Московский Государственный Университет имени М.В. Ломоносова

Механико-математический факультет

Кафедра вычислительной механики

Реферат

Подсистема автоматического распределения вычислительных объектов по процессорам

Выполнил

Ворошилов С.С.

Научный руководитель:

к.ф-м.н. Илюшин А.И.

Москва 2014

Содержание

Введение 2
Основные проблемы параллельных вычислений 4
Структура системы OST 6
Аналогия с виртуальной памятью 8
Проблемы при подкачке 10
Алгоритмы подкачек 11
Структурная схема планировщика 12
Обработка функции Object Fault 14
Реализация и результаты 15
Выводы 16
Список литературы 17
Приложения 18

Введение

Общепризнанно, что в настоящее время программирование многопроцессорных (многоядерных) вычислительных систем является задачей, которая требует от программиста и высокой квалификации, и больших трудозатрат.

Попытаемся разобраться, в чем же конкретно заключается проблема. Представим себе программную систему, состоящую из множества взаимодействующих программ, распределенных по множеству процессоров. В этом случае в отличие от одной последовательной программы, выполняющейся на одном процессоре, возникают две основные проблемы:

1 Проблема установления связей между удаленными друг от друга программами и управления этими связями.

2 Проблема частичного упорядочения действий, выполняемых этими программами, и синхронизации их взаимодействия.

Именно для решения этих проблем в «ручном» режиме в настоящее время требуются и незаурядные умственные способности, и большие трудозатраты. Оказывается, что параллельный случай можно свести к последовательному в смысле требований к программисту и процессу программирования путем ввода принципиально новых программных механизмов, основанных на понятиях «пространство-время».

Дело в том, что любая система состоит из некоторого множества частей с некоторым конкретным набором связей между ними. Это означает, что практически все прикладные программисты будут программировать примерно одинаковые действия для управления связями между частями программной системы. Аналогичная ситуация с арифметическими выражениями существовала, например, до появления Фортрана. Каждый программист программировал в Ассемблере арифметические выражения. Вынесение «за скобки» в системную часть повторяемых всеми действий – это естественный выход из такого рода ситуаций.

Такое вынесение может быть сделано только на основе формальной фиксации некоторого понятия, описывающего алгоритмизируемый класс действий. Для множеств, в которых определены связи между элементами (в другой формулировке для каждого элемента определена окрестность в виде подмножества элементов, связанных с данным), таким понятием является понятие топологического пространства.

Упорядочение действий, выполняющихся в различных параллельно эволюционирующих частях системы, описываются понятием времени.

Именно эти два понятия и были положены в основу при разработке комплекса программных средств, целью которого является сведение трудоемкости программирования параллельных систем к трудоемкости последовательного программирования.

Такая система была создана Михаилом Олениным под руководством Александра Ивановича Илюшина и была названа OST (Object - Space - Time). В данной системе за основу берется обычная объектно-ориентированную модель программирования, которая дополняется некоторыми новыми чертами. Программная система строится из множества взаимодействующих друг с другом объектов в рамках некоторой объектно-ориентированной среды программирования.

Первые ее версии проводили счет только в так называемом "статическом"случае, когда количество объектов было равно количеству выделенных для счета процессоров. Однако, если количетсво объектов больше количества процессоров (что имеет существенный практический смысл в организации счета прикладных задач), необходимо динамически распределять объекты по процессорам.

Собственно реализации такого "динамического"счета посвящена данная работа. Основная идея решения – статически до начала счета распределять виртуальные процессоры (их количество соответствует количеству объектов), а во время счета отображать каждый виртуальный процессор на реальный динамически. Поэтому постоянно нужно будет откачивать "ненужные" подобласти с процессоров в файл объектов и наоборот. От рациональности выбора объектов для этих действий будет зависеть скорость вычислений (необходимо свести простои процессоров к минимуму). Поскольку все элементы модели активно взаимодействуют друг с другом и могут довольно быстро продвигаться во времени, управление подкачкой вручную в некоторых случаях, вообще, не представляется возможным.

Разработка подсистемы подкачки объектов в системе OST - главная цель данной работы. (*Прим. - в дальнейшем будем называть подсистему "Планировщик объектов"*)

Основные проблемы параллельных вычислений

Целью системы OST является существенное упрощение программирования задач и повышение эффективности их счета при работе на многопроцессорных вычислительных системах. Подход, предлагаемый для построения параллельных программных моделей в рамках проекта OST, отличается от распространенных в настоящее время подходов тем, что в нем делается попытка «спрямить» путь от физической модели к программной реализации. Этот подход состоит в следующем.

Пусть имеется физическая модель (область), которую здесь мы будем рассматривать в виде множества физических объектов, эволюционирующих и взаимодействующих друг с другом. В любом случае нам нужно отобразить это множество физических объектов на множество программно-аппаратных объектов. Традиционный путь состоит в том, что строится математическая модель для этой области и вычислительная схема, позволяющая получить численные решения для некоторого множества начальных и граничных условий. Более конкретно вычислительная модель чаще всего представляется в виде некоторого набора матриц, векторов, чисел и некоторого последовательного алгоритма обработки этих объектов для получения искомых решений. Далее строится параллельная программно-аппаратная модель (здесь под ней понимается множество программ, работающих на разных процессорах и взаимодействующих друг с другом). При таком подходе изначально параллельная физическая модель сначала отображается на последовательную вычислительную модель, а затем решается обратная задача – последовательная вычислительная модель отображается на параллельную программно-аппаратную модель. Так как при переходе от параллельной физической модели к последовательному алгоритму вычислений теряется информация об изначальном параллелизме физической модели, то, основываясь только на анализе последовательного алгоритма, вообще говоря, невозможно восстановить исходный параллелизм для построения параллельной программно-аппаратной модели. Оказывается невозможным понять, почему, например, два конкретных действия выполняются последовательно. Возможно, порядок этих действий определяется сутью физической модели, а возможно это техническое решение автора последовательного вычислительного алгоритма.

В идеологии проекта OST естественно строить математические, вычислительные и программные модели независимо для частей физической модели, а затем с помощью средств OST , более или менее автоматически получать композицию из программных частей, которая будет описывать физическую модель в целом. Тем не менее, подчеркнем, что для

средств OST несущественен источник получения частей вычислительной модели. Это могут быть и вручную выделенные части последовательной вычислительной модели. Однако естественным будет «фотографическое» отображение частей физической модели и связей между ними на множество программно-аппаратных объектов, минуя стадию построения математической и вычислительной моделей для всей области в целом. Имеется в виду, что математическая и вычислительная модели должны описывать именно части физической модели и взаимодействие на границах (реальных или виртуальных) между ними, а не всю физическую модель в целом. Предлагается интеграция частей в стиле натурного моделирования на уровне программно-аппаратных объектов, которые в данном случае играют роль виртуальных частей «натурной» модели.

Теперь попробуем разобраться для чего нужны подкачки и какова их польза. Рассмотрим этот вопрос со стороны прикладного программиста и администратора многопроцессорной вычислительной системы.

Прикладной программист:

1 Время на решение задачи складывается из времени на счет (t_c) и времени ожидания в очереди на свободные процессоры (t_q). Так как при большом количестве запрашиваемых процессоров $t_q >> t_c$, то суммарное время решения задачи $t = t_c + t_q$ будет в основном зависеть от нахождения в очереди. С помощью планировщика, заказывая меньшее количество процессоров, время t_q, а, следовательно, и время t должно сильно снизиться.

2 Так же планировщик будет очень полезен при отладке задач.

Администратор МВС

1 На МВС считаются разные задачи: считать может студент, а может и государственная организация. Естественно необходимо учитывать приоритетность задач. Планировщик может помочь в решении данной проблемы, освобождая место на узлах для высокоприоритетных расчетов.

2 Уменьшение простоев процессоров и оптимизация работы вычислительного комплекса - так же одни из важных проблем любого администратора. Здесь планировщик также применим, например, следующим образом: «пассивные» объекты помещаются группой на один процессор или «выталкиваются» на диск, освобождая тем самым места на вычислительных узлах для расчетов других задач.

Структура системы OST

Как упоминалось во введении в основу решения основных проблем параллельного программирования в системе OST положены две идеи:

- «ВРЕМЯ» практически во всех областях деятельности человека всегда использовалось и используется для упорядочения действий. «Время» в нашем случае – это разметка числами всех рассматриваемых действий. Единственное условие синхронизации - взаимодействие пары объектов разрешается только при «равенстве» их локальных времен.

- «ПРОСТРАНСТВО», в котором расположено рассматриваемое множество объектов, является столь же широко и эффективно используемым понятием. В нашем случае для организации связей между взаимодействующими объектами используется понятие «близости» пары объектов в пространстве конкретного типа. Например, это может быть трехмерная решетка (шесть соседей) или кольцо (два соседа). Могут взаимодействовать (вызывать операции друг в друге) только «соседи» («близкие» объекты).

Использование программных механизмов, основанных на этих идеях, позволяет «автоматизировать» синхронизацию и управление связями между объектами. Такая автоматизация дает два основных результата:

- Локальность программирования для прикладного программиста. Он описывает алгоритм функционирования каждой подобласти локально, а функционированием модели всей области управляет система OST.

- Система OST получает возможность автоматически распределять части вычислительной модели (объекты) по процессорам в стиле «подкачек» страниц в операционных системах. Результат – возможность оптимизации загрузки для всей МВС и упрощения для прикладного программиста.

Прикладной программист строит вычислительную модель прикладной области в виде множества объектов, взаимодействующих путем вызова операций друг в друге. Он создает описания классов объектов и программу "раскрутки" модели.

Программа "раскрутки" модели, используя ранее созданные описания классов и специальное «объектохранилище» - файл объектов, создает все объекты, входящие в состав программной модели. Эти объекты хранятся в файле объектов в «сериализованном» виде. Естественно, что в дальнейшем в момент счета прикладной задачи «сериализованные» объекты

будут «подкачиваться» из файла в оперативную память, «десериализовываться», и «считаться». После того как в некотором множестве процессоров появилось множество объектов прикладной модели и во всех этих объектах вызвана операция запуска счета естественно возникает два вопроса:

- как они свяжутся друг с другом, а конкретно как в них появятся ссылки друг на друга, необходимые для вызова операций?
- как будет происходить синхронизация моментов вызова? Ведь если один объект запрашивает результаты счета у другого объекта, то в момент вызова эти результаты должны быть готовы.

Обе эти проблемы решаются путем введения механизма формализующего понятие «окружение» для программного объекта в виде списка «формальных» соседей. Это понятие можно рассматривать как существенное обобщение понятия списка формальных параметров для подпрограммы. Прикладной программист программирует свой прикладной класс полностью локально и использует при этом окружение в виде списка ссылок на соседей, считая, что в момент счета система подставит ему в список формальных соседей ссылки на фактических соседей в соответствии с текущими координатами и текущим временем конкретного объекта рассматриваемого класса и всех других объектов программной модели. А именно, объекты становятся фактическими соседями, между которыми возможен вызов операции, в соответствии с их координатами в пространстве конкретной топологии и в тот момент, когда их локальные времена совпадают.

Аналогия с виртуальной памятью

Ранее было замечено, что основная идея решения поставленной задачи – статически до начала счета распределять виртуальные процессоры, а отображать каждый виртуальный процессор на реальный – динамически во время счета.

Недостаток памяти, с которой может напрямую оперировать процессор, встречается не только в многопроцессорных системах. В обычных компьютерах часто не хватает оперативной памяти, особенно при запуске "тяжелых" программ. И операционной системе приходится решать эту проблему. Для этого используется технология управления памятью называемая виртуальной памятью.

В большинстве современных операционных систем виртуальная память организуется с помощью страничной адресации. Оперативная память делится на страницы: области памяти фиксированной длины (обычно 4096 байт), которые являются минимальной единицей выделяемой памяти. Процесс обращается к памяти с помощью адреса виртуальной памяти, который содержит в себе номер страницы и смещение внутри страницы. Процессор преобразует номер виртуальной страницы в адрес соответствующей ей физической страницы. Если ему не удалось это сделать, то требуется обращение к таблице страниц (так называемый Page Walk), что может сделать либо сам процессор, либо операционная система. Если страница выгружена из оперативной памяти, то операционная система подкачивает страницу с жёсткого диска. Это осуществляется на основании "алгоритма подкачки" аналогию которому и нужно построить для многопроцессорной системы. При запросе на выделение памяти операционная система может «сбросить» на жёсткий диск страницы, к которым давно не было обращений. Критические данные (например, код запущенных и работающих программ, код и память ядра системы) обычно находятся в оперативной памяти (исключения существуют, однако они не касаются тех частей, которые отвечают за обработку аппаратных прерываний, работу с таблицей страниц и использование файла подкачки).

Теперь проведем аналогии с нашим случаем. Объект достаточно сложная структура, однако в поставленной задаче он является аналогом страницы в виртуальной памяти. По сути нам необходимо сделать отображение множества объектов на множество процессоров, но так как объектов гораздо больше процессоров, то нужно вводить виртуальные процессоры, которые позволят сделать требуемое взаимо-однозначное соответствие.

Объекты можно грубо разделить на две группы: готовые к счету и неготовые. Первые имеют либо внешние вызовы, которые можно выполнить (мешать этому может временная

несогласованность вызываемого и вызывающего), либо внутренние (в том числе и повышение во времени, мешать чему может монитор, в соответствии с программой временной синхронизации). Готовые к счету являются аналогом страниц, которые затребовал процесс и которые, при загрузке в основную память, сразу будут считаться. Неготовые таковыми не являются.

Алгоритмы подкачки страниц достаточно просты. Более сложные не оправдали себя, так как, при большой интенсивности операций со страницами, они замедляют выполнение процессов. В нашем случае ситуация аналогична: при большом количестве подобластей, что вероятно для реальных физических моделей, придется довольно часто перекидывать объекты между файлом объектов и узлами. Если же это не так, то порядок подкачки, если он не совсем неправильный, вообще, будет иметь слабое влияние на время расчетов, так как система без проблем будет успевать загрузить объект на узел, пока на том идут длительные расчеты.

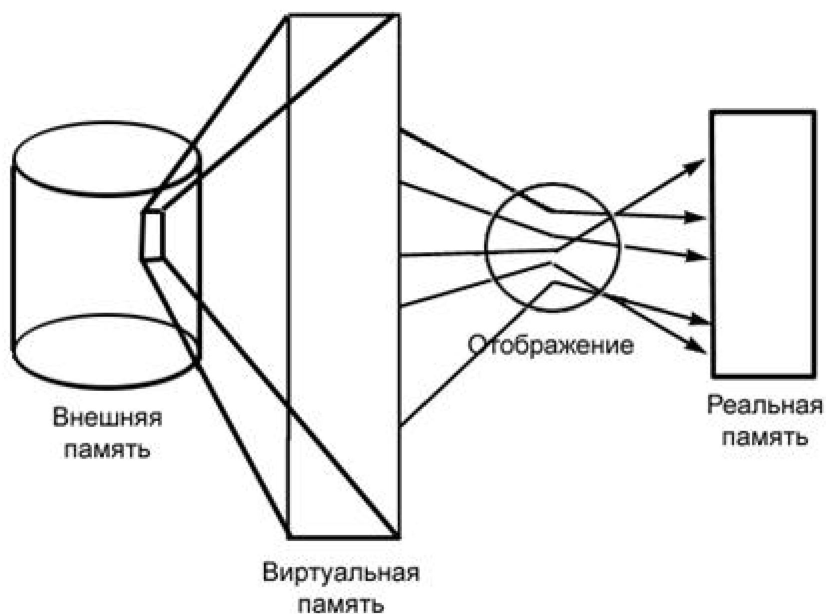


Рис. 1: Основная концепция виртуальной памяти

Проблемы при подкачке

Рассмотрим некоторые проблемы, которые могут возникнуть в работе подсистемы подкачки объектов.

1 Прежде всего, для загрузки объекта на процессор система должна выбрать подходящий для этого узел. Здесь все решается просто: выбирать нужно процессор, на котором меньше всего готовых объектов. Для этого необходимо в мониторе завести счетчик показывающий их количество на каждом из узлов. Вторичным фактором должно послужить количество запросов на объект. Чем их больше, тем он нужнее. Поэтому для каждого объекта необходимо ввести характеристику числа вызовов на него.

2 Пусть у нас есть объект, сделавший вызов на объект, которого нет на узле. Данный объект подкачивается, для чего выталкивается первый объект, после чего все происходит наоборот. Данная ситуация напоминает так называемый *thrashing* - аналогичную ситуацию в виртуальной памяти. Если у нас есть много других готовых объектов, то эти два потеряются среди них, они не будут считаться друг за другом и заставлять простаивать процессоры. Но допустим, что у нас получилось так, что вся модель ждет этих двух объектов. Непосредственно ждут только соседи (соседи по координатам внутри модели, а не по узлам), причем они не могут ждать из-за ожидания выполнения вызова, посланного на кого-то из них (он бы выполнился, во время расчета объекта). Значит, эти объекты ждут своего продвижения во времени. Проблема легко решается, если выталкиваться будет объект, остановленный по временной синхронизации, что логично, даже забудь мы про thrashing: лучше досчитать все в текущем времени, а потом двигаться дальше.

Алгоритмы подкачек

Прежде чем переходить к реализации нашей подсистемы обратимся к некоторой теории по алгоритмам замещения.

Логическим обращением к объекту p называется выполнение какой-либо операции над данными, содержащимися в объекте p. (***logical reference - LR***)

Физическим обращением к объекту p называется логическое обращение к этому объекту, приводящее к его считыванию с внешнего устройства хранения данных. (***physical reference - PR***)

Пусть E - множество объектов в файле объектов. Тогда алгоритм замещения φ определяется как

$$\varphi(E, \delta) \to p, p \subset E,$$

где p - объект, выбранный для замещения, δ - дополнительные данные об объекте, которые использует алгоритм замещения.

В контексте введенной терминологии задачей алгоритма подкачек является минимизация отношения числа **PR** к **LR** для заданной последовательности обращений к блокам данных.

Виды алгоритмов подкачек

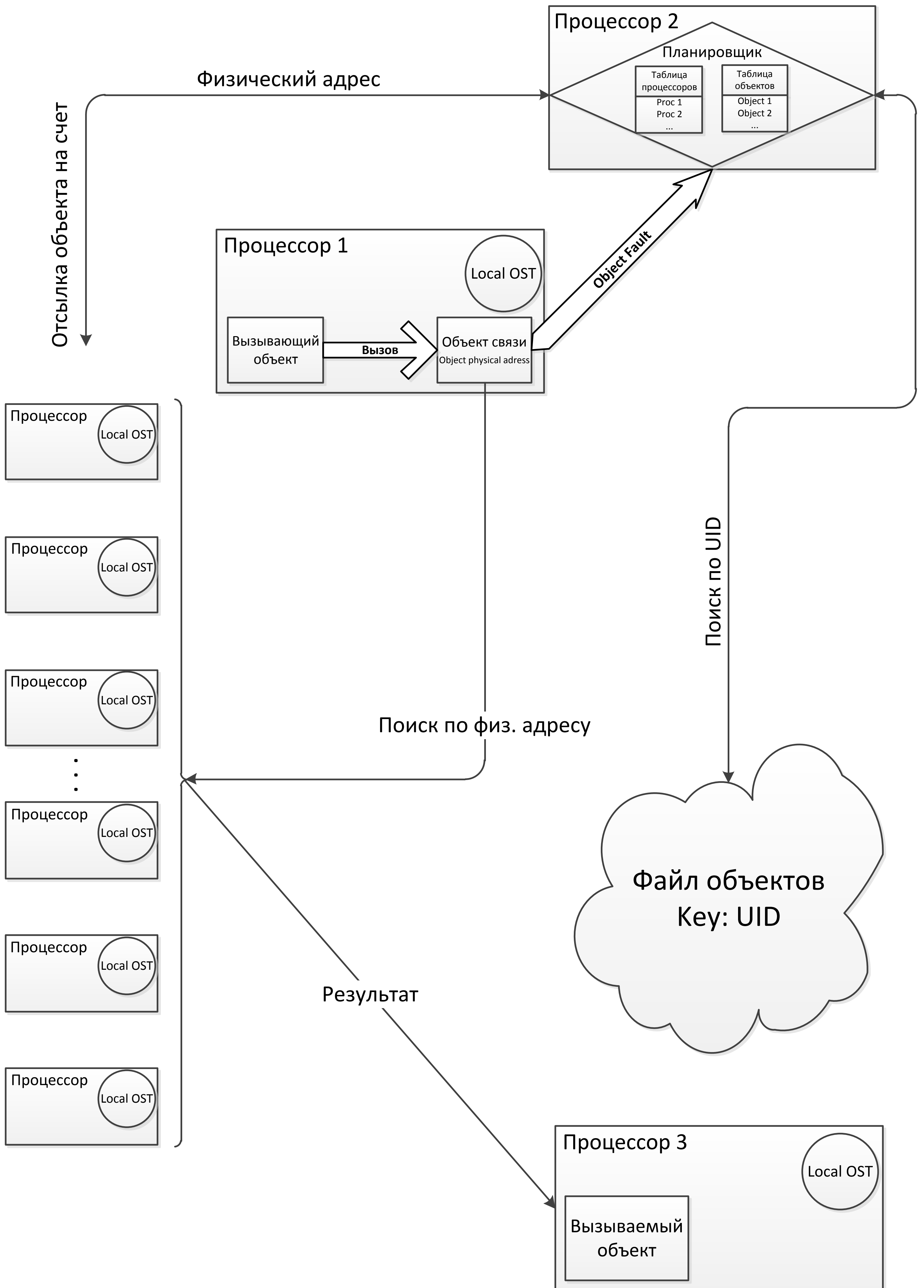
Demand paging algorithms - при необходимости замещения среди объектов один единственный и замещается

Prepaging algorithms - для замещения выбираются сразу несколько объектов. Алгоритмы данного типа могут быть эффективны в особых случаях, но потенциально сложны в реализации.

В работе применялся demand paging алгоритм

Структурная схема подсистемы подкачки

При инициализации объектов в системе OST им приписываются уникальные идентификаторы **(UID)** - аналог виртуального адреса. Объекты которые находятся на процессорах обращаются к соседям посредством удаленных вызовов с помощью технологии RPC. В нашей реализации для этого используется библиотека PyRo. Объекты осуществляют удаленный вызов по ссылкам **URI** - аналог физического адреса. Благодаря введенным определениям можно описать процесс работы подсистемы с помощью структурной схемы представленной ниже.



Процессор 2
Планировщик
Таблица процессоров
Proc 1
Proc 2
...
Таблица объектов
Object 1
Object 2
...
Физический адрес
Отсылка объекта на счет
Процессор 1
Local OST
Вызывающий объект
Вызов
Объект связи
Object physical adress
Object Fault
Процессор
Local OST
Процессор
Local OST
Процессор
Local OST
Процессор
Local OST
Процессор
Local OST
Процессор
Local OST
Поиск по UID
Поиск по физ. адресу
Файл объектов
Key: UID
Результат
Процессор 3
Local OST
Вызываемый объект

Обработка функции Object Fault

Как видно на схеме, в случае если вызываемого объекта нет на процессоре (его физический адрес пуст) в вызывающем объекте срабатывает функция Object Fault. Рассмотрим процесс ее работы подробнее.

1 Вызов операции по ссылке на объект идет по физическому адресу (из этой ссылки) реального процессора, в котором находится вызываемый объект. Если физ. адрес пуст или объект в данном процессоре отсутствует (ранее вытолкнут) то:

2 Планировщик идет в файл объектов и находит там нужный объект.

3 Планировщик ищет свободный процессор или, если такового нет, ищет объект – кандидат на выталкивание. Производится выталкивание объекта в файл объектов.

4 Вызываемый объект вталкивается на свободный или освобожденный процессор. В планировщик возвращается его новый физ. адрес и он записывает его в ссылку на объект.

5 Повтор вызова

Реализация и результаты

Реализация поставленной цели осуществлялась на многоядерной процессорной системе **Intel Xeon Phi** (архитектура MIC). Она представляет из себя сопроцессорную плату-ускоритель. Как и в графических ускорителях здесь используются ускорители типа Pentium (240 шт). Высокая ступень параллелизма в архитектуре MIC достигается за счет использования процессорных ядер меньшего размера, потребляющих меньше энергии. Результатом является высокая производительность в системах с интенсивной параллельной обработкой данных.

Для работы на MIC необходимо создать операционную и программную среду, которая состоит из операционной системы Linux на каждом процессоре с компиляторами Fortran, C++ поддерживающими многопроцессорную работу. Так же для связи процессоров необходимо установить MPI.

Была установлена операционная система $Linux 2.6.32-358.el6.x86_64$, установлены пакеты Intel ComposerXE включающие необходимые компиляторы поддерживающие многопроцессорную работу. Была осуществлена поддержка MPI. Система была успешно протестирована сотрудниками ИПМ (В. Жуков, Н.Новикова, М.Краснов) в ходе тестово-производственных счетов.

К сожалению, компанией Intel не предусмотрена поддержка языка Python на котором написана система OST. Необходимым шагом была установка Python'а и основных библиотек необходимых для работы и расчетов.

Этот шаг был одним из самых сложных в реализации, ввиду того, что даже в англоязычном интернете мало информации по этому поводу. Однако в ходе совместных обсуждений вместе с заинтересованными в этом вопросе людьми в конференции на форуме компании Intel решение было найдено. Python и необходимые библиотеки были установлены.

Следующим шагом был перенос системы OST на новую платформу с учетом внутренних особенностей MIC. Версия системы OST была успешно перенесена, что подтвердилось тестами.

Далее собственно можно переходить к реализации алгоритма подкачек. Была запрограммирована первая версия с подкачками и пропущены первые тесты. Исходный код приведен ниже в приложении.

Выводы

В рамках дипломной задачи ставилась цель разработать подсистему автоматического распределения объектов по процессорам для системы OST.

Реализация проводилась на новой многоядерной системе Intel Xeon Phi. В процессе этой работы пришлось установить и протестировать большое число программных компонент. Был осуществлен перенос системы OST на MIC, реализована подсистема подкачки объектов и проведены первые тесты.

Установленные системы были практически использованы в производственном счете.

Список литературы

[1]. А.И.Илюшин, М.А.Оленин, С.А.Васильев - Решение проблемы параллельных вычисленниий на основе понятий «пространство-время»
[2]. http://ost.kiam.ru/

Приложения