УДК 004.054

И. В. ГРУЗДО, С. В. РОССОХА, И. В. ШОСТАК

Национальный аэрокосмический университет им. Н. Е. Жуковского «ХАИ», Украина

ПРИМЕНЕНИЕ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ К РЕШЕНИЮ ЗАДАЧИ УСТАНОВЛЕНИЯ АВТОРСТВА ТЕКСТОВ

В работе проведен обзор и критический анализ существующих типов распараллеливания программ. Дано общее описание информационной технологии анализа и оценивания письменных учебных работ с учетом наличия в них текстовых заимствований. Выделены основные проблемы распараллеливания вычислений для решения задачи установления авторства текстов. Выполнена постановка формальной задачи распараллеливания процессов при определении авторства текстов. Показана необходимость усовершенствования математического и методического обеспечения систем поиска плагиата, направленного на повышение эффективности таких систем.

Ключевые слова: *параллельные вычисления, плагиат, распараллеливание, преобразование, распределенная память, степени параллелизма.*

Введение

В настоящее время использование современных высокопроизводительных вычислительных средств параллельного действия для обработки в оперативном режиме большого количества входной информации определило распараллеливание алгоритмов как одно из основных направлений на современном этапе развития науки и техники. С одной стороны, важной задачей является автоматическое распараллеливание алгоритмов обработки информации. От ее решения во многом зависит успех внедрения многопроцессорных систем в практику вычислений. С другой стороны, при разработке высокопроизводительных вычислительных средств параллельного действия требуется определить основные свойства внутреннего параллелизма алгоритмов и закономерности, которым подчиняется структура параллельных процессов обработки информации.

Особую актуальность заявленная проблема приобретает при автоматизации процессов, связанных с текстовым анализом письменных учебных работ (ПУР) на предмет выявления текстовых заимствований, в частности, студентов технических вузов.

В ходе исследования выявления заимствований в работах студентов технических ВУЗов [1, 2] был разработан инструментарий для контроля качества выполнения академических работ, а именно, специальная информационная технология (ИТ) анализа и оценивания письменных учебных работ с учетом наличия в них текстовых заимствований. При этом автоматизация анализа ПУР была осуществлена с использованием ПС «Plagiarizm», который представляет собой исследовательский прототип, функционирующий на основе данной технологии.

Использование ПС «Plagiarizm» в учебном процессе дает следующий эффект: накопление опыта преподавателей и нормоконтролеров, а также для поддержки принятия решений ими по оцениванию письменных учебных работ. Кроме того, следует отметить, что применение ПС «Plagiarizm» на практике предоставляет возможность оценивать эффективность результатов, достигнутых в ходе исследования.

Для того чтобы повысить эффективность от использования ПС «Plagiarizm» необходимо выполнить преобразование последовательно выполняемых процессов программы в несколько параллельно взаимодействующих процессов. При этом необходимо, чтобы машина автоматически выполняла однозначное определение отдельных файлов для каждого обнаруженного позаимствованного объекта, а также связывала полученную информацию по каждой проанализированной ПУР, с соответствующим файлом источником, и производила анализ полученных данных.

Цель данной статьи состоит в критическом обзоре существующих архитектур многопоточных приложений, а также необходимо, в зависимости от существующих типов распараллеливания, выполнить преобразовании ПС «Plagiarizm» в параллельную программу.

Общее описание информационной технологии анализа и оценивания письменных учебных работ с учетом наличия в них текстовых заимствований

В качестве примера использования специальной ИТ анализа и оценивания письменных учебных работ с учетом наличия в них текстовых заимствований был

рассмотрен процесс анализа ПУР в техническом университете, а именно, Национальном аэрокосмическом университете им. Н. Е. Жуковского «ХАИ». При этом автоматизация анализа ПУР была осуществлена с использованием ПС «Plagiarizm», который представляет собой исследовательский прототип, функционирующий на основе специально разработанной технологии. Использование ПС «Plagiarizm» в учебном процессе дает следующий эффект: накопление опыта преподавателей и нормоконтролеров, а также для поддержки принятия решений ими по оцениванию ПУР. Кроме того, следует отметить, что применение ПС «Plagiarizm» на практике предоставляет возможность оценивать эффективность результатов, достигнутых в ходе исследования.

Преподавателю в процессе анализа и оценивания ПУР необходимо принимать ответственные решения, опираясь на свой опыт и интуицию. Для этого ему необходимо владеть информацией, поступающей из различных структурных единиц учебного заведения: ректората факультетов, кафедр и т.д., а также владеть информацией о текущем состоянии БД ПУР и др.

Задача выявления плагиата в образовательном процессе сводится к следующим этапам: классификация видов плагиата, диагностика причин плагиата, создание технологий выявления плагиата, обоснование критериев плагиата, организация контроля и ответственности за плагиат и мер по его предотвращению.

Технология анализа ПУР с использованием ПС «Plagiarizm» предполагает использование документов, представленных в форматах TXT, DOC, DOCX и RTF. И включает в себя следующие этапы:

1. Загрузка пользователем ПУР для анализа.

2. Настройка дополнительных условий поиска, таких как метод анализа и релевантных условий.

3. Автоматическая проверка документа на наличие плагиата.

4. Формирование отчета по результатам поиска плагиата в анализируемой работе. В случае если объем заимствований, найденных в документе, превышает пороговое значение.

5. Формирование системой решения о наличии в анализируемом тексте плагиата с указанием источников заимствований.

6. Проведение, в случае необходимости, анализа при измененных параметрах.

7. Распечатка отчета по результатам анализа.

Основное назначение ПС «Plagiarizm» состоит в автоматизации процесса анализа в студенческих работах на предмет выявления заимствований в тексте, что позволит повысить качество проверки ПУР за счет упрощения деятельности экспертов при анализе результатов выполнения студентами индивидуальных заданий, также ускорить данный процесс и снизить при этом риск невыявления заимствований. Это особенно актуально в условиях возрастающих требований к качеству подготовки специалиста, необходимостью частого обновления требований к студенческим работам и повышения качества учебного процесса. Кроме того, внедрение системы послужит «стимулом» для студентов к самостоятельному написанию текстов, а не созданию их, например, путем компиляции из найденных в Интернете различных документов, касающихся заданной тематики.

На основе выделения минимальных необходимых функций системы и их составляющих, а также анализа требуемой функциональности к системе, была разработана функциональная схема ПС «Plagiarizm».

Основные методы, лежащие в основе функционирования системы представлены в таблице 1.

Разработанный прототип ПС «Plagiarism» предназначен для:

- использования преподавателем в учебно-методическом отделе университета (УМО);

- использования преподавателями и нормоконтролерами для получения информации об индивидуальности выполнения студентом выполненной работы; заведующими кафедрами и деканами факультетов;

- выполнения аналитических расчетов по работе студента;

- формирование отчета о степени самостоятельности выполнения студентом определенного вида работы, что обеспечивает своевременное подведение итогов контрольно-оценочной деятельности преподавателя.

Хотя результаты экспериментальных исследований оценки и анализа письменных учебных работ с использованием специальных моделей и методов, разработанных в ПС «Plagiarizm», показали результат выше, чем при использовании аналогов, Антиплагиат и WCopyfind, необходимо снизить время необходимое для анализа для более эффективного функционирования.

Существующие типы распараллеливания

Рассмотрим существующие типы распараллеливания приложений и выполним их анализ.

Наибольшее распространение получила классификация вычислительных систем, предложенная в 1966 г. профессором Стенфордского университета М. Д. Флином [3] - классификация Флина.

Эта классификация охватывает только два классификационных признака – тип потока команд и тип потока данных.

В соответствии с данной классификацией все

разнообразие архитектур ЭВМ сводится к четырем классам [4]:

1) вычислительная система с одиночным потоком команд и одиночным потоком данных (ОКОД, SISD, Single Instruction stream over a Single Data stream);

2) вычислительная система с одиночным потоком команд и множественным потоком данных (ОКМД, SIMD, Single Instruction, Multiple Data);

3) вычислительная система с множественным потоком команд и одиночным потоком данных (МКОД, MISD, Multiple Instruction Single Data);

4) вычислительная система с множественным потоком команд и множественным потоком данных (MIMD, Multiple Instruction Multiple Data).

Данная схема классификации является самой применяемой при начальной характеристике того или иного компьютера и в настоящее время. К недостаткам следует отнести то, что некоторые архитектуры, например dataflow и векторно-конвейерные машины, четко не вписываются в данную классификацию, а также чрезмерная заполненность класса MIMD. Также следует заметить, что существующие классификации [4] в основном рассматривают проблемы аппаратного обеспечения, а не программной реализации.

Необходимо более точно систематизировать архитектуры, которые по Флинну попадают в один класс, но совершенно различны по числу процессоров, природе и топологии связи между ними, по способу организации памяти и, конечно же, по технологии программирования, а также по уровням распараллеливания. Для этого необходимо выполнить дополнительный анализ с учетом составляющих данных.

В ходе изучения проблемы распараллеливания [4-7] с учетом составляющих данных были выделены следующие основные типы распараллеливания:

1) По средствам программирования

1.1 Системы с общей памятью (SMP).

1.2 Системы с неоднородным доступом (NUMA).

1.3 Системы с распределенной памятью (MPP).

1.4 Смешанные системы.

2) По используемым архитектурам

2.1 Многопроцессные приложения с автономными процессами.

2.2 Многопроцессные приложения, взаимодействующие через трубы, сокеты и очереди System V IPC.

2.3 Многопроцессные приложения, взаимодействующие через разделяемую память.

2.4 Собственно многопоточные приложения.

2.5 Событийно-ориентированные приложения.

2.6 Гибридные архитектуры.

3) По уровнями распараллеливания

Таблица 1

Основные методы функционирования ПС «Plagiarizm»

№ п/п	Название	Суть метода
1.	WinWordDocument.Algoritm_Srawnenia	Вычисляет процент совпадения двух строк с учётом наличия ссылок на первоисточники в проверяемой строке.
2.	WinWordDocument.Highlight	Подсвечивает результаты сравнения.
3.	WinWordDocument.Nchot_Analize_String	Производит анализ строк по нечеткому алгоритму поиска.
4.	WinWordDocument.TextStrip	Формирует содержимое документа Word, очищенное от стандартных структурных элементов Microsoft Office Word 2007, и от элементов, которые допустимо игнорировать при анализе (с учетом дополнительных настроек программы), заданны регулярным выражением.
5.	WinWordDocument.WinWordDocument	Создает объект Microsoft Office Word 2007 на основе файла MSWord.
6.	TSettings.Load	Объект для загрузки настроек в INI-файл, заполняет поля класса из объекта настроек.
7.	TSettings. Save	Объект для сохранения настроек в INI-файл, сохраняет все измененные настройки настроек.
8.	TSettings.TSettings	Создать настройки по умолчанию, если INI-файл был поврежден или не создан (при первом запуске программы).
9.	TButton_Analyse	Метод, определяющий, текущие вкладки, список файлов или файл.
10.	Form1. GetFilesFromDir	Формирует массив файлов, находящихся в данной директории и её поддиректориях, и игнорирует файлы, несоответствующие запросу анализа.
11.	DateTime	Отражает дату и время в текущем объекте.

3.1 Распараллеливание на уровне задач.

3.2 Уровень параллелизма данных.

3.3 Уровень распараллеливания алгоритмов.

3.4 Параллелизм на уровне инструкций.

4) По типу строения оперативной памяти

4.1 Системы разделяются на системы с общей (разделяемой) памятью.

4.2 Системы с распределенной памятью.

4.3 Системы с физически распределенной, а логически общедоступной памятью (гибридные системы).

Рассмотрим более подробно каждый тип.

По средствам программирования:

Системы с общей памятью (SMP- symmetric multiprocessing*)* – представляют собой набор параллельно работающих процессоров, имеющих доступ к общей для всех процессоров памяти, причем скорость доступа к памяти одинакова для всех процессоров.

SMP-системы позволяют работать с любой задачей, независимо от того, где в памяти хранятся данные для этой задачи. Чем больше процессоров, тем больше нагрузка на общую шину, тем дольше должен ждать каждый процессор, пока освободится шина, чтобы обратиться к памяти.

Системы с неоднородным доступом (NUMA-Non-Uniform Memory Access*)* – набор блоков, содержащих несколько процессоров и общую для них память. При этом допускается обращение любого процессора к удаленной памяти, т.е. памяти другого блока, но обращение происходит медленнее, чем обращение к локальной памяти. При программировании применяется, например, система Shmem.

Память физически распределена по различным частям системы, но логически она является общей, так, что пользователь видит единое адресное пространство. Система построена из однородных базовых модулей, состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей. При этом доступ к локальной памяти осуществляется в несколько раз быстрее, чем к удаленной.

Системы с распределенной памятью (MPP-massive parallel processing) – набор узлов, состоящих из процессора и памяти, и коммутационной среды для связи между узлами. Каждый процессор имеет непосредственный доступ только к своей локальной памяти. При программировании применяется модель передачи сообщений, используются библиотеки MPI, PVM и др. В системах этого типа на каждом вычислительном узле функционируют собственные копии операционной системы, под управлением которых выполняются независимые программы. Это могут быть как действительно независимые программы, так и параллельные ветви одной программы. В этом случае единственно возможным механизмом взаимодействия между ними является механизм передачи сообщений.

Главная особенность такой архитектуры состоит в том, что память физически разделена. В этом случае система строится из отдельных модулей, содержащих процессор, локальный банк операционной памяти, коммуникационные процессоры (рутеры) или сетевые адаптеры, иногда – жесткие диски и/или другие устройства ввода/вывода. По сути, такие модули представляют собой полнофункциональные компьютеры.

Смешанные системы – набор SMP-узлов, соединенных коммутационной средой. Применяется комбинация моделей передачи сообщений и общей памяти. Часто также используется чистая модель передачи сообщений, тогда каждый процессор в SMP-узле трактуется как независимый узел со своей локальной памятью.

По используемым архитектурам [3]:

Многопроцессные приложения с автономными процессами (МПсАП) представляют собой простые в реализации приложения, где для каждой пользовательской сессии или для каждого запроса создается свой процесс. Процесс обрабатывает запрос или несколько запросов, после чего сам завершается.

В Unix-системах предпринимается целый комплекс мер для того, чтобы сделать создание процесса и запуск новой программы в процессе как можно более дешевыми операциями. При этом следует обратить внимание, что создание нити в рамках существующего процесса всегда будет «дешевле», чем создание нового процесса.

Многопроцессные приложения, взаимодействующие через сокеты, трубы и очереди сообщений System V IPC. Перечисленные средства IPC (Interprocess communication) относятся к так называемым средствам гармонического межпроцессного взаимодействия. Они позволяют организовать взаимодействие процессов и потоков без использования разделяемой памяти. Теоретики программирования очень любят эту архитектуру, потому что она практически исключает многие варианты ошибок соревнования.

Многопроцессные приложения, взаимодействующие через разделяемую память. В качестве разделяемой памяти может использоваться разделяемая память System V IPC и отображение файлов на память. Для синхронизации доступа можно использовать семафоры System V IPC, мутексы и семафоры POSIX, при отображении файлов на память – захват участков файла.

Фактически, данная архитектура сочетает недостатки многопроцессных и собственно многопоточных приложений. Тем не менее, ряд популярных приложений, разработанных в 80е и начале 90х, до того, как в Unix были стандартизованы многопоточные API, используют эту архитектуру. Это многие серверы баз данных, как коммерческие (Oracle, DB2, Lotus Domino), так и свободно распространяемые, современные версии Sendmail и некоторые другие почтовые серверы.

Собственно многопоточные приложения. Потоки или нити приложения исполняются в пределах одного процесса. Все адресное пространство процесса разделяется между потоками. На первый взгляд кажется, что это позволяет организовать взаимодействие между потоками вообще без каких-либо специальных API. В действительности, это не так – если несколько потоков работает с разделяемой структурой данных или системным ресурсом, и хотя бы один из потоков модифицирует эту структуру, то в некоторые моменты времени данные будут несогласованными.

Поэтому потоки должны использовать специальные средства для организации взаимодействия.

В целом можно сказать, что многопоточные приложения имеют почти те же преимущества и недостатки, что и многопроцессные приложения, использующие разделяемую память. Однако стоимость исполнения многопоточного приложения ниже, а разработка такого приложения в некоторых отношениях проще, чем приложения, основанного на разделяемой памяти. Поэтому в последние годы многопоточные приложения становятся все более и более популярны.

Событийно-ориентированные архитектуры. Программа реализуется в виде набора функций или методов-обработчиков, которые вызываются в ответ на возникновение событий. События могут быть как внешними по отношению к программе (например, прибытие порции данных из сетевого соединения, сработка таймера или нажатие клавиши пользователем), так и внутренними. Внутренние события используются для взаимодействия между разными обработчиками.

Как правило, событийно-ориентированная архитектура предполагает наличие специального модуля, называемого диспетчером или менеджером событий. Этот модуль тем или иным образом собирает информацию о возникших событиях, организует сообщения о событиях и обработчиках в очереди в соответствии с теми или иными правилами и вызывает обработчики.

Главная сложность при разработке событийно-ориентированного приложения – это гарантировать, что все обработчики событий будут завершаться достаточно быстро. Как правило, данное утверждение означает, что обработчики не должны вызывать блокирующихся системных вызовов. Если требование соблюдено, то событийно-ориентированная архитектура позволяет оперировать преимуществами многозадачности и многопоточности.

При помощи данного типа архитектуры реализуются программы с графическим пользовательским интерфейсом, сетевые серверы и некоторые клиентские сетевые программы, приложения реального времени, игры и др. Ядра большинства современных операционных систем, таких, как Windows, Linux и BSD Unix, также имеют событийно-ориентированную архитектуру.

Гибридные архитектуры. Гибридные архитектуры отличаются большим разнообразием. Они позволяют сочетать преимущества, характерные для различных типов простых архитектур. Так, разработчик интерактивного приложения может реализовать пользовательский интерфейс в рамках событийно-ориентированной архитектуры, а для сложных вычислений (например, для переразбиения текста на страницы) запускать фоновые нити. Однако важно понимать, что очень часто гибридные архитектуры сочетают не только преимущества, но и недостатки используемых базовых архитектур. Поэтому создание приложения с гибридной архитектурой предъявляет высокие требования к квалификации архитектора приложения и большинства разработчиков.

По уровням распараллеливания

*Распараллеливание на уровне задач. Р*аспараллеливание на этом уровне является самым простым и самым эффективным способом. Следует учитывать, что данный тип распараллеливания целесообразно выполнять в тех случаях, когда решаемая задача состоит из независимых подзадач, каждую из которых можно решить отдельно. Примером распараллеливание на уровне задач может быть операционная система, запуская на многоядерной машине программы на разных ядрах. Если первая программа показывает нам фильм, а вторая является файлообменным клиентом, то операционная система без существенных затрат сможет организовать их параллельную работу.

Данный вид распараллеливания прост и в ряде случаев весьма эффективен, но если работа выполняется при помощи однородной задачи, то данный вид распараллеливания не применим. Операционная система никак не может ускорить программу, использующую только один процессор, сколько бы ядер ни было бы при этом доступно. Чтобы распараллелить однородные задачи, нужно спуститься на уровень ниже. Выполнение распараллеливания по данному типу возможно в тех случаях, когда решае-

мая задача естественным образом состоит из независимых подзадач, каждую из которых можно решить отдельно.

Уровень параллелизма данных. Заключается в применении одной и той же операции к множеству элементов данных. На практике используется, например, в современных архиваторах, СУБД, сетевых хранилищах и т.п. Следует заметить, что все данные разбиваются на блоки, которые затем по одному и тому же алгоритму обрабатываются на нескольких узлах.

Данный вид параллелизма широко используется при решении задач численного моделирования. Счетная область представлена в виде ячеек, описывающих состояние среды в соответствующих точках пространства — давление, плотность, процентное соотношение газов, температура и так далее. Количество таких ячеек может быть огромным — миллионы и миллиарды. Каждая из этих ячеек должна быть обработана одним и тем же способом. При этом счетная область разбивается на геометрические объекты, например параллелепипеды, и ячейки, вошедшие в эту область, отдаются на обработку определенному ядру (геометрический параллелизм).

В случае задач моделирования, геометрический параллелизм может показаться похожим на распараллеливание на уровне задач, он является более сложным в реализации, что влечет за собой необходимость передавать данные получаемые на границах геометрических областей другим ядрам и требует дополнительного анализа и создания алгоритма балансировки.

Уровень распараллеливания алгоритмов. К данному типу относятся алгоритмы параллельной сортировки, умножение матриц, решение системы линейных уравнений. Удобно использовать такую технологию параллельного программирования на уровне абстракций, а именно технологию OpenMP.

OpenMP (Open Multi-Processing) — это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах. В OpenMP используется модель параллельного выполнения «ветвление-слияние». Программа OpenMP начинается как единственный поток выполнения, называемый начальным потоком. Когда поток встречает параллельную конструкцию, он создает новую группу потоков, состоящую из себя и некоторого числа дополнительных потоков, и становится главным в новой группе. Все члены новой группы (включая главный поток) выполняют код внутри параллельной конструкции. В конце параллельной конструкции имеется неявный барьер. После параллельной конструкции выполнение пользовательского кода продолжает только главный поток. В параллельный регион могут быть вложены другие параллельные регионы.

За счет идеи «инкрементального распараллеливания» OpenMP идеально подходит для разработчиков, желающих быстро распараллелить свои вычислительные программы с большими параллельными циклами. Разработчик не создает новую параллельную программу, а просто последовательно добавляет в текст последовательной программы OpenMP-директивы.

Задача реализации параллельных алгоритмов достаточно сложна и поэтому существует достаточно большое количество библиотек параллельных алгоритмов, позволяющих строить программы как из кубиков, не вдаваясь в устройство реализаций параллельной обработки данных.

Параллелизм на уровне инструкций. Программа представляет собой поток инструкций, выполняемых процессором. Следует отметить, что можно изменить порядок этих инструкций, распределить их по группам, которые будут выполняться параллельно, без изменения результата работы всей программы (параллелизмом на уровне инструкций). Для реализации данного вида параллелизма в микропроцессорах используется несколько конвейеров команд, таких технологий как: предсказание команд, переименование регистров.

Программист редко заглядывает на этот уровень. Да и в этом нет смысла. Работу по расположению команд в наиболее удобной последовательности для процессора выполняет компилятор. Интерес этот уровень распараллеливания может представлять только для узкой группы специалистов, выжимающих все возможности из SSEx или разработчиков компиляторов.

Работа на уровне процессора даёт ощутимый прирост производительности в специально оптимизированных приложениях. Этот вид параллельности иногда выделяют в еще более глубокий уровень распараллеливания – параллелизм на уровне битов.

По типу строения оперативной памяти

Распределенная общая память (DSM - Distributed Shared Memory). Традиционно DSM - виртуальное адресное пространство, разделяемое всеми узлами (процессорами) распределенной системы, базируется на модели передачи сообщений, в которой данные передаются от процессора к процессору в виде сообщений.

Программы получают доступ к данным в DSM примерно так же, как они работают с данными в виртуальной памяти традиционных ЭВМ. В системах с DSM данные перемещаются между локальными памятями разных компьютеров аналогично тому, как они перемещаются между оперативной и внешней памятью одного компьютера.

В многопроцессорной системе с общей памятью один процессор осуществляет запись в конкретную ячейку, а другой процессор производит считывание из этой ячейки памяти. Чтобы обеспечить согласованность данных и синхронизацию процессов, обмен часто реализуется по принципу взаимно исключающего доступа к общей памяти методом "почтового ящика".

Однако накладные расходы на обмен в этих машинах имеются и определяются конфликтами шин, памяти и процессоров. Чем больше процессоров добавляется в систему, тем больше процессов соперничают при использовании одних и тех же данных и шины, что приводит к состоянию «насыщения». Модель системы с общей памятью очень удобна для программирования и иногда рассматривается как высокоуровневое средство оценки влияния обмена на работу системы, даже если основная система в действительности реализована с применением локальной памяти и принципа передачи сообщений.

С распределенной памятью. С ростом числа процессоров просто невозможно обойти необходимость реализации модели распределенной памяти с высокоскоростной сетью для связи процессоров.

Распределение памяти между отдельными узлами системы имеет два основных преимущества. Во-первых, это увеличение полосы пропускания памяти, поскольку большинство обращений могут выполняться параллельно к локальной памяти в каждом узле. Во-вторых, это уменьшение задержки обращения (время доступа) к локальной памяти. Эти два преимущества еще больше сокращают количество процессоров, для которых архитектура с распределенной памятью имеет смысл.

Таким образом, отличия разных машин с распределенной памятью определяются моделью памяти и механизмом обмена. Исторически машины с распределенной памятью первоначально были построены с использованием механизма передачи сообщений, поскольку это было очевидно проще и многие разработчики и исследователи не верили, что единое адресное пространство можно построить и в машинах с распределенной памятью.

В последнее время модели обмена с общей памятью начали поддерживаться практически в каждой разработанной машине (характерным примером могут служить системы с симметричной мультипроцессорной обработкой). Хотя машины с централизованной общей памятью, построенные на базе общей шины все еще доминируют в терминах размера компьютерного рынка, долговременные технические тенденции направлены на использование преимуществ распределенной памяти даже в машинах умеренного размера.

Проблема распараллеливания вычислений для решения задачи установления авторства текстов

В ходе проведенного анализа, было установлено, что существует несколько обобщающих особенностей при решении проблемы распараллеливания вычислений для задачи установления авторства текстов:

1. Основная цель распараллеливания программы – это улучшение качества программы и, как следствие, уменьшение времени необходимого на анализ данных.

2. Распараллеливание может производиться в динамическом процессе, т.е. независимые элементы программы могут выявляться в процессе вычисления и назначаться на обработку по мере освобождения требуемого ресурса.

3. Методы распараллеливания могут применяться как на уровне программ, процессов, процедур, так и на уровне команд и операций.

4. Допускается наличие не только последовательных, но и параллельных входных программ.

5. Задачу распараллеливания можно представить как задачу нахождения эффективного преобразования программ или спецификаций одного класса в параллельные программы другого класса, осуществляющие асинхронную обработку данных со взаимодействием над общей памятью.

6. При рассмотрении методов эффективного преобразования программ можно выделить двойственные постановки задачи распараллеливания.

7. Двойственные постановки задачи в свою очередь влекут к взаимной зависимости и независимости операторов при распараллеливании больших массивов данных при их вычислениях.

8. Эффективность работы аппаратных характеристик, также важна при распараллеливания вычислений, так как дает возможность оценить степень ее участия в общей работе при решении задачи установления авторства текстов.

Ни один из методов распараллеливания сам по себе не может являться основой для серьезной практической реализации, так как должен разрабатываться с учетом соответствующих теоретических результатов и представлять собой синтез некоторой совокупности универсальных и специальных методов.

Формулировка задачи распараллеливания процессов при определении авторства текстов

С учетом вышесказанного задача преобразования ПС «Plagiarizm» в параллельную программу

сводится к тому, что необходимо гарантировать, чтобы все обработчики событий завершались достаточно быстро. Также в соответствии с наданной функциональной декомпозицией преобразование ПС «Plagiarizm» в параллельную программу предполагает присвоение разным данным потока различных функций, но в силу того, что множество задач выполняется последовательно целесообразно провести распараллеливание на уровне анализа работ.

Процесс распараллеливания можно разделить на две части:

- анализ исходной программы;
- синтез параллельной программы.

Анализ необходим для выявления скрытого параллелизма в исходной последовательной программе. Прежде всего, сюда включается выявление зависимостей по данным между операторами программы. Такая зависимость возникает, например, в случае, если один оператор использует значение переменной, вычисленное некоторым другим оператором. Независимые операторы могут быть выполнены параллельно на разных процессорах. Обычно этот принцип применяется к циклам: если нет зависимостей по данным между разными витками цикла, то витки могут быть выполнены параллельно разными процессорами. Также на этапе анализа могут собираться сведения о необходимом размещении данных в случае, если используется система с распределенной памятью. Кроме того, возможен сбор сведений о времени выполнения различных участков программы с целью выбора наилучшего варианта распараллеливания программы.

Синтез параллельной программы включает выбор схемы распределения данных (если необходимо) и вычислений, а также непосредственно генерацию текста параллельной программы с использованием подходящих инструментов параллельного программирования.

Выявление зависимостей по данным в исходном тексте программы является важнейшим этапом анализа. Существуют различные методы выявления зависимостей: статические методы, анализирующие текст программы, динамические методы, исследующие ход выполнения программы на некоторых тестовых данных, различные тесты, проводимые в ходе выполнения готовой параллельной программы.

Помимо анализа программы система автоматизации распараллеливания получает различную информацию от пользователя.

В ходе анализа был сделан вывод о том, что по своей направленности ПС «Plagiarizm» относится к событийно-ориентированным приложениям, а также имеет сложно организованную структуру, когда в составе программы имеются и параллельные участки и циклические. При преобразовании ПС «Plagiarizm» в параллельную программу необходимо гарантировать, что все обработчики событий будут завершаться достаточно быстро.

Было отмечено, что важное место в преобразовании ПС «Plagiarizm» в параллельную программу занимают теоретические вопросы, включающие выбор модели распараллеливания, определение эквивалентности программ, определение степени параллелизма и синхронизацию параллельных процессов, анализ потенциального параллелизма алгоритмов и программ, определение максимального параллелизма и др.

Выводы

В настоящее время одной из наиболее острых проблем образовательного процесса в Украине является проблема нарушения академической добросовестности студентов. Поэтому на данном этапе актуальным является создание эффективного информационного инструментария для анализа естественно-языковых текстов в ПУР.

В исследовании были рассмотрены существующие типы распараллеливания приложений и проведен их анализ. В ходе проведенного анализа, было установлено, что существует несколько обобщающих особенностей при решении проблемы распараллеливания вычислений для задачи установления авторства текстов.

На основе проведенного анализа была сформулирована задача распараллеливания процессов при определении авторства текстов.

Кроме того, был сделан вывод о том, что по своей направленности ПС «Plagiarizm» относится к событийно-ориентированным приложениям, а также имеет сложно организованную структуру, когда в составе программы имеются и параллельные участки и циклические. При преобразовании ПС «Plagiarizm» в параллельную программу необходимо гарантировать, что все обработчики событий будут завершаться достаточно быстро.

Было отмечено, что важное место в преобразовании ПС «Plagiarizm» в параллельную программу занимают теоретические вопросы, включающие выбор модели распараллеливания, определение эквивалентности программ, определение степени параллелизма и синхронизация параллельных процессов, анализ потенциального параллелизма алгоритмов и программ, определение максимального параллелизма и др.

В перспективе необходимо выполнить усовершенствование математического и методического обеспечения системы поиска плагиата для компьютеризации процесса установления авторства текста,

на основе параллельных вычислений, которое, в свою очередь, направленно на повышение эффективности таких систем.

Литература

1 Груздо, И. В. Проблемы анализа естественно-языковых текстов для обнаружения плагиата в учебных работах [Текст] / И. В. Груздо // Радіоелектронні і комп'ютерні системи. –2011. - № 1 (49). - С. 130-138.

2 Груздо, И. В. Информационная технология выявления текстовых заимствований в работах студентов технических вузов [Текст] / И. В. Груздо // Системы управления навигации и связи. –2012. – № 3 (23). – С. 160-165.

3 Flynn, M. J. Some computer organizations and their effectiveness [Текст] / M. J. Flynn // IEEE Transactions on Computers. – 1972. - № 21(9). – С. 948-960.

4 Классификации архитектур вычислительных систем [Электронный ресурс]. – Режим доступа: http://www.parallel.ru/computers/taxonomy/. Заголовок с экрана. – 29.05.2014.

5 Часть I. Введение в архитектуры и средства программирования многопроцессорных вычислительных систем [Электронный ресурс]. – Режим доступа: http://rsusu1.rnd.runnet.ru/tutor/method/m1/content.html. Заголовок с экрана. – 18.02.2014.

6 Параллельные вычисления (базовый курс) [Электронный ресурс]. – Режим доступа: http://bigor.bmstu.ru/?cnt/?doc=Parallel/base.cou. Заголовок с экрана. – 18.04.2014.

7 Лекция 10. Архитектуры многопоточных приложений [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/studies/courses/570/426/lecture/9702. Заголовок с экрана. – 18.02.2014.

Поступила в редакцию 30.05.2014, рассмотрена на редколлегии 16.06.2014

Рецензент: д-р техн. наук, проф., зав. каф. экономико-математического моделирования В. М. Вартанян, Национальный аэрокосмический университет им. Н. Е. Жуковского «ХАИ», Харьков.

ЗАСТОСУВАННЯ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ ДО ВИРІШЕННЯ ЗАВДАННЯ ВСТАНОВЛЕННЯ АВТОРСТВА ТЕКСТІВ

І. В. Груздо, С. В. Россоха, І. В. Шостак

У роботі проведено огляд та критичний аналіз існуючих типів розпаралелювання програм. Дано загальний опис інформаційної технології аналізу та оцінювання письмових навчальних робіт з урахуванням наявності в них текстових запозичень. Виділено основні проблеми розпаралелювання обчислень для вирішення завдання встановлення авторства текстів. Виконано постановку формального завдання розпаралелювання процесів при визначенні авторства текстів. Показано необхідність удосконалення математичного та методичного забезпечення систем пошуку плагіату, спрямованого на підвищення ефективності таких систем.

Ключові слова: паралельні обчислення, плагіат, розпаралелювання, перетворення, розподілена пам'ять, ступені паралелізму.

APPLICATION PARALLEL COMPUTING TO THE PROBLEM OF AUTHORSHIP ESTABLISHMENT OF TEXTS

I. V. Gryzdo, S. V. Rossokha, I. V. Shostak

The paper contains an overview and critical analysis of existing types of parallelizing programs. Given a general description of information technology analysis and evaluation of written academic papers based on presence of textual borrowing. Basic problems of parallelization of computations to solve the problem of establishing the authorship of texts. Executed a formal statement of task parallelization process in determining authorship of texts. The necessity of improving mathematical and methodological support of finding plagiarism aimed at improving the efficiency of such systems.

Keywords: parallel computing, plagiarism, parallelization, transform, distributed memory, degree of parallelism.

Груздо Ирина Владимировна – канд. техн. наук, ассистент кафедры инженерии программного обеспечения, Национальный аэрокосмический университет им. Н. Е. Жуковского «Харьковский авиационный институт», Харьков, Украина, e-mail: tigralwovna@rambler.ru.

Россоха Сергей Владимирович – канд. техн. наук, доцент кафедры инженерии программного обеспечения, Национальный аэрокосмический университет им. Н. Е. Жуковского «Харьковский авиационный институт», Харьков, Украина, e-mail: sergey.rossokha@gmail.com.

Шостак Игорь Владимирович – д-р техн. наук, проф., проф. кафедры инженерии программного обеспечения, Национальный аэрокосмический университет им. Н. Е. Жуковского «Харьковский авиационный институт», Харьков, Украина, e-mail: iv_shostak@rambler.ru.