УДК 004.43

СТРУКТУРЫ ДЛЯ АСИНХРОННЫХ ВЫЧИСЛЕНИЙ В ЯЗЫКЕ “ПИФАГОР”

Матковский И.В.
научный руководитель докт. техн. наук Легалов А.И.
Сибирский Федеральный Университет

Параллельные вычисления на данный момент являются невероятно популярными; имеющиеся проблемы с повышением производительности отдельных элементов компенсируются возможностью делить вычислительную задачу между несколькими элементами. Существует большое количество подходов к построению параллельных вычислительных систем (ПВС); огромное разнообразие архитектур ПВС позволяет подбирать системы под конкретные ситуации, однако и недостатков имеет немало. Наиболее серьезным недостатком является усложнение разработки кроссплатформенных приложений – перенос программы с одной ПВС на другую в чистом виде либо невозможен, либо сопряжен со снижением эффективности вычислений. Решать данную проблему переписыванием программы под конкретную архитектуру зачастую оказывается либо слишком дорого, либо слишком сложно.

Существует ряд методов, позволяющих разрабатывать приложения, легко переносимые с одной ПВС на другую; одним из таких методов является архитектурно-независимое программирование (АНП). АНП построено на разработке программы для абстрактной системы, поддерживающей максимальный параллелизм и неограниченное количество потенциально необходимых ресурсов. Разработанный и отлаженный для такой системы алгоритм в условиях целевой ПВС исполняется специальным модулем-интерпретатором. Интерпретатор знаком с особенностями данной ПВС и способен исполнить программу в соответствии с этими особенностями. Разработка программы без привязки ей к конкретной ПВС производится заметно проще.

Функциональные языки программирования по сути своей хорошо подходят для воплощения принципов АНП; полезным также оказалось разделение управляющей и информационной составляющих графа. “Пифагор” [1,2] представляет собой язык функционально-потокового параллельного программирования, ориентированный на архитектурно-независимое программирование и представляющий программы в виде сочетания реверсивного информационного графа (РИГ) и управляющего графа (УГ).

Принципы устройства асинхронных списков.

Нередко в ходе организации параллельных вычисления приходится исходить из того, что различные модули, поставляющие данные для массовых операций, могут работать с разной скоростью. Как следствие, данные могут поступать в итоговой обработчик с большими задержками. Обработка данных лишь после поступления их в полном объеме проста с точки зрения управления, однако малоэффективна с точки зрения временных затрат. Зачастую начинать вычисления можно после поступления части исходных данных. Для работы с асинхронно поступающими данными в язык “Пифагор” была введена концепция асинхронных списков [3].

Асинхронные списки хранят поступившие в них фрагменты данных в порядке поступления (аналогично очереди данных). С точки зрения внешнего мира, список может принимать два состояния: он либо абсолютно пуст, либо содержит хотя бы один элемент. Точное количество элементов, порядок их поступления в асинхронный список и временные задержки между поступлением элементов внешнему миру неизвестны.

Асинхронный список из N элементов AN можно представить выражением

$$AN=(H, AN-1)$$

где H – голова, первый элемент списка,

AN-1 – “хвост”, асинхронный список, составленный из всех кроме первого элементов исходного. Асинхронный список A0 состоит из одного “пустого” элемента и “хвоста” не имеет.

Данное представление позволяет в каждый конкретный момент времени обрабатывать лишь один элемент данных; остальная часть списка будет обработана позже, по мере формирования.

Для работы с асинхронным списком необходимы две базовых операции:

- Получение головного элемента списка; в языке “Пифагор” реализуется операцией выборки первого элемента X:1. Так, головной элемент асинхронного списка A будет результатом вычисления выражения A:1.
- Получение хвоста списка; в языке “Пифагор” реализуется операцией исключающей выборки первого элемента X:-1. Так, хвост асинхронного списка A будет результатом вычисления выражения A:-1.

Пример использования асинхронных списков.

Приведенная ниже программа показывает, как асинхронные списки могут быть применены для суммирования элементов одномерного массива.

```
Async_Sum<< funcdef A
{
    x1<< A:1;
    tail_1<< A:-1;
    [(tail_1:[IsEmptyDataList, IsNotEmptyDataList]:?]^
    (
            x1,
            {
            block {
            x2<< tail_1:1;
            s<< (x1,x2):+;
            tail_2<< tail_1:-1;
             [(tail_2:[IsEmptyDataList,  IsNotEmptyDataList]:?]^
            (
                    s,
                    { asynch( tail_2:[], s):A_VecSum }
            ):. >>break;
            }
    }):. >>return;
}
```

В качестве входного аргумента A программа принимает асинхронный список из заранее неизвестного числа элементов. Для начала выделяется первый (головной) элемент асинхронного списка x1 (получаемый командой выборки A:1); после этого проверяется хвост списка tail_1 (получаемый командой выборки-исключения A:-1). Рассматриваются следующие альтернативы:

1. Хвост списка tail_1 пуст; следовательно, асинхронный список A состоит из одного элемента x1, который и следует вернуть в качестве результата суммирующей функции.
2. Хвост списка tail_1 не пуст. Тогда выделяем его головной элемент x2 и суммируем его с x1. Проверяем “хвост хвоста” tail_2. Возможны следующие варианты:
    a. tail_2 пуст; значит, A содержал только элементы x1 и x2. Возвращаем их сумму в качестве суммы списка.
    b. tail_2 не пуст. Создаем новый асихронный список, в который помещаем сумму x1 и x2 и tail_2; вызываем для него рекурсивно функцию суммирования.

Асинхронный список как информационная структура.

Классический функциональный подход к разделению массивов на пары [голова; список] оказался эффективным и в случае с асинхронными списками данных. Основной проблемой в случае с формированием асинхронных списков стала необходимость поддерживать связи между различными “хвостами”. Поступление новых элементов, с точки зрения РИГ, производится в самый первый, наиболее длинный асинхронный список; вместе с тем, элементы эти должны появляться и в различных “хвостах” списка.

В конечном итоге структура данных, предназначенная для хранения асинхронного списка, была организована следующим образом:

```
Class AsyncListValue
{
    Value* head;
    AsyncListValue *tail;
}
```

Для работы с асинхронным списком необходимо обеспечить возможность инициализировать его, добавлять в него элементы, производить выборку “головы” и выборку “хвоста”.

При создании очередного асинхронного списка “головному” и “хвостовому” его элементам присваивается значение Null.

Добавление элементов производится рекурсивно; если в данном асинхронном списке уже имеется головной элемент, то добавление производится в хвост. Если головного элемента нет, то им становится добавляемый, а хвост превращается в новый – пока что пустой – асинхронный список.

Выборка головы в данной схеме становится тривиальной операцией – достаточно просто вернуть указатель на головной элемент. Так же тривиально производится и возврат хвоста. Поскольку даже после выделения хвоста указатель на него будет храниться в “оригинальном”, изначальном асинхронном списке, проблем с добавлением новых элементов у системы возникнуть не должно и выделенные хвосты будут дополняться новыми элементами по мере их поступления в оригинальный.

Данная схема организации асинхронного списка никак не привязана к конкретному РИГ или УГ. В системах с общей памятью передача асинхронного списка в качестве аргумента во внешнюю функцию или возврат асинхронного списка в качестве результата вычисления функции трудностей не представляет. В случае с кластерными системами процесс поддержания целостности асинхронных списков затруднится – простого хранения указателей уже будет недостаточно. В таком случае оригинальным асинхронным спискам придется хранить перечень всех выделенных из них хвостов и, по мере необходимости, взаимодействовать с этими хвостами через центральный менеджер интерпретирующей системы.

Управление асинхронными вычислениям.

Трудности, возникающие при порождении управляющих сигналов асинхронных списков, также обусловлены прежде всего необходимостью поддерживать связь между оригинальными спискам и отделенными от них хвостами. В случае с наличием только одного асинхронного списка работа управляющей системы тривиальна – сигнал о готовности данных поступает в УГ сразу после того, как в асинхронный список попадает первый элемент. Такой асинхронный список может проходить и через операцию выделения головы, и через операцию выделения хвоста.

Проблемы возникают при дальнейшей обработки выделенных хвостов – хвосты N-го уровня будут готовы к обработке только после того, как в оригинальный асинхронный список поступит N элементов. Сигнал о готовности, однако, необходимо выдавать вершинам, которые могут отстоять от оригинальной вершины формирования списка на достаточно большое расстояние – и даже находиться в других функциях.

Проблема решается за счет введения специальных маркеров асинхронных списков. Маркер передается по РИГ функции на правах обычного фрагмента данных. Как только в

одной из вершин РИГ возникает необходимость обработать асинхронный список-хвост, между соответствующей вершиной УГ и вершиной УГ, в которой был сформирован данный асинхронный список, образуется динамическая связь. Специальная пометка на этой связи помогает уточнить, какой именно по счету сигнал с этой связи надлежит пропустить. Необходимая для формирования управляющей связи информация – номер “оригинальной вершины”, “уровень” хвоста – хранится в маркере асинхронного списка. Аналогичный механизм используется в языке для обработки задержанных списков [4], однако их маркеры имеют несколько иную структуру.

Список литературы

1. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. № 1 (10). С.71-89.
2. Редькин А.В., Легалов А.И. Событийное управление выполнением функционально-потоковых параллельных программ // Научный вестник НГТУ. 2008. № 3 (32). С. 111-120.
3. Легалов А.И., Редькин А.В. Расширение асинхронного управления по готовности данных / Труды III Международной конференции «Параллельные вычисления и задачи управления» PACO’2006. — ISBN 5-201-14990-1 / М.: Институт проблем управления им. В.А. Трапезникова РАН, 2006. С 1272-1281. (Электронное издание)
4. Матковский И.В. Интерпретация списочных структур в функциональном языке потоково-параллельного программирования / Многоядерные процессоры, параллельное программирование, ПЛИС, системы обработки сигналов: сб. ст. / [сост. А.В. Калачев, В.В. Белозерских]. — Барнаул: Барнаул, 2013 — 176 с.