // Proc. of the 7th symposium on the Frontiers of Massively Parallel Computation. Annapolis, 1999. 182–189.
14. *Cui H., Moore R., Olsen K., Chourasia A., Maechling P., Minster B., Day S., Hu Y., Zhu J., Majumdar A., Jordan T.* Enabling very-large scale earthquake simulations on parallel machines // ICCS 2007, Part I, LNCS 4487. 46–53.
15. *Cook A., Cabot W., Welcome M., Williams P., Miller B., de Supinski B., Yates R.* Tera-scalable algorithms for variable-density elliptic hydrodynamics with spectral accuracy // Proc IEEE/ACM SC05. Seattle, 2005.

Поступила в редакцию
02.02.2009