Лекция 2. Асимптотический анализ алгоритмов и распараллеливание

Для многих задач математического моделирования можно сформулировать несколько алгоритмов, решающих поставленную задачу. Конечно, речь идет лишь о корректно работающих алгоритмах, т.е. о таких, которые дают правильный результат на достаточно широком наборе входных данных. Именно с ними мы будем в дальнейшем иметь дело, поэтому вопросы, связанные с корректностью, останутся за пределами нашего рассмотрения.

Как понять, является ли выбранный алгоритм «достаточно хорошим» для его реализации или требуется поискать что-либо иное? Как правило, в большинстве реальных задач присутствуют некоторые *параметры масштаба*, связанные с объемом входных данных. Например, в задачах сортировки массива информации таким параметром является размер исходного массива. В задачах численного моделирования параметрами масштаба служат размеры расчетной сетки. В ряде работ, например в [12], вместо введенного названия принято использовать термин «параметры размерности», но с моей точки зрения это не совсем правильно. Термин «размерность» при вычислительном эксперименте обычно связывают с количеством независимых переменных. Мы постараемся избежать неоднозначности.

Параметры масштаба задачи влияют на время работы различных алгоритмов и на объем ресурсов, необходимых для их эффективной реализации (память, количество исполнителей и т.д.). Для современных вычислительных комплексов на первый план (хотя и не всегда) выходит именно время выполнения алгоритма. Чем оно меньше, тем лучше для пользователя. Введем следующие обозначения. Если задача имеет один параметр масштаба n, то время выполнения алгоритма A запишем так: $T_A(n)$. Для нескольких параметров масштаба $n_1, \ldots, n_k$ соответствующее обозначение будет $T_A(n_1, \ldots, n_k)$ или $T(\mathbf{n})$.. Мы будем сравнивать различные алгоритмы по времени выполнения при одних и тех же значениях параметров масштаба с соблюдением следующих условий (для простоты считаем, что у нас один параметр).

1. Оценка $T(n)$ не может быть привязана к конкретной вычислительной системе. Если для алгоритма A известна величина $T_A(n)$ для одного компьютера, то, измерив для алгоритма B время $T_B(n)$ на некоторой другой системе, нельзя ничего сказать о сравнительной эффективности A и B. Полученные таким способом оценки могут свидетельствовать о недостатках или преимуществах самих компьютерных комплексов, а не реализованных на них алгоритмов. Для получения корректных оценок необходимо использовать некоторую единую теоретическую модель вычислительной системы.

2. Сравнение $T_A(n)$ и $T_B(n)$ на теоретической модели при малых значениях n возможно, но бесполезно. Для малого значения параметра масштаба даже неэффективный алгоритм выполняется быстро. Для пользователя вряд ли существенна разница во временах исполнения алгоритмов, если один из них работает 0.5 секунды, а второй — 0.1 секунды. Сравнение эффективности A и B должно проводиться при больших значениях n, в идеале при $n \to \infty$.

3. Нас будет интересовать не сравнение собственно значений $T_A(n)$ и $T_B(n)$ при больших n, а сравнение их темпов роста. Если $T_A(n) = 10^6 \times n^2$, а $T_B(n) = n^3$, то при $n < 10^6$ алгоритм B эффективнее, но при $n > 10^6$ эффективнее окажется уже алгоритм A.

Иными словами, мы будем сравнивать асимптотическое поведение времен исполнения алгоритмов при $n \to \infty$ на теоретической модели ЭВМ.

Для дальнейшего изложения нам потребуется некоторые стандартные формы записи, используемые при асимптотическом анализе поведения функций. Предположим, что есть две функции f и g от целочисленного положительного аргумента n, принимающие положительные значения. Тогда:

O1.1. $f(n) = O(g(n))$ тогда и только тогда, когда существуют положительные константы c и n_0 такие, что $f(n) \le cg(n)$ при $n \ge n_0$.

O1.2. $f(n) = \Omega(g(n))$ тогда и только тогда, когда существуют положительные константы c и n_0 такие, что $cg(n) \le f(n)$ при $n \ge n_0$.

O1.3. $f(n) = \Theta(g(n))$ тогда и только тогда, когда существуют положительные константы c_1, c_2 и n_0 такие, что $c_1 g(n) \le f(n) \le c_2 g(n)$ при $n \ge n_0$.

Отметим, что если $f(n) = \Theta(g(n))$, то $f(n) = O(g(n))$ и $f(n) = \Omega(g(n))$.

Дадим ряд определений.

Пусть $T_A(n)$ и $T_B(n)$ — времена работы на одной и той же вычислительной системе последовательных алгоритмов A и B соответственно, а $T_0(n)$ — теоретическая оценка времени работы снизу произвольного последовательного алгоритма для решения той же самой задачи, т.е. $T_0(n) = \Omega(T(n))$ для любого алгоритма. Тогда будем говорить, что

O2.1. Если $T_A(n) = O(T_B(n))$, то алгоритм A по поведению не хуже алгоритма B.

O2.2. Если $T_A(n) = \Omega(T_B(n))$, то алгоритм A по поведению не лучше алгоритма B.

O2.3. Если $T_A(n) = \Theta(T_B(n))$, то алгоритмы A и B по поведению одинаковы.

O2.4. Если $T_A(n) = \Theta(T_0(n))$, то алгоритм A — *оптимален*.

При этом все времена измеряются для наихудших входных данных.

Для оценки времени работы последовательных алгоритмов обычно используется модель вычислительной системы, получившая название *RAM* (*Random Access Machine*). Ее поведение регламентируется следующими свойствами:

1. В системе имеется один процессор с одним ядром (один исполнитель)
2. Ячейки памяти для чтения и записи доступны в произвольном порядке
3. Время доступа к памяти есть $\Theta(1)$, независимо от того является операция доступа чтением или записью.
4. Время выполнения основных операций на исполнителе есть $\Theta(1)$.

Рассмотрим на простом примере то, как осуществляется асимптотическая оценка времени работы алгоритма. Возьмем **задачу выбора**.

Возьмем множество $S = \{s_1, s_2, \ldots, s_n\}$, на котором задан линейный порядок, т.е. для любых двух элементов множества можно определить: равны ли они, или один меньше другого. Элементом ранга k, $1 \le k \le n$, для множества S назовем s_i, если он является k-м наименьшим элементом этого множества. Для однозначного определения ранга будем считать, что если $s_i = s_j$, $i < j$, то s_i имеет меньший ранг, чем s_j.

Элемент ранга 1 имеет минимальное значение в множестве. Элемент ранга n — максимальное значение. Элемент ранга $\lceil n/2 \rceil$ ($\lceil\ \rceil$ — округление до целого сверху) называют медианой множества. Медиана множества имеет замечательное свойство: в множестве содержится не менее $\lceil n/2 \rceil$ элементов превышающих по значению медиану или равных ей, и не менее $\lceil n/2 \rceil$ элементов непревышающих этого значения.

Задача выбора заключается в поиске значения элемента с рангом k в множестве S. Конечно, возможно тривиальное решение задачи: отсортируем наше множество в порядке возрастания значений элементов и выберем нужное значение. Однако, оно предполагает выполнение лишней работы — нам не нужен весь отсортированный массив, нам нужно одно значение.

Мы пойдем другим путем. Для решения этой задачи возьмем простой рекурсивный алгоритм, состоящий из пяти шагов [13].

Шаг 1. Если мощность множества S меньше некоторой небольшой константы — $|S| < q$, то искомое значение ищем с помощью сортировки множества (при малом значении параметра масштаба это допустимо). Значение q определим позже. В противном случае разобьем исходное множество на $\lceil |S| / q \rceil$ подмножеств S_i, в каждое из которых, за исключением, быть может, последнего, войдет по q элементов.

Шаг 2. В каждом из подмножеств S_i сортировкой ищем его медиану m_i.

Шаг 3. Из найденных медиан строим множество $M = \left\{ m_1, m_2, \ldots, m_{\lceil |S|/q \rceil} \right\}$. Рекурсивно находим m_0 — медиану множества M.

Шаг 4. Исходное множество S разбиваем на три подмножества L, E и G следующим образом:

$$L : s_i \in L \leftrightarrow s_i < m_0$$

$$E : s_i \in E \leftrightarrow s_i = m_0$$

$$G : s_i \in G \leftrightarrow s_i > m_0$$

Шаг 5. Если $|L| \geq k$, то искомый элемент находится в множестве L, и мы рекурсивно запускаем наш алгоритм на поиск элемента ранга k в множестве L. В противном случае, если $|L| + |E| \geq k$, то искомый элемент принадлежит множеству E, и его значение есть m_0. Если же $|L| + |E| < k$, то наш элемент входит в множество G, и мы рекурсивно ищем в этом множестве элемент ранга $k - |L| - |E|$.

Оценим время работы $T(n)$ данного алгоритма сверху.

Шаг 1. Если $|S| < q$, то требуемое значение мы найдем за время $\Theta(1)$. В противном случае для разбиения множества S на $\lceil |S| / q \rceil$ подмножеств нам потребуется время $c_1 * |S| = c_1 * n$. Для первого шага получаем $T_1(n) = c_1 * n$.

Шаг 2. Время поиска каждой медианы есть $\Theta(1)$, но таких медиан у нас $\lceil |S| / q \rceil$ штук. Поэтому для второго шага оценка времени работы $T_2(n) = c_2 * |S| = c_2 * n$.

Шаг 3. Мощность множества M есть $\lceil |S| / q \rceil$. Для поиска медианы в этом множестве мы потратим время $T_3(n) = T(\lceil |S| / q \rceil) = T(n/q)$.

Шаг 4. Для построения множеств L, E и G мы должны каждый элемент исходного множества сравнить со значением m_0. Следовательно, для этого шага $T_4(n) = c_3 * n$.

Шаг 5. Если искомый элемент попал в множество E, то решение уже есть — время $\Theta(1)$. Допустим, что он попал в множество L. Оценим мощность множества L сверху. В множестве M не менее $\lceil |M| / 2 \rceil = \lceil n / (2q) \rceil$ элементов m_i, превышающих или равных m_0. В каждом из соответствующих множеств S_i не менее $\lceil |S_i| / 2 \rceil = \lceil q / 2 \rceil$ элементов множества S, превышающих или равных m_i. Стало быть, $|G| + |E| \geq \lceil n / (2q) \rceil \times \lceil q / 2 \rceil = \lceil n / 4 \rceil$. Поэтому $|L| \leq 3n/4$. Аналогичную оценку сверху можно получить и для мощности множества G. Поэтому время работы данного шага можно оценить как $T_5(n) = T(3n/4)$.

Окончательно

$$T(n) = T_1(n)+T_2(n)+T_3(n)+T_4(n)+T_5(n) = c_4 * n + T(n/q) + T(3n/4). \quad (2.1)$$

Строгое решение такого уравнения весьма трудоемко, поэтому мы прибегнем к некоторым нестрогим «шаманским танцам», чтобы обосновать результат.

Шаманский танец 1. Пусть $n/q + 3n/4 < n$, тогда $q \le 5$. Положим $q = 5$. Имеем $T(n) = c_4 * n + T(n/5) + T(3n/4)$.

Шаманский танец 2. Предположим, что $T(n) \le c_5 * n$. Тогда при $c_5 = 20 * c_4$ получаем $T(n) \le c_4 * n + 20 * c_4 * n/5 + 60 * c_4 * n/4 = c_5 * n$. Предположение не противоречит уравнению, и, значит, $T(n) = O(n)$.

Найдем теоретическую оценку времени работы алгоритма снизу. Для того, чтобы отыскать значение элемента с рангом k, нам нужно хотя бы раз взглянуть на значения всех элементов. Поэтому $T(n) = \Omega(n)$ и, окончательно, $T(n) = \Theta(n)$. Рассмотренный алгоритм к тому же оказался оптимальным.

Для аккуратного решения соотношений, подобных (2.1), используют **основную теорему асимптотического анализа.**

Если $T(n) = aT(n/b) + f(n)$, где a и b константы, $a \ge 1$, $b > 1$, $f(n) > 0$ и n принимает целые неотрицательные значения, то

1. Если $f(n) = O(n^{\log_b a - \varepsilon})$, где ε константа > 0, то $T(n) = \Theta(n^{\log_b a})$.

2. Если $f(n) = \Theta(n^{\log_b a})$, то $T(n) = \Theta(n^{\log_b a} \log n)$

3. Если $f(n) = O(n^{\log_b a + \varepsilon})$, где ε константа > 0 и существуют константы c и N, $0 < c < 1$, $N > 0$, такие, что при $(n/b) > N$ выполнено $af(n/b) \le cf(n)$, то $T(n) = \Theta(f(n))$.

Доказательство теоремы можно найти в [2.1], где оно занимает 10 страниц убористого текста. И хотя для уравнения (2.1) теорема прямо неприменима, способом, аналогичным ее доказательству, можно строго обосновать шаманские танцы.

Для оценки времени работы параллельных алгоритмов самой простой моделью вычислительной системы является обобщение модели RAM, которую принято обозначать *PRAM* (*Parallel Random Access Machine*). Ее основные характеристики:

1. В системе имеется много процессоров и/или ядер (несколько исполнителей)
2. Ячейки памяти для чтения и записи доступны в произвольном порядке
3. Время доступа к памяти есть $\Theta(1)$, независимо от того является операция доступа чтением или записью.
4. Время выполнения основных операций на исполнителе есть $\Theta(1)$.

Параллельные алгоритмы решения задачи помимо параметра масштаба задачи n имеют дополнительный параметр масштаба N — количество исполнителей, задействованных при их выполнении.

Зафиксируем количество исполнителей N. Тогда для двух параллельных алгоритмов A и B, работающих на одной и той же параллельной вычислительной системе, можно ввести определения по аналогии с определениями О2.1 – О2.3.

О2.5. Если $T_A(n) = O(T_B(n))$, то параллельный алгоритм A по поведению не хуже алгоритма B.

О2.6. Если $T_A(n) = \Omega(T_B(n))$, то параллельный алгоритм A по поведению не лучше алгоритма B.

О2.7. Если $T_A(n) = \Theta(T_B(n))$, то параллельные алгоритмы A и B по поведению одинаковы.

Сравнение параллельных алгоритмов по масштабируемости по числу исполнителей — это отдельный вопрос, который в рамках данного курса не рассматривается.

Для нас наиболее важным является не сравнение времен работы различных параллельных алгоритмов, а определение того насколько параллельный алгоритм лучше существующих последовательных алгоритмов.

Возьмем некоторый параллельный алгоритм, решающий задачу математического моделирования, и наилучший последовательный алгоритм для этой же задачи. Будем говорить, что параллельный алгоритм по сравнению с последовательным дает *ускорение*, определяемое соотношением:

Ускорение = (время работы наилучшего последовательного алгоритма при наихудших начальных данных) / (время работы параллельного алгоритма при тех же начальных данных)

Понятно, что при наличии N исполнителей ускорение не может превышать N, так как в противном случае, выполнив параллельные куски последовательно на одном исполнителе, мы получим последовательный алгоритм еще лучше.

Теоретическое определение ускорения на практике используется редко. Попробуйте-ка найти наилучший последовательный алгоритм (не оптимальный, а именно наилучший)! Поэтому обычно применяют практическое определение ускорения, сравнивая времена выполнения последовательного алгоритма и его параллельной версии.

Практическое ускорение = (время работы последовательного алгоритма при наихудших начальных данных) / (время работы параллельной версии этого алгоритма при тех же начальных данных)

Дополнительно к понятию ускорения вводится понятие *стоимости* параллельного алгоритма.

Стоимость = (время работы параллельного алгоритма) × (количество исполнителей)

Пусть время работы последовательного алгоритма $T_s(n) = \Theta(f(n))$, и стоимость параллельного алгоритма есть тоже $\Theta(f(n))$, тогда параллельный алгоритм называют *оптимальным по стоимости*.

Построение параллельного алгоритма для задачи выбора и оценку его стоимости можно проделать самостоятельно или найти в [13].