Гетерогенные вычислительные системы на основе объектно-атрибутной модели программирования

Салибекян С.М., Панфилов П.Б., Хакимуллин Е.Р.

В статье приводится описание применения объектно-арибутной архитектуры вычислительной системы для создания гетерогенных (состоящих из вычислительных узлов различной архитектуры) вычислительных систем

Object-attribute programming model implementation for design of heterogeneous computer system

Aminev D.A., Salibekyan S.M.

A new object- attribute architecture implementation for heterogeneous (consists of computation node of various architectures) distributed computer system

I. Перспективы гетерогенных вычислительных систем

В последнее время гетерогенным вычислительным системам уделяется большое внимание ввиду того, что системы, состоящие из вычислителей различной архитектуры, производят вычисления более эффективно, нежели однородные, состоящие из универсальных вычислителей: специализированные узлы, архитектура которых заточена для решения конкретной задачи, производят вычислительную работу значительно быстрее. Актуальность данного направления развития вычислительной техники подтверждает всплеск интереса к грид-технологияи и к архитектурам, сочетающим в себе классические (CPU) и графические процессоры (GPU) процессоры. Уже довольно широко применяется стандарт OpenCL, включающий в себя Си-подобный язык программирования и интерфейс программирования приложений (API). Компания Microsoft недавно анонсировал новый компилятор C++ AMP (accelerated massive parallelism) [1], который расширит возможности языков программирования C++ и Visual Studio по созданию приложений для распределенных гетерогенных ВС. А 12 июня 2011 года на конференции AMD Fusion'12 была представлена инициатива Heterogeneous System Architecture (HAS) Foundation, инициаторами которой выступили компании AMD, ARM, Texas Instruments, MediaTek и Imagination [2]. Целью данной некоммерческой организации является создание единой открытой промышленной архитектуры для гетерогенных ВС, в которые входят два и более процессоров на одном чипе.

Однако для того, чтобы теоретические преимущества гетерогенных вычислительных систем (ВС) реализовались на практике, такие системы должны обладать следующими качествами:

- масштабируемость;
- изоморфизм (способность программы сохранять свою целостность, когда вносятся изменения часть программы);
- простота программирования;
- программирование распределенной ВС как единого целого, а не программирование каждого вычислительного блока в отдельности и описание обмена информацией между ними;
- возможность перемещения программных блоков на различные вычислительные узлы, т.е. минимальная зависимость программы от топологии ВС, на которой она запускается;
- способность перераспределения вычислительных ресурсов (исполнительных устройств или процессорных ядер) между вычислительными задачами;
- простой протокол обмена информацией между вычислительными узлами;
- независимость работы ВС от скорости передачи данных по линиям связи;
- возможность автономного (без подключения реальной аппаратуры) имитационного моделирования ВС.

II. DATAFLOW И CONTROL FLOW ПАРАДИГМЫ ДЛЯ РЕАЛИЗАЦИИ ГЕТЕРОГЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Создать систему, обладающую почти всеми вышеперечисленными свойствами, весьма затруднительно. И основным препятствием на пути реализации эффективных гетерогенных систем является доминирование парадигмы control flow (управление вычислительным процессом с помощью потока команд), в которой алгоритм задается с помощью последовательности инструкций (команд). На аппаратном уровне данная концепция организации вычислительного процесса реализовалась в фон-неймановской архитектуре, а на программном – в декларативной парадигме программирования. Концепция control flow плоха тем, что операнды инструкций привязаны к ячейкам оперативной или регистровой памяти вычислительного узла и не могут быть доступными сразу с нескольких узлов. Ввиду же того, что систему команд одного вычислительного узла на аппаратном уровне невозможно подстраивать систему команд других узлов, затрудняется создание единого вычислительного пространства гетерогенной ВС. Декларативное программирование также привязывается к локальному адресному пространству вычислительного узла, не позволяя создавать распределенные ВС, и также затрудняет синхронизацию вычислений на распределенных гетерогенных ВС.

Более походящей для решения задачи построения гетерогенных ВС является парадигма dataflow (управление вычислительным процессом с помощью потока данных), в которой вычислительные операции активизируются по приходе всех необходимых для них операндов. Операнды передаются между вычислительными устройствами с помощью токенов – совокупность операнда и служебной информации (указание адресата, тип данных и т.д.). В отличие от control flow в данной парадигме не происходит привязки операндов к ячейкам локальной оперативной памяти и, соответственно, расширяются возможности по созданию распределенных ВС. Ввиду того, что операнды передаются по одиночке, а не в составе командного пакета, существенно снижается зависимость от архитектурных особенностей как аппаратной, так и программной частей ВС. Также облегчается синхронизация вычислений, т.к. вычислительные операции активизируются по приходу необходимого для выполнения операции данных.

Отличие двух подходов можно проиллюстрировать с помощью теории графов. Представим алгоритм с помощью потокового графа G(V,H) (рис. 1), где вершины V будут обозначать операции, производимые над данными, а дуги H – передачу данных от одного вычислительного узла графа другому (начало дуги обозначает выдачу результата вычисления, конец – приём операнда для другой команды). Пометим все вершины V и дуги H графа, причём дуги, обозначающие передачу одних и тех же данных разным «потребителям», будут иметь одинаковую метку. Для ВС, управляемых командами, на потоковый граф накладывается так называемый командный граф K(V,H) (штрих-линия на рис. 1), который задаёт последовательность выполнения команд, т.е. последовательность обхода узлов потокового графа. Классическая же команда – это не что иное, как способ описания вершины потокового графа алгоритма: номер (адрес) ячейки памяти, где находится команда, будет являться меткой данного узла; адреса ячеек памяти, где хранятся операнды и куда записывается результат вычислений, по сути, являются метками дуг потокового графа. Таким образом, память в фон неймановской архитектуре является не только местом хранения данных и команд, но и способом описания узлов и дуг потокового графа (для императивных языков высокого уровня метками передаваемых данных являются мнемоники переменных).

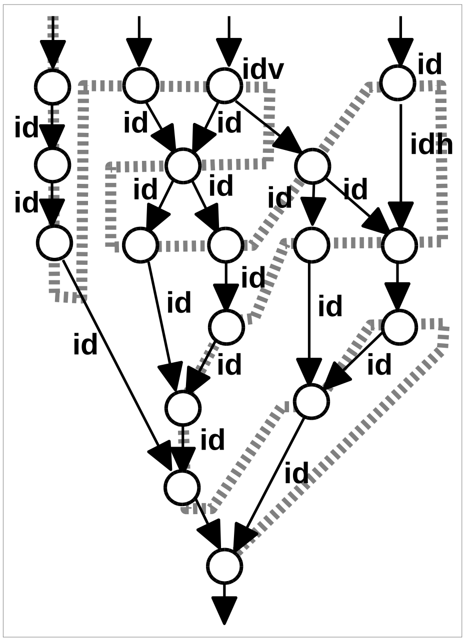

Рис. 1. Отличие control flow и dataflow парадигм

Однако в настоящее время парадигма dataflow не получила должного признания ввиду того, что до сих пор не было предложено удачных архитектур, способных конкурировать с классикой. Единственным исключением для программного уровня ВС является акторная модель программирования, на практике показавшая свою эффективность. Суть ее в следующем: вся программа разбивается на множество процедур, обменивающихся между собой данными, оформленными в виде токенов. Каждая такая процедура (актор) имеет интерфейс (описание входных и выходных данных) и тело, содержащее алгоритм обработки данных. Интерфейс объявляется заранее и в течение вычислительного процесса не изменяется. Связи между выходными и входными параметрами актора задаются с помощью специального языка (наподобие языка разметки XML) или с помощью графического редактора, когда программист с помощью мышки соединяет графические образы входных в выходных операндов акторов Link-ами. Вычисления в акторной модели активизируются по приходе комплекта данных, необходимого для выполнения вычислительной операции, что соответствует парадигме dataflow. Модель удобна для программирования гетерогенных ВС ввиду того, что, во-первых, в ней отсутствует привязка к локальной ОП вычислительного узла. Во-вторых, каждый актор является изолированным объектом и способ реализации логики его работы «скрывается» за его интерфейсом, что позволяет запускать акторы на вычислительных узлах различной архитектуры, обеспечивая вычислительное пространство, работающее по единым правилам. В качестве примера языков программирования, принадлежащих к данной парадигме, можно привести Caltrop и Erlang.

Акторная модель наиболее подходит для реализации гетерогенных ВС, однако ей присущи некоторые недостатки:

- связи между акторами, как правило, задаются перед запуском вычислительного процессе и во время вычислительного процесса не меняются;
- интерфейс актора не меняется во время вычислительного процесса, что существенно снижает гибкость организации вычислений;
- недостаточная абстракция данных (абстракция только на уровне процедур).

В качестве аналога предложенного нами решения также можно привести dataflow-архитектуру, разработанную в институте проблем проектирования в микроэлектронике (ИППМ) [3], которая весьма близка акторной модели, однако в отличие от классической модели создание акторов

и настройка информационных связей между ними производится не заранее, а непосредственно во время вычислительного процесса. Однако информационный обмен между акторами выглядит весьма громоздко: сбор комплекта данных производится с помощью системы, состоящей из нескольких блоком ассоциативной памяти, что требует создания специализированной ВС, не позволяет применять стандартные вычислительные средства и тем более реализовывать ее исключительно программными средствами.

III. Объектно-атрибутная архитектура для реализации гетерогенных вычислительных систем

Предлагаемая нами объектно-атрибутная (ОА) [4-6] архитектура наиболее близка к акторной модели, однако не имеет присущих ей недостатков. ОА-архитектура применима как к аппаратному, так и к программному уровням и в данной статье будет описан только программный способ реализации ВС АО-архитектуры. Итак, предложенная архитектура отличается от акторной модели в следующем. Во-первых, все программы-акторы (будем называть их функциональными устройствами (ФУ)) имеют универсальный интерфейс, через который операнды передаются и принимаются по одному. Операнд оформляется в виде милликоманды (простейший токен). Милликоманда состоит из двух полей: нагрузка (операнд или ссылка на информационную конструкцию) и атрибут операнда. Соответственно, в универсальном интерфейсе ФУ будут присутствовать два поля: атрибут (представляет собой целое число) и нагрузка (указатель на операнд). По атрибуту программа ФУ распознает пришедшие к нему по универсальному интерфейсу данные и соответствующим образом их обрабатывает. Промежуточные данные ФУ заносит в своей контекст – совокупность виртуальных регистров (при аппаратной реализации ФУ регистры будут «реальными»). Поэтому в универсальный интерфейс ФУ добавляется третье поле: указатель на контекст (контекст при программной реализации ОА-системы представляет собой запись (структуру), описывающую все операнды, которые необходимы для хранения промежуточных данных и результатов вычислений). Контекст задает состояние ФУ в текущий момент времени. ФУ начинает вычислительную работу по приходу к нему всех необходимых операндов, что соответствует парадигме dataflow.

ФУ могут создаваться и уничтожаться непосредственно во время вычислительного процесса, для чего существуют подпрограммы создания и уничтожения ФУ (наподобие конструктора и деструктора объекта в объектно-ориентированном программировании (ООП)). Причем, ФУ могут создавать и уничтожать другие ФУ. Подпрограмма создания выделяет память под контекст ФУ, производит инициализацию виртуальных регистров и возвращает указатель на контекст ФУ.

ФУ делятся на несколько типов (классов), в зависимости от логики своей работы. Тело (программа реализации логики работы ФУ), может быть написано на языке программирования низкого уровня или декларативном языке программирования высокого уровня, например, в уже реализованной системе ОА-программирования и моделирования, логика работы ФУ задается на языке Delphi. Алгоритм же работы всей ОА-системы задается посредством описания обмена данными между ФУ (ОА-образ) – для этой цели был разработан язык программирования (ОА-язык). ФУ могут быть как универсальными (например, арифметико-логическое устройства, устройство ввода-вывода и т.д.), так и специализированными на решении какой-либо вычислительной задачи. Совокупность типов ФУ образуют ОА-платформу, т.е. набор виртуальных устройств, с помощью которых можно описывать алгоритм решения задачи. При реализации распределенных ВС в ОА-платформу на конкретном вычислительном узле могут включаться лишь те типы ФУ, которые необходимы для реализации алгоритма работы данного вычислительного узла, что позволяет значительно экономить ресурсы и дает возможность включения в состав распределенной ВС узлов малой вычислительной мощности (например, микроконтроллеры). Следует заметить, что виртуальная машина (наподобие Java или Dot Net) должна быть запущена на вычислителе полностью, что порой делает невозможным ее запуск на маломощных вычислителях.

Обратите внимание, аппаратно зависимой будет только ОА-платформа, ОА-образ же независим, т.к. работает только с интерфейсами ФУ. Переносимость (кроссплатформенность) ОА-

образа обеспечивается так: для вычислителя, на который следует перенести ОА-образ, реализуются подпрограммы логики работы необходимых типов ФУ. Причем, логики работы ФУ, запущенных на разных вычислительных узлах идентична – особенности узла «скрываются» за универсальным интерфейсом ФУ.

Применение универсального интерфейса ФУ дает следующие преимущества:

- Возможность изменения количества операндов у ФУ непосредственно во время вычислительного процесса, т.е. имеется возможность изменять логику работы ФУ (например, изменять режим его работы). В классической акторной модели можно изменять логику работы актора, но не его интерфейс.
- Унификация гетерогенной вычислительной системы.
- Предельно простой протокол обмена информацией между вычислительными узлами: операнды передаются по одному в любой момент времени и потому отпадает необходимость описания интерфейсов вызываемых удаленных процедур (например, для удаленного вызова процедур (RPC) применяется язык IDL, служащий лишь для того, чтобы описывать интерфейс удаленной процедуры).

ОА-архитектура обладает свойством изоморфизма как на уровне программного кода, так и на уровне данных. Так, ФУ вполне успешно можно использовать в качестве замены классическим процедурам и функциям, также акторам: если процедурам передаются все параметры одновременно, то для ФУ они поступают последовательно в виде нескольких милликоманд. Так модификация ФУ заключается в добавлении новых милликоманд, которые расширяют его функционал. И если программист во время модификации кода программы ФУ не нарушит функционал старых милликоманд, то ОА-образ, написанный ранее, который использует модифицированный ФУ, будет работать корректно. Существую и еще несколько способов обеспечения изоморфизма, на которых мы не будем останавливаться.

Разработанный ОА-язык имеет следующий синтаксис. Атрибут милликоманды указывается в качестве атрибута ИП и состоит из двух частей: мнемоника ВФУ, которому милликоманда должна быть передана; мнемоника атрибута данных. Эти две части отделаются одна от другой с помощью знака «.». По мнемонике ОА-компилятор производит синтез индекса расширенной милликоманды и помещает его в атрибут создаваемой ИП. Например, ALU.Set=0. Существуют три стандартные мнемоники милликоманд: Reset (Сброс) — сбросить ВФУ в начальное состояние, «Set» – установить значение, «Pop» – выдать значение. В том случае, когда в ОА-языке после мнемоники расширенной милликоманды не указана нагрузка, то в качестве нагрузки компилятор подставляет nil (нулевой указатель), если в нагрузке милликоманды передается указатель; или 0, если передается константа).

Далее приведем программу на ОА-языке, для работы с ФУ «Целочисленное АЛУ» (АЛУ – арифметико-логическое устройство). Программа производит сложение двух чисел и вывод результата на консоль.

```
NewFU={Mnemo="ALU" FUType=FUIntALU}
ALU.Set=2 ALU.Add=2 ALU.PopMk=Console.Out
```

где перед знаком «.» стоит мнемоника, обозначающая ФУ, после знака «.» - мнемоника обозначающая милликоманду;

перед знаком «=» стоит обозначение атрибута милликоманды, после – указывается нагрузка;

NewFU – милликоманда создания ФУ;

FLU.Set – милликоманда установки значения в аккумулятор АЛУ;

ALU.Add=2 – милликоманда сложения;

ALU.PopMk – милликоманда выдачи результата вычислений;

Console.Out – милликоманда выдачи на выводную консоль.

Последняя милликоманда («ALU.PopMk = Console.Out») иллюстрирует прямую передачу данных от АЛУ к ФУ «Выводная консоль»). Однако такой способ подходит только для реализации обмена информацией в пределах одной ОП. Для распределенных же систем необходимо производить обмен данными через ФУ-посредников, каковыми являются «Шина», «Маршрутизатор» и «Шлюз».

ФУ «Шина» хранит адреса всех ФУ, между которыми она осуществляет коммутацию. Каждому ФУ в атрибуте милликоманды выделяется свой числовой диапазон, в котором могут передаваться милликоманды для него. ФУ, формируя милликоманды для своих «собратьев» также помещают милликоманды в соответствующие адресные диапазоны, и передают милликоманды на «Шину». «Шина», исходя из адресных диапазонов, определяет ФУ-адресата и передает ему милликоманду (совокупность «Шины» и контролируемых ей ФУ будем называть сегментом). ФУ же «Маршрутизатор» и «Шлюз» обеспечивают работу ВС в распределенном режиме. Структура распределенной ОА-системы представлена на рис. 2

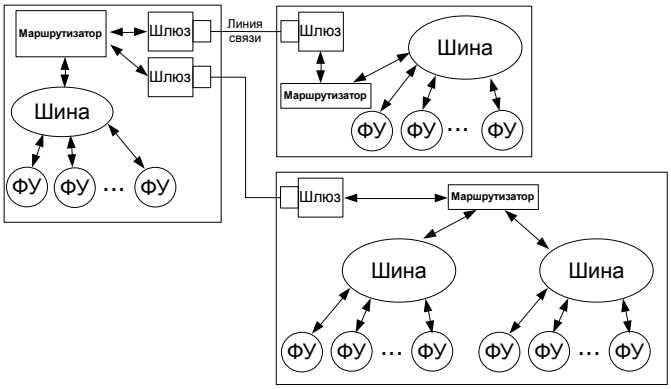

Рис. 2. Структура распределенной ОА-системы

Милликоманды, поступающие на «Шину», могут быть адресованы как ФУ, связанному с Шиной, так и ФУ расположенным на других вычислительных узлах. Для определения адресата милликоманды в состав Шины входят виртуальные регистры LowMkRange (нижний дипазон адресов вычислительного сегмента) и HieMkRange (верхний диапазон адресов вычислительного сегмента). Каждая «Шина», входящая в ОА-систему, имеет свой уникальный диапазон в атрибуте милликоманды. Если индекс милликоманды попадает в диапазон между LowMkrange и HieMkrange, то Шина вычисляет адрес «прикрепленного» к нему ФУ и передает для этого ФУ милликоманду. В противном случае «Шина» пересылает милликоманду «Маршрутизатору», который по индексу расширенной милликоманды определяет «Шлюз», на который следует передать милликоманду.

Для осуществления маршрутизации в ОА-системе используется следующее распределение адресного пространства атрибута милликоманды между сегментами: каждому вычислительному сегменту («Шина» вместе с «прикрепленными» к ней ФУ) выделяется определенный адресный диапазон, и сведения об адресных диапазонах помещаются в таблицу адресных диапазонов маршрутизатора.

ФУ «Маршрутизатор» осуществляет выбор канала связи, по которому следует передать милликоманду, т.е. осуществляет маршрутизацию информационных сообщений. Данное ФУ связано в ФУ «Шина», которое является поставщиком и приемником информации, и с одним или несколькими «Шлюзами», через которые осуществляется информационный обмен с другими вычислительными узлами (рис. 2). Ссылка на «Маршрутизатор» входит в состав контекста «Шины»: если индекс расширенной милликоманды не попадает в заданный для «Шины» диапазон адресов, то «Шина» передает информационную конструкцию на «Маршруризатор», который пересылает милликоманду в нужный канал. «Шина» производит передачу на другой вычислительный узел как

атрибут милликоманды, так и ее нагрузку. А в качестве нагрузку могут выступать не только скалярные величины, но и массивы, записи и прочие информационные конструкции.

IV. Среда ОА-программирования и моделирования

Была создана среда ОА-программирования (рис. 3), позволяющая запускать программы, написанные на ОА-языке и проводить имитационное моделирование вычислительной системы ОА-архитектуры (среда создавалась на протяжении 4-х лет). Среда обеспечивает функционирование виртуальной ОА-машины, создана в языке программирования Delphi, и обеспечивает работу более чем 70 типов ФУ, составляющих ОА-платформу. Также в состав программы входит ОА-транслятор, переводящий ОА-программу в ОА-образ и инструментальная среда программирования. ОА-язык может быть как скомпилирован (ОА-образ записывается в файл и может быть загружен и запущен в любое время), так и интерпретирован (таким образом имеется возможность управлять ОА-образом в реальном времени без его перезагрузки: выдавать на «Шину» тестовые наборы данных, создавать и уничтожать ФУ, выдавать милликоманды на ФУ – в этом случае инструментальная среда программирования превращается в консоль управления ОА-образом). Разработанный программный продукт позволяет проводить имитационное моделирование распределенных ВС.

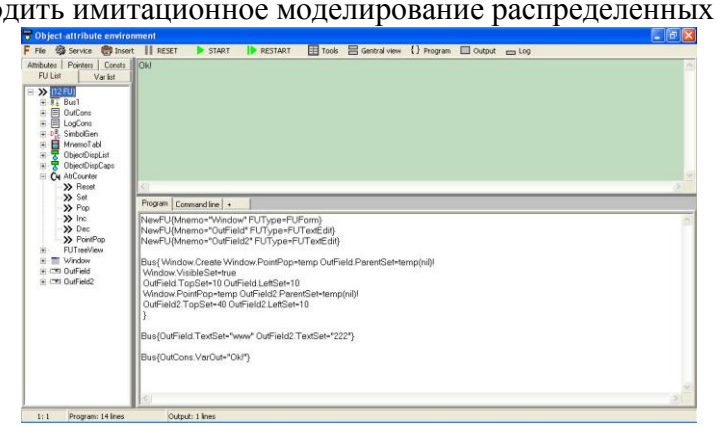

Рис. 3. Среда ОА-программирования и моделирования

Технология моделирования основывается на свойстве кроссплатформенности ФУ. Так, ФУ может быть запущено как на реальной ВС, так и не рабочей станции или персональном компьютере с сохранением своей функциональности. Входной и выходной информационные потоки эмулируются с помощью милликоманд, которые могут быть «запущены» в моделируемую ОА-систему программистом с целью отладки системы (реакцию системы на поступившие милликоманды можно отслеживать по милликомандам выходного потока данных – эти милликоманды, например, можно вывести на консоль). После отладки ОА-образ можно без дополнительных доработок переносить на реальную ВС, ведь на ней будут запущены ФУ с тем же функционалом.

V. Среда ОА-программирования и моделирования

В рамках НИР, проводимой в рамках Федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007—2013 годы» на модели ОА распределенной гетерогенной ВС был произведен запуск тестовых примеров, показавший эффективность предложенной архитектуры. В частности проводилось тестирование на тестовых примерах широкого класса физических задач (задачи с большим числом динамических степеней свободы, хорошо изученные двумерные и трехмерные спиновые системы (по модели Изинга), задачи моделирования глюонной плазмы, моделирование дискретного варианта электрослабых взаимодействий). Также проводилось моделирование ОА-системы на примере бенчмарка GRAPH500 (моделирование задачи по анализу скорости обработки графов вычислительной системой) и алгоритма сопоставления файловых строк по шаблонам команды GREP.

Так, архитектура для расчета широкого класса физических задач, обеспечивает расчет по сеточному алгоритму и имеет следующую архитектуру (рис. 4). В среде программирования вычислительная сетка строится так: каждый вычислительный узел сетки оформляется в виде ФУ-исполнителя, обменивающегося информацией со своими соседями (сетка может быть как 2-мерной, так и 3-мерной). ФУ-исполнитель реализует алгоритм расчета в узле сетки. При реализации распределенной ВС, расчетная сетка может делиться на несколько сегментов: каждый сегмент оформляется в сегмента под управлением ФУ-менеджера. В обязанности Менеджера входит синтез и настройка расчетной сетки, считывание информации с ФУ, образующих сетку, и управление вычислительным процессом на сегменте сетки. Координация всего вычислительного процесса возлагается на топ-Менеджера, контролирующего работу всех Менеджеров узлов сетки. Обмен информацией между сегментами расчетной сетки ВС обеспечивают «Шлюзы».

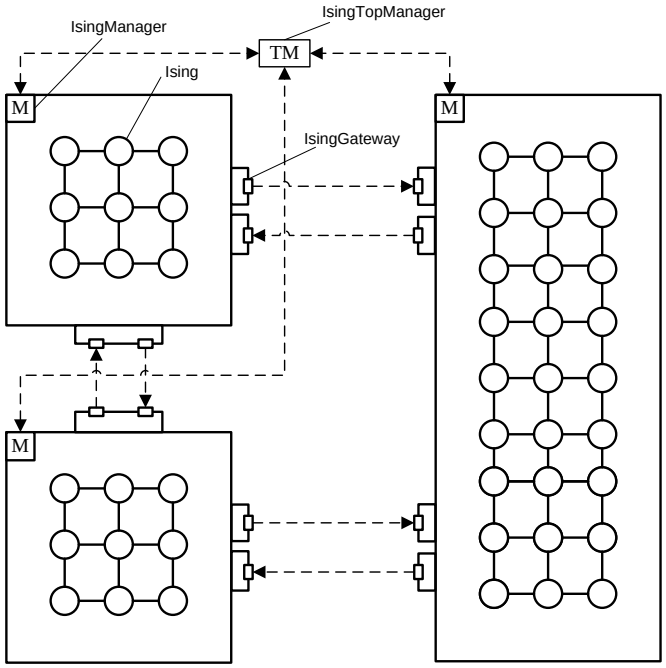

Рис. 4. Архитектура ОА-системы для расчета широкого класса физических задач

где

IsingTopManager – топ-менеджер;

IsingTopManager – менеджер;

IsingGateway – шлюз

Ising – ФУ-исполнитель расчета спиновых эффектов по модели Изинга.

Для запуска теста необходимо выполнить следующие действия:

1. Создать Менеджеров вычислительной сетки.

2. Настроить каждого Менеджера на синтез вычислительной сетки с заданными параметрами (размерность сетки, размеры по длине, ширине и высоте).

3. Настроить Шлюзы и ФУ сетки для передачи информации между сегментами сетки.

4. Задать для каждого Менеджера ссылку на топ-Менеджера.

5. Запустить топ-Менеджера для генерации волн расчета по вычислительной среде.

Для запуска тестовых пакетов Graph500 и Grep использовалась следующая архитектура (на рис.5 представлена архитектура для запуска пакета Grep).

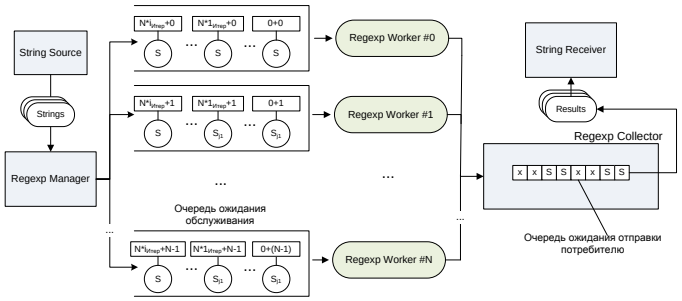

Рис. 5. Архитектура ОА-системы для запуска тестовых примеров Graph500 и Grep

ВС состоит из трех типов ФУ:

1. ФУ «Менеджер» (1 устройство) – осуществляет выдачу милликоманд с данными для обработки для ФУ.

2. ФУ-исполнители (число определяется условиями теста).

3. ФУ «Коллектор» (одно устройство) – на него ФУ-исполнители передают результаты своей работы, результаты поступают хаотично, т.к. ФУ тратят на обработку данных различное время; восстанавливает первоначальный порядок следования результатов и отправляет результаты, например, для дальнейшей обработки.

Моделирование показало эффективность ОА-архитектуры при работе на гетерогенных вычислительных системах. На рис. 6 представлены результаты моделирования теста Grep: по правой оси откладывает число исполнительных устройств (например, процессорных ядер) ВС, по левой – число ФУ.

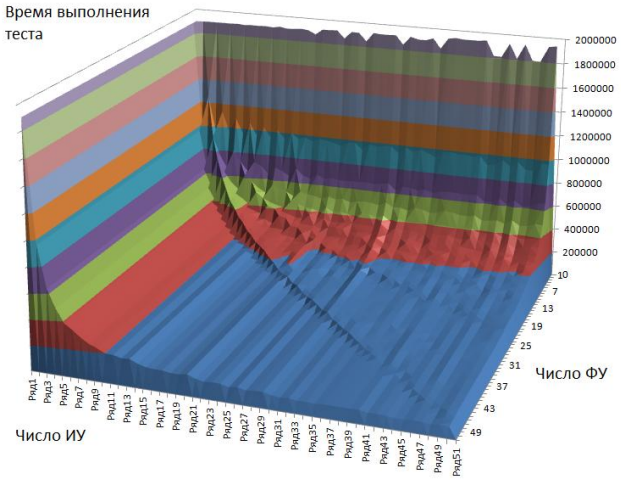

Рис. 6. Результаты моделирования тестового пакет Grep

VI. Заключение

В заключении можно обозначить работу проделанную в области методологии реализации гетерогенных вычислительных ОА-систем:

- Разработана методика создания гетерогенных ВС (способ обеспечения кроссплатформенности, метод организации обмена информацией между вычислительными узлами). Гетерогенная ВС может быть реализована в том числе и исключительно программным способом.
- Реализована ОА-среда программирования и моделирования, которая позволяет как создавать распределенные ВС, так и производить их автономное (т.е. без подключения реальной аппаратуры) моделирование.
- Было проведено автономное моделирование гетерогенной ВС, состоящей из РС и микроконтроллеров Atvega [5].

- Был произведен запуск на ОА среде программирования моделирования тестовых пакетов широкого класса физических задач, тестов Graph500 и Grep.

[1] Microsoft Demonstrates C++ AMP Heterogeneous Computing Favorite URL: http://panoramanews.com.ua/company-news/microsoft-demonstrates-c-amp-heterogeneous-computing-favorite.html

[2] AMD, ARM и Texas Instruments объединяют усилия в создании единой платформы гетерогенных вычислений URL: http://habrahabr.ru/post/145781/

[3] Арк.В., Климов, Н.Н.Левченко, А.С.Окунев Перспективы использования потоковой модели вычислений для в условиях иерархических коммутационных сред. URL: http://www.hpc-platform.ru/tiki-download_file.php?fileId=90.

[4] Салибекян С.М., Панфилов П.Б. Перспективная суперкомпьютерная система на основе объектно-атрибутной модели вычислений с управлением потоком данных / Международная конференция «Развитие суперкомпьютерных и грид-технологий в России» в рамках «Второго Московский Суперкомпьютерного форума» Россия, Москва, ВВЦ 26–27 октября 2011 года URL: http://www.hpc-platform.ru/tiki-download_file.php?fileId=82

[5] Салибекян С.М., Панфилов П.Б. Построение распределенных гетерогенных вычислительных систем на базе объектно-атрибутной архитектуры. // Объектные системы - 2011 (Зимняя сессия): материалы V Международной научно-практической конференции (Ростов-на-Дону, 10-12 декабря 2011 г.) / Под общ. ред. П.П. Олейника. - Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2011. - С. 83-88 URL: http://www.objectsystems.ru/files/2011WS/Object_Systems_2011_Winter_Session_Proceedings.pdf

S.M. Salibekyan, P.B. Panfilow Object-attribute architecture for design and modeling of distribute automation system. // Automation and remote control. Volume 73, Number 3, 587-595, DOI: 10.1134/S0005117912030174