Характеристики типовых алгоритмических структур

Вад.В. Воеводин

Существует предположение, что во многих предметных областях значительное множество задач построено на основе небольшого числа алгоритмических структур. Примерами таких структур могут служить перемножение матриц, скалярное произведение, БПФ или задача N тел. Поскольку типовые структуры лежат в основе многих задач, то, поняв, какими свойствами обладают эти структуры, можно будет сделать вывод о свойствах и характеристиках самих задач. В данной работе выделяются основные этапы процесса отображения задачи на некоторую аппаратную платформу и на каждом этапе описываются характеристики, которые определяют эффективность отображения. На примере типовой структуры «триада» показано влияние ее особенностей на получаемую эффективность отображения.

1. Введение

Существует предположение, основанное как на нашей собственной практике, так и на опыте других, что во многих предметных областях значительное множество задач построено на основе небольшого числа вычислительных структур [1-3]. В этих структурах сосредоточена основная вычислительная часть задачи. Подобные структуры мы будем называть типовыми алгоритмическими структурами. Примерами таких структур могут служить перемножение матриц, скалярное произведение, суммирование элементов массива, БПФ или задача N тел. Причем каждая из этих структур является типовой для некоторой определенной предметной области (или набора областей), например, БПФ является типовой структурой в области обработки сигналов.

Раз многие задачи построены на основе типовых структур, значит, поняв, как устроены типовые структуры, можно понять, как устроены и эти задачи. А это понимание чрезвычайно важно при построении эффективной реализации задачи на некоторой компьютерной платформе. Понимать, как устроена некоторая структура – это значит понимать, какими важными свойствами данные структуры обладают, какие характеристики их описывают, на какие параметры надо обращать внимание.

Задача состоит в нахождении подхода к формализации понятия типовой алгоритмической структуры, и после этого – в нахождении подхода к описанию алгоритмической структуры. Необходимо разработать методы исследования и эффективного отображения этих структур на конкретные классы параллельных систем и на конкретные архитектуры. В настоящий момент можно выделить следующие классы архитектур: реконфигурируемые процессоры – FPGA, графические процессоры – GPU, SMP-системы, кластеры и векторные машины. Эти параллельные платформы наиболее распространены на данный момент, однако нет желания ограничиваться только этими архитектурами, а хочется предусмотреть возможность добавления новых платформ, если таковые понадобятся.

2. Задача отображения

На рис. 1 приведена общая схема отображения алгоритма на некоторую платформу. Сверху расположены типовые структуры, снизу конкретные платформы. Хочется отработать путь сверху вниз, или, если более конкретно, хочется научиться решать, например, такие задачи:

1. Выявление причин неэффективного отображения типовых структур на некоторую платформу.
2. Упрощение процесса отображения в будущем при добавлении новых алгоритмических структур или новых платформ.
3. Оценка верхней границы степени эффективности отображения (в некоторой метрике) определенной структуры на определенную платформу.
4. Определение наиболее подходящей платформы для реализации конкретной структуры.

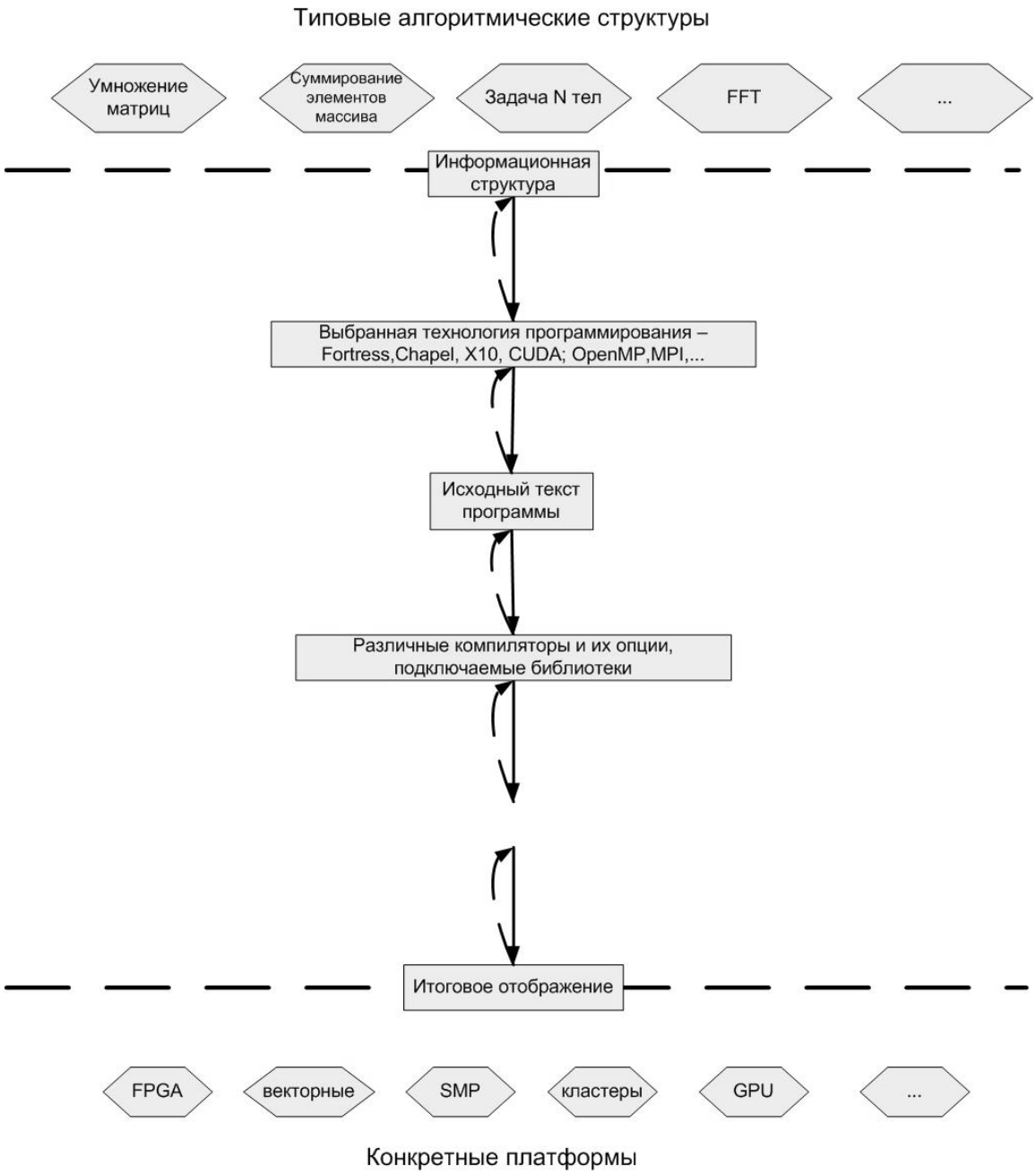

Рис. 1. Общая схема отображения типовых алгоритмических структур на вычислительные системы

Изучение процесса отображения структур на платформы позволит постепенно определять и фиксировать алгоритм отображения, поэтому в будущем некоторые шаги будут уже заранее определены и, возможно даже, будут выполняться автоматически.

Этапы процесса отображения типовой структуры на платформу, в общем, совпадают с этапами процесса написания обычной программы. Однако в данной работе важна не сама схема, отражающая последовательность шагов в процессе отображения алгоритма на архитектуру. Эффективность отображения – это является ключевым понятием. Мы хотим исследовать качество данного процесса, для чего нужно знать свойства и характеристики используемых в ней объектов: алгоритмов, программ и компьютерных платформ. Какие особенности архитектуры компьютера влияют на эффективность выполнения программы? По каким свойствам алгоритма можно судить о возможности его эффективного отображения на конкретную компьютерную

платформу? На эти вопросы нам поможет ответить исследование характеристик объектов, присутствующих на схеме отображения.

Исследуемые свойства и характеристики можно разделить на две группы: те, которые мы можем менять, и те, которые мы менять не можем. К первой группе можно отнести такие характеристики как число процессов или наличие точек синхронизации в программе, а примером из второй может служить характеристика, указывающая объем входных данных. Характеристики из первой группы определяют свободу при выборе отображения, и изменение значений этих характеристик приводит к различным вариантам реализации алгоритма. Характеристики из второй группы четко фиксированы и определяют потенциал эффективного отображения, который является показателем того, насколько хорошо в принципе можно отобразить алгоритм на выбранную платформу.

Указанную выше схему в более общем виде можно описать так: изначально задан алгоритм, который затем ложится в основу программы, а написанная и готовая к запуску программа затем исполняется на аппаратуре. Таким образом, в схеме можно выделить три основных составляющих – Алгоритм, Программа и Аппаратура. Каждая из этих составляющих обладает набором характеристик, которые могут влиять на эффективность реализации. Очень важным является то, что характеристики Алгоритма и Аппаратуры принадлежат ко второй группе, то есть не могут быть изменены, в то время как характеристики Программы могут меняться. Это означает, что эффективность отображения зависит только от того, насколько удачно будет сделан выбор значений характеристик Программы, который должен быть произведен с учетом характеристик двух других составляющих.

На каждом этапе изначально заложенный в алгоритм «потенциал отображения» (насколько хорошо он может быть распараллелен) может только уменьшаться, поэтому нужно выделить те характеристики алгоритма, которые влияют на эффективность отображения, а также научиться их оценивать и изменять для достижения лучших результатов. Невозможность повысить эффективность отображения при переходе от одного этапа к другому объясняется тем, что с каждым этапом происходит уточнение реализации алгоритма, все больше характеристик и параметров алгоритма четко фиксируется, и это накладывает все больше ограничений. Если рассматривать схему отображения, то на верхнем этапе схемы отображения задан только алгоритм и ничего более, поэтому и присутствуют только ограничения самого алгоритма. Затем задается технология программирования, в рамках которой необходимо писать программу, однако сама программа пока не зафиксирована. После написания программы становится гораздо меньше возможностей влиять на способ реализации начального алгоритма, поскольку он фиксируется в коде, и так далее. При этом очень часто нет возможности перейти к следующему этапу без потерь эффективности: например, при написании программы приходится заводить временные переменные и работать с ними, что естественно ведет к таким потерям. Поэтому чрезвычайно важно научиться понимать, какие свойства алгоритма на каждом этапе влияют на эффективность его реализации, и какие значения этих характеристик позволяют достигнуть наибольшей эффективности. Общей чертой всех этапов является то, что на каждом из них необходимо учитывать характеристики, которые получены не только на данном этапе, но и на всех предыдущих.

3. Описание характеристик различных этапов отображения

Перейдем к рассмотрению этапов отображения и выделенных в каждом из них характеристик. Общая схема приведена на рис. 2. Стоит отметить сразу, что данный набор характеристик не является окончательным и будет в дальнейшем дополняться.

Анализ информационной структуры алгоритма. Рассмотрим самый верхний уровень, когда известен только алгоритм, а, значит, известна и его информационная структура [4]. Для этого будем использовать граф алгоритма, который полностью отражает информационную структуру. Графом алгоритма называется ориентированный граф, в котором вершины – это множество всех операций алгоритма, и из одной вершины идет дуга в другую вершину тогда и только тогда, когда результат операции, вычисляемой в первой вершине, является аргументом операции, вычисляемой во второй вершине. Данный граф позволяет оценить целое множество полезных свойств, в частности, ресурс параллелизма алгоритма, его параллельную и последо-

вательную сложность, длину критического пути ярусно-параллельной формы графа, ширину и однородность ярусов, общее число операций [4]; объем входных/выходных данных, отношение числа операций к объему входных/выходных данных и другие.

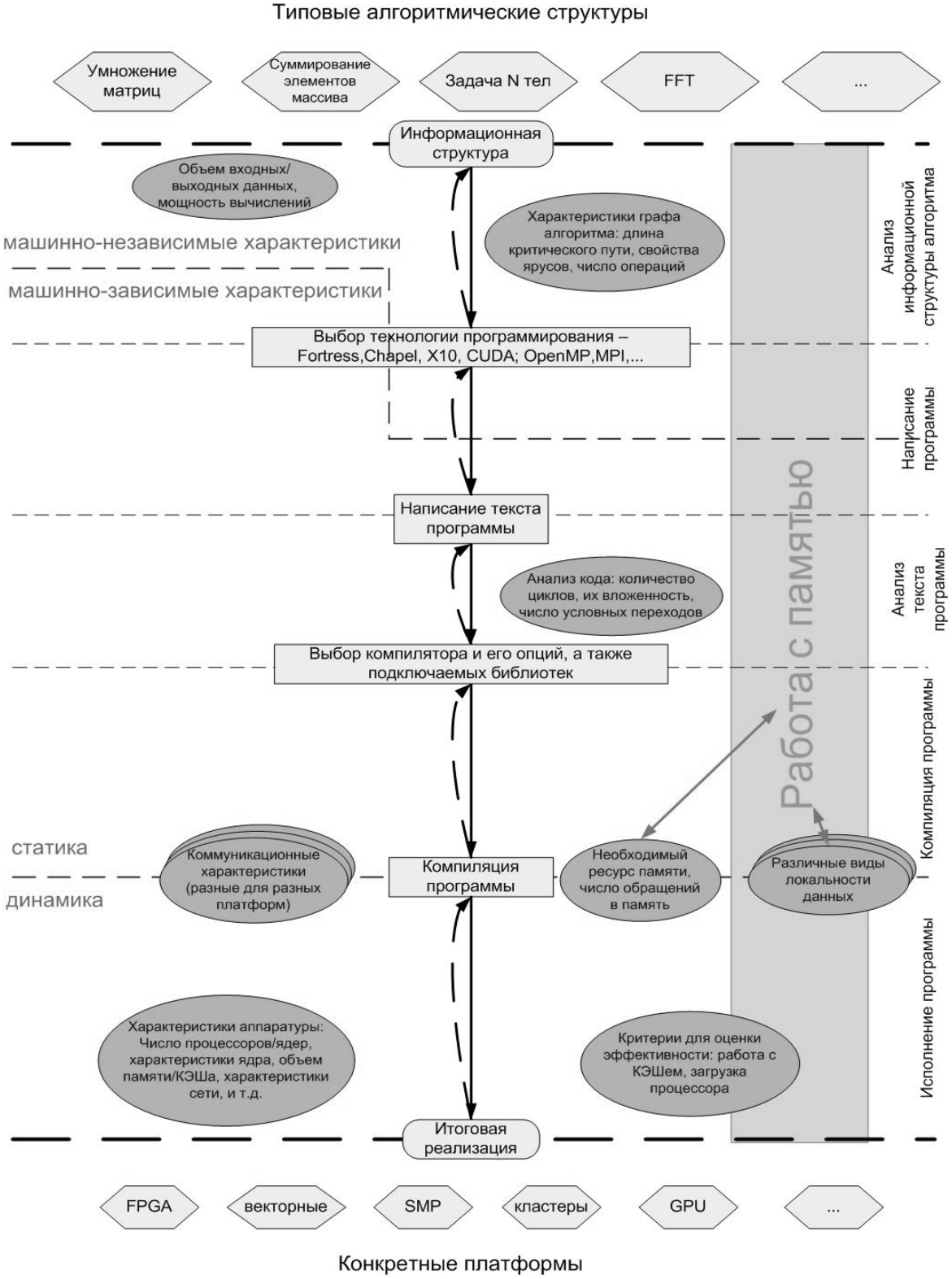

Рис. 2. Основные характеристики различных этапов отображения типовых алгоритмических структур

Помимо характеристик графа алгоритма, на данном этапе будут учитываться некоторые другие важные характеристики, такие как тип и разрядность данных, с которыми работает ал-

горитм. Разрядность важна, например, при работе с графическими процессорами, поскольку использование вещественных чисел с двойной точностью вместо одинарной часто существенно снижает скорость выполнения.

Стоит отметить, что большинство характеристик данного этапа относится к первой группе, то есть нельзя менять их значения, поскольку они являются свойствами исходного алгоритма. Однако характеристики графа алгоритма определяют потенциально возможный параллелизм данного алгоритма, верхнюю границу степени эффективности отображения. Они же могут помочь в принятии решения о том, по какому пути двигаться вниз по схеме отображения. К примеру, если мощность вычислений, то есть отношение общего числа операций к объему входных данных, слишком мала, то можно сделать вывод, что данный алгоритм вряд ли получится эффективно реализовать на графических процессорах, поскольку передача данных извне на графический процессор осуществляется медленно.

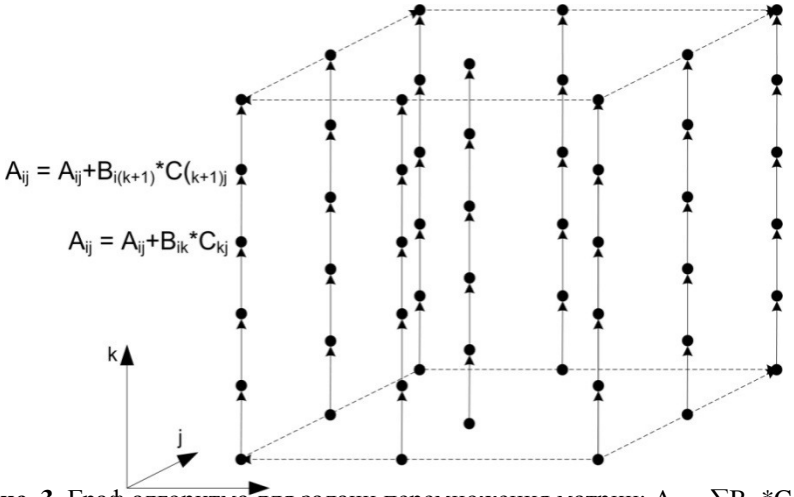

Рис. 3. Граф алгоритма для задачи перемножения матриц: $A_{ij} = \sum B_{ik}*C_{kj}$

На рис. 3 показан граф алгоритма умножения матриц размера NxN, который представляет собой куб со стороной N. Изучив граф, можно увидеть, что длина критического пути равна N-1 (каждая вершина графа – это одна операция сложения и одна операция умножения), ширина яруса постоянна и равна N^2, а общее число операций равно $2N^3$. Из полученных значений можно сделать вывод о том, что данный алгоритм можно разбить на не более чем N^2 параллельных ветвей вычислений, причем каждая из них будет состоять из 2N операций. Важной информацией является и то, что между разными ветвями отсутствует какая-либо передача данных.

После получения и анализа информационной структуры алгоритма желательно определить, на какую архитектуру исходный алгоритм будет отображаться, и на основании этой информации выбрать технологию программирования, поскольку не все технологии могут хорошо подходить для конкретной архитектуры.

Написание и анализ текста программы. После анализа графа алгоритма и последующего выбора технологии программирования наступает этап написания программы. Здесь накладываются новые ограничения на возможности отображения, поскольку необходимо программировать в рамках выбранной технологии.

Следующий этап – анализ текста программы. На данном этапе представляют интерес такие характеристики как: количество условных переходов и циклов в программе, уровень вложенности циклов; число объявленных переменных, и особенно массивов; характеристики параллельности программы – число нитей/процессов, каким образом они порождаются и т.д.; наличие в коде повторяющихся фрагментов программ (это может быть важно для реконфигурируемых процессоров); количество точек барьерной синхронизации. Такие характеристики как число циклов и уровень их вложенности, являются важными потому, что цикл является одним из основных источников параллелизма, и, следовательно, эти характеристики позволяют оценить потенциал распараллеливания в рамках написанной программы. В конце данного этапа определяются некоторые технические аспекты отображения, а именно происходит выбор наиболее

подходящего компилятора и его опций, а также определение подключаемых библиотек, если таковые требуются.

Компиляция и исполнение. Далее идет уровень, связанный с компиляцией программы и ее выполнением. Здесь анализируются основные характеристики, касающиеся работы с памятью (локальность данных, необходимый объем памяти, общее число обращений в память), а также коммуникационные характеристики, такие как коммуникационный профиль программы, объем передаваемых данных, неоднородность пересылок между процессами и т.д.

Одной из самых важных и интересных характеристик для рассмотрения является локальность данных. Локальность данных определяет, насколько близки последующие обращения в память. В зависимости от типа рассматриваемых объектов, выделяют локальность команд и локальность использования данных. Также различают пространственную и временную локальность. Пространственная локальность отражает среднее расстояние между несколькими последовательными обращениями в память; временная локальность показывает среднее число обращений по одному адресу в память за время исполнения всей программы. Анализ данных характеристик будет проводиться как минимум на двух уровнях, поскольку ясно, что большая часть информации о работе с памятью получается при выполнении программы, но часть информации можно извлечь и на этапе компиляции (можно оценить долю и профиль использования статических массивов).

Рассмотрим пример, показывающий влияние локальности обращений в память на эффективность программы. Возьмем обычную программу перемножения матриц:

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++) {
        A[i,j] = 0.0;
        for (k=0; k<n; k++)
              A[i,j] = A[i,j] + B[i,k]*C[k,j];
        }
```

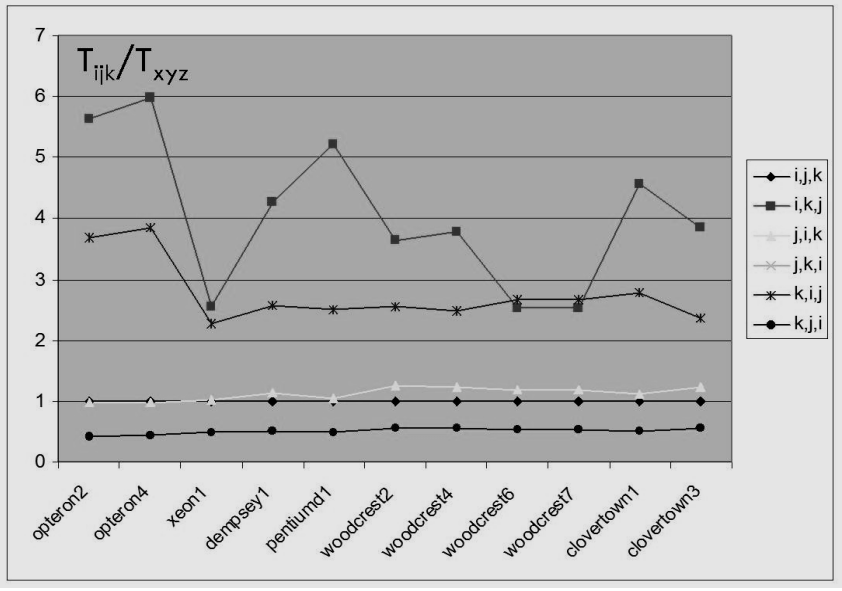

Рис. 4. Влияние локальности данных на эффективность реализации

Это самый очевидный способ реализации. Теперь единственное, что будем менять, – это порядок циклов. На рис. 4 приведены графики, которые соответствуют разным порядкам циклов в приведенном примере. По вертикали показано, во сколько раз время счета задачи при порядке циклов i,j,k больше (или меньше) времени счета задачи при других порядках циклов. Разница получается существенная, в самом лучшем варианте время счета уменьшается в 6 раз, и это только из-за перестановки циклов. Ускорение работы программы происходит именно из-

за улучшения локальности данных, и как следствие, более эффективного использования кэш-памяти.

Локальность использования данных является частью более общей характеристики – работы с памятью. Важным ее свойством, как и свойством некоторых других характеристик, является то, что они оказывают влияние почти на всех этапах отображения. Соответственно, оценивать их значение и влияние необходимо комплексно и, возможно, с применением отдельного, особого подхода.

Описание аппаратуры и оценка эффективности. Рассмотрим последний этап – выполнение программы. На этапе компиляции заканчивается часть процесса отображения, связанная со статикой, то есть с исследованием статических свойств алгоритма или программы, и на данном этапе начинается динамика – изучение программы во время ее исполнения. Здесь для нас важны аппаратные характеристики вычислительных платформ, для которых мы хотим найти эффективное отображение типовой алгоритмической структуры. Сюда входят такие характеристики как число процессоров/ядер, тактовая частота процессора, объем и структура ОЗУ и кэш-памяти, суперскалярность, характеристики коммуникационной сети и т.д. Важной особенностью подобных характеристик является то, что мы не можем изменять их значения, они только накладывают ограничения на потенциал эффективного отображения, и это приходится учитывать.

С указанными характеристиками тесно связаны критерии для оценки эффективности отображения. На их основе мы будем оценивать, насколько эффективно написана программа для данной платформы. Это такие критерии как загрузка процессора, работа с кэш-памятью, загрузка сети и т.д. Они будут сигнализировать о том, что эффективность где-то теряется, а для того, чтобы определить, в каком именно месте теряется эффективность, необходимо «подниматься» по схеме отображения по уровням вверх, чтобы понять, какую характеристику надо изменить, чтобы добиться большей эффективности. Например, некоторый критерий показывает, что происходит много кэш-промахов. Это значит, что в получившейся программе плохая локальность использования данных. Можно ли это исправить и как? Сначала исследуются характеристики на нижнем уровне, связанные с локальностью данных, однако причина не здесь, значит надо подниматься вверх; на этапе компиляции тоже все нормально; еще на уровень выше, на этапе написания программы видно, что надо было циклы написать в другом порядке (или использовать другие структуры данных). После внесения изменений в программе необходимо снова провести анализ характеристик эффективности.

4. Исследование эффективности реализации типовой структуры «триада»

Рассмотрим на конкретном примере влияние характеристик последнего уровня на эффективность реализации. Возьмем несколько вариаций типовой алгоритмической структуры «триада»:

A[i] = B[i]*X+C (1)
A[i] = B[i]*X[i]+C (2)
A[i] = B[i]*X+C[i] (3)
A[i] = B[i]*X[i]+C[i] (4)

Соответственно, переменные X и C в одних вариантах являются скалярами, в других – массивами. Также, будем рассматривать различные способы адресации массивов в этих вариантах – прямая и два варианта косвенной, с использованием дополнительного массива индексов ind. В первом варианте ind1[i] = i, во втором ind2[i] = (i*K)/N+(i*K)%N, где N – длина массива. Первый вариант помогает оценить накладные расходы, привносимые косвенной адресацией, а второй моделирует случай с плохой локальностью данных: константа K подбирается так, чтобы обеспечить максимальный процент кэш-промахов. При этом будем рассматривать самый простой, последовательный вариант решения этой задачи, в котором все исполняется только на одном ядре. При кажущейся простоте поставленной задачи ее анализ позволяет увидеть много интересного. Для реализации был выбран язык C (на ассемблере ничего не писалось), однако выбор языка не так важен в данном эксперименте, поскольку эффективность результатов рас-

сматривалась относительно друг друга. В качестве критерия эффективности использовалось отношение реальной производительности к пиковой.

Для изучения влияния аппаратных характеристик и получения реальных значений эффективности был проведен ряд экспериментов на различных системах. Информация о системах приведена в таблице 1.

Таблица 1. Описание систем, использованных в экспериментах с «триадой»

Название	Тип процессора	Частота процессора, GHz	Степень суперскалярности ядра	Частота FSB, MHz	Пиковая производительность, GFlops
Clovertown1	Xeon 5310	1,6	4	1066	6,4
Clovertown2	Xeon 5345	2,33	4	1333	9,32
Clovertown3	Xeon 5355	2,66	4	1333	10,64
Woodcrest1	Xeon 5150	2,66	4	1333	10,64
Woodcrest2	Xeon 5160	3,0	4	1333	12
Opteron1	Opteron 280	2.4	2	1000	4,8
Opteron2	Opteron 265	1.8	2	1000	3,6
Harpertown	Xeon E5472	3,02	4	1600	12

Основные аппаратные характеристики, которые могут оказывать влияние на эффективной последовательной реализации, следующие:

- Тактовая частота ядер
- Скорость шины FSB (системной шины)
- Частота оперативной памяти
- Объем ОЗУ на процессор
- Размер кэш-памяти разных уровней
- Суперскалярность
- Наличие векторных операций

Первые две характеристики сами по себе напрямую не определяют эффективность, однако их отношение является очень важным параметром, на который, как мы дальше увидим, нужно обращать внимание при рассмотрении эффективности реализаций. И вот почему: эффективность задачи определяется тем, насколько полно загружен процессор, однако скорость подачи данных по системной шине практически всегда меньше скорости обработки эти данных процессором, и значит, данные просто не будут успевать передаваться по шине процессору. Таким образом, узким местом является пропускная способность системной шины.

Взаимосвязь частоты процессора и частоты системной шины. Будем рассматривать следующую характеристику: $L=(F_{cpu}*s)/F_{fsb}$, где F_{cpu} – частота процессора, s – степень суперскалярности, F_{fsb} – частота системной шины. Рассмотрим вариант (1) типовой структуры «триада». В этом варианте идеальным является значение L=1, поскольку в таком случае системная шина перестает быть узким местом (можно и меньше 1, но к лучшим результатам это не приведет, поскольку узким место станет процессор, и эффективность остановится на уровне эффективности самого процессора). Однако для многих систем данное значение далеко от идеального. Рассмотрим связь характеристики L с эффективностью реализации нашей задачи. Для этого сначала посчитаем эффективность для каждой из систем, а затем посмотрим, как соотносятся значения эффективностей для различных систем и их значения характеристики L. Для этого мы считаем отношение эффективностей, деленное на обратное отношение характеристик L. Соответственно, чем ближе полученное значение к 1, тем ближе зависимость рассмотренных двух систем от параметра L друг к другу. Назовем этот отношение R, а эффективность будем обозначать EFF_i, где i – название системы. Приведенные отношения показаны на рис. 5: для каждой системы указано значение отношения R относительно системы clovertown1. Можно видеть, что для всех рассмотренных систем это значение близко к 1, и значит, для этих систем имеет место практически прямая зависимость эффективности от значения L. Это говорит о том, что измене-

ние эффективности на рассматриваемых процессорах определяется практически исключительно изменением характеристики L, что показывает ее важную роль в вопросах эффективности. Конечно, в других задачах эта зависимость может прослеживаться не столь явно, если обращения в память в них происходят не столь часто.

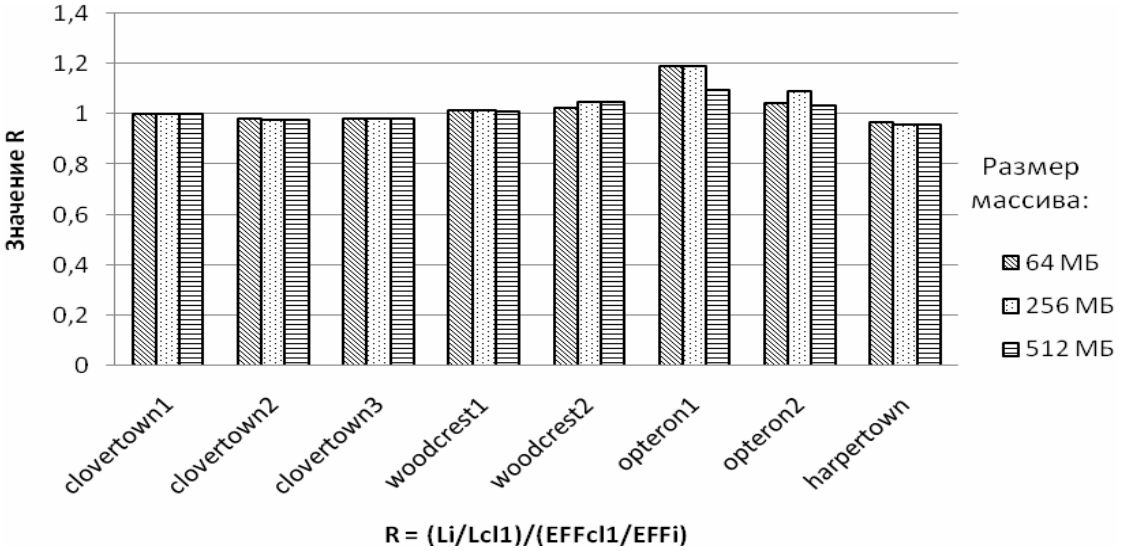

Рис. 5. Влияние характеристики L на эффективность реализации структуры «триада»

Из полученного вывода можно сделать интересные следствия. Рассмотрим эффективность для систем с процессорами из одной линейки – Xeon (рис. 6). У процессоров clovertown2 и clovertown3, а также у обоих процессоров woodcrest одинаковая частота системной шины. При этом увеличение тактовой частоты при переходе от одного к другому не приводит к какому-либо реальному ускорению работы программы, однако увеличивает значение пиковой производительности. Это в свою очередь приводит к обратно пропорциональному уменьшению эффективности программы. То есть, используя более мощный процессор, мы получаем уменьшение эффективности! Похожий эффект можно наблюдать и у процессора harpertown – у него и тактовая частота выше, и системная шина быстрее, однако отношение L не в его пользу: у harpertown это значение равно 7,55, а у clovertown2 – 6,99. Наилучшее значение характеристики L из рассматриваемых систем у clovertown1 – 6, что и выражается в наибольшей эффективности. При этом отношение эффективностей с точностью до сотых долей совпадает с отношениями их характеристик L.

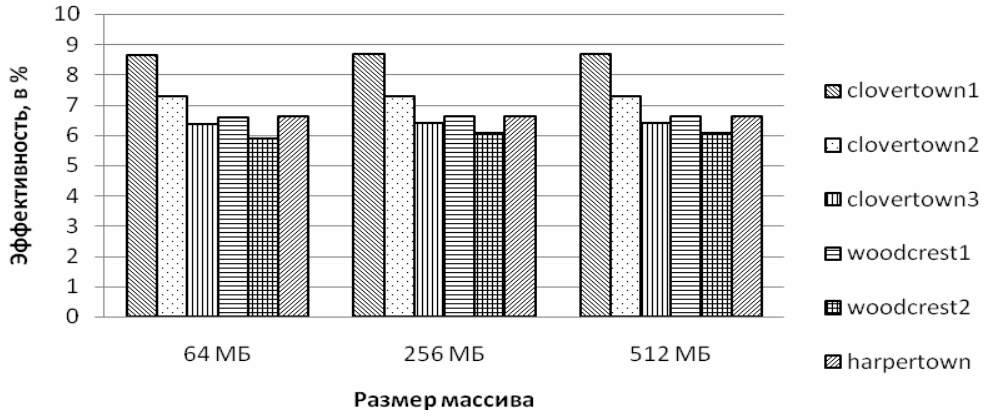

Рис. 6. Эффективность задачи для процессоров линейки Xeon

Еще одним интересным наблюдением является то, что эффективность данной задачи практически не зависит от объема входных данных – при увеличении размера массива эффективность остается практически на том же уровне. И это логично, поскольку работа с памятью происходит по одному сценарию, и кэш-память работает с одинаковой эффективностью – при занесении новой строки в кэш все ее элементы последовательно используются в программе. Важным замечанием является то, что поскольку пропускная способность системной шины у всех рассмотренных систем ниже, чем скорость, с которой оперативная память способна обрабатывать запросы на чтение/память, характеристики ОЗУ не влияют на эффективность реализаций. Все это приводит к тому, что такие обычно важные параметры как размер кэш-памяти разных уровней, размер и частота ОЗУ, в данной задаче практически не важны.

Кэш-промахи. Теперь рассмотрим примеры, когда работа с кэш-памятью происходит по разным сценариям. Для этого прогоним все 4 варианта задачи со всеми 3-мя способами адресации, и посмотрим изменения к эффективности. Соответственно, для каждой системы у нас будет 3 группы значений, по 4 в каждой. Динамика изменений для всех систем похожа, поэтому мы будем говорить безотносительно к какой-либо конкретной системе.

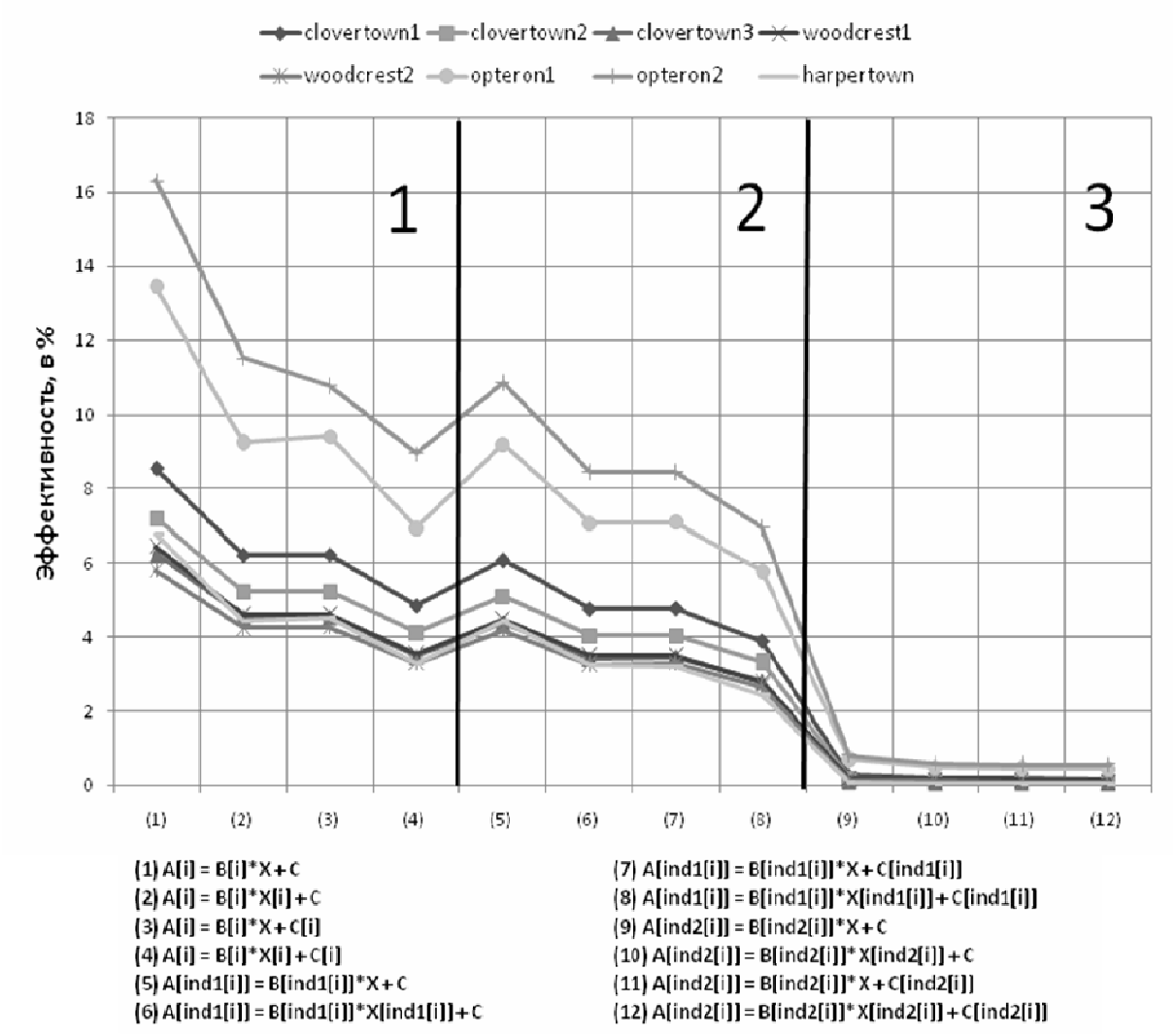

Рис. 7. Влияние степени локальности данных на эффективность реализации «триады»

В первой и второй группе результатов кэш-попадание происходило часто, поскольку выборка данных идет последовательная. Основным их отличием между собой является наличие дополнительного массива индексов ind1 – эффективность экспериментов (2), (3) и (5) практически равны, как и экспериментов (4), (6), и (7). Однако в третьей группе экспериментов эф-

фективность падает катастрофически, достигая значений менее 0.2% – кэш-промахи образуются практически при каждом обращении в память. Данный пример говорит о том, насколько важной характеристикой является локальность использования данных для оценки эффективности реализации.

5. Заключение

В данной статье рассмотрены некоторые теоретические и практические аспекты процесса выделения характеристик типовой алгоритмической структуры и оценки с их помощью эффективности реализации. На простом примере вычислительного ядра «триада» показано влияние этих характеристик. Этот пример также показывает, что эффективность и простой задачи может зависеть от довольно большого числа различных характеристик, и эта зависимость будет становиться все только сложнее с рассмотрением реальных больших задач.

Литература

1. В.В. Воеводин «Вычислительная математика и структура алгоритмов». – М: Изд-во МГУ, 2006. 112 с.
2. Asanovic, K. et al. The Landscape of Parallel Computing Research: A View from Berkeley. UCB/EECS-2006-183, University of California, Berkeley, Dec. 18, 2006.
3. Asanovic, K. et al. A View of the Parallel Computing Landscape. Communications of the ACM 52, 10 (Nov. 2009), 56-67.
4. В.В. Воеводин, Вл.В. Воеводин «Параллельные вычисления» - СПб: БХВ-Петербург, 2002. 608 с.