V. V. Voevodin, Transformation of problems of computational mathematics onto the architecture of computing systems, *Num. Meth. Prog.*, 2000, Volume 1, Issue 2, 37–44

УДК 681.3.06

ОТОБРАЖЕНИЕ ПРОБЛЕМ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ НА АРХИТЕКТУРУ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

В. В. Воеводин[1]

Рассматриваются основные положения фундаментального научного направления "Отображение проблем вычислительной математики на архитектуру вычислительных систем". Анализируется связь этого направления с различными областями, связанными с вычислениями: анализ ошибок округления, быстрые вычисления, структура алгоритмов, распараллеливание программ, проектирование систолических массивов и др.

Возникновение научного направления "Отображение проблем вычислительной математики на архитектуру вычислительных систем" можно отнести к концу 70-х годов. Бурное развитие микроэлектроники позволило достичь в те годы невиданного ранее скачка в развитии вычислительной техники. Стали функционировать вычислительные системы с производительностью в несколько сотен миллионов операций в секунду, мощность проектируемых систем определялась миллиардами операций. Появились многочисленные устройства, позволяющие очень быстро решать различные простые задачи, такие как матричные и векторные преобразования, быстрое преобразование Фурье, обработка сигналов, распознавание простейших изображений и т.п. Основной целью создания этих устройств было ускорение и упрощение процесса решения конкретных задач. Каждое из них имело свою собственную, связанную с конкретной задачей архитектуру; и не было ничего общего между различными устройствами. Тем не менее успехи в микроэлектронике привели к появлению весьма дерзкой по тем временам мысли о возможности в будущем построения заказных специализированных вычислительных систем, ориентированных на эффективное решение конкретных классов задач.

Кроме впечатляющих результатов и радужных надежд успехи микроэлектроники принесли немало серьезных проблем в деле освоения вычислительной техники, в особенности больших параллельных систем. Очень скоро стало ясно, что построение для таких систем эффективных численных методов является делом и трудным, и малоизученным. Трудности определялись главным образом значительным разнообразием архитектур самих систем и, как следствие, таким же разнообразием способов организации вычислений. Различные способы организации вычислений влекли за собой различные способы организации данных, требовали создания различных численных методов и алгоритмов, различного численного программного обеспечения, новых средств и языков общения с вычислительной техникой.

Глубокое понимание перспектив и проблем использования и развития вычислительной техники привело академика Г. И. Марчука к формированию в конце 70-х годов нового фундаментального научного направления, связанного с совместным исследованием численных методов и структур ЭВМ [1]. Оно получило название "Отображение проблем вычислительной математики на архитектуру вычислительных систем" и стало одним из ведущих направлений научных исследований в Отделе вычислительной математики АН СССР (в настоящее время Институт вычислительной математики РАН), созданном Г. И. Марчуком в 1980 г. В те годы это направление было на острие многих проблем, связанных с вычислениями. Таким оно остается и сейчас. Таким же оно останется и в будущем, по крайней мере, ближайшем.

Основные трудности развития нового направления были связаны с отсутствием строгих математических постановок нужных задач. Понятно, что нельзя даже было надеяться на существование или разработку в ближайшее время математической модели процессов функционирования ЭВМ, сколько-нибудь адекватно отражающей действительность. Только число названий различных классов вычислительной техники измерялось десятками: векторные, конвейерные, многопроцессорные, систолические, программируемые и т.п. А в каждом классе было немало существенно различных представителей. Тем не менее, несмотря на большое разнообразие, во всех представителях из всех классов можно было увидеть применение нескольких идей, решающим образом влияющих на производительность. Это, в первую очередь, — параллелизм и конвейерность вычислений, иерархическая структура памяти, использование коммутаторов и коммуникационных сетей для связи функциональных устройств между собой. Поэтому было ясно, что, как минимум, эти идеи должны находить свое отражение в структуре численных методов.

[1] Институт вычислительной математики РАН, ул. Губкина, 8, 117333, Москва;
e-mail: voevodin@vvv.srcc.msu.su

Несмотря на длительный период развития вычислительной математики вообще и численных методов и алгоритмов в частности, математики в действительности очень мало знают о том, как на самом деле устроены разрабатываемые и используемые ими методы и алгоритмы. Господствовавшая в течение нескольких десятилетий концепция однопроцессорных ЭВМ обращала внимание разработчиков алгоритмов в основном лишь на две характеристики, связанные с вычислительной техникой. Это — число операций и объем требуемой памяти. Даже такой важный фактор, как влияние ошибок округления, чаще всего в конкретных разработках выпадал из сферы внимания. Что же касается структурных свойств алгоритмов, например, таких как модульность, то их изучение так и не вышло из зачаточного состояния. Все это в конечном счете привело к тому, что к моменту широкого внедрения достижений микроэлектроники в создание вычислительных систем вычислительная математика оказалась без нужного багажа знаний, касающихся структуры алгоритмов. Без нужного багажа знаний оказались и смежные науки, в частности, связанные с разработкой алгоритмических языков, компиляторов и архитектуры вычислительных систем. Поэтому сразу появилось большое число вопросов, относящихся к тому, что же понимать под структурой алгоритмов, как ее конструктивно находить и исследовать, как решать с ее помощью нужные задачи отображения вычислительной математики на архитектуру вычислительных систем.

Вычислительная техника и алгоритмы — это две опоры, на которых строится проблема отображения. Но как бы ни велики были достижения в области развития вычислительной техники, она является всего лишь инструментом для решения прикладных задач. Инструментом, который создается и совершенствуется по своим законам, определяемым успехами микроэлектроники, авторскими идеями, чьими-то амбициями и много чем другим, не всегда полезным при проведении практических расчетов. Инструментом, который почти всегда претендует на универсальность своего использования и поэтому почти всегда используется с трудом при решении конкретных задач. Как уже отмечалось, создание общей математической модели процессов функционирования ЭВМ казалось тогда (да и кажется теперь) делом малоперспективным. Поэтому представлялось естественным, что продвижение в проблеме отображения должно начинаться с разработки фундаментального математического аппарата, позволяющего описывать и исследовать детальную информационную структуру алгоритмов. Структуру, показывающую, как в процессе реализации алгоритма отдельные его операции связаны между собой и с памятью ЭВМ.

Отметим, что такой аппарат удалось разработать. Однако наши исследования начались не с этого. По мере осмысления проблемы все большее беспокойство вызывало отсутствие какого-то базового математического формализма, помогающего оценивать качество работы многих функциональных устройств и соответственно этому предлагать схемы реализации алгоритмов. Дело в том, что одновременное использование многих функциональных устройств является одной из центральных идей, позволяющих получить дополнительное ускорение процесса решения задач. Для оценки их работы были введены различные характеристики, такие как пиковая производительность, средняя производительность, ускорение, эффективность, загруженность и т.п. Всем этим характеристикам надо было придать четкий математический смысл. С другой стороны, примерно в это же время в США, а под их влиянием и в Европе, начались активные исследования в области так называемых систолических массивов, которые представляют простейшие вычислительные системы с многими функциональными устройствами. Систолические массивы не имеют памяти и коммуникационной сети, реализуются на одном кристалле и, как утверждалось, достаточно дешевы в изготовлении. Конечно, каждый отдельный систолический массив мог решать только одну задачу, причем самую простую. Но их использование хорошо вписывалось в отображение проблем вычислительной математики на архитектуру вычислительных систем, так как мы как раз и предполагали научиться раскладывать большие задачи на простейшие. Вырисовывалась следующая схема: раскладываем задачу на простейшие, простейшие реализуем с помощью систолических массивов, а вычислительная система в целом получается как объединение этих массивов с помощью подходящей коммуникационной сети. Это и послужило основной причиной для построения и изучения математической модели процесса работы многих функциональных устройств без памяти и коммуникационной сети.

Рассмотрим произвольный ориентированный граф, считая, что в вершинах графа размещены функциональные устройства, простые или конвейерные. Сложность их не ограничивается. Каждое из устройств может выполнять только операции одного типа. Среди них имеются операции, обеспечивающие ввод или вывод информации. Дуги графа символизируют связи входов и выходов отдельных устройств между собой. Связи остаются неизменными в процессе работы устройств. Информация по ним передается мгновенно по мере необходимости. Несложно указать набор простых правил, при выполнении которых эта модельная вычислительная система будет “работать”. Самым существенным из них является необходимость разомкнуть некоторые циклы и все контуры графа на короткий период в начале процесса для загрузки начальных данных. Эта модельная система получила название “конвейерный вычислитель” [2].

Анализ процесса функционирования конвейерных вычислителей позволил придать четкий математический смысл всем характеристикам, связанным с эффективностью использования многих функциональных устройств. Были установлены различные соотношения между ними. Однако самым интересным

оказался анализ структуры алгоритма, реализуемого конвейерным вычислителем. Выяснилось, что этот алгоритм представляет объединение не связанных между собой одинаковых алгоритмов, отличающихся друг от друга только значениями входных данных. Другими словами, информационный поток, проходящий через конвейерный вычислитель, обязательно расщепляется на независимые однотипные ветви. Число этих ветвей определяется, главным образом, графом конвейерного вычислителя и может меняться при изменении графа от единицы до бесконечности.

Таким образом, был получен важный для того времени результат: для того чтобы задача эффективно решалась на вычислительной системе с многими функциональными устройствами конвейерного типа, необходимо, чтобы ее можно было представить как множество подзадач, состоящих из достаточно длинных, не зависимых друг от друга однотипных ветвей вычислений. Отметим, что этого же требовали различные векторные и матричные ускорители, векторно-конвейерные машины типа Крей-1 и т.п. Систолические массивы занимают в этом ряду особое место, и мы еще вернемся к их обсуждению.

Проводя наши исследования процессов функционирования многих устройств, мы не делали никаких предположений о способах организации вычислений. Автоматически оказались в поле зрения параллелизация вычислений, конвейерность функциональных устройств, возможность образовывать сложные конвейеры, векторные и матричные вычисления и т.п. Как показали результаты исследований, эффективное использование многих функциональных устройств, вообще говоря, эквивалентно возможности организовывать конвейерные вычисления. При этом конвейеры надо понимать не в традиционном, а в более широком смысле, в том числе с изменяемыми связями.

На одном полюсе конвейерных вычислений находятся макроконвейерные вычисления. Для процессов этого типа характерно наличие длинных, не обязательно однотипных независимых ветвей вычислений и возможность пренебрежения связями и временами передачи информации при реализации этих ветвей. На другом полюсе находятся конвейерные вычисления, реализуемые с помощью систолических массивов. Для процессов этого типа характерно осуществление всех необходимых, в том числе длинных, передач информации на фоне выполнения самих операций.

Вычисления, реализуемые на векторных системах, занимают промежуточное положение и дополнительно характеризуются всего лишь векторной организацией данных. Интересно отметить, что возможность векторной организации данных является отличительной чертой любых вычислений, реализуемых на системе устройств с редко изменяемыми связями. В частности, на любых конвейерных вычислителях длины векторов полностью определяются графом связей реализуемого алгоритма. Тот факт, что на конкретных вычислительных системах длины векторов часто выбираются постоянными, отражает особенности технических решений и не имеет никакого отношения к структуре алгоритмов.

Теперь центральной задачей проблемы отображения становится обнаружение независимых ветвей вычислений в алгоритмах. Но вначале нужно было точно определить, о каких алгоритмах пойдет речь. При выборе класса алгоритмов было принято во внимание несколько обстоятельств.

К моменту внедрения в практику вычислительных систем с многими функциональными устройствами начали разрабатываться специально приспособленные для них численные методы. Такие системы и методы обобщенно стали называться параллельными. Основной целью создания параллельных методов было решение задачи за как можно меньшее число параллельных шагов. Число шагов характеризовало время решения задачи на параллельной системе. На каждом шаге разрешалось выполнять сколь угодно много независимых друг от друга операций. Были получены удивительные математические результаты. Так, например, оказалось, что систему линейных алгебраических уравнений с матрицей порядка n можно решить всего за $0(\log_2^2 n)$ параллельных шагов, выполнить n шагов последовательного линейного рекуррентного процесса можно за $0(\log_2 n)$ параллельных шагов и т.п. Однако выполненный нами анализ показал, что у этих методов нет реальной перспективы. Как правило, они крайне неустойчивы к влиянию ошибок округления, весьма сложны в реализации и очень неэффективно используют функциональные устройства. Время показало, что наш прогноз оправдался.

Более перспективными виделись традиционные численные методы. Их привлекательная сторона заключалась в предсказуемости численного поведения. За длительный срок практического использования они уже были тщательно изучены и тестированы. Оставалось только научиться выделять в этих методах независимые ветви вычислений. Опыт работы на первых вычислительных системах с параллельной архитектурой показывал, что во многих случаях такие ветви удается обнаруживать, используя реорганизацию порядка выполнения операций. При этом довольно часто сохраняется или меняется незначительно численная устойчивость. Поиск нужного порядка вычислений, естественно, не был алгоритмизирован, и для его проведения всегда привлекались специалисты, хорошо знающие численные методы.

К тщательному анализу традиционных численных методов нас подталкивала и другая важная проблема. В течение многих лет профессиональные интересы автора лежали в области вычислительной математики и численного программного обеспечения. Так случилось, что за это время пришлось освоить более десятка различных ЭВМ. И хотя все они были одной и той же фон-неймановской архитектуры, каждый

раз при переходе к новой ЭВМ повторялась одна и та же ситуация: приходилось изменять уже созданные программы, хотя написаны они были по всем правилам алгоритмических машинно-независимых языков.

С точки зрения математика, для которого ЭВМ является всего лишь рабочим инструментом, эта ситуация в высшей степени ненормальна. Она противоречит тому, ради чего и создавались алгоритмические языки. Ведь первоначальной их целью было освобождение разработчиков алгоритмов от необходимости знать особенности конкретных ЭВМ. Все функции по адаптации программ к конкретным ЭВМ должны были взять на себя компиляторы и операционные системы. Первоначальная цель пока не достигнута. Трудно ожидать, что она будет достигнута в ближайшем будущем. Чтобы придти к такому выводу, не нужно быть большим в области алгоритмических языков. Попробуйте найти хотя бы один язык, для которого были бы даны гарантии, что записанные на нем алгоритмы будут без изменения эффективно реализовываться на разных ЭВМ. И очень скоро вы убедитесь, что таких языков нет.

Появление вычислительных систем параллельной архитектуры означало наступление нового периода пересмотра и передела численного программного обеспечения. Учитывая огромную сложность данного процесса, естественно, возникало желание хотя бы как-то его автоматизировать. Для этого была необходима детальная информация о структуре программ, вплоть до связей между отдельными операциями. Но, с другой стороны, программы интересны и с точки зрения проблемы отображения, так как представляют обширный багаж хорошо отработанных и используемых на практике алгоритмов.

Итак, выбор был сделан. В качестве класса алгоритмов при изучении проблемы отображения решено взять класс программ. Этот выбор подкреплялся еще и тем, что использование алгоритмического языка позволяло устранять многие недоговоренности, характерные для книжного описания алгоритмов, и тем самым сделать анализ более точным. В качестве языка описания программ был взят Фортран. Акцент на этом языке непринципиален. Некоторый аргумент в его пользу связан только с тем, что по своей структуре он наиболее близок к описанию алгоритмов с помощью математических соотношений. В силу простоты языка Фортран его использование не давало возможность программисту сильно запутывать описание алгоритмов. Конечно, основная цель изучения детальной структуры программ была связана с отображением проблем вычислительной математики на архитектуру вычислительных систем. Мы надеялись, что попутно удастся лучше понять проблему переноса численного программного обеспечения на вычислительные системы различной архитектуры и внести какой-то полезный вклад в ее решение.

Первым делом мы обратились к изучению методов анализа программ, используемых в компиляторах вычислительных систем с параллельной архитектурой. По своей сути эти методы неизбежно должны выделять независимые ветви вычислений, и поэтому интересно было понять, как они это делают. Мы уже имели достаточно большой свой собственный опыт анализа программ, знали многие узкие места и хорошо понимали, что надо искать в изучаемых методах. Несмотря на то что по методам анализа программ было опубликовано достаточно большое число работ, долгое время оставалось тайной, что же реально делается в компиляторе. Настоящим шоком явилось знакомство с работой [3]. В ней были проанализированы компиляторы большинства параллельных вычислительных систем. Оказалось, что на самых современных системах компиляторы реализуют самые тривиальные методики и, что для нас было особенно интересно, не дают никаких гарантий качеству результатов выполненного анализа.

Знакомство с работами по компиляции программ не прошло бесследно. Во-первых, стало ясно, что со стороны этой деятельности, по смыслу наиболее близкой к проблеме отображения, нет никаких оснований ожидать сколько-нибудь существенную помощь. Во-вторых, выяснилось, что сама эта деятельность остро нуждается в притоке свежих идей. В-третьих, мы получили много полезных сведений об общепринятых в программировании подходах к анализу программ. И, наконец, мы точно поняли, что надо делать.

Для исследования структуры связей на множестве операций в программах традиционно используются ориентированные графы. Две операции называются зависимыми, если при своей реализации они используют одну и ту же переменную. Будем считать отдельные операции вершинами графа. Зависимые операции соединим дугами. Дуга всегда идет из той вершины, которая соответствует операции, выполняемой раньше. Различают четыре вида зависимости, смотря по тому, перевычисляется или используется в качестве аргумента значение переменной в начальной или конечной операции. Соответственно этому с каждой программой связываются четыре типа графов. Каждый граф имеет дуги только одного вида зависимостей. Разные графы одного типа могут иметь разное число анализируемых дуг. При исследовании программ определенное значение имеют графы всех типов. Наиболее важным является тип зависимости, называемый истинной. Он соответствует тому случаю, когда в начальной операции значение переменной перевычисляется, а в конечной используется в качестве аргумента.

Если два множества операций не связаны путями графов зависимостей, то они представляют независимые ветви вычислений и их можно выполнять независимо друг от друга или, другими словами, параллельно. Определить, зависима или не зависима пара операций, относительно просто. Нужно лишь установить, используют или не используют они одни и те же переменные. Простота притягивает. Возможно, именно поэтому основные усилия специалистов в области изучения структуры программ были

направлены на разработку достаточных критериев независимости множеств операции. За исключением тривиальных случаев, графы зависимостей никогда не строились явно. Они служили, главным образом, иллюстративными объектами для объяснения того, что делается в конкретной методике. В том виде, как они определялись, графы зависимостей и нельзя было построить явно. При многократном использовании памяти ЭВМ в каждую вершину графов будет входить столько много дуг, что такие графы невозможно описать конечным набором каких-то простых функций.

Для нас с самого начала исследований было очевидно, что нужно стремиться к разработке необходимых и достаточных критериев независимости множеств операций. Использование достаточных критериев имеет серьезный недостаток. Если с их помощью не удается найти нужные множества операций, то непонятно, что в этой ситуации делать пользователю: или начинать разрабатывать новый метод решения задачи, или каким-то образом преобразовать программу, или применить какой-либо более изощренный метод анализа в надежде найти нужные множества операций. Реализация первых двух путей весьма обременительна для пользователя. Поэтому он более охотно идет по третьему пути. В свою очередь, это обстоятельство стимулирует разработку все новых и новых достаточных критериев независимости.

Поскольку традиционные графы зависимостей конструктивно построить нельзя, нами были предприняты попытки найти другие подходящие объекты, в первую очередь среди подграфов графов зависимостей. Рассмотрим граф зависимостей, у которого все пары зависимых вершин одного типа соединены дугами. Именно эти графы рассматриваются традиционно. Будем называть его максимальным. Программа однозначно определяет порядок выполнения операций. Это означает, что на множестве вершин графа вводится частичный порядок. Обычно его называют лексикографическим. Зафиксируем какую-нибудь вершину. Разобьем все множество дуг максимального графа, входящих в данную вершину, на группы. Отнесем к одной группе дуги зависимости от одной и той же переменной. В каждой группе выберем дугу, у которой начальная вершина лексикографически ближе всего к зафиксированной вершине. Построим остовный подграф максимального графа, оставив из каждой группы только одну выбранную дугу. Будем называть такой граф зависимостей минимальным.

Таким образом, с каждой программой однозначно связывались четыре минимальных графа зависимостей. Все эти графы имели одно и то же множество вершин, но различались дугами и их функциональным содержанием. Графы были относительно просты, так как теперь в каждую вершину входило лишь небольшое число дуг. В конкретных случаях графы всегда строились и описывались небольшим числом простых функций. Но прежде чем заняться разработкой общего метода построения минимальных графов по тексту программы, необходимо было убедиться, что с их помощью можно решать содержательные задачи. Особое внимание уделялось минимальному графу истинных зависимостей, или, как мы его называем, графу алгоритма [2].

Результаты посыпались как из рога изобилия. Первое, к чему мы обратились, следуя общей стратегии решения проблемы отображения, — это к построению математических моделей систолических массивов. Как уже отмечалось ранее, интерес к данной проблеме был большим. Общий подход к конструированию систолических массивов основан на следующей идее. Пусть в нашем распоряжении имеется достаточное число функциональных устройств, реализующих операции одного или нескольких типов. Будем называть устройства систолическими ячейками (процессорными элементами, элементарными процессорами, чипами и т.п.). Допустим, что конструктивно они выполнены в виде геометрически одинаковых многоугольников, на границы которых выведены входы и выходы. Теперь начнем складывать из многоугольников различные фигуры, присоединяя без наложения последовательно многоугольник за многоугольником. Если в местах соприкосновения сторон соединить входы и выходы соседних устройств, то получится некоторая вычислительная система. При некоторых дополнительных условиях она и называется систолическим массивом.

При различных операциях, выполняемых систолическими ячейками, и различных фигурах, составленных из многоугольников, мы получаем систолические массивы, реализующие различные алгоритмы. Анализ работы таких систем выполнялся довольно легко. Однако большие трудности вызывал синтез систем, т.е. выбор математического содержания систолических ячеек и составленных из них фигур, при котором систолический массив будет реализовывать заданный алгоритм. Используя минимальные графы зависимостей, нам удалось полностью решить задачу синтеза. В терминах свойств этих графов был указан класс алгоритмов, для которых вообще возможно построение систолических массивов. Процесс построения модели массива сводился всего лишь к проектированию минимальных графов зависимостей на подходящим образом выбранные поверхности. Соответствующие примеры приведены в [2].

Как мы также уже отмечали ранее, выделение независимых ветвей вычислений в программах часто сопровождалось переупорядочиванием множества операций. При переупорядочивании операций получается, строго говоря, другой алгоритм, чем тот, который был описан исходной программой. Вопрос об эквивалентности этих алгоритмов оставался открытым. Минимальные графы зависимостей и здесь оказались полезными. Будем называть алгоритмы эквивалентными по вычислениям, если при одних и тех

же входных данных и одном и том же способе округления результатов промежуточных вычислений они дают одни и те же конечные результаты, включая всю совокупность ошибок округления. Выяснилось, что с точностью до некоторых оговорок алгоритмы, выполняющие одно и то же множество операций, эквивалентны по вычислениям тогда и только тогда, когда их графы алгоритмов изоморфны.

В прямом или косвенном виде минимальные графы зависимостей связаны с многими проблемами, относящимися к изучению алгоритмов. Действительно, реализация записанного программой алгоритма означает рекуррентное вычисление некоторой последовательности величин. Составим для них функциональную матрицу Якоби из частных производных первого порядка. Назовем ее вариационной матрицей алгоритма. Структура ненулевых элементов этой матрицы такая же, как у матрицы смежностей графа алгоритма. Поэтому ее можно вычислять параллельно с реализацией самого алгоритма. Сложность обоих процессов примерно одинакова. Было установлено, что именно вариационная матрица алгоритма является общим звеном в таких разных задачах, как восстановление линейного функционала, быстрое вычисление градиента, исследование влияния ошибок округления и т.п. [4].

Четыре минимальных графа зависимостей, сопровождающих записанный программой алгоритм, предоставляют самую детальную информацию о связях между операциями. Это дает основание считать их носителями информационной структуры алгоритма. Но для эффективного использования сведений о структуре не хватает описания графов в удобной для использования форме.

Возьмем какой-нибудь ориентированный ациклический граф. Всегда можно написать программу на языке Фортран такую, что сопровождающий ее, например, граф алгоритма будет изоморфен исходному графу. Вроде бы это говорит о произвольной структуре графов зависимостей. Теперь зададим себе вопрос о соответствии подобных примеров практике. Как правило, программы зависят от внешних переменных, значения которых не известны. Такими переменными являются, в частности, размеры матрицы, величина шага сетки, точность и т.п. Написать подобную программу с произвольными графами нельзя, так как от внешних переменных будет зависеть ее длина. Индексные выражения используемых на практике программ обычно весьма просты. Чаще всего они линейны. Основная масса вычислений при наличии внешних переменных никогда не бывает произвольной. Она всегда организуется как многократное циклическое повторение каких-то простых последовательностей операций.

Все сказанное свидетельствует о том, что класс программ, имеющих отношение к реальной деятельности, значительно уже, чем класс всех программ. Это обстоятельство неизбежно должно было сказаться на проблеме отображения, по крайней мере, на эффективности ее решения. Но ведь неизвестно, где проходит и что собой представляет граница между программами.

Мы начали разработку методов построения минимальных графов зависимостей с относительно простого, но очень представительного класса линейных программ. Вычисления в нем описываются произвольной структурой циклов DO и ветвлений, исполняемыми являются только операторы присваивания и все вхождения переменных, условия ветвления и границы изменения параметров циклов задаются целочисленными выражениями, линейными как по параметрам циклов, так и по внешним переменным. В теории и практике использования программ линейный класс занимает примерно такое же место, как матрицы в конечномерном анализе, численные методы линейной алгебры в вычислительной математике, задачи линейного программирования в оптимизации и т.п. Очень многие программы или их основные фрагменты изначально являются линейными. Еще большее число различных программ, в том числе содержащих вызовы подпрограмм и функций, может быть сведено к линейным. Поэтому всестороннее изучение программ из линейного класса представляло несомненный интерес.

Нами был доказан фундаментальный факт, говорящий о том, что для любой линейной программы любой минимальный граф зависимостей описывается конечной системой функций, линейных как по параметрам циклов, так и по внешним переменным. Число этих функций определяется программой, но не зависит от значений внешних переменных. Возможность явного представления минимальных графов зависимостей конечным числом линейных функций открывало многообещающие перспективы как в изучении самых тонких элементов структуры программ, так и в деле преобразования программ под требования целевого компьютера. И, конечно, с помощью этих функций оказалось возможным строить самые совершенные, в том числе неулучшаемые, критерии независимости множеств операций. Все это стимулировало разработку эффективных методов их нахождения, исходя только из текста программы. Такие методы были разработаны [4].

Напомним, что одним из важнейших этапов решения проблемы отображения является разложение большой задачи на простейшие с известной структурой. Проведенные исследования показали, что этот этап настолько сложен, что ни о каком его “ручном” выполнении не может быть и речи. Теперь в качестве первоочередного встал вопрос о разработке автономной программной системы для всестороннего анализа больших прикладных программных комплексов. В довольно сжатые сроки такая система была создана в научно-исследовательском вычислительном центре МГУ Вл.В. Воеводиным. Она получила название V-Ray system. Система позволяла изучать структуру программ как на макро-, так и на микроуровне.

Анализ на микроуровне был основан на построении и исследовании минимальных графов зависимостей. Система оказалась исключительно эффективной. В качестве подтверждения этого отметим, например, что выполненная нами с ее помощью оптимизация программ из широко известного тестового пакета Perfect Club Benchmarks для компьютеров CRAY Y-MP M90 и CRAY Y-MP C90 оказалась самой лучшей среди тех, которые были доступны для сравнения. С некоторыми сведениями о V-Ray system и ее использовании можно познакомиться в работе [5].

V-Ray system создавалась как в интересах проблемы отображения для анализа структуры программ, так и в интересах проблемы эффективного использования вычислительных систем параллельной архитектуры для адаптации программ к требованиям таких систем. По замыслу главным был первый интерес. По факту главным на текущий момент стал второй интерес. Объясняется это следующим обстоятельством. За последние годы значительно расширились возможности доступа, в том числе удаленного, к вычислительным системам параллельной архитектуры. Пользователи этих систем, как правило, имеют очень сложные задачи, но не имеют достаточных знаний и инструментальных средств для адаптации своих программ. С другой стороны, вокруг V-Ray system образовался высококвалифицированный коллектив, который имеет нужные знания и нужный инструментарий. Поэтому вполне естественно, что какое-то время большее внимание уделялось просветительской и консультационной деятельности, а также адаптации больших конкретных программных комплексов к требованиям больших конкретных параллельных систем. Кстати, эта работа оказалась полезной и в интересах проблемы отображения. Мы увидели, что для эффективного освоения некоторых систем не хватает теоретических знаний о структуре программ.

В первую версию V-Ray system были включены главным образом те результаты исследования программ, которые были связаны с вычислительными системами конвейерного и векторно-конвейерного типов. Но стали появляться многопроцессорные системы и сети компьютеров. Для них узким местом является пересылка данных от процессора к процессору. Потребовались дополнительные теоретические исследования и соответствующее расширение V-Ray system.

Универсальным инструментом для изучения различных реализации любого конкретного алгоритма как на всех существующих, так и на проектируемых, в том числе даже гипотетических вычислительных системах является граф-машина [4]. Рассмотрим граф алгоритма. Будем считать, что в его вершинах помещены функциональные устройства, выполняющие соответствующие операции. Установим длительность срабатывания устройств. Пусть дуги соответствуют каналам передачи информации между устройствами. Установим на дугах длительности передач. При необходимости присоединим устройства ввода и вывода информации и установим моменты ввода входных данных. Это и есть граф-машина. Как и в случае конвейерного вычислителя, несложно указать набор простых правил, при выполнении которых эта модельная вычислительная система будет "работать". При любых установленных длительностях и моментах подачи входных данных мы будем получать один и тот же результат, включая всю совокупность ошибок округления. Он будет совпадать с результатом, полученным на конкретной системе, если только у ее функциональных устройств и устройств граф-машины будут реализованы одни и те же правила округления чисел. Как и конвейерный вычислитель, граф-машина не имеет памяти и может сохранять результаты промежуточных вычислений только в самих устройствах. Но в граф-машине каждое из функциональных устройств, включая устройства ввода-вывода, срабатывает только один раз.

Введем на вершинах графа алгоритма вещественный функционал $t = f(u)$, где t — момент времени, в который заканчивается работа устройства, помещенного в вершину u. Предположим, что длительности срабатывания всех функциональных устройств граф-машины отличны от нуля. Если в этом случае дуга графа алгоритма идет из вершины u в вершину v, то будем иметь $f(u) < f(v)$. Теперь зафиксируем момент t и разобьем множество Ω, всех вершин графа алгоритма на три непересекающиеся группы:

Ω_1^t — множество вершин, соответствующих операциям, выполнение которых закончилось к моменту t;

Ω_2^t — множество вершин, соответствующих операциям, выполнение которых захватывает момент t;

Ω_3^t — множество вершин, соответствующих операциям, выполнение которых не началось к моменту t.

Очевидно, что дуги графа алгоритма могут идти только из Ω_1^t в Ω_3^t и из Ω_2^t в Ω_3^t, но не наоборот. Кроме этого, в Ω_2^t нет ни одной пары вершин, которые были бы связаны дугами. Это означает, что функционал $f(u)$ позволяет выделять не связанные друг с другом множества операций и определять направленность дуг графа. Все это можно делать и без введения граф-машины. Но граф-машина дает очень полезные иллюстрации.

В самом деле, введем на вершинах u ориентированного ациклического графа вещественный функционал $f(u)$. Будем называть его строгой (обобщенной) разверткой графа, если для любой пары вершин v таких, что дуга идет из вершины u в вершину v, выполняется неравенство $f(u) < f(v) (f(u) \leqslant f(v))$. Множество обобщенных разверток замкнуто в отношении операций суммы, максимума, минимума и умножения на неотрицательное число. Все они удовлетворяют некоторому дискретному неравенству Беллмана, а оптимальные развертки — равенству Беллмана. Строгие развертки, как и в случае их привязки к граф-машине, позволяют выделять не связанные друг с другом множества операций на основе анализа

поверхностей уровня разверток. Обобщенные развертки также полезны в этом процессе. Например, с их помощью можно разбивать алгоритм на фрагменты, достаточно большие по объему вычислений, но слабо связанные между собой. Это является прямым откликом теории на требования эффективного использования многопроцессорных систем и сети компьютеров. Со многими нетривиальными свойствами разверток и их применением можно познакомиться в работах [4, 6].

Сейчас аппарат разверток включается во вторую версию V-Ray system. С его помощью мы надеемся сделать процесс анализа структуры программ и их адаптации к требованиям вычислительных систем с параллельной архитектурой еще более эффективным. Наша уверенность в успехе основана на том, что во многих случаях нам удалось разработать быстрые алгоритмы для нахождения разверток и их использования.

Много лет назад Г.И. Марчук предложил нам заняться рассмотренным выше научным направлением. Довольно скоро мы увидели, что к нему примыкают такие разные области, как вычислительные и операционные системы, компиляторы и автономные программные системы, языки программирования, численные методы, дискретная математика, теория оптимальных процессов и др. Специалисты в этих областях говорят на разных языках и далеко не всегда понимают или даже хотят понимать друг друга. Тем не менее, предстояло свести все это в нечто общее в интересах эффективного решения прикладных задач как на существующих вычислительных системах, так и на системах, которые только еще проектировались или рождались пока всего лишь в мыслях. Начиная наши исследования, мы даже не предполагали, во что они могут вылиться. Сейчас создан фундамент нового направления и пройдено несколько шагов. Открылись совершенно необозримые перспективы. Новое направление оказалось на острие многих проблем, связанных с вычислениями. Напомним некоторые из них:

— анализ ошибок округления;
— быстрое вычисление градиента и производной;
— быстрое восстановление линейного функционала;
— декомпозиция алгоритмов;
— восстановление математических формул;
— использование распределенной и иерархической памяти;
— построение систолических массивов;
— адаптация программ к конкретным компьютерам;
— выбор оптимальной архитектуры компьютера;
— обнаружение узких мест алгоритмов;
— конструирование параллельных численных методов;
— создание портабельного численного программного обеспечения;
— сравнение языков программирования.

СПИСОК ЛИТЕРАТУРЫ

1. *Марчук Г.И., Котов В.Е.* Проблемы вычислительной техники и фундаментальные исследования // Автом. и вычисл. техн. 1979. № 2. 3–14.
2. *Воеводин В.В.* Математические модели и методы в параллельных процессах. М.: Наука, 1986. 296 с.
3. *Zhiyu S., Zhiyuan L., Pen-Chung Y.* An empirical study of fortran programs for parallelizing compilers // IEEE Trans. on Parallel and Distributed Systems, July 1990. 350–364.
4. *Voevodin V.V.* Mathematical Foundation of Parallel Computing. World Scientific Publishing Co., Series in Computer Science. 1992. Vol. 33. 343 pp.
5. *Voevodin Vl.V., Voevodin V.V.* Analytical methods and software tools for enhancing scalability of parallel applications // Proc. of Intel. Conf. HiPer'99, Norway. 1999. 489–493.
6. *Воеводин В.В.* Информационная структура алгоритмов. М.: Изд-во Моск. ун-та, 1997. 139 с.

Поступила в редакцию
15.12.2000