Аннотация:

В области численного моделирования на мультикомпьютерах стоит проблема разработки эффективных параллельных программ (ПП). ПП могут работать эффективно только на определенных классах вычислительных систем (ВС). Для использования других ВС, вообще говоря, потребуются разные ПП. Возникает вопрос о переносимости ПП с учетом различных особенностей ВС. Системы автоматизации конструирования ПП позволяют обеспечивать такую переносимость, в том числе система LuNA. Она реализует подход фрагментированного программирования и исполняет фрагментированные программы (ФП). Не любая ПП является ФП, поэтому требуется преобразовать ее в ФП. Развитие отрасли знаний, позволяющей выполнять такое преобразование, может происходить при помощи исследования ручной разработки ФП для частных случаев среди задач численного моделирования. Данная работа посвящена такому исследованию для задачи моделирования процесса фильтрации двухфазной жидкости на основе законов сохранения в интегральной форме.

Тема работы:

Разработка эффективной фрагментированной программы решения задачи фильтрации двухфазной жидкости

Состав коллектива:

Студент: Спирин Виталий Андреевич, Факультет информационных технологий, кафедра параллельных вычислений, 4-й курс, группа 19203

Научный руководитель: Малышкин Виктор Эммануилович, зав. каф. параллельных вычислений ФИТ НГУ, г.н.с. ИВМиМГ СО РАН

Соруководитель: Перепёлкин Владислав Александрович, ст. преп. каф. параллельных вычислений ФИТ НГУ, н.с. ИВМиМГ СО РАН

Научное содержание работы:

1. Современное состояние проблемы и постановка задачи

В области численного моделирования на мультикомпьютерах / суперЭВМ стоит проблема разработки эффективных параллельных программ. Тут и далее эффективность понимается с точки зрения времени выполнения программы, расхода памяти, нагрузки на сеть и других аналогичных свойств.

Для обеспечения хорошей эффективности программист берет на себя ответственность за: настройку параллельной программы на доступные ресурсы, экономное использование памяти, равномерное распределение вычислительной нагрузки, и т. п. В таком случае

эффективность обеспечена (с точки зрения аппаратных характеристик) только для определенных классов вычислительных систем, а именно тех, под которые программист адаптирует программы.

Можно использовать системы автоматизации конструирования параллельных программ. Они выполняют конструирование эффективных программ по их высокоуровневым спецификациям. От программиста в данном случае не требуется вручную обеспечивать свойства, связанные с эффективностью. Но задача такого конструирования, в общем случае, является алгоритмически труднорешаемой, поэтому различные системы рассматривают лишь частные подходы к обеспечению эффективности. Развитие таких систем с этой точки зрения зачастую происходит при помощи исследования ручной разработки эффективных параллельных программ для частных случаев среди задач численного моделирования. Это позволяет накапливать и систематизировать опыт создания программ и улучшать системные алгоритмы. Также такая разработанная вручную программа может служить эталоном эффективности, с которым должна будет сравниваться система при конструировании аналогичной программы на вычислительные системы.

Данная работа посвящена такому исследованию для задачи моделирования процесса фильтрации двухфазной жидкости на основе законов сохранения в интегральной форме [1]. В качестве системы автоматизации конструирования параллельных программ рассматривается система фрагментированного программирования LuNA [2]. Она представляет интерес с точки зрения последующего улучшения системных алгоритмов. Чтобы иметь возможность произвести такие улучшения, требуется разработать вручную такую эффективную параллельную программу, которая была бы совместима с этой системой. Под совместимостью здесь понимается соответствие программы модели вычислений, которая используется в системе LuNA. Это означает, что высокоуровневая спецификация разработанной вручную программы должна быть общей с той, которая будет использоваться в системе LuNA. Отличие будет состоять в том, что система LuNA будет конструировать программу по спецификации автоматически, а вручную разработанная программа будет обеспечивать нефункциональные свойства непосредственно. Такая высокоуровневая спецификация в системе LuNA называется фрагментированным алгоритмом. А реализующая такой алгоритм программа, сконструированная автоматически или разработанная вручную, называется фрагментированной программой.

Целью работы ставится разработка эффективной фрагментированной программы решения задачи фильтрации двухфазной жидкости.

Для достижения цели решаются следующие задачи: анализ решения задачи фильтрации с точки зрения параллельной реализации, выработка требований к программе,

разработка схемы эффективной фрагментированной программы, реализация схемы и проведение тестирования.

2. Подробное описание работы, включая используемые алгоритмы.

a. Анализ решения задачи в существующей многопоточной реализации.

В формуле представлена матрично-векторная система, полученная в статьях [3,4], посвященных решению задачи: $[B^TD^{-1}B+\varepsilon C]*u=f$. Здесь матрица D – это симметричная блочно-диагональная матрица с трехдиагональными блоками, $C=e*e^T$ – симметричная квадратная плотная матрица, матрица B – прямоугольная (количество строк примерно в 3 раза превышает количество столбцов) и разреженная (в каждой строке матрицы по 1-2 элемента и в каждом столбце по 3-6 элементов). В указанных статьях доказывается положительная определенность матрицы $[B^TD^{-1}B+\varepsilon C]$ для любого $\varepsilon>0$, что дает возможность решать данную систему с использованием итерационного метода сопряженных градиентов.

Данный метод сопряженных градиентов был рассмотрен в существующей многопоточной реализации. Для реализации параллелизма она использует только библиотеку OpenMP. На основе тестов и анализа кода, выполнение именно метода сопряженных градиентов занимает большую часть времени. Одна итерация метода сопряженных градиентов состоит из комбинации векторных и матрично-векторных операций. Среди векторных операций присутствуют: скалярное произведение, вычисление евклидовой нормы, составление линейной комбинации векторов, копирование векторов. Матрично-векторными операциями являются: умножение матрицы на вектор, умножение транспонированной матрицы на вектор и умножение обратной матрицы на вектор.

b. Требования к фрагментированной программе.

Были составлены приведенные ниже требования к фрагментированной программе. Они разделены на две группы.

Первой группой являются требования к фрагментированной программе с прицелом на LuNA:

- Процедуры, выполняющие вычисления над фрагментами данных, не должны иметь побочных эффектов: обращаться к глобальным переменным, изменять входные фрагменты.
- Использовать примитивные типы данных языка программирования C++.

Второй группой являются требования, связанные с нефункциональными характеристиками:

- Программа должна решать задачу на разном количестве процессов и потоков так же, как и другие реализации.
- Размеры фрагментов для одного и того же вектора или матрицы на разных процессах должны отличаться не более чем на 1.
- Программа не должна делать лишних коммуникаций при передаче фрагментов между процессами. Например, если в конце одной операции фрагменты вектора собираются вместе, а потом в начале следующей операции вектор опять фрагментируется, то эти две коммуникации можно считать лишними.
- Полученная программа должна работать достаточно эффективно.

c. Фрагментация каждой функции

Для реализации параллельных версий векторных процедур используются стандартные подходы: после фрагментации вектора, процедуры выполняются над фрагментами независимо; для некоторых векторных процедур, например, скалярного произведения, в конце выполняется редукция. Теперь рассмотрим матрично-векторные операции.

Операция умножения матрицы на вектор. На рисунке 1 показано ее визуальное представление с акцентированием внимания (при помощи цвета) на зависимость между строками матрицы и компонентами выходного вектора.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n-1} & a_{2,n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n-1} & a_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \dots \\ f_m \end{bmatrix} = \begin{bmatrix} a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n \\ a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n \\ \dots \\ a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,n}x_n \end{bmatrix}$$

Рисунок 1 – Наглядное представление умножения матрицы на вектор

Для получения j-ого элемента результирующего вектора достаточно рассмотреть j-ую строку матрицы A. В программе матрица разбивается на фрагменты, каждый из которых содержит какое-то множество строк матрицы; входной вектор хранится полностью, как один фрагмент. Перед выполнением умножения входной вектор собирается с помощью коммуникаций, так как он разбивается на фрагменты.

Операция умножения транспонированной матрицы на вектор. Она во многом схожа с предыдущей операцией, но есть важное отличие в обходе матрицы. Так как матрица разреженная, и она хранится в формате CSR, то тут важно, является ли матрица транспонированной или нет. Ее визуальное представление с аналогичным акцентированием внимания на зависимость между столбцами матрицы и компонентами выходного вектора содержится в рисунке 2.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n-1} & a_{2,n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n-1} & a_{m,n} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_m \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \dots \\ f_{n-1} \\ f_n \end{bmatrix} = \begin{bmatrix} a_{1,1}x_1 + a_{2,1}x_2 + \dots + a_{m,1}x_m \\ a_{1,2}x_1 + a_{2,2}x_2 + \dots + a_{m,2}x_m \\ \dots \\ a_{1,n-1}x_1 + a_{2,n-1}x_2 + \dots + a_{m,n-1}x_m \\ a_{1,n}x_1 + a_{2,n}x_2 + \dots + a_{m,n}x_m \end{bmatrix}$$

Рисунок 2 – Наглядное представление умножения транспонированной матрицы на вектор

Теперь j-ый элемент результирующего вектора зависит не от j-ой строки, а от j-ого столбца матрицы. Разбиение матрицы в фрагментированной программе происходит по строкам. Это позволяет также разбить входной вектор на фрагменты с соответствующими размерами. Выполнение операции над каждым фрагментом в качестве результата даёт частичный вектор, и для получения полного результата выполняется их редукция.

Последняя матричная операция выполняет *умножение обратной матрицы на вектор*. В данной задаче матрица является блочно-трехдиагональной, что позволяет выполнять умножение при помощи решения СЛАУ методом прогонки. Он применяется к каждому трехдиагональному блоку матрицы независимо. Это единственный способ произвести вычисления параллельно, так как сам метод прогонки является последовательным.

3. Полученные результаты.

Было проведено тестирование. Для сравнения, помимо многопоточной реализации, также присутствуют две MPI реализации, одна из которых не реализует параллелизм на уровне потоков, а вторая была разработана без учета теории фрагментированного программирования и с использованием другого подхода к распределению вычислений.

Результаты работы программ на кластере представлены ниже на рисунке 3.

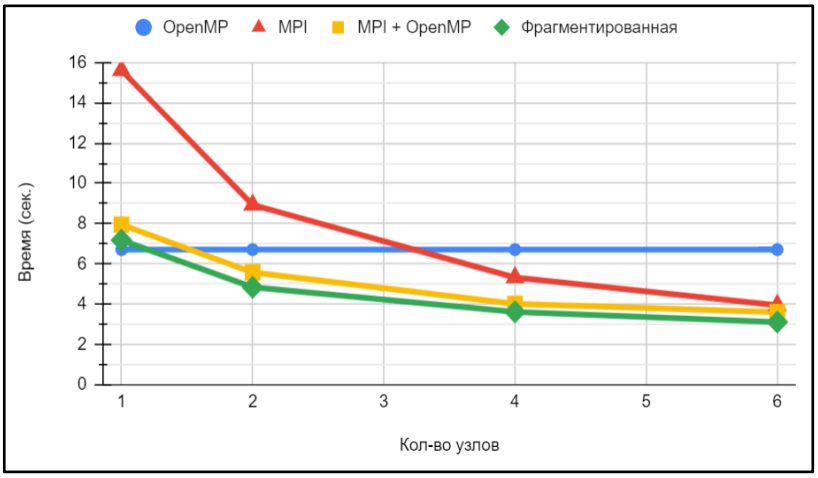

Рисунок 3 – Зависимость времени от количества узлов

MPI реализации на нескольких узлах дают выигрыш по сравнению с OpenMP реализацией. Более того, фрагментированная реализация получилась быстрее всех.

Рассмотрим понятия ускорения и эффективности распараллеливания. Ускорение определяется следующей величиной: $S_N=T_1/T_N$, где T_1 – время работы программы на 1 узле, а T_N – время работы параллельной программы на N узлах. В качестве T_1 выбирается время работы многопоточной реализации. Далее, с использованием этой величины, определяется эффективность распараллеливания: $E_N=S_N/N$, где S_N – ускорение параллельной программы с использованием N узлов.

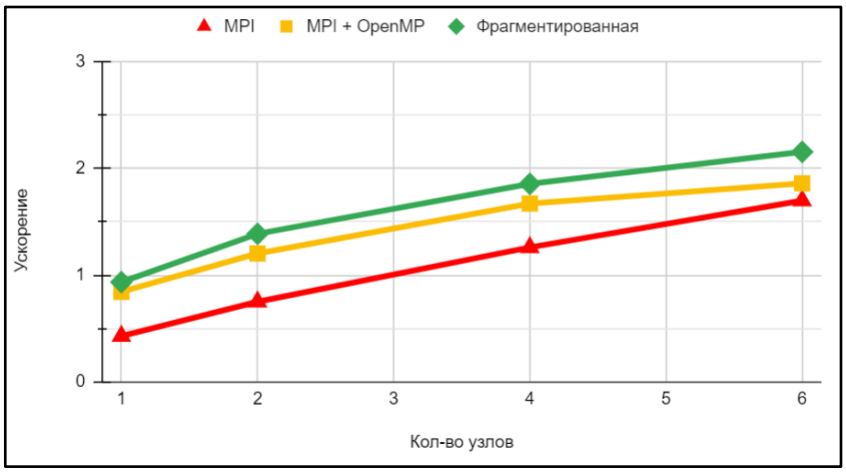

Рисунок 4 – Зависимость ускорения от количества узлов

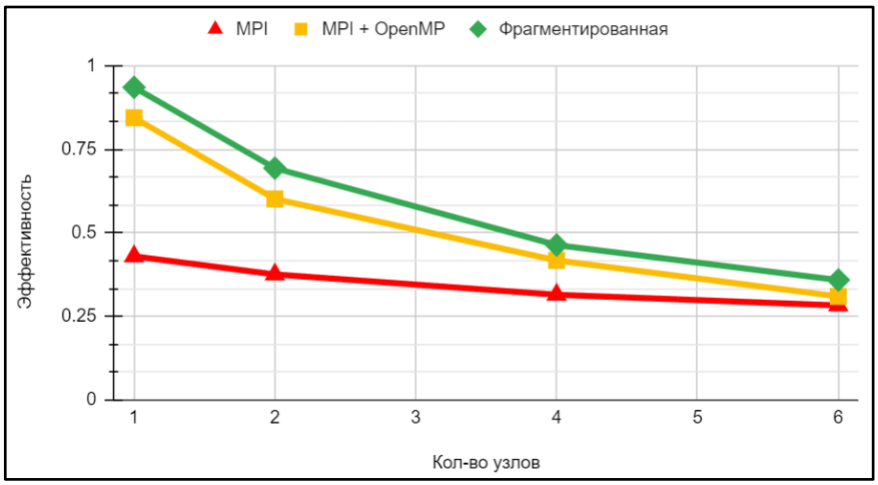

Рисунок 5 – Зависимость эффективности распараллеливания от количества узлов

Для созданной фрагментированной реализации полученная эффективность распараллеливания является удовлетворительной для решаемого класса задач.

1. Иванов М.И., Кремер И.А., Лаевский Ю.М. Моделирование процесса фильтрации двухфазной жидкости на основе законов сохранения в интегральной форме, 2020. - 35 с

2. LuNA: Language for Numerical Algorithms [Электронный ресурс] / – Режим доступа: https://ssd.sscc.ru/luna
3. Ivanov M. I., Kremer I. A., Laevsky Y. M. On the streamline upwind scheme of solution to the filtration problem //Sib. Electronic Math. Reports. – 2019. – Т. 16. – С. 757-776.
4. Ivanov M. I., Kremer I. A., Laevsky Y. M. On wells modeling in filtration problems //Sib. Electronic Math. Reports. – 2019. – Т. 16. – С. 1868-1884.

Эффект от использования кластера в достижении целей работы.

Удалось с использованием кластера проверить работоспособность распределенной фрагментированной программы решения задачи фильтрации двухфазной жидкости. Результаты смогли показать хорошую эффективность.

Перечень публикаций, содержащих результаты работы

1. Всероссийская летняя XXXIX молодежная Школа-конференция по параллельному программированию с международным участием (г. Новосибирск, 4 – 15.07.2022 г.): Тез. докл. / отв. ред. д-р техн. наук, проф. В. Э. Малышкин. Новосиб. гос. унт. – Новосибирск : ИПЦ НГУ, 2023
2. Спирин В. А. Разработка эффективной фрагментированной программы решения задачи фильтрации двухфазной жидкости / В. А. Спирин // Информационные технологии. Цифровые технологии нефтегазовой индустрии : Материалы 61-й Междунар. науч. студ. конф. 17-26 апреля 2023 г. / Новосиб. гос. ун-т. — Новосибирск : ИПЦ НГУ, 2023
3. Кудрявцев А.А., Малышкин В.Э., Нуштаев Ю.Ю., Перепёлкин В.А., Спирин В.А. Эффективная фрагментированная реализация краевой задачи фильтрации двухфазной жидкости // журнал "Проблемы информатики", 2023, №2

Приложение. Табличное представление результатов тестирования

Таблица 1 – Время работы (сек.) программ на разном количестве узлов

Кол-во узлов	OpenMP	MPI	MPI + OpenMP	Фрагмент-ная
1	6.71136	15.6164	7.95094	7.17733
2	6.71136	8.93176	5.58007	4.84004

4	6.71136	5.32752	4.01865	3.61975
6	6.71136	3.95258	3.60899	3.11598

Таблица 2 – Ускорение и эффективность программ

Кол-во узлов	MPI Ускорение / Эффективность	MPI + OpenMP Ускорение / Эффективность	Фрагмент-ная Ускорение / Эффективность
1	0.42976 / 0.42976	0.84410 / 0.84410	0.93508 / 0.93508
2	0.75140 / 0.37570	1.20274 / 0.60137	1.38663 / 0.69332
4	1.25975 / 0.31494	1.67005 / 0.41751	1.85409 / 0.46352
6	1.69797 / 0.28299	1.85962 / 0.30994	2.15385 / 0.35898