ПРОБЛЕМЫ РАЗВИТИЯ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

В. Е. Котов

Введение. Центральной проблемой параллельного программирования была и остается проблема совершенствования методов программного описания параллелизма сложных дискретных систем и процессов, а также методов организации параллельного вычислительного процесса в многопроцессорных параллельных ЭВМ.

Более конкретная постановка задачи заключается в разработке языков параллельного программирования, содержащих развитие средства описания естественного параллелизма задач и средства задания параллелизма вычислений в многопроцессорных системах. Вообще говоря, здесь две разные задачи, так как методология описания параллелизма и методология задания параллельных вычислений различаются; разными являются также требования, предъявляемые к реальным параллельным языкам обоих типов. Однако обе задачи проблемы имеют много общего с точки зрения базовых принципов, лежащих в основе механизмов организации параллельных структур, а различия между ними сводятся в основном к обычным различиям между языками верхних и нижних уровней.

К языкам параллельного программирования предъявляются обычные требования: 1) надежность и одновременно эффективность программ; 2) концептуальная простота и одновременно адекватность широкому кругу задач.

Но комплексу этих требований трудно удовлетворить так, чтобы язык отвечал всем им одновременно, потому что в параллельном программировании контраст между противоречивыми требованиями исключительно велик.

1. Параллелизм в программах и вычислениях. Для обсуждения проблем, связанных с параллелизмом в программировании, нужно уточнить это понятие, выделить его абстрактные и реальные формы на разных стадиях программирования, на разных уровнях программной иерархии.

Трудности начинаются в самом начале: как определить понятие параллельных фрагментов программы (процессов, операторов, операций и т. п.). Определения, связанные с введением времени (реального или условного) и приводящие к таким понятиям, как „процессы, протекающие одновременно“ или „процессы, имеющие возможность протекать одновременно“, вызывают трудности и двусмысленности, если только речь не идет о тривиальных случаях: пара процессов, никак не взаимодействующих друг с другом. Отражением этих трудностей в теории является сложность формальных определений параллелизма, основанных на временных отношениях. Другой вариант определения параллелизма — пространственный: параллельные фрагменты или процессы — это такие, которые реализуются или могут реализоваться на разных, разделенных физических или виртуальных

Статья была опубликована в Трудах Всесоюзного симпозиума „Перспективы системного и теоретического программирования“. Новосибирск, 1979. С. 58–72.

устройствах. Такое определение оказывается более удобным для анализа ситуаций, происходящих при взаимодействии параллельных процессов, имеющих общие ресурсы. Наличие общего ресурса заставляет параллельные процессы реализовываться в одних и тех же точках пространства, что невозможно или нежелательно, откуда и возникает ресурсный конфликт, каким бы этот ресурс не был: внешнее устройство, процессор, ячейка памяти.

Наш подход состоит в том, чтобы вообще уклониться от необходимости определять отношение параллелизма, а определять наоборот отношение зависимости (временной или пространственной) фрагментов или процессов. Этот подход дополнителен по отношению к традиционному и его можно назвать „презумпцией параллельности“: все фрагменты и действия программы считаются изначально независимыми, то есть параллельными, на них накладываются отношения зависимости в виде необходимых ограничений, причем эти отношения носят, в общем случае, динамический характер. Этот подход к организации параллельных структур программ можно считать подходом „от хаоса к порядку“, в отличие от традиционного подхода, когда параллельные структуры конструируются встраиванием в последовательные структуры параллельных фрагментов.

Мы не можем сейчас более детально обсуждать проблему определения понятия параллельных процессов, но хотелось бы подчеркнуть исключительную важность уточнения базовых, фундаментальных понятий, связанных с параллелизмом в программах, так как отсутствие единой основы в трактовке параллелизма приводит и в теории, и на практике к путанице при анализе возможностей, достоинств и недостатков тех или иных методов и средств распараллеливания и синхронизации параллельных процессов.

2. Разнообразие параллельных структур. Для того, чтобы сравнить разные методы организации параллельных программ, нужно выделить из всего разнообразия параллельных структур некоторые типовые ситуации. Ясно, что разнообразие таких ситуаций и соответствующих им параллельных структур велико. Можно ориентировочно указать некоторые основные факторы, определяющие источники этого многообразия, а именно: 1) число (параллельных) фрагментов (процессов); 2) относительные размеры фрагментов; 3) интенсивность и характер взаимодействий между фрагментами (обмены, конфликты из-за ресурсов); 4) наличие и глубина иерархии в параллельной структуре; 5) (потенциальное) число независимых параллельных устройств, реализующих структуру; 6) характер управления сложным параллельным процессом (централизованное, распределенное и т.п.).

Чтобы упростить обсуждение, представим все многообразие параллельных структур в виде прямоугольного пространства, снизу ограниченного параллелизмом вычислительной системы, сверху — параллелизмом задач. В этом пространстве слева направо растет сложность программных фрагментов (и параллельных структур), снизу вверх растет уровень абстракции средств.

3. Современные параллельные языки. Если проследить развитие практики параллельного программирования к настоящему времени, то можно заметить, что она продвигается от периферии этого прямоугольника к центру. Действительно, сверху развиваются неалгоритмические (непроцедурные) методы программирования в языках сверхвысокого уровня, в основе которых лежит отказ от концепции последовательного алгоритма в пользу спецификации моделей задач, которые по существу являются параллельными структурами. Снизу — развитие параллелизма в архитектуре вычислительных машин: от независимых устройств, через параллельные процессоры, до микропрограммирования. Справа — развитие мультипрограммирования и опирающе-

гося на него параллелизма взаимодействующих последовательных процессов. Наконец, слева — развитие векторных матричных, конвейерных и других методов выполнения параллельных операций над регулярными структурами данных.

Параллелизм взаимодействующих процессов (левая часть пространства структур) и параллелизм групповых операций (правая часть пространства) развиваются в рамках последовательно-параллельного подхода. В обоих случаях в последовательную основу языка вводятся средства указания параллельных фрагментов (процессов, ветвей, операций и т. п.). Наибольшее развитие получили в настоящее время языки с асинхронно взаимодействующими процессами (ветвями); для этого типа параллельных структур характерно относительно небольшое число достаточно мощных процессов. Примерами могут служить такие языки как Алгол-68, ПЛ-1, Симула-67, Параллельный Паскаль, Автокод Эльбруса и ряд других, менее известных или экспериментальных языков. Концепция параллелизма процессов родилась в недрах мультипрограммирования и легла в основу архитектуры универсальных высокопроизводительных ЭВМ 4-го поколения и языков программирования для них. Для организации взаимодействия между параллельно (или псевдопараллельно) протекающими процессами необходимы средства синхронизации; наиболее известные из них — семафоры и критические интервалы, общие переменные (shared variables) вместе с критическими интервалами и мониторы. В нашу задачу не входит сравнение этих синхронизирующих примитивов. Отметим только лишь следующее. Семафор — средство нижнего уровня; как недостаток отмечается „неструктурированность“ семафоров, приводящая к трудностям отладки больших программных систем. Наоборот, монитор — средство высокого уровня, обеспечивающий хорошую структурированность программ, но приводящий к потере параллелизма и относительно большим „накладным расходам“, особенно при программировании операционных систем.

Что касается правой части выделенного пространства параллельных структур, то кажется, что проблема создания языка со средствами задания параллельного исполнения выражений и групповых операций проще из-за возможности использования принципа синхронного параллелизма, не требующего специальных синхронизирующих примитивов, из-за возможности организации динамического распараллеливания, в том числе аппаратного. Известны простые расширения последовательных языков программирования, таких как ФОРТРАН, APL, и ряд специально разработанных языков для ЭВМ с матричными, конвейерными и ассоциативными процессорами (LRL TRAN, IVTRAN, TRANQUIL и др.). Однако в перспективе ситуация не представляется простой. Синхронный параллелизм не обладает достаточной гибкостью, чтобы приспосабливаться к динамическим ситуациям, возникающим в универсальных вычислительных системах; средства задания синхронного параллелизма в значительной степени машинно-ориентированы; для обеспечения эффективности вычислений необходима дополнительная работа по приведению алгоритма и структур данных к виду, удобному для синхронного исполнения. С таким положением можно мириться в специализированных вычислениях, например, при программировании определенных классов задач, решаемых на „системах-гигантах“ 4-го поколения (ILLIAC IV, STAR 100 и др.), но для универсального параллельного языка (высокого уровня) требуется переход к более гибким, динамичным непроцедурным методам представления параллельных групповых операций и выражений. При этом возникает проблема улаживания противоречий между непроцедурностью и эффективностью вычислений.

4. Проблема тотального параллелизма. Итак, правая и левая периферии выделенного пространства параллельных структур развиваются, взаимно дополняя друг друга. Раз-

виваются они, в общем, независимо и навстречу друг к другу, но центр этого пространства, содержащий фрагменты ~~ей~~ мощности (такие, как операторы), остается пустым в том смысле, что для этой области нет практически освоенных подходящих средств задания параллелизма. Действительно, механизм параллельных процессов оказывается довольно громоздким в этом диапазоне, а синхронный параллелизм не обеспечивает гибкости вычислений. Возникает вопрос, нужно ли развивать средства описания параллелизма для этой области? Да, такие средства нужны; необходимость создания языков со „сквозным“, тотальным параллелизмом, то есть языков, в которых есть возможность удобно и эффективно задавать параллелизм на любом уровне, диктуется, во-первых, развитием параллелизма в вычислительных системах, где он распространяется от периферий и от ядра (параллельные процессоры) к средней части, и, во-вторых, расширением круга задач, для эффективного решения которых необходим такой сквозной параллелизм. К задачам такого типа относятся задачи описания и конструирования операционных систем, задачи моделирования, особенно моделирование сложных многоуровневых дискретных систем и происходящих в них процессов, задачи искусственного интеллекта, комбинаторики.

Второй вопрос — как нужно развивать средства организации параллелизма в средней части рассматриваемого пространства структур. Уже отмечалось, что применение техники взаимодействующих параллельных процессов (ветвей) и синхронного параллелизма неудобно на этом уровне. Их комбинирование можно рассматривать лишь как временную меру, так как растет общая сложность структуры управления программ и одновременно суммируются недостатки каждой из этих двух техник. Остается поиск новых методов организации параллельных структур для этого диапазона. Если новый метод будет ортогонален двум предыдущим, то сложность структуры управления программ возрастет недопустимо. Следовательно, новый метод должен быть более общим, чтобы механизм организации параллельных процессов и синхронный параллелизм оказались бы его частными случаями. Единый метод организации сквозного параллелизма должен быть концептуально простым, надежным, эффективно реализуемым и адекватным широкому классу „параллельных“ задач. Кроме того, теоретическую основу его должен составлять некоторый полный и замкнутый формализм, допускающий строгую формулировку утверждений о параллельных программах, автоматическое манипулирование программами (для последовательных программ такой теоретической основой является, например, формализм схем программ, включающих набор схем разной степени абстракции — от графов, моделирующих структуры управления программ, до стандартных схем и обобщений. В свою очередь, теория схем программ базируется на теории автоматов и формальных языков).

Единство метода организации параллельных структур на всем выделенном пространстве не означает, что во всех его частях необходимо использовать один и тот же набор программных примитивов для описания или задания параллелизма, так как трудно ожидать, что найдется такой набор, отвечающий одновременно требованиям, предъявляемым к языкам высокого и низкого уровней, одинаково эффективный и в левой, и правой частях этого пространства. Речь идет об унифицированном принципе построения структур управления параллельных программ, но конкретные управляющие примитивы и их число может меняться в разных областях пространства. В частности, нам представляется целесообразным говорить не об одном универсальном языке, а о семействе „совместимых“ языков разных уровней, структуры управления, которых построены на едином принципе.

5. Асинхронное программирование. В настоящее время основой единого метода организации тотального параллелизма становится так называемый асинхронный метод

программирования, базирующийся на упоминавшемся выше подходе „от хаоса к порядку“. В отличие от последовательно-параллельного программирования, при котором последовательная организация программ дополняется введением средств указания параллельности фрагментов, в этом случае все фрагменты программы считаются изначально неупорядоченными, потенциально параллельными. Любое ограничение на порядок их следования (в том числе, задание последовательности фрагментов), формулируется в виде явно указываемых или неявно подразумеваемых условий готовности, которые связаны с каждым фрагментом, динамически проверяют и в любой момент проверки разрешают или не разрешают выполнение этого фрагмента. Если программа имеет иерархическую структуру, то каждый „составной“ фрагмент организован внутри по такому же принципу. Асинхронный метод обладает наивысшей гибкостью в смысле возможности описания или задания „максимально параллельных“ структур, он достаточно легко моделирует последовательно-параллельную организацию программ. Поэтому он и является подходящей основой для разработки программных средств задания тотального параллелизма.

Среди модификаций асинхронного программирования можно выделить два направления, отличающиеся способами организации обменов информацией между фрагментами и характером управления исполнением параллельной программы. В централизованных асинхронных вычислениях существует единая „управляющая“ надстройка, задача которой — проверка условий готовности фрагментов и выбор инициируемых процессов. Обмен информацией (в том числе, управляющей) осуществляется через общую среду (память), доступную как централизованному управлению, так и всем фрагментам. Условия готовности определены на той же памяти или на ее части. В распределенных асинхронных вычислениях (потоковые вычисления) связи по информации и управлению осуществляются через изолированные друг от друга каналы непосредственной связи, представляющие собой памяти динамической структуры (чаще всего, очереди). В потоковых вычислениях каждый фрагмент может самостоятельно определять степень готовности начать вычисления, наиболее частным условием готовности является наличие информации (операндов) во „входных“ каналах.

Теоретически каждый из этих асинхронных методов универсален в смысле обеспечения гибкости параллельных вычислений, разнообразия порождаемых или описываемых параллельных структур и т. п., но в реальной обстановке они имеют свои слабые и сильные стороны, как правило, взаимно дополняющие друг друга. Так, использование неявных условий готовности в потоковом программировании облегчает „естественное“ описание параллельных структур на языках высокого уровня, а организация взаимодействий через распределенные программные каналы связи хорошо соответствует реализации вычислений в распределенных вычислительных системах. С другой стороны, потоковая организация программ не учитывает конфликтных ситуаций, возникающих при наличии общих ресурсов. В связи с этим возникают трудности, связанные с отображением „идеальной“ потоковой схемы вычислений на „реальные“ конфигурации систем с ограниченными ресурсами (в том числе, с ограниченным числом процессоров). Наконец, определенные трудности вызывает использование потокового метода при программировании вычислений „непотокового“ характера, например, при обработке сложных структур данных. Централизованные асинхронные структуры обладают большими возможностями при разрешении конфликтных ситуаций, более разнообразными могут быть условия готовности, легче осуществляется отображение структур программ на структуры вычислительных систем. В целом, централизованная организация оказывается более мощной, чем распределенная, но становится

громоздкой там, где по существу нет необходимости централизовать процесс управления параллельными вычислениями. Поэтому разумным компромиссом можно считать такой вариант асинхронной организации программ, когда явные средства указания готовности фрагментов (спусковые функции) сочетаются с неявными (наличие операндов на входах операторов и т. п.), а управление частично централизовано, причем его централизованная часть контролирует только ту часть программы, в которой есть явные средства указания готовности фрагментов. Управление и обмены осуществляются через общую память, имеются общие ресурсы.

6. Программируемые структуры управления. Таким образом, принцип частично централизованного асинхронного программирования является основой для создания универсальных языков параллельного программирования. Для того чтобы языки отвечали упоминавшимся выше требованиям и позволяли удобно и эффективно программировать все многообразие параллельных структур и процессов, они должны иметь развитый инструмент конструирования разнообразных структур управления. В современных языках имеются, например, развитые средства конструирования выражений и структур данных: те же разделы языков, которые содержат управляющие программные примитивы, довольно бедны. Так, в последовательных языках это — фиксированный набор условных и безусловных средств композиции операторов, в параллельно-последовательных языках добавлены средства параллельной композиции и синхронизирующие примитивы, в асинхронных языках — спусковые функции и их аналоги. Семантика исполнения выражений и групповых операций над сложными структурами данных, как правило, фиксируется языком и неуправляема. В то же время, разнообразие параллельных структур и необходимость эффективного, надежного и удобного параллельного программирования ставит новую задачу, а именно — снабжение языков развитыми механизмами конструирования разнообразных структур управления. Универсальный параллельный язык должен давать программисту возможность создавать и фиксировать в виде подходящих программных конструкций различные варианты структур управления, необходимые и удобные для его программы и для вычислительной системы, на которой эта программа будет исполняться. Другими словами, язык должен предоставлять возможность программировать структуры управления на всех уровнях — от уровня крупных процессов до уровня групповых операций — так как, например, в современных языках описываются разнообразные и сложные структуры данных. Для этого необходимо создание формального аппарата, образующего теоретическую базу структур управления, и развитие в рамках этого аппарата алгебры структур управления.

Идея программируемых структур управления реализуется в концепции управляющих выражений (path expression), развиваемой в работах Хабермана, Кэмпбелла, Лауера, и в концепции типов управления, разрабатываемой в Вычислительном центре СО АН СССР. В обоих случаях формальной основой механизмов порождения структур управления служит некоторая алгебра сетей Петри или их обобщений. Управляющие выражения и типы управления описываются программистом в виде формул, построенных из операндов, которыми являются операторы (или идентификаторы, имеющие операторы), и операций над структурами (сетями). Структура программы и ее составных фрагментов описывается в явном виде, наравне с описанием выражений и структур данных. Управление исполнением выражений и групповых операций также может описываться этими средствами, но целесообразней (особенно в языках

верхних уровней) использовать непроцедурные методы с привлечением неявных методов управления, семантика которых задается той же техникой.

7. Параллельная обработка структурных данных. Значительным резервом повышения скорости вычислений является параллельное исполнение операций над независимыми компонентами сложных структур данных, таких как векторы, матрицы, множества, деревья и т. п. Аналогично, резервом повышения эффективности программирования является переход к непроцедурным „неделимым“ (с точки зрения программиста) операциям над такими структурами. Для этой цели необходимо дальнейшее усовершенствование и развитие средств описания и конструирования разнообразных структур данных. В параллельных вычислениях особую роль играет распараллеливание доступа к хранимым данным, децентрализация памяти, предварительная аранжировка данных для наиболее эффективной обработки. Структура памяти в мультипроцессорных системах усложняется вместе с усложнением структур данных пользователя, растет разнообразие специально организованных типов запоминающих устройств. Поэтому в параллельных языках проблема хранения информации и доступа к ней должна трактоваться как важная самостоятельная функция, реализуемая специальными подъязыками для работы с памятью. Одновременно, операции, определяемые обычно для скалярных данных, должны автоматически обобщаться на произвольные структуры за счет применения явных или неявных механизмов „просачивания“ этих операций до уровня скалярных компонент. Применение неявного асинхронного принципа управления исполнением выражений со сложными структурами данных (например, потокового принципа) и сочетание его с автоматическим просачиванием операций позволяют осуществить удобное непроцедурное программирование выражений и гарантирует их последующее эффективное исполнение на параллельных процессорах разнообразных конфигураций.

8. Распараллеливание программ, алгоритмов и задач. Распараллеливание вычислений может осуществляться на разных этапах программирования: непосредственно при спецификации и программировании задачи на параллельном языке, при компиляции последовательной программы, непосредственно в процессе исполнения программы на многопроцессорной системе. На каждом из этих этапов имеются особенности в методике распараллеливания и в выборе тех типов параллелизма, которые целесообразно обнаруживать в программах.

В процессе вычислений целесообразно осуществлять динамическое распараллеливание небольших отрезков программы и отдельных операций (в том числе, скалярных выражений и групповых операций над структурными данными), основываясь на результатах промежуточных вычислений. Чтобы избежать потерь времени на анализ и реализацию параллелизма, необходимо стремиться к аппаратной реализации алгоритмов динамического распараллеливания.

Распараллеливание последовательных программ основано на анализе их информационно-логических связей. Автоматизация такого распараллеливания в общей постановке вряд ли окажется реализуемой ввиду большой сложности алгоритмов анализа. Следует выделить наиболее легко распараллеливаемые ситуации (независимые процедуры, циклически повторяемая обработка массивов и т. п.), а также использовать дополнительные средства индикации возможного параллелизма в помощь автоматическому распараллеливанию.

Сейчас накоплен большой арсенал оптимальных последовательных алгоритмов, которые следовало бы автоматически распараллелить. Однако сложность автоматического

распараллеливания, а также тот факт, что прямолинейно распараллеленные последовательные алгоритмы могут приводить к неоптимальным параллельным программам, заставляет обращать все большее внимание на методы непосредственного конструирования параллельных алгоритмов и программ. Создание новых параллельных вычислительных методов решения задач позволило бы более внимательно учитывать специфику задач из разных проблемных областей и специфику разных типов многопроцессорных ЭВМ. Кроме того, исчез бы „третий лишний“ этап на пути от задачи к вычислениям, а именно — этап составления последовательного алгоритма с последующим его распараллеливанием. Работы в данном направлении начаты сравнительно недавно, но интенсивно развиваются. Важно, чтобы они были подкреплены адекватными языковыми средствами, о чем говорилось выше.

9. Проблемы семантики параллельных языков и программ. Исследования в области семантики и языков программирования нацелены на автоматизацию синтеза и верификации программ, а также на выделение и обоснование конструкций перспективных языков, обеспечивающих эффективное и надежное программирование. Особенно важны эти исследования для параллельного программирования, так как процессы конструирования и исполнения параллельных программ существенно сложнее. Первыми с проблемой надежного программирования: сложных недетерминированных взаимодействий параллельных процессов столкнулись разработчики операционных систем. Типичным подходом к решению этих проблем было выделение типичных ситуаций, в которых могли возникнуть ошибки (задачи о читателях-писателях, пяти философах и т.п.), и поиск правильных частных решений. Общие подходы начали развиваться позднее, и они отталкивались от разработанных к этому времени методов формализации семантики последовательных программ и языков. Например, предлагались обобщения аксиоматического метода Хоара на случай параллельных программ. Денотационный подход к описанию математической семантики переносился на простые параллельные программы, организованные на потоковом принципе. Однако сколько-нибудь удовлетворительного решения эта проблема еще не нашла. По-видимому, причина такой ситуации не только в объективной сложности параллельных вычислений. Отсутствие достаточно широкого практического опыта в области не позволяет еще четко выделить в ней новые понятия и явления, в терминах которых можно было бы свободно и единообразно рассуждать о параллельных программах и их поведении, а затем формализовать эти понятия.

10. Модель – язык – архитектура. Значение как теоретических исследований, так и практических разработок в области параллельного программирования усиливается тем фактом, что они непосредственно воздействуют на поиск новых архитектурных решений при создании перспективных ЭВМ. Общепринятым стал взгляд на вычислительную машину как языковый процессор, и многие современные проекты ЭВМ начинаются с выбора или разработки концептуального языка, лежащего в основе архитектуры машины. Естественно, что для создания высокопроизводительных и одновременно простых для проектирования и эксплуатации параллельных вычислительных систем необходимы параллельные языки, обладающие теми же свойствами. Именно в языках программирования формируются те конструкции, с помощью которых можно приводить к общему знаменателю математические понятия из решаемых задач и инженерные схемы машин. В свою очередь, для разработки параллельных языков, которые могли бы составить надежную базу для дальнейшего развития архитектуры, необходимы создание и основательная теоретическая проработка моделей вычислений, лежащих в основе этих языков, и, следовательно, архи-

тектуры. Такие модели должны отличаться: 1) принципиальной новизной, обеспечивающей качественно ее высокие характеристики машинных вычислений; 2) концептуальной простотой, упрощающей структурную сложность программ и машин; 3) универсальностью, преемственностью и расширяемостью, практичностью.

К настоящему времени уже предложено большое число интересных моделей параллельных вычислений и задача состоит в практической проверке их свойств путем овеществления их в форме языков программирования.

В целом, в намеченной цепочке „модель – язык – архитектура“ центральной проблемой сейчас является проблема параллельного языка, как и отмечалось в самом начале доклада.