Подходы к проектированию алгоритмов

Луцив Дмитрий Вадимович

Кафедра системного программирования

Содержание I

1. Распараллеливание
2. Примеры параллельных алгоритмов
3. Жадное и динамическое программирование
4. Теорема о рекуррентных соотношениях
5. Упражнения и вопросы

Распараллеливание

Основы

- Никакие алгоритмы не помогут ускорить процесс при помощи нескольких процессоров с такой же суммарной вычислительной мощностью. Только замедлят.
- Разбиение данных с параллельной обработкой фрагментов и разбиение алгоритмов с конвейеризацией — наиболее универсальные подходы.

Основная проблема в том, что поднять мощность одного процессора в n раз очень непросто.

Уровни параллелизма

- Уровень заданий. Несколько независимых заданий одновременно выполняются на разных процессорах, практически не взаимодействуя друг с другом. Реализуется на ВС с множеством процессоров в многозадачном режиме.

Уровни параллелизма

- Уровень заданий. Несколько независимых заданий одновременно выполняются на разных процессорах, практически не взаимодействуя друг с другом. Реализуется на ВС с множеством процессоров в многозадачном режиме.
- Уровень программ. Части одной задачи выполняются на множестве процессоров. Достигается на параллельных ВС.

Уровни параллелизма

- Уровень заданий. Несколько независимых заданий одновременно выполняются на разных процессорах, практически не взаимодействуя друг с другом. Реализуется на ВС с множеством процессоров в многозадачном режиме.
- Уровень программ. Части одной задачи выполняются на множестве процессоров. Достигается на параллельных ВС.
- Уровень команд. Выполнение команды разделяется на фазы, а фазы нескольких последовательных команд м.б. перекрыты за счет конвейеризации. Достижим на ВС с одним процессором.

Уровни параллелизма

- Уровень заданий. Несколько независимых заданий одновременно выполняются на разных процессорах, практически не взаимодействуя друг с другом. Реализуется на ВС с множеством процессоров в многозадачном режиме.
- Уровень программ. Части одной задачи выполняются на множестве процессоров. Достигается на параллельных ВС.
- Уровень команд. Выполнение команды разделяется на фазы, а фазы нескольких последовательных команд м.б. перекрыты за счет конвейеризации. Достижим на ВС с одним процессором.
- Уровень битов и скаляров (арифметический уровень) Биты слова обрабатываются одновременно. Реализуется в обычных и суперскалярных процессорах. Скаляры параллельно обрабатываются в векторных процессорах.

Типы архитектур

- Суперскалярные процессоры
- VLIW
- Векторные, матричные и тензорные процессоры
- Кластеры, гриды

Языки программирования

- Обычные
- С использованием библиотек параллельной обработки данных
- С использованием параллельных расширений
- Специализированные параллельные языки
- Обычные языки с естественной поддержкой параллельности

Топологии

- Шина
- Кольцо
- Плоскость
- Цилиндр
- Тор
- Матрица (полный граф)
- M динамических линий между N процессорами
- Гиперкуб
- Топологии, имитирующие физику задач

Показатели эффективности распараллеливания

- $O(n), O(1)$ – количество операций на системе с n или 1 процессорами
- $T(n), T(1)$ – время выполнения
- $S(n) = \frac{T(1)}{T(n)}$ – ускорение

Показатели эффективности распараллеливания

- $O(n), O(1)$ – количество операций на системе с n или 1 процессорами
- $T(n), T(1)$ – время выполнения
- $S(n) = \frac{T(1)}{T(n)}$ – ускорение

Берём $T(1) = O(1)$

Параллельное выполнение д.б., по крайней мере, не медленнее, чем послдовательное, так что:

- $T(n) \leq O(n) \leq nO(1) = nT(1)$
- $1 \leq S(n) \leq n$ – ускорение

Показатели эффективности распараллеливания

- $O(n), O(1)$ – количество операций на системе с n или 1 процессорами
- $T(n), T(1)$ – время выполнения
- $S(n) = \frac{T(1)}{T(n)}$ – ускорение

Берём $T(1) = O(1)$

Параллельное выполнение д.б., по крайней мере, не медленнее, чем послдовательное, так что:

- $T(n) \leq O(n) \leq nO(1) = nT(1)$
- $1 \leq S(n) \leq n$ – ускорение
- $1/n \leq E(n) = S(n)/n = \frac{T(1)}{nT(n)} \leq 1$ – эффективность

Закон Амдала

f – доля последовательного кода, $(1 - f)$ – параллельного.

$$T_{par} = fT_{seq} + \frac{(1-f)T_{seq}}{n}$$

$\Rightarrow$

$$S = T_{seq}/T_{par} = \frac{n}{1 + (n-1)f} \overset{n\to\infty}{\longrightarrow} 1/f$$

Примеры параллельных алгоритмов

Алгоритм Штрассена

$$C = AB \qquad A, B, C \in \mathbb{R}^{2^n \times 2^n}$$

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}.$$

$$A_{i,j}, B_{i,j}, C_{i,j} \in \mathbb{R}^{2^{n-1} \times 2^{n-1}}$$

Алгоритм Штрассена

$$C = AB \qquad A, B, C \in \mathbb{R}^{2^n \times 2^n}$$

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}.$$

$$A_{i,j}, B_{i,j}, C_{i,j} \in \mathbb{R}^{2^{n-1} \times 2^{n-1}}$$

Это 8 умножений:

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}; C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2};$$
$$C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}; C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}.$$

А можно за 7!

Поразрядная сортировка

Пример для двоичной системы.

- LSD – сперва младшие, затем консервативно (стабильно) относительно порядка младших старшие:
  - Сперва фильтруются те, у которых старший бит 0,
  - затем те, у кого 1,
  - затем они сливаются.

Поразрядная сортировка

Пример для двоичной системы.

- LSD – сперва младшие, затем консервативно (стабильно) относительно порядка младших старшие:
  - Сперва фильтруются те, у которых старший бит 0,
  - затем те, у кого 1,
  - затем они сливаются.
  - Это частный случай блочной (корзинной, карманной) сортировки
- MSD – сперва старшие, затем рекурсивно младшие:
  - Фактически, это QuickSort, но разделение по значению очередного разряда

Поразрядная сортировка

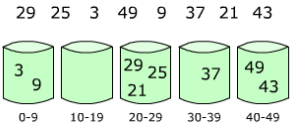
Пример для двоичной системы.

- LSD – сперва младшие, затем консервативно (стабильно) относительно порядка младших старшие:
  - Сперва фильтруются те, у которых старший бит 0,
  - затем те, у кого 1,
  - затем они сливаются.
  - Это частный случай блочной (корзинной, карманной) сортировки
- MSD – сперва старшие, затем рекурсивно младшие:
  - Фактически, это QuickSort, но разделение по значению очередного разряда
- И старшие с младшими можно сортировать параллельно!

BucketSort

Развитие идеи QuickSort

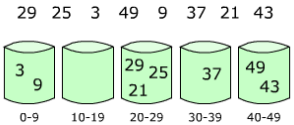
- Распределяем по блокам



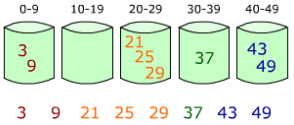
BucketSort

Развитие идеи QuickSort

- Распределяем по блокам



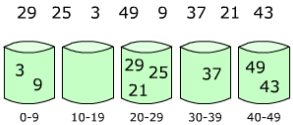
- Сортируем блоки (можно параллельно и распределённо)



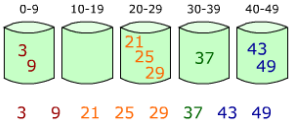
BucketSort

Развитие идеи QuickSort

- Распределяем по блокам



- Сортируем блоки (можно параллельно и распределённо)

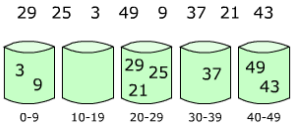


- Сливаем

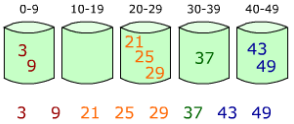
BucketSort

Развитие идеи QuickSort

- Распределяем по блокам



- Сортируем блоки (можно параллельно и распределённо)



- Сливаем

Как и QuickSort, можно затормозить специально подобранными данными.

Аппаратная реализация

Раскладочно-подборочные машины

Map-Reduce: модель

Парадигма и паттерн. Но не «технология».

Map-Reduce: модель

Парадигма и паттерн. Но не «технология».

V – пространство значений

$m : V \to C$ – функция разбиения на классы эквивалентности

$r : W \in 2^V \to R \mid \forall v \in W \; f(v) = c \in C$ – функция редукции

Map-Reduce: модель

Парадигма и паттерн. Но не «технология».

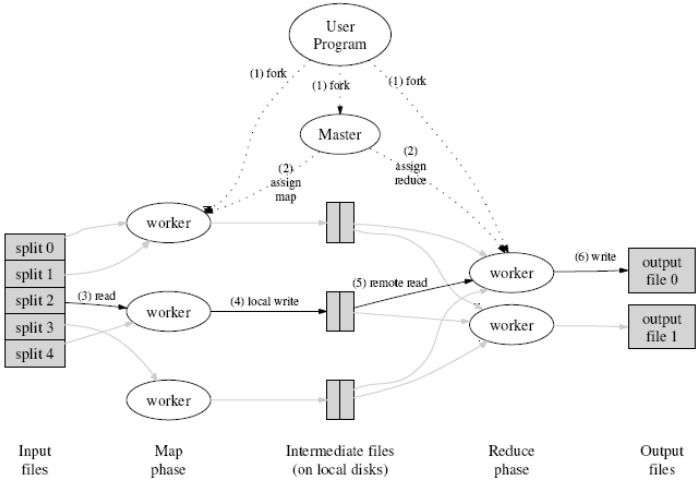
V – пространство значений

$m : V \to C$ – функция разбиения на классы эквивалентности

$r : W \in 2^V \to R \mid \forall v \in W \; f(v) = c \in C$ – функция редукции

Редукция может быть иерархической

Map-Reduce: топология



Map-Reduce: некомпьютерный пример

- Несколько человек берут стаканы с деньгами, выбирают монетки и сортируют на кучки с одним достоинством (map).
- Считается количество в каждой кучке и, таким образом, её достоинство (reduce1).
- Суммируются достоинства кучек монет: сперва для одного человека (reduce2), потом общее (reduce3).

Жадное и динамическое программирование

Суть жадных алгоритмов

- Принимать решение, дающее максимальную выгоду на текущем шаге
- Бери, пока (и что) дают

Суть жадных алгоритмов

- Принимать решение, дающее максимальную выгоду на текущем шаге
- Бери, пока (и что) дают

Не претендуют на поиск оптимального решения, но для каких-то задач находят его

Жадный алгоритм на базе металлоприёмки

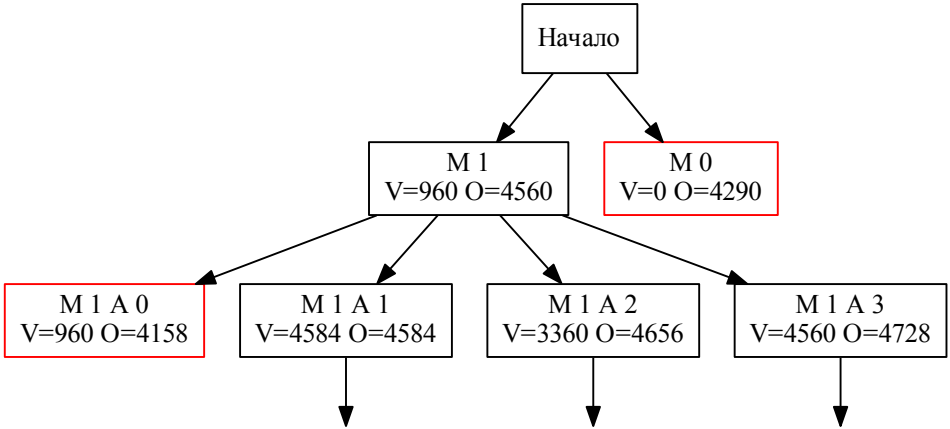
1. На складе мелкий лом
2. Можем унести ограниченное количество
3. Берём самый дорогой, сколько есть и сколько влезет
4. Если осталось место, то повторяем (3) для следующего по удельной цене

Динамическое программирование

- Решаем задачу в несколько шагов, «ветвясь в ширину»
- На следующем шаге пользуемся частичными данными со всех ветвей предыдущих шагов (как правило одного предыдущего шага)
  - важно: на каждом шаге мы должны обрабатывать лишь наилучшие промежуточные результаты, полученные на предыдущем, в этом и выгода

ДП: Метод ветвей и границ / Задача о рюкзаке

На складе не лом, а изделия (стоят дороже)



ДП: Метод ветвей и границ / Выключка абзаца

Надо минимизировать штраф за переносы подряд, слишком узкие или широкие апроши и расстояния между словами, слишком короткую последнюю строку и т.д.

Вот как например здесь

ДП: Кратчайший путь

Кратчайший путь в графе между двумя точками.

- Идём «фронтом» от конечной.
- В каждой доступной вершине перезаписываем расстояние, если оно станет меньше, и помечаем, откуда пришли.
- Пока не дойдём до начальной.

ДП: Кратчайший путь

Кратчайший путь в графе между двумя точками.

- Идём «фронтом» от конечной.
- В каждой доступной вершине перезаписываем расстояние, если оно станет меньше, и помечаем, откуда пришли.
- Пока не дойдём до начальной.
- Каждый раз используется весь фронт (и только он)
- «Расстояние» в самом общем смысле. Например для ракеты взвешенная сумма расхода топлива, времени полёта и риска быть сбитой в данной точке.

Теорема о рекуррентных соотношениях

О чём речь?

- Задача на n элементах решается за $T(n)$ (не путать с параллельным программированием)
- Есть возможность разбить её на a подзадач, каждая для $\frac{n}{b}$ элементов, тогда $T(n) = aT(\frac{n}{b}) + f(n)$, где $f(n)$ — накладные расходы на создание подзадач и слияние результатов

Jon Louis Bentley, Dorothea Haken, and James B. Saxe. 1980. A general method for solving divide-and-conquer recurrences. SIGACT News 12, 3 (Fall 1980), 36–44. https://doi.org/10.1145/1008861.1008865

Вспоминаем O, Θ и Ω

И смотрим для этого русскую, английскую и даже немецкую (потому что немцы придумали) Википедии

Частные случаи (1)

Когда f «меньше» T, а именно

$f(n) = O(n^c)$, при этом $c < \log_b a$,

справедливо

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

Примеры:

- Алгоритм Штрассена: $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$, выполняется за время $O(n^{\log_2 7}) \approx O(n^{2,81})$
- Алгоритм Карацубы $T(n) = 3T(\lceil n/2 \rceil) + cn + d = 3T(\lceil n/2 \rceil) + O(n)$, выполняется за время $T(n) = \Theta(n^{\log_2 3})$

Частные случаи (2)

Когда f «сравнимо» с T, а именно

$\exists k \geq 0$: $f(n) = \Theta(n^c \log^k n)$, где $c = \log_b a$,

справедливо

$$T(n) = \Theta(n^c \log^{k+1} n)$$

Примеры:

- Двоичный поиск: $T(n) = T\left(\frac{n}{2}\right) + O(1)$, выполняется за время $O(\log n)$
- Сортировка слиянием: $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$, выполняется за время $O(n \log n)$

Частные случаи (3)

Когда f «больше» T, а именно

$f(n) = \Omega(n^c)$, где $c > \log_b a$ и $\exists N: af\left(\frac{n}{b}\right) \leq kf(n)$ для некоторой константы $k < 1$ и $n > N$,

справедливо

$$T(n) = \Theta(f(n))$$

Упражнения и вопросы

Упражнения и вопросы

Вопросы

- Назовите основные показатели эффективности распараллеливания
- Сформулируйте и обоснуйте закон Амдала
- Приведите примеры алгоритмов, допускающих эффективную параллельную и распределённую реализацию
- Что такое жадные и динамические алгоритмы
- Сформулируйте теорему о рекуррентных соотношениях и её частные случаи

Упражнения

- Найдите алгоритм, который может подходить под третий случай теоремы о рекуррентных соотношениях и условия, при которых это произойдёт