Dataflow – парадигма программирования будущего

А.В. Климов, Н.Н. Левченко, А.С. Окунев, Д.Н. Змеев

Институт проблем проектирования в микроэлектронике Российской академии наук,
(IPPM RAS), Москва, Россия
klimov@ippm.ru, nick@ippm.ru, oku@ippm.ru, zmejevdn@ippm.ru

С ростом параллелизма «железа» становится все сложнее его «насытить», оставаясь в рамках традиционного программирования по фон-Нейману. В чем причина проблем?

Можно выделить две основные:

1. Представление об алгоритме как о линейной последовательности действий – нити управления. Нить при работе имеет всегда ровно одну точку управления, поэтому будем называть ее *одноточечной*.
2. Преставление о взаимодействии по линии процессор – память как о состоящем, по сути, только из двух операций: чтения и записи по адресу. При этом есть только одна возможность гарантированно прочитать то, что записано – выполнить чтение позже записи. Когда нить одна, проблем нет, не считая необходимости в поддержке механизма кэширования (для скорости). Но если много?

(Эти два представления восходят еще к Тьюрингу. Только там вместо программной нити – конечный автомат, а вместо памяти – бесконечная в две стороны лента. Работа выполняется пошагово, то есть последовательно.)

Посмотрим, к чему это ведет в связи с задачей загрузить работой систему из большого числа элементов, способных работать параллельно.

По п.1 не приходит в голову ничего иного, кроме того как представить параллельный алгоритм как семейство одноточечных нитей, которые друг с другом «взаимодействуют» - одни передают сообщения, другие ждут и принимают. Так возникает модель CSP, она же SPMD с MPI. (GPGPU – та же нить, только выполняемая над длинным вектором. Или, что иногда приятнее – много нитей, по одной для каждого элемента, но уже не взаимодействующих.). И как развитие возникает представление о вложенном параллелизме: одна нить распадается на много, которые должны быть все завершены, прежде чем родительская нить продолжит работу. Последнее означает барьер, который может приводить к простоям.

Однако, п.2 теперь приносит одни проблемы: как обеспечить прочтение другой нитью записанное данное? как узнать в другой нити, что данные записаны и можно их читать?

Мы видим альтернативу в том, чтобы алгоритм сразу представлять как многоточечный, содержащий много точек управления. Одна активная операция, команда может «передать управление» нескольким, тогда как новая для начала своей работы может требовать нескольких передач управления ей. Вместе с «передачей управления» логично всегда передавать некоторый объем требуемых данных. Так возникает модель программирования dataflow, со специфическим механизмом управления – по готовности входных данных. Для обеспечения многократности повторения однотипных действий, на чем стоит всякое программирование – вводим для всякой команды (узла) *контекст*, он же адрес или набор индексов. Передача управления на начало узла должна содержать, кроме имени (номера) узла еще и необходимый набор индексов, определяющих конкретный экземпляр узла-получателя.

Заметим, что теперь обычная память не нужна. Всякое значение, необходимое для совершения операции узла просто должно быть отправлено (другим каким-то узлом или инициатором) ему на один из входов. С другой стороны, когда на одни входы данные уже пришли, то до прихода остальных

они должны где-то храниться. Это и есть память, только теперь – ассоциативная, в ней «адресом» является имя/номер узла с набором его индексов. Будем называть это *ключом*.

Теперь интерфейс между процессором - исполнителем (многих) готовых активностей и этой «памятью» выглядит сложнее: процессор посылает в память данные с указанием ключа и номера-имени входа, а «память» добавляет их к накапливаемым на входах узлов, и, если возникает полный комплект – отправляет в исполнитель задание на выполнение операции узла (она может представлять собой небольшую программу в обычной парадигме, которая с переданными ей данными отработает от начала до конца, без каких-либо ожиданий, и, возможно породит посылки новых данных в другие узлы), одновременно стирая (по умолчанию) использованные данные со своих входов.

Но сразу решается много проблем: узловая операция всегда имеет нужные ей данные «под рукой», при этом никто никого не ждет: активность возникает ровно тогда, когда все нужные ей данные есть, и, сделав свое дело, исчезает, передав потенциалы активностей в виде результирующих данных другим. Причем ответа ждать не надо: все такие сообщения односторонние.

В докладе будут показаны примеры программирования в новой парадигме (от простых до сложных), обсуждены ее достоинства и проблемы. Также будет представлен эскиз архитектуры эффективной исполняющей системы [1-4].

Литература

1. Стемпковский А.Л., Левченко Н.Н., Окунев А.С, Цветков В.В. Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов // Информационные технологии. 2008. №10. С. 2 – 7.

2. А. В. Климов, Н. Н. Левченко, А. С. Окунев, А. Л. Стемпковский.

Суперкомпьютеры, иерархия памяти и потоковая модель вычислений // Программные системы: теория и приложения: электрон. научн. журн. 2014. Т. 5, № 1(19), с. 15–36.

3. Н.Н. Левченко, А.С. Окунев, А.Л. Стемпковский. Использование модели вычислений с управлением потоком данных и реализующей ее архитектуры для систем эксафлопсного уровня производительности // Сборник трудов V Всероссийской научно-технической конференции «Проблемы разработки перспективных микро- и наноэлектронных систем – 2012» (МЭС-2012), Москва, ИППМ РАН, 2012. С. 459-462.

4. Д.Н. Змеев, Н.Н. Левченко, А.С. Окунев, А.В. Климов. Способы регулирования вычислений в параллельной потоковой вычислительной системе // Сборник трудов VI Всероссийской научно-технической конференции «Проблемы разработки перспективных микро- и наноэлектронных систем – 2014» (МЭС-2014), Москва, ИППМ РАН, 2014.