**Микроанализ производительности автоматизации учебных процессов связанных с системой учебных планов вуза.**

**П.С. Кондратьев**[[1]](#footnote-1)

Краткая аннотация: в работе представляются результаты экспериментов оценки быстродействия параллельных реализаций парсинга и записи полученных данных в базу данных, в среде бюджетного персонального компьютера.

**Введение**

Бывает такое то, что приходится сталкивались с различными XML-файлами, который нужно быстро проанализировать, чтобы получить важные значения.

Руководствуясь идеей о том, что, какие методы распараллеливания существуют было принято решение проанализировать производительность автоматизации учебных процессов связанных с системой учебных планов вуза.

Результатом работы будет являться программа, выполняющие парсинг данных из xml файлов с последующим заполнением схем в базе данных, по данным из файла.

Для оценки скорости вычисления, проверим работу программы на различном количестве входных данных (10 файлов, 50 файлов, 100 файлов и т.д).

**Планирование экспериментов (получаем максимальный эффект от распараллеливания)**

Для объекта вычисления загрузки данных в базу данных, был получен набор учебных планов, состоящий из 2-ух файлов, прошлых лет, с которыми и производилась оценка затрат времени и составление таблиц и парсинга данных из plx(xml) файлов в бд.

В задаче экспериментирования с процедурами параллельного вычисления имеется два критериальных параметра: время вычисления и степень ускорения, определяемая отношением времени последовательной обработки ко времени параллельной обработки.

Значения критериальных параметров зависят от двух факторов: размера рабочей нагрузки на процедуру и количества параллельных ветвей.

Для эксперимента, был написан парсер, на языке Python, xml(plx) файлов, который записывает данные из таблиц (тэгов) в бд.

Стоит учесть, то что файлы бывают разные по размеру и это будет сказывается на скорости обработки, поэтому было принято решения проверить сначала на более простых файлах, после чего на более тяжеловесных файлах.

**Проведение экспериментов и анализ результатов**

Для эксперимента было решено сделать тестовую нагрузку из предложенных 2-уз файлов. Был написан маленький скрипт на Python, который брал эти файлы и создавал копии данных файлов.

Рабочая нагрузка была определена по 150 копий на файл.

Были реализованы основные этапы алгоритма для реализации эксперимента:

Для параллелизма на Python была использована библиотека multiprocessing, для замера времени time (Функция time() модуля time вернет время в секундах с начала эпохи как число с плавающей запятой).

Стоит отметить то, что в multiprocessing процессы порождаются путём создания объекта Process и последующего вызова его метода start().

Сама программа работает следующим образом:

1. Проверка файлов на существования их в бд.
2. Вычисление количества циклов вычисления на 1 ядре (разбиваем на ровное количество файлов на каждом потоке).
3. Запускаем ядра с высчитанным количеством операций на нем.
4. В каждом потоке собираем данные с xml (plx) файла и составляем запрос на запись в базу данных.

**Примечание!** Каждый процессор поддерживает определенное количество потоков на ядро, заложенное производителем, при которых он работает оптимально быстро. Нельзя создавать безгранично много потоков. При увеличении числа потоков на величину, большую, чем заложил производитель, программа будет выполняться дольше или вообще поведет себя непредсказуемым образом (вплоть до зависания).

Ниже представлены таблицы результатов времени вычисления работы программы от увеличения количества файлов.

Первый тест произведен с файлом 09.03.01.02-1234-2016.plx с размером **829КБ** (табл. 1).

Таблица 1. Результаты замеров времени (c) вычисления при варьировании числа потоков и гранулярности задачи (кол-во файлов)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потоки** | **10** | **25** | **50** | **100** | **150** |
| **1** | 7,267730 | 18,55141 | 36,713355 | 82,005270 | 110,67462 |
| **2** | 5,574509 | 11,83215 | 27,218761 | 43,925798 | 65,417901 |
| **3** | 4,226635 | 9,436400 | 18,448173 | 35,615691 | 54,696882 |
| **4** | 3,630880 | 9,678485 | 15,7115521 | 31,4356100 | 44,7100415 |

Рис. 1. Графики коэффициентов ускорения при варьировании числа потоков

Второй тест будет произведен с файлом 09.03.01\_02-2019.plx с размером **3082КБ** (табл. 2).

Таблица 2. Результаты замеров времени (c) вычисления при варьировании числа потоков и гранулярности задачи (кол-во файлов)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потоки** | **10** | **25** | **50** | **100** | **150** |
| **1** | 30,39569 | 76,04511 | 152,6322 | 312,35483 | 449,47826 |
| **2** | 17,99670 | 43,44480 | 85,63571 | 181,74103 | 272,01687 |
| **3** | 16,01764 | 38,79354 | 80,55757 | 152,78097 | 235,82868 |
| **4** | 13,222541 | 35,3233423 | 62,353862 | 131,69074 | 198,1465 |

Рис. 2. Графики коэффициентов ускорения при варьировании числа потоков

Из полученных результатов видно, что время, затраченное на сбор данных из файла и последующего составления запроса на обновление схемы бд, сильно зависит от размера файла. Чем меньше файл, тем быстрее будет работать парсер и быстрее данные попадут в бд.

Таблица 3. Основные технические характеристики ПК

|  |  |
| --- | --- |
| Компонент | Характеристики |
| Процессор (CPU) | I5 – 5700U 2,5 – 2,5 ГГц, 2 ядра, 4 потока |
| Оперативная память (RAM) | DDR4 8 ГБ + 16G |
| Компонент | Характеристики |
| Графический контроллер | Intel HD Graphics 620  NVIDIA GeForce 940MX |
| Объем памяти видеокарты | 1 Гб  2 Гб |

Заключение

В ходе исследовательской работы были выявлены результаты экспериментов оценки быстродействия параллельных реализаций парсинга и записи полученных данных в базу данных, в среде бюджетного персонального компьютера.

Из проведенных экспериментов и анализов результатов были выявлены будущие улучшения данной программы:

1. Сделать декларативную проверку схемы бд и содержащихся таблиц в файле на наличия присутствия.

Первый вариант программы делал следующие – вовремя парсинга проверять наличие таблиц и столбцов таблицы в бд, но это ведёт к сильной нагрузке бд из-за множества запросов к бд. Поэтому продуктивнее заранее получать схему файла и сверять ее с бд, а после записывать данные.

1. Делать 1 запрос на 1 файл. Подготовить все данные по файлу и отправить одним запросом.

Программа работает сейчас по принципу, если есть файлы на загрузку которых нету в бд грузим их, после чего происходит разбивка списка файлов на ядра компьютера. Там уже по получившемуся списку происходит парсинг, выявления таблиц, и сбор всех признаков данной таблицы после чего происходит запись в бд (т.е. загрузку производится по 1 таблице из файла)

Список литературы

1. https://codecamp.ru/blog/python-manipulating-xml/ - Работа с XML из Python (Дата обращения: 13.06.2021).
2. <http://top50.supercomputers.ru> - TOP50 суперкомпьютеров процессорах (Дата обращения: 17.06.2021).
3. https://docs-python.ru/tutorial/mnogopotochnost-python/- Многопоточность в Python (Дата обращения: 14.06.2021).
4. https://digitology.tech/docs/python\_3/library/xml.etree.elementtree.html - Инструменты обработки структурированной разметки (Дата обращения: 12.06.2021).

1. 432027, Ульяновск, ул. Северный Венец, 32, УлГТУ, e-mail: Pablo.osamu@yandex.ru [↑](#footnote-ref-1)