слайд 1.

В современном мире существует множество задач, в которых требуется параллельная обработка данных. Ярким примером могут послужить вычислительные задачи большого масштаба. Для этих целей в CERN была разработана библиотека Colt Parallel, которая реализует многопоточные алгоритмы для работы с линейной алгеброй, сигналами в дискретной форме и так далее.

Другой пример это системы сбора и обработки данных, которые используются на экспериментальных установках.

слайд 2

Типичная система обработки и сбора данных представляет собой 2 больших модуля. Один из них отвечает за сбор данных в режиме жесткого реального времени. Он состоит из набора устройств для измерения показаний, которые способны аккумулировать свои данные в локальную память. Затем эта информация передается по сети второму модулю. Он отвечает за пред обработку полученного набора данных и запись результатов на диск, либо в базу данных. А для увеличения пропускной способности нужно параллельно обрабатывать данные, поступающие из каналов связи.

слайд 3

Второй модуль можно реализовать как систему мягкого реального времени. Для этих целей допустимо использовать язык программирования Java, поскольку он упрощает разработку, так как обладает автоматической системой управления памятью, контролем ошибок при исполнении и так далее.

слайд 4

Традиционно, параллелизм реализуется внутри операционной системы с помощью механизма потоков. Потоки абстрагируют независимо работающие задачи и представляют доступ к ядрам процессора: аппаратному механизму параллелизма. Однако, у потоков есть минусы. Они предлагают достаточно "тяжеловесный" механизм: их создание и переключение несет в себе крупные накладные расходы. Избежать накладных расходов на использование потоков можно, применяя вместо них сопрограммы.

слайд 5

<пересказ>

слайд 6

Целью же моей работы является изучение применимости сопрограмм как альтернативу потокам в программах Java. Для этого были поставлены следующие задачи: <слайд>

слайд 7

Первым шагом работы стал анализ существующих реализаций. Для этого был создан набор тестов производительности различных реализаций сопрограмм и потоков. Анализ предметной области показал, что не существует подходящего набора тестов, который бы измерял 2 интересующих параметра:

1) Скорость переключения

2) потребление физической памяти.

Разработанные тесты можно будет использовать для оценки эффективности сопрограмм в системах сбора данных.

Тесты доступны по ссылке на слайде.

слайд 8

Следующим шагом работы стала реализация прототипа сопрограмм. Для его работы минимального прототипа необходимо переключение сопрограмм и сборка мусора объектов, чьи ссылки лежат на стеках сопрограмм. к переключению контекста есть несколько подходов. Для реализации в HuaweiJDK использовался механизм как в языке Go, поскольку он показался более эффективным.

слайд 9

Наконец были измерены скорости переключения сопрограмм в управляемых средах HuaweiJDK, OpenJDK и языка Go. Как видно на слайде, HuaweiJDK выигрывает в скорости переключения у OpenJDK на всех измерениях, но проигрывает языку Go.

слайд 10

Так же была измерена скорость переключения потоков. Как видно из слайда, сопрограммы имеют лучшую скорость переключения, чем потоки на всем диапазоне измерений.

слайд 11

По сравнению с другими реализациями, сопрограммы в HuaweiJDK имеют неплохой результат по потреблению памяти: они обходят OpenJDK, но проигрывают сопрограммам в Go. Измерение потребления физической памяти потоков показало, что сопрограммы имеют меньшее потребление памяти, чем потоки при любом их числе.

слайд 12

Так же были проведены измерения производительности сопрограмм и потоков. Для этих целей была модифицирована библиотека Colt Parallel c использованием сопрограмм. Эта библиотека используется для многопоточной обработки данных. В ней потоки были подменены практически один к одному. Результаты измерений представлены на слайде <пересказ>

слайд 13

После проведения замеров можно сформулировать ключевые отличия потоков от сопрограмм. <пересказ>

<остальные слайды пересказываются>