Уже по первому взгляду на дисциплину становится ясно, насколько много существует различных способов организации параллельных вычислительных систем. Здесь можно назвать векторно-конвейерные компьютеры, массивно-параллельные и матричные системы, компьютеры с широким командным словом, спецпроцессоры, кластеры, компьютеры с многопоточной архитектурой

, систолические массивы или dataflow-компьютеры. Если же к подобным названиям для полноты описания добавить и сведения о таких важных параметрах, как организация памяти, топология связи между процессорами, синхронность работы отдельных устройств или способ исполнения операций, то число различных архитектур станет и вовсе необозримым.

Почему параллельных архитектур так много? Как они взаимосвязаны между собой? Какие основные факторы характеризуют каждую архитектуру? Поиск ответа на такие вопросы так или иначе приводит к необходимости *классификации архитектур вычислительных систем* [23]. Активные попытки в этом направлении начались после опубликования М. Флинном в конце 60-х годов прошлого столетия первого варианта классификации, который, кстати, используется и в настоящее время.

Вообще говоря, при введении классификации можно преследовать самые разные цели. С точки зрения главного инженера организации, где устанавливается компьютер, деление компьютеров по мощности потребляемой электроэнергии, конечно же, будет классификацией. Планово-финансовый отдел больше интересует стоимость. Если окажется, что установка двадцати компьютеров в локальной сети обойдется во столько же, во сколько и восьмипроцессорный сервер с общей памятью, то для него это будут вычислительные системы одного класса. И правильно. Что заложили в основу классификации, такие следствия и получаем.

(Майкл Флинн – американский профессор Стэнфордского университета в сфере вычислительной техники).

Ясно, что навести порядок в хаосе очень важно для лучшего понимания исследуемой предметной области. В самом деле, вспомним открытый в 1869 году Д. И. Менделеевым периодический закон. Выписав на карточках названия химических элементов и указав их важнейшие свойства, он сумел найти такое расположение, при котором четко прослеживалась закономерность в изменении свойств элементов, расположенных в каждом столбце и каждой строке. Теперь, зная положение элемента в таблице, он мог с большой степенью точности описать его свойства, не проводя с ним никаких непосредственных экспериментов. Другим, поистине фантастическим следствием, явилось то, что данный закон указал на несколько "белых пятен" в таблице и позволил предсказать не только существование, но и свойства пока неиз-

вестных элементов. В 1875 году французский химик Поль Эмиль Лекок де Буабодран, изучая спектры минералов, открыл предсказанный Менделеевым галлий и впервые подробно описал его свойства. В свою очередь Менделеев, никогда прежде не видевший данного химического элемента, на основании введенной классификации смог и указать на ошибку в определении плотности галлия, и вычислить правильное значение.

Что-то похожее хотелось бы найти и для архитектур параллельных вычислительных систем. Главный вопрос — что заложить в основу классификации, может решаться по-разному. Однако с позиций наших исследований классификация должна давать ключ к пониманию того, как решать задачу эффективного отображения алгоритмов на архитектуру вычислительных систем. В некоторых случаях вводят описания классов компьютеров. На основе информации о принадлежности компьютера к конкретному классу пользователь сам принимает решение о способе записи алгоритма в терминах выбранной технологии параллельного программирования. Иногда в качестве классификации пытаются ввести формальное описание архитектуры, используя специальную нотацию. Такое направление интересно тем, что создается благоприятная основа для построения автоматизированных систем отображения. В самом деле, с одной стороны, есть формальное описание архитектуры целевого компьютера, с другой стороны, есть формальное описание алгоритма. Созданы условия для совместного исследования этих двух объектов и последующего синтеза оптимальной программы. Правда, результат очень сильно зависит как от качества введенных описаний, так и от методов их совместного исследования. О степени решенности даже упрощенного варианта данной проблемы читатель может судить по качеству компиляторов, работающих на существующих параллельных компьютерах.

Классификация М. Флинна (M. Flynn). По-видимому, самой ранней и наиболее известной является классификация архитектур вычислительных систем, предложенная в 1966 году М. Флинном. Классификация базируется на понятии *потока*, под которым понимается последовательность команд или данных, обрабатываемая процессором. На основе числа потоков команд и потоков данных Флинн выделяет четыре класса архитектур.

SISD (Single Instruction stream/Single Data stream) — одиночный поток команд и одиночный поток данных (рис. 3.1). На рисунках, иллюстрирующих классификацию М. Флинна, использованы следующие обозначения:

ПР — это один или несколько процессорных элементов, УУ — устройство управления, ПД — память данных. К классу SISD относятся, прежде всего, классические последовательные машины или, иначе, машины фоннеймановского типа, например, PDP-11 или VAX 11/780. В таких машинах есть только один поток команд, все команды обрабатываются последовательно друг за другом и каждая команда инициирует одну скалярную операцию. Не имеет значения тот факт, что для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка: как машина CDC 6600 со скалярными функциональными устройствами, так и CDC 7600 с конвейерными попадают в этот класс.

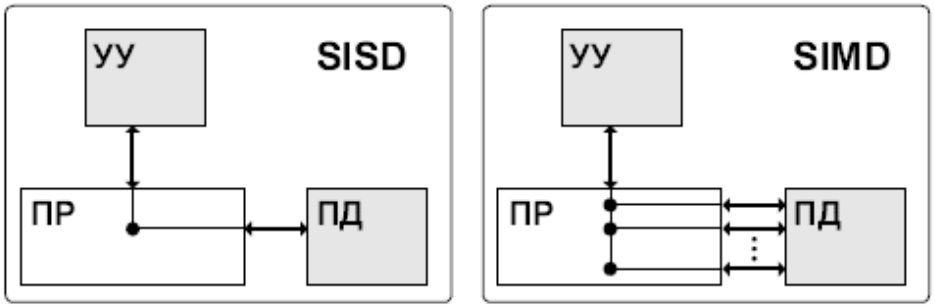


Рис. 3.1. Классы SISD и SIMD классификации М. Флинна

SIMD (Single Instruction stream/Multiple Data stream) — одиночный поток команд и множественный поток данных (см. рис. 3.1). В архитектурах подобного рода сохраняется один поток команд, включающий, в отличие от предыдущего класса, векторные команды. Это позволяет выполнять одну арифметическую операцию сразу над многими данными, например, над элементами вектора. Способ выполнения векторных операций не оговаривается, поэтому обработка элементов вектора может производиться либо процессорной матрицей, как в ILLIAC IV, либо с помощью конвейера, как, например, в машине Cray-1.

MISD (Multiple Instruction stream/Single Data stream) — множественный поток команд и одиночный поток данных (рис. 3.2). Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных. Однако ни Флинн, ни другие специалисты в области архитектуры компьютеров до сих пор не смогли представить убедительный пример реально существующей вычислительной системы, построенной на данном принципе. Ряд исследователей относят конвейерные машины к данному классу, однако это не нашло окончательного признания в научном сообществе. Будем считать, что пока данный класс пуст.

MIMD (Multiple Instruction stream/Multiple Data stream) — множественный поток команд и множественный поток данных (см. рис. 3.2). Этот класс

предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных.

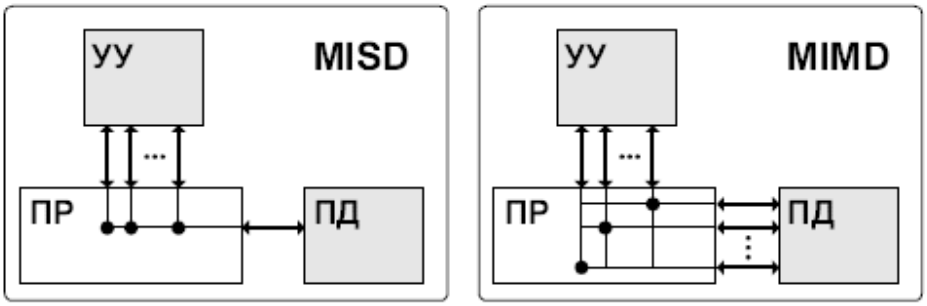


Рис. 3.2. Классы MISD и MIMD классификации М. Флинна

Итак, что же собой представляет каждый класс? В SISD, как уже говорилось, входят однопроцессорные последовательные компьютеры типа VAX 11/780. Однако многими критиками подмечено, что в этот класс можно включить и векторно-конвейерные машины, если рассматривать вектор как одно неделимое данное для соответствующей команды. В таком случае в этот класс попадут и такие системы, как Cray-1, CYBER 205, машины семейства FACOM VP и многие другие.

Бесспорными представителями класса SIMD считаются матрицы процессоров: ILLIAC IV, ICL DAP, Goodyear Aerospace MPP, Connection Machine 1 и т. п. В таких системах единое управляющее устройство контролирует множество процессорных элементов. Каждый процессорный элемент получает от устройства управления в каждый фиксированный момент времени одинаковую команду и выполняет ее над своими локальными данными. Для классических процессорных матриц никаких вопросов не возникает. Однако в этот же класс можно включить и векторно-конвейерные машины, например, Cray-1. В этом случае каждый элемент вектора надо рассматривать как отдельный элемент потока данных.

Класс MIMD чрезвычайно широк, поскольку включает в себя всевозможные мультипроцессорные системы: Cm*, C.mmp, Cray Y-MP, Denelcor HEP, BBN Butterfly, Intel Paragon, Cray T3D и многие другие. Интересно то, что если конвейерную обработку рассматривать как выполнение последовательности различных команд (операций ступеней конвейера) не над одиночным векторным потоком данных, а над множественным скалярным потоком, то все рассмотренные выше векторно-конвейерные компьютеры можно расположить и в данном классе.

Предложенная схема классификации вплоть до настоящего времени является самой применяемой при начальной характеристике того или иного компьютера. Если говорится, что компьютер принадлежит классу SIMD или MIMD,

то сразу становится понятным базовый принцип его работы, и в некоторых случаях этого бывает достаточно. Однако видны и явные недостатки. В частности, некоторые заслуживающие внимания архитектуры, например dataflow и векторно-конвейерные машины, четко не вписываются в данную классификацию. Другой недостаток — это чрезмерная заполненность класса MIMD. Необходимо средство, более избирательно систематизирующее архитектуры, которые по Флинну попадают в один класс, но совершенно различны по числу процессоров, природе и топологии связи между ними, по способу организации памяти и, конечно же, по технологии программирования.

Классификация Р. Хокни (R. Hockney). Р. Хокни разработал свой подход к классификации для более детальной систематизации компьютеров, попадающих в класс MIMD по систематике М. Флинна. Как отмечалось выше, класс MIMD чрезвычайно широк и объединяет целое множество различных типов архитектур. Пытаясь систематизировать архитектуры внутри этого класса, Р. Хокни получил иерархическую структуру, представленную на рис. 3.3.

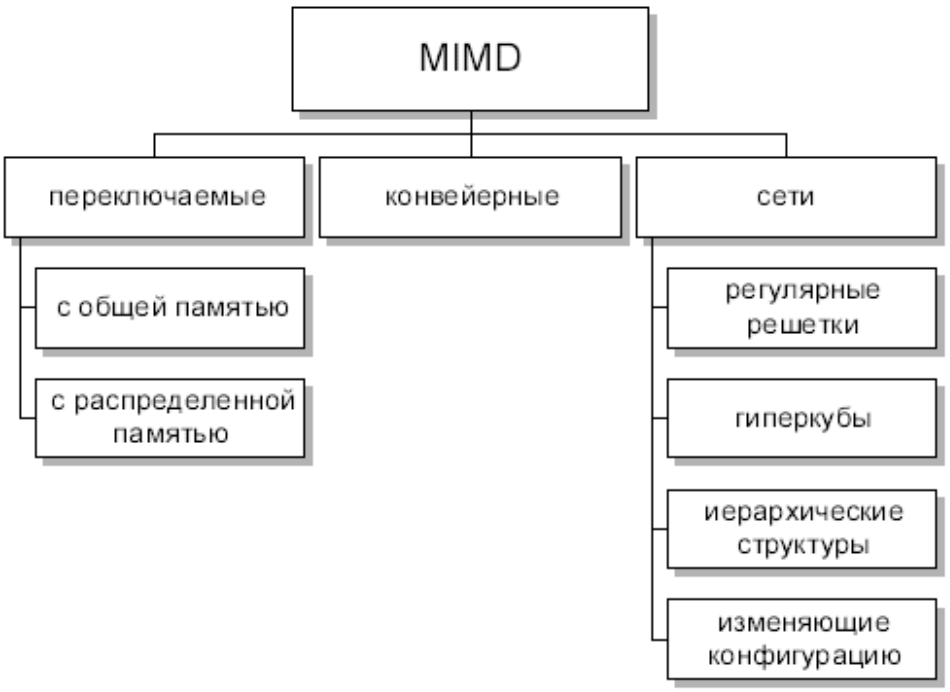


Рис. 3.3. Дополнительная классификация Р. Хокни класса MIMD

Основная идея классификации состоит в следующем. Множественный поток команд может быть обработан двумя способами: либо одним конвейерным устройством обработки, работающим в режиме разделения времени для отдельных потоков, либо каждый поток обрабатывается своим собственным устройством. Первая возможность используется в MIMD-компьютерах, которые автор называет конвейерными. Сюда можно отнести, например, процессорные модули в Denelcor HEP или компьютеры семейства Tera MTA.

Архитектуры, использующие вторую возможность, в свою очередь, опять делятся на два класса. В первый класс попадают MIMD-компьютеры, в которых возможна прямая связь каждого процессора с каждым, реализуемая с помощью переключателя. Во втором классе находятся MIMD-компьютеры, в которых прямая связь каждого процессора возможна только с ближайшими соседями по сети, а взаимодействие удаленных процессоров поддерживается специальной системой маршрутизации.

Среди MIMD-машин с переключателем Хокни выделяет те, в которых вся память распределена среди процессоров как их локальная память (например, PASM, PRINGLE, IBM SP2 без SMP-узлов). В этом случае общение самих процессоров реализуется с помощью сложного переключателя, составляющего значительную часть компьютера. Такие машины носят название MIMD-машин с распределенной памятью. Если память это разделяемый ресурс, доступный всем процессорам через переключатель, то MIMD-машины являются системами с общей памятью (BBN Butterfly, Cray C90). В соответствии с типом переключателей можно проводить классификацию и далее: простой переключатель, многокаскадный переключатель, общая шина и т. п. Многие современные вычислительные системы имеют как общую разделяемую память, так и распределенную локальную. Такие системы автор рассматривает как гибридные MIMD с переключателем.

При рассмотрении MIMD-машин с сетевой структурой считается, что все они имеют распределенную память, а дальнейшая классификация проводится в соответствии с топологией сети: звездообразная сеть (ICAP), регулярные решетки разной размерности (Intel Paragon, Cray T3D), гиперкубы (NCube, Intel iPSC), сети с иерархической структурой, такой как деревья, пирамиды, кластеры (Cm*, CEDAR) и, наконец, сети, изменяющие свою конфигурацию.

Классификация Т. Фенга (T. Feng). В 1972 году Т. Феng предложил классифицировать вычислительные системы на основе двух простых характеристик. Первая — число n бит в машинном слове, обрабатываемых параллельно при выполнении машинных инструкций. Практически во всех современных компьютерах это число совпадает с длиной машинного слова. Вторая характеристика равна числу слов m, обрабатываемых одновременно данной вычислительной системой. Немного изменив терминологию, функ-

ционирование любого компьютера можно представить как параллельную обработку n битовых слоев, на каждом из которых независимо преобразуются m бит. Опираясь на такую интерпретацию, вторую характеристику называют шириной битового слоя.

Каждую вычислительную систему C можно описать парой чисел (n, m). Произведение $P = n \times m$ определяет интегральную характеристику потенциала параллельности архитектуры, которую Фенг назвал *максимальной степенью параллелизма* вычислительной системы. По существу, данное значение есть не что иное, как пиковая производительность, выраженная в других единицах. В период появления данной классификации, а это начало 70-х годов прошлого столетия, еще казалось возможным перенести понятие пиковой производительности как универсального средства сравнения и описания потенциальных возможностей компьютеров с традиционных последовательных машин на параллельные. Понимание того факта, что пиковая производительность сама по себе не столь важна, пришло позднее, и данный подход отражает, естественно, степень осмысления специфики параллельных вычислений того времени.

Рассмотрим компьютер Advanced Scientific Computer фирмы Texas Instruments (TI ASC). В основном режиме он работает с 64-разрядным словом, причем все разряды обрабатываются параллельно. Арифметико-логическое устройство имеет четыре одновременно работающих конвейера, содержащих по восемь ступеней. При такой организации $4 \times 8 = 32$ слова могут обрабатываться одновременно (то есть 32 бита в каждом битовом слое), и значит компьютер TI ASC может быть представлен в виде (64, 32).

На основе введенных понятий все вычислительные системы можно разделить на четыре класса.

Разрядно-последовательные, пословно-последовательные ($n = m = 1$). В каждый момент времени такие компьютеры обрабатывают только один двоичный разряд. Представителем данного класса служит давняя система MINIMA с естественным описанием (1, 1).

Разрядно-параллельные, пословно-последовательные ($n > 1$, $m = 1$). Большинство классических последовательных компьютеров, так же как и многие вычислительные системы, используемые сейчас, принадлежат к данному классу: IBM 701 с описанием (36, 1), PDP-11 с описанием (16, 1), IBM 360/50 и VAX 11/780 — обе с описанием (32, 1).

Разрядно-последовательные, пословно-параллельные ($n = 1$, $m > 1$). Как правило вычислительные системы данного класса состоят из большого числа одноразрядных процессорных элементов, каждый из которых может независимо от остальных обрабатывать свои данные. Типичными примерами служат STARAN (1, 256) и MPP (1, 16384) фирмы Goodyear Aerospace, прототип известной системы ILLIAC IV компьютер SOLOMON (1, 1024) и ICL DAP (1, 4096).

Разрядно-параллельные, пословно-параллельные ($n > 1$, $m > 1$). Подавляющее большинство параллельных вычислительных систем, обрабатывая одновременно $m \times n$ двоичных разрядов, принадлежит именно к этому классу: ILLIAC IV (64, 64), TI ASC (64, 32), C.mmp (16, 16), CDC 6600 (60, 10), BBN Butterfly GP1000 (32, 256).

Недостатки предложенной классификации достаточно очевидны и связаны со способом вычисления ширины битового слоя m. По существу Фенг не делает никакого различия между процессорными матрицами, векторно-конвейерными и многопроцессорными системами. Не делается акцент на том, за счет чего компьютер может одновременно обрабатывать более одного слова: множественности функциональных устройств, их конвейерности или же какого-то числа независимых процессоров. Если в системе N независимых процессоров имеют каждый по F конвейерных функциональных устройств с длиной конвейера L, то для вычисления ширины битового слоя надо просто найти произведение $N \times F \times L$.

Конечно же, опираясь на данную классификацию, достаточно трудно, а иногда и просто невозможно, осознать специфику той или иной вычислительной системы. Достоинством же можно считать *введение единой числовой метрики* для всех типов компьютеров, позволяющей сравнить любые два компьютера между собой.

Классификация В. Хендлера (W. Handler). В основу классификации В. Хендлер закладывает явное описание возможностей параллельной и конвейерной обработки информации вычислительной системой. При этом он намеренно не рассматривает различные способы связи между процессорами и блоками памяти, а считает, что коммуникационная сеть может быть нужным образом сконфигурирована и будет способна выдержать предполагаемую нагрузку.

Предложенная классификация базируется на различии между тремя уровнями обработки данных в процессе выполнения программ:

- уровень выполнения программы; опираясь на счетчик команд и некоторые другие регистры, устройство управления (УУ) производит выборку и дешифрацию команд программы;
- уровень выполнения команд; арифметико-логическое устройство компьютера (АЛУ) исполняет команду, выданную ему устройством управления;
- уровень битовой обработки; все элементарные логические схемы процессора (ЭЛС) разбиваются на группы, необходимые для выполнения операций над одним двоичным разрядом.

Подобная схема выделения уровней предполагает, что вычислительная система содержит какое-то число процессоров, каждый со своим устройством управления. Каждое устройство управления связано с несколькими арифметико-логическими устройствами, исполняющими одну и ту же операцию в каждый конкретный момент времени. Наконец, каждое АЛУ объединяет

несколько групп элементарных логических схем, ассоциированных с обработкой одного двоичного разряда (число групп ЭЛС есть не что иное, как длина машинного слова). Если на какое-то время не рассматривать возможность конвейеризации, то число устройств управления k, число арифметико-логических устройств d в каждом устройстве управления и число групп ЭЛС w в каждом АЛУ составят тройку для описания данной вычислительной системы C: $t(C) = (k, d, w)$.

В таких обозначениях описания некоторых хорошо известных вычислительных систем будут выглядеть следующим образом:

t(MINIMA) = (1,1,1);

t(IBM 701) = (1,1,36);

t(SOLOMON) = (1,1024,1);

t(ILLIAC IV) = (1,64,64);

t(STARAN) = (1,8192,1) — в полной конфигурации;

t(C.mmp) = (16,1,16) — основной режим работы;

t(PRIME) = (5,1,16);

t(BBN Butterfly GP1000) = (256,1,32).

Несмотря на то, что перечисленным системам присущ параллелизм разного рода, он без особого труда может быть отнесен к одному из трех выделенных уровней.

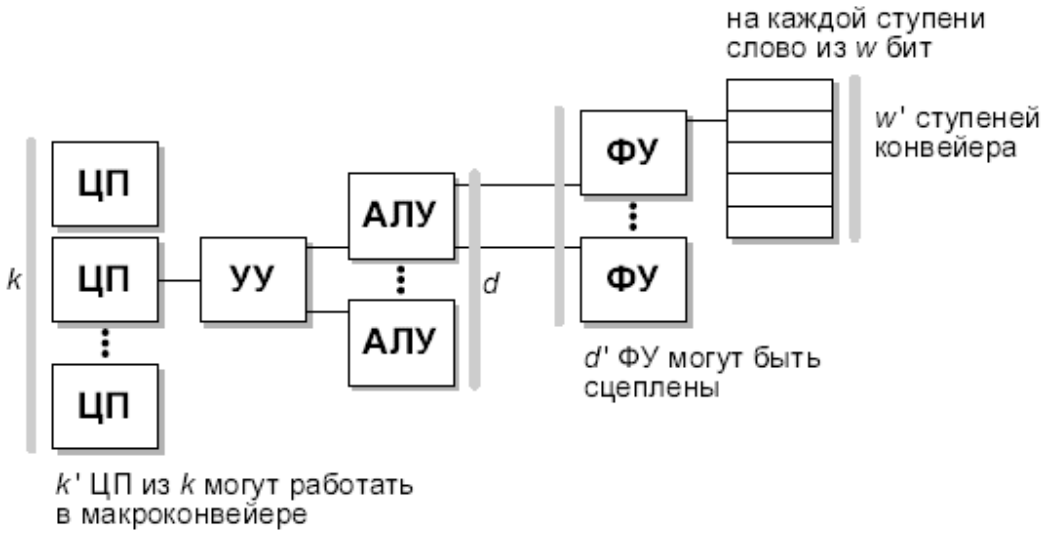


Рис. 3.4. Уровни параллелизма в классификации В. Хендлера

Теперь можно расширить возможности описания, допустив возможность конвейерной обработки на каждом из уровней (рис. 3.4). В самом деле, конвейерность на самом нижнем уровне, т. е. на уровне ЭЛС, это конвейер-

ность функциональных устройств. Если функциональное устройство обрабатывает w-разрядные слова на каждой из w' ступеней конвейера, то для характеристики параллелизма данного уровня естественно рассмотреть произведение $w \times w'$. Знак "×" будем использовать на каждом уровне, чтобы отделить число, представляющее степень параллелизма, от числа ступеней в конвейере. Компьютер TI ASC имеет четыре конвейерных устройства по восемь ступеней в каждом для обработки 64-разрядных слов, следовательно, он может быть описан так:

$$t(\text{TI ASC}) = (1, 4, 64 \times 8).$$

Следующий уровень конвейерной обработки — это конвейеризация на уровне команд. Предполагается, что в вычислительной системе есть несколько функциональных устройств, которые могут работать одновременно в режиме конвейера (для обозначения данной возможности часто используют другой термин — зацепление функциональных устройств). Классическим примером этому служат векторно-конвейерные компьютеры фирмы Cray Research. Другим примером является машина CDC 6600, содержащая десять независимых последовательных функциональных устройств, способных подавать результат своей работы на вход другим функциональным устройствам: $t(\text{CDC 6600}) = (1{,}1 \times 10{,}60)$ (описан только центральный процессор без учета управляющих и периферийных подсистем).

Наконец, нам осталось рассмотреть конвейеризацию на самом верхнем уровне, известную как макроконвейер. Поток данных, проходя через один процессор, поступает на вход другому. Компьютер PEPE, имея фактически три независимых системы из 288 устройств, описывается так:

$$t(\text{PEPE}) = (1 \times 3, 288, 32).$$

После расширения трехуровневой модели параллелизма средствами описания потенциальных возможностей конвейеризации каждая тройка

$$t(C) = (k \times k', d \times d', w \times w')$$

интерпретируется так:

- k — число процессоров (каждый со своим УУ), работающих параллельно;
- k' — глубина макроконвейера из отдельных процессоров;
- d — число АЛУ в каждом процессоре, работающих параллельно;
- d' — глубина конвейера из функциональных устройств АЛУ;
- w — число разрядов в слове, обрабатываемых в АЛУ параллельно;
- w' — число ступеней в конвейере функциональных устройств АЛУ.

Очевидна связь между классификацией Фенга и классификацией Хендлера: для получения максимальной степени параллелизма в терминах Фенга надо найти произведение всех шести величин в описании Хендлера. Здесь же за-

метим, что, заложив в основу своей схемы явное указание на присутствующий параллелизм и возможную конвейеризацию, Хендлер сразу снимает некоторые вопросы, характерные для схем Флинна и Фенга, по крайней мере, в плане описания векторно-конвейерных машин.

В дополнение к изложенному способу описания архитектур Хендлер предлагает использовать три операции, которые, будучи примененными к тройкам, позволят описать, во-первых, сложные структуры с подсистемами ввода/вывода, хост-компьютером или какими-то другими особенностями, и, во-вторых, возможные режимы функционирования вычислительных систем, поддерживаемые для оптимального соответствия структуре программ.

Первая операция ($\times$) в каком-то смысле отражает конвейерный принцип обработки и предполагает последовательное прохождение данных сначала через первый ее аргумент-подсистему, а затем через второй. В частности, описание компьютера CDC 6600 можно уточнить следующим образом:

$$t(\text{CDC 6600}) = (10,1,12) \times (1,1 \times 10,60),$$

где первый аргумент отражает существование десяти 12-разрядных периферийных процессоров и тот факт, что любая программа должна сначала быть обработана одним из них и лишь после этого передана центральному процессору для исполнения. Аналогично можно получить описание машины PEPE, принимая во внимание, что в качестве хост-компьютера она использует CDC 7600:

$$t(\text{PEPE}) = t(\text{CDC 7600}) \times (1 \times 3,\ 288,\ 32) =$$

$$= (15,\ 1,\ 12) \times (1,\ 1 \times 9,\ 60) \times (1 \times 3,\ 288,\ 32).$$

Поток данных последовательно проходит через три подсистемы, что мы и отразили, соединив их знаком "$\times$".

Заметим, что все подсистемы последнего примера достаточно сложны и по данному описанию могут представляться по-разному. Чтобы внести бóльшую ясность, аналогично операции конвейерного исполнения, Хендлер вводит операцию параллельного исполнения (+), фиксирующую возможность независимого использования процессоров разными задачами, например:

$$t(4,\ d,\ w) = [(1,\ d,\ w) + (1,\ d,\ w) + (1,\ d,\ w) + (1,\ d,\ w)].$$

В случае CDC 7600 уточненная запись вида:

$$(15,\ 1,\ 12) \times (1[,\ 1 \times 9,\ 60) =$$

$$= [(1,\ 1,\ 12) + \ldots + (1,\ 1,\ 12)]\ \{15\ раз\} \times (1,\ 1 \times 9,\ 60)$$

говорит о том, что каждая задача может захватить свой периферийный процессор, а затем одна за одной они будут поступать в центральный процессор.

И, наконец, третья операция — операция альтернативы ($\vee$), показывает возможные альтернативные режимы функционирования вычислительной сис-

темы. Например, компьютер C.mmp может быть запрограммирован для использования в трех принципиально разных режимах:

$$t(\text{C.mmp}) = (16, 1, 16) \vee (1 \times 16, 1,16) \vee (1, 16, 16).$$

Классификация Л. Шнайдера (L. Snyder). В 1988 году Л. Шнайдер предложил выделить этапы выборки и непосредственно исполнения в потоках команд и данных. Именно разделение потоков на адреса и их содержимое позволило описать такие ранее "неудобные" для классификации архитектуры, как компьютеры с длинным командным словом, систолические массивы и целый ряд других.

Введем необходимые для дальнейшего изложения понятия и обозначения. Назовем *потоком ссылок* S некоторой вычислительной системы конечное множество бесконечных последовательностей пар:

$$S = \{(a_1 <t_1>) (a_2 <t_2>), (b_1 <u_1>) (b_2 <u_2>),$$
$$(c_1 <v_1>) (c_2 <v_2>)\},$$

где первый компонент каждой пары — это неотрицательное целое число, называемое *адресом*, второй компонент — это набор из n неотрицательных целых чисел, называемых *значениями*, причем n одинаково для всех наборов всех последовательностей. Например, пара $(b_2 <u_2>)$ определяет адрес b_2 и значение u_2. Если значения рассматривать как команды, то из потока ссылок получим *поток команд* I; если же значения интерпретировать как данные, то соответствующий поток — это *поток данных* D.

Интерпретация введенных понятий очень проста. Элементы каждой последовательности это адрес и его содержимое, выбираемое из памяти (или записываемое в память). Последовательность пар адрес—значение можно рассматривать как историю выполнения команд либо перемещения данных между процессором и памятью компьютера во время выполнения программы. Число инструкций, которое данный компьютер может выполнять одновременно, определяет число последовательностей в потоке команд. Аналогично, число различных данных, которое компьютер может обработать одновременно, определяет число последовательностей в потоке данных.

Пусть S произвольный поток ссылок. *Последовательность адресов* потока S, обозначаемая S_a, это последовательность наборов, сформированная из адресов каждой последовательности из S:

$$S_a = <a_1\ b_1\ \ c_1>, <a_2\ b_2\ \ c_2>.$$

Последовательность значений потока S, обозначаемая S_v, это последовательность наборов, сформированная из значений каждой последовательности из S:

$$S_v = <t_1\ u_1\ \ v_1>, <t_2\ u_2\ \ v_2>.$$

Если S_x — последовательность элементов, где каждый элемент является набором из n чисел, то для обозначения "ширины" последовательности будем пользоваться обозначением: $w(S_x) = n$.

Из определений S_a, S_v и w следует, что если S это поток ссылок со значениями из n чисел, то $w(S_a) = |S|$ и $w(S_v) = n|S|$, где $|S|$ обозначает мощность множества S.

Каждую пару (I, D) с потоком команд I и потоком данных D будем называть *вычислительным шаблоном*, а все компьютеры будем разбивать на классы в зависимости от того, какой шаблон они могут исполнить. В самом деле, компьютер может исполнить шаблон (I, D), если он в состоянии:

- выдать $w(I_a)$ адресов команд для одновременной выборки из памяти;
- декодировать и проинтерпретировать одновременно $w(I_v)$ команд;
- выдать одновременно $w(D_a)$ адресов операндов;
- выполнить одновременно $w(D_v)$ операций над различными данными.

Если все эти условия выполнены, то компьютер может быть описан как $I_{w(Ia)w(Iv)}D_{w(Da)w(Dv)}$. Рассмотрим классическую последовательную машину. Согласно классификации Флинна, она попадает в класс SISD, следовательно $|I| = |D| = 1$ и $w(I_a) = w(D_a) = 1$. Из-за того, что в подобного рода компьютерах команды декодируются последовательно, следует равенство $w(I_v) = 1$, а последовательное исполнение команд дает $w(D_v) = 1$. Поэтому описание однопроцессорной машины с фоннеймановской архитектурой будет выглядеть так: $I_{1,1}D_{1,1}$.

Теперь возьмем две машины из класса SIMD: Goodyear Aerospace MPP и ILLIAC IV, причем не будем принимать во внимание разницу в способах обработки данных отдельными процессорными элементами. Единственный поток команд означает $|I| = 1$ для обеих машин. Аналогично последовательной машине, для потока команд получаем равенство $w(I_a) = w(I_v) = 1$. Далее, вспомним, что для доступа к операндам устройство управления MPP рассылает один и тот же адрес всем процессорным элементам, поэтому в этой терминологии MPP имеет единственную последовательность в потоке данных, т. е. $|D| = 1$. Однако затем выборка данных из памяти и последующая обработка осуществляются в каждом процессорном элементе, поэтому $w(D_v) = 16\,384$, а вся система MPP может быть описана $I_{1,1}D_{1,16\,384}$.

В ILLIAC IV устройство управления, так же, как и в MPP, рассылает один и тот же адрес всем процессорным элементам, однако каждый из них может получить свой уникальный адрес, добавляя содержимое локального индексного регистра. Это означает, что $|D| = 64$ и в системе присутствуют 64 потока адресов данных, определяющих одиночные потоки операндов, т. е. $w(D_a) = w(D_v) = 64$. Суммируя сказанное, приходим к описанию ILLIAC IV: $I_{1,1}D_{64,64}$.

Для более детальной классификации Шнайдер вводит три предопределенных значения, которые могут принимать величины $w(I_a)$, $w(I_v)$, $w(D_a)$ *и* $w(D_v)$:

- s — значение равно 1;

- c — значение от 1 до некоторой (небольшой) константы;
- m — значение от 1 до произвольно большого конечного числа.

В частности, в этих обозначениях фоннеймановская машина принадлежит к классу $I_{ss}D_{ss}$.

Несмотря на то, что и c и m в принципе не имеют определенной верхней границы, они отражают разные свойства архитектуры компьютера. Описатель c предполагает жесткие ограничения сверху со стороны аппаратуры, и соответствующий параметр не может быть значительно увеличен относительно простыми средствами.

С другой стороны, описатель m используется тогда, когда обозначаемая величина может быть легко изменена, т. е. другими словами, компьютер по данному параметру *масштабируем*. Например, относительная простота увеличения числа процессорных элементов в системе MPP является основанием для того, чтобы отнести ее к классу $I_{ss}D_{sm}$.

Конечно же, различие между c и m в достаточной мере условное и, как правило, порождает массу вопросов. Например, как описать машину, в которой процессоры связаны через общую шину? С одной стороны, нет никаких принципиальных ограничений на число подключаемых процессоров. Однако каждый дополнительный процессор увеличивает загруженность шины, и при достижении некоторого порога подключение новых процессоров бессмысленно. С помощью какого символа описать такую систему: c или m? Мы оставляем данный вопрос открытым.

На основе этих обозначений можно выделить следующие классы компьютеров:

- $I_{ss}D_{ss}$ — классические машины фоннеймановского типа;
- $I_{ss}D_{sc}$ — фоннеймановские машины, в которых заложена возможность выбирать данные, расположенные с разным смещением относительно одного и того же адреса и над которыми будет выполнена одна и та же операция. Примером могут служить компьютеры, имеющие команды типа одновременного выполнения двух операций сложения над данными в формате полуслова, расположенными по указанному адресу;
- $I_{ss}D_{sm}$ — SIMD-компьютеры без возможности получения уникального адреса для данных в каждом процессорном элементе. Сюда входят, например, MPP, Connection Machine 1 и систолические массивы;
- $I_{ss}D_{mm}$ — это SIMD-компьютеры, имеющие возможность независимой модификации адресов операндов в каждом процессорном элементе, например, ILLIAC IV и Connection Machine 2;
- $I_{sc}D_{cc}$ — вычислительные системы, выбирающие и исполняющие одновременно несколько команд, для доступа к которым используется один адрес. Типичным примером являются VLIW-компьютеры;
- $I_{mm}D_{mm}$ — к этому классу относятся все компьютеры типа MIMD.

Достаточно ясно, что не нужно рассматривать все возможные комбинации описателей s, c и m, т. к. архитектура реальных компьютеров накладывает ряд вполне разумных ограничений, в частности $w(I_a) \le w(I_v)$ и $w(D_a) \le w(D_v)$.

Подводя итог, можно отметить два положительных момента в классификации Шнайдера: более избирательная систематизация SIMD-компьютеров и возможность описания нетрадиционных архитектур типа систолических массивов или компьютеров с длинным командным словом. Вместе с тем, почти все вычислительные системы типа MIMD опять попали в один и тот же класс $I_{mm}D_{mm}$. Это означает, что критерий классификации, основанный лишь на потоках команд и данных без учета распределенности памяти и топологии межпроцессорной связи, слишком слаб для подобных систем.

Классификация Д. Скилликорна (D. Skillicorn). В 1989 году была сделана очередная попытка расширить классификацию Флинна и тем самым преодолеть ее недостатки. Д. Скилликорн разработал подход, пригодный для описания свойств многопроцессорных систем и некоторых нетрадиционных архитектур, в частности, dataflow.

Предлагается рассматривать архитектуру любого компьютера, как абстрактную структуру, состоящую из четырех компонентов:

- *процессор команд* (IP — Instruction Processor) — функциональное устройство, работающее как интерпретатор команд; в системе, вообще говоря, может отсутствовать;
- *процессор данных* (DP — Data Processor) — функциональное устройство, работающее как преобразователь данных в соответствии с арифметическими операциями;
- *иерархия памяти* (IM — Instruction Memory, DM — Data Memory) — запоминающее устройство, в котором хранятся данные и команды, пересылаемые между процессорами;
- *переключатель* — абстрактное устройство, обеспечивающее связь между процессорами и памятью.

Функции процессора команд во многом схожи с функциями устройств управления последовательных машин и, согласно Д. Скилликорну, сводятся к следующим. На основе своего состояния и полученной от DP информации IP выполняет такие действия: определяет адрес команды, которая будет выполняться следующей; осуществляет доступ к IM для выборки команды; получает и декодирует выбранную команду; сообщает DP команду, которую надо выполнить; определяет адреса операндов и посылает их в DP; получает от DP информацию о результате выполнения команды.

Функции процессора данных делают его во многом похожим на арифметическое устройство традиционных процессоров. DP получает от IP команду, которую надо выполнить; получает от IP адреса операндов; выбирает опе-

ранды из DM; выполняет команду; запоминает результат в DM; возвращает в IP информацию о состоянии после выполнения команды.

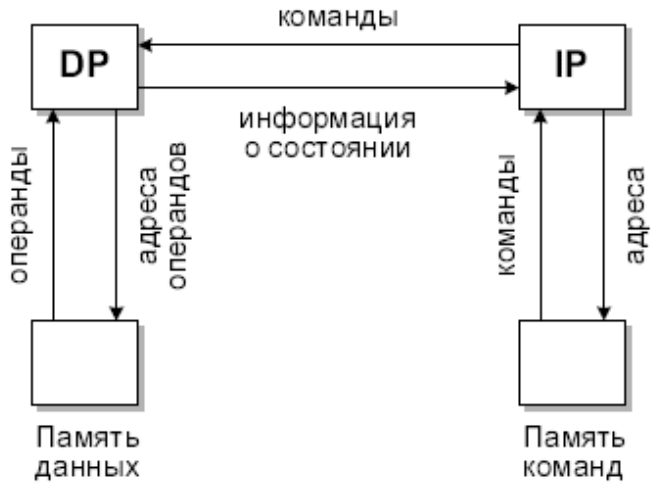


Рис. 3.5. Фоннеймановская архитектура
в терминах классификации Д. Скилликорна

Структура традиционной фон-неймановской архитектуры в терминах таким образом определенных основных частей компьютера показана на рис. 3.5. Это один из самых простых видов архитектуры, не содержащих переключателей. Для описания сложных параллельных вычислительных систем Д. Скилликорн зафиксировал четыре типа переключателей без какой-либо явной связи с типом устройств, которые они соединяют:

- $1 - 1$ — переключатель такого типа связывает пару функциональных устройств;
- $n - n$ — переключатель связывает каждое устройство из одного множества устройств с соответствующим ему устройством из другого множества, т. е. фиксирует попарную связь;
- $1 - n$ — переключатель соединяет одно выделенное устройство со всеми функциональными устройствами из некоторого набора;
- $n \times n$ — каждое функциональное устройство одного множества может быть связано с любым устройством другого множества, и наоборот.

Примеров подобных переключателей можно привести много. Так, все матричные процессоры имеют переключатель типа $1 - n$ для связи единственного процессора команд со всеми процессорами данных. В компьютерах семейства Connection Machine каждый процессор данных имеет свою локальную память, следовательно, такая связь будет описываться как $n - n$. В то же время, каждый процессор команд может связаться с любым другим процессором, что отвечает описанию $n \times n$.

Классификация Д. Скилликорна строится на основе следующих восьми характеристик:

- количество процессоров команд IP;
- число запоминающих устройств (модулей памяти) команд IM;
- тип переключателя между IP и IM;
- количество процессоров данных DP;
- число запоминающих устройств (модулей памяти) данных DM;
- тип переключателя между DP и DM;
- тип переключателя между IP и DP;
- тип переключателя между DP и DP.

Рассмотрим компьютер Connection Machine 2. В терминах данных характеристик его можно описать следующим образом:

$$(1, 1, 1 - 1, n, n, n - n, 1 - n, n \times n).$$

Условное изображение его архитектуры приведено на рис. 3.6.

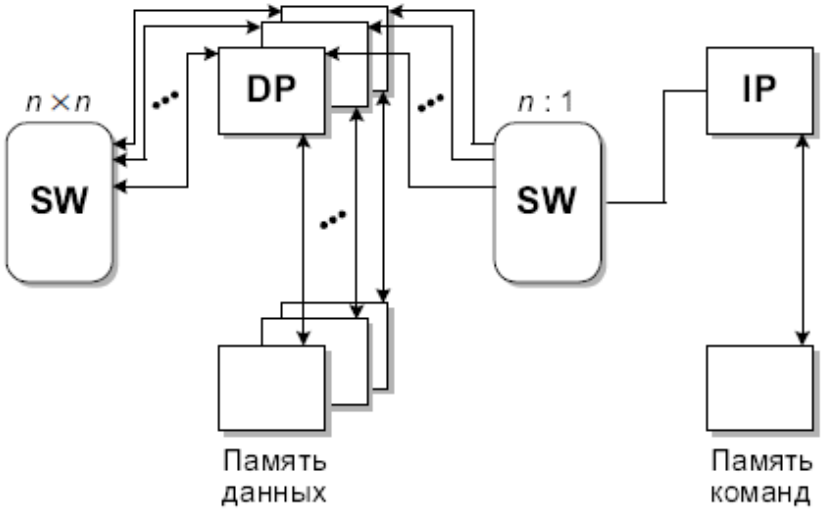


Рис. 3.6. Архитектура Connection Machine 2
в классификации Д. Скилликорна

Для сильно связанных мультипроцессоров типа BBN Butterfly ситуация иная. Такие системы состоят из множества процессоров, соединенных с модулями памяти с помощью переключателя. Задержка при доступе любого процессора к любому модулю памяти примерно одинакова. Связь и синхронизация между процессорами осуществляется через общие (разделяемые) переменные. Описание таких машин в рамках данной классификации выглядит так:

$$(n, n, n - n, n, n, n \times n, n - n, \text{нет}).$$

Саму архитектуру можно изобразить так, как показано на рис. 3.7.

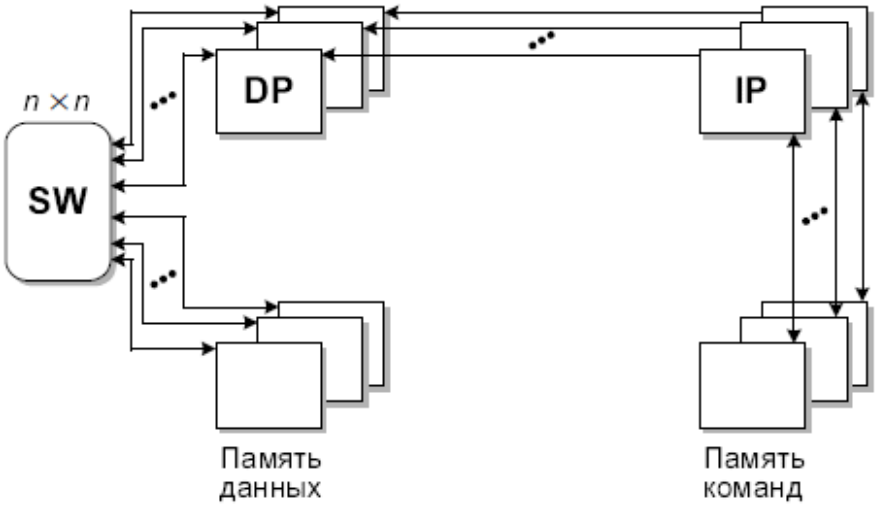


Рис. 3.7. Архитектура BBN Butterfly в классификации Д. Скилликорна

Используя введенные характеристики и предполагая, что рассмотрение количественных характеристик можно ограничить только тремя возможными вариантами значений: 0, 1 и *n* (то есть больше единицы), можно получить различные классы архитектур.

Интересно, что среди сформулированных Скилликорном целей, которым должна служить хорошо построенная классификация архитектур, есть и такая: "классификация должна показывать, за счет каких структурных особенностей достигается увеличение производительности различных вычислительных систем". Эта цель очень созвучна целям наших исследований. Когда пользователь приходит на новую вычислительную систему, то, как правило, ему известна лишь ее пиковая производительность. Однако этого явно недостаточно. Если ему действительно нужна высокая производительность, то, хочет он этого или нет, ему придется иметь дело с целым спектром смежных вопросов. Практически все они связаны с особенностями архитектуры компьютера. Вот тут-то и пригодилась бы хорошая классификация.

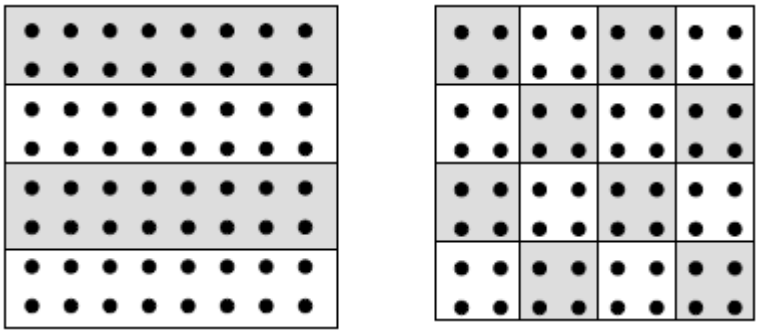
Попадание компьютера в тот или иной класс сразу давало бы пользователю информацию об особенностях его программирования, методах достижения высокой производительности, причинах низкой производительности. Косвенно такую информацию получить можно, но только в самых общих чертах. В частности, все компьютеры в первом приближении можно поделить на компьютеры с общей и распределенной памятью. Для компьютеров с общей памятью пользователю не нужно заботиться о распределении данных, а для программирования можно использовать простую технологию OpenMP. На компьютерах с распределенной памятью ему, скорее всего, придется работать в терминах MPI и самому заботиться о распределении и пересылках данных. В общих чертах все верно, но, честно говоря, только в самых об-

щих. В таком описании опущено слишком много важных деталей, без которых составление эффективных параллельных программ просто невозможно. Раз эти детали существенны, то они должны быть отражены в классификации. Компьютеров много, деталей еще больше, отсюда и множество классификаций, и интерес к данной проблеме в целом.

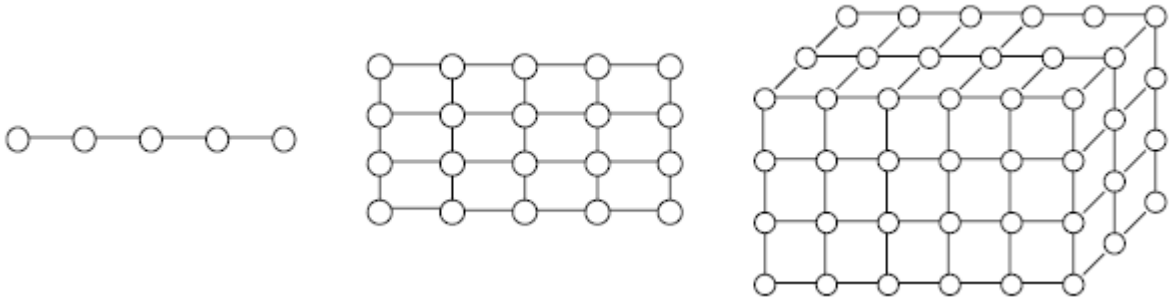
Большинство задач в программировании имеет несколько параллельных решений. Наилучшее решение может отличаться от предложенного используемыми последовательными алгоритмами. Существует методология проектирования, предназначенная для разработки параллельных приложений. Эта методология разбивает процесс проектирования на четыре отдельных стадии: декомпозиция, коммуникация, агломерация и отображение.

1. Декомпозиция. Вычисления, которые могут быть выполнены, и данные, которыми оперируют эти вычисления, разделяются на меньшие (локальные) задачи. Внимание сосредотачивается на оценки возможностей для параллельного исполнения.

Выбор способа разделения вычислений на независимые части основывается на анализе вычислительной схемы решения исходной задачи. Требования, которым должен удовлетворять выбираемый подход, обычно состоят в обеспечении равного объема вычислений в выделяемых подзадачах и минимума информационных зависимостей между этими подзадачами (при прочих равных условиях нужно отдавать предпочтение редким операциям передачи большего размера сообщений по сравнению с частыми пересылками данных небольшого объема). В общем случае, проведение анализа и выделение задач представляет собой достаточно сложную проблему – ситуацию помогает разрешить существование двух часто встречающихся типов вычислительных схем:



Для большого класса задач вычисления сводятся к выполнению однотипной обработки элемент элементов большого набора данных – к такому виду задач относятся, например, матричные вычисления, численные методы решения уравнений в частных производных и др. В этом случае говорят, что существует параллелизм по данным, и выделение подзадач сводится к разделению имеющихся данных. Так, например, для нашей учебной задачи поиска максимального значения при формировании подзадач исходная матрица A может быть разделена на отдельные строки (или последовательные группы строк) – ленточная схема разделения данных (см. рис. 6.3) или на прямоугольные наборы элементов – блочная схема разделения данных. Для большого количества решаемых задач разделение вычислений по данным приводит к порождению одно-, двух- и трехмерных наборов подзадач, для которых информационные связи существуют только между ближайшими соседями (такие схемы обычно именуются сетками или решетками),



Для другой части задач вычисления могут состоять в выполнении разных операций над одним и тем же набором данных – в этом случае говорят о существовании функционального параллелизма (в качестве примеров можно привести задачи обработки последовательности запросов к информационным базам данных, вычисления с одновременным применением разных алгоритмов расчета и т.п.). Очень часто функциональная декомпозиция может быть использована для организации конвейерной обработки данных (так, например, при выполнении каких-либо преобразований данных вычисления могут быть сведены к функциональной последовательности ввода, обработки и сохранения данных).

Важный вопрос при выделении подзадач состоит в выборе нужного уровня декомпозиции вычислений. Формирование максимально возможного количества подзадач обеспечивает использование предельно достижимого уровня параллелизма решаемой задачи, однако затрудняет анализ параллельных вычислений. Использование при декомпозиции вычислений только достаточно "крупных" подзадач приводит к ясной схеме параллельных вычислений, однако может затруднить эффективное использование достаточно большого количества процессоров. Возможное разумное сочетание этих двух подходов может состоять в использовании в качестве конструктивных элементов декомпозиции только тех подзадач, для которых методы параллельных вычислений являются известными. Так, например, при анализе задачи матричного умножения в качестве подзадач можно использовать методы скалярного произведения векторов или алгоритмы матрично-векторного произведения. Подобный промежуточный способ декомпозиции вычислений позволит обеспечить и простоту представления вычислительных схем, и эффективность параллельных расчетов. Выбираемые подзадачи при таком подходе будем именовать далее базовыми, которые могут быть элементарными (неделимыми), если не допускают дальнейшего разделения, или составными в противном случае.

Для рассматриваемой учебной задачи достаточный уровень декомпозиции может состоять, например, в разделении матрицы A на множество отдельных строк и получении на этой основе набора подзадач поиска максимальных значений в отдельных строках; порождаемая при этом структура информационных связей соответствует линейному графу – см. рис. 6.5.

Для оценки корректности этапа разделения вычислений на независимые части можно воспользоваться контрольным списком вопросов, предложенных в Foster (1995):

• Выполненная декомпозиция не увеличивает объем вычислений и необходимый объем памяти?

• Возможна ли при выбранном способе декомпозиции равномерная загрузка всех имеющихся процессоров?

• Достаточно ли выделенных частей процесса вычислений для эффективной загрузки имеющихся процессоров (с учетом возможности увеличения их количества)?

2. Коммуникация. Определяется координирование выполнения задачи и подходящие коммуникационные структуры и алгоритмы.

При наличии вычислительной схемы решения задачи после выделения базовых подзадач определение информационных зависимостей между подзадачами обычно не вызывает больших затруднений. При этом, однако, следует отметить, что на самом деле этапы выделения подзадач и информационных зависимостей достаточно сложно поддаются разделению. Выделение подзадач должно происходить с учетом возникающих информационных связей; после анализа объема и частоты необходимых информационных обменов между подзадачами может потребоваться повторение этапа разделения вычислений.

При проведении анализа информационных зависимостей между подзадачами следует различать (предпочтительные формы информационного взаимодействия выделены подчеркиванием):

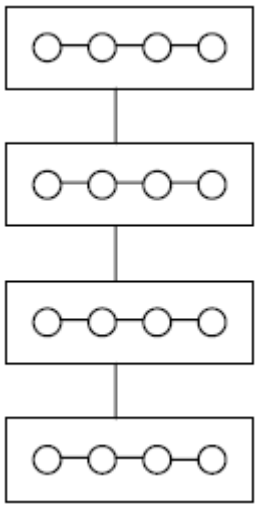
• Локальные и глобальные схемы передачи данных – для локальных схем передачи данных в каждый момент времени выполняются только между небольшим числом подзадач (располагаемых, как правило, на соседних процессорах), для глобальных операций передачи данных в процессе коммуникации принимают участие все подзадачи,

• Структурные и произвольные способы взаимодействия – для структурных способов организация взаимодействий приводит к формированию некоторых стандартных схем коммуникации (например, в виде кольца, прямоугольной решетки и т.д.), для произвольных структур взаимодействия схема выполняемых операций передач данных не носит характер однородности,

• Статические или динамические схемы передачи данных – для статических схем моменты и участники информационного взаимодействия фиксируются на этапах проектирования и разработки параллельных программ, для динамического варианта взаимодействия структура операции передачи данных определяется в ходе выполняемых вычислений,

• Синхронные и асинхронные способы взаимодействия – для синхронных способов операции передачи данных выполняются только при готовности всех участников взаимодействия и завершаются только после полного окончания всех коммуникационных действий, при асинхронном выполнении операций участники взаимодействия могут не дожидаться полного завершения действий по передаче данных. Для представленных способов взаимодействия достаточно сложно выделить предпочтительные формы организации передачи данных: синхронный вариант, как правило, более прост для использования, в то время как асинхронный способ часто позволяет существенно снизить временные задержки, вызванные операциями информационного взаимодействия.

Как уже отмечалось в предыдущем пункте, для учебной задачи поиска максимального значения при использовании в качестве базовых элементов подзадач поиска максимальных значений в отдельных строках исходной матрицы A структура информационных связей имеет вид, представленный на рис. 6.5.



Как и ранее, для оценки правильности этапа выделения информационных зависимостей можно воспользоваться контрольным списком вопросов, предложенных в Foster (1995):
• Соответствует ли вычислительная сложность подзадач интенсивности их информационных взаимодействий?
• Является ли одинаковой интенсивность информационных взаимодействий для разных подзадач?
• Является ли схема информационного взаимодействия локальной?
• Не препятствует ли выявленная информационная зависимость параллельному решению подзадач?
3. Агломерация. Структуры задач и коммуникаций, определенные на первых двух стадиях проектирования, оцениваются в плане стоимости реализации и требований рабочим характеристикам. При необходимости локальные задачи объединяются в большие, чтобы улучшить рабочие характеристики или уменьшить стоимость разработки.
Масштабирование разработанной вычислительной схемы параллельных вычислений проводится в случае, если количество имеющихся подзадач отличается от числа планируемых к использованию процессоров. Для сокращения количества подзадач необходимо выполнить укрупнение (агрегацию) вычислений. Применяемые здесь правила совпадают с рекомендациями начального этапа выделения подзадач – определяемые подзадачи, как и ранее, должны иметь одинаковую вычислительную сложность, а объем и интенсивность информационных взаимодействий между подзадачами должны оставаться на минимально-возможном уровне. Как результат, первыми претендентами на объединение являются подзадачи с высокой степенью информационной взаимозависимости.
При недостаточном количестве имеющегося набора подзадач для загрузки всех доступных к использованию процессоров необходимо выполнить детализацию (декомпозицию) вычислений. Как правило, проведение подобной декомпозиции не вызывает каких-либо затруднений, если для базовых задач методы параллельных вычислений являются известными.
Выполнение этапа масштабирования вычислений должно свестись, в конечном итоге, к разработке правил агрегации и декомпозиции подзадач, которые должны параметрически зависеть от числа процессоров, применяемых для вычислений.
Для рассматриваемой учебной задачи поиска максимального значения агрегация вычислений может состоять в объединении отдельных строк в группы (ленточная схема

разделения матрицы – см. рис. 6.3а), при декомпозиции подзадач строки исходной матрицы A могут разбиваться на несколько частей (блоков).
Список контрольных вопросов, предложенный в Foster (1995) для оценки правильности этапа масштабирования, выглядит следующим образом:
• Не ухудшится ли локальность вычислений после масштабирования имеющегося набора подзадач?
• Имеют ли подзадачи после масштабирования одинаковую вычислительную и коммуникационную сложность?
• Соответствует ли количество задач числу имеющихся процессоров?
• Зависят ли параметрически правила масштабирования от количества процессоров?
4. Отображение. Каждая задача назначается процессору таким способом, чтобы удовлетворить конкурентным требованиям: максимизации загрузки процессоров и минимизации стоимости коммуникаций. Отображение может быть определено статически или во время выполнения при помощи алгоритмов балансирования нагрузки.
Распределение подзадач между процессорами является завершающим этапом разработки параллельного метода. Надо отметить, что управление распределением нагрузки для процессоров возможно только для вычислительных систем с распределенной памятью, для мультипроцессоров (систем с общей памятью) распределение нагрузки обычно выполняется операционной системой автоматически. Кроме того, данный этап распределения подзадач между процессорами является избыточным, если количество подзадач совпадает с числом имеющихся процессоров, а топология сети передачи данных вычислительной системы представляет собой полный граф (т.е., все процессоры связаны между собой прямыми линиями связи).
Основной показатель успешности выполнения данного этапа – эффективность использования процессоров, определяемая как относительная доля времени, в течение которого процессоры использовались для вычислений, связанных с решением исходной задачи. Пути достижения хороших результатов в этом направлении остаются прежними – как и ранее, необходимо обеспечить равномерное распределение вычислительной нагрузки между процессорами и минимизировать количество сообщений, передаваемых между процессорами. Точно так же, как и на предшествующих этапах проектирования, оптимальное решение проблемы распределения подзадач между процессорами основывается на анализе информационной связности графа "подзадачи - сообщения". Так, в частности, подзадачи, между которыми имеются информационные взаимодействия, целесообразно размещать на процессорах, между которыми существуют прямые линии передачи данных.
Следует отметить, что требование минимизации информационных обменов между процессорами может противоречить условию равномерной загрузки процессов. Так, мы можем разместить все подзадачи на одном процессоре и полностью устранить межпроцессорную передачу сообщений, однако, понятно, загрузка большинства процессоров в этом случае будет минимальной.
Для нашей учебной задачи поиска максимального значения распределение подзадач между процессорами не вызывает каких-либо затруднений – достаточно лишь обеспечить размещение подзадач, между которыми имеются информационные связи, на процессорах, для которых существуют прямые каналы передачи данных. Поскольку структура информационной связей учебной задачи имеет вид линейного графа,

выполнение данного требования может быть обеспечено практически при любой топологии сети вычислительной системы.

Решение вопросов балансировки вычислительной нагрузки значительно усложняется, если схема вычислений может изменяться в ходе решения задачи. Причиной этого могут быть, например, неоднородные сетки при решении уравнений в частных производных, разреженность матриц и т.п.3). Кроме того, используемые на этапах проектирования оценки вычислительной сложности решения подзадач могут иметь приближенный характер и, наконец, количество подзадач может изменяться в ходе вычислений. В таких ситуациях может потребоваться перераспределение базовых подзадач между процессорами уже непосредственно в процессе выполнения параллельной программы (или, как обычно говорят, придется выполнить динамическую балансировку вычислительной нагрузки). Данные вопросы являются одними из наиболее сложных (и наиболее интересных) в области параллельных вычислений – к сожалению, рассмотрение данных вопросов выходит за рамки данного учебного материала (дополнительная информация может быть получена, например, в Buyya (1999) и Wilkinson and Allen (1999)).

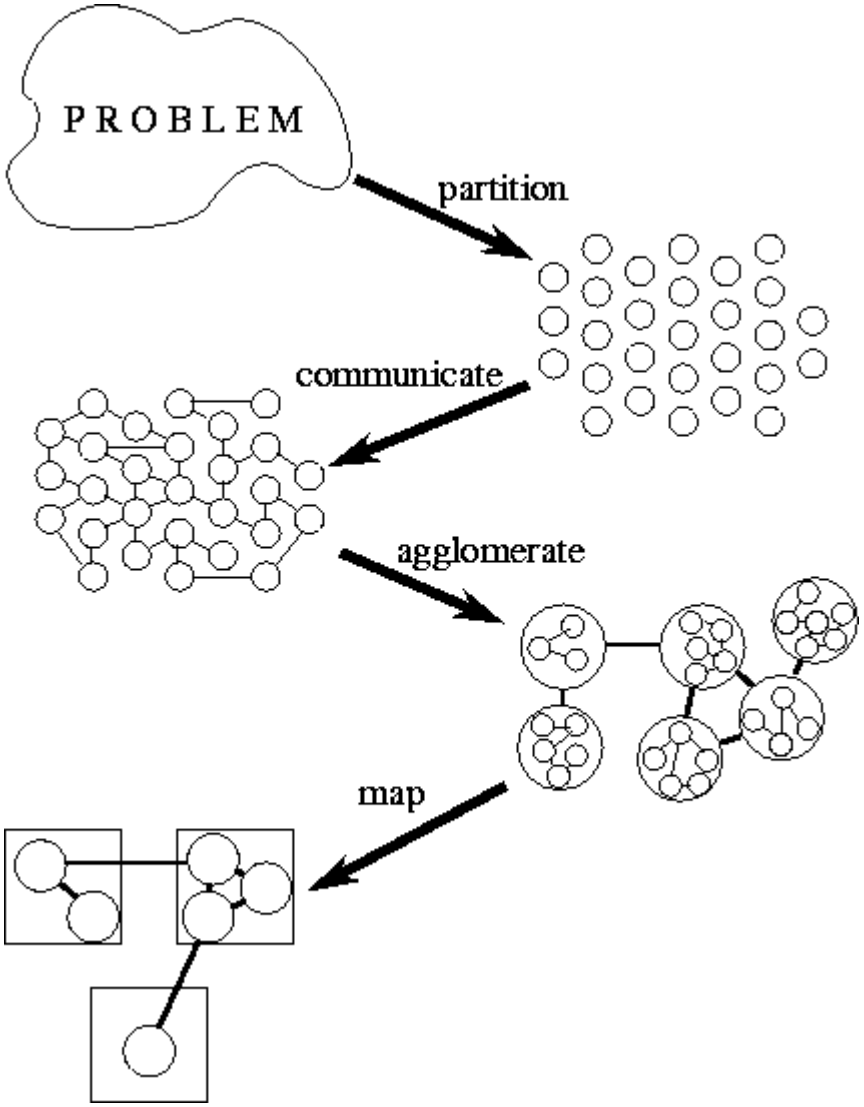
В качестве примера дадим краткую характеристику широко используемого способа динамического управления распределением вычислительной нагрузки, обычно именуемого схемой "менеджер - исполнитель" (manager-worker scheme). При использовании данного подхода предполагается, что подзадачи могут возникать и завершаться в ходе вычислений, при этом информационные взаимодействия между подзадачами либо полностью отсутствует, либо минимальны. В соответствии с рассматриваемой схемой для управления распределением нагрузки в системе выделяется отдельный процессор-менеджер, которому доступна информация обо всех имеющихся подзадачах. Остальные процессоры системы являются исполнителями, которые для получения вычислительной нагрузки обращаются к процессору-менеджеру. Порождаемые в ходе вычислений новые подзадачи передаются обратно процессору-менеджеру и могут быть получены для решения при последующих обращениях процессоров-исполнителей. Завершение вычислений происходит в момент, когда процессоры-исполнители завершили решение всех переданных им подзадач, а процессор-менеджер не имеет каких-либо вычислительных работ для выполнения.

Предложенный в Foster (1995) перечень контрольных вопросов для проверки этапа распределения подзадач состоит в следующем:

• Не приводит ли распределение нескольких задач на один процессор к росту дополнительных вычислительных затрат?

• Существует ли необходимость динамической балансировки вычислений?

• Не является ли процессор-менеджер "узким" местом при использовании схемы "менеджер-исполнитель"?


PROBLEM
partition
communicate
agglomerate
map