# Отчет по лабораторной работе N4

Цель работы:

Познакомить студента с принципами параллельных вычислений. Составить несколько

программ в простейшими вычислительными действиями, чтобы освоить принципы

параллельных вычислений (когда одни алгоритмы зависят / не зависят от других).

Итог:

Последовательные вычисления:

Sequence evaluation

Time spent for 10000 calculations is 0.0012487 seconds

Time spent for 100000 calculations is 0.0879951 seconds

\*\* Process exited - Return Code: 0 \*\*

Параллельные вычисления через потоки:

Treads paralleling evaluation

Time spent for 10000 calculations is 2.19642 seconds

Time spent for 100000 calculations is 27.9474 seconds

\*\* Process exited - Return Code: 0 \*\*

Параллельные вычисления через процессоры:

Evaluation with parallel processes

Time spent for 10000 calculations is 0.0044449 seconds

Time spent for 100000 calculations is 0.0104925 seconds

\*\* Process exited - Return Code: 0 \*\*

# Анализ результатов параллельных вычислений

На основе предоставленных данных можно сделать следующие выводы:

## Сравнение производительности методов

| **Метод** | **10,000 вычислений** | **100,000 вычислений** |
| --- | --- | --- |
| Последовательный | 0.0012 сек | 0.088 сек |
| Потоки (threads) | 2.196 сек | 27.947 сек |
| Процессы (processes) | 0.0044 сек | 0.0105 сек |

## Ключевые выводы

1. **Самый производительный метод**: Последовательное выполнение
   * Причина: минимальные накладные расходы, нет затрат на создание и синхронизацию потоков/процессов
   * Для простых вычислений (как в примере) параллелизация часто неэффективна
2. **Почему потоки работают медленнее**:
   * **Накладные расходы**: Создание потоков требует времени
   * **Конкуренция за ресурсы**: Потоки используют общую память, требуется синхронизация
   * **Проблемы реализации**: В вашем коде future.get() блокирует выполнение, сводя на нет преимущества параллелизма
   * **Глобальная блокировка**: В C++ rand() не потокобезопасен и использует внутреннюю блокировку
3. **Процессы работают лучше потоков**:
   * Изолированная память - нет конкуренции за ресурсы
   * Лучшая утилизация многоядерных процессоров
   * Но все равно медленнее последовательного выполнения для этой задачи
4. **Почему процессы быстрее потоков**:
   * Нет contention на общих ресурсах
   * Лучше распределяются по ядрам процессора
   * В вашей реализации pipe работают эффективнее, чем синхронизация потоков

## Рекомендации по оптимизации

1. Для простых вычислений (как в примере) лучше использовать последовательное выполнение
2. Если нужно использовать параллелизм:
   * Для CPU-bound задач (тяжелые вычисления) - процессы
   * Для IO-bound задач (ожидание ввода/вывода) - потоки
   * Использовать пулы потоков/процессов вместо создания новых
3. Оптимизации для вашего кода:
   * Заменить rand() на более эффективный генератор
   * Упростить избыточные формулы
   * Для потоков - использовать std::async с launch::deferred для небольших задач

**Итог**: Для тривиальных вычислений параллелизация часто приносит больше вреда, чем пользы из-за накладных расходов. Параллельные методы становятся эффективными только для достаточно сложных задач, где выигрыш от распараллеливания перекрывает затраты на его организацию.