Поиск независимых вычислений для распараллеливании алгоритма

Метод независимых вычислений для распараллеливания алгоритмов основывается на разбиении задачи на независимые фрагменты, которые могут выполняться параллельно без необходимости обмена данными или синхронизации между процессами. Этот метод широко применяется в случаях, когда задачи могут быть решены отдельно друг от друга, и результаты этих подзадач не влияют на выполнение остальных.

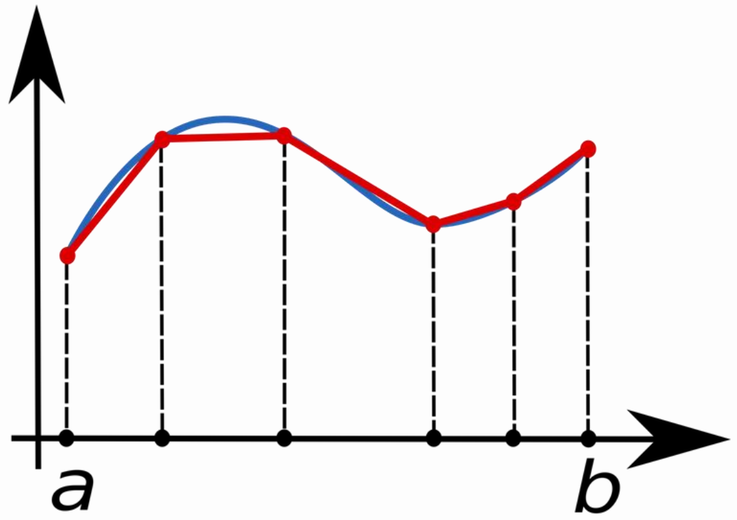
Рассмотрим, как можно распараллелить задачу численного интегрирования методом трапеций, используя метод независимых вычислений, при котором каждый процессор обрабатывает свою независимую часть отрезка интегрирования.

Введение в метод трапеций

Метод трапеций — это численный метод для оценки определённого интеграла функции f(x) на интервале [a,b]. Интервал разбивается на n равных частей, и для каждой части (маленького подынтервала) рассчитывается площадь трапеции, приближенно представляющей кривую функции на этом подынтервале. Суммирование площадей всех трапеций дает оценку интеграла.

$$area \approx \sum_{i=0}^{n} \frac{[f(a) + f(b)]}{2} \cdot \Delta x = \left[\frac{f(a) + f(b)}{2} + \sum_{i=0}^{n} f(a + i\Delta x) + f(a + (i+1)\Delta x) \right] \cdot \Delta x$$

Где, $\Delta x = (b - a)/n$.



Независимые вычисления и их параллелизация с помощью MPI

Алгоритм разрабатывается таким образом, чтобы каждая подзадача могла выполняться независимо, и, в случае метода трапеций, каждая трапеция может быть рассчитана независимо от других. Это позволяет нам делить интервал на подинтервалы и распределять их по нескольким процессам.

В параллельной реализации каждый процесс будет отвечать за расчет интеграла на своем подинтервале и затем отправлять результат процессу 0, который собирает и суммирует все результаты.

Разбор кода trapParallel_1.py

```
import numpy
import sys
from mpi4py import MPI
from mpi4py.MPI import ANY_SOURCE


comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

comm = MPI.COMM_WORLD — инициализируется коммуникатор, который позволяет всем процессам взаимодействовать.
rank и size — переменные для идентификации процесса и общего количества процессов. **rank** определяет номер текущего процесса, а **size** — количество задействованных процессов.

Чтение командных аргументов и определение функции интегрирования

```
a = float(sys.argv[1])
b = float(sys.argv[2])
n = int(sys.argv[3])

def f(x):
    return x * x
```

a, b, n — границы интегрирования и количество трапеций, передаваемые как аргументы командной строки.
f(x) — функция, которую мы интегрируем.

```
def integrateRange(a, b, n):
    integral = -(f(a) + f(b)) / 2.0
    for x in numpy.linspace(a, b, n + 1):
        integral = integral + f(x)
    integral = integral * (b - a) / n
    return integral
```

integrateRange(a, b, n) — функция для вычисления интеграла на интервале [a,b] с использованием метода трапеций. Сначала вычисляется сумма функции в начале и конце интервала, умноженная на -0.5, чтобы учесть коэффициенты в формуле метода трапеций.
Цикл for добавляет значения функции в каждой точке разбиения интервала, получаемой через **numpy.linspace.**
После суммирования всех значений вычисленная сумма умножается на шаг h для получения приближенного значения интеграла.

Определение локального интервала и расчет интеграла

```
h = (b - a) / n
local_n = n / size


local_a = a + rank * local_n * h
local_b = local_a + local_n * h
```

h — шаг разбиения интервала на трапеции.
local_n — количество трапеций для каждого процесса. Здесь предполагается, что nnn делится на количество процессов без остатка.
local_a и **local_b** — начало и конец подинтервала, обрабатываемого конкретным процессом. Для каждого процесса подинтервал рассчитывается независимо, что исключает необходимость передачи данных от процесса к процессу, улучшая производительность.

Расчет интеграла на каждом процессе

```
integral = numpy.zeros(1)
recv_buffer = numpy.zeros(1)


integral[0] = integrateRange(local_a, local_b, local_n)
```

integral и **recv_buffer** — массивы numpy, предназначенные для хранения результатов. MPI требует, чтобы передаваемые данные были массивами numpy.
Каждый процесс вычисляет свой локальный интеграл для подынтервала [local_a, local_b] и сохраняет результат в integral[0].

Передача данных между процессами

```
if rank == 0:
    total = integral[0]
    for i in range(1, size):
        comm.Recv(recv_buffer, ANY_SOURCE)
        total += recv_buffer[0]
else:
    comm.Send(integral)
```

rank == 0 — процесс 0, который собирает результаты.
Он получает данные от всех других процессов с помощью **comm.Recv(recv_buffer, ANY_SOURCE)**, где ANY_SOURCE позволяет получать сообщения от любого процесса.
Каждый полученный результат добавляется к total, где накапливается общий результат интеграла.
rank > 0 — процессы, которые отправляют свои результаты с помощью **comm.Send(integral)**.

Вывод результата

```
if comm.rank == 0:
    print "With n =", n, "trapezoids, our estimate of the integral from", a, "to", b, "is", total
```

Только процесс 0 выводит окончательный результат, который представляет собой суммарное приближение интеграла для функции f(x) = x^2 на интервале [a, b].

Запуск программы

Чтобы запустить программу на 4 процессах, используйте команду:

mpiexec -n 4 python trapParallel_1.py 0.0 1.0 10000

Заключение

Эта реализация демонстрирует классический пример метода независимых вычислений. Каждый процесс обрабатывает независимую часть данных, выполняя расчеты на своём подынтервале, что помогает значительно ускорить выполнение алгоритма, особенно при большом количестве трапеций.