МАТЕМАТИЧЕСКИЕ ПРОБЛЕМЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ*

В.В. Воеводин

ИВМ РАН, Москва

Возможность быстрого решения задач на вычислительной технике параллельной архитектуры вынуждает пользователей изменять весь привычный стиль взаимодействия с компьютерами. По сравнению, например, с персональными компьютерами и рабочими станциями меняется практически всё: применяются другие языки программирования, видоизменяется большинство алгоритмов, от пользователей требуется предоставление многочисленных нестандартных и трудно добываемых характеристик решаемых задач, интерфейс перестает быть дружественным и т.п. Важным является то обстоятельство, что неполнота учета новых условий работы может в значительной мере снизить эффективность использования новой и к тому же достаточно дорогой техники.

Надо заметить, что общий характер трудностей, сопровождающих развитие параллельных вычислений, в целом выглядит таким же, каким он был и во времена последовательных. Только для параллельных вычислений все трудности проявляются в более острой форме. Во многом из-за большей сложности самой предметной области. Но, возможно, главным образом вследствие того, что к началу активного внедрения вычислительных систем параллельной архитектуры в практику решения больших прикладных задач не был построен нужный теоретический фундамент и не был развит математический аппарат исследований. В конце концов, из-за этого оказался своевременно не подготовленным весь образовательный цикл в области параллельных вычислений, отголоски чего проявляются до сих пор. Отсюда непонимание многочисленных трудностей освоения современной вычислительной техники, пробелы в подготовке нужных специалистов и многое другое.

Теоретический фундамент. Последовательные вычисления развивались не одну сотню лет. За это время пришло довольно чёткое понимание, что такое последовательный алгоритм. Вокруг данного понятия сформировался большой раздел математики, называемый теорией алгоритмов, изучающий общие свойства последовательных вычислений. Уточнённое понятие алгоритма в терминах идеализированных вычислительных машин

* Сстатья перепечатана из сборника трудов Всероссийской конференции «Научный сервис в сети Интернет: технологии распределенных вычислений». Новороссийск, 19–24 сентября 2005 г. – М.:Изд-во Моск. ун-та, 2005. – С. 3–8.

привело к очень важному понятию машины Тьюринга. По существу этот автомат стал теоретическим прообразом первых ЭВМ. Присоединение к машине Тьюринга памяти сделало её весьма полезным и даже приближенным к реальности инструментом исследований. Но в основе всего лежали последовательные действия. Не удивительно поэтому, что и ЭВМ в течение длительного периода также развивались по пути реализации именно последовательных действий.

Хотя общая направленность последовательного выполнения операций сохранялась довольно долго, в разработке вычислительной техники незаметно назревали *революционные изменения*, приведшие, в конце концов, к радикальному пересмотру всех представлений о вычислениях. Причиной возникновения этих изменений стал параллелизм, внедряемый в вычислительную технику во имя повышения производительности. Пока параллельное выполнение в компьютере любых операций, передач информации и обращений к памяти не приводило к принципиальным изменениям в языках программирования, у пользователей не было особых причин думать о параллелизме. Но в определённый момент параллелизма в компьютере стало столь много, что его присутствие уже нельзя было прикрывать техническими решениями. И тогда от пользователя стали требовать предоставления дополнительной информации о структуре используемых им алгоритмов, требовать как раз для того, чтобы эффективно использовать заложенный в компьютер параллелизм. Однако сам пользователь оказался к выполнению этих требований не готов.

Очень скоро выяснилось, что *пользователь знает эту информацию далеко не всегда.* Более того, чаще всего он даже не понимает, откуда и как ее получать. И оказалось, что добывается она, как правило, с большим трудом.

К началу массового внедрения вычислительных систем параллельной архитектуры многие математические вопросы параллельных процессов оказались в зачаточном состоянии. Не было никакой целостной теории параллельных алгоритмов, аналогичной теории алгоритмов для последовательных вычислений. Существовали лишь отдельные разрозненные результаты. Не было даже сколько-нибудь ясного представления, что же нужно понимать под параллельным алгоритмом. И, конечно, отсутствовал какой-либо формальный математический аппарат, который можно было бы назвать параллельным аналогом машины Тьюринга.

Скорее всего, именно эти причины привели к тому, что в течение долгого времени параллельные вычисления не удавалось сформировать как самостоятельную математическую науку, и рассматривались они как совокупность каких-то полуэвристических, граничащих с искусством приемов приспособления алгоритмов к требованиям новой техники.

Новые параллельные алгоритмы. Несмотря на все это, параллельные алгоритмы начали создаваться уже давно. Значительный интерес к их

построению на основе математически эквивалентных преобразований возник в 60 – 70-х годах прошлого столетия в связи с появлением первых вычислительных систем параллельной архитектуры.

Чтобы оценить время реализации алгоритма на параллельной системе, алгоритм представляют в виде последовательно выполняемых ансамблей операций, причём в каждом ансамбле все операции не должны быть связаны друг с другом. Если архитектура параллельной системы позволяет реализовывать одновременно все операции каждого ансамбля, то без учёта времени на передачи данных время выполнения алгоритма будет пропорционально числу ансамблей. Число ансамблей стали называть *высотой* алгоритма. Алгоритмы, в которых высота меньше общего числа операций, стали называть *параллельными,* а их представление через последовательность ансамблей из независимых операций – *параллельной формой.*

Очевидно, что в зависимости от структуры связей между операциями один и тот же алгоритм может быть представлен различными способами в виде совокупности ансамблей. В частности, обычная последовательная реализация означает, что в каждом ансамбле содержится только одна операция. Для большинства алгоритмов даже таких представлений может существовать очень много. Ясно, что для каждой задачи особый интерес представляет нахождение алгоритмов *минимальной* высоты. Согласно теории последовательных алгоритмов представления *одного и того же алгоритма* различными ансамблями необходимо рассматривать как *разные алгоритмы*, так как изменяется, как минимум, порядок выполнения операций. Следовательно, некоторые характеристики этих разных алгоритмов окажутся заведомо различными, но какие-то наверняка сохранятся.

Так что же меняется и что сохраняется в алгоритмах при тех или иных преобразованиях? На все подобные вопросы требуется дать четкие математические ответы. Это необходимо сделать еще и потому, что различные программы, для создания которых авторами часто используется *один и тот же* алгоритм, на самом деле почти всегда описывают *разные* алгоритмы, хотя и математически эквивалентные. А это приводит на практике к разным результатам.

Чтобы задача построения быстрых параллельных алгоритмов стала математически корректной, необходимо сделать какие-то предположения относительно свойств параллельной вычислительной системы. Они очень просты: система имеет бесконечно много параллельно работающих процессоров; все они работают синхронно под общим управлением и выполняют любую операцию точно и за одно и то же время; система имеет бесконечно большую память; все обмены информацией между процессорами и памятью, а также между самими процессорами осуществляются мгновенно и без конфликтов. Концепция построения алгоритмов для подобных параллельных систем получила название *концепции неограниченного па-*

раллелизма. Конечно, она идеализирована. Тем не менее полученные в её рамках результаты интересны и поучительны.

Рассмотрим обычный процесс суммирования n чисел, когда на каждом шаге к частичной сумме прибавляется очередное слагаемое. Этот алгоритм имеет только одну параллельную форму, в каждом ансамбле которой имеется лишь одна операция. Следовательно, никакой возможности использовать параллелизм *в этом алгоритме* нет. Поскольку операция суммирования большого числа слагаемых является очень распространённой, был придуман другой способ суммирования, обладающий лучшим параллелизмом. Разобьём все слагаемые на пары и осуществим суммирование двух чисел внутри каждой пары. Все эти операции независимы. Полученные частные суммы также разобьём на пары и снова осуществим суммирование двух чисел внутри каждой пары. Снова все операции независимы. Вся сумма будет получена через $\log_2 n$ шагов. Это и будет высота *нового* алгоритма. В нём уже имеется значительный ресурс параллелизма, хотя он не равномерен. На первом временном шаге может быть использовано $n/2$ процессоров, на втором $n/4$ и т.д. Назван новый алгоритм процессом *сдваивания.*

Заметим, что оба алгоритма основаны на реализации математически эквивалентных выражений суммирования чисел, но они имеют разные свойства, по крайней мере, с точки зрения параллельных вычислений. На самом деле у них много и других различий: они по-разному реагируют на ошибки округления, по-разному используют память и т.п. Поэтому эти алгоритмы следует считать *принципиально различными*, несмотря на то, что они *математически эквивалентны*!

Пусть какой-то алгоритм существенно зависит от n входных данных и реализуется через некоторую совокупность операций, имеющих не более p аргументов. Легко показать, что такой алгоритм не может иметь высоту меньше, чем $\log_p n$. Очевидно также, что высота любого алгоритма ограничена сверху общим числом выполняемых операций. Эти две границы являются ориентирами для построения алгоритмов минимальной высоты. Например, сразу становится ясно, что суммирование чисел по принципу сдваивания относится к оптимальным алгоритмам.

Легко построить алгоритм наименьшей высоты для задачи умножения матрицы размера $n?m$ на вектор размера m. Компоненты вектор-результата могут быть вычислены независимо, и каждая из них определяется только $2m$ входными данными. Поэтому оценка высоты снизу должна иметь порядок $\log_2 m$. Соответствующий алгоритм получается очевидным образом на основе суммирования по принципу сдваивания. Задачу вычисления произведения двух матриц порядка n можно рассматривать как задачу вычисления n произведений одной и той же матрицы и n независимых векторов порядка n. Если все эти произведения вычислять независимо по опи-

санному правилу, то полученный алгоритм будет иметь высоту порядка $\log_2 n$.

Рассмотренные задачи исключительно просты, и построение для них алгоритмов наименьшей высоты не вызывает никаких трудностей. Но это скорее исключения, чем правило. Другие задачи оказываются значительно сложнее. Например, задача обращения плотной квадратной матрицы порядка n. Всего имеется n^2 входных данных, и каждый элемент обратной матрицы в общем случае существенно зависит от всех элементов исходной матрицы. Согласно сказанному ранее, оценка снизу минимальной высоты алгоритма обращения матрицы имеет порядок $\log_2 n$. В настоящее время построены алгоритмы с высотой порядка $\log_2^2 n$, и не известно, существуют ли алгоритмы существенно меньшей высоты. Аналогичное положение имеет место по отношению ко многим другим задачам: построены алгоритмы высоты, существенно меньшей, чем общее число операций, но *не известно*, можно ли эти алгоритмы улучшить по высоте.

Высота алгоритма является очень важной характеристикой, так как показывает потенциальную возможность быстрого решения задачи на вычислительной системе параллельной архитектуры. Однако пока параллельные алгоритмы малой высоты не вошли в практику использования сколько-нибудь широко. Причина очень проста: подавляющее большинство из них требует огромного числа процессоров, имеет сложные коммуникационные связи и катастрофически неустойчиво. Например, некоторые быстрые алгоритмы обращения матрицы размера $n?n$ требуют порядка n^4 процессоров. На самых современных системах этими методами можно обращать матрицы не более 10-го порядка, да и то лишь теоретически. На практике даже такие задачи будут решаться очень долго из-за исключительно сложных передач данных. Среди всех быстрых параллельных алгоритмов заметным исключением являются только суммирование чисел по принципу сдваивания и некоторые его аналоги. Подобные алгоритмы используются на практике достаточно широко.

Несмотря на отмеченные недостатки, концепция неограниченного параллелизма оказалась исключительно живучей. Предельная абстрагированность от реалий вычислительной техники сделала её привлекательной для математиков. Тем не менее на сегодняшний день все достижения в рамках этой концепции скорее представляют *набор отдельных изобретений* в области численных методов, чем систематически развивающийся раздел математики. Вполне возможно, что здесь ещё *не сказано последнее слово,* и к построению быстрых параллельных алгоритмов всё же будет разработан систематизированный подход, приводящий к более эффективным решениям.

Заметим, что практически все быстрые параллельные алгоритмы на самом деле могут рассматриваться как результат математически эквивалентных преобразований формульных выражений, описывающих хорошо

известные последовательные алгоритмы. При этом набор допустимых преобразований очень прост: ассоциативность, коммутативность, дистрибутивность, приведение подобных членов, а также замена нулевого слагаемого разностью, а единичного множителя отношением любых одинаковых выражений. А какой разброс в алгоритмических свойствах!

Ошибки округления. До сих пор мы рассматривали различные изменения свойств алгоритмов при математически эквивалентных преобразованиях. Основой таких преобразований было предположение о *точном* выполнении операций. Однако на всех без исключения компьютерах на представление любого числа отводится только *конечное,* строго фиксированное число разрядов. Поэтому после выполнения каждой операции результат «обрезается» до нужной длины. Эта процедура вносит в результат ошибку, которая называется *ошибкой округления*.

Сами по себе ошибки округления отдельных операций *очень малы*. Малы настолько, что часто возникает соблазн не учитывать их влияние на общий результат. К этому подталкивает и то обстоятельство, что во всех языках программирования любые формульные выражения записываются как математические без какого-либо указания на наличие ошибок округления. Более того, отметим как факт, что ни одна используемая на практике стандартная программная среда не имеет *инструментальных средств* для контроля за распространением этих ошибок. А это распространение происходит.

Важнейшим фактором, объясняющим влияние ошибок округления компьютерных операций на окончательный результат, является радикальное изменение свойств математических операций. Именно на множестве чисел, представленных в компьютере в форме с плавающей запятой, все операции *перестают обладать свойствами* коммутативности, ассоциативности и дистрибутивности. Аналогичная потеря свойств происходит и для чисел с фиксированной запятой, но в несколько меньшей мере. Из сказанного следует исключительно важный вывод: записывая в программах математически эквивалентные выражения, *мы не должны поддаваться иллюзии*, что эти программы будут давать на компьютере хотя бы похожие результаты. Известно, что даже такая простая операция, как перестановка слагаемых в суммах чисел, может привести из-за ошибок округления к катастрофически большим различиям.

Имеется много других задач, аспектов и вопросов, связанных с заменой одних формульных выражений другими, математически эквивалентными. Обратим внимание на следующие моменты. Во-первых, сфера замен формульных выражений исключительно обширна. Во-вторых, в теории и практике осуществления замен остаётся очень много белых пятен. И, наконец, даже если замены делаются математически эквивалентными, это ещё не гарантирует, что на практике мы не встретимся с большими неожиданностями.

Информационная структура алгоритмов. Вычислительные эксперименты показывают, что практически все *новые* параллельные алгоритмы, даже те из них, которые очень эффективны в теоретическом отношении, на практике не конкурентоспособны. Поэтому на текущий момент единственно надёжным источником создания параллельных программ является подходящая *реструктуризация* проверенных временем последовательных программ и математических описаний. Выбор этих форм записей объясняется тем, что только они позволяют описать алгоритмы более или менее точно.

Формально реструктуризация сводится к *математически эквивалентным* заменам в записях всех или части формульных выражений с целью явно указать обнаруженные в алгоритмах скрытый параллелизм, возможность использования распределенной памяти и т.п. Уже отмечалось, насколько внимательно нужно относиться к таким заменам. Снова ключевым моментом становится контроль над влиянием ошибок округления на результат. Ясно, что нужно иметь эффективные технологии как для выявления требуемых свойств алгоритмов, так и для выполнения преобразований самих записей к виду, в котором все эти свойства можно описать с помощью специальных комментариев. Очень важно, чтобы все такие технологии были максимально независимы от пользовательских знаний, касающихся решаемых задач и используемых алгоритмов.

Рассмотрим все *математически эквивалентные* записи какого-либо алгоритма. Пусть каждая из них сделана на своем языке и реализуется на своем компьютере. Будем лишь считать едиными правила приближенного выполнения операций над числами. Среди указанных записей заведомо существует какое-то множество, которое для одних и тех же входных данных будет давать при реализации один и тот же результат с учетом влияния всех ошибок округления. Естественно предположить, что у всего этого множества должно быть какое-то общее ядро. И тогда возникают вопросы, как оно выглядит, как его находить, как использовать и т.п.

Чтобы найти общее ядро, необходимо, прежде всего, очистить записи от всех языковых наслоений. После такой очистки остается лишь некоторая совокупность выполняемых операций, связанных между собой отношениями «результат–аргумент». Это задает граф, получивший название *граф алгоритма*. Можно показать, что для того чтобы в одинаковых условиях разные записи алгоритмов приводили к одним и тем же результатам, необходимо и достаточно, чтобы были *изоморфны* их графы. Построенные графы описывают информационные сущности алгоритмов. Они не зависят ни от используемых языков описания, ни от применяемых вычислительных средств. Поэтому вполне естественно их считать *информационными ядрами* самих алгоритмов.

Граф алгоритма имеет очень прозрачный смысл. Поэтому его легко применять в теоретических исследованиях. Однако чтобы этот граф ис-

пользовать в реальных приложениях, он должен быть явно задан в какой-либо форме, приемлемой для таких целей. На практике он никогда не бывает известен в нужном виде, и граф алгоритма приходится находить с помощью специальных методик из описывающих сам алгоритм программ или математических соотношений. Эти методики чрезвычайно сложны, и на их разработку ушло много лет. При этом пришлось поставить и решить довольно много новых и нетрадиционных математических проблем.

Отметим одну проблему, не очень заметную на первый взгляд, которая, тем не менее, на всех этапах исследований вызывала огромнейшие трудности. В графе алгоритма столько вершин, а по порядку и столько же дуг, сколько выполняется элементарных машинных операций за время реализации алгоритма. Для задач, решаемых на компьютерах часами и днями, их число настолько велико, что для прямого описания всех вершин и дуг соответствующего графа не хватит памяти самого большого компьютера в мире. Но ведь граф алгоритма нужно не только описать, его нужно и анализировать. В частности, необходимо находить различные ориентированные разрезы, критический путь, какие-то подграфы и т.п. Все такие задачи имеют полиномиальную или даже экспоненциальную сложность. Следовательно, при прямом задании графа любой его анализ, скорее всего, займёт гораздо больше времени, чем решение исходной задачи. Такой анализ практически бесполезен. Граф алгоритма приходится находить либо по текстам программ, либо из математических соотношений. Подобные формы записи почти всегда содержат параметры, например, размеры массивов, точность и т.п., которые на момент исследования записей не определены. Следовательно, вся работа по нахождению графа алгоритма и его исследованию неизбежно должна сводиться к решению параметризованных задач. Это очень серьёзная трудность. Тем не менее она была успешно преодолена.

В настоящее время решены различные связанные с реструктуризацией теоретические вопросы. Разработаны различные методы обнаружения параллельных ветвей вычислений. Предложены способы минимизации коммуникационных затрат при передачах информации между процессорами, а также между процессорами и памятью. Последнее имеет особое значение в связи с развитием распределенных вычислений. Сделано многое другое. Всё это сформировало новую область исследований, называемую *информационной структурой алгоритмов*. В её основе лежит выделение из записей алгоритма его информационного ядра, очищенного от всех элементов описания. Доказаны очень важные утверждения, из которых следует, что для широкого класса алгоритмов информационное ядро может быть описано и исследовано с помощью конечных наборов простых функций, как правило, кусочно-линейных. Построены эффективные методы вычисления, исследования и использования таких функций. Для изучения структуры алгоритмов создана и успешно функционирует автономная программная

система V-Ray. Она позволяет исследовать информационную структуру алгоритмов, описанных на языках Фортран и Си, и адаптировать программы под требования компьютеров параллельной архитектуры. Ясно, что параллельная структура программ является составной частью информационной структуры алгоритмов и может быть исследована с учетом всего сказанного выше.

Образование. Теперь можно попытаться ответить на вопрос, почему освоение вычислительной техники параллельной архитектуры идет с большими трудностями. На наш взгляд, это связано с тем, что знакомство с параллельными вычислениями, как и образование в этой области в целом, начинается не с того, с чего надо бы начинать. К тому же то, с чего надо начинать, не рассказывается ни в каких курсах вообще.

Как правило, знакомство с параллельными вычислениями начинается с изучения суперкомпьютеров и вычислительных систем параллельной архитектуры, языков и систем параллельного программирования. Безусловно, все это надо знать, и знать основательно. Однако и техника, и языки являются всего лишь *инструментами* для решения целевых задач пользователя. А *объектами*, обрабатываемыми с помощью таких инструментов, оказываются алгоритмы, с помощью которых задачи решаются. Из общих соображений ясно, что для проведения качественной обработки объекты и инструменты должны быть хорошо согласованы. Но как раз в осуществлении этого согласования и кроются многие трудности.

Для обычного пользователя инструментарий, т.е. техника и программное окружение, всегда является чем-то заданным. Если инструментарий и может быть изменен, то в очень ограниченных пределах. Остается подстраивать под его требования объекты, т.е. алгоритмы. Для этого необходимо обнаружить и описать параллельные ветви вычислений, разрезать алгоритм на фрагменты, минимизировать коммуникационные затраты, перестроить массивы данных. В зависимости от конкретных особенностей имеющейся вычислительной системы, возможно, придется делать и многое другое. Хорошо, если пользователь детально знает и задачу, и используемые алгоритмы, и особенности системы. В этом случае перестройка алгоритмов может и не вызвать серьезных затруднений. Но это относительно редкая ситуация. Чаще всего большие трудности приходится преодолевать и специалистам высокой квалификации. Что же тогда говорить о тех, кто только вступает в область параллельных вычислений!

В настоящее время основные вычислительные мощности достигаются на системах с общим числом отдельных процессоров, исчисляемых сотнями, тысячами и даже десятками тысяч. Эффективно решать задачи на таких системах трудно любым специалистам. Технологии и языки программирования, широко используемые на последовательных компьютерах и суперкомпьютерах с малым числом процессоров, на больших системах могут применяться лишь частично. Особенно трудно переносить большие

программные комплексы, создававшиеся разными людьми в течение долгого времени, на многопроцессорные кластеры и большие распределенные системы. Чтобы ни говорили сейчас специалисты по вычислительной технике и программированию, не видно никаких веских аргументов для предположений, что в обозримом будущем освоение новой вычислительной техники станет существенно проще. Поэтому к встрече с этим будущим надо серьезно готовиться уже сегодня.

Наиболее остро проблемы освоения новой техники стоят в связи с подготовкой молодых специалистов. Начать изменять создавшееся положение можно, например, корректируя те курсы, в которых будущий специалист впервые начинает серьезно изучать алгоритмы. Скорее всего, это курсы, касающиеся описания численных методов. Необходимо как можно раньше вводить в них первые элементы параллельных вычислений, например, в рамках упоминавшейся выше концепции неограниченного параллелизма. При описании численных методов совсем не трудно построить графы алгоритмов, указать в них параллельные ветви вычислений, ориентированные разрезы, которые определяют коммуникации, и многое другое, с чем придется столкнуться, работая с вычислительной техникой параллельной архитектуры.

Основная цель подобных изменений заключается в том, чтобы на самом раннем этапе обучения вычислительному делу вызвать интерес к информационным структурам алгоритмов и показать перспективность работы с ними. А перспективность действительно имеется.

Построение графов для большого числа конкретных алгоритмов выявило удивительную закономерность: большое разнообразие существующих методов не приводит к такому же разнообразию их информационных структур. Точнее, многие графы формально совершенно различных алгоритмов оказались изоморфными, отличаясь друг от друга только содержанием вершин и дуг. Поэтому была выдвинута гипотеза о том, что в конкретных вычислительных областях *типовых информационных структур немного*. Пока практика подтверждает эту гипотезу. Например, на всем множестве алгоритмов линейной алгебры типовых информационных структур оказалось всего лишь порядка десятка.

Специалистам по вычислительной и прикладной математике нужно сделать еще очень много, чтобы понять, как же в действительности устроены используемые ими алгоритмы. Если выдвинутая гипотеза окажется верной, то откроется много новых связей и направлений исследований. Изложение численных методов может быть поставлено на общий информационный фундамент, распараллеливание типовых информационных структур может быть заранее изучено и реализовано с помощью специальных программных средств, по типовым структурам могут быть построены спецпроцессоры, реализующие быстрое решение нужных алгоритмов. Отсюда уже недалеко и до построения заказных вычислительных систем,

ориентированных на эффективное решение классов задач из конкретных прикладных областей. В любом случае за выдвинутой гипотезой тянется много интересных следствий. Насколько гипотеза верна в реальности, покажет будущее. Но что-то интересное в ней всё-таки есть.

Возможно, следует более критически осмыслить научный фундамент параллельных вычислений. Мы уже отмечали значение машины Тьюринга для теории и практики последовательных вычислений. Этот формальный автомат позволяет реализовать любой последовательный алгоритм и в определённом смысле является прообразом любой однопроцессорной ЭВМ. Но любая многопроцессорная ЭВМ также может реализовать если не любой, то почти любой алгоритм. Тем не менее в развитии таких ЭВМ и всего того, что их окружает, совсем не ощущается необходимость введения формального автомата, который позволял бы реализовать *любой параллельный алгоритм*. Однако появилась острая потребность в некоторой идеализированной машине, на которой можно было бы реализовать для конкретного алгоритма *любой режим*, как параллельный, так и последовательный. С подобными задачами вполне справилась *граф-машина*, построенная на основе графа алгоритма.

Вообще говоря, не очень понятно, почему не появляется необходимость введения идеализированной параллельной машины, на которой можно было бы реализовывать любые алгоритмы в любом режиме. Может быть, просто потому, что развитие параллельной вычислительной техники до сих пор осуществляется в значительной мере стихийно и пока ещё не сформировались устойчивые принципы её конструирования?

Конечно, многое из сказанного ориентировано на перспективу. Однако использование различных знаний, касающихся информационной структуры алгоритмов, уже сейчас приносит свои плоды. Например, решена проблема построения математических моделей систолических массивов, представляющих специального вида вычислительные системы для сверхбыстрой реализации некоторых алгоритмов. Установлена связь графа алгоритма и таких задач, как быстрое вычисление производной и градиента, быстрое восстановление линейного функционала, оценивание влияния ошибок округления. Хотя граф алгоритма и является важнейшим понятием, связанным с изучением параллельных свойств алгоритмов, тем не менее, ни этот граф, ни все перечисленные только что задачи не имеют к параллелизму никакого прямого отношения. Может быть, между различными алгоритмическими проблемами имеется куда больше связей, чем известно сегодня?

Складывается впечатление, что значение сведений об информационной структуре алгоритмов выходит далеко за рамки параллельных вычислений и, возможно, эти сведения являются стержнем многих исследований в области алгоритмов, реализуемых на вычислительной технике. Если это

так, то первичные знания из данной области должны внедряться в образование. И чем скорее и на более ранних этапах, тем лучше.

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. − СПб:. БХВ-Петербург (2-е изд. 2004), 2002. − 608 с.