Распределенный интерпретатор сложных алгебраических выражений

Дмитриева О.А.
Донецкий национальный технический университет
dmitriv@r5.dgtu.donetsk.ua

Abstract

Dmitrieva O.A. The distributed interpreter of complex algebraic expressions. *In this article the questions of the distributed interpreter of complex arithmetic expressions development are considered. As far as the calculable processes based on interpretation of expressions, are extremely resource-intensive and take considerable time during realization, the attempt of complex rated problems parallelization is carried out. Paralleling of computations process gave a considerable increase to productivity. The algorithm of assembly of the received results, which allows to carry out arrangement in a random order, is developed. The phased multilevel system of paralleling is realized.*

Введение

Целью данной работы является создание распределенного интерпретатора сложных арифметических выражений. Поскольку вычислительные процессы, основанные на интерпретации выражений, являются крайне ресурсоемкими и требуют значительного времени при реализации [1], в данной работе осуществлена попытка распараллеливания сложных расчетных задач.

Преимущества распределенных сетевых вычислений по сравнению с суперкомпьютерными очевидны. Очень многим пользователям иногда требуется решать сложные вычислительные задачи, но специальные ресурсы, приобретенные для непериодического использования, как правило, не окупают себя. Кроме того, те же суперкомпьютеры имеют свойство устаревать. Инсталлировав такой суперкомпьютер, администрация вынуждена постоянно поддерживать его в форме - обслуживать, ремонтировать, модернизировать, обучать персонал. Очевидно, что распределенные вычисления лишены этих недостатков.

Но при этом следует отметить, что серьезным препятствием для организации распределенного синтаксического разбора арифметических выражений является тот факт, что известные стековые методы (алгоритмы Бауэра и Замельзона, Рутисхаузера [2], обратная польская нотация [3]) не поддерживают параллельную реализацию разбора. Например, обратную польскую нотацию невозможно разбить на независимые части. Алгоритмы Замельзона и Рутисхаузера работают только с выражениями, представленными полными скобочными структурами (порядок действий задается расстановкой скобок, неявное старшинство операций при этом не учитывается).

Для всех указанных алгоритмов разбора обработка выражения осуществляется последовательно, основываясь на присваивании каждому очередному символу из строки номера уровня по некоторому правилу.

Для параллельной обработки выражения требуется соблюдение таких требований, как возможность разделения выражения на куски произвольной или равной длины и их независимая обработка, возможность сбора полученных результатов в единый ответ, независимость результата от очередности обработки частей выражения. Поэтому в работе предлагается алгоритм интерпретации выражений [4], на основе которого проводятся сетевые распределенные вычисления.

Разработанный метод позволяет с легкостью находить независимые и логически завершенные участки выражений, при этом максимальное внимание уделено обработке лексической свертки на логически незавершенных участках.

Парсинг введенного выражения

Для реализации распределенных вычислений был разработан и реализован метод представления и обработки выражений. Предлагаемый алгоритм базируется на подходе, предложенном Рутисхаузером, но имеет некоторые существенные отличия. Основное отличие состоит в отсутствии необходимости поиска оператора, который имеет наибольший глобальный индекс уровня. Разработанный алгоритм способен осуществлять расчеты на локальном максимуме среди двух соседних операторов, что является серьезным упрощением разбивки выражения на части. Также данный подход разрешает значительно уменьшить время обработки выражения. Если алгоритм Рутисхаузера для разбора и интерпретации выражения осуществляет N-1 проходов, где N - число операторов, то предлагаемый алгоритм потребует n проходов, где n - число уровней в выражении.

Работа алгоритма складывается из следующей последовательности шагов:

1. На этапе сканирования выражения выполняется расстановка уровней (приоритетов).
2. Выполняется разбиение лексической свертки. Количество составляющих разбиения определяется количеством доступных процессоров в сети.
3. В каждом процессоре осуществляется поиск элементов строки и проверяется возможность выделения тройки (для бинарных операций) или двойки (для унарных операций). В случае положительного результата такого поиска выполняется операция, и результат вычисления обозначается вспомогательной переменной.
4. Из исходной строки удаляются выделенные тройки (двойки), а на их места помещаются вспомогательные переменные, обозначающие результат.
5. П.п.3 - 4 выполняются до тех пор, пока происходят изменения во входных строках процессоров сети.
6. Осуществляется сборка всех частично распознанных конструкций с последующим «доразбором». Результатом успешной интерпретации выражения является значение, обозначающее общий результат вычислений. В противном случае выдается диагностическое сообщение об ошибках в исходной записи.

Также в алгоритме применена оптимизация для циклических вычислений, которая состоит в упрощении постоянной части выражения, участвующей в вычислениях. Оптимизация производится таким образом, что она не влияет на полученные результаты, но позволяет значительно сократить время и количество машинных операций, необходимых для интерпретации выражения. Суть оптимизации состоит в сведении сложного выражения, которое условно можно подразделить на две составляющие – переменную и постоянную, к некоторому упрощенному выражению с вычисленной постоянной частью, которая заменяется константой. Например, в выражении ∫(sin(2*x)+cos(2*Pi))dx присутствует неизменная часть cos(2*Pi), вычисление которой требует много времени, особенно если применяются численные методы высокого порядка точности. После проведения преобразований эта часть будет заменена константой, которая равна значению cos(2*Pi). Это позволяет в сотни раз сократить объем вычислений. Таким образом, в работе создан интерпретатор, способный разбирать и интерпретировать выражения, распределенные между вычислительными элементами сети.

Принадлежность каждого оператора к своему уровню действует глобально и сохраняется в любой части выражения. Во время синтаксического разбора в массив заносится код оператора вместе с его глобальным уровнем. Также генерируется некоторая вспомогательная информация, которая не принимает непосредственного участия в вычислениях, но является необходимой для поиска возможных синтаксических ошибок и упрощения работы пользователя с интерфейсом программы. В частности, вспомогательной информацией является положение оператора, количество символов, занимаемых оператором.

В процессе интерпретации происходит проверка на наличие корректных данных для текущего оператора и отсутствие в строке операторов с большим уровнем. Если необходимые данные отсутствуют или оператор соседствует с оператором более высокого уровня, вычисления не проводятся, рассматривается следующий оператор. Если оба эти условия выполняются, то происходит вычисление, а вместо оператора и его аргументов в текущую позицию заносится результат промежуточных вычислений. По завершению прохода, выражение рассматривается снова, до тех пор, пока в результате разбора выражения не останется одно число. Это число и будет являться результатом вычислений.

Данный метод разрешает довольно легко обрабатывать и выделять операцию унарного минуса, на что неспособно множество других алгоритмов. Также в процессе решения довольно легко выделяются синтаксические ошибки и указывается их точное местоположение. Для этого достаточно осуществлять проверку на наличие изменений при очередном проходе выражения. Если выражение осталось неизменным и не состоит из единственного числа, значит выражение содержит ошибку.

В этом случае выделить ошибку довольно просто. Надо найти оператор с наибольшим уровнем, не имеющим аргументов. Это и будет оператор, который логически не связан с выражением.

Введенное выражение хранится в структуре, которая имеет вид:

```
type
data = record
Val:extended;
pos:int64;
pr:byte;
tp:integer;
len:byte;
end;
```

Рисунок 1 - Структура хранения введенного выражения

где

Val- значение (только для констант и идентификаторов);

pos - позиция в выражении, данная переменная не принимает участие в расчетах и является вспомогательной при реализации пользовательского интерфейса;

pr – приоритет выполнения (большее значение соответствует более высокому приоритету);

tp – тип оператора;

len - длина оператора, данная переменная также не принимает участие в расчетах и является вспомогательной при реализации пользовательского интерфейса.

При синтаксическом разборе заполняется массив типа приведенной выше структуры. Заполнение массива выполняется однопроходно. Функция заполнения также осуществляет распознавание структуры, в которой сохраняются введенные операторы. Данная процедура в процессе распознавания осуществляет очистку введенного выражения от посторонних символов (например, удаляет комментарии, множественные пробелы), которые не являются операторами для данного алгоритма. Во введенных константах осуществляется поиск и возможное исправление ошибок, а также перевод чисел введенных в двоичной, восьмеричной и шестнадцатиричной системах счисления в десятичную систему, одновременно при переводе выполняется проверка правильности записи числа в этой системе счисления.

Процесс распознавания и заполнения состоит из нескольких этапов. На первом этапе производится очистка массива операторов от предшествующих вычислений. Затем все введенное выражение переводится в нижний регистр для облегчения поиска и распознавания операторов. При этом удаляются пробелы и символы, не являющиеся операторами. После этого производится распознавание операторов и заполнение массива, в котором сохраняются распознанные операторы и константы.

Рисунок 2 - Экранная форма интерпретации поиска корней функций

Для обозначения начала и конца распознанных данных употребляется специально зарезервированное значение, которое заносится в дополнительное поле структуры, сохраняющей значение приоритета выполнения оператора. Зарезервированное значение равняется 127. Оператор в силу особенностей алгоритма с таким приоритетом никогда не будет вычислен и, по сути, первый и конечный элементы массива выполняют роль пустых операторов - меток.

Данная процедура распознает следующие арифметические операторы: +, -, *, /, Sin, Cos, Tan, ArcSin, ArcCos, ArcTan, Ln, Lg, Log, возведение в степень, унарный минус, и скобки. Результаты такого распознавания и интерпретации получены в разработанной распределенной системе, одна из экранных форм которой приведена на рис. 2

Распараллеливание вычислений

В процессе распараллеливания происходит разбивка задачи на приблизительно равные по трудоемкости задачи между всеми клиентами. Для балансировки нагрузки количество подзадач распознавания значительно превышает количество клиентов, что позволяет управлять распределением заданий. Распараллеливание процесса расчета интегралов, поиска корней нелинейных уравнений, нахождения векторов неизвестных решений линейных и нелинейных систем уравнений дало значительный прирост производительности. Также значительное ускорение процесса интерпретации получено от того, что синтаксический и семантический разборы частично могут осуществляются на клиентских компьютерах. Также клиентские компьютеры загружены расчетами. Этот факт обеспечивает в некоторых случаях прирост производительности в несколько раз.

Дополнительно был разработан алгоритм сбора полученных результатов, который осуществляет «доразбор» полученных частичных сверток и формирует выполняемую последовательность лексем.

Работу многоуровневой системы распараллеливания можно описать следующей последовательностью шагов. Вначале производится расчет всех интегралов, найденных в выражении. После чего осуществляется попытка упрощения выражения, что позволяет избежать применения итерационных методов расчета для неизменяемых частей выражений. Потом выражение снова разбивается на порции и передается клиентам. Окончательная сборка начинается уже при поступлении первого результата от любого из клиентов. При получении последнего результата собранное выражение передается к одному из клиентов, где производится его окончательное вычисление.

Распараллеливание проводится с помощью распределенных сетевых ресурсов. Передача частей задачи и результатов производится с помощью локальной сети. Использование этого подхода обосновано тем фактом, что, во-первых, не все задачи можно разбить на абсолютно независимые этапы,

пригодные к обработке на изолированных друг от друга компьютерах. К тому же возникает проблема ввода исходных данных и считывания результатов, которая легко разрешима в локальных сетях. К преимуществам использования локальной сети для процесса распределенных вычислений также следует отнести относительно низкую стоимость эксплуатации и высокую скорость передачи данных. Конечно, глобальная сеть Интернет имеет более привлекательные ресурсы и возможности, но в силу своей масштабности при работе с ею тяжело гарантировать доставку пакета с промежуточными результатами в приемлемые промежутки времени.

Результаты тестирования

В процессе проведенных экспериментов исследовались зависимости времени, необходимого на обработку сложных выражений, от количества вычислительных элементов (ВЭ), задействованных в системе и от сложности решаемой задачи (определялась количеством операций с плавающей точкой). Рассматривались численные алгоритмы поиска корней нелинейных уравнений, находились вектора неизвестных при численном решении систем линейных и нелинейных алгебраических уравнений, также для тестирования были привлечены задачи численного интегрирования. Результаты тестирования алгоритма на поиске корней нелинейных уравнений приведены на рис. 3.

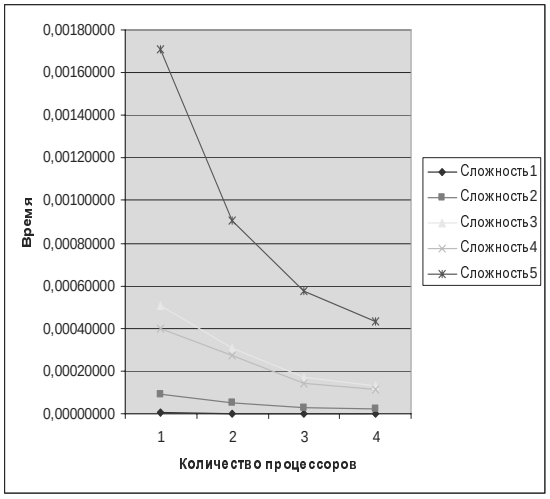

Рисунок 3 - Зависимость времени поиска корней нелинейных уравнений от сложности функции и количества ВЭ в системе

Как видно из рис. 3, с ростом количества ВЭ в локальной сети время, необходимое на обработку выражения существенно сокращается. При этом ускорение тем больше, чем выше сложность введенного выражения. Такой рост ускорения определяет и второй показатель параллелизма – коэффициент эффективности, который в нашем случае близок к потенциальному единичному значению (рис. 4). Коэффициент эффективности определялся из следующего соотношения

$$E=t*N/T, \quad (1)$$

Где t – лучшее время реализации алгоритма на одном вычислительном элементе;

N – количество вычислительных элементов;

T – время реализации алгоритма на N ВЭ.

Более высокую эффективность, достигнутую на сложных задачах, можно объяснить сокращением доли времени, расходуемого на передачу информации между ВЭ по сравнению с общим временем, необходимым для вычисления.

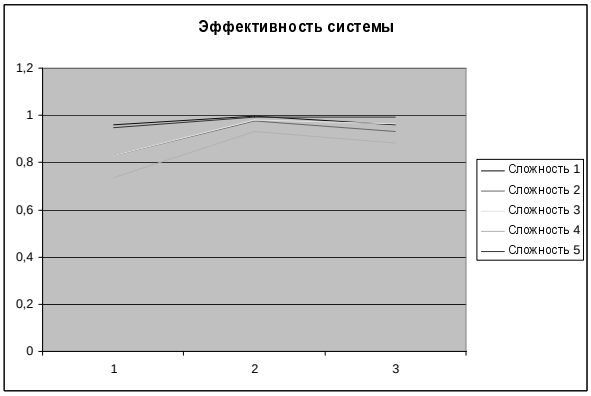

Рисунок 4. - Зависимость эффективности поиска корней нелинейных уравнений от сложности функции и количества ВЭ в системе

Показатели параллелизма, полученные при тестировании алгоритмов поиска векторов неизвестных при решении систем уравнений и при численном интегрировании, сохраняют те же зависимости, что и приведенные на рис. 3-4. При этом необходимо отметить хорошую результативность алгоритмов. Кроме того, разработанная распределенная система распознавания и интерпретации выражений может быть использована как автономно, так и в качестве встроенной подсистемы в математических средах, что может значительно увеличить их производительность за счет распределенной обработки.

Литература

1. Ахо А., Ренди С., Ульман Дж. Компиляция. – М.:Мир, 2002. – 834 с.
2. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов, - М.: Мир, 1979.- 654 с.
3. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 536 с.
4. Дмитриева О.А., Иванченко А.В. Параллельная интерпретация арифметических выражений.//Тезисы докладов международной научно-практической конференции «Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій». – Запорожье: ЗНТУ, 2006. С. 132-134.