Компьютеры постоянно совершенствуются. Главный вектор их развития — повышение производительности, т. е. возможности выполнять большее число операций в единицу времени. Данный вектор очень важен. Об этом говорит хотя бы тот факт, что в последние годы производительность компьютеров увеличивается на порядок примерно каждые пять лет. За производительность в компьютере отвечает, в первую очередь, арифметико-логическое устройство. Но оно занимает лишь около 20% всего оборудования. Следовательно, решая главную задачу — обеспечение пользователя информационно-вычислительными услугами, — компьютер решает и какие-то вспомогательные задачи, видимые или невидимые для пользователя. Меняется состав этих маленьких задач, меняются методы их решения. Вообще говоря, все они развиваются синхронно. Но иногда некоторые из маленьких задач или даже одна из них становится настолько критичной, что на определенном этапе развития компьютера оказывается самой главной, затмевая собой даже компьютер в целом.

Чтобы лучше представлять, как и почему в процессе развития компьютера возникают различные узкие места, рассмотрим следующую *иллюстративную модель.*

Будем считать, что компьютер и его программное окружение есть некоторое *предприятие широкого профиля* по оказанию информационно-вычислительных услуг. Услугой является обработка какой-то совокупности данных. Потребители этих услуг — пользователи. Пользователь представляет предприятию согласованное с его требованиями *техническое задание* на услугу. Это — программа, написанная на том или ином языке. Отдавая заказ на конкретную услугу, пользователь, естественно, не хотел бы ничего знать о том, как предприятие будет ее реализовывать. Требования пользователя просты. Услуга должна быть выполнена правильно, быстро и дешево. На мелких услугах обычно так и бывает. Но если услуга масштабная, могут происходить сбои. Чаще всего предприятие не может обеспечить выполнение услуг в срок. Конечно, пользователь может обратиться в другое предприятие. Иногда это помогает. Однако нередко возникают и другие ситуации. Например, выполнение заказа в срок может потребовать от пользователя очень больших дополнительных затрат. Или может оказаться, что вообще ни одно из предприятий не берется за выполнение заказа. И тогда пользователю самому приходится разбираться, почему это происходит и что надо делать.

Любое предприятие по обеспечению информационно-вычислительными услугами организовано довольно сложно. Но всегда его ядром является *производственный сектор.* Это — арифметико-логическое устройство. В его составе имеется некоторый набор оборудования, такого как сумматоры, умножители, сопроцессоры и т. п. Все оно выполняет какие-то простейшие операции, вообще говоря, не имеющие прямого отношения к конечной услуге. Текущей организацией работы производственного отдела занимается *сектор оперативного управления.* Это — устройство управления. На предприятии существует *склад* для хранения исходных материалов, а также проме-

жуточной и конечной продукции. Это — память. Структура склада разветвленная. Ее сложность объясняется необходимостью хранить самую разную по характеру использования продукцию. При работе производственного сектора появляются промежуточные результаты, которые будут востребованы им же или сразу, или почти сразу. Нет никакого смысла отсылать их в дальний угол склада, что неизбежно привело бы к общему замедлению процесса. Поэтому отдельные ячейки склада располагаются среди оборудования производственного сектора. Это — регистровая память. Другие рядом с ним. Это — кэш-память. Несколько дальше располагается быстрая оперативная память. И дальше всего — медленная память.

Одним из важнейших является *технологический сектор*. Именно он на основе технического задания разрабатывает план использования оборудования производственного сектора, склада и некоторых других подразделений предприятия для реализации каждой конкретной услуги. Это — компилятор. Осуществление данного плана неизбежно требует перемещения различных объектов из одного места в другое. Обеспечением всех перемещений занимается *транспортный сектор*. Это — различные соединения, шины, каналы и т. п. И наконец, организует реальное выполнение работ в соответствии с разработанным планом *организационный сектор*. Это — операционная система.

Такова грубая модель устройства и процесса функционирования компьютера. В реальности все сложнее: отдельных подразделений в предприятии гораздо больше, функции между ними разделены не так четко и т. д. Тем не менее, даже на этой иллюстративной модели можно увидеть довольно много.

Вообще говоря, любой универсальный компьютер способен решить любую задачу, если только у него имеется достаточно памяти. Другое дело, что она может решаться настолько долго, что не хватит времени дождаться результатов. В рассмотренной модели видно, что время решения задачи определяется многими факторами. Главный из них — мощность оборудования производственного сектора. Но и от других секторов зависит немало. Если оборудование транспортного сектора несовершенно, перемещения объектов из одного места в другое будут осуществляться медленно. В этих условиях либо очень трудно, либо даже невозможно добиться высокой производительности всего предприятия. Если технологический сектор составит неэффективный план, требующий, например, больших накладных расходов, — эффект будет похожим. Очень многое зависит от того, как организационный сектор будет реализовывать план, предложенный технологическим сектором. Если из дальних углов склада необходимо часто перемещать большие количества объектов, то нерационально перемещать их малыми порциями. Гораздо эффективнее осуществлять перемещение большими группами. Как будет это организовано, зависит от решений, принимаемых организационным сектором. А эти решения, в частности, определяются структурой склада. И т. п.

Таким образом, работа нашего модельного предприятия по обеспечению пользователей информационно-вычислительными услугами зависит от действий всех секторов. Его базовая производительность определяется мощно-

стью оборудования производственного сектора. Суммарная производительность всего оборудования этого сектора называется *пиковой производительностью*. Это — теоретическое понятие. Оно не учитывает никакие временны́е затраты от деятельности других секторов. *Реальная производительность* — это производительность предприятия, которую оно реально достигает при выполнении конкретной услуги. Вообще говоря, она может зависеть от вида услуги. Реальная производительность никогда не может превышать пиковую. Степень ее близости к пиковой говорит о степени согласованности работы секторов между собой и об эффективности работы предприятия в целом. Отношение реальной производительности к пиковой называется *эффективностью* работы предприятия. Свою долю в уменьшение эффективности вносят все сектора. Одни больше, другие меньше, какие-то очень много.

Каждый из секторов решает свою маленькую задачу. И только вместе они способны сделать нечто большое. В процессе развития предприятия наступает момент, когда эволюционным путем уже нельзя поднять производительность. Тогда приходится проводить глобальную модернизацию. В первую очередь необходимо решить, как радикально увеличить пиковую производительность. Есть два пути, отработанные веками в самых разных отраслях. Грубо говоря, один из них связан с наращиванием однотипного оборудования, другой — с организацией конвейеров. В нашем случае годятся оба и даже их комбинация. Об этом мы также будем подробно говорить в *главе 2*. На период реконструкции производственного сектора его задача становится главной. Но затем неизбежно приходится модернизировать и все другие сектора. Главными становятся задачи реконструкции склада, транспортных путей, управления и т. д. Любая малая задача становится главной тогда, когда она оказывается самым узким местом в достижении наивысшей реальной производительности предприятия в целом.

Все сказанное, в том числе касающееся эффективности, имеет прямое отношение к любому компьютеру. Если на большинстве программ эффективность работы компьютера находится в пределах 0,5—1, ситуацию можно считать хорошей. Если же эффективность намного меньше, а задачи требуется решать как можно быстрее, приходится разбираться, где же теряется производительность и что надо сделать для ее повышения. Другими словами, надо установить узкие места процесса функционирования компьютера и его программного окружения в целом. Согласно рассмотренной модели, в первую очередь они могут определяться:

- составом, принципом работы и временны́ми характеристиками арифметико-логического устройства;
- составом, размером и временны́ми характеристиками памяти;
- структурой и пропускной способностью коммуникационной среды;
- компилятором, создающим неэффективные коды;

- операционной системой, организующей неэффективную работу с памятью, особенно медленной.

Как мы увидим в дальнейшем, постоянное совершенствование именно перечисленных составляющих сделало возможным превращение "обыкновенного" компьютера в суперкомпьютер. Хотя и в суперкомпьютерах эти составляющие остаются потенциальными источниками возникновения узких мест.

Любая вычислительная система есть работающая во времени совокупность функциональных устройств (ФУ). Для оценки качества ее работы вводятся различные характеристики. Рассмотрим их в рамках некоторой модели процесса работы устройств, вполне достаточной для практического применения. Мы не будем интересоваться *содержательной* частью операций, выполняемых функциональными устройствами. Они могут означать арифметические или логические функции, ввод/вывод данных, пересылку данных в память и извлечение данных из нее или что-либо иное. Более того, допускается, что при разных обращениях операции могут означать *разные* функции. Для нас сейчас важны лишь времена выполнения операций и то, на устройствах, какого типа они реализуются [9].

Пусть введена система отсчета времени и установлена его единица, например, секунда. Будем считать, что длительность выполнения операций измеряется в долях единицы. Все устройства, реальные или гипотетические, основные или вспомогательные, могут иметь любые времена срабатывания. Единственное существенное ограничение состоит в том, что все срабатывания одного и того же ФУ должны быть *одинаковыми* по длительности. Всегда мы будем интересоваться работой каких-то конкретных наборов ФУ. По умолчанию будем предполагать, что все другие ФУ, необходимые для обеспечения процесса функционирования этих наборов, срабатывают мгновенно. Поэтому, если не сделаны специальные оговорки, мы не будем в таких случаях принимать во внимание факт их реального существования. Времена срабатываний изучаемых ФУ будем считать *ненулевыми*.

Назовем функциональное устройство *простым*, если никакая последующая операция не может начать выполняться раньше, чем закончится предыдущая. Простое ФУ может выполнять операции одного типа или разные операции. Разные ФУ могут выполнять операции, в том числе одинаковые, за разное время. Примером простого ФУ могут служить не конвейерные сумматоры или умножители. Эти ФУ реализуют только один тип операции. Простым устройством можно считать многофункциональный процессор, если он не способен выполнять различные операции одновременно, и мы не принимаем во внимание различия во временах реализации операций, предполагая, что они одинаковы. Основная черта простого ФУ только одна: оно монопольно использует свое оборудование для выполнения каждой отдельной операции.

В отличие от простого ФУ, *конвейерное* ФУ распределяет свое оборудование для одновременной реализации нескольких операций. Очень часто оно конструируется как линейная цепочка простых элементарных ФУ, имеющих одинаковые времена срабатывания. На этих элементарных ФУ последовательно реализуются отдельные этапы операций. В случае конвейерного ФУ, выполняющего операцию сложения чисел с плавающей запятой, соответствующие элементарные устройства последовательно реализуют такие операции, как сравнение порядков, сдвиг мантиссы, сложение мантисс и т. п. Тем не менее, ничто не мешает, например, считать конвейерным ФУ линейную цепочку универсальных процессоров. Принцип функционирования конвейерного ФУ остается одним и тем же. Сначала на первом элементарном ФУ выполняется первый этап первой операции, и результат передается второму элементарному ФУ. Затем на втором элементарном ФУ реализуется второй этап первой операции. Одновременно на освободившемся первом ФУ реализуется первый этап второй операции. После этого результат срабатывания второго ФУ передается третьему ФУ, результат срабатывания первого ФУ передается второму ФУ. Освободившееся первое ФУ готово для выполнения первого этапа третьей операции и т. д. После прохождения всех элементарных ФУ в конвейере операция оказывается выполненной. Элементарные ФУ называются *ступенями* конвейера, число ступеней в конвейере — *длиной* конвейера. Простое ФУ всегда можно считать конвейерным с длиной конвейера, равной 1. Как уже говорилось, конвейерное ФУ часто является линейной цепочкой простых ФУ, но возможны и более сложные конвейерные конструкции.

Поскольку уже конвейерное ФУ само является системой связанных устройств, необходимо установить наиболее общие принципы работы систем ФУ. Будем считать, что любое ФУ не может *одновременно* выполнять операцию и сохранять результат предыдущего срабатывания, т. е. оно *не имеет никакой собственной памяти.* Однако будем допускать, что результат предыдущего срабатывания может сохраняться в ФУ до момента начала очередного его срабатывания, *включая сам этот момент.* После начала очередного срабатывания ФУ результат предыдущего срабатывания *пропадает.* Предположим, что все ФУ работают по индивидуальным командам. В момент подачи команды на входы конкретного ФУ передаются аргументы выполняемой операции либо как результаты срабатывания других ФУ с их выходов, либо как входные данные, либо как-нибудь иначе. Сейчас нам безразлично, как именно осуществляется их подача. Важно то, что в момент начала очередного срабатывания ФУ входные данные для этого ФУ доступны, а сам процесс подачи не приводит к задержке общего процесса. Конечно мы предполагаем, что программы, определяющие моменты начала срабатываний всех ФУ, составлены корректно и не приводят к тупиковым ситуациям.

При определении различных характеристик, связанных с работой ФУ, так или иначе приходится находить число операций, выполняемых за какое-то время. Это число должно быть целым. Если отрезок времени равен T, а длительность операции есть τ, то за время T можно выполнить либо T/τ, либо $\lceil T/\tau \rceil - 1$ операций, где символ $\lceil * \rceil$ есть ближайшее к $*$ сверху целое число. При больших по сравнению с τ значениях T величина $\lceil T/\tau \rceil - 1$ приблизительно равна T/τ. Чтобы в дальнейшем не загромождать выкладки и формулы символами целочисленности и различными членами малого порядка, мы будем всюду приводить результаты в главном, что эквивалентно переходу к пределу при $T \to \infty$. Говоря иначе, все характеристики и соотношения будут носить *асимптотический* характер, хотя об этом не будет напоминаться специально. Данное обстоятельство нисколько не снижает практическую ценность получаемых результатов, но делает их более наглядными.

Назовем *стоимостью операции* время ее реализации, а *стоимостью работы* — сумму стоимостей всех выполненных операций. Стоимость работы — это время последовательной реализации всех рассматриваемых операций на простых ФУ с аналогичными временами срабатываний. *Загруженностью устройства* на данном отрезке времени будем называть отношение стоимости реально выполненной работы к максимально возможной стоимости. Ясно, что загруженность p всегда удовлетворяет условиям $0 \le p \le 1$. Имеет место также очевидное

Утверждение 2.1

Максимальная стоимость работы, которую можно выполнить за время T, равна T для простого ФУ и nT для конвейерного ФУ длины n.

Будем называть *реальной производительностью* системы устройств количество операций, реально выполненных в среднем за единицу времени. *Пиковой производительностью* будем называть максимальное количество операций, которое может быть выполнено той же системой за единицу времени при отсутствии связей между ФУ. Из определений вытекает, что как реальная, так и пиковая производительности системы суть суммы соответственно реальных и пиковых производительностей всех составляющих систему устройств.

Утверждение 2.2

Пусть система состоит из s устройств, в общем случае простых или конвейерных. Если устройства имеют пиковые производительности $\pi_1, \ldots, \pi_s$ и работают с загруженностями $p_1, \ldots, p_s$, то реальная производительность r системы выражается формулой

$$r = \sum_{i=1}^{s} p_i \pi_i. \qquad (2.1)$$

Поскольку реальная производительность системы равна сумме реальных производительностей всех ФУ, то достаточно доказать утверждение для одного устройства. Пусть для выполнения одной операции ФУ требует время τ и за время T выполнено N операций. Независимо от того, каков тип устройства, стоимость выполненной работы равна $N\tau$. Если устройство простое, то согласно утверждению 2.1 максимальная стоимость работы равна T. Поэтому загруженность устройства равна $N\tau/T$. По определению реальная производительность ФУ есть N/T, а его пиковая производительность — $1/\tau$. И равенство (2.1) становится очевидным. Предположим теперь, что устройство конвейерное длины n. Согласно тому же утверждению 2.1 максимальная стоимость работы в данном случае равна nT. Поэтому загруженность устройства равна $N\tau/nT$. Реальная производительность снова есть N/T, а пиковая производительность будет равна n/τ. И опять равенство (2.1) становится очевидным.

Если r, π, p суть соответственно реальная производительность, пиковая производительность и загруженность одного устройства, то имеет место равенство $r = p\pi$. Отсюда видно, что для достижения наибольшей реальной производительности устройства нужно обеспечить наибольшую его загруженность. Для практических целей понятие производительности наиболее важно потому, что именно оно показывает, насколько эффективно устройство выполняет полезную работу. По отношению к производительности понятие загруженности является вспомогательным. Тем не менее, оно полезно в силу того, что указывает путь повышения производительности, причем через вполне определенные действия.

Хотелось бы и для системы устройств ввести понятие загруженности, играющее аналогичную роль. Его можно определять по-разному. Например, как и для одного ФУ, можно было бы считать, что загруженность системы ФУ есть отношение стоимости реально выполненной работы к максимально возможной стоимости. Такое определение вполне приемлемо и позволяет сделать ряд полезных выводов. Однако имеются и возражения. В этом определении медленные и быстрые устройства оказываются равноправными и если необходимо повысить загруженность системы, то не сразу видно, за счет какого ФУ это лучше сделать. К тому же, в данном случае не всегда будет иметь место равенство $r = p\pi$ с соответствующими характеристиками системы.

Правильный путь введения понятия загруженности системы подсказывает соотношение (2.1). Пусть система состоит из s устройств, в общем случае простых или конвейерных. Если устройства имеют пиковые производительности π_1, ..., π_s и работают с загруженностями p_1, ..., p_s, то будем считать по определению, что *загруженность системы* есть величина

$$p = \sum_{i=1}^{s} \alpha_i p_i, \quad \alpha_i = \frac{\pi_i}{\sum_{j=1}^{s} \pi_j}. \tag{2.2}$$

Загруженность системы есть взвешенная сумма загруженностей отдельных устройств, т. к. из (2.2) следует, что

$$\sum_{i=1}^{s} \alpha_i = 1, \quad \alpha \geq 0, \quad 1 \leq i \leq s. \tag{2.3}$$

Поэтому, как и должно быть для загруженности, выполняются неравенства $0 \leq p \leq 1$. Из определения (2.2) с учетом (2.3) получаем, что для того, чтобы загруженность системы устройств равнялась 1, необходимо и достаточно, чтобы равнялись 1 загруженности каждого из устройств. Это вполне логично. Если система состоит из одного устройства, т. е. $s = 1$, то из (2.2) вытекает, что на ней понятия загруженности системы и загруженности устройства совпадают. Данный факт говорит о согласованности только что введенного понятия загруженности системы с ранее введенными понятиями. По определению, пиковая производительность π системы устройств равна $\pi_1 + \ldots + \pi_s$. Следовательно, согласно (2.1), (2.2) всегда выполняется очень важное равенство

$$r = p\pi. \tag{2.4}$$

Большое число ФУ, так же как и конвейерные ФУ, используются тогда, когда возникает потребность решить задачу быстрее. Чтобы понять, насколько быстрее это удается сделать, нужно ввести понятие "ускорение". Как и в случае загруженности, оно может вводиться различными способами, многообразие которых зависит от того, что с чем и как сравнивается. Нередко ускорение определяется, например, как отношение времени решения задачи на одном универсальном процессоре к времени решения той же задачи на системе из s таких же процессоров. Очевидно, что в наилучшей ситуации ускорение может достигать s. Отношение ускорения к s называется *эффективностью*. Заметим, что подобное определение ускорения применимо только к системам, состоящим из одинаковых устройств, и не распространяется на смешанные системы. Понятие же "эффективность" в рассматриваемом случае просто совпадает с понятием загруженности. При введении понятия "ускорение" мы поступим иначе.

Пусть алгоритм реализуется за время T на вычислительной системе из s устройств, в общем случае простых или конвейерных и имеющих пиковые производительности $\pi_1, \ldots, \pi_s$. Предположим, что $\pi_1 \le \pi_2 \le \ldots \le \pi_s$. При реализации алгоритма система достигает реальной производительности r из (2.1). Будем сравнивать скорость работы системы со скоростью работы гипотетического простого универсального устройства, имеющего такую же пиковую производительность π_s, как самое быстрое ФУ системы, и обладающего возможностью выполнять те же операции, что все ФУ системы.

Итак, будем называть отношение $R = r/\pi_s$ ускорением реализации алгоритма на данной вычислительной системе или просто *ускорением*. Выбор в качестве гипотетического простого, а не какого-нибудь другого, например, конвейерного ФУ объясняется тем, что одно простое универсальное ФУ может быть полностью загружено на любом алгоритме. Принимая во внимание (2.1), имеем

$$R = \frac{\sum_{i=1}^{s} p_i \pi_i}{\max\limits_{1 \le i \le s} \pi_i}. \qquad (2.5)$$

Анализ определяющего ускорение выражения (2.5) показывает, что ускорение системы, состоящей из s устройств, никогда не превосходит s и может достигать s в том и только в том случае, когда все устройства системы имеют одинаковые пиковые производительности и полностью загружены.

Подводя итог проведенным исследованиям, приведем для одного частного случая полезное, хотя и очевидное

Утверждение 2.3

Если система состоит из s устройств одинаковой пиковой производительности простых или конвейерных, то:

- загруженность системы равна среднему арифметическому загруженностей всех устройств;
- реальная производительность системы равна сумме реальных производительностей всех устройств;
- пиковая производительность системы в s раз больше пиковой производительности одного устройства;
- ускорение системы равно сумме загруженностей всех устройств;
- если система состоит только из простых устройств, то ее ускорение равно отношению времени реализации алгоритма на одном универсальном простом устройстве с той же пиковой производительностью к времени реализации алгоритма на системе.

Пусть система устройств функционирует и показывает какую-то реальную производительность. Если производительность недостаточна, то в соответствии с (2.4) для ее повышения необходимо увеличить загруженность системы. Согласно (2.2) для этого, в свою очередь, нужно повысить загруженность любого устройства, у которого она еще не равна 1. Но остается открытым вопрос, всегда ли это можно сделать. Если устройство загружено не полностью, то его загруженность заведомо можно повысить в том случае, когда данное устройство не связано с другими. В случае же связанности устройств ситуация не очевидна.

Снова рассмотрим систему из s устройств. Не ограничивая общности, будем считать все устройства простыми, т. к. любое конвейерное ФУ всегда можно представить как линейную цепочку простых устройств. Допустим, что между устройствами установлены направленные связи, и они не меняются в процессе функционирования системы. Построим ориентированный мультиграф, в котором вершины символизируют устройства, а дуги — связи между ними. Дугу из одной вершины будем проводить в другую в том и только том случае, когда результат каждого срабатывания устройства, соответствующего первой вершине, обязательно передается в качестве аргумента для очередного срабатывания устройству, соответствующему второй вершине. Назовем этот мультиграф *графом системы.* Предположим, что каким-то образом в систему введены все исходные данные и она начала функционировать согласно описанным ранее правилам. Если в процессе функционирования какие-то ФУ будут требовать для своих срабатываний другие исходные данные, то будем предполагать, что они подаются на входы ФУ без задержек. Исследуем *максимальную* производительность системы, т. е. ее максимально возможную реальную производительность при достаточно большом времени функционирования.

Утверждение 2.4

Пусть система состоит из s простых устройств с пиковыми производительностями $\pi_1, \ldots, \pi_s$. Если граф системы связный, то максимальная производительность $r_{\max}$ системы выражается формулой

$$r_{\max} = s \min_{1 \le i \le s} \pi_i. \qquad (2.6)$$

Предположим, что дуга графа системы идет из i-го ФУ в j-ое ФУ. Пусть за достаточно большое время i-ое ФУ выполнило N_i операций, j-ое ФУ — N_j операций. Каждый результат i-го ФУ обязательно является одним из аргументов очередного срабатывания j-го ФУ. Поэтому количество операций, реализованных j-ым ФУ за время T, не может более, чем на 1, отличаться от

количества операций, реализованных i-ым ФУ, т. е. $N_i - 1 \le N_j \le N_i + 1$. Так как граф системы связный, то любые две вершины графа могут быть связаны цепью, составленной из дуг. Допустим, что граф системы содержит q дуг. Если k-ое ФУ за время T выполнило N_k операций, а l-ое ФУ — N_l операций, то из последних неравенств вытекает, что $N_l - q \le N_k \le N_l + q$ для любых k, l, $1 \le k, l \le s$. Пусть устройства перенумерованы так, что $\pi_1 \le \pi_2 \le \ldots \le \pi_s$. Принимая во внимание эту упорядоченность и полученные для числа выполняемых операций соотношения из (2.1), находим, что

$$r = \sum_{i=1}^{s} \left(\frac{N_i}{\pi_i T} \right) \pi_i \le \frac{N_1 s}{T} + \frac{q(s-1)}{T};$$

$$r \ge \frac{N_1 s}{T} - \frac{q(s-1)}{T}.$$

Вторые слагаемые в этих неравенствах стремятся к нулю при T, стремящемся к бесконечности. Для всех k, $1 \le k \le s$, обязаны выполняться неравенства $N_k \le \pi_k T$. В силу предполагаемой упорядоченности производительностей π_k и того, что все N_k асимптотически равны между собой, число операций, реализуемых каждым ФУ, будет асимптотически максимальным, если выполняется равенство $N_1 = \pi_1 T$. Это означает, что максимально возможная реальная производительность системы асимптотически будет равна $s\pi_1$, что совпадает с (2.6).

Следствие

Пусть вычислительная система состоит из s простых устройств с пиковыми производительностями $\pi_1, \ldots, \pi_s$. Если граф системы связный, то:

- асимптотически каждое из устройств выполняет одно и то же число операций;
- загруженность любого устройства не превосходит загруженности самого непроизводительного устройства;
- если загруженность какого-то устройства равна 1, то это — самое непроизводительное устройство;
- загруженность системы не превосходит

$$p_{\max} = \frac{s \min\limits_{1 \le i \le s} \pi_i}{\sum\limits_{i=1}^{s} \pi_i};$$

- ускорение системы не превосходит

$$R_{\max} = \frac{s \min\limits_{1 \le i \le s} \pi_i}{\max\limits_{1 \le i \le s} \pi_i}.$$

Установление всех приведенных здесь фактов осуществляется почти дословным повторением доказательства утверждения 2.4 и, естественно, использованием формул (2.2), (2.5).

Следствие (1-ый закон Амдала)

Производительность вычислительной системы, состоящей из связанных между собой устройств, в общем случае определяется самым непроизводительным ее устройством.

Заметим, что утверждение 2.4 указывает на одно из узких мест процесса функционирования системы. Некоторые узкие места вычислительных систем, описанные в литературе, так или иначе связываются с именем Амдала, американского специалиста в области вычислительной техники. Чтобы не терять узнаваемость различных фактов, мы не станем нарушать эту традицию и будем оставлять именными соответствующие утверждения, даже если они совсем простые. Возможно лишь несколько изменим формулировки, приспособив их к текущему изложению материала. Именно по этим причинам мы назвали *1-ым законом Амдала* последнее следствие из утверждения 2.4.

Следствие

Пусть система состоит из простых устройств и граф системы связный. Асимптотическая производительность системы будет максимальной, если все устройства имеют одинаковые пиковые производительности.

Когда мы говорим о максимально возможной реальной производительности, то подразумеваем, что функционирование системы обеспечивается таким расписанием подачи команд, которое минимизирует простой устройств. Максимальная производительность может достигаться при разных режимах. В частности, как следует из утверждения 2.4, она достигается при синхронном режиме с тактом, обратно пропорциональным производительности самого медленного из ФУ, если, конечно, система состоит из простых устройств и граф системы связный. Пусть система состоит из s простых устройств одинаковой производительности. Тогда как в случае связанной системы, так и в случае не связанной, максимально возможная реальная производительность при больших временах функционирования оказывается одной и той же и равной s-кратной пиковой производительности одного устройства.

Предположим, что по каким-либо причинам n операций из N мы вынуждены выполнять последовательно. Причины могут быть разными. Например, операции могут быть последовательно связаны информационно. И тогда без изменения алгоритма их нельзя реализовать иначе. Но вполне возможно, что мы просто не распознали параллелизм, имеющийся в той части алгоритма, которая описывается этими операциями. Отношение $\beta = n/N$ назовем *долей последовательных вычислений.*

Следствие (2-й закон Амдала)

Пусть система состоит из s одинаковых простых универсальных устройств. Предположим, что при выполнении параллельной части алгоритма все s устройств загружены полностью. Тогда максимально возможное ускорение равно

$$R = \frac{s}{\beta s + (1 - \beta)}. \tag{2.7}$$

Обозначим через π пиковую производительность отдельного ФУ. Согласно утверждению 2.3:

$$R = \sum_{i=1}^{s} p_i.$$

Если всего выполняется N операций, то среди них βN операций выполняется последовательно и $(1-\beta)N$ параллельно на s устройствах по $(1-\beta)N/s$ операций на каждом. Не ограничивая общности, можно считать, что все последовательные операции выполняются на первом ФУ. Всего алгоритм реализуется за время

$$T_1 = \frac{\beta N + (1-\beta)N/s}{\pi}.$$

На параллельной части алгоритма работают как первое, так и все остальные устройства, тратя на это время

$$T_i = \frac{(1-\beta)N/s}{\pi}$$

для $2 \le i \le n$. Поэтому $\rho_1 = 1$ и

$$\rho_i = \frac{(1-\beta)N/s}{\beta N + (\beta-1)N/s}.$$

Следовательно

$$R = 1 + \sum_{i=2}^{s} \frac{(1-\beta)N/s}{\beta N + (1-\beta)N/s} = \frac{s}{\beta s + (1-\beta)}.$$

Следствие (3-й закон Амдала)

Пусть система состоит из простых одинаковых универсальных устройств. При любом режиме работы ее ускорение не может превзойти обратной величины доли последовательных вычислений.

Необходимость оценки производительности и последующего сравнения компьютеров появилась практически одновременно с их рождением. Какой компьютер выбрать? Какому компьютеру отдать предпочтение? На каком компьютере задача будет решаться быстрее? Подобные вопросы задавались пользователями всегда и будут задаваться ими еще долго. Казалось бы, что тут сложного, запусти программу и проверь. Но не все так просто. Перенос параллельной программы на новую платформу требует времени, зачастую значительного. Много попыток в таких условиях не сделаешь, да и не всегда есть свободный доступ к новой платформе. А если на новом компьютере должен работать набор программ, как быть в такой ситуации?

Хотелось бы иметь простую единую методику априорного сравнения вычислительных систем между собой. В идеальной ситуации вычислять бы для каждой системы по некоторому закону одно число, которое и явилось его обобщенной характеристикой. Со сравнением компьютеров проблем не стало бы, с выбором тоже.

Сопоставить одно число каждому компьютеру можно по-разному. Например, можно вычислить это значение, опираясь на параметры самого компьютера. С этой точки зрения, естественной характеристикой любого компьютера является его *пиковая производительность*. Данное значение определяет тот максимум, на который способен компьютер. Вычисляется оно очень просто. Для этого достаточно предположить, что все устройства компьютера работают в максимально производительном режиме. Если в процессоре есть два конвейерных устройства, то рассматривается режим, когда оба конвейера одновременно работают с максимальной нагрузкой. Если в компьютере есть 1000 таких процессоров, то пиковая производительность одного процессора просто умножается на 1000.

Иногда пиковую производительность компьютера называют его теоретической производительностью. Этот нюанс в названии лишний раз подчеркивает тот факт, что производительность компьютера на любой реальной программе никогда не только не превысит этого порога, но и не достигнет его точно.

Пиковая производительность компьютера вычисляется однозначно, спорить тут не о чем, и уже этим данная характеристика хороша. Более того, подсознательно всегда возникает связь между пиковой производительностью компьютера и его возможностями в решении задач. Чем больше пиковая производительность, тем, вроде бы, быстрее пользователь сможет решить свою задачу. В целом данный тезис не лишен некоторого смысла, но лишь некоторого и даже "очень некоторого". Полагаться на него полностью нельзя ни в коем случае. С момента появления первых параллельных компьютеров пользователи убедились, что разброс в значениях реальной производительности может быть огромным. На одних задачах удавалось получать 90% от пиковой производительности, а на других лишь 2%. Если кто-то мог использовать независимость и конвейерность всех функциональных устройств компьютера CDC 7600, то производительность получалась высокой. Если в вычислениях были информационные зависимости, то конвейерность не использовалась, и произво-

дительность снижалась. Если в алгоритме явно преобладал один тип операций, то часть устройств простаивала, вызывая дальнейшее падение производительности. Об этом мы уже начинали говорить в *§ 2.3*.

Структура программы и архитектура компьютера тесно связаны. Пользователя не интересуют потенциальные возможности вычислительной системы. Ему нужно решить его конкретную задачу. Именно с этой точки зрения он и хочет оценить "качество" компьютера. Для обработки данных эксперимента в физике высоких энергий не требуется высокоскоростной коммуникационной среды. Главное — это большое число вычислительных узлов. Для таких целей вполне подойдет локальная 10-мегабитная сеть организации из ста рабочих станций. Ее можно рассматривать в качестве параллельного компьютера, и ночью целиком отдавать под такие задачи. Теперь попробуйте на таком компьютере запустить серьезную модель расчета изменения климата. Скорее всего, никакого ускорения решения задачи не будет. Имеем компьютер, с хорошим показателем пиковой производительности. Но для одних задач он подходит идеально, а для других никуда не годится.

Традиционно используются два способа оценки пиковой производительности компьютера. Один из них опирается на *число команд, выполняемых компьютером в единицу времени*. Единицей измерения, как правило, является MIPS (Million Instructions Per Second). Для определенного класса приложений такой подход является вполне приемлемым, поскольку общее представление о скорости работы компьютера получить можно. Но, опять-таки, только самое общее. Производительность, выраженная в MIPS, говорит о скорости выполнения компьютером своих же инструкций. Но в какое число инструкций отобразится программа пользователя или отдельный ее оператор? Заранее не ясно. К тому же, каждая программа обладает своей спецификой, число команд той или иной группы от программы к программе может меняться очень сильно. В этом контексте данная характеристика действительно дает лишь самое общее представление о производительности компьютера.

Интересный эффект в оценке производительности компьютера на основе MIPS наблюдается в компьютерах, в которых для выполнения вещественной арифметики применяются сопроцессоры. В самом деле, операции над числами, представленными в форме с плавающей запятой, выполняются дольше простых управляющих инструкций. Если такие операции выполняются без сопроцессора в режиме эмуляции, то срабатывает целое множество небольших инструкций. Время эмуляции намного больше, чем выполнение операции сопроцессором, но каждая небольшая инструкция срабатывает быстрее, чем команда сопроцессору. Вот и получается, что использование сопроцессора уменьшает время работы программы, зато в режиме эмуляции производительность компьютера, выраженная в MIPS, может оказаться значительно больше.

При обсуждении производительности компьютера не менее важен и вопрос о *формате используемых данных*. Если процессор за один такт может выполнять операции над 32-разрядными вещественными числами, то его же производительность при работе с 64-разрядной арифметикой может упасть во много раз. Известно, что первый матричный компьютер ILLIAC IV мог выполнять до 10 миллиардов операций в секунду. И это в 1974 году! Если брать за основу только цифры, то впечатляет. А если посмотреть вглубь, то окажется, что это верно лишь для простых команд, оперирующих с байтами. Помимо работы с 64-разрядными числами, процессорные элементы ILLIAC IV могли интерпретировать и обрабатывать данные уменьшенного формата. Например, одно слово они могли рассматривать как два 32-разрядных числа или восемь однобайтовых. Именно этот дополнительный внутренний параллелизм и позволял получить столь внушительные характеристики.

Для задач вычислительного характера, в которых важна высокая скорость выполнения операций над вещественными числами, подход к определению производительности также не должен опираться на скорость выполнения машинных инструкций. Во многих случаях операции над вещественными числами выполняются медленнее, чем, скажем, управляющие или регистровые операции. За время выполнения одной операции умножения двух чисел в форме с плавающей запятой может выполниться десяток простых инструкций. Это послужило причиной введения другого способа измерения пиковой производительности: *число вещественных операций, выполняемых компьютером в единицу времени*. Единицу измерения назвали Flops, что является сокращением от Floating point operations per second. Такой способ, безусловно, ближе и понятнее пользователю. Операция $a + b$ в тексте программы всегда соответствует одной операции сложения в коде программы. Пользователь знает вычислительную сложность своей программы, а на основе этой характеристики может получить нижнюю оценку времени ее выполнения.

Да, к сожалению, оценка будет только нижней. В этом и кроется причина недовольства и разочарований. Взяв за ориентир пиковую производительность компьютера, пользователь рассчитывает и на своей программе получить столько же. Совершенно не учитывается тот факт, что пиковая производительность получается при работе компьютера в идеальных условиях. Нет конфликтов при обращении к памяти, все берется с регистров, все устройства постоянно и равномерно загружены и т. п. Но в жизни так бывает очень редко. В каждой программе пользователя есть свои особенности, которые эти идеальные условия нарушают, обнажая узкие места архитектуры. Особенности структуры процессора, система команд, состав функциональных устройств, строение и объем кэш-памяти, архитектура подсистемы доступа в память, реализация ввода/вывода — все это, и не только это, может повлиять на выполнение реальной программы. Причем заметим, мы перечислили только части аппаратуры, а говорить надо о *программно-аппаратной среде* выполнения программ в целом. Если для компьютера нет эффектив-

ных компиляторов, то пиковая производительность будет вводить в заблуждение еще сильнее.

Значит нужно отойти от характеристик аппаратуры и оценить эффективность работы программно-аппаратной среды на фиксированном наборе задач. Бессмысленно для каждого компьютера показывать свой набор. Разработчики без труда придумают такие программы, на которых их компьютер достигает производительности, близкой к пиковой. Такие примеры никого не убедят. Набор тестовых программ должен быть зафиксирован. Эти программы будут играть роль эталона, по которому будут судить о возможностях вычислительной системы.

Такие попытки неоднократно предпринимались. На основе различных критериев формировались тестовые наборы программ или фиксировались отдельные эталонные программы (такие программы иногда называют *бенчмарками*, отталкиваясь от английского слова *benchmark*). Программы запускались на различных системах, замерялись те или иные параметры, на основе которых в последствии проводилось сравнение компьютеров между собой. В дальнейшем изложении для подобных программ мы чаще всего будем использовать слово тест, делая акцент не на проверке правильности работы чего-либо, а на тестировании эффективности работы вычислительной системы.

Что имело бы смысл взять в качестве подобного теста? Что-то несложное и известное всем. Таким тестом стал LINPACK. Эта программа предназначена для решения системы линейных алгебраических уравнений с плотной матрицей с выбором главного элемента по строке. Простой алгоритм, регулярные структуры данных, значительная вычислительная емкость, возможность получения показателей производительности, близких к пиковым, — все эти черты сделали тест исключительно популярным.

Этот тест рассматривается в трех вариантах. В первом варианте решается система с матрицей размера 100×100. Предоставляется уже готовый исходный текст, в который не разрешается вносить никаких изменений. Изначально предполагалось, что запрет внесения изменений в программу не позволит использовать каких-либо специфических особенностей аппаратуры, и показатели производительности будут сильно занижены. В настоящее время ситуация полностью изменилась. Матрица столь небольшого размера легко помещается целиком в кэш-память современного процессора (нужно лишь 80 Кбайт), что завышает его характеристики.

Во втором варианте теста размер матрицы увеличивается до 1000×1000. Разрешается как внесение изменений в текст подпрограммы, реализующей заложенный авторами метод решения системы, так и изменение самого метода. Единственное ограничение — это использование стандартной вызывающей части теста, которая выполняет инициализацию матрицы, вызывает подпрограмму решения системы и проверяет корректность результатов.

В этой же части вычисляется и показанная компьютером производительность, исходя из формулы для числа операций $2n^3/3 + 2n^2$ (n — это размер матрицы), вне зависимости от вычислительной сложности реально использованного алгоритма.

В данном варианте достигались значения производительности, близкие к пиковой. Этому способствовали и значительный размер матрицы, и возможность внесения изменений в программу или алгоритм. Отсюда появилось и специальное название данного варианта — LINPACK TPP, Toward Peak Performance.

С появлением больших массивно-параллельных компьютеров вопрос подбора размера матрицы стал исключительно актуальным. На матрицах 100×100 или 1000×1000 никаких разумных показателей получить не удавалось. Эти задачи были слишком малы. Матрица размера 1000×1000 занимает лишь 0,01—0,001% всей доступной оперативной памяти компьютера. Первые эксперименты с тестом LINPACK на реальных массивно-параллельных компьютерах показали, что и фиксировать какой-либо один размер задачи тоже нельзя. В результате в третьем варианте теста было разрешено использовать изложенную во втором варианте методику для матриц сколь угодно большого размера. Сколько есть оперативной памяти на всех вычислительных узлах системы, столько и можно использовать. Для современных компьютеров размер матрицы уже перевалил за миллион, поэтому такой шаг был просто необходим.

Для работы теста LINPACK на вычислительных системах с распределенной памятью была создана специальная версия HPL (High-Performance LINPACK). В отличие от стандартной реализации, в HPL пользователь может управлять бóльшим числом параметров для достижения высокой производительности на своей установке.

В настоящее время LINPACK используется для формирования списка Top500 — пятисот самых мощных компьютеров мира (**www.top500.org**). Кроме пиковой производительности R_{peak} для каждой системы указывается величина R_{max}, равная производительности компьютера на тесте LINPACK с матрицей максимального для данного компьютера размера N_{max}. По значению R_{max} отсортирован весь список. Как показывает анализ представленных данных, величина R_{max} составляет 50—70% от значения R_{peak}. Интерес для анализа функционирования компьютера представляет и указанное в каждой строке значение $N_{1/2}$, показывающее, на матрице какого размера достигается половина производительности R_{max}.

Что же показывают данные теста LINPACK? Только одно — насколько хорошо компьютер может решать системы уравнений с плотной матрицей указанным методом. Поскольку задача имеет хорошие свойства, то и корреляция производительности на тесте LINPACK с пиковой производительностью компьютера высока. Операции ввода/вывода не затрагиваются, отношение числа выполненных операций к объему используемых данных высокое, регулярность структур данных и вычислений, простая коммуникационная схема, относительно небольшой объем передач между процессорами и другие свойства программы делают данный тест "удобным". Безусловно, по данным этого теста можно получить много полезной информации о вычислительной системе и с этим никто не спорит. Но нужно ясно понимать и то, что высокая производительность на LINPACK совершенно не означает того, что и на вашей конкретной программе будет достигнута высокая производительность.

Интересное свойство LINPACK связано с законом Мура. Согласно этому закону производительность вычислительных систем удваивается каждые 18 месяцев. Не имея строгих доказательств, этот закон, тем не менее, подтверждается уже не один десяток лет. Согласно закону, объем памяти увеличивается в четыре раза каждые три года. Это позволит каждые три года удваивать размер используемой матрицы. За эти же три года производительность компьютеров вырастет в четыре раза. Поскольку вычислительная сложность теста LINPACK есть куб от размера матрицы, то время выполнения третьего варианта теста должно удваиваться каждые три года. Получается, что фиксировать размер матрицы нельзя, а использование матриц максимального размера со временем приведет к очень большим затратам. Желателен компромисс, который пока не найден.