МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра Параллельных Вычислений
(название кафедры)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Никитин Александр Алексеевич
(фамилия, имя, отчество автора - студента –выпускника)

Реализация распределенного динамического массива для системы фрагментированного программирования
(тема работы)

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Руководитель

Перепелкин В.А
(фамилия , И., О.)
……..
(уч.степень, уч.звание)

……..
(подпись, дата)

Автор

Никитин А.А.
(фамилия , И., О.)
ФИТ, 8201
(факультет, группа)

……..
(подпись, дата)

Новосибирск, 2012 г.

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра Параллельных Вычислений
(название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой……
(фамилия, И., О.)

……
(подпись, дата)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту *Никитину Александру Алексеевичу*
(фамилия, имя, отчество)

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема «Реализация распределенного динамического массива для системы фрагментированного программирования»
(полное название темы выпускной квалификационной работы бакалавра)

Исходные данные (или цель работы): Реализовать программный модуль поддержки распределенных динамических массивов для применения в численных методах в рамках системы фрагментированного программирования LuNA.

Структурные части работы: Изучение проблемы управления распределенными данными на примере метода «частиц в ячейках». Анализ требований к динамическим распределенным массивам для использования при решении задач численного моделирования. Разработка алгоритмов управления распределением и локальными данными динамического массива. Реализация модуля динамических массивов для системы фрагментированного программирования LuNA.

Содержание

ВВЕДЕНИЕ

В настоящее время большие вычислительные мощности, представленные различными параллельными компьютерными системами (многопроцессорные серверы, компьютерные сети, суперкомпьютеры, грид-системы и т.п.), стали доступны широкому кругу пользователей. Это позволяет за приемлемое время решать задачи с большим объемом данных и вычислений, которые раньше невозможно было реализовать на последовательных компьютерах. Но возможности параллельных вычислений непосредственно связаны со сложностями параллельного программирования, которые в свете расширенного круга пользователей встают с особой остротой. Параллельной программе приходится управлять во времени системами, достигающими тысяч процессоров и терабайт оперативной памяти. Рассмотрим основные трудности, возникающие перед программистом [1].

- Требуется разработать параллельный алгоритм, управляющий вычислительной системой («распараллелить» задачу). Последовательные программы не могут, в чистом виде, исполняться параллельно. Корректность параллельного алгоритма отследить трудно, так как приходится иметь дело с несколькими или многими взаимодействующими во времени процессами.
- Необходимо отобразить этот алгоритм на имеющиеся ресурсы – распределить вычислительные задачи и данные, обеспечить необходимую синхронизацию в работе и обмен данными, скоординировать совместную работу над исходной задачей. Настройка на имеющиеся ресурсы включает учет особенностей аппаратного и программного обеспечения конкретной системы, таких как объемы дисковой, оперативной и кэш-памяти, тип и производительность вычислительных устройств, задержка и пропускная способность сети, ее топология, имеющиеся программные библиотеки и возможности операционной системы. Для решения этой задачи требуются специализированные знания в области аппаратного обеспечения. В частности, актуальна задача осуществления коммуникаций на фоне вычислений, решение которой требуется для задач, активно использующих как вычисления, так и коммуникации.
- Работа с распределенными данными представляет собой нетривиальную задачу. Она требует дополнительных усилий и имеет определенные ограничения. Это необходимо учитывать при разработке параллельного алгоритма. Требуется реализовать перемещение данных по мере необходимости в конкретных процессорах. Где-то может потребоваться дублирование или поддержка

когерентности данных. Каждая из этих задач не имеет универсального решения и должна каждый раз решаться в конкретных условиях задачи. Эффективная реализация может потребовать использования алгоритмов планирования.

- Распределение работ конкретных процессоров не всегда известно заранее. Во время исполнения одни процессоры могут простаивать без работы, в то время, как другие полностью нагружены. Такая ситуация требует динамической балансировки нагрузки – перераспределение подзадач в соответствии с нагрузкой на процессоры во время исполнения. Широкий спектр задач численного моделирования без динамической балансировки невозможно решить за приемлемое время.

Таким образом, программисту приходится тратить значительную часть своих усилий на программирование системных составляющих, не связанных с его задачей. Разработка, отладка и сопровождение параллельных программ требует высокой квалификации системного программирования, которой не обладают, а, главное, и не должны обладать пользователи параллельных компьютеров.

В идеале, пользователь, например физик или биолог, должен иметь возможность с минимальными усилиями запрограммировать алгоритм, получив при этом эффективную параллельную программу, автоматически и динамически настраивающуюся на имеющиеся ресурсы, обеспечивающую полную нагрузку на вычислитель и т.д. Словом, трудности параллельного программирования должны быть скрыты, а нужные качества программы – реализовываться автоматически.

Из рассмотренных проблем возникающих при параллельном программировании особый интерес представляет управление распределением данных. Особенно это касается тех задач, в которых требуется динамическое перераспределение данных между процессами в соответствии с определенными критериями.

Существует много подходов к управлению распределением данных, большинство из которых ориентированно на определенный круг задач. Данная работа посвящена реализации поддержки управления распределением данных с помощью динамических массивов для метода PIC в системе фрагментированного программирования LuNA.

1 Задача распределения данных

1.1 Проблема распределения данных в методе PIC

Метод частиц в ячейках (PIC, Particle-in-cell) [2] используется для решения некоторых видов дифференциальных уравнений в частных производных, например для моделирования траекторий заряженных частиц под действием электромагнитного (или электростатического) поля. В данном методе пространство делится на ячейки, в которых в каждый момент модельного времени находится некоторое количество частиц, а значения поля рассчитываются в стационарных узлах сетки.

При реализации решения задач, использующих метод PIC возникают задачи распределения данных и динамической балансировки.

Рассмотрим пример работы метода PIC: пусть в начальный момент времени нам известно распределение частиц по ячейкам и значение поля в узлах сетки. Вычисления на каждой итерации заключаются в нахождении перемещения каждой частицы под действием внешнего поля, и дальнейший пересчет значений поля в узлах сетки после перемещения частиц. Данные алгоритмы реализованы в виде процедур, которые работают с отдельными ячейками. Множество частиц в ячейке представлено в виде массива структур с необходимыми данными о частицах (например, заряд и координаты частицы). Исходя из специфики задачи, необходима возможность переноса данных из одного массива частиц в другой, а также возможность изменять объем доступной памяти, так как заранее не известно какое количество частиц может оказаться в данной ячейке.

Распределение частиц в начальный момент времени может быть неравномерным, поэтому если осуществлять простое отображение ячеек на узлы суперкомпьютера, то сразу же возникнет ситуация с неравномерной загрузкой узлов, и как следствие, возможно значительное падение производительности. Поэтому необходимо изначально распределить данные так, чтобы загрузка узлов была равномерной. В дальнейшем ситуация с неравномерной загрузкой может повторится — в одних ячейках может быть значительно больше частиц, чем в других, и соответственно время работы алгоритмов на данных ячейках будет превышать время работы на ячейках с меньшим количеством частиц. В такой ситуации необходимо провести динамическую балансировку нагрузки узлов — распределить данные с более загруженных узлов на менее загруженные.

Для этого необходимо определить, на какие узлы и какие части массива частиц необходимо перенести, при этом необходимо учитывать загрузку узлов и топологию коммуникаций для наиболее эффективного распределения. Для дальнейшего повышения

эффективности при передаче больших объемов данных, или при ожидании данных одной ячейки с нескольких узлов, можно также производить вычисления над доступными данными одновременно с обменом/ожиданием данных с других узлов.

Исходя из рассмотренных требований метода PIC, для хранения множества частиц можно применить динамические массивы (ДМ), которые должны обладать следующими свойствами:

- поддержка пользовательских типов данных;
- массив представляет собой неупорядоченное множество элементов;
- возможность добавлять/удалять элементы массива;
- возможность изменять доступный объем памяти для хранения элементов массива;
- поддержка распределения частей массива по разным узлам/процессам;
- необходимость обращения к нелокальным для процесса данным массива отсутствует. Обработка элементов массива может осуществляться независимо.

В случае, когда вычисления производятся над элементами массива индивидуально, как например в случае с расчетом перемещения частиц под действием поля, такой динамический массив можно разбить на части и распределить вычисления над этими частями по разным процессам. Таким образом, применение динамических массивов позволит осуществить в ряде задач поддержку динамической балансировки прозрачной для пользователя.

Как видно из примера метода PIC, реализация динамических массивов является достаточно сложной задачей, поэтому актуальна задача разработки инструментальных средств, облегчающих реализацию динамических массивов в прикладных программах.

Данная работа посвященна разработке модуля для реализации динамических структур данных с указанными выше свойствами для использования в качестве библиотеки и в составе системы LuNA, который бы скрывал в себе сложности реализации таких массивов.

1.2 Анализ средств параллельного программирования

Рассмотрим средства параллельного программирования для систем с распределенной памятью и их возможности по реализации динамических массивов применительно к задачам численного моделирования, использующих метод PIC.

1.2.1 Message Passing Interface

Message Passing Interface (MPI) является стандартизованным интерфейсом передачи

сообщений [3]. MPI определяет ряд функций, позволяющих передавать и принимать сообщения, осуществлять групповые операции (суммирование, минимизация и т.п.) и синхронизацию. Операции могут выполняться как синхронно, так и асинхронно. Коммуникации осуществляются в рамках т.н. «коммуникатора» - подмножества имеющихся процессоров и связей между ними.

В MPI используется модель передачи сообщений – работает несколько параллельных процессов с раздельной памятью, которые обмениваются сообщениями.

MPI широко используется в большинстве вычислительных кластеров. Он не предоставляет ничего кроме реализации интерфейса и является своего рода «ассемблером» параллельного программирования в том смысле, что реализует лишь низкоуровневые коммуникации, оставляя за пользователем всю работу по управлению работой процессов, коммуникаций и распределением данных между процессами.

MPI является слоем виртуализации аппаратного обеспечения и автоматически распределяет процессы по имеющимся процессорам. Это распределение сохраняется во времени, то есть динамической балансировки нагрузки не происходит. Передача сообщений на фоне вычислений поддерживается, но не автоматически. В целом MPI – достаточно низкоуровневое средство и, так как управление распределенными данными пользователю приходится осуществлять самостоятельно, не предоставляет достаточно удобного и эффективного по затратам времени на разработку способа реализации решения задач численного моделирования.

1.2.2 High Performance Fortran

High Performance Fortran (HPF) – это расширение языка Fortran-90, дополняющее его новыми операторами для параллельной работы с массивами [4]. HPF позволяет осуществлять параллельную обработку массивов как в системах с общей, так и распределенной памятью. Программист может задавать рекомендуемый способ разбиения, объявлять «чистые» (от побочных эффектов[1]) процедуры, осуществлять вызовы процедур, использующих собственные библиотеки коммуникаций (таких как MPI).

Таким образом, HPF предоставляет средства для параллельной работы с массивами и совместного использования других средств параллельного программирования. HPF, как таковой, не является универсальным средством параллельного программирования, а лишь позволяет в указанных случаях обрабатывать данные параллельно. Для указанного круга

[1] Функция называется функцией без побочных эффектов, если она не использует глобальных ресурсов (переменные, устройства ввода/вывода и т.п.). Функции без побочных эффектов могут безопасно работать параллельно.

задач HPF обеспечивает необходимые прикладные свойства программ.

HPF позволяет пользователю не беспокоится о написании низкоуровневых коммуникаций между процессами, однако задача распределения данных по процессам остается на пользователе. Таким образом, для задач численного моделирования, HPF предоставляет пользователю возможность не беспокоится о низкоуровневым управлением коммуникаций между процессами, однако остальные задачи распределения данных по процессам остаются на пользователе.

1.2.3 Системы с распределенной общей памятью

Модель DSM/PGAS (Distributed Shared Memory/Partitioned Global Address Space) представляет собой модель памяти, в которой существует глобальное адресное пространство с заданными участками, локальными индивидуально для отдельных процессов. Примерами систем, реализующих данную модель, являются системы параллельного программирования Unified Parallel C и Co-Array Fortran [5,6].

В данных системах пользователь явно задает распределение данных между процессами. Коммуникации обращений в нелокальную память происходят неявно для пользователя, однако соответствующие задержки коммуникаций сохраняются.

Системы параллельного программирования на основе модели DSM облегчают пользователю задачу ручного распределения данных между процессами, однако, так же как и в случае с HPF, многие задачи управления данными, специфические для численного моделирования, остаются на пользователе.

1.2.4 KeLP

Система программирования Kernel Lattice Parallelism (KeLP) предназначена для упрощения реализации научных задач на сетках [7]. Поддерживается динамическая балансировка за счет перемещения атомарных кусков сетки. Данные представляются в виде множества блоков, на которых определяется граф зависимостей. Данные будут размещены по процессорам автоматически в соответствии с заявленными зависимостями. Программа пишется на двух уровнях – KeLP для описания структур данных и параллелизма и Си или Фортран для реализации численных вычислений.

KeLP обладает развитым инструментарием в области сеточных задач. В частности, есть возможность динамической детализации сетки, а динамическая балансировка нагрузки на процессоры осуществляется передачей атомарных фрагментов сетки. Тем не менее, из-за своей узкой специализации, удобство управления распределением данных в KeLP ограничено сеточными задачами.

1.2.5 Charm++

Система программирования Charm++ предназначена для создания асинхронных параллельных программ для систем с общей и распределенной памятью (с поддержкой процессоров Cell и GPU) [8]. Она включает язык (расширение языков C++, Fortran 90), компилятор, runtime-систему, а также множество подключаемых библиотечных модулей, реализующих системные функции, такие как различные стратегии коллективных обменов данными (рассылка, редукция), алгоритмы динамической балансировки загрузки, работа с контрольными точками.

Программа на Charm++ представляет собой распределенное множество объектов (инкапсулирующих в себе и данные, и код), называемых chare, которые взаимодействуют между собой посредством передачи сообщений. При написании Charm++ программ, пользователь может не думать о том, какая сеть используется для коммуникаций между узлами мультикомпьютера и сколько узлов доступно для работы программы. Задачи по отображению данных программы и осуществлению коммуникаций между chare объектами решает исполнительная система. Любой chare объект может обращаться к любому другому chare объекту программы, даже находящемуся на другом узле. При этом сохраняются задержки, связанные с проведением коммуникаций. В системе Charm++ можно создавать массивы chare объектов, элементы которых исполнительная система может динамически распределять по вычислительным узлам в соответствии с их нагрузкой. Кроме того, можно привязывать элементы одного массива к соответствующим элементам другого массива chare объектов, так называемые Shadow Arrays (при этом элементы без соответствующих элементов в другом массиве могут перемещаться свободно). Для поддержки миграции chare объектов между узлами, например при динамической балансировке, пользователь должен самостоятельно запрограммировать функцию упаковки/распаковки данных chare объекта в буфер для пересылки.

В целом, Charm++ облегчает задачу управления распределенными данными при реализации решений задач численного моделирования, однако реализация операций с распределенными данными не является достаточно удобной.

1.2.6 Результаты анализа

Анализ средств параллельного программирования показал, что для решения задач, использующих метод PIC и аналогичных удовлетворительного решения нет. Задача управления распределением данных остается на пользователе. В особенности это касается таких универсальных и распространенных решений, как MPI. Остальные средства

параллельного программирования не предоставляют поддержку автоматического динамического перераспределения данных и управлением распределенных данных, которая была бы достаточно удобной для применения при решении численных задач методом PIC.

1.3 Цель работы

Требуется разработать в рамках Технологии Фрагментированного Программирования и системы фрагментированного программирования LuNA [10] программный модуль поддержки динамических массивов, управляющий распределением данных массива по процессам, а также необходимые операции над локальными данными, отвечающий следующим требованиям:

- интерфейс управления локальными данными:
  - операции добавления/удаления элементов;
  - возможность изменять доступный объем памяти для локальных данных;
  - операция дефрагментации памяти;
  - максимальная производительность работы с локальными данными.
- интерфейс управления распределением данных:
  - операции управления распределением данных массива по заданным узлам;
  - хорошая масштабируемость и производительность управления распределением данных в условиях большого количества данных массива и задействованных узлов для их хранения.
  - переносимость модуля для использования в различных версиях исполнительной системы и в качестве самостоятельной библиотеки.

В главе 1 рассмотрены основные понятия технологии фрагментированного программирования и предлагаемое решение по реализации поддержки динамических массивов в рамках системы фрагментированного программирования LuNA. В главе 2 описаны результаты реализации модуля ДМ.

2 Модуль поддержки динамических массивов

2.1 Технология фрагментированного программирования

Технология фрагментированного программирования является технологией параллельного программирования, реализующей сборочный подход [9] и ориентированной на задачи численного моделирования. Эта технология разрабатывается в ИВМиМГ для поддержки конструирования параллельных программ.

Основной задачей технологии фрагментированного программирования является конструирование параллельных программ численного моделирования, генерация кода и автоматическое обеспечение динамических свойств параллельной программы. К таким свойствам относятся динамическая балансировка загрузки, настройка на ресурсы параллельного компьютера или реализация коммуникаций параллельно с вычислениями.

Для решения этой задачи программа собирается из фрагментов данных и фрагментов вычислений. Такая фрагментация сохраняется и во время выполнения программы. Это позволяет манипулировать программой во время исполнения – например, перемещать отдельные фрагменты данных или вычислений с процессора на процессор, изменять грануляцию вычислений и данных.

Исполнением фрагментированной программы (ФП) занимается исполнительная система (ИС), которая реализует системную составляющую программы и автоматически обеспечивает ее динамические свойства. Для эффективной реализации программ исполнительная система должна учитывать специфику предметной области, т.е. быть специализированной. В технологии фрагментированного программирования эта специализация – на задачах численного моделирования. Для таких задач характерны регулярные структуры данных – массивы, сетки; и преимущественно простые способы их обработки. Такие особенности представляются во фрагментированной программе явно, что позволяет эффективно исполнять ее автоматически.

При создании ФП, пользователь производит фрагментацию алгоритма программы и представляет ее в виде фрагментов кода (ФК), множества фрагментов данных (ФД), фрагментов вычислений (ФВ) и задает отношения информационной зависимости между ФВ. Фрагменты данных представляют переменные алгоритма. ФД — это блок памяти, в котором хранятся значения переменных. Он может хранить как значения базовых типов (целочисленный, вещественный), так и структурированных типов (массивы, структуры). ФК — это модули, реализующие некоторые функции над ФД. ФВ — это исполняемая единица, обрабатывающая заданные ФД, то есть выполнение ФК на заданных ФД.

Описание ФП производится на языке высокого уровня LuNA, реализация ФК — на языке C++. Кроме того, пользователь может задавать рекомендации по выполнению ФП для ИС, такие как приоритет выполнения ФВ, отношение соседства для ФД, и шаблон исполнения ФВ, что позволяет оптимизировать исполнение ФП.

2.2 Анализ задачи

Динамический распределенный массив в системе фрагментированного программирования — это множество ФД, отображенное на множество узлов мультикомпьютера. При выполнении ФВ имеет доступ только к выделенному ему локальному ФД ДМ, остальные ФД ДМ не доступны для использования и о них ФВ ничего не известно.

Над динамическим массивом определены локальные операции, такие как добавление/удаление элементов, которые производятся над ФД ДМ, и глобальные операции, например перераспределение элементов по заданным узлам, получение глобальной информации о ДМ (например, полное количество элементов в ДМ). Добавление/удаление элементов может происходить в любой ФД ДМ.

При выполнении операции перераспределения ДМ между узлами могут перемещаться как целые ФД так и отдельные элементы ФД, заранее не известные. Эта операция происходит прозрачно для пользователя. Алгоритм перераспределения в общем виде выглядит так: осуществляется сбор данных о текущем состоянии ДМ, таких как количество элементов на узлах группы ДМ; затем происходит поиск наилучшей стратегии размещения данных ДМ на новые выделенные узлы для его хранения; после чего происходит перемещение ФД ДМ со старых узлов на новые согласно принятой стратегии.

При поиске наилучшей стратегии необходимо учитывать несколько факторов, такие как нагрузка новых узлов, а также топология коммуникаций. Первый фактор необходимо учитывать в задачах где требуется проводить динамическую балансировку нагрузки, например в методе PIC, когда ДМ частиц одной ячейки распределяется на узлы, на которых происходит вычисления над другими ДМ частиц из других ячеек. Для эффективной балансировки нагрузки таким образом необходимо учитывать количество элементов других ДМ на этих узлах, и отдавать на новые узлы столько элементов перераспределяемого ДМ, чтобы количество работы на всех узлах было одинаковым.

Второй фактор необходимо учитывать для снижения задержек на коммуникации между узлами. Рассмотрим ситуацию, представленную на рисунке 1.1. Узлы A,B,C,D соединены в кольцевую сеть. Необходимо перераспределить массив, находящийся на

узлах A,B,C, на узлы B,C,D. Возможны несколько стратегий перераспределения элементов. На рисунке 1.1а представлен вариант с перемещением всех элементов массива, находящихся на узле A, на узел D. Происходит только одна транзакция, максимальное количество элементов массива остаются на своих местах. Однако такая стратегия для данной топологии может быть не достаточно эффективна. На рисунке 1.1б представлено другое решение: данные с узла A перераспределяются на узлы B и C, а определенный объем данных с узлов B и C перераспределяется на узел D. Таким образом осуществляется четыре транзакции между соседними узлами. В зависимости от объема данных на узлах, загруженности коммуникаций между данными узлами, и задержек при осуществлении транзакций, такая стратегия может быть значительно более эффективной в данной топологии.

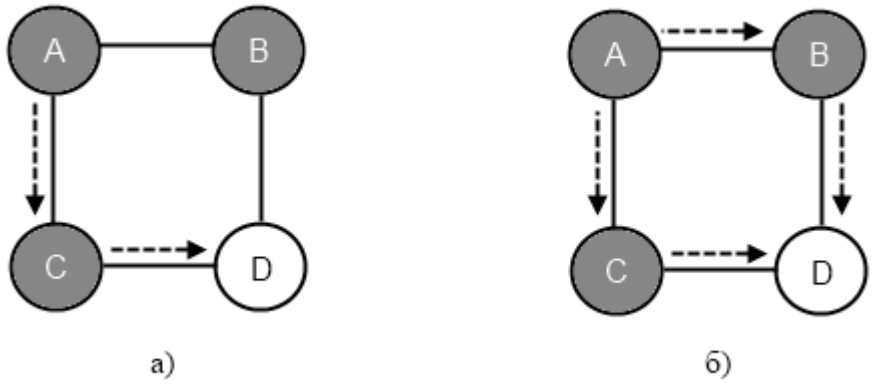

Рисунок 1.1. Пример различных решений задачи о перераспределении данных с узлов A,B,C на узлы B,C,D. а) перенос всех данных с узла A на узел D. б) перенос данных с узла A на узлы B и C, перенос части данных с узлов B и C на узел D.

Осуществление глобальных операций над ДМ возможно несколькими схемами управления распределением данных: централизованной, децентрализованной и иерархической. У каждой из этих схем есть свои преимущества и недостатки.

В централизованной схеме управление ДМ осуществляется из одного узла. Это позволяет получить полную информацию о распределении данных из одного источника и упрощает реализацию данной схемы, все глобальные операции над ДМ осуществляются просто. Однако при большом количестве узлов и данных, задержки на управление и коммуникации могут значительно понизить производительность, кроме того

увеличивается нагрузка на центральный узел ДМ.

В децентрализованной схеме нет центрального управления, поэтому исчезают многие задержки связанные с централизованной схемой. Решения принимаются и реализовываются локально, однако при необходимости наличия информации о всем массиве возникает проблема поддержания когерентности этой информации между всеми узлами ДМ, задержки на сбор такой информации со всех узлов, кроме того значительно усложняется реализация такой схемы.

В иерархической схеме управляющие узлы объединены в дерево иерархий по принципу локальности в топологии — управляющие узлы самого низкого уровня отвечают за узлы ДМ, близкие к ним в топологии, и управляются узлами верхнего уровня. Управляющий узел более верхнего уровня собирает информацию и управляет узлами более низкого уровня, он располагается в топологии так, чтобы коммуникациями со всеми подчиненными узлами были наиболее эффективны. Таким образом, корневой узел находится в центре сети коммуникаций узлов ДМ и может собирать полную информацию о состоянии ДМ с управляющих узлов более низкого уровня. Данная схема позволяет решать проблемы децентрализованной и централизованной схемы, такие как необходимость опрашивать все узлы и поддерживать когерентность данных на всех узах в децентрализованной схеме, и растущую нагрузку на центральный узел в централизованной схеме, однако реализация иерархической схемы является более сложной задачей.

Для выбора стратегии перераспределения данных ДМ необходимо собрать актуальную информацию о состоянии его ФД со всех его узлов. Существуют различные варианты решения данной задачи, например сбор информации с узлов ДМ только при необходимости, либо постоянное обновление информации по завершению задач, работающих с ФД ДМ. В зависимости от реализации системы в которой используется ДМ, может быть целесообразно применить тот или иной подход. Например, если информация о состоянии ДМ требуется только при выполнении каких-то глобальных операций, и перед сбором этой информации следует запрос к узлам ДМ, требующий ответа. Примером такой операции может служить запрос на блокирование ФД ДМ на использование вычислительными задачами на время проведения перераспределения. Он может требовать ответного подтверждения по завершению блокировки, в которое можно встроить информацию о текущем состоянии для устранения лишних передач информации между узлами. В данном случае постоянное обновление информации о состоянии не требуется.

ФД ДМ массива может представляться несколькими способами, например как блок

памяти, в котором хранятся элементы ДМ, или связный список из элементов ДМ. Они принципиально отличаются по производительности и способу работы пользователя с ДМ. В случае с использованием связного списка удаление и добавление элементов в ФД ДМ происходит при вызове соответствующих методов немедленно — в список просто добавляется еще один элемент, или удаляется существующий с простой модификацией связей соседних с ним элементов. Однако при этом падает производительность работы с ДМ, а также эффективность использования памяти. При частом добавлении/удалении элементов растет количество дорогостоящих вызов операций выделения памяти, кроме того выделенные области памяти могут располагаться в непоследовательных областях, из-за чего может неэффективно использоваться кеш процессора. Кроме того, для доступа к элементу ДМ необходимо проводить дополнительные операции или вызовы соответствующих функций. Если в ФД хранятся типы данных небольшого размера, например базовые типы такие как double, то для каждого элемента дополнительно расходуется память на хранение указателей на соседние элементы, что может отнимать необходимое место в памяти, которое можно было бы использовать для хранения большего количества элементов — это может быть важно при больших объемах данных ДМ.

Реализация ДМ блоками памяти является наиболее производительным решением. В данной схеме ФД представлен как целый блок памяти, используемый как контейнер с заданным объемом для хранения элементов ДМ. Доступный объем контейнера можно изменять стандартными операциями работы с памятью. Необходимо хранить информацию о количестве элементов ДМ в контейнере для осуществления операций с ФД ДМ. При использовании такого контейнера усложняется работа с ФД — операция добавления требует наличия достаточного свободного места в контейнере для добавления нового элемента и может требовать дополнительного вызова операции увеличения доступного объема памяти, а осуществление удаления элемента из ФД ДМ требует проведения дефрагментации блока памяти ФД. Данная схема с точки зрения пользователя может быть менее удобна для работы с ДМ, однако появляется возможность дать пользователю прямой доступ к указателю на блок памяти ФД ДМ, поэтому при работе пользовательского кода можно полностью избавиться от дополнительных задержек на вызовы операций работы с элементами ДМ.

Для работы с локальными данными также существует две схемы, отличающиеся возможностью выполнения параллельных задач над частями ДМ на данном узле: схема с одним ФД на узел или с несколькими ФД. В случае с несколькими ФД ДМ на узел можно

выполнять задачи над свободными ФД, в то время как другие ФД заняты другими задачами, что позволяет например проводить вычисления на фоне коммуникаций. Кроме того, при реализации ФД ДМ блоками памяти, разбиение данных ДМ на несколько ФД позволяет эффективнее осуществлять операции по выделению памяти с более мелкими блоками. Данная схема требует дополнительных накладных расходов на управление несколькими ФД, которые могут понизить производительность при достаточно небольших объемах данных ДМ на узле. Схема с одним ФД в данном случае позволяет избавится от накладных расходов и является более простой в реализации, однако с увеличением объемов данных она значительно проигрывает схеме с несколькими ФД по эффективности выделения памяти, обмену данными с другими узлами и задержками связанными с ожиданием готовности данных к обработке.

2.3 Описание предлагаемого решения

В данной работе предлагается реализовать ДМ, используя централизованную схему управления с одним ФД на узел, со стратегией распределения без учета топологии и нагрузки узлов – эти улучшения планируется провести в последующих работах на данную тему. ФД предлагается реализовать как блоки памяти, указатели на которые доступны пользователю.

Для увеличения производительности работы с ФД ДМ, операция дефрагментации производится только по завершении работы пользовательского кода с данным ФД. При вызове операции удаления элемента из ФД ДМ, выбранные элементы помечаются на удаление, но остаются в памяти до завершения работы ФВ, в котором они были помечены. После этого автоматически при необходимости производится дефрагментация. Поэтому пользователь должен следить за тем, какие области памяти ФД были помечены на удаление и не использовать их до завершения ФВ.

Для управления и хранения данных о ДМ предлагается использовать дескрипторы: глобальный GD и локальный LD. Глобальный дескриптор располагается на одном центральном узле ДМ и хранит информацию о распределении ФД ДМ по узлам. LD хранит информацию о ФД ДМ, находящихся на данном узле. Локальный и глобальный дескрипторы обладают двумя состояниями – захвачен или свободен, что позволяет определить, обрабатывается ли уже дескриптор другой задачей или нет. Так, ЛД может быть захвачен на вычисления или на обработку запроса от ГД. ГД может быть захвачен пока не завершится перераспределение его ДМ на новые узлы.

Для управления ДМ используется сообщения, передаваемые между ГД и ЛД

используя внешний интерфейс передачи сообщений. Это позволяет при реализации данного интерфейса эффективнее осуществлять передачу данных между узлами, объединяя данные, отправляемые на один и тот же узел, в одну транзакцию, что может значительно повысить производительность.

Информация о состоянии ДМ, хранящаяся в ГД, обновляется только по необходимости наличия актуальной информации, например при перераспределении данных массива. Это позволяет избежать лишних транзакций.

Разработан следующий алгоритм перераспределения данных ДМ:

1. При поступлении запроса на перераспределение ДМ, ГД блокируется для всех операций, кроме операций перераспределения ДМ, и отправляет запрос на обновление информации о состоянии всем ЛД. Если ЛД уже был захвачен на выполнение другой операции, то ответ отправляется сразу после завершения текущей операции, захватившей ЛД, затем ЛД блокируется для всех операций, кроме обработки запросов ГД.
2. На основании актуальной информации, собранной с ЛД, ГД вычисляет новое распределение ДМ, и отсылает соответствующие задания о переносе данных ЛД. При необходимости ГД переносится на новый узел.
3. По завершении перераспределения на новый узел, ЛД отправляет ГД подтверждение о готовности, и блокировка ЛД снимается.
4. При получении подтверждения о готовности от всех новых ЛД, с ГД также снимается блокировка.

Так как в данном алгоритме используется блокировка ГД и ЛД, данные ДМ защищены от проведения сторонних операций при его перераспределении.

Ниже представлен алгоритм выбора нового распределения элементов ДМ. На вход алгоритма подается массив индексов текущих узлов и количества элементов на них — current_nodes_map, и массив индексов узлов на которые необходимо распределить элементы — future_nodes.

1. находится требуемое количество элементов на узел, такое, чтобы на каждом узле из future_nodes было одинаковое количество элементов (оставшиеся элементы от деления с остатком количества элементов на количество будущих узлов будут размещены на последнем узле в списке future_nodes).
2. формируются списки избыточного количества элементов на текущих узлах и требуемого количества элементов на будущих узлах (эти два множества могут пересекаться). Списки сортируются по убыванию количества элементов.

3. В цикле по сортированному списку избыточных элементов происходит расчет заданий для пересылки так, чтобы переслать максимальное количество элементов из текущего узла с избытком в первый еще незаполненный узел из сортированного списка узлов с недостатком.

Данный алгоритм обладает существенными ограничениями, так как он не учитывает при расчете задач для перераспределения нагрузку узлов, топологию коммуникаций между узлами и задержки на пересылку данных.

3 Реализация модуля динамических массивов

3.1 Описание реализации

В рамках дипломной работы был реализован модуль динамических массивов как самостоятельная библиотека на языке C++. Главный дескриптор ДМ реализован как класс GD, локальный дескриптор — как класс LD. Кроме того, в модуле реализованы классы с виртуальными методами для работы с внешними интерфейсами отправки сообщений и организации вычислений над ДМ. Все реализованные методы данного модуля не являются потоково-безопасными, поэтому задача обеспечения корректного управления использованием методов и данных дескрипторов в многопоточной исполнительной системе лежит на самой ИС.

Управление распределением данных осуществляется с помощью передачи сообщений между ГД и ЛД, используя внешний интерфейс обмена сообщениями (коммуникаций) для отправки, и функции recv локального и глобального дескриптора для обработки принятых сообщений. Сообщения состоят из идентификатора типа сообщения, заголовка сообщения с необходимой информацией и области данных, в которой, например, хранятся элементы ДМ при миграции их на другой узел. Для отправки сообщений предназначается метод send2DDA внешнего интерфейса, который принимает в качестве параметров идентификатор узла-получателя, указатель на буфер сформированного для модуля динамических массивов сообщения и количество байт в нем. Метод recv аналогично принимает в качестве параметров идентификатор отправителя и указатель и размер полученного сообщения.

Для определения, какому ДМ принадлежат ГД и ЛД используется уникальный глобальный идентификатор GID. Исполнительная система хранит таблицу связи объектов классов GD и LD с глобальными идентификаторами ДМ, которая используется в частности при определении получателя сообщения при обмене сообщениями между глобальным дескриптором и локальным.

3.2 Класс глобального дескриптора GD

Класс ГД реализует интерфейс управления распределением данных массива. Он содержит ассоциативный массив el_map, содержащий идентификаторы узлов и количество элементов на этих узлах и предоставляет метод recv для получения и обработки сообщений, а также метод rearrange который запускает перераспределение данных. Так как процесс перераспределения может занимать долгое время, он был реализован с использованием событийной модели — запуск процесса осуществляется из потока ИС,

дескрипторы переводятся из одного состояния в другое при получении сообщений друг от друга, и поток ИС не блокируется на время передачи сообщений и ожидания ответа. Рассмотрим данный процесс подробнее:

1. Поток ИС вызывает метод GD::rearrange у ГД заданного ДМ с указанием узлов, на которые необходимо перераспределить массив. Устанавливается флаг reloc_flag дескриптора – он позволяет другим потокам ИС определить состояние дескриптора. Происходит рассылка сообщений о начале перераспределения всем локальным дескрипторам ДМ, используя метод send2DDA внешнего интерфейса, а также инициализация счетчика reloc_num_notready ГД, значение которого ставится равным текущему количеству узлов ДМ. В случае если флаг reloc_flag уже был установлен, или если счетчик reloc_num_notready не равен нулю, функция rearrange возвращается срезу с кодом ошибки. Таким образом, эти две переменные реализуют механизм захвата глобального дескриптора.
2. При получении сообщения о начале перераспределения, поток ИС вызывает метод recv локального дескриптора данного ДМ. Метод recv определяет тип сообщения и устанавливает флаг GDLock ЛД. В случае, если локальный дескриптор блокирован на вычисления (установленный флаг busy LD) то метод recv сразу же возвращает управление вызвавшему его потоку. В дальнейшем, при вызове метода release после дефрагментации происходит проверка флага GDLock и соответствующее отправление сообщения с отчетом о состоянии ЛД глобальному дескриптору, включающее в себя количество элементов и объем памяти ФД ЛД. Если во время вызова метода recv флаг busy ЛД не был установлен (то есть не производились вычисления или другие операции над ФД ЛД), то отправление сообщения с отчетом производится сразу же из метода recv.
3. При получении сообщений от ЛД для ГД, происходит вызов метода recv ГД. При получении каждого отчета массив el_map обновляется актуальными значениями и декрементируется счетчик reloc_num_notready, после чего проверяется его значение. Если оно равно 0, то происходит расчет нового распределения и составление задач миграции данных массива на новые узлы функцией GD::calcDist. Так как в данной реализации не учитывается топология коммуникаций, то GD будет перемещен на первый в списке новый узел, если среди новых узлов нет узла, на котором GD располагается в данный момент. Происходит рассылка сообщений с задачами миграции всем текущим локальным дескрипторам данного массива. Так как в данной реализации используется схема с единственным ФД ДМ на узел, в

заголовке сообщения передается информация о полном количестве транзакций (частей) нового ФД, а также идентификатор узла, на котором будет находится GD после его миграции.

4. При получении сообщения с задачей миграции, метод recv формирует сообщение передачи данных из заголовка сообщения и копией n заданных элементов ФД. Счетчик элементов LD уменьшается на n. После чего происходит передача сообщения на указанный узел. Если в LD не осталось элементов, освобождается занимаемая им память.
5. При получении сообщения с данными на новом узле, ИС проверяет наличие экземпляра LD с указанным GID в сообщении, если такого объекта нет, то он инициализируется с размером блока памяти, необходимого для размещения всех элементов ДМ, мигрирующих на данный узел. Счетчик remaining_parts LD инициализируется количеством всех транзакций на данный узел. При получении каждого сообщения с данными для ФД вызывается метод recv и производится копирование элементов в блок памяти, а также декрементация и проверка счетчика remaining_parts. Если счетчик достигает 0, отсылается сообщение о готовности ГД, и ФД становится готовым для использования, например, для проведения вычислений.
6. При получении сообщения с глобальным дескриптором на новом центральном узле ДМ, ИС инициализирует объект GD из переданных данных, после чего инициализируется счетчик reloc_num_notready количеством новых узлов ДМ. Поток ИС вызывает метод recv глобального дескриптора для каждого сообщения из очереди уже принятых ее интерфейсом сообщений и последующих для ГД с данным GID, счетчик декрементируется и проверяется для каждого сообщения. Если он равен 0, то GD становится валидным и операция rearrange может быть снова вызвана.

3.3 Класс локального дескриптора LD

Класс ЛД является реализацией интерфейса управления ФД ДМ. ФД представляется в виде целого блока памяти, указатель на который доступен пользователю. ФД может быть захвачен для эксклюзивного использования вычислительными задачами ИС, или задачами перераспределения данных ДМ. На каждом узле может быть только один ЛД с данным GID.

При инициализации LD выделяется блок памяти заданного размера, который

служит контейнером для элементов ФД и указывается размер элемента ФД. Размер блока памяти можно менять вызовом метода resize.

Для определения состояния LD используются флаги busy, GDLock, dirty и счетчик remaining_parts. Установленный флаг busy указывает на захват данного LD какой-либо задачей, например, выполнением пользовательского кода. Флаг GDLock указывает на то, что данный LD захвачен глобальным дескриптором для проведения перераспределения. Флаг dirty устанавливается при вызове метода удаления элемента remElement, и снимается при проведении дефрагментации. Счетчик remaining_parts определяет количество частей ФД данного LD, которые еще не были получены при перераспределении ДМ.

Захват LD исполнительной системой осуществляется вызовом его метода capture. Данный метод проверяет состояние флагов busy, GDLock и счетчика remaining_parts. Если хотя бы один из флагов установлен или счетчик не равен нулю, то захват невозможен — LD занят другой задачей – и метод сразу же возвращает ошибку. В противном случае, производится захват LD установкой флага busy. Захват на проведение перераспределения GD осуществляется неявно при получении методом recv сообщения о начале перераспределения.

Освобождение LD от захвата осуществляется вызовом его метода release. В данном методе происходит проверка состояния флага dirty и выполнения дефрагментации если он установлен. Если установлен флаг GDLock то отсылается сообщение о готовности главному дескриптору. После чего снимается флаг busy.

Рассмотрим методы LD, которые исполнительная система может предоставить пользователю для использования в его коде:

- void *addElements(size_t num). Данный метод добавляет num элементов к счетчику элементов numel и возвращает указатель на начало блока добавленных элементов. Пользователь должен сам проинициализировать новые элементы, используя указатель. При невозможности добавить указанное количество элементов из-за недостатка доступного объема блока памяти LD, возвращает NULL.
- void *addElement(void *el). Добавляет единственный элемент, на который указывает el, в LD, и возвращает указатель на добавленный элемент в блоке памяти LD. При невозможности добавить элемент возвращает NULL.
- void remElement(size_t idx). Данный метод помечает элемент с выбранным индексом для удаления. Выбранный элемент остается в памяти до вызова операции дефрагментации при вызове метода release исполнительной системой.

- void *resize(size_t new_size). Изменяет доступный объем блока памяти LD на new_size. Возвращает указатель на новый блок памяти, все прошлые указатели становятся невалидными. В случае ошибки возвращает NULL.
- size_t getMaxNumel(). Возвращает максимальное количество элементов, которое может быть размещено в блоке данных LD с текущим размером.
- size_t getNumel(). Возвращает текущее количество элементов LD.
- size_t getMaxSize(). Возвращает текущий доступный объем блока памяти LD в байтах.
- size_t getElsize(). Возвращает размер элемента LD в байтах.

Для дефрагментации был разработан следующий алгоритм дефрагментации памяти без сохранения порядка:

1. Из массива индексов удаляемых элементов ФД ДМ remIds удаляются идентичные элементы и он сортируется в возрастающем порядке
2. В цикле по элементам remIds на место текущего удаляемого элемента перемещается последний элемент ФД.

Кроме того, был разработан следующий алгоритм с сохранением порядка:

1. Из массива remIds удаляются идентичные элементы и он сортируется в убывающем порядке
2. В цикле по элементам remIds обнаруживается начало блока удаляемых элементов, в который входит текущий элемент remIds. Текущий последний целый блок элементов, не помеченных к удалению, сдвигается на найденное начало.

Алгоритм с сохранением порядка является более медленным чем алгоритм без сохранения порядка, однако его применение может быть необходимо в ситуациях, когда важно сохранять порядок элементов в динамическом массиве.

3.4 Особенности и ограничения реализации

В данной реализации возможна существенная задержка операций над ДМ по причине использования только одного атомарного фрагмента ДМ на узел. Например, в ситуации когда не все части фрагмента при перераспределении пришли на данный узел, нельзя начать вычисления над той частью фрагмента ДМ, что уже находится на узле. Данную проблему планируется решить поддержкой локальных дескрипторов с несколькими фрагментами, которые можно обрабатывать независимо.

Остается нерешенной задача нахождения оптимального распределения динамического массива в зависимости от загрузки узлов и топологии суперкомпьютера.

Решение этой задачи требует отдельного исследования.

3.5 Тестирование

Для тестирования работоспособности разработанного модуля была реализована тестовая исполнительная система с двумя потоками, один — управляющий, осуществляющий обмен сообщениями между узлами, и один — вычислительный, в котором происходит выполнение тестов над локальными данными ДМ (т. е., проверка работоспособности интерфейса пользователя ДМ). Было выполнено три теста:

1. проверка пользовательского интерфейса ДМ. Данный тест состоит из проверки работы функций пользовательского интерфейса ДМ, рассмотренных в главе 2. Тест запускается на одном процессе, без перераспределения данных. Создается объект класса LD заданного размера, и в вычислительном потоке тестовой ИС инициализируются элементы ФД ДМ данного LD, после чего проверяется корректность функций добавления элементов, функции получения информации о состоянии ФД, а также функции дефрагментации при различных комбинациях удаляемых элементов.
2. проверка перераспределения ДМ на новое множество узлов без пересечений со старым множеством узлов ДМ. В данном тесте происходит создание объекта класса LD заданного размер на выбранном главном узле и инициализация его ФД. Затем происходит инициализация объекта класса GD с данным объектом LD в качестве ФД ДМ, после чего запускается процесс распределения данных LD на выбранную группу узлов, по завершении которой запускается перераспределение данных ДМ на группу узлов, не имеющую пересечений с первой выбранной группой узлов. Тест завершается перераспределением данных ДМ обратно на главный узел.
3. проверка перераспределения ДМ на множество узлов с пересечением со старым множеством. Данный тест аналогичен предыдущему, за исключением того, что имеются пересечения между новой и старой группами узлов в процессе перераспределения.

Проведенные тесты были проведены успешно, они показали полную работоспособность разработанного модуля. В зависимости от выбранных групп узлов для перераспределения и объема данных ДМ были подтверждены существенные задержки при использовании схемы управления ДМ с одним ФД на узел, а также задержки, связанные с центральным управлением ДМ.

4 Заключение

В данной работе была исследована проблема системной поддержки параллельной реализации динамических распределенных структур данных на примере динамического массива в методе PIC. Был выполнен аналитический обзор существующих средств программирования, применяемых для реализации подобных структур данных. Предложены и проанализированы распределенные структуры данных и параллельные алгоритмы их системной обработки. Был разработан базовый программный модуль поддержки динамических массивов с рядом важных ограничений, указанных выше.

На защиту выносятся:

1. Распределенные структуры данных и алгоритмы модуля поддержи динамических массивов.
2. Реализация предложенных алгоритмов и структур в виде программного модуля.

В дальнейшем планируется продолжить данную работу и усовершенствовать модуль, улучшив масштабируемость управления динамическими массивами, обчеспечить возможность производить вычисления на фоне передачи данных динамического массива путем разбиения локальных фрагментов динамического массива на части, которые могут обрабатываться независимо. Кроме того, планируется улучшить модуль принятия решений по оптимальному распределению фрагментов динамического массива на узлы суперкомпьютера в соответствии с их загрузкой и топологией коммуникаций.

Результаты данной работы были представлены на 50-й юбилейной Международной научной студенческой конференции «Студент и научно-технический прогресс» [11].

5 Список использованных источников

1. Малышкин, В.Э Параллельное программирование мультикомпьютеров. // Изд-во НГТУ, Новосибирск, 2006. – 296 с.
2. Kraeva, M.A., Malyshkin, V.E.: Implementation of PIC Method on MIMD Multicomputers with Assembly Technology. // HPCN Europe 1997, Springer Verlag, LNCS, Vol. 1277, 1997
3. Message Passing Interface (MPI) Forum Home Page // [Электронный ресурс]. Режим доступа: http://www.mpi-forum.org/, свободный (дата обращения: 10.10.2011).
4. The High Performance Fortran Forum (HPFF) // [Электронный ресурс]. Режим доступа: http://hpff.rice.edu/, свободный (дата обращения: 10.10.2011).
5. Unified Parallel C at George Washington University // [Электронный ресурс]. Режим доступа: http://upc.gwu.edu, свободный (дата обращения: 10.10.2011).
6. Co-Array Fortran // [Электронный ресурс]. Режим доступа: http://www.co-array.org,

свободный (дата обращения: 10.10.2011).

7. The KeLP Programming System // [Электронный ресурс]. Режим доступа: http://cseweb.ucsd.edu/groups/hpcl/scg/kelp/, свободный (дата обращения: 10.10.2011).
8. Charm++ // [Электронный ресурс]. Режим доступа: http://charm.cs.uiuc.edu, свободный (дата обращения: 10.10.2011).
9. Вальковский, В.А. Синтез параллельных программ и систем на вычислительных моделях. // Новосибирск: Наука. Сибирское отделение, 1988. – 128 с.
10. V.Malyshkin and V.Perepelkin. Optimization of Parallel Execution of Numerical Programs in LuNA Fragmented Programming System // In the Proceedings of the second Russia-Taiwan symposium on Method and tools of Parallel Programming Multicomputers (MTPP) // Springer Verlag, LNCS series, Vol. 6083, pp. 1-10, 2010
11. Никитин А.А. Реализация динамических массивов в системе фрагментированного программирования // Материалы 50-й Международной научной студенческой конференции ≪Студент и научно-технический прогресс≫: Программирование и вычислительные системы / Новосиб. гос. ун-т. Новосибирск, 2012. 49 с. ISBN 978-5-4437-0048-9