Система управления распределёнными структурами данных

Е.Г. Стадник
Новосибирский государственный технический университет

Рассмотрены основные проблемы, возникающие при работе с распределёнными данными, а также их возможные решения. Представлена библиотека для работы с распределёнными данными. Особое внимание уделено работе с асинхронной передачей данных.

Введение

Многие задачи численных методов связаны с обработкой большого объема данных. Часто задачи такого типа требуется решать с использованием мультикомпьютеров. При этом большие массивы разрезаются на части и части обрабатываются с выполнением коммуникаций для удовлетворения зависимостей по данным. Такой подход требует существенных усилий для разработки и реализации алгоритмов организации распределенной обработки данных.

Целью проекта является разработка библиотеки DDS (Distributed Data System), реализующей распределенные структуры данных.

Наиболее распространённой структурой данных для численных методов является n-мерный массив. Для распределённой реализации такой структуры можно выделить следующие требования:

- Должна сохраняться регулярная структура. Это позволит обрабатывать данные с помощью циклов так же, как и в исходном массиве.
- Должны удовлетворяться все зависимости по данным на границах разреза посредством коммуникаций.
- Коммуникации должны выполняться на фоне вычислений.
- Должна быть предусмотрена возможность выполнения балансировки вычислительной нагрузки.

Чтобы система удовлетворяла первому требованию, необходимо использовать блоки данных той же размерности и структуры, что и исходный массив. Под сохранением структуры понимается аналогичный способ размещения данных в памяти (например, построчное хранение массивов в Си). Второе требование предполагает хранение границ соседних блоков, что позволит уменьшить количество пересылок данных. Третье требование подразумевает асинхронную работу потоков пересылки данных и вычислений. Это можно обеспечить, размещая на одном узле несколько блоков: одни участвуют в коммуникаци-

ях, над другими производятся вычисления. Такие блоки будем называть фрагментами данных. Для удовлетворения четвертого требования нужно либо уметь перестраивать блоки в ходе вычислений, либо иметь на каждом узле множество блоков и переносить их для балансировки.

Анализ существующих систем

Наиболее известные системы для автоматизации распределения многомерных массивов – HPF университета William Marsh Rice (Хьюстон, Техас, США) и DVM Института прикладной математики им. М.В. Келдыша РАН. Системы имеют в основе сходные принципы распределения данных. Рассмотрим систему DVM подробнее. Инструмент представляет собой расширение стандартного языка Си или FORTRAN средствами спецификации параллельного выполнения программы:

- распределение элементов массива между процессорами;
- распределение витков цикла между процессорами;
- спецификация параллельно выполняющихся секций программы (параллельных задач) и отображение их на процессоры;
- организация эффективного доступа к удаленным (расположенным на других процессорах) данным;
- организация эффективного выполнения редукционных операций – глобальных операций с расположенными на различных процессорах данными (таких, как их суммирование или нахождение их минимального значения).

Достоинства:

- Повышение скорости разработки параллельных программ, реализующих численные методы с регулярными структурами данных и вычислений.
- Возможность контролировать процесс распараллеливания, добиваясь максимальной эффективности программ.
- Имеется встроенный отладчик и профайлер.

Недостатки:

- Отсутствие средств для автоматизации динамической балансировки.
- Элементы массивов не могут иметь произвольный тип, только стандартные int, long, float, double.
- Не обеспечивается правильная обработка указателей на элементы массивов при перераспределении данных.

В целом система DVM представляет собой удобный инструмент, позволяющий легко получить параллельную программу из последова-

тельной. Однако для того чтобы параллельная реализация была эффективной, на исходную программу должны быть наложены существенные ограничения. Во-первых, итерации при работе с распределёнными данными должны быть независимы. Во-вторых, время выполнения итераций должно быть примерно одинаково. Система даёт хороший результат на задачах, не требующих динамических свойств параллельной программы. Перераспределение вычислений в DVM означает глобальные действия по перераспределению всего массива, что может быть неэффективно при использовании большого количества процессоров и когда часто возникает потребность в балансировке.

Описание библиотеки DDS

Последовательность действий пользователя при работе с библиотекой выглядит следующим образом: определяется переменная типа «распределённый массив» и для неё указываются требуемые параметры распределения. На основании полученных данных система формирует локальные фрагменты на каждом вычислительном узле и передаёт управление пользователю. Далее пользователь получает возможность обращаться к локальным фрагментам и обрабатывать содержащиеся в них данные. Фрагменты данных, расположенные на одном узле, представляются пользователю как контейнер фрагментов. Доступ к фрагментам возможен посредством итератора. По окончании обработки фрагмента пользователь уведомляет об этом систему и информация о граничных элементах автоматически обновляется по мере готовности соседних фрагментов. Обработка данных фрагмента реализуется пользователем в основном потоке, а приём данных с других узлов автоматически осуществляется в параллельном потоке, что позволяет снизить временные затраты на синхронизацию за счёт передачи данных в фоновом режиме.

Интерфейс библиотеки:

Функции инициализации и завершения работы:

```
//инициализация библиотеки
int DDS_Init(int *argc, char ***argv);
//запуск потока для асинхронного приёма данных
int DDS_Begin();
//завершение работы с библиотекой
int DDS_Finalize();
```

Функции для работы с контейнером фрагментов (класс DDS_Array):

```
//задаёт размеры и количество фрагментов
int set_all_size(int _Nx, int _Ny, int _kx, int _ky);
```

//задаёт идентификатор переменной и тип распределения фрагментов

int set_parameters(int _id, int _distrib_type);

//возвращает итератор на первый фрагмент

iterator begin();

//возвращает итератор на последний фрагмент

iterator end();

//возвращает итератор фрагмента с индексом i

iterator operator[] (unsigned i);

Функции для работы с фрагментом данных (класс DDS_Array_fragment):

//возвращает указатель на данные фрагмента

T* Get_Data();

//заменяет данные фрагмента

int Replace_Data(T* source, int delete_flag);

//возвращает координаты фрагмента

void Get_block_index(int *number_x, int *number_y);

//возвращает размеры фрагмента

void Get_block_dim(int *count_x, int *count_y);

//возвращает глобальные координаты элемента с индексом (0,0)

void Get_start_element_index(int *x, int *y);

//указывает системе, что фрагмент готов для обмена данными

int border_ready();

Библиотека написана на языке C++ с использованием средств MPI и POSIX THREAD. В структуре системы можно выделить 3 уровня:

1. **Транспортный**. На этом уровне формируются пакеты данных, которые отсылаются на другие вычислительные узлы средствами MPI. Также обеспечивается работа потока для асинхронного приёма данных. Для обмена данными между локальными фрагментами используются буферы.

2. **Организационный**. Этот уровень отвечает за разбиение пользовательских данных на фрагменты. Здесь хранится информация о расположении фрагментов на узлах, а также задаются связи, определяющие зависимости по данным и алгоритмы балансировки.

3. **Прикладной** уровень содержит интерфейс для доступа к данным, а также команды управления: задание параметров распределения, команды синхронизации и ручной балансировки.

Разбиение на уровни даёт широкие возможности для модификации библиотеки. Например, реализация новой структуры данных потребует внесения изменений только на 2-м уровне.

Тестирование

Для тестирования и оценки эффективности использовалась задача численного решения уравнения Пуассона явным методом на пятиточечном шаблоне. Результаты измерений вычислительных затрат представлены на графике (рис. 1).

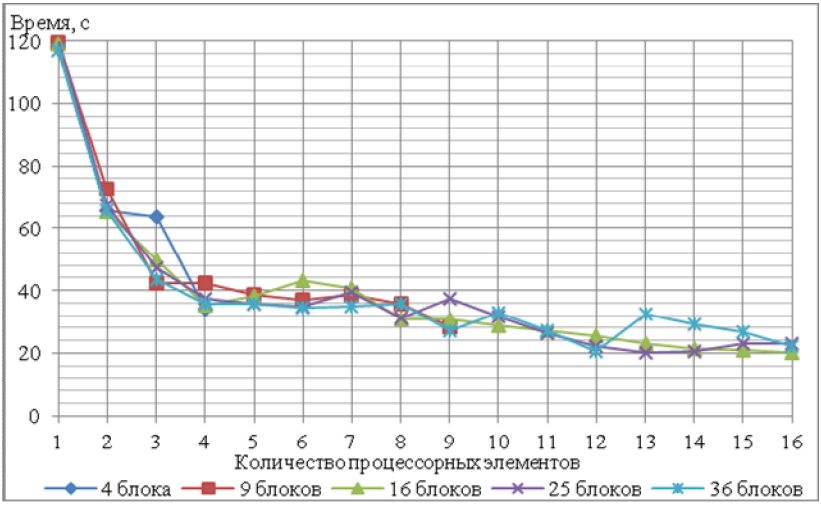


Рис. 1. Зависимость времени выполнения программы от числа процессорных элементов

В результате тестирования виден прирост скорости с увеличением числа вычислительных узлов, особенно это заметно при равномерной загрузке вычислителей: когда на процессорные элементы приходится равное количество блоков. Не удалось получить значительного ускорения за счёт увеличения количества фрагментов на узле. Также видна недостаточная масштабируемость системы. Это связано с накладными расходами на коммуникации между локальными фрагментами, а также отсутствием балансировщика.

Заключение

Разработана библиотека для работы с распределёнными данными. Реализована поддержка асинхронных коммуникаций. Библиотеку удобно применять для численных методов на сетках. Программисту предоставляются инструменты автоматического распределения и синхронизации данных. Его задача сводится к описанию алгоритма решения задачи на отдельных фрагментах. Асинхронные коммуникации

полностью реализуются системой и остаются невидимы для пользователя.

Система балансировки на данный момент находится в стадии разработки. За счёт перераспределения фрагментов ожидается повышение производительности, особенно на алгоритмах, где время фрагментов существенно варьируется и заранее неизвестно.

В перспективе планируются модификация и оптимизация алгоритмов передачи данных между фрагментами и реализация автоматических систем балансировки, также будет расширяться функционал для реализации более сложных численных методов, таких как метод частиц в ячейках.

Литература

1. **Бахтин В.А., Воронков А.В., Голубев А.С. и др.** Использование языка Fortran-DVM/OpenMP для разработки больших задач // Труды Всероссийской научной конференции «Научный сервис в сети Интернет: решение больших задач», сентябрь 2008 г., г. Новороссийск. – М.: Изд-во МГУ, 2008. – С. 185–191.
2. **Merlin J.H., Hey A.J.G.** An introduction to High Performance Fortran // Scientific Programming. – 1995. – Vol. 4. – P. 87–113.